

MATHEMATICAL STRUCTURES FOR | COMPUTER SCIENCE

DISCRETE
MATHEMATICS
AND ITS
APPLICATIONS

SEVENTH EDITION

JUDITH L. GERSTING

Mathematical Structures for Computer Science

This page intentionally left blank

Edition

7

Mathematical Structures for Computer Science

DISCRETE MATHEMATICS AND ITS APPLICATIONS

Judith L. Gersting

Indiana University-Purdue University at Indianapolis



W. H. Freeman and Company

A Macmillan Higher Education Company

To my 0110_2 favorite
discrete structures:
(Adam \wedge Francine),
(Jason \wedge Cathryn) \rightarrow
(Sammie \wedge Johnny)

Senior Publisher: Ruth Baruth
Executive Editor: Terri Ward
Senior Editor: Roland Cheyney
Assistant Editor: Liam Ferguson
Marketing Manager: Steve Thomas
Media Editor: Laura Judge
Senior Project Editor: Georgia Lee Hadler
Copy Editor: Penelope Hull
Production Coordinator: Susan Wein
Text Designer: Patrice Sheridan
Cover Designer: Victoria Tomaselli
Illustrations: Network Graphics
Composition: codeMantra
Printing and Binding: RR Donnelley

Library of Congress Control Number: 2013951442

ISBN-13: 978-1-4292-1510-7
ISBN-10: 1-4292-1510-0

© 2014, 2007, 2003, 1999, 1993, 1987, 1982 by W. H. Freeman and Company
All rights reserved

Printed in the United States of America
First printing

W. H. Freeman and Company
41 Madison Avenue, New York, NY 10010
Houndmills, Basingstoke RG21 6XS, England
www.whfreeman.com

Contents in Brief

	Preface	xiii
	Note to the Student	xvi
CHAPTER 1	Formal Logic	001
CHAPTER 2	Proofs, Induction, and Number Theory	097
CHAPTER 3	Recursion, Recurrence Relations, and Analysis of Algorithms	157
CHAPTER 4	Sets, Combinatorics, and Probability	221
CHAPTER 5	Relations, Functions, and Matrices	327
CHAPTER 6	Graphs and Trees	475
CHAPTER 7	Graph Algorithms	553
CHAPTER 8	Boolean Algebra and Computer Logic	617
CHAPTER 9	Modeling Arithmetic, Computation, and Languages	685
APPENDIX A	Derivation Rules for Propositional and Predicate Logic	803
APPENDIX B	Summation and Product Notation	805
APPENDIX C	The Logarithm Function	809
	Answers to Practice Problems	813
	Answers to Odd-Numbered Exercises	851
	Answers to Self-Tests	949
	Index	959

This page intentionally left blank

Contents

CHAPTER 1	Formal Logic	1		
1.1	STATEMENTS, SYMBOLIC REPRESENTATION, AND TAUTOLOGIES	2		
	<i>Connectives and Truth Values</i>	2		
	<i>Tautologies</i>	8		
	<i>Logical Connectives in the Real World</i>	10		
	<i>An Algorithm</i>	12		
<hr/>				
	SPECIAL INTEREST PAGE			
	Can “And” Ever Be “Or”?	15		
<hr/>				
	SECTION 1.1 Review	16		
	EXERCISES 1.1	16		
1.2	PROPOSITIONAL LOGIC	25		
	<i>Valid Arguments</i>	25		
	<i>Derivation Rules for Propositional Logic</i>	28		
	<i>Deduction Method and Other Rules</i>	32		
	<i>Verbal Arguments</i>	33		
	SECTION 1.2 Review	35		
	EXERCISES 1.2	35		
1.3	QUANTIFIERS, PREDICATES, AND VALIDITY	39		
	<i>Quantifiers and Predicates</i>	39		
	<i>Translation</i>	42		
	<i>Validity</i>	48		
	SECTION 1.3 Review	50		
	EXERCISES 1.3	50		
1.4	PREDICATE LOGIC	58		
	<i>Derivation Rules for Predicate Logic</i>	58		
	Universal Instantiation	59		
	Existential Instantiation	60		
	Universal Generalization	61		
	Existential Generalization	62		
	<i>More Work with Rules</i>	62		
	<i>Verbal Arguments</i>	67		
	<i>Conclusion</i>	68		
	SECTION 1.4 Review	69		
	EXERCISES 1.4	69		
1.5	LOGIC PROGRAMMING	73		
	<i>Prolog</i>	73		
	<i>Horn Clauses and Resolution</i>	75		
	<i>Recursion</i>	79		
	<i>Expert Systems</i>	81		
	SECTION 1.5 Review	82		
	EXERCISES 1.5	82		
1.6	PROOF OF CORRECTNESS	84		
	<i>Assertions</i>	85		
	<i>Assignment Rule</i>	87		
	<i>Conditional Rule</i>	90		
	SECTION 1.6 Review	92		
	EXERCISES 1.6	92		
	Chapter 1 Review	95		
	On the Computer	96		
CHAPTER 2	Proofs, Induction, and Number Theory	97		
2.1	PROOF TECHNIQUES	98		
	<i>Theorems and Informal Proofs To Prove or Not to Prove</i>	98		
	<i>Exhaustive Proof</i>	100		
	<i>Direct Proof</i>	101		
	<i>Contraposition</i>	103		
	<i>Contradiction</i>	104		

<i>Serendipity</i>	106	3.2 RECURRENCE RELATIONS	180
<i>Common Definitions</i>	107	<i>Linear First-Order Recurrence Relations</i>	180
SECTION 2.1 Review	107	Expand, Guess, and Verify	180
EXERCISES 2.1	107	A Solution Formula	182
2.2 INDUCTION	110	<i>Linear Second-Order Recurrence Relations</i>	188
<i>First Principle of Induction</i>	110	<i>Divide-and-Conquer Recurrence Relations</i>	193
<i>Proofs by Mathematical Induction</i>	112	SECTION 3.2 Review	197
<i>Second Principle of Induction</i>	118	EXERCISES 3.2	197
SECTION 2.2 Review	122	3.3 ANALYSIS OF ALGORITHMS	203
EXERCISES 2.2	122	<i>The General Idea</i>	203
2.3 MORE ON PROOF OF CORRECTNESS	129	<i>Analysis Using Recurrence Relations</i>	206
<i>Loop Rule</i>	129	<i>Upper Bound (Euclidean Algorithm)</i>	210
<i>Euclidean Algorithm</i>	133	<hr/>	
SPECIAL INTEREST PAGE		SPECIAL INTEREST PAGE	
Making Safer Software	136	Of Trees ... and Pancakes	211
<hr/>		<hr/>	
SECTION 2.3 Review	137	SECTION 3.3 Review	212
EXERCISES 2.3	137	EXERCISES 3.3	212
2.4 NUMBER THEORY	143	Chapter 3 Review	217
<i>The Fundamental Theorem of Arithmetic</i>	144	On the Computer	218
<i>More on Prime Numbers</i>	148	CHAPTER 4 Sets, Combinatorics, and Probability	221
<i>Euler Phi Function</i>	149	4.1 SETS	222
SECTION 2.4 Review	152	<i>Notation</i>	222
EXERCISES 2.4	152	<i>Relationships Between Sets</i>	224
Chapter 2 Review	155	<i>Sets of Sets</i>	227
On the Computer	156	<i>Binary and Unary Operations</i>	228
CHAPTER 3 Recursion, Recurrence Relations, and Analysis of Algorithms	157	<i>Operations on Sets</i>	230
3.1 RECURSIVE DEFINITIONS	158	<i>Set Identities</i>	233
<i>Recursively Defined Sequences</i>	158	<i>Countable and Uncountable Sets</i>	236
<i>Recursively Defined Sets</i>	162	SECTION 4.1 Review	239
<i>Recursively Defined Operations</i>	165	EXERCISES 4.1	239
<i>Recursively Defined Algorithms</i>	166	4.2 COUNTING	252
SECTION 3.1 Review	171	<i>Multiplication Principle</i>	252
EXERCISES 3.1	171	<i>Addition Principle</i>	254
		<i>Using the Principles Together</i>	255
		<i>Decision Trees</i>	257

SECTION 4.2 Review	258	SECTION 4.6 Review	315
EXERCISES 4.2	259	EXERCISES 4.6	315
4.3 PRINCIPLE OF INCLUSION AND EXCLUSION; PIGEONHOLE PRINCIPLE	263	Chapter 4 Review	323
<i>Principle of Inclusion and Exclusion</i>	264	On the Computer	324
<i>Pigeonhole Principle</i>	269	CHAPTER 5 Relations, Functions, and Matrices	327
SECTION 4.3 Review	269	5.1 RELATIONS	328
EXERCISES 4.3	270	<i>Binary Relations</i>	328
4.4 PERMUTATIONS AND COMBINATIONS	272	<i>Properties of Relations</i>	332
<i>Permutations</i>	272	<i>Closures of Relations</i>	334
<i>Combinations</i>	274	<i>Partial Orderings</i>	336
<i>Eliminating Duplicates</i>	277	<i>Equivalence Relations</i>	339
<i>Permutations and Combinations with Repetitions</i>	279	SECTION 5.1 Review	344
<i>Generating Permutations and Combinations</i>	280	EXERCISES 5.1	345
SPECIAL INTEREST PAGE		5.2 TOPOLOGICAL SORTING	356
Archimedes and the Stomachion	286	SECTION 5.2 Review	361
SECTION 4.4 Review	288	EXERCISES 5.2	362
EXERCISES 4.4	288	5.3 RELATIONS AND DATABASES	365
4.5 BINOMIAL THEOREM	294	<i>Entity-Relationship Model</i>	365
<i>Pascal's Triangle</i>	294	<i>Relational Model</i>	366
<i>Binomial Theorem and Its Proof</i>	296	<i>Operations on Relations</i>	369
<i>Applying the Binomial Theorem</i>	298	<i>Null Values and Three-valued Logic</i>	373
SECTION 4.5 Review	299	<i>Database Integrity</i>	375
EXERCISES 4.5	299	SECTION 5.3 Review	376
4.6 PROBABILITY	301	EXERCISES 5.3	376
<i>Introduction to Finite Probability</i>	301	5.4 FUNCTIONS	381
<i>Probability Distributions</i>	304	<i>Definition</i>	381
<i>Conditional Probability</i>	306	<i>Properties of Functions</i>	388
<i>Bayes' Theorem</i>	308	Onto Functions	388
<i>Expected Value</i>	310	One-to-One Functions	389
<i>Binomial Distributions</i>	313	Bijections	390
<i>Average Case Analysis of Algorithms</i>	314	<i>Composition of Functions</i>	390
		<i>Inverse Functions</i>	392
		<i>Permutation Functions</i>	394
		<i>How Many Functions</i>	397
		<i>Equivalent Sets</i>	401
		SECTION 5.4 Review	402
		EXERCISES 5.4	402

5.5	ORDER OF MAGNITUDE	412			
	<i>Function Growth</i>	412			
	<i>More on Analysis of Algorithms</i>	415			
	<i>The Master Theorem</i>	417			
	<i>Proof of the Master Theorem</i>	419			
SECTION 5.5	Review	421			
EXERCISES 5.5		421			
5.6	THE MIGHTY MOD FUNCTION	423			
	<i>Hashing</i>	424			
	<i>Computer Security</i>	427			
	Cryptography	427			
	Hashing for Password				
	Encryption	433			
	<i>Miscellaneous Applications</i>	435			
	Identification Codes	435			
	Generating and Decomposing				
	Integers	437			
	Modular Arithmetic Designs	438			
SECTION 5.6	Review	440			
EXERCISES 5.6		440			
5.7	MATRICES	446			
	<i>Terminology</i>	446			
	<i>Matrix Operations</i>	448			
	<i>Gaussian Elimination</i>	453			
	<i>Boolean Matrices</i>	458			
<hr/>					
SPECIAL INTEREST PAGE					
	Solve Millions of Equations, Faster than Gauss	460			
SECTION 5.7	Review	461			
EXERCISES 5.7		461			
	Chapter 5 Review	470			
	On the Computer	472			
CHAPTER 6	Graphs and Trees	475			
6.1	GRAPHS AND THEIR REPRESENTATIONS	476			
	<i>Definitions of a Graph</i>	476			
	<i>Applications of Graphs</i>	479			
	<i>Graph Terminology</i>	481			
	<i>Isomorphic Graphs</i>	484			
	<i>Planar Graphs</i>	487			
	<i>Computer Representation of Graphs</i>	492			
	Adjacency Matrix	492			
	Adjacency List	494			
<hr/>					
SPECIAL INTEREST PAGE					
	Isomorphic Protein Graphs	497			
SECTION 6.1	Review	498			
EXERCISES 6.1		498			
6.2	TREES AND THEIR REPRESENTATIONS	509			
	<i>Tree Terminology</i>	509			
	<i>Applications of Trees</i>	511			
	<i>Binary Tree Representation</i>	513			
	<i>Tree Traversal Algorithms</i>	514			
	<i>Results about Trees</i>	519			
SECTION 6.2	Review	521			
EXERCISES 6.2		521			
6.3	DECISION TREES	529			
	<i>Searching</i>	529			
	Lower Bounds on Searching	532			
	Binary Tree Search	533			
	<i>Sorting</i>	535			
SECTION 6.3	Review	536			
EXERCISES 6.3		536			
6.4	HUFFMAN CODES	539			
	<i>Problem and Trial Solution</i>	539			
	<i>Huffman Encoding Algorithm</i>	542			
	<i>Justification</i>	544			
	<i>Application of Huffman Codes</i>	546			
SECTION 6.4	Review	547			
EXERCISES 6.4		548			
	Chapter 6 Review	551			
	On the Computer	552			
CHAPTER 7	Graph Algorithms	553			
7.1	DIRECTED GRAPHS AND BINARY RELATIONS; WARSHALL'S ALGORITHM	554			
	<i>Directed Graphs and Binary Relations</i>	555			
	<i>Reachability</i>	557			
	<i>Warshall's Algorithm</i>	562			

SECTION 7.1 Review	566	What is Isomorphism?	626
EXERCISES 7.1	566	Isomorphism as Applied to Boolean Algebra	628
7.2 EULER PATH AND HAMILTONIAN CIRCUIT	571		
<i>Euler Path Problem</i>	571		
<i>Hamiltonian Circuit Problem</i>	576		
SECTION 7.2 Review	577		
EXERCISES 7.2	577		
7.3 SHORTEST PATH AND MINIMAL SPANNING TREE	581		
<i>Shortest-Path Problem</i>	581		
<i>Minimal Spanning Tree Problem</i>	587		
<hr/>			
SPECIAL INTEREST PAGE			
Pathfinding	589		
<hr/>			
SECTION 7.3 Review	591		
EXERCISES 7.3	591		
7.4 TRAVERSAL ALGORITHMS	596		
<i>Depth-First Search</i>	596		
<i>Breadth-First Search</i>	598		
<i>Analysis</i>	601		
<i>Applications</i>	601		
SECTION 7.4 Review	604		
EXERCISES 7.4	604		
7.5 ARTICULATION POINTS AND COMPUTER NETWORKS	607		
<i>The Problem Statement</i>	607		
<i>The Idea behind the Algorithm</i>	608		
<i>The Algorithm Itself</i>	610		
SECTION 7.5 Review	612		
EXERCISES 7.5	612		
Chapter 7 Review	614		
On the Computer	615		
CHAPTER 8 Boolean Algebra and Computer Logic	617		
8.1 BOOLEAN ALGEBRA STRUCTURE	618		
<i>Models or Abstractions</i>	619		
<i>Definition and Properties</i>	620		
<i>Isomorphic Boolean Algebras</i>	626		
		What is Isomorphism?	626
		Isomorphism as Applied to Boolean Algebra	628
		SECTION 8.1 Review	631
		EXERCISES 8.1	631
		8.2 LOGIC NETWORKS	638
		<i>Combinational Networks</i>	638
		Basic Logic Elements	638
		Boolean Expressions	639
		Truth Functions	640
		Networks and Expressions	641
		Canonical Form	642
		Minimization	645
		Programmable Logic Devices	647
		<i>A Useful Network</i>	648
		<i>Other Logic Elements</i>	650
		<i>Constructing Truth Functions</i>	652
<hr/>			
		SPECIAL INTEREST PAGE	
		Pruning Chips and Programs	654
		SECTION 8.2 Review	655
		EXERCISES 8.2	655
		8.3 MINIMIZATION	663
		<i>Minimization Process</i>	663
		<i>Karnaugh Map</i>	665
		Maps for Three and Four Variables	666
		Using the Karnaugh Map	668
		<i>Quine–McCluskey Procedure</i>	673
		SECTION 8.3 Review	677
		EXERCISES 8.3	678
		Chapter 8 Review	683
		On the Computer	684
		CHAPTER 9 Modeling Arithmetic, Computation, and Languages	685
		9.1 ALGEBRAIC STRUCTURES	686
		<i>Definitions and Examples</i>	686
		<i>Basic Results about Groups</i>	695
		<i>Subgroups</i>	698
		<i>Isomorphic Groups</i>	702

SECTION 9.1 Review	708	<i>Church–Turing Thesis</i>	769
EXERCISES 9.1	708	<i>Decision Problems and Uncomputability</i>	771
9.2 CODING THEORY	714	Examples of Decision Problems	772
<i>Introduction</i>	714	Halting Problem	773
<i>Background: Homomorphisms and Cosets</i>	715	<i>Computational Complexity</i>	776
<i>Generating Group Codes</i>	717	SECTION 9.4 Review	778
<i>Decoding Group Codes</i>	723	EXERCISES 9.4	779
SECTION 9.2 Review	727	9.5 FORMAL LANGUAGES	782
EXERCISES 9.2	727	<i>Classes of Grammars</i>	789
9.3 FINITE-STATE MACHINES	728	<i>Formal Languages and Computational Devices</i>	792
<i>Definition</i>	729	<i>Context-Free Grammars</i>	793
<i>Examples of Finite-State Machines</i>	729	SECTION 9.5 Review	795
<i>Recognition</i>	733	EXERCISES 9.5	795
<i>Regular Sets and Kleene’s Theorem</i>	735	Chapter 9 Review	799
<i>Machine Minimization</i>	737	On the Computer	800
Unreachable States	737	Appendix A Derivation Rules for Propositional and Predicate Logic	803
Minimization Procedure	739	Appendix B Summation and Product Notation	805
<i>Sequential Networks and Finite-State Machines</i>	744	Appendix C The Logarithm Function	809
SPECIAL INTEREST PAGE		Answers to Practice Problems	813
FSMs Behind the Game	749	Answers to Odd-Numbered Exercises	851
SECTION 9.3 Review	750	Answers to Self-Tests	949
EXERCISES 9.3	750	Index	959
9.4 TURING MACHINES	759		
<i>Definition</i>	760		
<i>Turing Machines as Set Recognizers</i>	764		
<i>Turing Machines as Function Computers</i>	767		

Preface

A course in discrete structures (discrete mathematics) played an important role in Curriculum 68, the very first ACM Computer Science Curriculum Guide: “This course introduces the student to those fundamental algebraic, logical, and combinatoric concepts from mathematics needed in the subsequent computer science courses and shows the applications of these concepts to various areas of computer science.”¹ Fast forward 45 years or so (through mobile computing, wireless networks, robotics, virtual reality, 3-D graphics, the Internet ...) to the joint ACM/IEEE-CS Computer Science Curricula 2013, where—still—discrete structures are of fundamental importance. “The material in discrete structures is pervasive in the areas of data structures and algorithms but appears elsewhere in computer science as well. For example, an ability to create and understand a proof—either a formal symbolic proof or a less formal but still mathematically rigorous argument—is important in virtually every area of computer science, including (to name just a few) formal specification, verification, databases, and cryptography. Graph theory concepts are used in networks, operating systems, and compilers. Set theory concepts are used in software engineering and in databases. Probability theory is used in intelligent systems, networking, and a number of computing applications.”²

This Seventh Edition was guided by Curricula 2013, and virtually all of the Core Tier 1 and Tier 2 topics for discrete structures from that document are included. Covering all those topics can fill a one-semester course, but there is certainly enough material in this edition to make for a very respectable two-semester course.

However much we as instructors may see the value in this foundational course, it is a difficult experience for many students, who often view it as a series of unconnected topics with little or no application to the rest of their chosen field of study. In the big picture, these topics are bound together by themes of

- importance of logical thinking
- power of mathematical notation
- usefulness of abstractions

but such themes are best appreciated in hindsight. Telling students, “You will need ideas from this course in many of your future computer science courses,” is also of little motivation. That’s why it is important to carve out time in your course syllabus (for either a one-semester or two-semester course) for some of the applications of this material. Here are topics in this edition that you may

¹*Communications of the ACM*, Vol. 11, Issue 3 (March 1968), pp. 151–197.

²Computer Science Curricula 2013, Pre-release version, <http://cs2013.com>

choose from, according to your interests and the interests of your students. Yes, students will probably see most of these topics in more detail in later computer science courses, but a quick introduction now can keep their interest and make your claim of relevance more credible.

Section 1.5	Logic programming
Sections 1.6 and 2.3	Proof of correctness
Section 3.3	Analysis of algorithms
Section 5.3	Relations and databases
Section 5.6	The mighty mod function
Section 6.4	Huffman codes
Section 8.2	Logic networks
Section 9.2	Coding theory

In addition, there is a Special Interest Page in each chapter that highlights interesting applications culled from “the real world.”

NEW IN THE SEVENTH EDITION


- The former Chapters 2 and 3 have been reorganized as Chapters 2, 3, and 4 for better clarity and sequencing
- New sections or subsections have been added:
 - Probability**
 - Bayes’ Theorem
 - Binomial Distribution
 - Order of Magnitude (new section)**
 - The Master Theorem
 - Proof of the Master Theorem
 - Matrices**
 - Gaussian Elimination
 - Coding Theory (new section)**
 - Introduction
 - Background: Homomorphisms and Cosets
 - Generating Group Codes
 - Decoding Group Codes
- “Special interest pages”—one per chapter—have been introduced to add relevance and interest to the material being presented.
- Answers to all odd-numbered exercises, as opposed to answers to fewer, selected exercises, appear in the back of the book. When an exercise asks

for a proof, the complete proof is given. Otherwise, the answer is just the answer, not necessarily the solution. A Student Solutions Manual with solutions for odd-numbered exercises from the book is available from the Web site at www.whfreeman.com/gersting. The student manual also includes two sample tests per chapter. A complete Solutions Manual is available to instructors from the publisher.

- Many new exercises have been added, particularly with an eye toward pairing odd-numbered exercises with similar even-numbered exercises.
- Of course, student learning aids such as chapter objectives, practice problems, reminders, section reviews, and chapter reviews remain.

WEB SITE

Online Study Guide

A Web site for the book may be found at www.whfreeman.com/gersting. The Web pages contain representative new example problems (not contained in the text) for many of the end-of-section Techniques. Each Technique that has a corresponding Web page example is marked with the icon .

Each example on the Web first states the problem. Then succeeding pages develop the solution, much as the student would be expected to write it. As the student navigates the pages, the solution unfolds step-by-step. A compressed audio file is also part of each Web page after the initial problem statement. The audio file contains a first-person stream-of-consciousness thought process about that step of the solution—why it occurred to the narrator to try this, why it looked promising, what knowledge was being called on to suggest that this step should come next, and so on. The point is, students see perfect and complete worked-out proofs in the textbook and often see them performed by the instructor. Yet when a student goes home and tries to produce such a solution by himself or herself, he or she is unsure where to start or how to think about the problem or how to see any pattern to enable a guess as to what to do next. Consequently the student gives up in frustration. The purpose of the audio narration is to share the “secret picture” that mathematicians use to solve problems.

To access the problems, after you go to www.whfreeman.com/gersting, select a chapter section, then select a sample problem and follow its step-by-step process with the “Next” button.

PowerPoint Slides

Instructors who visit the web site will also have access to PowerPoint slides accompanying each section of the text.

ACKNOWLEDGMENTS

My thanks to the reviewers of this addition, as well to reviewers of earlier editions, all of whose help is greatly appreciated.

Elizabeth Adams, *James Madison University*
 Kemal Akkaya, *Southern Illinois University*
 Charles Ashbacher, *Mount Mercy College*
 Barnabas Bede, *DigiPen Institute of Technology*
 Terry J. Bridgeman, *Colorado School of Mines*
 David Casperson, *University of Northern British Columbia*
 Adrienne Decker, *SUNY Buffalo*
 Steve Donaldson, *Samford University*
 Mordechai S. Goodman, *Dominican University*
 Michael A. Gray, *American University*
 Jerrold R. Griggs, *University of South Carolina*
 Joseph Hobart, *Okanagan College*
 Mark Jacobson, *University of Northern Iowa*
 Lisa A. Jamba, *University of Northern Florida*

Tim Lin, *Cal Poly*
 David Lugenbuhl, *Western Carolina University*
 Damian Lyons, *Fordham University*
 Mariana Maris, *Arizona State University*
 Mikel D. Petty, *University of Alabama in Huntsville*
 Amar Raheja, *Cal Poly*
 J. Ben Schafer, *University of Northern Iowa*
 Ali Shaykhian, *Florida Institute of Technology*
 Shunichi Toida, *Old Dominion University*
 William J. Weber, *Southeast Missouri State University*
 Eric Westlund, *Luther University*
 Hua Yan, *Borough of Manhattan Community College*
 Yu Zhang, *Texas A&M Corpus Christi*

The folks at W.H. Freeman were very helpful in shepherding this edition to completion, especially Penny Hull (veteran of many previous editions), Terri Ward, Roland Cheyney, Liam Ferguson, Georgia Lee Hadler, and Vicki Tomaselli.

Thanks to Russell Kackley for the audio files on the Web site.

My deepest thanks go to my husband, John, ever my most ardent supporter and dearest friend.

NOTE TO THE STUDENT

As you go through this book, you'll encounter many new terms and new ideas. Try reading with pencil and paper at hand and work the Practice problems as you encounter them. They are intended to reinforce or clarify some new terminology or method just introduced; answers are given at the back of the book. Pay attention also to the Reminders that point out common pitfalls or provide helpful hints.

Be sure to visit the Web site at www.whfreeman.com/gersting for detailed, worked-out solutions to additional example problems tied to the Techniques in each section. The Web site solutions are accompanied by audio files that explain each step. A Student Solutions Manual with solutions for odd-numbered exercises from the book is available from the Web site. The student manual also includes two sample tests per chapter.

You may find at first that the thought processes required to solve the exercises in the book are new and difficult. Your biggest attribute for success will be perseverance. Here's what I tell my students: "If you do not see at first how to solve a problem, don't give up, think about it some more; be sure you understand all the terminology used in the problem, play with some ideas. If no approach presents itself, let it be and think about it again later. Repeat this process for days on end. When you finally wake up in the middle of the night with an idea, you'll know you are putting in the right amount of effort for this course." Mathematical results don't spring fully formed from the foreheads of mathematical geniuses; well, maybe from mathematical geniuses, but for the rest of us, it takes work, patience, false starts, and **perseverance**.

Enjoy the experience!

Formal Logic

CHAPTER OBJECTIVES

After studying this chapter, you will be able to:

- Use the formal symbols of propositional logic.
- Find the truth value of an expression in propositional logic.
- Construct formal proofs in propositional logic, and use such proofs to determine the validity of English language arguments.
- Use the formal symbols of predicate logic.
- Find the truth value in some interpretation of an expression in predicate logic.
- Use predicate logic to represent English language sentences.
- Construct formal proofs in predicate logic, and use such proofs to determine the validity of English language arguments.
- Understand how the programming language Prolog is built on predicate logic.
- Mathematically prove the correctness of programs that use assignment statements and conditional statements.

You have been selected to serve on jury duty for a criminal case. The attorney for the defense argues as follows:

If my client is guilty, then the knife was in the drawer. Either the knife was not in the drawer or Jason Pritchard saw the knife. If the knife was not there on October 10, it follows that Jason Pritchard did not see the knife. Furthermore, if the knife was there on October 10, then the knife was in the drawer and also the hammer was in the barn. But we all know that the hammer was not in the barn. Therefore, ladies and gentlemen of the jury, my client is innocent.

Question: Is the attorney's argument sound? How should you vote?

It's much easier to answer this question if the argument is recast in the notation of formal logic. Formal logic strips away confusing verbiage and allows us to concentrate on the underlying reasoning being applied. In fact, formal logic—the subject of this chapter—provides the foundation for the organized, careful method of thinking that characterizes any reasoned activity—a criminal investigation, a scientific experiment, a sociological study. In addition, formal logic has direct applications in computer science. The last two sections of this chapter explore a programming language based on logic and the use of formal logic to verify the correctness of computer programs. Also, circuit logic (the logic governing

computer circuitry) is a direct analog of the statement logic of this chapter. We will study circuit logic in Chapter 8.

SECTION 1.1 STATEMENTS, SYMBOLIC REPRESENTATION, AND TAUTOLOGIES

Formal logic can represent the statements we use in English to communicate facts or information. A **statement** (or **proposition**) is a sentence that is either true or false.

EXAMPLE 1

Consider the following:

- a. Ten is less than seven.
- b. Cheyenne is the capital of Wyoming.
- c. She is very talented.
- d. There are life forms on other planets in the universe.

Sentence (a) is a statement because it is false. Sentence (b) is a statement because it is true. Sentence (c) is neither true nor false because “she” is not specified; therefore (c) is not a statement. Sentence (d) is a statement because it is either true or false; we do not have to be able to decide which. ●

Connectives and Truth Values

In English, simple statements are combined with connecting words like *and* to make more interesting compound statements. The truth value of a compound statement depends on the truth values of its components and which connecting words are used. If we combine the two true statement, “Elephants are big,” and, “Baseballs are round,” we would consider the resulting statement, “Elephants are big and baseballs are round,” to be true. In this book, as in many logic books, capital letters near the beginning of the alphabet, such as A , B , and C , are used to represent statements and are called **statement letters**; the symbol \wedge is a **logical connective** representing *and*. We agree, then, that if A is true and B is true, $A \wedge B$ (read “ A and B ”) should be considered true.

PRACTICE 1

¹

- a. If A is true and B is false, what truth value would you assign to $A \wedge B$?
- b. If A is false and B is true, what truth value would you assign to $A \wedge B$?
- c. If A and B are both false, what truth value would you assign to $A \wedge B$? ■

The expression $A \wedge B$ is called the **conjunction** of A and B , and A and B are called the **conjuncts** of this expression. Table 1.1 summarizes the truth value of $A \wedge B$ for all possible truth values of the conjuncts A and B . Each row of the table

¹Answers to practice problems are in the back of the text.

A	B	$A \wedge B$
T	T	T
T	F	F
F	T	F
F	F	F

A	B	$A \vee B$
T	T	T
T	F	
F	T	
F	F	

represents a particular truth value assignment to the statement letters, and the resulting truth value for the compound expression is shown.

Another connective is the word *or*, denoted by the symbol \vee . The expression $A \vee B$ (read “ A or B ”) is called the **disjunction** of A and B , and A and B are called the **disjuncts** of this expression. If A and B are both true, then $A \vee B$ would be considered true, giving the first line of the truth table for disjunction (Table 1.2).

PRACTICE 2

Use your understanding of the word *or* to complete the truth table for disjunction, Table 1.2.

Statements may be combined in the form “if statement 1, then statement 2.” If A denotes statement 1 and B denotes statement 2, the compound statement would be denoted by $A \rightarrow B$ (read “ A implies B ”). The logical connective here is **implication**, and it conveys the meaning that the truth of A implies or leads to the truth of B . In the implication $A \rightarrow B$, A stands for the **antecedent** statement and B stands for the **consequent** statement.

The truth table for implication is less obvious than that for conjunction or disjunction. To understand its definition, let’s suppose your friend remarks, “If I pass my economics test, then I’ll go to the movie Friday.” If your friend passes the test and goes to the movie, the remark was true. If your friend passes the test but doesn’t go to the movie, the remark was false. If your friend doesn’t pass the test, then—whether he or she goes to the movie or not—you could not claim that the remark was false. You would probably want to give the benefit of the doubt and say that the statement was true. By convention, $A \rightarrow B$ is considered true if A is false, regardless of the truth value of B .

PRACTICE 3

Summarize this discussion by writing the truth table for $A \rightarrow B$.

The **equivalence** connective is symbolized by \leftrightarrow . Unlike conjunction, disjunction, and implication, the equivalence connective is not really a fundamental connective but a convenient shortcut. The expression $A \leftrightarrow B$ stands for $(A \rightarrow B) \wedge (B \rightarrow A)$. We can write the truth table for equivalence by constructing, one piece at a time, a table for $(A \rightarrow B) \wedge (B \rightarrow A)$, as in Table 1.3. From this truth table, $A \leftrightarrow B$ is true exactly when A and B have the same truth value.

A	B	$A \rightarrow B$	$B \rightarrow A$	$(A \rightarrow B) \wedge (B \rightarrow A)$
T	T	T	T	T
T	F	F	T	F
F	T	T	F	F
F	F	T	T	T

The connectives we’ve seen so far are called **binary connectives** because they join two expressions together to produce a third expression. Now let’s consider a **unary connective**, a connective acting on one expression to produce a second

expression. **Negation** is a unary connective. The negation of A —symbolized by A' —is read “not A .”

PRACTICE 4 Write the truth table for A' . (It will require only two rows.)

Table 1.4 summarizes the truth values for all of the logical connectives. This information is critical to an understanding of logical reasoning.

A	B	$A \wedge B$	$A \vee B$	$A \rightarrow B$	$A \leftrightarrow B$	A'
T	T	T	T	T	T	F
T	F	F	T	F	F	
F	T	F	T	T	F	T
F	F	F	F	T	T	

Because of the richness of the English language, words that have different shades of meaning are nonetheless represented by the same logical connective. Table 1.5 shows the common English words associated with various logical connectives.

English Word	Logical Connective	Logical Expression
and; but; also; in addition; moreover	Conjunction	$A \wedge B$
or	Disjunction	$A \vee B$
If A , then B . A implies B . A , therefore B . A only if B . B follows from A . A is a sufficient condition for B . B is a necessary condition for A .	Implication	$A \rightarrow B$
A if and only if B . A is necessary and sufficient for B .	Equivalence	$A \leftrightarrow B$
not A It is false that A ... It is not true that A ...	Negation	A'

REMINDER

A only if B means
 $A \rightarrow B$

Suppose that $A \rightarrow B$ is true. Then, according to the truth table for implication, the consequent, B , can be true even though the antecedent, A , is false. So while the truth of A leads to (implies) the truth of B , the truth of B does not imply the truth of A . The phrase “ B is a necessary condition for A ” to describe $A \rightarrow B$ simply

means that if A is true, then B is necessarily true, as well. “ A only if B ” describes the same thing, that A implies B .

EXAMPLE 2

The statement, “Fire is a necessary condition for smoke,” can be restated, “If there is smoke, then there is fire.” The antecedent is “there is smoke,” and the consequent is “there is fire.”

PRACTICE 5

Name the antecedent and consequent in each of the following statements. (*Hint: Rewrite each statement in if-then form.*)

- If the rain continues, then the river will flood.
- A sufficient condition for network failure is that the central switch goes down.
- The avocados are ripe only if they are dark and soft.
- A good diet is a necessary condition for a healthy cat.

EXAMPLE 3

Expressing the negation of a statement must be done with care, especially for a compound statement. Table 1.6 gives some examples.

TABLE 1.6

Statement	Correct Negation	Incorrect Negation
It will rain tomorrow.	It is false that it will rain tomorrow. It will not rain tomorrow.	
Peter is tall and thin.	It is false that Peter is tall and thin. Peter is not tall or he is not thin. Peter is short or fat.	Peter is short and fat. Too strong a statement. Peter fails to have both properties (tallness and thinness) but may still have one property.
The river is shallow or polluted.	It is false that the river is shallow or polluted. The river is neither shallow nor polluted. The river is deep and unpolluted.	The river is not shallow or not polluted. Too weak a statement. The river fails to have either property, not just fails to have one property.

PRACTICE 6

Which of the following represents A' if A is the statement “Julie likes butter but hates cream”?

- Julie hates butter and cream.
- Julie does not like butter or cream.
- Julie dislikes butter but loves cream.
- Julie hates butter or likes cream.

We can string statement letters, connectives, and parentheses (or brackets) together to form new expressions, as in

$$(A \rightarrow B) \wedge (B \rightarrow A)$$

Of course, just as in a computer programming language, certain *syntax rules* (rules on which strings are legitimate) prevail; for example,

$$A)) \wedge \wedge \rightarrow BC$$

would not be considered a legitimate string. An expression that is a legitimate string is called a **well-formed formula**, or **wff**. To reduce the number of parentheses required in a wff, we stipulate an order in which connectives are applied. This *order of precedence* is

1. connectives within parentheses, innermost parentheses first
2. ' (negation)
3. \wedge, \vee (conjunction, disjunction)
4. \rightarrow (implication)
5. \leftrightarrow (biconditional)

This means that the expression $A \vee B'$ stands for $A \vee (B')$, not $(A \vee B)'$. Similarly, $A \vee B \rightarrow C$ means $(A \vee B) \rightarrow C$, not $A \vee (B \rightarrow C)$. However, we often use parentheses anyway, just to be sure that there is no confusion.

In a wff with a number of connectives, the connective to be applied last is the **main connective**. In

$$A \wedge (B \rightarrow C)'$$

the main connective is \wedge . In

$$((A \vee B) \wedge C) \rightarrow (B \vee C')$$

the main connective is \rightarrow . Capital letters near the end of the alphabet, such as P, Q, R , and S , are used to represent wffs. Thus P could represent a single statement letter, which is the simplest kind of wff, or a more complex wff. We might represent

$$((A \vee B) \wedge C) \rightarrow (B \vee C')$$

as

$$P \rightarrow Q$$

if we want to hide some of the details for the moment and only concentrate on the main connective.

Wffs composed of statement letters and connectives have truth values that depend on the truth values assigned to their statement letters. We write the truth table for any wff by building up the component parts, just as we did for $(A \rightarrow B) \wedge (B \rightarrow A)$. The main connective is addressed in the last column of the table.

EXAMPLE 4

The truth table for the wff $A \vee B' \rightarrow (A \vee B)'$ is given in Table 1.7. The main connective, according to the rules of precedence, is implication. ●

A	B	B'	$A \vee B'$	$A \vee B$	$(A \vee B)'$	$A \vee B' \rightarrow (A \vee B)'$
T	T	F	T	T	F	F
T	F	T	T	T	F	F
F	T	F	F	T	F	T
F	F	T	T	F	T	T

If we are making a truth table for a wff that contains n different statement letters, how many rows will the truth table have? From truth tables done so far, we know that a wff with only one statement letter has two rows in its truth table, and a wff with two statement letters has four rows. The number of rows equals the number of true-false combinations possible among the statement letters. The first statement letter has two possibilities, T and F. For each of these possibilities, the second statement letter has two possible values. Figure 1.1a pictures this as a two-level “tree” with four branches showing the four possible combinations of T and F for two statement letters. For n statement letters, we extend the tree to n levels, as in Figure 1.1b. The total number of branches then equals 2^n . The total number of rows in a truth table for n statement letters is also 2^n .

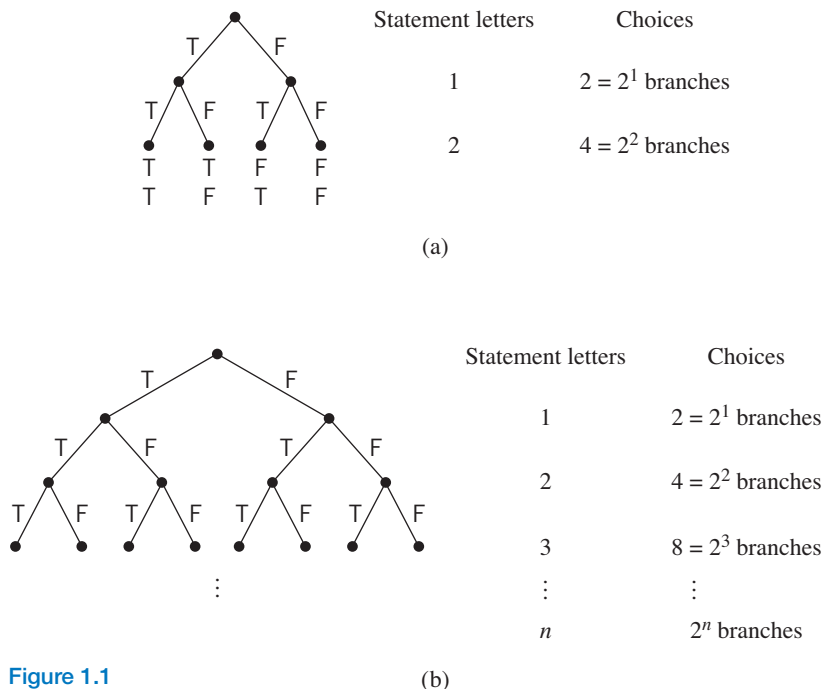


Figure 1.1

A	B	C
T	T	T
T	T	F
T	F	T
T	F	F
F	T	T
F	T	F
F	F	T
F	F	F

This tree structure also tells us how to enumerate all the T–F combinations among the n statement letters when setting up a truth table. If we read each level of the tree from bottom to top, it says that the T–F values for statement letter n (which will compose the last column of the truth table) alternate, those for statement letter $n - 1$ alternate every two values, those for statement letter $n - 2$ alternate every four values, and so forth. Thus a truth table for three statement letters would begin as shown in Table 1.8. The values for statement letter C alternate, those for statement letter B alternate in groups of two, and those for statement letter A alternate in groups of four, resulting in something like a sideways version of the tree. (Reading the rows from the bottom up and using 1 for T and 0 for F shows that we are simply counting up from zero in binary numbers.)

PRACTICE 7

Construct truth tables for the following wffs.

- $(A \rightarrow B) \leftrightarrow (B \rightarrow A)$ (Remember that $C \leftrightarrow D$ is true precisely when C and D have the same truth value.)
- $(A \vee A') \rightarrow (B \wedge B')$
- $[(A \wedge B') \rightarrow C']'$
- $(A \rightarrow B) \leftrightarrow (B' \rightarrow A')$

Tautologies

A wff-like item (d) of Practice 7, whose truth values are always true, is called a **tautology**. A tautology is “intrinsically true” by its very structure; it is true no matter what truth values are assigned to its statement letters. A simpler example of a tautology is $A \vee A'$; consider, for example, the statement “Today the sun will shine or today the sun will not shine,” which must always be true because one or the other of these must happen. A wff like item (b) of Practice 7, whose truth values are always false, is called a **contradiction**. A contradiction is “intrinsically false” by its very structure. A simpler example of a contradiction is $A \wedge A'$; consider “Today is Tuesday and today is not Tuesday,” which is false no matter what day of the week it is.

REMINDER

A, B, C stand for single statement letters; P, Q, R, S stand for wffs.

Suppose that P and Q represent two wffs, and it happens that the wff $P \leftrightarrow Q$ is a tautology. If we did a truth table using the statement letters in P and Q , then the truth values of the wffs P and Q would agree for every row of the truth table. In this case, P and Q are said to be **equivalent wffs**, denoted by $P \Leftrightarrow Q$. Thus $P \Leftrightarrow Q$ states a fact, namely, that the particular wff $P \leftrightarrow Q$ is a tautology. Practice 7(d) has the form $P \leftrightarrow Q$, where P is the wff $(A \rightarrow B)$ and Q is the wff $(B' \rightarrow A')$, and $P \leftrightarrow Q$ was shown to be a tautology. Therefore, $(A \rightarrow B) \Leftrightarrow (B' \rightarrow A')$.

We will list some basic equivalences, prove one or two of them by constructing truth tables, and leave the rest as exercises. We represent any contradiction by 0 and any tautology by 1.

Some Tautological Equivalences

- | | | |
|--|--|---------------------------|
| 1a. $A \vee B \Leftrightarrow B \vee A$ | 1b. $A \wedge B \Leftrightarrow B \wedge A$ | (commutative properties) |
| 2a. $(A \vee B) \vee C \Leftrightarrow A \vee (B \vee C)$ | 2b. $(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$ | (associative properties) |
| 3a. $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$ | 3b. $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$ | (distributive properties) |
| 4a. $A \vee 0 \Leftrightarrow A$ | 4b. $A \wedge 1 \Leftrightarrow A$ | (identity properties) |
| 5a. $A \vee A' \Leftrightarrow 1$ | 5b. $A \wedge A' \Leftrightarrow 0$ | (complement properties) |

Note that 2a allows us to write $A \vee B \vee C$ with no need for parentheses because the grouping doesn't matter; similarly, 2b allows us to write $A \wedge B \wedge C$.

EXAMPLE 5

The truth table in Table 1.9a verifies equivalence 1a, the commutative property for disjunction, and that in Table 1.9b verifies 4b, the identity property for conjunction. Note that only two rows are needed for Table 1.9b because 1 (a tautology) cannot take on false truth values.

TABLE 1.9										
(a)	A	B	$A \vee B$	$B \vee A$	$A \vee B \leftrightarrow B \vee A$	(b)	A	1	$A \wedge 1$	$A \wedge 1 \leftrightarrow A$
	T	T	T	T	T		T	T	T	T
	T	F	T	T	T		F	T	F	T
	F	T	T	T	T					
	F	F	F	F	T					

PRACTICE 8 Verify equivalence 5a.

The equivalences in the list are grouped into five pairs. In each pair, one equivalence can be obtained from the other by replacing \wedge with \vee , \vee with \wedge , 0 with 1, or 1 with 0. Each equivalence in a pair is called the **dual** of the other. Thus, 1a and 1b (commutativity of disjunction and commutativity of conjunction) are duals of each other. This list of equivalences appears in a more general setting in Chapter 8.

Two additional equivalences that are very useful are **De Morgan's laws**, named for the nineteenth-century British mathematician Augustus De Morgan, who first stated them. This theorem is easy to prove (see Exercises 26e and 26f).

THEOREM DE MORGAN'S LAWS

$$(A \vee B)' \Leftrightarrow A' \wedge B' \quad \text{and} \quad (A \wedge B)' \Leftrightarrow A' \vee B'$$

Each is the dual of the other. De Morgan's laws help in expressing the negation of a compound statement, as in Practice 6.

Think of these tautological equivalences as patterns; in order to use one of them, you must match its pattern exactly. For example, you can't say that $(A \wedge B) \vee C \Leftrightarrow A \wedge (B \vee C)$ by either of the associative properties because neither of these properties uses both conjunction and disjunction.

Suppose that P and Q are equivalent wffs. Then in any wff where P appears, P can be replaced by Q with no change in the overall truth values. It's like replacing a \$20 bill in your wallet with two \$10 bills—the total value of your money hasn't changed.

EXAMPLE 6

From Practice 7(d), $A \rightarrow B$ is equivalent to $B' \rightarrow A'$. The wff $(A \rightarrow B) \rightarrow B$ should therefore be equivalent to $(B' \rightarrow A') \rightarrow B$. This equivalence is verified by Tables 1.10a and 1.10b.

TABLE 1.10											
(a)	A	B	A → B	(A → B) → B	(b)	A	B	A'	B'	B' → A'	(B' → A') → B
	T	T	T	T		T	T	F	F	T	T
	T	F	F	T		T	F	F	T	F	T
	F	T	T	T		F	T	T	F	T	T
	F	F	T	F		F	F	T	T	T	F

Logical Connectives in the Real World

Web search engines allow exploration of the vast resources available on the Web, but a little care in your search query can help focus the results more quickly. For example, if you enter

used cars

in a Web search engine, you may get back references to any Web site containing either the word *used* or the word *cars*; this could include sites for antique dealers and sites for the latest auto racing results. Entering the phrase

“used cars”

in quotes restricts the search, on most search engines, to Web sites containing this exact phrase. Most search engines also allow you to enter an expression using logical connectives as your search query, which can help make the query even more specific. To further narrow your used car search, for example, you could enter

“used cars” AND (Ford OR Buick)

This would tend to limit your search to sites that mention only particular brands of used cars, although you could still wind up with a link to Jim Bob Ford’s Loan Shark Agency, which will lend you money for any used car. The query

“used cars” AND (Ford OR Buick) AND NOT trucks

would eliminate sites mentioning trucks. Many search engines use + (a plus sign) in place of AND, and – (a minus sign) in place of AND NOT.

The logical connectives AND, OR, and NOT are also available in many programming languages, as well as on programmable graphing calculators. These connectives, in accordance with the truth tables we have defined, act on combinations of true or false expressions to produce an overall truth value. Such truth values provide the decision-making capabilities fundamental to the flow of control in computer programs. Thus, at a conditional branch in a program, if the truth value of the conditional expression is true, the program will next execute one section of code; if the value is false, the program will next execute a different section of code. If the conditional expression is replaced by a simpler, equivalent expression, the truth value of the expression and hence the flow of control of the program is not affected, but the new code is easier to understand and may execute faster.

EXAMPLE 7

Consider a statement in a computer program that has the form

```

if ((outflow > inflow) and not ((outflow > inflow) and (pressure < 1000)))
  do something;
else
  do something else;

```

Here the conditional expression has the form

$$A \wedge (A \wedge B)'$$

where A is “outflow > inflow” and B is “pressure < 1000.” This expression can be simplified by replacing some wffs with equivalent wffs.

$$\begin{aligned}
 A \wedge (A \wedge B)' &\Leftrightarrow A \wedge (A' \vee B') && \text{(De Morgan's laws)} \\
 &\Leftrightarrow (A \wedge A') \vee (A \wedge B') && \text{(tautology 3b)} \\
 &\Leftrightarrow 0 \vee (A \wedge B') && \text{(tautology 5b)} \\
 &\Leftrightarrow (A \wedge B') \vee 0 && \text{(tautology 1a)} \\
 &\Leftrightarrow A \wedge B' && \text{(tautology 4a)}
 \end{aligned}$$

The statement form can therefore be written

```

if ((outflow > inflow) and not (pressure < 1000))
  do something;
else
  do something else;

```

Finally, the truth tables for conjunction, disjunction, and negation are implemented by electronic devices called “gates” (AND gate, OR gate, inverter, respectively) that are fundamental building blocks in computer circuitry. We’ll see in Chapter 8 (Boolean Algebra and Computer Logic) how to combine these gates into more complex logic networks to carry out specific tasks.

An Algorithm

To test whether a wff is a tautology, we can always write its truth table. For n statement letters, 2^n rows will be needed for the truth table. Suppose, however, that the wff has implication as its main connective, so that it has the form $P \rightarrow Q$ where P and Q are themselves wffs. Then we can use a quicker procedure than constructing a truth table to determine whether $P \rightarrow Q$ is a tautology. We assume that $P \rightarrow Q$ is *not* a tautology, and we see whether this leads to some impossible situation. If it does, then the assumption that $P \rightarrow Q$ is not a tautology is also impossible, and $P \rightarrow Q$ must be a tautology after all.

To assume that $P \rightarrow Q$ is not a tautology is to say that it can take on false values, and, by the truth table for implication, $P \rightarrow Q$ is false only when P is true and Q false. By assigning P true and Q false, we determine possible truth values for the wffs making up P and Q . We continue assigning the truth values so determined until all occurrences of statement letters have a truth value. If some statement letter is assigned both true and false values by this process, we have an impossible situation, so the wff $P \rightarrow Q$ must be a tautology. Otherwise, we have found a way to make $P \rightarrow Q$ false, and it is not a tautology.

What we have described is a set of instructions—a procedure—for carrying out the task of determining whether $P \rightarrow Q$ is a tautology. This procedure can be executed by mechanically following the instructions; in a finite amount of time, we will have the answer. In computer science terms, the procedure is an *algorithm*.

DEFINITION ALGORITHM

An **algorithm** is a set of instructions that can be mechanically executed in a finite amount of time in order to solve some problem.

Algorithms constitute the very heart of computer science, and we will have much to say about them throughout this book. You are probably already aware that the major task in writing a computer program for solving a problem consists of devising an algorithm (a procedure) to produce the problem solution.

Algorithms are often described in a form that is a middle ground between a purely verbal description in paragraph form (as we gave for deciding whether $P \rightarrow Q$ is a tautology) and a computer program (that, if executed, would actually carry out the steps of the algorithm) written in a programming language. This compromise form to describe algorithms is called **pseudocode**. An algorithm written in pseudocode should not be hard to understand even if you know nothing about computer programming. The only thing to note about the pseudocode used in this book is that lines preceded by double slashes (//) are explanatory comments, not part of the algorithm itself.

Following is a pseudocode form of the algorithm to determine whether $P \rightarrow Q$ is a tautology.

ALGORITHM *TAUTOLOGYTEST*

```

TautologyTest (wff  $P$ ; wff  $Q$ )
//Given wffs  $P$  and  $Q$ , decides whether the wff  $P \rightarrow Q$  is a tautology.

//Assume  $P \rightarrow Q$  is not a tautology
 $P = \text{true}$            //assign T to  $P$ 
 $Q = \text{false}$         //assign F to  $Q$ 

repeat
  for each compound wff already assigned a truth value,
    assign the truth values determined for its components
until all occurrences of statements letters have truth values

if some letter has two truth values
then //contradiction, assumption false
  write (" $P \rightarrow Q$  is a tautology.")
else //found a way to make  $P \rightarrow Q$  false
  write (" $P \rightarrow Q$  is not a tautology.")
end if
end TautologyTest

```

The algorithm first assigns the truth values “true” to P and “false” to Q , consistent with the assumption that $P \rightarrow Q$ is not a tautology. The algorithm then enters a *loop*, where a sequence of steps is repeated until some condition is met. Within the loop, truth assignments continue to be made to smaller and smaller components of the original P and Q until all occurrences of individual statement letters have truth values. Then the algorithm tests whether a contradiction has occurred, and writes out the information about whether $P \rightarrow Q$ is a tautology.

EXAMPLE 8

Consider the wff $(A \rightarrow B) \rightarrow (B' \rightarrow A')$. This matches the pattern needed in order to use algorithm *TautologyTest*, namely $P \rightarrow Q$, where P is $A \rightarrow B$ and Q is $B' \rightarrow A'$. Following the algorithm, we first assign truth values

$$A \rightarrow B \text{ true and } B' \rightarrow A' \text{ false}$$

Moving on to the loop, the assignment of false to the compound statement $B' \rightarrow A'$ determines the further assignments

$$B' \text{ true and } A' \text{ false}$$

or

$$B \text{ false and } A \text{ true}$$

Now working with P , A true and $A \rightarrow B$ true determines the assignment

$$B \text{ true}$$

At this point all occurrences of statement letters have truth values, as follows:

$$\begin{array}{ccc} A-T & B-T & \\ \hline (A \rightarrow B) & \rightarrow & (B' \rightarrow A') \\ \hline T & & F \end{array}$$

This terminates the loop. In the final step of the algorithm, B now has an assignment of both T and F , so the algorithm decides that $(A \rightarrow B) \rightarrow (B' \rightarrow A')$ is a tautology. Actually, we learned this earlier (in Practice 7(d)) by building a truth table. ●

REMINDER

Algorithm *TautologyTest* applies only when the main connective is \rightarrow .

Algorithm *TautologyTest* decides whether wffs of a certain form, namely, those where the main logical connective is \rightarrow , are tautologies. However, the process of building a truth table and then examining all the truth values in the final column constitutes an algorithm to decide whether an arbitrary wff is a tautology. This second algorithm is therefore more powerful because it solves a more general problem, but algorithm *TautologyTest* is usually faster for those wffs to which it applies.

Can “And” Ever Be “Or”?

In 2003, OfficeMax sued the United States government (the Internal Revenue Service) for a return of excise tax that OfficeMax had paid for telephone service. This was not a trivial financial matter—OfficeMax had paid over \$380,000.00 in telephone excise tax. To understand the nature of the argument, we need a brief history of the federal tax on telephone service.

The first telephone tax was enacted by Congress in 1898 (22 years after the invention of the telephone by Alexander Graham Bell). The tax was intended to help pay the federal debt incurred by the Spanish-American War, and it was repealed, as planned, in 1902. Over subsequent years, the telephone tax came and went with fluctuating rates as the government incurred debt. Resurrected again in 1932, the tax, in one form or another, has been in effect ever since. In 1965, Congress defined local phone service and “toll telephone service” (long-distance calls) as two categories of taxable service, and it set the tax rate at 3%. Of interest to this discussion is the definition Congress gave at this time of “toll telephone service,” which states in part that it is “a telephonic quality communication for which there is a toll charge which varies in amount with the distance and elapsed transmission time of each individual communication.” Keep in mind that in 1965, there was essentially only a single telephone service provider in the United States, namely AT&T, and at that time AT&T charges were based on both the duration and the distance of each call. By the 1990s, AT&T had been broken up and there were a number of competitive telephone companies. In addition, telephone companies began to charge a flat rate per minute for nationwide long-distance calls. The phone companies collected the federal excise tax from their customers and passed the tax on to the federal government.

OfficeMax used MCI as its phone service provider from 1999 to 2002, during which time MCI collected the excise tax from OfficeMax. In 2003, OfficeMax sued the federal government for a refund of the excise taxes MCI had collected on the basis that MCI was not providing “toll telephone service” as defined by Congress in 1965 because MCI was charging a rate based not on time and distance but only on time. Here is the issue: What exactly is the meaning of the word “and” in “varies in amount with the distance *and* elapsed transmission time”?

OfficeMax argument: “And” means the conjunctive “and,” as the truth table for “and” is defined in

formal logic. For the tax to apply, the phone company had to charge its customers a rate based on both time and distance.

Internal Revenue Service argument: Elsewhere in this same legislation, Congress did use “and” in a disjunctive sense when it defined “communication services” as “local telephone service, toll telephone service, and teletypewriter exchange service.” Because these three are mutually exclusive, “and” here could not have a conjunctive meaning.



The majority opinion of the United States Court of Appeals for the Sixth Circuit, in 2005, agreed with OfficeMax. Its reasoning was (1) dictionary definitions, legal usage guides, and case law assert that “and” is generally conjunctive, (2) the conjunctive usage is consistent with the billing mechanism used by the only telephone company in existence at the time the law was written, (3) the disjunctive interpretation would allow the possibility of a telephone charge based solely on distance, which is a ridiculous idea that Congress surely did not intend, and (4) lower courts had found in favor of OfficeMax. In short, the IRS lost this case and a number of similar cases, and in 2006 it announced that phone service that is charged on time and not distance is not taxable. (The 3% excise tax on local telephone service is still in effect.)

One must, however, appreciate the humor of the dissenting opinion in the OfficeMax case: “A host separately asked two prospective guests what they liked to drink. One said, “I like bourbon and water.” The other said, “I like beer and wine.” When the second guest arrived at the event, the host served the guest a glass of beer mixed with wine. “What’s that awful drink?” said the guest, to which the host answered, “You said you liked beer and wine.” Sometimes we apparently do use “and” in a disjunctive sense. So here is a legal—and financial—case that hinged on the truth table for the logical connective AND. How cool is that?!

OFFICEMAX, INC., Plaintiff-Appellee, v. UNITED STATES of America, Defendant-Appellant, No. 04-4009, United States Court of Appeals, Sixth Circuit, Argued: July 29, 2005, Decided and Filed: November 2, 2005, 428 F.3d 583. Online at <http://law.justia.com/cases/federal/appellate-courts/F3/428/583/565375/>

SECTION 1.1 REVIEW

TECHNIQUES

-  Construct truth tables for compound wffs.
-  Recognize tautologies and contradictions.

MAIN IDEAS

- Wffs are symbolic representations of statements.

- Truth values for compound wffs depend on the truth values of their components and the types of connectives used.
- Tautologies are “intrinsically true” wffs—true for all truth values.

EXERCISES 1.1

1. Which of the following sentences are statements?
 - a. The moon is made of green cheese.
 - b. He is certainly a tall man.
 - c. Two is a prime number.
 - d. The game will be over by 4:00.
 - e. Next year interest rates will rise.
 - f. Next year interest rates will fall.
 - g. $x^2 - 4 = 0$
2. What is the truth value of each of the following statements?
 - a. 8 is even or 6 is odd.
 - b. 8 is even and 6 is odd.
 - c. 8 is odd or 6 is odd.
 - d. 8 is odd and 6 is odd.
 - e. If 8 is odd, then 6 is odd.
 - f. If 8 is even, then 6 is odd.
 - g. If 8 is odd, then 6 is even.
 - h. If 8 is odd and 6 is even, then $8 < 6$.
3. Given the truth values A true, B false, and C true, what is the truth value of each of the following wffs?

a. $A \wedge (B \vee C)$	c. $(A \wedge B)' \vee C$
b. $(A \wedge B) \vee C$	d. $A' \vee (B' \wedge C)'$
4. Given the truth values A false, B true, and C true, what is the truth value of each of the following wffs?

a. $A \rightarrow (B \vee C)$	c. $C \rightarrow (A' \wedge B')$
b. $(A \vee B) \rightarrow C$	d. $A \vee (B' \rightarrow C)$
5. Rewrite each of the following statements in the form “If A , then B .”
 - a. Healthy plant growth follows from sufficient water.
 - b. Increased availability of information is a necessary condition for further technological advances.
 - c. Errors were introduced only if there was a modification of the program.
 - d. Fuel savings implies good insulation or storm windows throughout.

6. Rewrite each of the following statements in the form “If A , then B .”
 - a. Candidate Lu winning the election will be a sufficient condition for property taxes to increase.
 - b. The user clicks Pause only if the game level changes.
 - c. The components are scarce, therefore the price increases.
 - d. Healthy hair is a necessary condition for good shampoo.
7. Common English has many ways to describe logical connectives. Write a wff for each of the following expressions.
 - a. Either A or B
 - b. Neither A nor B
8. Common English has many ways to describe logical connectives. Write a wff for each of the following expressions.
 - a. B whenever A
 - b. A is derived from B
 - c. A indicates B
 - d. A exactly when B
9. Several forms of negation are given for each of the following statements. Which are correct?
 - a. The answer is either 2 or 3.
 1. Neither 2 nor 3 is the answer.
 2. The answer is not 2 or not 3.
 3. The answer is not 2 and it is not 3.
 - b. Cucumbers are green and seedy.
 1. Cucumbers are not green and not seedy.
 2. Cucumbers are not green or not seedy.
 3. Cucumbers are green and not seedy.
 - c. $2 < 7$ and 3 is odd.
 1. $2 > 7$ and 3 is even.
 2. $2 \geq 7$ and 3 is even.
 3. $2 \geq 7$ or 3 is odd.
 4. $2 \geq 7$ or 3 is even.
10. Several forms of negation are given for each of the following statements. Which are correct?
 - a. The carton is sealed or the milk is sour.
 1. The milk is not sour or the carton is not sealed.
 2. The carton is not sealed and also the milk is not sour.
 3. If the carton is not sealed, then the milk will be sour.
 - b. Flowers will bloom only if it rains.
 1. The flowers will bloom but it will not rain.
 2. The flowers will not bloom and it will not rain.
 3. The flowers will not bloom or else it will not rain.
 - c. If you build it, they will come.
 1. If you build it, then they won't come.
 2. You don't build it, but they do come.
 3. You build it, but they don't come.
11. Write the negation of each statement.
 - a. If the food is good, then the service is excellent.
 - b. Either the food is good or the service is excellent.

- c. Either the food is good and the service is excellent, or else the price is high.
 d. Neither the food is good nor the service excellent.
 e. If the price is high, then the food is good and the service is excellent.
12. Write the negation of each statement.
- The processor is fast but the printer is slow.
 - The processor is fast or else the printer is slow.
 - If the processor is fast, then the printer is slow.
 - Either the processor is fast and the printer is slow, or else the file is damaged.
 - If the file is not damaged and the processor is fast, then the printer is slow.
 - The printer is slow only if the file is damaged.
13. Using the letters indicated for the component statements, translate the following compound statements into symbolic notation.
- A : prices go up; B : housing will be plentiful; C : housing will be expensive
 If prices go up, then housing will be plentiful and expensive; but if housing is not expensive, then it will still be plentiful.
 - A : going to bed; B : going swimming; C : changing clothes
 Either going to bed or going swimming is a sufficient condition for changing clothes; however, changing clothes does not mean going swimming.
 - A : it will rain; B : it will snow
 Either it will rain or it will snow but not both.
 - A : Janet wins; B : Janet loses; C : Janet will be tired
 If Janet wins or if she loses, she will be tired.
 - A : Janet wins; B : Janet loses; C : Janet will be tired
 Either Janet will win or, if she loses, she will be tired.
14. Using the letters indicated for the component statements, translate the following compound statements into symbolic notation.
- A : the tractor wins; B : the truck wins; C : the race will be exciting.
 Whether the tractor wins or the truck wins, the race will be exciting
 - A : snow; B : rain; C : yesterday was cloudy
 Yesterday was cloudy but there was neither snow nor rain.
 - A : Koalas will be saved; B : climate change is addressed; C : rising water levels
 Koalas will be saved only if climate change is addressed; furthermore, failure to address climate change will cause rising water levels.
 - A : the city's economy will improve; B : a strong school system
 The city's economy will improve conditional upon a strong school system.
 - A : the city's economy will improve; B : a strong school system
 A strong school system is a necessary condition for the city's economy to improve.
15. Let A , B , and C be the following statements:
- A Roses are red.
 B Violets are blue.
 C Sugar is sweet.
- Translate the following compound statements into symbolic notation.
- Roses are red and violets are blue.
 - Roses are red, and either violets are blue or sugar is sweet.

- c. Whenever violets are blue, roses are red and sugar is sweet.
 d. Roses are red only if violets aren't blue or sugar is sour.
 e. Roses are red and, if sugar is sour, then either violets aren't blue or sugar is sweet.
16. Let A , B , and C , and D be the following statements:
 A The villain is French.
 B The hero is American.
 C The heroine is British.
 D The movie is good.
 Translate the following compound statements into symbolic notation.
- a. The hero is American and the movie is good.
 b. Although the villain is French, the movie is good.
 c. If the movie is good, then either the hero is American or the heroine is British.
 d. The hero is not American, but the villain is French.
 e. A British heroine is a necessary condition for the movie to be good.
17. Use A , B , and C as defined in Exercise 15 to translate the following statements into English.
- | | |
|--------------------------------------|-----------------------------------|
| a. $B \vee C'$ | e. $(B \wedge C')' \rightarrow A$ |
| b. $B' \vee (A \rightarrow C)$ | f. $A \vee (B \wedge C')$ |
| c. $(C \wedge A') \leftrightarrow B$ | g. $(A \vee B) \wedge C'$ |
| d. $C \wedge (A' \leftrightarrow B)$ | |
18. Use A , B , and C as defined in Exercise 16 to translate the following statements into English.
- | | |
|--------------------------------|--|
| a. $B \rightarrow A'$ | e. $A \leftrightarrow (B \vee C)$ |
| b. $B \wedge C \wedge D'$ | f. $D' \rightarrow (A \vee C)'$ |
| c. $B \rightarrow (C \vee A)$ | g. $(C \rightarrow D) \wedge (A \rightarrow B')$ |
| d. $(A \vee C) \rightarrow B'$ | |
19. Using letters H , K , A for the component statements, translate the following compound statements into symbolic notation.
- a. If the horse is fresh, then the knight will win.
 b. The knight will win only if the horse is fresh and the armor is strong.
 c. A fresh horse is a necessary condition for the knight to win.
 d. The knight will win if and only if the armor is strong.
 e. A sufficient condition for the knight to win is that the armor is strong or the horse is fresh.
20. Using letters A , T , E for the component statements, translate the following compound statements into symbolic notation.
- a. If Anita wins the election, then tax rates will be reduced.
 b. Tax rates will be reduced only if Anita wins the election and the economy remains strong.
 c. Tax rates will be reduced if the economy remains strong.
 d. A strong economy will follow from Anita winning the election.
 e. The economy will remain strong if and only if Anita wins the election or tax rates are reduced.
21. Using letters F , B , S for the component statements, translate the following compound statements into symbolic notation.
- a. Plentiful fish are a sufficient condition for bears to be happy.
 b. Bears are happy only if there are plentiful fish.

- c. Unhappy bears means that the fish are not plentiful and also that there is heavy snow.
 d. Unhappy bears are a necessary condition for heavy snow.
 e. The snow is heavy if and only if the fish are not plentiful.
22. Using letters P, C, B, L for the component statements, translate the following compound statements into symbolic notation.
- If the project is finished soon, then the client will be happy and the bills will be paid.
 - If the bills are not paid, then the lights will go out.
 - The project will be finished soon only if the lights do not go out.
 - If the bills are not paid and the lights go out, then the client will not be happy.
 - The bills will be paid if and only if the project is finished soon, or else the lights go out.
 - The bills will be paid if and only if either the project is finished soon or the lights go out.
23. Construct truth tables for the following wffs. Note any tautologies or contradictions.
- $(A \rightarrow B) \leftrightarrow A' \vee B$
 - $(A \wedge B) \vee C \rightarrow A \wedge (B \vee C)$
 - $A \wedge (A' \vee B)'$
 - $A \wedge B \rightarrow A'$
 - $(A \rightarrow B) \rightarrow [(A \vee C) \rightarrow (B \vee C)]$
24. Construct truth tables for the following wffs. Note any tautologies or contradictions.
- $A \rightarrow (B \rightarrow A)$
 - $A \wedge B \leftrightarrow B' \vee A'$
 - $(A \vee B') \wedge (A \wedge B)'$
 - $[(A \vee B) \wedge C'] \rightarrow A' \vee C$
 - $A' \rightarrow (B \vee C')$
25. Verify the equivalences in the list on page 9 by constructing truth tables. (We have already verified 1a, 4b, and 5a.)
26. Verify by constructing truth tables that the following wffs are tautologies. Note that the tautologies in parts b, e, f, and g produce equivalences such as $(A')' \leftrightarrow A$.
- $A \vee A'$
 - $(A')' \leftrightarrow A$
 - $A \wedge B \rightarrow B$
 - $A \rightarrow A \vee B$
 - $(A \vee B)' \leftrightarrow A' \wedge B'$ (De Morgan's law)
 - $(A \wedge B)' \leftrightarrow A' \vee B'$ (De Morgan's law)
 - $A \vee A \leftrightarrow A$
27. Prove the following tautologies by starting with the left side and finding a series of equivalent wffs that will convert the left side into the right side. You may use any of the equivalencies in the list on page 9 or the equivalencies from Exercise 26.
- $(A \wedge B') \wedge C \leftrightarrow (A \wedge C) \wedge B'$
 - $(A \vee B) \wedge (A \vee B') \leftrightarrow A$
 - $A \vee (B \wedge A') \leftrightarrow A \vee B$
28. Prove the following tautologies by starting with the left side and finding a series of equivalent wffs that will convert the left side into the right side. You may use any of the equivalencies in the list on page 9 or the equivalencies from Exercise 26.
- $(A \wedge B')' \vee B \leftrightarrow A' \vee B$
 - $A \wedge (A \wedge B')' \leftrightarrow A \wedge B$
 - $(A \wedge B)' \wedge (A \vee B') \leftrightarrow B'$
29. We mentioned that $(A \wedge B) \vee C$ cannot be proved equivalent to $A \wedge (B \vee C)$ using either of the associative tautological equivalences, but perhaps it can be proved some other way. Are these two wffs equivalent? Prove or disprove.

30. Let P be the wff $A \rightarrow B$. Prove or disprove whether P is equivalent to any of the following related wffs.
- the *converse* of P , $B \rightarrow A$
 - the *inverse* of P , $A' \rightarrow B'$
 - the *contrapositive* of P , $B' \rightarrow A'$
31. Write a logical expression for a Web search engine to find sites pertaining to dogs that are not retrievers.
32. Write a logical expression for a Web search engine to find sites pertaining to oil paintings by Van Gogh or Rembrandt but not Vermeer.
33. Write a logical expression for a Web search engine to find sites pertaining to novels or plays about AIDS.
34. Write a logical expression for a Web search engine to find sites pertaining to coastal wetlands in Louisiana but not in Alabama.
35. Consider the following pseudocode.

```

repeat
  i = 1
  read a value for x
  if ((x < 5.0) and (2x < 10.7)) or ( $\sqrt{5x} > 5.1$ ) then
    write the value of x
  end if
  increase i by 1
until i > 5

```

The input values for x are 1.0, 5.1, 2.4, 7.2, and 5.3. What are the output values?

36. Suppose that A , B , and C represent conditions that will be true or false when a certain computer program is executed. Suppose further that you want the program to carry out a certain task only when A or B is true (but not both) and C is false. Using A , B , and C and the connectives AND, OR, and NOT, write a statement that will be true only under these conditions.
37. Rewrite the following statement form with a simplified conditional expression, where the function $odd(n)$ returns true if n is odd.

```

if not((Value1 < Value2) or odd(Number))
or (not(Value1 < Value2) and odd(Number)) then
  statement1
else
  statement2
end if

```

38. You want your program to execute statement 1 when A is false, B is false, and C is true, and to execute statement 2 otherwise. You wrote

```

if not(A and B) and C then
  statement 1
else
  statement 2
end if

```

Does this do what you want?

39. Verify that $A \rightarrow B$ is equivalent to $A' \vee B$.
40. a. Using Exercise 39 and other equivalences, prove that the negation of $A \rightarrow B$ is equivalent to $A \wedge B'$
- b. Write the negation of the statement "If Sam passed his bar exam, then he will get the job."

41. Use algorithm *TautologyTest* to prove that the following expressions are tautologies.
- $[B' \wedge (A \rightarrow B)] \rightarrow A'$
 - $[(A \rightarrow B) \wedge A] \rightarrow B$
 - $(A \vee B) \wedge A' \rightarrow B$
42. Use algorithm *TautologyTest* to prove that the following expressions are tautologies.
- $(A \wedge B) \wedge B' \rightarrow A$
 - $(A \wedge B') \rightarrow (A \rightarrow B)'$
 - $(A \wedge B)' \vee B' \rightarrow A' \vee B'$
43. A memory chip from a digital camera has 2^5 bistable (ON-OFF) memory elements. What is the total number of ON-OFF configurations?
44. In each case, construct compound wffs P and Q so that the given statement is a tautology.
- $P \wedge Q$
 - $P \rightarrow P'$
 - $P \wedge (Q \rightarrow P')$
45. From the truth table for $A \vee B$, the value of $A \vee B$ is true if A is true, if B is true, or if both are true. This use of the word “or,” where the result is true if both components are true, is called the *inclusive or*. It is the inclusive or that is understood in the sentence, “We may have rain or drizzle tomorrow,” which might also be expressed as, “We may have rain or drizzle or both tomorrow.” Another use of the word “or” in the English language is the *exclusive or*, sometimes written **XOR**, in which the result is false when both components are true. The exclusive or is understood in the sentence, “At the intersection, you should turn north or south,” (but obviously not both). Exclusive or is symbolized by $A \oplus B$. Write the truth table for the exclusive or.
46. Prove that $A \oplus B \leftrightarrow (A \leftrightarrow B)'$ is a tautology. Explain why this makes sense.

Exercises 47–50 show that defining four basic logical connectives (conjunction, disjunction, implication, and negation) is a convenience rather than a necessity because certain pairs of connectives are enough to express any wff. Exercises 51–52 show that a single connective, properly defined, is sufficient.

47. Every compound statement is equivalent to a statement using only the connectives of conjunction and negation. To see this, we need to find equivalent wffs for $A \vee B$ and for $A \rightarrow B$ that use only \wedge and $'$. These new statements can replace, respectively, any occurrences of $A \vee B$ and $A \rightarrow B$. (The connective \leftrightarrow was defined in terms of other connectives, so we already know that it can be replaced by a statement using these other connectives.)
- Show that $A \vee B$ is equivalent to $(A' \wedge B)'$
 - Show that $A \rightarrow B$ is equivalent to $(A \wedge B)'$
48. Show that every compound wff is equivalent to a wff using only the connectives of \vee and $'$. (*Hint*: See Exercise 47.)
49. Show that every compound wff is equivalent to a wff using only the connectives of \rightarrow and $'$. (*Hint*: See Exercise 47.)
50. Prove that there are compound statements that are not equivalent to any statement using only the connectives \rightarrow and \vee .
51. The binary connective $|$ is called the *Sheffer stroke*, named for the American logic professor Henry Sheffer, who proved in 1913 that this single connective is the only one needed. The truth table for $|$ is given here. Sheffer also coined the term “Boolean algebra,” the topic of Chapter 8, where we will see that this truth table represents the NAND gate.

A	B	A B
T	T	F
T	F	T
F	T	T
F	F	T

Show that every compound wff is equivalent to a wff using only the connective $|$. (*Hint*: Use Exercise 47 and find equivalent statements for $A \wedge B$ and A' in terms of $|$.)

52. The binary connective \downarrow is called the *Peirce arrow*, named for American philosopher Charles Peirce (not for the antique automobile). The truth table for \downarrow is given here. In Chapter 8 we will see that this truth table represents the NOR gate

A	B	A\downarrowB
T	T	F
T	F	F
F	T	F
F	F	T

Show that every compound statement is equivalent to a statement using only the connective \downarrow . (*Hint*: See Exercise 51.)

53. Propositional wffs and truth tables belong to a system of *two-valued logic* because everything has one of two values, False or True. *Three-valued logic* allows a third value of Null or “unknown” (Section 5.3 discusses the implications of three-valued logic on databases). The truth tables for this three-valued system follow.

A	B	A\wedgeB
T	T	T
T	F	F
T	N	N
F	T	F
F	F	F
F	N	F
N	T	N
N	F	F
N	N	N

A	B	A\veeB
T	T	T
T	F	T
T	N	T
F	T	T
F	F	F
F	N	N
N	T	T
N	F	N
N	N	N

A	A'
T	F
F	T
N	N

- a. Viewing N as “unknown”, explain why it is reasonable to define $T \wedge N = N$, $F \vee N = N$, and $N' = N$.

Suppose the statement, “Flight 237 is on time,” is true, the statement, “Runway conditions are icy,” is false, and the truth value of the statement, “Flight 51 is on time,” is unknown. Find the truth values of the following statements.

- b. Runway conditions are not icy and flight 51 is on time.
 c. Flight 51 is on time and flight 237 is not.
 d. Flight 51 is not on time or runway conditions are not icy.

54. Propositional wffs and truth tables belong to a system of *two-valued logic* because everything has one of two values, F or T, which we can think of as 0 or 1. In *fuzzy logic*, or *many-valued logic*, statement letters are assigned values in a range between 0 and 1 to reflect some “probability” to which they are false or true. A statement letter with a truth value of 0.9 is “mostly true” or “has a high probability of being true” while a statement letter with a truth value of 0.05 “has a very high probability of being false.” Fuzzy logic

is used to manage decisions in many imprecise situations such as robotics, manufacturing, or instrument control. Truth values for compound statements are determined as follows.

A' has the truth value $1 - A$.

$A \wedge B$ has the truth value that is the minimum of the values of A and of B .

$A \vee B$ has the truth value that is the maximum of the values of A and B .

a. Explain why these are reasonable assignments for the truth values of A' , $A \wedge B$, and $A \vee B$.

Suppose the statement, “Flight 237 is on time,” is estimated to have a truth value of 0.84 and the statement, “Runway conditions are icy,” is estimated to have a truth value of 0.12. Find the truth values of the following statements.

b. Runway conditions are not icy.

c. Runway conditions are icy and flight 237 is on time.

d. Runway conditions are icy or flight 237 is not on time.

55. In a three-valued logic system as described in Exercise 53, how many rows are needed for a truth table with n statement letters?

56. In 2003, then U.S. Secretary of Defense Donald Rumsfeld won Britain’s Plain English Campaign 2003 Golden Bull Award for this statement: “Reports that say that something hasn’t happened are always interesting to me, because as we know, there are known knowns, there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns—the ones we don’t know we don’t know.”

What possibility did Secretary Rumsfeld omit?

57. Four machines, A , B , C , and D , are connected on a computer network. It is feared that a computer virus may have infected the network. Your security team makes the following statements:

a. If D is infected, then so is C .

b. If C is infected, then so is A .

c. If D is clean, then B is clean but C is infected.

d. If A is infected, then either B is infected or C is clean.

Assuming that these statements are all true, what can you conclude? Explain your reasoning.

58. The Dillies have five teenaged children, two boys named Ollie and Rollie, and three girls named Mellie, Nellie, and Pollie. Each is a different number of years old, from 13 to 17. There are three bedrooms for the children in the Dillie house, so two share the yellow room, two share the white room, and one alone has the smaller green room. Can you match each one’s name and age, and tell who sleeps where?

a. No one shares a room with a sibling of the opposite sex.

b. Pollie is exactly one year older than Mellie.

c. The two teenagers who share the yellow room are two years apart in age.

d. The two who share the white room are three years apart in age.

e. Rollie is somewhat older than Ollie but somewhat younger than the sibling who has the green room.²

Determine who sleeps in each room and what their ages are. Explain your reasoning.

59. An advertisement for a restaurant at an exclusive club in Honolulu says, “Members and nonmembers only.” Give two possible interpretations of this statement.

60. The following newspaper headline was printed during a murder trial:

“I am a liar” says murder defendant!

Can the jury reach any conclusion from this statement?

²Scott Marley, Dell Logic Puzzles, April, 1998

In Exercises 61–64, you are traveling in a certain country where every inhabitant is either a truth teller who always tells the truth or a liar who always lies.³

61. You meet two of the inhabitants of this country, Percival and Llewellyn. Percival says, “At least one of us is a liar.” Is Percival a liar or a truth teller? What about Llewellyn? Explain your answer.
62. Traveling on, you meet Merlin and Meredith. Merlin says, “If I am a truth teller, then Meredith is a truth teller.” Is Merlin a liar or a truth teller? What about Meredith? Explain your answer.
63. Next, you meet Rothwold and Grymlin. Rothwold says, “Either I am a liar or Grymlin is a truth teller.” Is Rothwold a liar or a truth teller? What about Grymlin? Explain your answer.
64. Finally, you meet Gwendolyn and Merrilaine. Gwendolyn says, “I am a liar but Merrilaine is not.” Is Gwendolyn a liar or a truth teller? What about Merrilaine?

SECTION 1.2 PROPOSITIONAL LOGIC

The argument of the defense attorney at the beginning of this chapter made a number of (supposedly true) statements and then asked the jury to draw a specific conclusion based on those statements. In Section 1.1, we used the notation of formal logic to represent statements in symbolic form as wffs; because statements are sometimes called *propositions*, these wffs are also called **propositional wffs**. Now we want to use tools from formal logic to see how to reach logical conclusions based on given statements. The formal system that uses propositional wffs is called **propositional logic**, **statement logic**, or **propositional calculus**. (The word *calculus* is used here in the more general sense of “calculation” or “reasoning,” not “differentiating” or “integrating.”)

Valid Arguments

An argument can be represented in symbolic form as

$$P_1 \wedge P_2 \wedge P_3 \wedge \dots \wedge P_n \rightarrow Q$$

where P_1, P_2, \dots, P_n are the given statements, called the **hypotheses**, of the argument, and Q is the **conclusion** of the argument. As usual, the P 's and the Q represent wffs, not merely statement letters. When should this be considered a *valid argument*? This question can be stated in several equivalent ways:

- When can Q be *logically deduced from* P_1, \dots, P_n ?
- When is Q a *logical conclusion from* P_1, \dots, P_n ?
- When does P_1, \dots, P_n *logically imply* Q ?
- When does Q *follow logically from* P_1, \dots, P_n ?

and so forth.

³For more puzzles about “knights” and “knaves,” see *What Is the Name of This Book?* by the logician—and magician—Raymond Smullyan (Prentice-Hall, 1978).

An informal answer is that Q is a logical conclusion from P_1, \dots, P_n whenever the truth of P_1, \dots, P_n implies the truth of Q . In other words, when the implication

$$P_1 \wedge P_2 \wedge P_3 \wedge \dots \wedge P_n \rightarrow Q$$

is true. (Of course, this implication is true if any of the hypotheses is false, but in an argument we usually care about what happens when all the hypotheses are true.) Furthermore, this implication should be true based on the relationship of the conclusion to the hypotheses, not on any incidental knowledge we may happen to have about Q .

EXAMPLE 9

Consider the following argument:

George Washington was the first president of the United States. Thomas Jefferson wrote the Declaration of Independence. Therefore, every day has 24 hours.

This argument has the two hypotheses

1. George Washington was the first president of the United States.
2. Thomas Jefferson wrote the Declaration of Independence.

and the conclusion

Every day has 24 hours.

Even though each of the individual hypotheses, as well as the conclusion, is a true statement, we would *not* consider this argument valid. The conclusion is merely an isolated true fact, not at all related to or “following from” the hypotheses. ●

A valid argument should therefore be true based entirely on its internal structure; it should be “intrinsically true.” Therefore we make the following formal definition.

● **DEFINITION** **VALID ARGUMENT**
The propositional wff

$$P_1 \wedge P_2 \wedge P_3 \wedge \dots \wedge P_n \rightarrow Q$$

is a **valid argument** when it is a tautology.

The argument in Example 9 would be symbolized as

$$A \wedge B \rightarrow C$$

which is clearly not a tautology.

EXAMPLE 10

Consider the following argument. If George Washington was the first president of the United States, then John Adams was the first vice president. George Washington was the first president of the United States. Therefore John Adams was the first vice president.

This argument has the two hypotheses:

1. If George Washington was the first president of the United States, then John Adams was the first vice president.
2. George Washington was the first president of the United States.

and the conclusion

John Adams was the first vice president.

A symbolic representation of this argument has the form

$$(A \rightarrow B) \wedge A \rightarrow B$$

A truth table or algorithm *TautologyTest* establishes that this argument is a tautology. The argument is valid; its form is such that the conclusion follows inevitably from the hypotheses. In fact, this form of argument, known by its Latin name of *modus ponens* (“method of assertion”), is one of the rules of reasoning we will use to build propositional logic. ●

To test whether a wff $P_1 \wedge P_2 \wedge P_3 \wedge \dots \wedge P_n \rightarrow Q$ is a tautology, we could build a truth table or use algorithm *TautologyTest*. Instead, we will turn to formal logic, which uses a system of **derivation rules** that manipulate wffs in a truth-preserving manner. You begin with the hypotheses P_1, \dots, P_n (assumed true) and attempt to apply the manipulation rules in such a way as to end up with the conclusion Q (which must then also be true because truth is preserved under the rules).

DEFINITION PROOF SEQUENCE

A **proof sequence** is a sequence of wffs in which each wff is either a hypothesis or the result of applying one of the formal system’s derivation rules to earlier wffs in the sequence.

Using formal logic to prove that Q is a valid conclusion from P_1, \dots, P_n , we must produce a proof sequence of the form

P_1	(hypothesis)
P_2	(hypothesis)
⋮	
P_n	(hypothesis)
wff ₁	(obtained by applying a derivation rule to earlier wffs)
wff ₂	(obtained by applying a derivation rule to earlier wffs)
⋮	
Q	(obtained by applying a derivation rule to earlier wffs)

The derivation rules for a formal system must be carefully chosen. If they are too powerful, then they won’t be truth preserving and we’ll be able to deduce anything at all from a given set of hypotheses. If they are too weak, there will be logical conclusions that we won’t be able to prove from given hypotheses. We want a formal logic system that is **correct** (*only* valid arguments should be provable) and **complete** (*every* valid argument should be provable). In addition, the

derivation rules should be kept to a minimum in order to make the formal system manageable. We would like the system to have the smallest set of rules that still allows it to be complete.

Derivation Rules for Propositional Logic

The derivation rules for propositional logic fall into two categories, equivalence rules and inference rules. Equivalence rules allow individual wffs to be rewritten, while inference rules allow new wffs to be derived from previous wffs in the proof sequence.

Equivalence rules state that certain pairs of wffs R and S are equivalent. Remember from Section 1.1 that $R \Leftrightarrow S$ means that $R \leftrightarrow S$ is a tautology and that S can be substituted for R in any wff with no change to the truth value of that wff. Equivalence rules are therefore truth-preserving; a true wff remains true if such a substitution is done within it.

Table 1.11 lists the equivalence rules we will use in our formal system for propositional logic. (Additional rules could be formulated based on other tautologies, but we are trying to keep our rule set to a minimum.) Each is given a name to make it easier to identify its use in a proof sequence. We saw the commutative and associative rules, as well as De Morgan's laws, in Section 1.1. There they were given for statement letters only, here they are given for any wffs P, Q, R , but they are still tautologies.

TABLE 1.11		
Equivalence Rules		
Expression	Equivalent to	Name/Abbreviation for Rule
$P \vee Q$ $P \wedge Q$	$Q \vee P$ $Q \wedge P$	Commutative—comm
$(P \vee Q) \vee R$ $(P \wedge Q) \wedge R$	$P \vee (Q \vee R)$ $P \wedge (Q \wedge R)$	Associative—ass
$(P \vee Q)'$ $(P \wedge Q)'$	$P' \wedge Q'$ $P' \vee Q'$	De Morgan's Laws—De Morgan
$P \rightarrow Q$	$P' \vee Q$	Implication—imp
P	$(P)'$	Double negation—dn
$P \leftrightarrow Q$	$(P \rightarrow Q) \wedge (Q \rightarrow P)$	Definition of equivalence—equ

PRACTICE 9 Prove the implication rule.

That is, prove that

$$(P \rightarrow Q) \leftrightarrow (P' \vee Q)$$

is a tautology. ■

EXAMPLE 11

Suppose that one hypothesis of a propositional argument can be symbolized as

$$(A' \vee B') \vee C$$

Then a proof sequence for the argument could begin with the following steps:

1. $(A' \vee B') \vee C$ hyp (hypothesis)
2. $(A \wedge B)' \vee C$ 1, De Morgan
3. $(A \wedge B) \rightarrow C$ 2, imp

The justification given for each step is not a required part of the proof sequence, but it does confirm that the step is a legitimate one. Step 1 is a hypothesis. Step 2 is derived from step 1 by applying one of De Morgan's Laws. Step 3 is derived from step 2 by using the implication rule that $P \rightarrow Q$ is equivalent to $P' \vee Q$, where P is the wff $A \wedge B$, and Q is the wff C . ●

The equivalence rules allow substitution in either direction. That is, in Example 11 we replaced $A' \vee B'$ with $(A \wedge B)'$, but in some other proof sequence, using the same rule, we might replace $(A \wedge B)'$ with $A' \vee B'$.

Inference rules say that if one or more wffs that match the first part of the rule pattern are already part of the proof sequence, we can add to the proof sequence a new wff that matches the last part of the rule pattern. Table 1.12 shows the propositional inference rules we will use, again along with their identifying names.

Inference Rules		
From	Can Derive	Name/Abbreviation for Rule
$P, P \rightarrow Q$	Q	Modus ponens—mp
$P \rightarrow Q, Q'$	P'	Modus tollens—mt
P, Q	$P \wedge Q$	Conjunction—con
$P \wedge Q$	P, Q	Simplification—sim
P	$P \vee Q$	Addition—add

Unlike equivalence rules, inference rules do not work in both directions. We cannot “reverse” the addition rule in Table 1.12; from $P \vee Q$, we cannot infer either P or Q .

EXAMPLE 12

Suppose that $A \rightarrow (B \wedge C)$ and A are two hypotheses of an argument. A proof sequence for the argument could begin with the following steps:

1. $A \rightarrow (B \wedge C)$ hyp
2. A hyp
3. $B \wedge C$ 1, 2, mp

The justification at step 3 is that steps 1 and 2 exactly match the pattern required for modus ponens, where P is A and Q is $B \wedge C$. Modus ponens says that Q can be derived from P and $P \rightarrow Q$. ●

PRACTICE 10 Give a next step and a justification for a proof sequence that begins

1. $(A \wedge B') \rightarrow C$ hyp
2. C' hyp

REMINDER

To use a derivation rule, wffs must exactly match the rule pattern.

The inference rules are also truth-preserving. For example, suppose that P and $P \rightarrow Q$ are both true wffs in a proof sequence. Then Q is deducible from these two wffs by modus ponens. If P and $P \rightarrow Q$ are both true, then—by the truth table for implication— Q is also true.

The derivation rules, like the tautological equivalencies of Section 1.1, represent recipes or patterns for transforming wffs. A rule can be applied only when the wffs exactly match the pattern.

EXAMPLE 13

Suppose that $(A \rightarrow B) \vee C$ and A are two hypotheses of an argument. A proof sequence for the argument could begin with the following steps:

1. $(A \rightarrow B) \vee C$ hyp
2. A hyp

Unlike Example 12, however, nothing further can be done. Modus ponens requires the presence of wffs matching the pattern P and $P \rightarrow Q$. In $P \rightarrow Q$, the main connective is an implication. The wff $(A \rightarrow B) \vee C$ has disjunction, not implication, as its main connective. Modus ponens does not apply, nor does anything else. ●

Now we are ready to work our way through a complete proof of an argument.

EXAMPLE 14

Using propositional logic, prove that the argument

$$A \wedge (B \rightarrow C) \wedge [(A \wedge B) \rightarrow (D \vee C')] \wedge B \rightarrow D$$

is valid.

We must produce a proof sequence that begins with the hypotheses and ends with the conclusion. There are four hypotheses, so this gives us lots of “ammunition” to use in the proof. The beginning of the proof is easy enough because it just involves listing the hypotheses:

1. A hyp
2. $B \rightarrow C$ hyp
3. $(A \wedge B) \rightarrow (D \vee C')$ hyp
4. B hyp

Our final goal is to arrive at D , the conclusion. But without even looking ahead, there are a couple of fairly obvious steps we can take that may or may not be helpful.

5. C 2, 4, mp
6. $A \wedge B$ 1, 4, con
7. $D \vee C'$ 3, 6, mp

At least at this point we have introduced D , but it's not by itself. Note that from step 5 we have C , which we haven't made use of. If only we had $C \rightarrow D$, we'd be

home free. Ah, look at the form of step 7; it's a disjunction, and the implication rule says that we can transform a disjunction of a certain form into an implication. The disjunction must have a negated wff on the left. We can do that:

8. $C' \vee D$ 7, comm
9. $C \rightarrow D$ 8, imp

so

10. D 5, 9, mp

As in Example 14, proof sequences involve a certain amount of rewriting just because you can and a certain amount of keeping an eye on the desired goal and what it would take to get there. Although not as mechanical as constructing a truth table, the strict rules of the game nevertheless provide a more or less mechanical way to construct the proof sequence. There are only a certain number of legitimate things that can be done at any one point in the sequence. If one choice seems to lead down a blind alley, go back and make another. Also, there may be more than one correct proof sequence; as a relatively trivial instance, steps 6 and 7 could have been done before step 5 in Example 14.

An analogy with programming, if not taken too literally, may be helpful. In traditional programming, you have known input, a desired output, and you write code to transform the given input into the desired output. You figure out the sequence of statements that will accomplish this transformation, and each program statement in that sequence must conform to the exact syntax rules of the programming language you are using, be it C++, Java, Python, or whatever. In propositional logic, you have known “input” (the hypotheses), a desired “output” (the conclusion), and you write “code statements” (a sequence of wffs) to transform the hypotheses into the conclusion. The sequence of code statements, or at least their justification, must conform to the exact syntax of the derivation rules for propositional logic.

PRACTICE 11 Using propositional logic, prove the validity of the argument.

$$[(A \vee B') \rightarrow C] \wedge (C \rightarrow D) \wedge A \rightarrow D$$

TABLE 1.13

Derivation Hints

1. Modus ponens is probably the most intuitive inference rule. Think often about trying to use it.
2. Wffs of the form $(P \wedge Q)'$ or $(P \vee Q)'$ are seldom helpful in a proof sequence. Try using De Morgan's laws to convert them into $P' \vee Q'$ and $P' \wedge Q'$, respectively, which breaks out the individual components.
3. Wffs of the form $P \vee Q$ are also seldom helpful in a proof sequence because they do not imply either P or Q . Try using double negation to convert $P \vee Q$ to $(P')' \vee Q$, and then using implication to convert to $P' \rightarrow Q$.

Deduction Method and Other Rules

REMINDER

Use the deduction method when the conclusion of what you want to prove is an implication.

Suppose the argument we seek to prove has the form

$$P_1 \wedge P_2 \wedge P_3 \wedge \dots \wedge P_n \rightarrow (R \rightarrow S)$$

where the conclusion is itself an implication. Instead of using P_1, \dots, P_n as the hypotheses and deriving $R \rightarrow S$, the *deduction method* lets us add R as an additional hypothesis and then derive S . In other words, we can instead prove

$$P_1 \wedge P_2 \wedge P_3 \wedge \dots \wedge P_n \wedge R \rightarrow S$$

This change is to our advantage because it gives us one more hypothesis, i.e., additional ammunition for the proof, and it simplifies the desired conclusion.

The deduction method approach agrees with our understanding of implication, but Exercise 55 at the end of this section provides a formal justification.

EXAMPLE 15

Use propositional logic to prove

$$[A \rightarrow (A \rightarrow B)] \rightarrow (A \rightarrow B)$$

Using the deduction method, we get two hypotheses instead of one, and we want to derive B .

1. $A \rightarrow (A \rightarrow B)$ hyp
2. A hyp
3. $A \rightarrow B$ 1, 2, mp
4. B 2, 3, mp

PRACTICE 12

Use propositional logic to prove

$$(A \rightarrow B) \wedge (B \rightarrow C) \rightarrow (A \rightarrow C)$$

The formal system we have described is correct and complete. Every argument we can prove is a tautology (the system is correct), and every implication that is a tautology is provable (the system is complete). We can easily argue for correctness because each of the derivation rules is truth-preserving. Completeness would be more difficult to prove, and we will not do so.

Correctness and completeness say that the set of derivation rules we have used is exactly right—not too strong, not too weak. Nonetheless, many formal systems for propositional logic use additional truth-preserving inference rules. We can prove these additional rules using our original rule set. Once such a rule is proved, it can be used as justification in a proof sequence because, if required, the single step invoking this rule could be replaced with the proof sequence for the rule. Nothing more can be proved by the addition of these rules, but the proof sequences might be shorter. (See Exercises 1.2 for a list of additional rules.)

EXAMPLE 16

The rule of hypothetical syllogism (hs) is

From $P \rightarrow Q$ and $Q \rightarrow R$, one can derive $P \rightarrow R$.

This rule is making the claim that

$$(P \rightarrow Q) \wedge (Q \rightarrow R) \rightarrow (P \rightarrow R)$$

is a valid argument. The proof sequence for this argument looks just like that for Practice 12. Because it is a legitimate derivation rule, hypothetical syllogism can be used to justify a step in a proof sequence. ●

EXAMPLE 17

Use propositional logic to prove

$$(A' \vee B) \wedge (B \rightarrow C) \rightarrow (A \rightarrow C)$$

The following proof sequence will do.

1. $A' \vee B$ hyp
2. $B \rightarrow C$ hyp
3. $A \rightarrow B$ 1, imp
4. $A \rightarrow C$ 2, 3, hs

Without use of the new rule, we could still have produced a proof sequence by essentially proving the new rule as part of this proof:

1. $A' \vee B$ hyp
2. $B \rightarrow C$ hyp
3. $A \rightarrow B$ 1, imp
4. A hyp
5. B 3, 4, mp
6. C 2, 5, mp

Additional rules thus can shorten proof sequences but at the expense of having to remember additional rules! ●

PRACTICE 13 Prove

$$(A \rightarrow B) \wedge (C' \vee A) \wedge C \rightarrow B$$

Verbal Arguments

An argument in English (an attorney's trial summary, an advertisement, or a political speech) that consists of simple statements can be tested for validity by a two-step process:

1. Symbolize the argument using propositional wffs.
2. Prove that the argument is valid by constructing a proof sequence for it using the derivation rules for propositional logic.

The first step, translating the argument from verbal to symbolic form, is often the most difficult. Look for keys in the verbal representation of the argument. “If... then” and “either... or” indicate implication and disjunction, respectively. A period (or sometimes a semicolon) signifies the end of a hypothesis. The separate hypotheses are joined by conjunctions. “Therefore” is a big key word; it indicates the end of the hypotheses and announces that the conclusion is about to be stated.

EXAMPLE 18

Consider the argument, “If interest rates drop, the housing market will improve. Either the federal discount rate will drop or the housing market will not improve. Interest rates will drop. Therefore the federal discount rate will drop.” Using

- I Interest rates drop.
 H The housing market will improve.
 F The federal discount rate will drop.

the argument is

$$(I \rightarrow H) \wedge (F \vee H') \wedge I \rightarrow F$$

A proof sequence to establish validity is

1. $I \rightarrow H$ hyp
2. $F \vee H'$ hyp
3. I hyp
4. $H' \vee F$ 2, comm
5. $H \rightarrow F$ 4, imp
6. $I \rightarrow F$ 1, 5, hs
7. F 3, 6, mp

EXAMPLE 19

Is the following argument valid? “My client is left-handed, but if the diary is not missing, then my client is not left-handed; therefore, the diary is missing.” There are only two simple statements involved here, so we symbolize them as follows:

- L My client is left-handed.
 D The diary is missing.

The argument is then

$$L \wedge (D' \rightarrow L') \rightarrow D$$

The validity of the argument is established by the following proof sequence.

1. L hyp
2. $D' \rightarrow L'$ hyp
3. $(D')' \vee L'$ 2, imp
4. $D \vee L'$ 3, dn
5. $L' \vee D$ 4, comm
6. $L \rightarrow D$ 5, imp
7. D 1, 6, mp

The argument says that *if* the hypotheses are true, then the conclusion will be true. The validity of the argument is a function only of its logical form and has nothing to do with the actual truth of any of its components. We still have no idea about whether the diary is really missing. Furthermore, the argument “Skooses are pink, but if Gingoos does not like perskees, then skooses are not pink; therefore Gingoos does like perskees,” which has the same logical form, is also valid, even though it does not make sense. ●

PRACTICE 14 Use propositional logic to prove that the following argument is valid. Use statement letters S , R , and B .

If security is a problem, then regulation will be increased. If security is not a problem, then business on the Web will grow. Therefore if regulation is not increased, then business on the Web will grow. ■

Formal logic is not necessary to prove the validity of propositional arguments. A valid argument is represented by a tautology, and truth tables provide a mechanical test for whether a wff is a tautology. So, what was the point of all of this? In the next section we will see that propositional wffs are not sufficient to represent everything we would like to say, and we will devise new wffs called *predicate wffs*. There is no mechanical test for the predicate wff analogue of tautology, and in the absence of such a test, we will have to rely on formal logic to justify arguments. We have developed formal logic for propositional arguments as a sort of dry run for the predicate case.

In addition, the sort of reasoning we have used in propositional logic carries over into everyday life. It is the foundation for logical thinking in computer science, mathematics, the courtroom, the marketplace, and the laboratory. Although we have approached logic as a mechanical system of applying rules, enough practice should ingrain this way of thinking so that you no longer need to consult tables of rules, but can draw logical conclusions and recognize invalid arguments on your own.

SECTION 1.2 REVIEW

TECHNIQUES

- W Apply derivation rules for propositional logic.
- W Use propositional logic to prove the validity of a verbal argument.

MAIN IDEAS

- A valid argument can be represented by a wff of the form $P_1 \wedge P_2 \wedge P_3 \wedge \dots \wedge P_n \rightarrow Q$ that is a tautology.
- A proof sequence in a formal logic system is a sequence of wffs that are either hypotheses or derived from earlier wffs in the sequence by the derivation rules of the system.
- The propositional logic system is complete and correct; valid arguments and only valid arguments are provable.

EXERCISES 1.2

For Exercises 1–4, what inference rule is illustrated by the argument given?

1. If Martina is the author, then the book is fiction. But the book is nonfiction. Therefore Martina is not the author.
2. If the business declares bankruptcy, then all assets must be confiscated. The business declared bankruptcy. It follows that all assets must be confiscated.

3. The dog has a shiny coat and loves to bark. Consequently, the dog loves to bark.
4. If Paul is a good swimmer, then he is a good runner. If Paul is a good runner, then he is a good biker. Therefore if Paul is a good swimmer, then he is a good biker.

For Exercises 5–8, decide what conclusion, if any, can be reached from the given hypotheses and justify your answer.

5. If the car was involved in the hit-and-run, then the paint would be chipped. But the paint is not chipped.
6. Either the weather will turn bad or we will leave on time. If the weather turns bad, then the flight will be canceled.
7. If the bill was sent today, then you will be paid tomorrow. You will be paid tomorrow.
8. The grass needs mowing and the trees need trimming. If the grass needs mowing, then we need to rake the leaves.
9. Justify each step in the proof sequence of

$$A \wedge (B \rightarrow C) \rightarrow (B \rightarrow (A \wedge C))$$

- | | |
|----------------------|-----------------|
| 1. A | 4. C |
| 2. $B \rightarrow C$ | 5. $A \wedge C$ |
| 3. B | |

10. Justify each step in the proof sequence of

$$B \wedge [(B \wedge C) \rightarrow A'] \wedge (B \rightarrow C) \rightarrow A'$$

- | | |
|----------------------------------|-----------------|
| 1. B | 4. C |
| 2. $(B \wedge C) \rightarrow A'$ | 5. $B \wedge C$ |
| 3. $B \rightarrow C$ | 6. A' |

11. Justify each step in the proof sequence of

$$[A \rightarrow (B \vee C)] \wedge B' \wedge C' \rightarrow A'$$

- | | |
|-------------------------------|-------------------|
| 1. $A \rightarrow (B \vee C)$ | 4. $B' \wedge C'$ |
| 2. B' | 5. $(B \vee C)'$ |
| 3. C' | 6. A' |

12. Justify each step in the proof sequence of

$$A' \wedge B \wedge [B \rightarrow (A \vee C)] \rightarrow C$$

- | | |
|-------------------------------|-----------------------|
| 1. A' | 5. $(A')' \vee C$ |
| 2. B | 6. $A' \rightarrow C$ |
| 3. $B \rightarrow (A \vee C)$ | 7. C |
| 4. $A \vee C$ | |

In Exercises 13–24, use propositional logic to prove that the argument is valid.

13. $(A \vee B')' \wedge (B \rightarrow C) \rightarrow (A' \wedge C)$
14. $A' \wedge (B \rightarrow A) \rightarrow B'$
15. $(A \rightarrow B) \wedge [A \rightarrow (B \rightarrow C)] \rightarrow (A \rightarrow C)$
16. $[(C \rightarrow D) \rightarrow C] \rightarrow [(C \rightarrow D) \rightarrow D]$
17. $A' \wedge (A \vee B) \rightarrow B$

18. $[A \rightarrow (B \rightarrow C)] \wedge (A \vee D') \wedge B \rightarrow (D \rightarrow C)$
 19. $(A' \rightarrow B') \wedge B \wedge (A \rightarrow C) \rightarrow C$
 20. $(A \rightarrow B) \wedge [B \rightarrow (C \rightarrow D)] \wedge [A \rightarrow (B \rightarrow C)] \rightarrow (A \rightarrow D)$
 21. $[A \rightarrow (B \rightarrow C)] \rightarrow [B \rightarrow (A \rightarrow C)]$
 22. $(A \wedge B) \rightarrow (A \rightarrow B)'$
 23. $(A \rightarrow C) \wedge (C \rightarrow B') \wedge B \rightarrow A'$
 24. $[A \rightarrow (B \vee C)] \wedge C' \rightarrow (A \rightarrow B)$

Use propositional logic to prove the validity of the arguments in Exercises 25–33. These will become additional derivation rules for propositional logic, summarized in Table 1.14.

25. $(P \vee Q) \wedge P' \rightarrow Q$
 26. $(P \rightarrow Q) \rightarrow (Q' \rightarrow P')$
 27. $(Q' \rightarrow P') \rightarrow (P \rightarrow Q)$
 28. $P \rightarrow P \wedge P$
 29. $P \vee P \rightarrow P$ (*Hint:* Instead of assuming the hypothesis, begin with a version of Exercise 28; also make use of Exercise 27.)
 30. $[(P \wedge Q) \rightarrow R] \rightarrow [P \rightarrow (Q \rightarrow R)]$
 31. $P \wedge P' \rightarrow Q$
 32. $P \wedge (Q \vee R) \rightarrow (P \wedge Q) \vee (P \wedge R)$ (*Hint:* First rewrite the conclusion.)
 33. $P \vee (Q \wedge R) \rightarrow (P \vee Q) \wedge (P \vee R)$ (*Hint:* Prove both $P \vee (Q \wedge R) \rightarrow (P \vee Q)$ and $P \vee (Q \wedge R) \rightarrow (P \vee R)$; for each proof, first rewrite the conclusion.)

TABLE 1.14		
More Inference Rules		
From	Can Derive	Name/Abbreviation for Rule
$P \rightarrow Q, Q \rightarrow R$	$P \rightarrow R$ [Example 16]	Hypothetical syllogism—hs
$P \vee Q, P'$	Q [Exercise 25]	Disjunctive syllogism—ds
$P \rightarrow Q$	$Q' \rightarrow P'$ [Exercise 26]	Contraposition—cont
$Q' \rightarrow P'$	$P \rightarrow Q$ [Exercise 27]	Contraposition—cont
P	$P \wedge P$ [Exercise 28]	Self-reference—self
$P \vee P$	P [Exercise 29]	Self-reference—self
$(P \wedge Q) \rightarrow R$	$P \rightarrow (Q \rightarrow R)$ [Exercise 30]	Exportation—exp
P, P'	Q [Exercise 31]	Inconsistency—inc
$P \wedge (Q \vee R)$	$(P \wedge Q) \vee (P \wedge R)$ [Exercise 32]	Distributive—dist
$P \vee (Q \wedge R)$	$(P \vee Q) \wedge (P \vee R)$ [Exercise 33]	Distributive—dist

For Exercises 34–42, use propositional logic to prove the arguments valid; you may use any of the rules in Table 1.14 or any previously proved exercise.

34. $A' \rightarrow (A \rightarrow B)$

35. $(P \rightarrow Q) \wedge (P' \rightarrow Q) \rightarrow Q$
 36. $(A' \rightarrow B') \wedge (A \rightarrow C) \rightarrow (B \rightarrow C)$
 37. $(A' \rightarrow B) \wedge (B \rightarrow C) \wedge (C \rightarrow D) \rightarrow (A' \rightarrow D)$
 38. $(A \vee B) \wedge (A \rightarrow C) \wedge (B \rightarrow C) \rightarrow C$
 39. $(Y \rightarrow Z') \wedge (X' \rightarrow Y) \wedge [Y \rightarrow (X \rightarrow W)] \wedge (Y \rightarrow Z) \rightarrow (Y \rightarrow W)$
 40. $(A \wedge B) \wedge (B \rightarrow A') \rightarrow (C \wedge B')$
 41. $(A \wedge B)' \wedge (C' \wedge A)' \wedge (C \wedge B')' \rightarrow A'$
 42. $(P \vee (Q \wedge R)) \wedge (R' \vee S) \wedge (S \rightarrow T') \rightarrow (T \rightarrow P)$

In Exercises 43–54, write the argument using propositional wffs (use the statement letters shown). Then, using propositional logic, including the rules in Table 1.14, prove that the argument is valid.

43. If the program is efficient, it executes quickly. Either the program is efficient, or it has a bug. However, the program does not execute quickly. Therefore, it has a bug. E, Q, B
44. If Jane is more popular, then she will be elected. If Jane is more popular, then Craig will resign. Therefore, if Jane is more popular, she will be elected and Craig will resign. J, E, C
45. If chicken is on the menu, then don't order fish, but you should have either fish or salad. So, if chicken is on the menu, have salad. C, F, S
46. The crop is good, but there is not enough water. If there is a lot of rain or not a lot of sun, then there is enough water. Therefore, the crop is good and there is a lot of sun. C, W, R, S
47. If the ad is successful, then the sales volume will go up. Either the ad is successful or the store will close. The sales volume will not go up. Therefore, the store will close. A, S, C
48. If DeWayne is not tall then Jayden is not DeWayne's brother. If DeWayne is tall then Trevor is DeWayne's brother. Therefore, if Jayden is DeWayne's brother, then Trevor is DeWayne's brother. D, J, T
49. Russia was a superior power, and either France was not strong or Napoleon made an error. Napoleon did not make an error, but if the army did not fail, then France was strong. Hence, the army failed and Russia was a superior power. R, F, N, A
50. It is not the case that if electric rates go up, then usage will go down, nor is it true that either new power plants will be built or bills will not be late. Therefore, usage will not go down and bills will be late. R, U, P, B
51. If Jose took the jewelry or Mrs. Krasov lied, then a crime was committed. Mr. Krasov was not in town. If a crime was committed, then Mr. Krasov was in town. Therefore, Jose did not take the jewelry. J, L, C, T
52. If the birds are flying south and the leaves are turning, then it must be fall. Fall brings cold weather. The leaves are turning but the weather is not cold. Therefore, the birds are not flying south. B, L, F, C
53. If a Democrat is elected then taxes will go up. Either a Democrat will be elected or the bill will pass. Therefore, if taxes do not go up, then the bill will pass. D, T, B
54. Either Emily was not home or if Pat did not leave the tomatoes, then Sophie was ill. Also, if Emily was not home, then Olivia left the peppers. But it is not true that either Sophie was ill or Olivia left the peppers. Therefore, Pat left the tomatoes and Olivia did not leave the peppers. E, P, S, O
55. a. Use a truth table to verify that $A \rightarrow (B \rightarrow C) \leftrightarrow (A \wedge B) \rightarrow C$ is a tautology.
 b. Prove that $A \rightarrow (B \rightarrow C) \leftrightarrow (A \wedge B) \rightarrow C$ by using a series of equivalences.
 c. Explain how this equivalence justifies the deduction method that says: to prove $P_1 \wedge P_2 \wedge \dots \wedge P_n \rightarrow (R \rightarrow S)$, deduce S from P_1, P_2, \dots, P_n , and R .
56. The argument of the defense attorney at the beginning of this chapter was

If my client is guilty, then the knife was in the drawer. Either the knife was not in the drawer or Jason Pritchard saw the knife. If the knife was not there on October 10, it follows that Jason Pritchard

didn't see the knife. Furthermore, if the knife was there on October 10, then the knife was in the drawer and also the hammer was in the barn. But we all know that the hammer was not in the barn. Therefore, ladies and gentlemen of the jury, my client is innocent.

Use propositional logic to prove that this is a valid argument.

SECTION 1.3 QUANTIFIERS, PREDICATES, AND VALIDITY

Quantifiers and Predicates

Propositional wffs have rather limited expressive power. For example, we would consider the sentence “For every x , $x > 0$ ” to be a true statement about the positive integers, yet it cannot be adequately symbolized using only statement letters, parentheses, and logical connectives. It contains two new features, a *quantifier* and a *predicate*. Quantifiers are phrases such as “for every” or “for each” or “for some” that tell in some sense *how many* objects have a certain property. The **universal quantifier** is symbolized by an upside down A, \forall , and is read “for all,” “for every,” “for each,” or “for any.” Thus the example sentence can be symbolized by

$$(\forall x)(x > 0)$$

A quantifier and its named variable are always placed in parentheses. The second set of parentheses shows that the quantifier acts on the enclosed expression, which in this case is “ $x > 0$.”

The phrase “ $x > 0$ ” describes a property of the variable x , that of being positive. A property is also called a **predicate**; the notation $P(x)$ is used to represent some unspecified predicate or property that x may have. Thus, our original sentence is an example of the more general form

$$(\forall x)P(x)$$

The truth value of the expression $(\forall x)(x > 0)$ depends on the domain of objects in which we are “interpreting” this expression, that is, the collection of objects from which x may be chosen. This collection of objects is called the *domain of interpretation*. We have already agreed that if the domain of interpretation consists of the positive integers, the expression has the truth value true because every possible value for x has the required property of being greater than zero. If the domain of interpretation consists of all the integers, the expression has the truth value false, because not every x has the required property. We impose the condition that the domain of interpretation contain at least one object so that we are not talking about a trivial case.

An interpretation of the expression $(\forall x)P(x)$ would consist of not only the collection of objects from which x could take its value but also the particular property that $P(x)$ represents in this domain. Thus an interpretation for $(\forall x)P(x)$ could be the following: The domain consists of all the books in your local library, and $P(x)$ is the property that x has a red cover. In this interpretation, $(\forall x)P(x)$ says that every book in your local library has a red cover. The truth value of this expression, in this interpretation, is undoubtedly false.

PRACTICE 15 What is the truth value of the expression $(\forall x)P(x)$ in each of the following interpretations?

- $P(x)$ is the property that x is yellow, and the domain of interpretation is the collection of all daffodils.
- $P(x)$ is the property that x is yellow, and the domain of interpretation is the collection of all flowers.
- $P(x)$ is the property that x is a plant, and the domain of interpretation is the collection of all flowers.
- $P(x)$ is the property that x is either positive or negative, and the domain of interpretation consists of the integers. ■

The **existential quantifier** is symbolized by a backward E, \exists , and is read “there exists one,” “for at least one,” or “for some.” Thus the expression

$$(\exists x)(x > 0)$$

is read “there exists an x such that x is greater than zero.”

Again, the truth value of this expression depends on the interpretation. If the domain of interpretation contains a positive number, the expression has the value true; otherwise, it has the value false. The truth value of $(\exists x)P(x)$, if the domain consists of all the books in your local library and $P(x)$ is the property that x has a red cover, is true if there is at least one book in the library with a red cover.

REMINDER

all, every, each, any—use \forall
 some, one, at least one—
 use \exists

PRACTICE 16

- Construct an interpretation (i.e., give the domain and the meaning of $P(x)$) in which $(\forall x)P(x)$ has the value true.
- Construct an interpretation in which $(\forall x)P(x)$ has the value false.
- Can you find one interpretation in which both $(\forall x)P(x)$ is true and $(\exists x)P(x)$ is false?
- Can you find one interpretation in which both $(\forall x)P(x)$ is false and $(\exists x)P(x)$ is true? ■

The predicates we have seen so far, involving properties of a single variable, are **unary predicates**. Predicates can be **binary**, involving properties of two variables, **ternary**, involving properties of three variables, or, more generally, **n -ary**, involving properties of n variables.

EXAMPLE 20

The expression $(\forall x)(\exists y)Q(x, y)$ is read “for every x there exists a y such that $Q(x, y)$.” Note that there are two quantifiers for the two variables of the binary property. In the interpretation where the domain consists of the integers and $Q(x, y)$ is the property that $x < y$, this just says that for any integer, there is a larger integer. The truth value of the expression is true. In the same interpretation, the expression $(\exists y)(\forall x)Q(x, y)$ says that there is a single integer y that is larger than any integer x . The truth value here is false. ●

Example 20 illustrates that the order in which the quantifiers appear is important.

In expressions such as $(\forall x)P(x)$ or $(\exists x)P(x)$, x is a *dummy variable*; that is, the truth values of the expressions remain the same in a given interpretation if they are written, say, as $(\forall y)P(y)$ or $(\exists z)P(z)$, respectively. Similarly, the truth value of $(\forall x)(\exists y)Q(x, y)$ is the same as that of $(\forall z)(\exists w)Q(z, w)$ in any interpretation. However, $(\forall x)(\exists x)Q(x, x)$ says something quite different. In the interpretation of Example 20, for instance, $(\forall x)(\exists x)Q(x, x)$ says that for every integer x , there is an integer x such that $x < x$. This statement is false, even though $(\forall x)(\exists y)Q(x, y)$ was true in this interpretation. We cannot collapse separate variables together into one without changing the nature of the expression we obtain.

Constants are also allowed in expressions. A constant symbol ($a, b, c, 0, 1, 2$, etc.) is interpreted as some specific object in the domain. This specification is part of the interpretation. For example, the expression $(\forall x)Q(x, a)$ is false in the interpretation where the domain consists of the integers, $Q(x, y)$ is the property $x < y$, and a is assigned the value 7; it is not the case that every integer is less than 7.

Now we can sum up what is required in an interpretation.

DEFINITION INTERPRETATION

An **interpretation** for an expression involving predicates consists of the following:

- A collection of objects, called the **domain** of the interpretation, which must include at least one object
- An assignment of a property of the objects in the domain to each predicate in the expression
- An assignment of a particular object in the domain to each constant symbol in the expression

Expressions can be built by combining predicates with quantifiers, grouping symbols (parentheses or brackets), and the logical connectives of Section 1.1. As before, an expression must obey rules of syntax to be considered a well-formed formula. Well-formed formulas containing predicates and quantifiers are called **predicate wffs** to distinguish them from propositional wffs, which contain only statement letters and logical connectives.

The expression $P(x)(\forall x) \wedge \exists y$ is not a well-formed formula. Examples of predicate wffs are

$$P(x) \vee Q(y) \tag{1}$$

$$(\forall x)[P(x) \rightarrow Q(x)] \tag{2}$$

$$(\forall x)((\exists y)[P(x, y) \wedge Q(x, y)] \rightarrow R(x)) \tag{3}$$

and

$$(\exists x)S(x) \vee (\forall y)T(y) \tag{4}$$

“Grouping symbols” such as parentheses and brackets identify the **scope** of a quantifier, the section of the wff to which the quantifier applies. (This is analogous to the scope of an identifier in a computer program as the section of the program in which that identifier has meaning.) There are no quantifiers in wff (1). In (2), the scope of the quantifier $(\forall x)$ is $P(x) \rightarrow Q(x)$. In (3), the scope of $(\exists y)$ is $P(x, y) \wedge Q(x, y)$, while the scope of $(\forall x)$ is the entire expression in parentheses

following it. In (4), the scope of $(\exists x)$ is $S(x)$ and the scope of $(\forall y)$ is $T(y)$; parentheses or brackets can be eliminated when the scope is clear.

If a variable occurs somewhere in a wff where it is not part of a quantifier and is not within the scope of a quantifier involving that variable, it is called a **free variable**. For example, y is a free variable in

$$(\forall x)[Q(x, y) \rightarrow (\exists y)R(x, y)]$$

because of the first occurrence of y , which is neither the variable of a quantifier nor within the scope of a quantifier using y . A wff with free variables may not have a truth value at all in a given interpretation. For example, in the interpretation where the domain is all of the integers, the predicate $P(x)$ means “ $x > 0$ ”, and 5 means (of course) the integer 5, the wff

$$P(y) \wedge P(5)$$

has no truth value because we don’t know which element of the domain y refers to. Some elements of the domain are positive and others are not. The wff

$$P(y) \vee P(5)$$

is true in this interpretation even though we don’t know what y refers to because $P(5)$ is true. In both of these wffs y is a free variable.

EXAMPLE 21

In the wff

$$(\forall x)(\exists y)[S(x, y) \wedge L(y, a)]$$

the scope of $(\exists y)$ is all of $S(x, y) \wedge L(y, a)$. The scope of $(\forall x)$ is $(\exists y)[S(x, y) \wedge L(y, a)]$. Consider the interpretation where the domain consists of all the cities in the United States, $S(x, y)$ is the property “ x and y are in the same state,” $L(y, z)$ is the property “ y ’s name begins with the same letter as z ’s name,” and a is assigned the value Albuquerque. So the interpretation of the entire wff is that for any city x there is a city y in the same state that begins with the letter A . The wff is true in this interpretation. (At least it is true if every state has a city beginning with the letter A .)

PRACTICE 17

What is the truth value of the wff

$$(\exists x)(A(x) \wedge (\forall y)[B(x, y) \rightarrow C(y)])$$

in the interpretation where the domain consists of all integers, $A(x)$ is “ $x > 0$,” $B(x, y)$ is “ $x > y$,” and $C(y)$ is “ $y \leq 0$ ”? Construct another interpretation with the same domain in which the statement has the opposite truth value.

Translation

Many English language statements can be expressed as predicate wffs. For example, “Every parrot is ugly,” is really saying, “For any thing, if it is a parrot,

then it is ugly.” Letting $P(x)$ denote “ x is a parrot” and $U(x)$ denote “ x is ugly,” the statement can be symbolized as

$$(\forall x)[P(x) \rightarrow U(x)]$$

Other English language variations that take the same symbolic form are, “All parrots are ugly,” and, “Each parrot is ugly.” Notice that the quantifier is the universal quantifier and the logical connective is implication; \forall and \rightarrow almost always belong together. The wff $(\forall x)[P(x) \wedge U(x)]$ is an incorrect translation because it says that everything in the domain—understood here to be the whole world—is an ugly parrot. This says something much stronger than the original English statement.

Similarly, “There is an ugly parrot,” is really saying, “There exists something that is both a parrot and ugly.” In symbolic form,

$$(\exists x)[P(x) \wedge U(x)]$$

REMINDER

Think

$\forall \rightarrow$

and

$\exists \wedge$

Variations are, “Some parrots are ugly,” and, “There are ugly parrots.” Here the quantifier is the existential quantifier and the logical connective is conjunction; \exists and \wedge almost always belong together. The wff $(\exists x)[P(x) \rightarrow U(x)]$ is an incorrect translation. This wff is true as long as there is anything, call it x , in the domain (the whole world) that is not a parrot, because then $P(x)$ is false and the implication is true. Indeed, this wff is true if there are no parrots in the world at all!

To translate an English statement into a wff, it may help to first write an intermediate English language statement and then symbolize that statement. We did this with the parrot examples.

The word “only” seems particularly troublesome in translations because its placement in a sentence can completely change the meaning. For example, the English statements

1. John loves only Mary.
2. Only John loves Mary
3. John only loves Mary.

say three entirely different things. Using the predicate symbols $J(x)$ for “ x is John,” $M(x)$ for “ x is Mary,” and $L(x, y)$ for “ x loves y ,” they can be rewritten as

1. If John loves any thing, then that thing is Mary.

or

1. For any thing, if it is John then, if it loves anything, that thing is Mary.

$$(\forall x)(J(x) \rightarrow (\forall y)(L(x, y) \rightarrow M(y)))$$

2. If any thing loves Mary, then that thing is John.

or

2. For any thing, if it is Mary then, if anything loves it, that thing is John.

$$(\forall x)(M(x) \rightarrow (\forall y)(L(y, x) \rightarrow J(y)))$$

3. If John does any thing to Mary, then that thing is love.

or

3. For any thing, if it is John then, for any other thing, if that thing is Mary, then John loves it.

$$(\forall x)(J(x) \rightarrow (\forall y)(M(y) \rightarrow L(x, y)))$$

In each case, the consequent of the implication is the word following “only” in the original English statement.

EXAMPLE 22

Given the predicate symbols

$D(x)$ is “ x is a dog”

$R(x)$ is “ x is a rabbit”

$C(x, y)$ is “ x chases y ”

Table 1.15 shows examples of an English statement, an intermediate English statement, and a wff translation. Note that in wff 2, the connective associated with \exists is \wedge and the connective associated with \forall is \rightarrow . In wff 3, the first version shows two implications associated with the two \forall quantifiers. The second version is equivalent because of the tautology $[A \wedge B \rightarrow C] \leftrightarrow [A \rightarrow (B \rightarrow C)]$. This version may appear to violate the rule that universal quantifiers should be used with implication, not conjunction, but this tautology provides another way to write two implications. The second version also shows more clearly that “dogs,” the word following “only,” is the conclusion.

TABLE 1.15

English Statement	Intermediate Statement	Wff
1. All dogs chase all rabbits.	For any thing, if it is a dog, then for any other thing, if that thing is a rabbit, then the dog chases it.	$(\forall x)[D(x) \rightarrow (\forall y)(R(y) \rightarrow C(x, y))]$
2. Some dogs chase all rabbits.	There is some thing that is a dog and, for any other thing, if that thing is a rabbit, then the dog chases it.	$(\exists x)[D(x) \wedge (\forall y)(R(y) \rightarrow C(x, y))]$
3. Only dogs chase rabbits.	For any thing, if it is a rabbit then, if anything chases it, that thing is a dog. For any two things, if one is a rabbit and the other chases it, then the other is a dog.	$(\forall y)[R(y) \rightarrow (\forall x)(C(x, y) \rightarrow D(x))]$ $(\forall y)(\forall x)[R(y) \wedge C(x, y) \rightarrow D(x)]$

Often more than one wff exists that is a correct representation of an English statement, as seen with statement (3) in Table 1.15. Also wff (2) is equivalent to

$$(\exists x)[D(x) \wedge (\forall y)([R(y)]' \vee C(x, y))]$$

because of the implication equivalence rule that says $(R \rightarrow C) \leftrightarrow (R' \vee C)$, even though here R and C are predicates instead of just statement letters.

In addition, it is legitimate to “slide” a quantifier over a predicate that does not involve the variable of that quantifier. Because $D(x)$ does not involve y , we could slide the universal quantifier in wff (2) to the front (but not past the existential quantifier), giving the equivalent wff

$$(\exists x)(\forall y)[D(x) \wedge (R(y) \rightarrow C(x, y))] \quad (\text{a})$$

In wff (a), we still have grouping symbols around $R(y) \rightarrow C(x, y)$. Without the grouping symbols, this wff becomes

$$(\exists x)(\forall y)[D(x) \wedge R(y) \rightarrow C(x, y)] \quad (\text{b})$$

which, according to the order of precedence of connectives, is equivalent to

$$(\exists x)(\forall y)[(D(x) \wedge R(y)) \rightarrow C(x, y)]$$

A quick truth table exercise shows that $D \wedge (R \rightarrow C)$ is *not* equivalent to $(D \wedge R) \rightarrow C$, so wff (b) is not equivalent to wff (a) and thus does *not* represent statement (2) in the table. ●

Translation from an English statement into a predicate wff is a little harder than translation into a propositional wff, partly because of the added expressiveness of the verbal form and partly because there can be multiple correct predicate wffs. Here is a summary of translation tips:

- Look for the key words that signify the type of quantifier:
for all, for every, for any, for each: use a universal quantifier
for some, there exists: use an existential quantifier.
- English sometimes uses “understood” universal quantifiers. For example, “Dogs chase rabbits,” is understood to mean, “All dogs chase all rabbits.”
- If you use a universal quantifier, the connective that goes with it is almost always implication.
- If you use an existential quantifier, the connective that goes with it is almost always conjunction.
- Whatever comes after the word “only” is the conclusion of an implication; that is, it comes after “then” in an “if–then” statement.
- You are most apt to arrive at a correct translation if you follow the order of the English words.

EXAMPLE 23

Let’s do a couple of examples in great detail. The first is

All giraffes are tall.

The property of being a giraffe and the property of being tall are unary predicates. We’ll use $G(x)$ for “ x is a giraffe” and $T(x)$ for “ x is tall”. Following the sentence structure, we first see “All,” which tells us that there’s a universal quantifier, so the wff begins with

$$(\forall x)(\dots)$$

All what? All giraffes, so

$$(\forall x)(G(x) \dots)$$

Because of the universal quantifier, we expect to use the implication connective, so now we have

$$(\forall x)(G(x) \rightarrow \dots)$$

Thinking of the implication as an “if–then,” we have “if a giraffe, then” Then what? Then it's tall. The final wff is

$$(\forall x)(G(x) \rightarrow T(x))$$

The second example is

Only giraffes are taller than elephants.

The property of being a giraffe and the property of being an elephant are unary predicates, and we'll use $G(x)$ and $E(x)$ to represent them. But “taller than” is a property that compares two things, so it's a binary predicate; $T(x, y)$ will mean “ x is taller than y ”. There are no obvious quantifier key words, so we understand that we are talking about all giraffes and all elephants (universal quantifiers). The word “giraffes” follows the word “only,” so the property of being a giraffe is going to be the conclusion of an implication and the overall form will be “if xxx , then a giraffe.” Indeed, if something is taller than an elephant, then it's a giraffe. Putting in the universal quantifiers, “if any thing is taller than any elephant, then that thing is a giraffe,” or (even more tortured English), “for any thing, if it is an elephant, then for any other thing, if it's taller than the elephant, then it's a giraffe.” Now we can pretty much translate directly into a wff. “For any thing, if it is an elephant, then” becomes

$$(\forall x)(E(x) \rightarrow \dots)$$

and, “for any other thing, if it's taller than the elephant, then,” adds a second implication to the wff:

$$(\forall x)(E(x) \rightarrow (\forall y)(T(y, x) \rightarrow \dots))$$

Notice that we introduced y here, a second variable, because we've already given x the elephant property. Also, we've written $T(y, x)$, not $T(x, y)$, because we want this new thing to be taller than the elephant, and our definition of the taller predicate was that the first variable was taller than the second. We are ready for the final conclusion—this new thing is a giraffe.

$$(\forall x)(E(x) \rightarrow (\forall y)(T(y, x) \rightarrow G(y)))$$

As in Table 1.15(3), a tautology allows us to also write this wff as

$$(\forall x)(\forall y)(E(x) \wedge T(y, x) \rightarrow G(y))$$

“For any two things, if one is an elephant and the other is taller than the elephant, then the other thing is a giraffe.”

With some practice, you won't have to go quite this slowly! ●

PRACTICE 18 Using the predicate symbols $S(x)$ for “ x is a student,” $I(x)$ for “ x is intelligent,” and $M(x)$ for “ x likes music,” write wffs that express the following statements. (The domain is the collection of all people.)

- All students are intelligent.
- Some intelligent students like music.
- Everyone who likes music is a stupid student.
- Only intelligent students like music.

PRACTICE 19 Using the predicate symbols $F(x)$ for “ x is a fruit,” $V(x)$ for “ x is a vegetable,” and $S(x,y)$ for “ x is sweeter than y ,” write wffs that express the following statements. (The domain is the whole world.)

- Some vegetable is sweeter than all fruits.
- Every fruit is sweeter than all vegetables.
- Every fruit is sweeter than some vegetable.
- Only fruits are sweeter than vegetables.

Negating statements with quantifiers, as in negating compound statements, requires care. The negation of the statement, “Everything is beautiful,” is, “It is false that everything is beautiful,” or, “Something is nonbeautiful.” Symbolically,

$$[(\forall x)A(x)]' \text{ is equivalent to } (\exists x)[A(x)]'$$

Note that, “Everything is nonbeautiful,” or $(\forall x)[A(x)]'$, says something *stronger* than the negation of the original statement.

The negation of, “Something is beautiful,” is, “Nothing is beautiful,” or, “Everything fails to be beautiful.” Symbolically,

$$[(\exists x)A(x)]' \text{ is equivalent to } (\forall x)[A(x)]'$$

In English, the statement, “Everything is not beautiful,” would often be misinterpreted as, “Not everything is beautiful,” or, “There is something nonbeautiful.” However, this misinterpretation, symbolized by $(\exists x)[A(x)]'$, is *not as strong* as the negation of the original statement.

PRACTICE 20 Which of the following statements expresses the negation of, “Everybody loves somebody sometime”?

- Everybody hates somebody sometime.
- Somebody loves everybody all the time.
- Everybody hates everybody all the time.
- Somebody hates everybody all the time.

Validity

The truth value of a propositional wff depends on the truth values assigned to the statement letters. The truth value of a predicate wff depends on the interpretation. Choosing an interpretation for a predicate wff is thus analogous to choosing truth values in a propositional wff. However, there are an infinite number of possible interpretations for a predicate wff and only 2^n possible rows in the truth table for a propositional wff with n statement letters.

A tautology is a propositional wff that is true for all rows of the truth table. The analogue to tautology for predicate wffs is *validity*—a predicate wff is **valid** if it is true in all possible interpretations. The validity of a wff must be derived from the form of the wff itself, since validity is independent of any particular interpretation; a valid wff is “intrinsically true.”

An algorithm exists to decide whether a propositional wff is a tautology—construct the truth table and examine all possible truth assignments. How can we go about deciding validity for predicate wffs? We clearly cannot look at all possible interpretations, because there are an infinite number of them. As it turns out, no algorithm to decide validity for any wff exists. (This does not mean simply that no algorithm has yet been found—it means that it has been proved that there is no such algorithm.) We must simply use reasoning to determine whether the form of a particular wff makes the wff true in all interpretations. Of course, if we can find a single interpretation in which the wff has the truth value false or has no truth value at all, then the wff is not valid.

Table 1.16 compares propositional and predicate wffs.

TABLE 1.16		
	Propositional Wffs	Predicate Wffs
Truth values	True or false, depending on truth value assignments to statement letters	True, false, or perhaps (if the wff has a free variable) neither, depending on interpretation
“Intrinsic truth”	Tautology—true for all truth value assignments	Valid wff—true for all interpretations
Methodology	Algorithm (truth table) to determine whether wff is a tautology	No algorithm to determine whether wff is valid

Now let’s try our hand at determining validity for specific wffs.

EXAMPLE 24

a. The wff

$$(\forall x)P(x) \rightarrow (\exists x)P(x)$$

is valid. In any interpretation, if every element of the domain has a certain property, then there exists an element of the domain that has that property. (Remember that the domain of any interpretation must have at least one object in it.) Therefore,

whenever the antecedent is true, so is the consequent, and the implication is therefore true.

b. The wff

$$(\forall x)P(x) \rightarrow P(a)$$

is valid because in any interpretation, a is a particular member of the domain and therefore has the property that is shared by all members of the domain.

c. The wff

$$(\forall x)[P(x) \wedge Q(x)] \leftrightarrow (\forall x)P(x) \wedge (\forall x)Q(x)$$

is valid. If both P and Q are true for all the elements of the domain, then P is true for all elements and Q is true for all elements, and vice versa.

d. The wff

$$P(x) \rightarrow [Q(x) \rightarrow P(x)]$$

is valid, even though it contains a free variable. To see this, consider any interpretation, and let x be any member of the domain. Then x either does or does not have property P . If x does not have property P , then $P(x)$ is false; because $P(x)$ is the antecedent of the main implication, this implication is true. If x does have property P , then $P(x)$ is true; regardless of the truth value of $Q(x)$, the implication $Q(x) \rightarrow P(x)$ is true, and so the main implication is also true.

e. The wff

$$(\exists x)P(x) \rightarrow (\forall x)P(x)$$

is not valid. For example, in the interpretation where the domain consists of the integers and $P(x)$ means that x is even, it is true that there exists an integer that is even, but it is false that every integer is even. The antecedent of the implication is true and the consequent is false, so the value of the implication is false. ●

We do not necessarily have to go to a mathematical context to construct an interpretation in which a wff is false, but it is frequently easier to do so because the relationships among objects are relatively clear.

PRACTICE 21 Is the wff valid or invalid? Explain.

$$(\forall x)[P(x) \vee Q(x)] \rightarrow (\forall x)P(x) \vee (\forall x)Q(x)$$

SECTION 1.3 REVIEW

TECHNIQUES

- W Determine the truth value of a predicate wff in a given interpretation.
- W Translate English language statements into predicate wffs, and vice versa.
 - Recognize a valid wff and explain why it is valid.
- W Recognize a nonvalid wff and construct an interpretation in which it is false or has no truth value.

MAIN IDEAS

- The truth value of predicate wffs depends on the interpretation considered.
- Valid predicate wffs are “intrinsically true”—true in all interpretations.

EXERCISES 1.3

1. What is the truth value of each of the following wffs in the interpretation where the domain consists of the integers, $O(x)$ is “ x is odd,” $L(x)$ is “ $x < 10$,” and $G(x)$ is “ $x > 9$ ”?
 - a. $(\exists x)O(x)$
 - b. $(\forall x)[L(x) \rightarrow O(x)]$
 - c. $(\exists x)[L(x) \wedge G(x)]$
 - d. $(\forall x)[L(x) \vee G(x)]$
2. What is the truth value of each of the following wffs in the interpretation where the domain consists of the integers, $A(x)$ is “ $x < 5$ ” and $B(x)$ is “ $x < 7$ ”?
 - a. $(\exists x)A(x)$
 - b. $(\exists x)[A(x) \wedge B(x)]$
 - c. $(\forall x)[A(x) \rightarrow B(x)]$
 - d. $(\forall x)[B(x) \rightarrow A(x)]$
3. What is the truth value of each of the following wffs in the interpretation where the domain consists of the integers?
 - a. $(\forall x)(\exists y)(x + y = x)$
 - b. $(\exists y)(\forall x)(x + y = x)$
 - c. $(\forall x)(\exists y)(x + y = 0)$
 - d. $(\exists y)(\forall x)(x + y = 0)$
 - e. $(\forall x)(\forall y)(x < y \vee y < x)$
 - f. $(\forall x)[x < 0 \rightarrow (\exists y)(y > 0 \wedge x + y = 0)]$
 - g. $(\exists x)(\exists y)(x^2 = y)$
 - h. $(\forall x)(x^2 > 0)$
4. What is the truth value of each of the following wffs in the interpretation where the domain consists of the real numbers?
 - a. $(\forall x)(\exists y)(x = y^2)$
 - b. $(\forall x)(\forall y)(x = y^2)$
 - c. $(\exists x)(\forall y)(x = y^2)$
 - d. $(\exists x)(\exists y)(x = y^2)$
5. Give the truth value of each of the following wffs in the interpretation where the domain consists of the states of the United States, $Q(x, y)$ is “ x is north of y ,” $P(x)$ is “ x starts with the letter M ,” and a is “Massachusetts.”
 - a. $(\forall x)P(x)$
 - b. $(\forall x)(\forall y)(\forall z)[Q(x, y) \wedge Q(y, z) \rightarrow Q(x, z)]$
 - c. $(\exists y)(\exists x)Q(y, x)$
 - d. $(\forall x)(\exists y)[P(y) \wedge Q(x, y)]$
 - e. $(\exists y)Q(a, y)$
 - f. $(\exists x)[P(x) \wedge Q(x, a)]$
6. Give the truth value of each of the following wffs in the interpretation where the domain consists of people, $M(x, y)$ is “ x is the mother of y ,” $F(x)$ is “ x is female,” $M(x)$ is “ x is male.”
 - a. $(\forall x)(\exists y)(M(y, x))$
 - b. $(\exists x)(\forall y)(M(x, y))$
 - c. $(\forall x)(\forall y)(M(x, y) \rightarrow M(y))$
 - d. $(\exists x)(\exists y)(M(x, y) \wedge M(y))$
 - e. $(\exists x)(\forall y)(M(x, y) \rightarrow F(y))$

7. For each wff, find an interpretation in which it is true and one in which it is false.
- $(\forall x)([A(x) \vee B(x)] \wedge [A(x) \wedge B(x)])'$
 - $(\forall x)(\forall y)[P(x, y) \rightarrow P(y, x)]$
 - $(\forall x)[P(x) \rightarrow (\exists y)Q(x, y)]$
8. For each wff, find an interpretation in which it is true and one in which it is false.
- $(\exists x)[A(x) \wedge (\forall y)B(x, y)]$
 - $[(\forall x)A(x) \rightarrow (\forall x)B(x)] \rightarrow (\forall x)[A(x) \rightarrow B(x)]$
 - $(\exists x)[P(x) \vee Q(x)] \wedge (\forall x)[P(x) \rightarrow Q(x)]$
9. Identify the scope of each of the quantifiers in the following wffs and indicate any free variables.
- $(\forall x)[P(x) \rightarrow Q(y)]$
 - $(\exists x)[A(x) \wedge (\forall y)B(y)]$
 - $(\exists x)[(\forall y)P(x, y) \wedge Q(x, y)]$
 - $(\exists x)(\exists y)[A(x, y) \wedge B(y, z) \rightarrow A(a, z)]$
10. Explain why each of the following expressions is written incorrectly.
- $(\exists)(Q(x) \wedge P(x))$
 - $(\forall y)(Q(y) P(y))$
 - $(\forall x)(\forall y)Q(x) \rightarrow P(y)$
11. Which of the following sentences are equivalent to the statement
All circles are round.
- If it's round, it's a circle.
 - Roundness is a necessary property of circles.
 - Something that isn't round can't be a circle.
 - Some round things are circles.
12. Which of the following sentences are equivalent to the statement
Cats are smarter than dogs.
- Some cats are smarter than some dogs.
 - There is a cat that is smarter than all dogs.
 - All cats are smarter than all dogs.
 - Only cats are smarter than dogs.
 - All cats are smarter than any dog.
13. Using the predicate symbols shown and appropriate quantifiers, write each English language statement as a predicate wff. (The domain is the whole world.)
- $D(x)$: x is a day
 $S(x)$: x is sunny
 $R(x)$: x is rainy
 M : Monday
 T : Tuesday
- All days are sunny.
 - Some days are not rainy.
 - Every day that is sunny is not rainy.
 - Some days are sunny and rainy.

- e. No day is both sunny and rainy.
f. It is always a sunny day only if it is a rainy day.
g. No day is sunny.
h. Monday was sunny; therefore, every day will be sunny.
i. It rained both Monday and Tuesday.
j. If some day is rainy, then every day will be sunny.
14. Using the predicate symbols shown and appropriate quantifiers, write each English language statement as a predicate wff. (The domain is the whole world.)
- $B(x)$: x is a ball
 $R(x)$: x is round
 $S(x)$: x is a soccer ball
- a. All balls are round.
b. Not all balls are soccer balls.
c. All soccer balls are round.
d. Some balls are not round.
e. Some balls are round but soccer balls are not.
f. Every round ball is a soccer ball.
g. Only soccer balls are round balls.
h. If soccer balls are round, then all balls are round.
15. Using the predicate symbols shown and appropriate quantifiers, write each English language statement as a predicate wff. (The domain is the whole world.)
- $M(x)$: x is a man
 $W(x)$: x is a woman
 $T(x)$: x is tall
- a. All men are tall.
b. Some women are tall.
c. All men are tall but no woman is tall.
d. Only women are tall
e. No man is tall.
f. If every man is tall, then every woman is tall.
g. Some woman is not tall.
h. If no man is tall, then some woman is not tall.
16. Using the predicate symbols shown and appropriate quantifiers, write each English language statement as a predicate wff. (The domain is the whole world.)
- $A(x)$: x is an animal
 $B(x)$: x is a bear
 $H(x)$: x is hungry
 $W(x)$: x is a wolf
- a. Bears are animals.
b. No wolf is a bear.

- c. Only bears are hungry.
d. If all wolves are hungry, so are bears.
e. Some animals are hungry bears.
f. Bears are hungry but some wolves are not.
g. If wolves and bears are hungry, so are all animals.
h. Some wolves are hungry but not every animal is hungry.
17. Using the predicate symbols shown and appropriate quantifiers, write each English language statement as a predicate wff. (The domain is the whole world.)
- $P(x)$: x is a person
 $T(x)$: x is a time
 $F(x, y)$: x is fooled at y
- a. You can fool some of the people all of the time.
b. You can fool all of the people some of the time.
c. You can't fool all of the people all of the time.
18. Using the predicate symbols shown and appropriate quantifiers, write each English language statement as a predicate wff. (The domain is the whole world.)
- $L(x)$: x is a lion
 $R(x)$: x roars
 $P(x)$: x is a predator
 $Z(x)$: x is a zebra
 $E(x, y)$: x eats y
- a. All lions are predators.
b. Some lions roar.
c. Only lions roar.
d. Some lions eat all zebras.
e. All lions eat all zebras.
19. Using the predicate symbols shown and appropriate quantifiers, write each English language statement as a predicate wff. (The domain is the whole world.)
- $G(x)$: x is a game
 $M(x)$: x is a movie
 $F(x, y)$: x is more fun than y
- a. Any movie is more fun than any game.
b. No game is more fun than every movie.
c. Only games are more fun than movies.
d. All games are more fun than some movie.
20. Using the predicate symbols shown and appropriate quantifiers, write each English language statement as a predicate wff. (The domain is the whole world.)
- $C(x)$: x is a child
 $T(x)$: x is a toy

$V(x)$: x is a vegetable

$W(x, y)$: x wants y

- a. Every child wants toys.
- b. Only children want toys.
- c. Some child wants only toys.
- d. No child wants vegetables.

21. Using the predicate symbols shown and appropriate quantifiers, write each English language statement as a predicate wff. (The domain is the whole world.)

$J(x)$: x is a judge

$L(x)$: x is a lawyer

$W(x)$: x is a woman

$C(x)$: x is a chemist

$A(x, y)$: x admires y

- a. There are some women lawyers who are chemists.
- b. No woman is both a lawyer and a chemist.
- c. Some lawyers admire only judges.
- d. All judges admire only judges.
- e. Only judges admire judges.
- f. All women lawyers admire some judge.
- g. Some women admire no lawyer.

22. Using the predicate symbols shown and appropriate quantifiers, write each English language statement as a predicate wff. (The domain is the whole world.)

$C(x)$: x is a Corvette

$F(x)$: x is a Ferrari

$P(x)$: x is a Porsche

$S(x, y)$: x is slower than y

- a. Nothing is both a Corvette and a Ferrari.
- b. Some Porsches are slower than only Ferraris.
- c. Only Corvettes are slower than Porsches.
- d. All Ferraris are slower than some Corvettes.
- e. Some Porsches are slower than no Corvette.
- f. If there is a Corvette that is slower than a Ferrari, then all Corvettes are slower than all Ferraris.

23. Using the predicate symbols shown and appropriate quantifiers, write each English language statement as a predicate wff. (The domain is the whole world.)

$B(x)$: x is a bee

$F(x)$: x is a flower

$L(x, y)$: x loves y

- | | |
|--------------------------------|----------------------------------|
| a. All bees love all flowers. | d. Every bee hates only flowers. |
| b. Some bees love all flowers. | e. Only bees love flowers. |
| c. All bees love some flowers. | f. Every bee loves only flowers. |

- g. No bee loves only flowers. j. Every bee hates some flowers.
 h. Some bees love some flowers. k. Every bee hates all flowers.
 i. Some bees love only flowers. l. No bee hates all flowers.

24. Using the predicate symbols shown and appropriate quantifiers, write each English language statement as a predicate wff. (The domain is the whole world.)

$S(x)$: x is a spy novel

$L(x)$: x is long

$M(x)$: x is a mystery

$B(x, y)$: x is better than y

- a. All spy novels are long.
 b. Not every mystery is a spy novel.
 c. Only mysteries are long.
 d. Some spy novels are mysteries.
 e. Spy novels are better than mysteries.
 f. Some mysteries are better than all spy novels.
 g. Only spy novels are better than mysteries.

25. Give English language translations of the following wffs if

$L(x, y)$: x loves y

$H(x)$: x is handsome

$M(x)$: x is a man

$P(x)$: x is pretty

$W(x)$: x is a woman

j : John

k : Kathy

$L(x, y)$: x loves y

- a. $H(j) \wedge L(k, j)$
 b. $(\forall x)[M(x) \rightarrow H(x)]$
 c. $(\forall x)(W(x) \rightarrow (\forall y)[L(x, y) \rightarrow M(y) \wedge H(y)])$
 d. $(\exists x)[M(x) \wedge H(x) \wedge L(x, k)]$
 e. $(\exists x)(W(x) \wedge P(x) \wedge (\forall y)[L(x, y) \rightarrow H(y) \wedge M(y)])$
 f. $(\forall x)[W(x) \wedge P(x) \rightarrow L(j, x)]$

26. Give English language translations of the following wffs if

$M(x)$: x is a man

$W(x)$: x is a woman

i : Ivan

p : Peter

$W(x, y)$: x works for y

- a. $(\exists x)(W(x) \wedge (\forall y)(M(y) \rightarrow [W(x, y)]'))$
 b. $(\forall x)[M(x) \rightarrow (\exists y)(W(y) \wedge W(x, y))]$
 c. $(\forall x)[M(x) \rightarrow (\forall y)(W(x, y) \rightarrow W(y))]$

d. $(\forall x)(\forall y)(M(x) \wedge W(y, x) \rightarrow W(y))$

e. $W(i, p) \wedge (\forall x)[W(p, x) \rightarrow (W(x))']$

f. $(\forall x)[W(x, i) \rightarrow (W(x))']$

27. Three forms of negation are given for each statement. Which is correct?

- a. Some people like mathematics.
 1. Some people dislike mathematics.
 2. Everybody dislikes mathematics.
 3. Everybody likes mathematics.
- b. Everyone loves ice cream.
 1. No one loves ice cream.
 2. Everyone dislikes ice cream.
 3. Someone doesn't love ice cream.
- c. All people are tall and thin.
 1. Someone is short and fat.
 2. No one is tall and thin.
 3. Someone is short or fat.
- d. Some pictures are old or faded.
 1. Every picture is neither old nor faded.
 2. Some pictures are not old or faded.
 3. All pictures are not old or not faded.

28. Three forms of negation are given for each statement. Which is correct?

- a. Nobody is perfect.
 1. Everyone is imperfect.
 2. Everyone is perfect.
 3. Someone is perfect.
- b. All swimmers are tall.
 1. Some swimmer is not tall.
 2. There are no tall swimmers.
 3. Every swimmer is short.
- c. Every planet is cold and lifeless.
 1. No planet is cold and lifeless.
 2. Some planet is not cold and not lifeless.
 3. Some planet is not cold or not lifeless.
- d. No bears are hungry.
 1. Only bears are hungry.
 2. All bears are hungry.
 3. There is a hungry bear.

29. Write the negation of each of the following statements.

- a. Some Web sites feature audio.
- b. Every Web site has both audio and video.
- c. Every Web site has either audio or video.

- d. Some Web sites have neither audio nor video.
 e. Every Web site either has text or else has both audio and video.
30. Write the negation of each of the following statements.
 a. Only students eat pizza.
 b. Every student eats pizza
 c. Some students eat only pizza.
31. Write the negation of each of the following statements.
 a. Some farmer grows only corn.
 b. All farmers grow corn.
 c. Corn is grown only by farmers.
32. Write the negation of each of the following statements
 a. Some child fears all clowns.
 b. Some children fear only clowns.
 c. No clown fears any child.
33. Explain why each wff is valid.
 a. $(\forall x)(\forall y)A(x, y) \leftrightarrow (\forall y)(\forall x)A(x, y)$
 b. $(\exists x)(\exists y)A(x, y) \leftrightarrow (\exists y)(\exists x)A(x, y)$
 c. $(\exists x)(\forall y)P(x, y) \rightarrow (\forall y)(\exists x)P(x, y)$
 d. $A(a) \rightarrow (\exists x)A(x)$
 e. $(\forall x)[A(x) \rightarrow B(x)] \rightarrow [(\forall x)A(x) \rightarrow (\forall x)B(x)]$
34. Give interpretations to prove that each of the following wffs is not valid:
 a. $(\exists x)A(x) \wedge (\exists x)B(x) \rightarrow (\exists x)[A(x) \wedge B(x)]$
 b. $(\forall x)(\exists y)P(x, y) \rightarrow (\exists x)(\forall y)P(x, y)$
 c. $(\forall x)[P(x) \rightarrow Q(x)] \rightarrow [(\exists x)P(x) \rightarrow (\forall x)Q(x)]$
 d. $(\forall x)[A(x)]' \leftrightarrow [(\forall x)A(x)]'$
35. Decide whether each of the following wffs is valid or invalid. Justify your answer.
 a. $(\exists x)A(x) \leftrightarrow ((\forall x)[A(x)]')'$
 b. $(\forall x)P(x) \vee (\exists x)Q(x) \rightarrow (\forall x)[P(x) \vee Q(x)]$
36. Decide whether each of the following wffs is valid or invalid. Justify your answer.
 a. $(\forall x)A(x) \rightarrow ((\exists x)[A(x)]')'$
 b. $(\forall x)[P(x) \rightarrow Q(x)] \wedge (\exists x)[P(x) \vee Q(x)] \rightarrow (\exists x)[P(x) \wedge Q(x)]$
 c. $(\forall x)[P(x) \vee Q(x)] \rightarrow (\forall x)P(x) \vee (\exists y)Q(y)$
37. From Example 24c, we know that $(\forall x)[P(x) \wedge Q(x)] \leftrightarrow (\forall x)P(x) \wedge (\forall x)Q(x)$ is valid. From Practice 21, we know that $(\forall x)[P(x) \vee Q(x)] \leftrightarrow (\forall x)P(x) \vee (\forall x)Q(x)$ is not valid. From Exercise 34a, we know that $(\exists x)[P(x) \wedge Q(x)] \leftrightarrow (\exists x)P(x) \wedge (\exists x)Q(x)$ is not valid. Explain why $(\exists x)[P(x) \vee Q(x)] \leftrightarrow (\exists x)P(x) \vee (\exists x)Q(x)$ is valid.
38. A predicate wff is in *prenex normal form* if all the quantifiers appear at the front of the wff. Write each of the following expressions as an equivalent wff in prenex normal form.
 a. $(\forall x)P(x) \wedge (\forall y)Q(y)$
 b. $(\forall x)(P(x) \rightarrow (\forall y)[Q(y) \rightarrow W(x, y)])$
 c. $(\exists x)P(x) \wedge (\exists x)Q(x)$

SECTION 1.4 PREDICATE LOGIC

We can imagine arguments of the form

$$P_1 \wedge P_2 \wedge P_3 \wedge \dots \wedge P_n \rightarrow Q$$

where the wffs are built from predicates and quantifiers as well as logical connectives and grouping symbols. For a **valid argument**, Q must follow logically from P_1, \dots, P_n based solely on the internal structure of the argument, not on the truth or falsity of Q in any particular interpretation. In other words, the wff

$$P_1 \wedge P_2 \wedge P_3 \wedge \dots \wedge P_n \rightarrow Q$$

must be valid—true in all possible interpretations. No equivalent of the truth table exists to easily prove validity, so we turn to a formal logic system called **predicate logic**. We again use a system of derivation rules to build a proof sequence leading from the hypotheses to the conclusion. The rules should once more be truth-preserving, so that if in some interpretation, all the hypotheses are true, then the conclusion will also be true in that interpretation. The system will then be correct (*only* valid arguments will be provable). We also want the system to be complete (*every* valid argument should be provable), yet at the same time the rule set should be minimal.

Derivation Rules for Predicate Logic⁴

The equivalence rules and inference rules of propositional logic are still part of predicate logic. An argument of the form

$$P \wedge (P \rightarrow Q) \rightarrow Q$$

is still valid by modus ponens, even if the wffs involved are predicate wffs.

EXAMPLE 25

Use predicate logic to prove the validity of the argument

$$(\forall x)R(x) \wedge [(\forall x)R(x) \rightarrow (\forall x)S(x)] \rightarrow (\forall x)S(x)$$

A proof sequence is

1. $(\forall x)R(x)$ hyp
2. $(\forall x)R(x) \rightarrow (\forall x)S(x)$ hyp
3. $(\forall x)S(x)$ 1, 2, mp

However, there are many arguments with predicate wffs that are not tautologies but are still valid because of their structure and the meaning of the universal and existential quantifiers (see Example 24). The overall approach to proving these arguments is to strip off the quantifiers, manipulate the unquantified wffs,

⁴A complete list of derivation rules for propositional and predicate logic is given in Appendix A.

and then put the quantifiers back in. The new rules of inference provide mechanisms to strip off and insert quantifiers. Hence there are four new rules—one each to strip off the universal and existential quantifier, respectively, and one each to insert the universal and existential quantifier, respectively. The four rules are given in Table 1.17; their details will be explained shortly. In Table 1.17, the notation $P(x)$ does *not* imply that P is a unary predicate with x as its only variable; it simply means that x is one of the variables in the predicate P . Thus $P(x)$ might actually be something like $(\exists y)(\forall z)Q(x, y, z)$.

TABLE 1.17

Inference Rules			
From	Can Derive	Name/Abbreviation for Rule	Restrictions on Use
$(\forall x)P(x)$	$P(t)$, where t is a variable or constant symbol	Universal instantiation—ui	If t is a variable, it must not fall within the scope of a quantifier for t .
$(\exists x)P(x)$	$P(a)$ where a is a constant symbol not previously used in proof sequence	Existential instantiation—ei	Must be the first rule used that introduces a .
$P(x)$	$(\forall x)P(x)$	Universal generalization—ug	$P(x)$ has not been deduced from any hypotheses in which x is a free variable nor has $P(x)$ been deduced by ei from any wff in which x is a free variable.
$P(x)$ or $P(a)$ where a is a constant symbol	$(\exists x)P(x)$	Existential generalization—eg	To go from $P(a)$ to $(\exists x)P(x)$, x must not appear in $P(a)$.

Now let's examine these rules more closely, particularly the necessity for their restrictions.

Universal Instantiation

The **universal instantiation** rule says that from $(\forall x)P(x)$ we can derive $P(x)$, $P(y)$, $P(z)$, $P(a)$, and so on, thus stripping off a universal quantifier. The justification is that if P is true for every element of the domain, we can name such an element by an arbitrary variable name like x , y , or z , or we can specify a particular constant in the domain, and P is still true for all of these things.

EXAMPLE 26

Universal instantiation can be used to prove one of the classical “syllogisms” of the Greek philosopher and scientist Aristotle, who lived from 384 to 322 B.C.E. and who first developed a system of formal logic.

The argument has the form, “All humans are mortal. Socrates is human. Therefore Socrates is mortal.” Using the notation

$H(x)$ is “ x is human.”

s is a constant symbol (Socrates)

$M(x)$ is “ x is mortal.”

the argument is

$$(\forall x)[H(x) \rightarrow M(x)] \wedge H(s) \rightarrow M(s)$$

and a proof sequence is

$(\forall x)(H(x) \rightarrow M(x))$	hyp
$H(s)$	hyp
$H(s) \rightarrow M(s)$	1, ui
$M(s)$	2, 3, mp

In step 3, a constant symbol has been substituted for x throughout the scope of the universal quantifier, as allowed by universal instantiation. ●

Without the restriction on universal instantiation, a hypothesis of the form $(\forall x)(\exists y)P(x, y)$ could lead to the wff $(\exists y)P(y, y)$; here y has been substituted for x within the scope of a quantifier on y . This would be invalid. For example, in the domain of the integers, if $P(x, y)$ means “ $y > x$,” then $(\forall x)(\exists y)P(x, y)$ is true (for every integer there is a bigger integer) but $(\exists y)P(y, y)$ is false (no integer has the property that it is bigger than itself).

PRACTICE 22 Prove the argument

$$(\forall x)[P(x) \rightarrow R(x)] \wedge [R(y)]' \rightarrow [P(y)]'$$

Existential Instantiation

The **existential instantiation** rule allows us to strip off an existential quantifier. It says that from $(\exists x)P(x)$ we can derive $P(a)$ or $P(b)$ or $P(c)$ provided that these are new constant symbols. The justification is that if P is true for some element of the domain, we can give that element a specific name, but we cannot assume anything else about it.

EXAMPLE 27

The following expressions would be legitimate steps in a proof sequence:

- $(\forall x)[P(x) \rightarrow Q(x)]$ hyp
- $(\exists y)P(y)$ hyp
- $P(a)$ 2, ei
- $P(a) \rightarrow Q(a)$ 1, ui
- $Q(a)$ 3, 4, mp

In step 3, the specific element with property P was given the name a . In step 4, ui was then used to say that an implication that is universally true in the domain is certainly true for this a . Steps 3 and 4 *cannot be reversed*. If ui is first used on hypothesis 1 to name a constant a , there is then no reason to assume that this particular a is the one that is guaranteed by hypothesis 2 to have property P . ●

REMINDER

Use existential instantiation early in the proof sequence.

The effect of the restriction on existential instantiation is that you should look at all your hypotheses and, if you plan to use ei on any of them, do it first.

Universal Generalization

Universal generalization allows a universal quantifier to be inserted. This must be done pretty carefully, however. If we know that $P(x)$ is true and that the x is absolutely arbitrary, i.e., that x could be any element of the domain, then we can conclude $(\forall x)P(x)$. But if x is supposed to represent some specific element of the domain that has property P , then we can't generalize that every element of the domain has property P .

EXAMPLE 28

Use predicate logic to prove

$$(\forall x)[P(x) \rightarrow Q(x)] \wedge (\forall x)P(x) \rightarrow (\forall x)Q(x)$$

Here is a proof sequence.

- | | |
|---|---|
| 1. $(\forall x)[P(x) \rightarrow Q(x)]$ | hyp |
| 2. $(\forall x)P(x)$ | hyp |
| 3. $P(x) \rightarrow Q(x)$ | 1, ui |
| 4. $P(x)$ | 2, ui Note that there is no restriction on ui about reusing a name. |
| 5. $Q(x)$ | 3, 4, mp |
| 6. $(\forall x)Q(x)$ | 5, ug |

The use of universal generalization at step 6 is legitimate because x was not a free variable in any hypothesis nor was ei used anywhere in the proof. The variable x in steps 3 and 4 is just an arbitrary name, representative of any element in the domain. ●

There are two restrictions on universal generalization. Without the first restriction, the sequence

1. $P(x)$ hyp
2. $(\forall x)P(x)$ 1, incorrect ug; x was free in the hypothesis.

would be a proof of the wff $P(x) \rightarrow (\forall x)P(x)$, but this is not a valid wff. Element x of the domain may have property P , but that does not mean that every element of the domain has property P . In the hypothesis, x is naming some fixed if unspecified element of the domain. For instance, in the interpretation where the domain consists of automobiles and $P(x)$ means “ x is yellow,” some particular car may be yellow but it is certainly not true that all cars are yellow.

Without the second restriction, the sequence

1. $(\forall x)(\exists y)Q(x, y)$ hyp
2. $(\exists y)Q(x, y)$ 1, ui
3. $Q(x, a)$ 2, ei
4. $(\forall x)Q(x, a)$ 3, incorrect ug; $Q(x, a)$ was deduced by ei from the wff in step 2, in which x is free.

would be a proof of the wff $(\forall x)(\exists y)Q(x, y) \rightarrow (\forall x)Q(x, a)$. This is also not a valid wff. For instance, in the interpretation where the domain consists of the integers and $Q(x, y)$ means that $x + y = 0$, then it is the case that for every integer x there is an integer y (the negative of x) such that $x + y = 0$. However, if a is a particular fixed element in the domain, then it will not be true that adding that same integer a to every x will always produce zero.

PRACTICE 23 Prove the argument

$$(\forall x)[P(x) \wedge Q(x)] \rightarrow (\forall x)[Q(x) \wedge P(x)].$$

Existential Generalization

The last rule allows insertion of an existential quantifier. From $P(x)$ or $P(a)$ we can derive $(\exists x)P(x)$; something has been named as having property P , so we can say that there exists something that has property P .

EXAMPLE 29

Prove the argument $(\forall x)P(x) \rightarrow (\exists x)P(x)$.
Here is a proof sequence.

1. $(\forall x)P(x)$ hyp
2. $P(x)$ 1, ui
3. $(\exists x)P(x)$ 2, eg

Without the restriction on existential generalization, from $P(a, y)$ one could derive $(\exists y)P(y, y)$; here the quantified variable y , which replaced the constant symbol a , already appeared in the wff to which existential generalization was applied. But the argument $P(a, y) \rightarrow (\exists y)P(y, y)$ is not valid. In the domain of integers, if $P(x, y)$ means “ $y > x$ ” and a stands for 0, then if $y > 0$, this does not mean that there is an integer y that is greater than itself.

More Work with Rules

As is the case with propositional logic rules, predicate logic rules can be applied only when the exact pattern of the rule is matched (and, of course, when no restrictions on use of the rule are violated). In particular, notice that the instantiation rules strip off a quantifier from the front of an entire wff that is in the scope of that quantifier. Both of the following would be illegal uses of existential instantiation:

1. $(\exists x)P(x) \vee (\exists x)Q(x)$ hyp
2. $P(a) \vee Q(a)$ 1, incorrect ei. The scope of the first existential quantifier in step 1 does not extend to the whole rest of the wff.

1. $(\forall x)(\exists y)Q(x, y)$ hyp
2. $(\forall x)Q(x, a)$ 1, incorrect ei. The existential quantifier in step 1 is not at the front.

Similarly, the rules to insert a quantifier put the quantifier in the front of a wff that is then entirely within its scope.

The new derivation rules all have restrictions, but in practice, you are most likely to violate the ei restriction. Pay special attention to this restriction and be sure that when you substitute a constant using ei, it's not a previously used constant.

Even though we have added only four new derivation rules, the rule set is complete and correct. We can prove every valid argument and only valid arguments using these rules. Application of the rules, as in the case of propositional logic, is somewhat mechanical because there are only a limited number of options at each step. Again, the general plan of attack is usually as follows:

- Strip off the quantifiers.
- Work with the separate wffs.
- Insert quantifiers as necessary.

EXAMPLE 30

Using predicate logic, prove the argument

$$(\forall x)[P(x) \wedge Q(x)] \rightarrow (\forall x)P(x) \wedge (\forall x)Q(x)$$

In Example 24(c) we noted that this wff is valid, so if all valid arguments are provable, we should be able to find a proof sequence. As usual, the hypothesis gives us a starting point.

1. $(\forall x)[P(x) \wedge Q(x)]$ hyp

Stripping off the universal quantifier that appears in step 1 will yield access to $P(x) \wedge Q(x)$, which can then be separated. The universal quantifier can then be inserted separately on each of those two wffs using universal generalization. The conclusion $(\forall x)P(x) \wedge (\forall x)Q(x)$ will follow. A proof sequence is

1. $(\forall x)[P(x) \wedge Q(x)]$ hyp
2. $P(x) \wedge Q(x)$ 1, ui
3. $P(x)$ 2, sim
4. $Q(x)$ 2, sim
5. $(\forall x)P(x)$ 3, ug
6. $(\forall x)Q(x)$ 4, ug
7. $(\forall x)P(x) \wedge (\forall x)Q(x)$ 5, 6, con

Neither restriction on universal generalization has been violated because x is not free in the hypothesis and existential instantiation has not been used. ●

PRACTICE 24

Using predicate logic, prove the following argument. (*Hint*: The deduction method still applies.)

$$(\forall y)[P(x) \rightarrow Q(x, y)] \rightarrow [P(x) \rightarrow (\forall y)Q(x, y)]$$

As an extension to the deduction method, we can insert a “temporary” hypothesis into a proof. If some wff T is introduced into a proof sequence as a temporary hypothesis, and eventually a wff W is deduced from T and other hypotheses, then the wff $T \rightarrow W$ has been deduced from the other hypotheses and can be inserted in the proof sequence.

EXAMPLE 31

The argument

$$[P(x) \rightarrow (\forall y)Q(x, y)] \rightarrow (\forall y)[P(x) \rightarrow Q(x, y)]$$

is valid. In the following proof sequence, $P(x)$ is introduced at step 2 as a temporary hypothesis, which allows us to deduce $Q(x, y)$ at step 4. The indented steps show that these wffs depend on the temporary hypothesis. At step 5, the temporary hypothesis is “discharged,” as the dependency of $Q(x, y)$ on the temporary hypothesis is explicitly acknowledged as an implication. Of course, the entire wff at step 5, $P(x) \rightarrow Q(x, y)$, still depends on the hypothesis of step 1. At step 6, neither restriction on universal generalization is violated because y is not a free variable in step 1 (the only hypothesis at this point) and existential instantiation is not used in the proof.

- | | |
|--|----------------------|
| 1. $P(x) \rightarrow (\forall y)Q(x, y)$ | hyp |
| 2. $P(x)$ | temporary hyp |
| 3. $(\forall y)Q(x, y)$ | 1, 2, mp |
| 4. $Q(x, y)$ | 3, ui |
| 5. $P(x) \rightarrow Q(x, y)$ | temp. hyp discharged |
| 6. $(\forall y)[P(x) \rightarrow Q(x, y)]$ | 5, ug |

Notice how the temporary hypothesis gives us enough ammunition to make something happen. Without this technique, it would be difficult to know what to do after step 1. ●

The technique of introducing a temporary hypothesis is seldom needed. Again, think of this as an extension to the deduction method. If the desired conclusion is of the form $P \rightarrow Q$, the deduction method says we can assume P as a hypothesis and deduce Q as the conclusion. If the desired conclusion is of the form $(\forall x)(P(x) \rightarrow Q(x))$ or $(\exists x)(P(x) \rightarrow Q(x))$, then the deduction method does not apply, but $P(x)$ can be used as a temporary hypothesis.

Practice 24 and Example 31 show that the wff

$$(\forall y)[P(x) \rightarrow Q(x, y)] \leftrightarrow [P(x) \rightarrow (\forall y)Q(x, y)]$$

is valid. It says that the universal quantifier can “slide over” subwffs that do not contain the quantified variable; in this case, $(\forall y)$ is passed over $P(x)$. A similar result holds for the existential quantifier. We noted this feature in Example 22, and here is the formal justification. This is one reason why there may be two or more equivalent ways of expressing English language sentences as predicate wffs, as in Exercises 13 through 24 of Section 1.3.

PRACTICE 25 Prove the argument

$$(\forall x)[(B(x) \vee C(x)) \rightarrow A(x)] \rightarrow (\forall x)[B(x) \rightarrow A(x)]$$

In Section 1.3 we observed that, based on our understanding of negation and the meaning of the quantifiers, $[(\exists x)A(x)]'$ is equivalent to $(\forall x)[A(x)]'$. We should be able to formally prove that

$$[(\exists x)A(x)]' \leftrightarrow (\forall x)[A(x)]'$$

is a valid wff.

EXAMPLE 32 Prove that

$$[(\exists x)A(x)]' \leftrightarrow (\forall x)[A(x)]'$$

is valid. We must prove the implication in each direction.

a. $[(\exists x)A(x)]' \leftrightarrow (\forall x)[A(x)]'$

The hypothesis alone gives us little to work with, so we introduce a (somewhat surprising) temporary hypothesis. A proof sequence is

- | | |
|---------------------------------------|--------------------------|
| 1. $[(\exists x)A(x)]'$ | hyp |
| 2. $A(x)$ | temporary hyp |
| 3. $(\exists x)A(x)$ | 2, eg |
| 4. $A(x) \rightarrow (\exists x)A(x)$ | temporary hyp discharged |
| 5. $[A(x)]'$ | 1, 4, mt |
| 6. $(\forall x)[A(x)]'$ | 5, ug |

b. $(\forall x)[A(x)]' \rightarrow [(\exists x)A(x)]'$

This proof also requires a temporary hypothesis. It is even more surprising than case (a) because we assume the exact opposite of the conclusion we are trying to reach.

- | | |
|--|--------------------------|
| 1. $(\forall x)[A(x)]'$ | hyp |
| 2. $(\exists x)A(x)$ | temporary hyp |
| 3. $A(a)$ | 2, ei |
| 4. $[A(a)]'$ | 1, ui |
| 5. $[(\forall x)[A(x)]']'$ | 3, 4, inc |
| 6. $(\exists x)A(x) \rightarrow [(\forall x)[A(x)]']'$ | temporary hyp discharged |
| 7. $[[(\forall x)[A(x)]']']'$ | 1, dn |
| 8. $[(\exists x)A(x)]'$ | 6, 7, mt |

The proofs of Example 32 are rather difficult because they require considerably more imagination than most and an unexpected use of a temporary

hypothesis. As a result, however, we do have the following equivalence, to which we've given a name:

$$[(\exists x)A(x)]' \leftrightarrow (\forall x)[A(x)]' \quad (\text{Negation—neg})$$

This equivalence might be useful in a proof sequence. As an extension of the equivalence rules, whenever $P \leftrightarrow Q$ is valid, Q can be substituted for P within an expression in a proof sequence.

EXAMPLE 33

Is the wff

$$(\forall x)[P(x) \vee Q(x)] \rightarrow (\exists x)P(x) \vee (\forall x)Q(x)$$

a valid argument? Prove or disprove.

Let's first consider whether the wff seems valid. If so, we should try to find a proof sequence for it; if not, we should try to find an interpretation in which it is not true. This wff says that if every element of the domain has either property P or property Q , then at least one element must have property P or else all elements have property Q . This seems very reasonable, so we'll try to find a proof.

First we'll use an equivalence to rewrite the conclusion in a more useful form. Changing the \vee to an implication will allow use of the deduction method. Thus we want to prove

$$(\forall x)[P(x) \vee Q(x)] \rightarrow [[(\exists x)P(x)]' \rightarrow (\forall x)Q(x)]$$

A proof sequence is

- | | |
|----------------------------------|----------|
| 1. $(\forall x)[P(x) \vee Q(x)]$ | hyp |
| 2. $[(\exists x)P(x)]'$ | hyp |
| 3. $(\forall x)[P(x)]'$ | 2, neg |
| 4. $[P(x)]'$ | 3, ui |
| 5. $P(x) \vee Q(x)$ | 1, ui |
| 6. $Q(x)$ | 4, 5, ds |
| 7. $(\forall x)Q(x)$ | 6, ug |

EXAMPLE 34

Is the wff

$$(\exists x)P(x) \wedge (\exists x)Q(x) \rightarrow (\exists x)[P(x) \wedge Q(x)]$$

a valid argument? Prove or disprove.

If something in a domain has property P and something has property Q , that does not mean that some one thing has both property P and Q . For example, in the domain of integers, if $P(x)$ means “ x is even” and $Q(x)$ means “ x is odd,” then the hypotheses are true, but the conclusion is false because there is no single integer that is both even and odd. One interpretation in which the wff is false is enough to disprove it.

It is useful, however, to see where a potential proof sequence goes wrong. We begin with the two hypotheses and then remove one of the existential quantifiers.

1. $(\exists x)P(x)$ hyp
2. $(\exists x)Q(x)$ hyp
3. $P(a)$ 1, ei

Now here's the problem. The next step would be to remove the existential quantifier from the wff at step 2, but, according to the rules for ei, we have to name the object that has property Q by some different name, not a . So we could eventually get to a wff in the proof sequence that looks like

$$P(a) \wedge Q(b)$$

but this does us no good. Existential generalization could not be used to replace both constant symbols with a single variable. At best, we could arrive at

$$(\exists y)(\exists x)[P(x) \wedge Q(y)]$$

which is not what we want. •

PRACTICE 26 Is the wff a valid argument? Prove or disprove.

$$(\exists x)R(x) \wedge [(\exists x)[R(x) \wedge S(x)]]' \rightarrow (\exists x)[S(x)]'$$
■

Verbal Arguments

To prove the validity of a verbal argument, we proceed much as before. We cast the argument in symbolic form and show that the conclusion can be deduced from the hypotheses. If the argument involves predicate wffs, then the derivation rules of predicate logic are available.

EXAMPLE 35

Show that the following argument is valid: "Every laptop has an internal disk drive. Some laptops have a DVD drive. Therefore some laptops have both an internal disk drive and a DVD drive." Using

$L(x)$ is "x is a laptop."

$I(x)$ is "x has an internal disk drive."

$D(x)$ is "x has a DVD drive."

the argument is

$$(\forall x)[L(x) \rightarrow I(x)] \wedge (\exists x)[L(x) \wedge D(x)] \rightarrow (\exists x)[L(x) \wedge I(x) \wedge D(x)]$$

Note that if we attempt to symbolize this argument in propositional logic, we get $A \wedge B \rightarrow C$, which is not a valid argument. Propositional logic is simply not expressive enough to capture the interrelationships among the parts of this argument that serve to make it valid.

A proof sequence is

- | | |
|--|-----------|
| 1. $(\forall x)[L(x) \rightarrow I(x)]$ | hyp |
| 2. $(\exists x)[L(x) \wedge D(x)]$ | hyp |
| 3. $L(a) \wedge D(a)$ | 2, ei |
| 4. $L(a) \rightarrow I(a)$ | 1, ui |
| 5. $L(a)$ | 3, sim |
| 6. $I(a)$ | 4, 5, mp |
| 7. $L(a) \wedge D(a) \wedge I(a)$ | 3, 6, con |
| 8. $L(a) \wedge I(a) \wedge D(a)$ | 7, comm |
| 9. $(\exists x)[L(x) \wedge I(x) \wedge D(x)]$ | 8, eg |

Once again, it is the form of the argument that matters, not the content. ●

PRACTICE 27

Show that the following argument is valid: “All rock music is loud music. Some rock music exists, therefore some loud music exists.” Use predicates $R(x)$ and $L(x)$. ■

Conclusion

We’ve now finished our study of formal logic. What has been accomplished? The goal of formal logic, often called *symbolic logic*, is to make arguments as meaningless as possible! The symbolic notation of propositional and predicate logic allows us to symbolize arguments. An argument cast in symbolic notation removes any possibility that we will be swayed by our opinions or our external knowledge about the topic of the argument, and we can concentrate solely on its structure to determine its logical validity. Furthermore, the derivation rules allow the proof of an argument’s validity to be produced by symbol manipulation. A proof requires no external knowledge, only a careful adherence to the forms and restrictions of the rules. In theory, then, producing a proof sequence should be almost mechanical. Again, one objective of practice with this mechanical process of applying rules is that it will ultimately transform into a habit of logical thinking in everyday life.

Nonetheless, you may still feel that it is difficult to produce a proof sequence. Practice does make the process easier, because after a while, you become familiar with the various forms an argument might take and you recognize which rules you should try to apply. At any rate, you should at least find it easy at this point to *check* whether a *proposed* proof sequence is logically correct.

Philosophers through the ages believed logical thinking to be one of the highest achievements of human existence. One additional example, however, will show how even the most careful application of logic can be frustrated.

EXAMPLE 36

The hunting grounds of a medieval king were forbidden to commoners, and anyone caught poaching the royal deer was subject to death. The hapless poacher was, however, granted a means to choose the manner of death. He (or she) was allowed to make a final statement. If the statement were judged to be true, death would be by beheading with a sword; if false, death would come by arrow shot from the bow of the best royal marksman. One day a particularly clever poacher was apprehended and allowed the usual final statement. The poacher said, “I will be shot to death by an arrow.”

The king’s court faced a conundrum. If the poacher were shot to death by an arrow, then the statement he made would prove to be true, in which case he should have been beheaded. But if he were beheaded, then the statement he made would be false, in which case he should have been shot by an arrow. Unable to decide the manner of death, the court appointed the clever poacher to the post of king’s press secretary, where he served happily for many years.

This sort of *paradox*—a riddle with no solution—has to be carefully constructed, and we will not spend any more time reflecting on the potential shortcomings of classical logic systems that it may reveal. ●

In addition to logical thinking in its pure sense, the notions of formal rules of inference have two very direct applications to computer science. An entire system of programming, and some programming languages, are based on applying rules of inference. We will see such a language in Section 1.5. Similarly, rules of inference can be applied to formally prove program correctness, leading to increased confidence that code is error-free. We’ll look at some of the inference rules for program correctness in Section 1.6.

SECTION 1.4 REVIEW**TECHNIQUES**

- W Apply derivation rules for predicate logic.
- W Use predicate logic to prove the validity of a verbal argument.

MAIN IDEA

- The predicate logic system is correct and complete; valid arguments and only valid arguments are provable.

EXERCISES 1.4

For Exercises 1–6, decide what conclusion, if any, can be reached from the given hypotheses and justify your answer.

1. All flowers are plants. Pansies are flowers.
2. All flowers are plants. Pansies are plants.
3. All flowers are red or purple. Pansies are flowers. Pansies are not purple.
4. Some flowers are purple. All purple flowers are small.
5. Some flowers are red. Some flowers are purple. Pansies are flowers.
6. Some flowers are pink and have thorns. All thorny flowers smell bad. Every flower that smells bad is a weed.

7. Justify each step in the following proof sequence of

$$(\exists x)[P(x) \rightarrow Q(x)] \rightarrow [(\forall x)P(x) \rightarrow (\exists x)Q(x)]$$

1. $(\exists x)[P(x) \rightarrow Q(x)]$
2. $P(a) \rightarrow Q(a)$
3. $(\forall x)P(x)$
4. $P(a)$
5. $Q(a)$
6. $(\exists x)Q(x)$

8. Justify each step in the following proof sequence of

$$(\exists x)P(x) \wedge (\forall x)(P(x) \rightarrow Q(x)) \rightarrow (\exists x)Q(x)$$

1. $(\exists x)P(x)$
2. $(\forall x)(P(x) \rightarrow Q(x))$
3. $P(a)$
4. $P(a) \rightarrow Q(a)$
5. $Q(a)$
6. $(\exists x)Q(x)$

9. Consider the wff

$$(\forall x)[(\exists y)P(x, y) \wedge (\exists y)Q(x, y)] \rightarrow (\forall x)(\exists y)[P(x, y) \wedge Q(x, y)]$$

- a. Find an interpretation to prove that this wff is not valid.
- b. Find the flaw in the following “proof” of this wff.

1. $(\forall x)[(\exists y)P(x, y) \wedge (\exists y)Q(x, y)]$ hyp
2. $(\forall x)[P(x, a) \wedge Q(x, a)]$ 1, ei
3. $(\forall x)(\exists y)[P(x, y) \wedge Q(x, y)]$ 2, eg

10. Consider the wff

$$(\forall y)(\exists x)Q(x, y) \rightarrow (\exists x)(\forall y)Q(x, y)$$

- a. Find an interpretation to prove that this wff is not valid.
- b. Find the flaw in the following “proof” of this wff.

1. $(\forall y)(\exists x)Q(x, y)$ hyp
2. $(\exists x)Q(x, y)$ 1, ui
3. $Q(a, y)$ 2, ei
4. $(\forall y)Q(a, y)$ 3, ug
5. $(\exists x)(\forall y)Q(x, y)$ 4, eg

In Exercises 11–16, prove that each wff is a valid argument.

11. $(\forall x)P(x) \rightarrow (\forall x)[P(x) \vee Q(x)]$
12. $(\forall x)P(x) \wedge (\exists x)Q(x) \rightarrow (\exists x)[P(x) \wedge Q(x)]$
13. $(\exists x)(\exists y)P(x, y) \rightarrow (\exists y)(\exists x)P(x, y)$
14. $(\forall x)(\forall y)Q(x, y) \rightarrow (\forall y)(\forall x)Q(x, y)$

15. $(\forall x)P(x) \wedge (\exists x)[P(x)]' \rightarrow (\exists x)Q(x)$
 16. $(\forall x)[S(x) \rightarrow (\exists y)(P(x,y) \wedge T(y))] \wedge (\exists x)(C(x) \wedge S(x)) \rightarrow (\exists x)(\exists y)(C(x) \wedge T(y) \wedge P(x,y))$

In Exercises 17–30, either prove that the wff is a valid argument or give an interpretation in which it is false.

17. $(\exists x)[A(x) \wedge B(x)] \rightarrow (\exists x)A(x) \wedge (\exists x)B(x)$
 18. $(\exists x)[R(x) \vee S(x)] \rightarrow (\exists x)R(x) \vee (\exists x)S(x)$
 19. $(\exists x)P(x) \wedge (\exists x)(\exists y)Q(x,y) \rightarrow (\exists x)(\exists y)[P(x) \wedge Q(x,y)]$
 20. $(\forall x)[P(x) \rightarrow Q(x)] \rightarrow [(\forall x)P(x) \rightarrow (\forall x)Q(x)]$
 21. $(\forall x)(P(x))' \rightarrow (\forall x)(P(x) \rightarrow Q(x))$
 22. $[(\forall x)P(x) \rightarrow (\forall x)Q(x)] \rightarrow (\forall x)[P(x) \rightarrow Q(x)]$
 23. $(\exists x)(\forall y)Q(x,y) \rightarrow (\forall y)(\exists x)Q(x,y)$
 24. $(\forall x)P(x) \vee (\exists x)Q(x) \rightarrow (\forall x)[P(x) \vee Q(x)]$
 25. $(\forall x)[A(x) \rightarrow B(x)] \rightarrow [(\exists x)A(x) \rightarrow (\exists x)B(x)]$
 26. $(\forall y)[Q(x,y) \rightarrow P(x)] \rightarrow [(\exists y)Q(x,y) \rightarrow P(x)]$
 27. $[P(x) \rightarrow (\exists y)Q(x,y)] \rightarrow (\exists y)[P(x) \rightarrow Q(x,y)]$
 28. $(\forall x)(P(x) \vee Q(x)) \wedge (\exists x)Q(x) \rightarrow (\exists x)P(x)$
 29. $(\exists x)[P(x) \wedge Q(x)] \wedge (\forall y)[Q(y) \rightarrow R(y)] \rightarrow (\exists x)[P(x) \wedge R(x)]$
 30. $(\forall x)(\forall y)[(P(x) \wedge S(x,y)) \rightarrow Q(y)] \wedge (\exists x)B(x) \wedge (\forall x)(B(x) \rightarrow P(x)) \wedge (\forall x)(\exists y)S(x,y) \rightarrow (\exists x)Q(x)$

31. The Greek philosopher Aristotle (384–322 B.C.E.) studied under Plato and tutored Alexander the Great. His studies of logic influenced philosophers for hundreds of years. His four “perfect” syllogisms are identified by the names given them by medieval scholars. For each, formulate the argument in predicate logic notation and then provide a proof.

a. “Barbara”

All M are P

All S are M

Therefore all S are P

b. “Celarent”

No M are P

All S are M

Therefore no S are P

c. “Darii”

All M are P

Some S are M

Therefore some S are P

d. “Ferio”

No M are P

Some S are M

Therefore some S are not P

Using predicate logic, prove that each argument in Exercises 32–42 is valid. Use the predicate symbols shown.

32. Some plants are flowers. All flowers smell sweet. Therefore, some plants smell sweet. $P(x), F(x), S(x)$

33. Every crocodile is bigger than every alligator. Sam is a crocodile. But there is a snake, and Sam isn't bigger than the snake. Therefore, something is not an alligator. $C(x), A(x), B(x, y), s, S(x)$
34. There is an astronomer who is not nearsighted. Everyone who wears glasses is nearsighted. Furthermore, everyone either wears glasses or wears contact lenses. Therefore, some astronomer wears contact lenses. $A(x), N(x), G(x), C(x)$
35. Every member of the board comes from industry or government. Everyone from government who has a law degree is in favor of the motion. John is not from industry, but he does have a law degree. Therefore, if John is a member of the board, he is in favor of the motion. $M(x), I(x), G(x), L(x), F(x), j$
36. There is some movie star who is richer than everyone. Anyone who is richer than anyone else pays more taxes than anyone else does. Therefore, there is a movie star who pays more taxes than anyone. $M(x), R(x, y), T(x, y)$
37. Everyone with red hair has freckles. Someone has red hair and big feet. Everybody who doesn't have green eyes doesn't have big feet. Therefore someone has green eyes and freckles. $R(x), F(x), B(x), G(x)$
38. Cats eat only animals. Something fuzzy exists. Everything that's fuzzy is a cat. And everything eats something. So animals exist. $C(x), E(x, y), A(x), F(x)$
39. Every computer science student works harder than somebody, and everyone who works harder than any other person gets less sleep than that person. Maria is a computer science student. Therefore, Maria gets less sleep than someone else. $C(x), W(x, y), S(x, y), m$
40. Every ambassador speaks only to diplomats, and some ambassador speaks to someone. Therefore, there is a diplomat. $A(x), S(x, y), D(x)$
41. Some elephants are afraid of all mice. Some mice are small. Therefore there is an elephant that is afraid of something small. $E(x), M(x), A(x, y), S(x)$
42. Every farmer owns a cow. No dentist owns a cow. Therefore no dentist is a farmer. $F(x), C(x), O(x, y), D(x)$
43. Prove that

$$[(\forall x)A(x)]' \leftrightarrow (\exists x)[A(x)]'$$

is valid. (*Hint:* Instead of a proof sequence, use Example 32 and substitute equivalent expressions.)

44. The equivalence of Exercise 43 says that if it is false that every element of the domain has property A , then some element of the domain fails to have property A , and vice versa. The element that fails to have property A is called a counterexample to the assertion that every element has property A . Thus a counterexample to the assertion

$$(\forall x)(x \text{ is odd})$$

in the domain of integers is the number 10, an even integer. (Of course, there are lots of other counterexamples to this assertion.) Find counterexamples in the domain of integers to the following assertions. (An integer $x > 1$ is prime if the only factors of x are 1 and x .)

- $(\forall x)(x \text{ is negative})$
- $(\forall x)(x \text{ is the sum of even integers})$
- $(\forall x)(x \text{ is prime} \rightarrow x \text{ is odd})$
- $(\forall x)(x \text{ prime} \rightarrow (-1)^x = -1)$
- $(\forall x)(x \text{ prime} \rightarrow 2^x - 1 \text{ is prime})$

SECTION 1.5 LOGIC PROGRAMMING

The programming languages with which you are probably familiar, such as C++ or Java, are known as **procedural languages**. Much of the content of a program written in a procedural language consists of instructions to carry out the algorithm the programmer believes will solve the problem at hand. The programmer, therefore, is telling the computer how to solve the problem in a step-by-step fashion.

Some programming languages, rather than being procedural, are **declarative languages** or **descriptive languages**. A declarative language is based on predicate logic; such a language comes equipped with its own rules of inference. A program written in a declarative language consists only of statements—actually predicate wffs—that are declared as hypotheses. Execution of a declarative program allows the user to pose queries, asking for information about possible conclusions that can be derived from the hypotheses. After obtaining the user’s query, the language turns on its “inference engine” and applies its rules of inference to the hypotheses to see which conclusions fit the user’s query. The program, remember, contains only the hypotheses, not any explicit instructions as to what steps to perform in what order. The inference engine of the language acts behind the scenes, so to speak, to construct a proof sequence. It is the mechanical nature of applying inference rules that makes this “automated theorem proving” possible.

Prolog

The programming language Prolog, which stands for PROgramming in LOGic, is a declarative programming language. The set of declarations that constitutes a Prolog program is also known as a **Prolog database**. Items in a Prolog database take on one of two forms, known in Prolog as *facts* and *rules*. (Prolog rules, however, are just another kind of fact and should not be confused with a rule of inference.)

Prolog facts allow predicates to be defined by stating which items in some domain of interpretation satisfy the predicates. As an example, suppose we wish to create a Prolog program that describes food chains in a given ecological region. We might begin with a binary predicate *eat*. We then describe the predicate by giving the pairs of elements in the domain that make *eat* true. Thus we might have the facts

```
eat(bear, fish)
eat(bear, fox)
eat(deer, grass)
```

in our database. (The exact details of Prolog statements vary from one Prolog implementation to another, so in this section we are only giving the spirit of the language by using a Prolog-like pseudocode.) Here “bear,” “fish,” “fox,” “deer,” and “grass” are constants because they represent specific elements in the domain. Because the domain itself is never specified except by describing predicates, at this point we may take the domain to consist of “bear,” “fish,” “fox,” “deer,” and

“grass.” It is up to the user to maintain a consistent understanding and use of the predicates in a Prolog program. Thus

```
eat(bear, fish)
```

can be used either to represent the fact that bears eat fish or the fact that fish eat bears!

We impose the convention that $eat(X, Y)$ means “ X eats Y .” We could add descriptions of two unary predicates, *animal* and *plant*, to the database by adding the facts

```
animal(bear)
animal(fish)
animal(fox)
animal(deer)
plant(grass)
```

Armed with this Prolog program (database), we can pose some simple queries.

EXAMPLE 37

The query

```
?animal(bear)
```

merely asks if the fact *animal(bear)* is in the database. Because this fact is in the database, Prolog would respond to the query by answering yes. (This is a one-step proof sequence—no rules of inference are required). Further dialogue with Prolog could include

```
?eat(deer, grass)
yes
?eat(bear, rabbit)
no
```

Queries may include variables, as shown in the next example.

EXAMPLE 38

The query

```
?eat(bear, X)
```

produces

```
fish
fox
```

as a response. Prolog has answered the query by searching the database for all facts that match the pattern $eat(bear, X)$, where X is a variable. The answer “fish” is given first because the rules are searched in order from top to bottom.

Queries may contain the logical connectives **and**, **or**, and **not**.

PRACTICE 28 | Given the database

```
eat(bear, fish)
eat(bear, fox)
eat(deer, grass)
animal(bear)
animal(fish)
animal(fox)
animal(deer)
plant(grass)
```

what will be Prolog's response to the query

```
?eat(X, Y) and plant(Y)
```

The second type of item in a Prolog database is a **Prolog rule**. A rule is a description of a predicate by means of an implication (the implication arrow in the rule goes from right to left). For example, we might use a rule to define a predicate of *prey*:

$$prey(X) \leftarrow eat(Y, X) \text{ and } animal(X)$$

This statement says that X is a prey if it is an animal that is eaten. If we add this rule to our database, then in response to the query

```
?prey(X)
```

we would get

```
fish
fox
```

Horn Clauses and Resolution

How do Prolog facts and rules relate to more formal predicate logic? We can describe the facts in our database as wffs:

```
E(b, fi)
E(b, fo)
E(d, g)
A(b)
A(fi)
A(fo)
A(d)
P(g)
```

and the rule by the wff

$$E(y, x) \wedge A(x) \rightarrow Pr(x)$$

Universal quantifiers are not explicitly part of the rule as it appears in a Prolog program, but Prolog treats the rule as being universally quantified

$$(\forall y)(\forall x)[E(y, x) \wedge A(x) \rightarrow Pr(x)]$$

and repeatedly uses universal instantiation to strip off the universal quantifiers and allow the variables to assume in turn each value of the domain.

Both facts and rules are examples of Horn clauses. A **Horn clause** is a wff composed of predicates or the negations of predicates (with either variables or constants as arguments) joined by disjunctions, where at most one predicate is unnegated. Thus the fact

$$E(d, g)$$

is an example of a Horn clause because it consists of a single unnegated predicate. The wff

$$[E(y, x)]' \vee [A(x)]' \vee Pr(x)$$

is an example of a Horn clause because it consists of three predicates joined by disjunction where only $Pr(x)$ is unnegated. By De Morgan's law, it is equivalent to

$$[E(y, x) \wedge A(x)]' \vee Pr(x)$$

which in turn is equivalent to

$$E(y, x) \wedge A(x) \rightarrow Pr(x)$$

and therefore represents the rule in our Prolog program.

The single rule of inference used by Prolog is called **resolution**. Two Horn clauses in a Prolog database are resolved into a single new Horn clause if one contains an unnegated predicate that matches a negated predicate in the other clause. The new clause eliminates the matching term and is then available to use in answering the query. For example,

REMINDER

Prolog's resolution rule looks for a term and its negation to infer one Horn clause from two.

$$A(a) \\ [A(a)]' \vee B(b)$$

resolves to $B(b)$. This says that from

$$A(a), [A(a)]' \vee B(b)$$

which is equivalent to

$$A(a), A(a) \rightarrow B(b)$$

Prolog infers

$$B(b)$$

which is just an application of modus ponens. Therefore Prolog's rule of inference includes modus ponens as a special case.

In applying the resolution rule, variables are considered to "match" any constant symbol. (This is the repeated application of universal instantiation.) In any resulting new clause, the variables are replaced with their associated constants in a consistent manner. Thus in response to the query $?prey(X)$, Prolog searches the database for a rule with the desired predicate $Pr(x)$ as the consequent. It finds

$$[E(y, x)]' \vee [A(x)]' \vee Pr(x)$$

It then proceeds through the database looking for other clauses that can be resolved with this clause. The first such clause is the fact $E(b, fi)$. These two clauses resolve into

$$[A(fi)]' \vee Pr(fi)$$

(Note that the constant fi has replaced x everywhere.) Using this new clause, it can be resolved with the fact $A(fi)$ to conclude $Pr(fi)$. Having reached all conclusions possible from resolution with the fact $E(b, fi)$, Prolog backtracks to search for another clause to resolve with the rule clause; this time around it would find $E(b, fo)$.

As a more complex example of resolution, suppose we add the rule

$$hunted(X) \leq preys(X)$$

to the database. This rule in symbolic form is

$$[Pr(x)] \rightarrow H(x)$$

or, as a Horn clause,

$$[Pr(x)]' \vee H(x)$$

It resolves with the rule defining prey

$$[E(y, x)]' \vee [A(x)]' \vee Pr(x)$$

to give the new rule

$$[E(y, x)]' \vee [A(x)]' \vee H(x)$$

The query

$$?hunted(X)$$

will use this new rule to conclude

fish
fox

EXAMPLE 39

Suppose that a Prolog database contains the following entries:

```
eat(bear, fish)
eat(fish, littlefish)
eat(littlefish, algae)
eat(raccoon, fish)
eat(bear, raccoon)
eat(bear, fox)
eat(fox, rabbit)
eat(rabbit, grass)
eat(bear, deer)
eat(deer, grass)
eat(wildcat, deer)
```

```
animal(bear)
animal(fish)
animal(littlefish)
animal(raccoon)
animal(fox)
animal(rabbit)
animal(deer)
animal(wildcat)
```

```
plant(grass)
plant(algae)
prey(X) <= eat(Y, X) and animal(X)
```

Then the following dialog with Prolog could take place:

```
?animal(rabbit)
yes
```

```
?eat(wildcat, grass)
no
```

```
?eat(X, fish)
bear
raccoon
```

```
?eat(X, Y) and plant(Y)
littlefish    algae
rabbit        grass
deer          grass
```

```
?prey(X)
fish
littlefish
fish
```

```

raccoon
fox
rabbit
deer
deer

```

Note that fish is listed twice as satisfying the last query because fish are eaten by bear (fact 1) and by raccoon (fact 3). Similarly, deer are eaten by both bear and wildcat. ●

PRACTICE 29

- Formulate a Prolog rule that defines the predicate *predator*.
- Adding this rule to the database of Example 39, what would be the response to the query

?predator(X) ■

Recursion

Prolog rules are implications. Their antecedents (remember that these will appear on the right side of the rules) may depend on facts, as in

$$prey(X) \leq eat(Y, X) \text{ and } animal(X)$$

or on other rules, as in

$$hunted(X) \leq prey(X)$$

The antecedent of a rule may also depend on that rule itself, in which case the rule is defined in terms of itself. A definition in which the item being defined is itself part of the definition is called a **recursive definition**.

As an example, suppose we wish to use the ecology database of Example 39 to study food chains. We can then define a binary relation *infoodchain*(*X*, *Y*), meaning “*Y* is in *X*’s food chain.” This, in turn, means one of two things:

- X* eats *Y* directly

or

- X* eats something that eats something that eats something ... that eats *Y*.

Case 2 can be rewritten as follows:

- X* eats *Z* and *Y* is in *Z*’s food chain.

Case 1 is simple to test from our existing facts, but without (2’), *infoodchain* means nothing different from *eat*. On the other hand, (2’) without (1) sends us

down an infinite path of something eating something eating something and so on, with nothing telling us when to stop. Recursive definitions always need a stopping point that consists of specific information.

The Prolog rule for *infoodchain* incorporates (1) and (2'):

$$\begin{aligned} \text{infoodchain}(X, Y) &\leq \text{eat}(X, Y) \\ \text{infoodchain}(X, Y) &\leq \text{eat}(X, Z) \text{ and } \text{infoodchain}(Z, Y) \end{aligned}$$

It is a recursive rule because it defines the predicate *infoodchain* in terms of *infoodchain*.

A recursive rule is necessary when the predicate being described is passed on from one object to the next. The predicate *infoodchain* has this property:

$$\text{infoodchain}(X, Y) \wedge \text{infoodchain}(Y, Z) \rightarrow \text{infoodchain}(X, Z)$$

EXAMPLE 40

After the *infoodchain* rule is added to the database of Example 39, the following query is made:

$$?\text{infoodchain}(\text{bear}, Y)$$

The response follows (numbers are added for reference purposes):

- | | |
|---------------|---------------|
| 1. fish | 7. fish |
| 2. raccoon | 8. littlefish |
| 3. fox | 9. algae |
| 4. deer | 10. rabbit |
| 5. littlefish | 11. grass |
| 6. algae | 12. grass |

Prolog applies the simple case of

$$\text{infoodchain}(\text{bear}, Y) \leq \text{eat}(\text{bear}, Y)$$

first, obtaining answers 1 through 4 directly from the facts *eat*(bear, fish), *eat*(bear, raccoon), and so on. Moving to the recursive case,

$$\text{infoodchain}(\text{bear}, Y) \leq \text{eat}(\text{bear}, Z) \text{ and } \text{infoodchain}(Z, Y)$$

a match of *eat*(bear, Z) occurs with Z equal to “fish.” Prolog then looks for all solutions to the relation *infoodchain*(fish, Y). Using first the simple case of *infoodchain*, a match occurs with the fact *eat*(fish, littlefish). This results in response 5, littlefish. There are no other facts of the form *eat*(fish, Y), so the next thing to try is the recursive case of *infoodchain*(fish, Y):

$$\text{infoodchain}(\text{fish}, Y) \leq \text{eat}(\text{fish}, Z) \text{ and } \text{infoodchain}(Z, Y)$$

A match of *eat*(fish, Z) occurs with Z equal to “littlefish.” Prolog then looks for all solutions to the relation *infoodchain*(littlefish, Y). Using the simple case of

infoodchain, a match occurs with the fact *eat*(littlefish, algae). This results in response 6, algae. There are no other facts of the form *eat*(littlefish, *Y*), so the next thing to try is the recursive case of *infoodchain*(littlefish, *Y*):

$$\text{infoodchain}(\text{littlefish}, Y) \leftarrow \text{eat}(\text{littlefish}, Z) \text{ and } \text{infoodchain}(Z, Y)$$

A match of *eat*(littlefish, *Z*) occurs with *Z* equal to “algae.” Prolog then looks for all solutions to the relation *infoodchain*(algae, *Y*). A search of the entire database reveals no facts of the form *eat*(algae, *Y*) (or *eat*(algae, *Z*)), so neither the simple case nor the recursive case of *infoodchain*(algae, *Y*) can be pursued further.

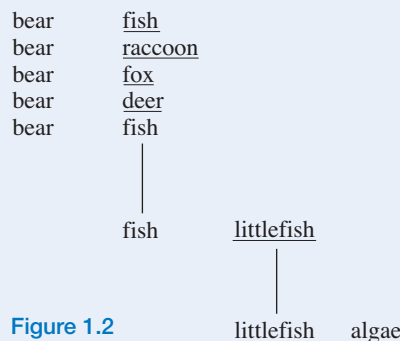


Figure 1.2

Figure 1.2 shows the situation at this point. Prolog has reached a dead-end with *infoodchain*(algae, *Y*) and will backtrack up the path. Because there are no other facts of the form *eat*(littlefish, *Z*), the search for solutions to *infoodchain*(littlefish, *Y*) terminates. Then, because there are no other facts of the form *eat*(fish, *Z*), the search for solutions to *infoodchain*(fish, *Y*) terminates. Backing up still further, there is another match of *eat*(bear, *Z*) with *Z* equal to “raccoon” that will generate another search path. ●

In Example 40, once Prolog began to investigate *infoodchain*(fish, *Y*), all query answers that could be obtained from exploring this path (responses 5 and 6) were generated before other answers (responses 7–12). Exploring as far as possible down a given path and then backtracking up that path before exploring other paths is called a **depth-first search** strategy.

PRACTICE 30

Trace the execution of the Prolog program of Example 40 and explain why responses 7–12 occur. ■

Expert Systems

Many interesting applications programs have been developed, in Prolog and similar logic programming languages, that gather a database of facts and rules about some domain and then use the database to draw conclusions. Such programs are known as **expert systems**, **knowledge-based systems**, or **rule-based systems**.

The database in an expert system attempts to capture the knowledge (“elicit the expertise”) of a human expert in a particular field, including both the facts known to the expert and the expert’s reasoning path in reaching conclusions from those facts. The completed expert system not only simulates the human expert’s actions but can be questioned to reveal why it made certain choices and not others.

Expert systems have been built that simulate a medical specialist’s diagnosis from a patient’s symptoms, a factory manager’s decisions regarding valve control in a chemical plant based on sensor readings, the decisions of a fashion buyer for a retail store based on market research, the choices made by a consultant specifying a computer system configuration based on customer needs, and many more. The challenging part of building an expert system lies in extracting all pertinent facts and rules from the human expert.

SECTION 1.5 REVIEW

TECHNIQUES

- W Formulate Prolog-like facts and rules.
- W Formulate Prolog-like queries.
 - Determine the answer(s) to a query using a Prolog database.

MAIN IDEA

- A declarative language incorporates predicate wffs and rules of inference to draw conclusions from hypotheses. The elements of such a language are based on predicate logic rather than instructions that carry out an algorithm.

EXERCISES 1.5

Exercises 1–8 refer to the database of Example 39; find the results of the query in each case.

1. $?animal(wildcat)$
2. $?plant(raccoon)$
3. $?eat(bear, littlefish)$
4. $?eat(fox, rabbit)$
5. $?eat(raccoon, X)$
6. $?eat(X, grass)$
7. $?eat(bear, X)$ and $eat(X, rabbit)$
8. $?prey(X)$ and not $eat(fox, X)$
9. Formulate a Prolog rule that defines “herbivore” to add to the database of Example 39.
10. If the rule of Exercise 9 is included in the database of Example 39, what is the response to the query

$$?herbivore(X)$$

11. After *infoodchain* is added to the database of Example 39, add the facts $eat(wolf, fox)$ and $eat(wolf, deer)$. What is the result of the query

$$?eat(wolf, X) \text{ and not } eat(X, grass)$$

12. After the modifications in Exercise 11, what is the result of the query

$$?infoodchain(wolf, X)$$

13. A Prolog database contains the following, where $\text{boss}(X, Y)$ means “ X is Y ’s boss” and $\text{supervisor}(X, Y)$ means “ X is Y ’s supervisor”:

```

boss(mike, joan)
boss(judith, mike)
boss(anita, judith)
boss(judith, kim)
boss(kim, enrique)
boss(anita, sam)
boss(enrique, jefferson)
boss(mike, hamal)

supervisor(X, Y) <= boss(X, Y)
supervisor(X, Y) <= boss(X, Z) and supervisor(Z, Y)

```

Find the results of the following queries:

- a. $?boss(X, \text{sam})$
 - b. $?boss(\text{judith}, X)$
 - c. $?supervisor(\text{anita}, X)$
14. Using the Prolog database from Exercise 13, what are the results of the following queries?
- a. $?boss(\text{hamal}, X)$
 - b. $?supervisor(X, \text{kim})$
15. Suppose a Prolog database exists that gives information about authors and the books they have written. Books are classified as fiction, biography, or reference.
- a. Write a query to ask whether Mark Twain wrote *Hound of the Baskervilles*.
 - b. Write a query to find all books written by William Faulkner.
 - c. Formulate a rule to define nonfiction authors.
 - d. Write a query to find all nonfiction authors.
16. Suppose a Prolog database exists that gives information about states and capital cities. Some cities are big, others small. Some states are eastern, others are western.
- a. Write a query to find all the small capital cities.
 - b. Write a query to find all the states with small capital cities.
 - c. Write a query to find all the eastern states with big capital cities.
 - d. Formulate a rule to define cosmopolitan cities as big capitals of western states.
 - e. Write a query to find all the cosmopolitan cities.
17. Suppose a Prolog database exists that gives information about a family. Predicates of *male*, *female*, and *parentof* are included.
- a. Formulate a rule to define *fatherof*.
 - b. Formulate a rule to define *daughterof*.
 - c. Formulate a recursive rule to define *ancestorof*.
18. Suppose a Prolog database exists that gives information about the parts in an automobile engine. Predicates of *big*, *small*, and *partof* are included.
- a. Write a query to find all small items that are part of other items.
 - b. Write a query to find all big items that have small subitems.
 - c. Formulate a recursive rule to define *componentof*.

19. Suppose a Prolog database exists that gives information about the ingredients in the menu items of a restaurant. Predicates of *dry*, *liquid*, *perishable*, and *ingredientof* are included.
 - a. Write a query to find all the dry ingredients of other ingredients.
 - b. Write a query to find all perishable ingredients that contain liquid subingredients.
 - c. Formulate a recursive rule to define *foundin*.
20. Suppose a Prolog database exists that gives information about flights for AA (Always Airborne) airline. Predicates of *city* and *flight* are included. Here *flight*(*X*, *Y*) means that AA has a direct (nonstop) flight from city *X* to city *Y*.
 - a. Write a query to find all cities you can get to on a direct flight from Indianapolis.
 - b. Write a query to find all cities that have direct flights to San Francisco.
 - c. Formulate a recursive rule to define *route* where *route*(*X*, *Y*) means that you can get from city *X* to city *Y* using AA but it might not be a direct flight.

Exercises 21–22 refer to a “Toy Prolog interpreter” that can be found online at <http://www.csse.monash.edu.au/~lloyd/tildeLogic/Prolog.toy>. The syntax used in this section matches that of this online version except that in the online version each statement and query must end with a period. Here is a shortened version of Example 39 as entered into the code window, followed by the response to the query:

```
eat(bear, fish).
eat(fish, littlefish).
eat(littlefish, algae).
eat(raccoon, fish).
eat(bear, raccoon).
eat(bear, fox).
animal(bear).
animal(fish).
animal(littlefish).
animal(raccoon).
animal(fox).
prey(X) <= eat(Y, X) and animal(X).


```

In addition, you can look at and run the online sample program to be sure you understand the syntax rules.

21. Using the online Toy Prolog program, enter the Prolog database of Exercise 13. Run the queries from Exercises 13 and 14 and compare the results with your previous answers.
22. Using the online Toy Prolog program, create a database for Exercise 20. Run the queries from Exercise 20. Also run a query using the *route* predicate.

SECTION 1.6 PROOF OF CORRECTNESS

As our society becomes ever more dependent on computers, it is more and more important that the programs computers run are reliable and error-free. **Program verification** attempts to ensure that a computer program is correct.

“Correctness” has a narrower definition here than in everyday usage. A program is **correct** if it behaves in accordance with its specifications. However, this does not necessarily mean that the program solves the problem that it was intended to solve; the program’s specifications may be at odds with or not address all aspects of a client’s requirements. **Program validation**, which we won’t discuss further, attempts to ensure that the program indeed meets the client’s original requirements. In a large program development project “program V & V” or “software quality assurance” is considered so important that a group of people separate from the programmers is often designated to carry out the associated tasks.

Program verification may be approached both through program testing and through *proof of correctness*. **Program testing** seeks to show that particular input values produce acceptable output values. Program testing is a major part of any software development effort, but it is well-known folklore that “testing can prove the presence of errors but never their absence.” If a test run under a certain set of conditions with a certain set of input data reveals a “bug” in the code, then the bug can be corrected. But except for rather simple programs, multiple tests that reveal no bugs do not guarantee that the code is bug-free, that there is not some error lurking in the code waiting to strike under the right circumstances.

As a complement to testing, computer scientists have developed a more mathematical approach to “prove” that a program is correct. **Proof of correctness** uses the techniques of a formal logic system to prove that if the input variables satisfy certain specified predicates or properties, the output variables produced by executing the program satisfy other specified properties.

To distinguish between proof of correctness and program testing, consider a program to compute the length c of the hypotenuse of a right triangle, given positive values a and b for the lengths of the legs. Proving the program correct would establish that whenever a and b satisfy the predicates $a > 0$ and $b > 0$, then after the program is executed, the predicate $a^2 + b^2 = c^2$ is satisfied. Testing such a program would require taking various specific values for a and b , computing the resulting c , and checking that $a^2 + b^2$ equals c^2 in each case. However, only representative values for a and b can be tested, not all possible values.

Again, testing and proof of correctness are complementary aspects of program verification. All programs undergo program testing; they may or may not undergo proof of correctness as well. Proof of correctness is labor-intensive, hence expensive; it generally is applied only to small and critical sections of code rather than to the entire program.

Assertions

Describing proof of correctness more formally, let us denote by X an arbitrary collection of input values to some program or program segment P . The actions of P transform X into a corresponding group of output values Y ; the notation $Y = P(X)$ suggests that the Y values depend on the X values through the actions of program P .

A predicate $Q(X)$ describes conditions that the input values are supposed to satisfy. For example, if a program is supposed to find the square root of a positive number, then X consists of one input value, x , and $Q(x)$ might be “ $x > 0$.”

A predicate R describes conditions that the output values are supposed to satisfy. These conditions will often involve the input values as well, so R has the form $R(X, Y)$ or $R[X, P(X)]$. In our square root case, if y is the single output value, then y is supposed to be the square root of x , so $R(x, y)$ would be “ $y^2 = x$.” Program P is correct if the implication

$$(\forall X)(Q(X) \rightarrow R[X, P(X)]) \quad (1)$$

is valid. In other words, whenever Q is true about the input values, R should be true about the input and output values. For the square root case, (1) is

$$(\forall x)(x > 0 \rightarrow [P(x)]^2 = x)$$

The implication (1) is standard predicate wff notation, but the traditional program correctness notation for (1) is

$$\{Q\}P\{R\} \quad (2)$$

$\{Q\}P\{R\}$ is called a **Hoare triple**, named for the British computer scientist Anthony Hoare. Condition Q is called the **precondition** for program P , and condition R is the **postcondition**. In the Hoare notation, the universal quantifier does not explicitly appear; it is understood.

Rather than simply having an initial predicate and a final predicate, a program or program segment is broken down into individual statements s_i , with predicates inserted between statements as well as at the beginning and end. These predicates are also called **assertions** because they assert what is supposed to be true about the program variables at that point in the program. Thus we have

$$\begin{array}{c} \{Q\} \\ s_0 \\ \{R_1\} \\ s_1 \\ \{R_2\} \\ \vdots \\ s_{n-1} \\ \{R\} \end{array}$$

where $Q, R_1, R_2, \dots, R_n = R$ are assertions. The intermediate assertions are often obtained by working backward from the output assertion R .

P is provably correct if each of the following implications holds:

$$\begin{array}{c} \{Q\}s_0\{R_1\} \\ \{R_1\}s_1\{R_2\} \\ \{R_2\}s_2\{R_3\} \\ \vdots \\ \{R_{n-1}\}s_{n-1}\{R\} \end{array}$$

A proof of correctness for P consists of producing this sequence of valid implications, that is, producing a proof sequence of predicate wffs. Some new rules of inference can be used, based on the nature of the program statement s_i .

Assignment Rule

Suppose that statement s_i is an assignment statement of the form $x = e$, that is, the variable x takes on the value of e , where e is some expression. The Hoare triple to prove correctness of this one statement has the form

$$\{R_i\} x = e \{R_{i+1}\}$$

For this triple to be valid, the assertions R_i and R_{i+1} must be related in a particular way.

EXAMPLE 41

Consider the following assignment statement together with the given precondition and postcondition:

$$\begin{array}{l} \{x - 1 > 0\} \\ x = x - 1 \\ \{x > 0\} \end{array}$$

For every x , if $x - 1 > 0$ before the statement is executed (note that this says that $x > 1$), then after the value of x is reduced by 1, it will be the case that $x > 0$. Therefore,

$$\{x - 1 > 0\} x = x - 1 \{x > 0\}$$

is valid. ●

In Example 41, we just reasoned our way through the validity of the wff represented by the Hoare triple. The point of predicate logic is to allow us to determine validity in a more mechanical fashion by the application of rules of inference. (After all, we don't want to just "reason our way through" the entire program to convince ourselves of its correctness; the programmer already did that when the program was written!)

The appropriate rule of inference for assignment statements is the **assignment rule**, given in Table 1.18. It says that if the precondition and postcondition are appropriately related, the Hoare triple can be inserted at any time in a proof sequence without having to be inferred from something earlier in the proof sequence. This makes the Hoare triple for an assignment statement akin to a hypothesis in our previous proofs. And what is the relationship? In the postcondition, locate all instances of the variable to which an assignment is being made in the assignment statement right above the postcondition. For each of those instances, substitute the expression being assigned. The result will be the precondition.

TABLE 1.18

From	Can Derive	Name of Rule	Restrictions on Use
	$\{R_i\}s_i\{R_{i+1}\}$	assignment	<ol style="list-style-type: none"> s_i has the form $x = e$. R_i is R_{i+1} with e substituted everywhere for x.

EXAMPLE 42

For the case of Example 41,

$$\begin{aligned} &\{x - 1 > 0\} \\ &\quad x = x - 1 \\ &\{x > 0\} \end{aligned}$$

the triple

$$\{x - 1 > 0\} \ x = x - 1 \ \{x > 0\}$$

is valid by the assignment rule. The postcondition is

$$x > 0$$

Substituting $x - 1$ for x throughout the postcondition results in

$$x - 1 > 0 \quad \text{or} \quad x > 1$$

which is the precondition. Here we didn't have to think at all; we just checked that the assignment rule had been followed. ●

Certainly Example 41 seems easier than Example 42, and for such a trivial case you may be tempted to skip use of the assignment inference rule and just talk your way through the code. Resist this temptation. For one thing, real-world usage isn't this trivial. But more to the point, just as in our previous formal logic systems, you want to rely on the rules of inference instead of on some possibly flawed thought process.

PRACTICE 31

According to the assignment rule, what should be the precondition in the following program segment?

$$\begin{aligned} &\{\text{precondition}\} \\ &\quad x = x - 2 \\ &\{x = y\} \end{aligned}$$
REMINDER

To use the assignment rule, work from the bottom to the top.

Because the assignment rule tells us what a precondition should look like based on what a postcondition looks like, a proof of correctness often begins with the final desired postcondition and works its way back up through what the earlier assertions should look like according to the assignment rule. Once it has been determined what the topmost assertion must be, a check is done to see that this assertion is really true.

EXAMPLE 43

Verify the correctness of the following program segment to exchange the values of x and y :

$$\begin{aligned} &\text{temp} = x \\ &\quad x = y \\ &\quad y = \text{temp} \end{aligned}$$

At the beginning of this program segment, x and y have certain values. Thus we may express the actual precondition as $x = a$ and $y = b$. The desired postcondition is then $x = b$ and $y = a$. Using the assignment rule, we can work backward from the postcondition to find the earlier assertions (read the following from the bottom to the top).

$$\begin{aligned} &\{y = b, x = a\} \\ &\quad \text{temp} = x \\ &\{y = b, \text{temp} = a\} \\ &\quad x = y \\ &\{x = b, \text{temp} = a\} \\ &\quad y = \text{temp} \\ &\{x = b, y = a\} \end{aligned}$$

The first assertion agrees with the precondition; the assignment rule, applied repeatedly, assures us that the program segment is correct. ●

PRACTICE 32 Verify the correctness of the following program segment with the precondition and postcondition shown:

$$\begin{aligned} &\{x = 3\} \\ &\quad y = 4 \\ &\quad z = x + y \\ &\{z = 7\} \end{aligned}$$

Sometimes the necessary precondition is trivially true, as shown in the next example.

EXAMPLE 44 Verify the correctness of the following program segment to compute $y = x - 4$.

$$\begin{aligned} &y = x \\ &y = y - 4 \end{aligned}$$

Here the desired postcondition is $y = x - 4$. Using the assignment rule to work backward from the postcondition, we get (again, read bottom to top)

$$\begin{aligned} &\{x - 4 = x - 4\} \\ &\quad y = x \\ &\{y - 4 = x - 4\} \\ &\quad y = y - 4 \\ &\{y = x - 4\} \end{aligned}$$

The precondition is always true; therefore, by the assignment rule, each successive assertion, including the postcondition, is true. ●

Conditional Rule

A **conditional statement** is a program statement of the form

```

if condition  $B$  then
     $P_1$ 
else
     $P_2$ 
end if
  
```

When this statement is executed, a condition B that is either true or false is evaluated. If B is true, program segment P_1 is executed, but if B is false, program segment P_2 is executed.

A **conditional rule of inference**, shown in Table 1.19, determines when a Hoare triple

$$\{Q\}s_i\{R\}$$

can be inserted in a proof sequence if s_i is a conditional statement. The Hoare triple is inferred from two other Hoare triples. One of these says that if Q is true and B is true and program segment P_1 is executed, then R holds; the other says that if Q is true and B is false and program segment P_2 is executed, then R holds. This simply says that each branch of the conditional statement must be proved correct.

TABLE 1.19			
From	Can Derive	Name of Rule	Restrictions on Use
$\{Q \wedge B\} P_1 \{R\},$ $\{Q \wedge B'\} P_2 \{R\}$	$\{Q\}s_i\{R\}$	conditional	s_i has the form if condition B then P_1 else P_2 end if

EXAMPLE 45

Verify the correctness of the following program segment with the precondition and postcondition shown.

```

 $\{n = 5\}$ 
if  $n \geq 10$  then
     $y = 100$ 
else
     $y = n + 1$ 
end if
 $\{y = 6\}$ 
  
```

Here the precondition is $n = 5$, and the condition B to be evaluated is $n \geq 10$. In order to apply the conditional rule, we must first prove that

$$\{Q \wedge B\} P_1 \{R\}$$

or

$$\{n = 5 \text{ and } n \geq 10\} y = 100 \{y = 6\}$$

holds. Remember that this stands for an implication, which will be true because its antecedent, $n = 5$ and $n \geq 10$, is false. We must also show that

$$\{Q \wedge B'\} P_2 \{R\}$$

or

$$\{n = 5 \text{ and } n < 10\} y = n + 1 \{y = 6\}$$

holds. Working back from the postcondition, using the assignment rule, we get

$$\begin{aligned} \{n + 1 = 6 \text{ or } n = 5\} \\ y = n + 1 \\ \{y = 6\} \end{aligned}$$

Thus

$$\{n = 5\} y = n + 1 \{y = 6\}$$

is true by the assignment rule and therefore

$$\{n = 5 \text{ and } n < 10\} y = n + 1 \{y = 6\}$$

is also true because the condition $n < 10$ adds nothing new to the assertion. The conditional rule allows us to conclude that the program segment is correct. ●

PRACTICE 33

Verify the correctness of the following program segment with the precondition and postcondition shown.

```
{x = 4}
if x < 5 then
  y = x - 1
else
  y = 7
end if
{y = 3}
```

EXAMPLE 46

Verify the correctness of the following program segment to compute $\max(x, y)$, the maximum of two distinct values x and y .

```
{x ≠ y}
if x >= y then
  max = x
else
  max = y
end if
```

The desired postcondition reflects the definition of the maximum, $(x > y$ and $\max = x)$ or $(x < y$ and $\max = y)$. The two implications to prove are

$$\{x \neq y \text{ and } x \geq y\} \max = x \{(x > y \text{ and } \max = x) \text{ or } (x < y \text{ and } \max = y)\}$$

and

$$\{x \neq y \text{ and } x < y\} \max = y \{(x > y \text{ and } \max = x) \text{ or } (x < y \text{ and } \max = y)\}$$

Using the assignment rule on the first case (substituting x for \max in the postcondition) would give the precondition

$$(x > y \wedge x = x) \vee (x < y \wedge x = y)$$

Since the second disjunct is always false, this is equivalent to

$$(x > y \wedge x = x)$$

which in turn is equivalent to

$$x > y \quad \text{or} \quad x \neq y \text{ and } x \geq y$$

The second implication is proved similarly. ●

In Chapter 2, we will see how to verify correctness for a loop statement, where a section of code can be repeated many times.

As we have seen, proof of correctness involves a lot of detailed work. It is a difficult tool to apply to large programs that already exist. It is generally easier to prove correctness while the program is being developed. Indeed, the list of assertions from beginning to end specifies the intended behavior of the program and can be used early in its design. In addition, the assertions serve as valuable documentation after the program is complete.

SECTION 1.6 REVIEW

TECHNIQUES

- W Verify the correctness of a program segment that includes assignment statements.
- W Verify the correctness of a program segment that includes conditional statements.

MAIN IDEA

- A formal system of rules of inference can be used to prove the correctness of program segments.

EXERCISES 1.6

In the following exercises, * denotes multiplication.

1. According to the assignment rule, what is the precondition in the following program segment?

$$\begin{array}{l} \{\text{precondition}\} \\ x = x + 1 \\ \{x = y - 1\} \end{array}$$

2. According to the assignment rule, what is the precondition in the following program segment?


```
{precondition}
  x = 2 * x
{x > y}
```
3. According to the assignment rule, what is the precondition in the following program segment?


```
{precondition}
  x = 3 * x - 1
{x = 2 * y - 1}
```
4. According to the assignment rule, what is the precondition in the following program segment?


```
{precondition}
  y = 3x + 7
{y = x + 1}
```
5. Verify the correctness of the following program segment with the precondition and postcondition shown.


```
{x = 1}
  y = x + 3
  y = 2 * y
{y = 8}
```
6. Verify the correctness of the following program segment with the precondition and postcondition shown.


```
{x > 0}
  y = x + 2
  z = y + 1
{z > 3}
```
7. Verify the correctness of the following program segment with the precondition and postcondition shown.


```
{x = 0}
  z = 2 * x + 1
  y = z - 1
{y = 0}
```
8. Verify the correctness of the following program segment with the precondition and postcondition shown.


```
{x < 8}
  z = x - 1
  y = z - 5
{y < 2}
```
9. Verify the correctness of the following program segment to compute $y = x(x - 1)$.


```
y = x - 1
y = x * y
```
10. Verify the correctness of the following program segment to compute $y = 2x + 1$.


```
y = x
y = y + y
y = y + 1
```
11. Verify the correctness of the following program segment with the precondition and postcondition shown.


```
{y = 0}
  if y < 5 then
    y = y + 1
  else
    y = 5
  end if
{y = 1}
```

12. Verify the correctness of the following program segment with the precondition and postcondition shown.

```
{x = 7}
  if x ≤ 0 then
    y = x
  else
    y = 2 * x
  end if
{y = 14}
```

13. Verify the correctness of the following program segment with the precondition and postcondition shown.

```
{x ≠ 0}
  if x > 0 then
    y = 2 * x
  else
    y = (-2) * x
  end if
{y > 0}
```

14. Verify the correctness of the following program segment to compute $\min(x, y)$, the minimum of two distinct values x and y .

```
{x ≠ y}
  if x ≤ y then
    min = x
  else
    min = y
  end if
```

15. Verify the correctness of the following program segment to compute $|x|$, the absolute value of x , for a nonzero number x .

```
{x ≠ 0}
  if x ≥ 0 then
    abs = x
  else
    abs = -x
  end if
```

16. Verify the correctness of the following program segment with the assertions shown.

```
{z = 3}
  x = z + 1
  y = x + 2
{y = 6}
  if y > 0 then
    z = y + 1
  else
    z = 2 * y
  end if
{z = 7}
```

CHAPTER 1 REVIEW

TERMINOLOGY

- algorithm (p. 12)
- antecedent (p. 3)
- assertion (p. 86)
- assignment rule (p. 87)
- binary connective (p. 3)
- binary predicate (p. 40)
- complete formal logic system (p. 27)
- conclusion (p. 25)
- conditional rule of inference (p. 90)
- conditional statement (p. 90)
- conjunct (p. 2)
- conjunction (p. 2)
- consequent (p. 3)
- contradiction (p. 8)
- correct formal logic system (p. 27)
- correct program (p. 85)
- De Morgan's laws (p. 10)
- declarative language (p. 73)
- depth-first search (p. 81)
- derivation rule (p. 27)
- descriptive language (p. 73)
- disjunct (p. 3)
- disjunction (p. 3)
- domain (p. 41)
- dual of an equivalence (p. 9)
- equivalence (p. 3)
- equivalence rule (p. 28)
- equivalent wffs (p. 8)
- existential generalization (p. 62)
- existential instantiation (p. 60)
- existential quantifier (p. 40)
- expert system (p. 81)
- free variable (p. 42)
- Hoare triple (p. 86)
- Horn clause (p. 76)
- hypothesis (p. 25)
- implication (p. 3)
- inference rule (p. 29)
- interpretation (p. 41)
- knowledge-based system (p. 81)
- logical connective (p. 2)
- main connective (p. 6)
- n -ary predicate (p. 40)
- negation (p. 4)
- postcondition (p. 86)
- precondition (p. 86)
- predicate (p. 39)
- predicate logic (p. 58)
- predicate wff (p. 41)
- procedural language (p. 73)
- program testing (p. 85)
- program validation (p. 85)
- program verification (p. 84)
- Prolog database (p. 73)
- Prolog fact (p. 73)
- Prolog rule (p. 75)
- proof of correctness (p. 85)
- proof sequence (p. 27)
- proposition (p. 2)
- propositional calculus (p. 25)
- propositional logic (p. 25)
- propositional wff (p. 25)
- pseudocode (p. 12)
- recursive definition (p. 79)
- resolution (p. 76)
- rule-based system (p. 81)
- scope (p. 41)
- statement (p. 2)
- statement letter (p. 2)
- statement logic (p. 25)
- tautology (p. 8)
- ternary predicate (p. 40)
- unary connective (p. 3)
- unary predicate (p. 40)
- universal generalization (p. 61)
- universal instantiation (p. 59)
- universal quantifier (p. 39)
- valid argument (p. 26, 58)
- valid predicate wff (p. 48)
- well-formed formula (wff) (p. 6)

SELF TEST

Answer the following true–false questions without looking back in the chapter.

Section 1.1

1. A contradiction is any propositional wff that is not a tautology.
2. The disjunction of any propositional wff with a tautology has the truth value true.
3. Algorithm *TautologyTest* determines whether any propositional wff is a tautology.
4. Equivalent propositional wffs have the same truth values for every truth value assignment to the components.
5. One of De Morgan's laws states that the negation of a disjunction is the disjunction of the negations (of the disjuncts).

Section 1.2

1. An equivalence rule allows one wff to be substituted for another in a proof sequence.
2. If a propositional wff can be derived using modus ponens, then its negation can be derived using modus tollens.
3. Propositional logic is complete because every tautology is provable.
4. A valid argument is one in which the conclusion is always true.
5. The deduction method applies when the conclusion is an implication.

Section 1.3

1. A predicate wff that begins with a universal quantifier is universally true, that is, true in all interpretations.
2. In the predicate wff $(\forall x)P(x, y)$, y is a free variable.
3. An existential quantifier is usually found with the conjunction connective.
4. The domain of an interpretation consists of the values for which the predicate wff defined on that interpretation is true.
5. A valid predicate wff has no interpretation in which it is false.

Section 1.4

1. The inference rules of predicate logic allow existential and universal quantifiers to be added or removed during a proof sequence.
2. Existential instantiation should be used only after universal instantiation.
3. $P(x) \wedge (\exists x)Q(x)$ can be deduced from $(\forall x)[P(x) \wedge (\exists y)Q(y)]$ using universal instantiation.
4. Every provable wff of propositional logic is also provable in predicate logic.
5. A predicate wff that is not valid cannot be proved using predicate logic.

ON THE COMPUTER

For Exercises 1–5, write a computer program that produces the desired output from the given input.

1. *Input:* Truth values for two statement letters A and B
Output: Corresponding truth values (appropriately labeled, of course) for

$$A \wedge B, A \vee B, A \rightarrow B, A \leftrightarrow B, A'$$

2. *Input:* Truth values for two statement letters A and B
Output: Corresponding truth values for the wffs

$$A \rightarrow B' \text{ and } B' \wedge [A \vee (A \wedge B)]$$

3. *Input:* Truth values for three statement letters A , B , and C
Output: Corresponding truth values for the wffs

$$A \vee (B \wedge C') \rightarrow B' \text{ and } A \vee C' \leftrightarrow (A \vee C)'$$

4. *Input:* Truth values for three statement letters A , B , and C , and a representation of a simple

Section 1.5

1. A Prolog rule describes a predicate.
2. Horn clauses are wffs consisting of single negated predicates.
3. Modus ponens is a special case of Prolog resolution.
4. A Prolog recursive rule is a rule of inference that is used more than once.
5. A Prolog inference engine applies its rule of inference without guidance from either the programmer or the user.

Section 1.6

1. A provably correct program always gives the right answers to a given problem.
2. If an assertion after an assignment statement is $y > 4$, then the precondition must be $y \geq 4$.
3. Proof of correctness involves careful development of test data sets.
4. Using the conditional rule of inference in proof of correctness involves proving that two different Hoare triples are valid.
5. The assertions used in proof of correctness can also be used as a program design aid before the program is written, and as program documentation.

propositional wff. Special symbols can be used for the logical connectives, and postfix notation can be used; for example,

$$A B \wedge C \vee \text{ for } (A \wedge B) \vee C$$

or

$$A' B \wedge \text{ for } A' \wedge B$$

Output: Corresponding truth value of the wff

5. *Input:* Representation of a simple propositional wff as in the previous exercise

Output: Decision on whether the wff is a tautology

6. Using the online Toy Prolog program that can be found at <http://www.csse.monash.edu.au/~lloyd/tildeLogic/Prolog.toy>, enter the Prolog database of Example 39 and perform the queries there. Note that each database entry requires a period. Also add the recursive rule for *infoodchain* and perform the query

$$?infoodchain(bear, Y)$$

Proofs, Induction, and Number Theory

CHAPTER OBJECTIVES

After studying this chapter, you will be able to:

- Attack the proofs of conjectures using the techniques of direct proof, proof by contraposition, and proof by contradiction.
- Recognize when a proof by induction is appropriate and carry out such a proof using either the first or second principle of induction.
- Mathematically prove the correctness of programs that use loop statements.
- Test whether a given positive integer is prime; if not, find its prime factorization.
- Work with number theoretic ideas of prime factorization, greatest common divisor, and the Euler phi function.

The nonprofit organization at which you volunteer has received donations of 792 bars of soap and 400 bottles of shampoo. You want to create packages to distribute to homeless shelters such that each package contains the same number of shampoo bottles and each package contains the same number of bars of soap.

Question: How many packages can you create?

This problem is solvable by trial and error, but it's much easier to use an ancient algorithm that is discussed in this chapter.

First, however, we consider how to prove “real-world” arguments as opposed to the formal arguments of Chapter 1. It is helpful to have an arsenal of techniques for attacking a proof. Direct proof, proof by contraposition, and proof by contradiction are examined in Section 2.1. Many of the proofs given in this section are simple “number theory” results, that is, results about integers, such as “The product of two even integers is even.”

Section 2.2 concentrates on mathematical induction, a proof technique with particularly wide application in computer science. In Section 2.3 we see how, using induction, proof of correctness can be extended to cover looping statements. Finally, Section 2.4 explores some further number theory results, particularly concerning prime numbers.

SECTION 2.1 PROOF TECHNIQUES

Theorems and Informal Proofs

The formal arguments of Chapter 1 have the form $P \rightarrow Q$, where P and Q may represent compound statements. The point there was to prove that an argument is valid—true in all interpretations by nature of its internal form or structure, not because of its content or the meaning of its component parts. However, we often want to prove arguments that are not universally true but just true within some context. Meaning becomes important because we are discussing a particular subject—graph algorithms, or Boolean algebra, or compiler theory, or whatever—and we want to prove that if P is true in this context, then so is Q . If we can do this, then $P \rightarrow Q$ becomes a *theorem* about that subject. To prove a theorem about subject XXX, we can introduce facts about XXX into the proof; these facts act like additional hypotheses. Note that as we add more hypotheses, the universe of discourse shrinks; we are no longer considering universally valid arguments, only arguments that are true within the context in which the hypotheses hold.¹

It may not be easy to recognize which subject-specific facts will be helpful or to arrange a sequence of steps that will logically lead from P to Q . Unfortunately, there is no formula for constructing proofs and no practical general algorithm or computer program for proving theorems. Experience is helpful, not only because you get better with practice, but also because a proof that works for one theorem can sometimes be modified to work for a new but similar theorem.

Theorems are often stated and proved in a somewhat less formal way than the propositional and predicate arguments of Chapter 1. For example, a theorem may express the fact that every object in the domain of interpretation (the subject matter under discussion) having property P also has property Q . The formal statement of the theorem would be $(\forall x)[P(x) \rightarrow Q(x)]$. But the theorem would be informally stated as $P(x) \rightarrow Q(x)$. If we can prove $P(x) \rightarrow Q(x)$ where x is treated as an arbitrary element of the domain, universal generalization would then give $(\forall x)[P(x) \rightarrow Q(x)]$.

As another example, we may know that all objects in the domain have some property; that is, something of the form $(\forall x)P(x)$ can be considered as a subject-specific fact. An informal proof might proceed by saying, “Let x be any element of the domain. Then x has property P .” (Formally, we are making use of universal instantiation to get $P(x)$ from $(\forall x)P(x)$.)

Similarly, proofs are usually not written a step at a time with formal justifications for each step. Instead, the important steps and their rationale are outlined in narrative form. Such a narrative, however, can be translated into a formal proof if required. In fact, the value of a formal proof is that it serves as a sort of insurance—if a narrative proof *cannot* be translated into a formal proof, it should be viewed with great suspicion.

¹In the world of “pure predicate logic,” which is a correct and complete formal system, every true (valid) argument is provable. But in these more restrictive contexts, not everything that is “true” is necessarily provable, no matter how clever we are in adding additional hypotheses or “axioms.” In other words, these systems may not be complete. At the age of 25, the German logician Kurt Gödel proved in 1931 that, using reasonable hypotheses, even elementary arithmetic is an incomplete system. This result shocked the mathematical community of the time, which had been depending on axiomatic systems since the days of Euclid.

To Prove or Not to Prove

A textbook will often say, “Prove the following theorem,” and the reader will know that the theorem is true; furthermore, it is probably stated in its most polished form. But suppose you are doing research in some subject. You observe a number of cases in which whenever P is true, Q is also true. On the basis of these experiences, you may formulate a conjecture: $P \rightarrow Q$. The more cases you find where Q follows from P , the more confident you are in your conjecture. This process illustrates **inductive reasoning**, drawing a conclusion based on experience.

No matter how reasonable the conjecture sounds, however, you will not be satisfied until you have applied **deductive reasoning** to it as well. In this process, you try to verify the truth or falsity of your conjecture. You produce a proof of $P \rightarrow Q$ (thus making it a theorem), or else you find a **counterexample** that disproves the conjecture, a case in which P is true but Q is false. (We were using deductive reasoning in predicate logic when we either proved that a wff was valid or found an interpretation in which the wff was false.)

If you are simply presented with a conjecture, it may be difficult to decide which of the two approaches you should try—to prove the conjecture or to disprove it! A single counterexample to a conjecture is sufficient to disprove it. Of course, merely hunting for a counterexample and being unsuccessful does not constitute a proof that the conjecture is true.

REMINDER

One counterexample is enough to disprove a conjecture.

EXAMPLE 1

For a positive integer n , **n factorial** is defined as $n(n-1)(n-2) \cdots 1$, and is denoted by $n!$. Prove or disprove the conjecture, “For every positive integer n , $n! \leq n^2$.”

Let’s begin by testing some cases:

n	$n!$	n^2	$n! \leq n^2$
1	1	1	yes
2	2	4	yes
3	6	9	yes

So far, this conjecture seems to be looking good. But for the next case,

n	$n!$	n^2	$n! \leq n^2$
4	24	16	no

we have found a counterexample. The fact that the conjecture is true for $n = 1, 2$, and 3 does nothing to prove the conjecture, but the single case $n = 4$ is enough to disprove it. ●

PRACTICE 1 Provide counterexamples to the following conjectures.

- All animals living in the ocean are fish.
- Every integer less than 10 is bigger than 5.

If a counterexample is not forthcoming, perhaps the conjecture is true and we should try to prove it. What techniques can we use to try to do this? For the rest of this section, we'll examine various methods of attacking a proof.

Exhaustive Proof

While “disproof by counterexample” always works, “proof by example” seldom does. The one exception to this situation occurs when the conjecture is an assertion about a finite collection. In this case, the conjecture can be proved true by showing that it is true for each member of the collection. **Proof by exhaustion** means that all possible cases have been exhausted, although it often means that the person doing the proof is exhausted as well!

EXAMPLE 2

Prove the conjecture, “If an integer between 1 and 20 is divisible by 6, then it is also divisible by 3.” (“Divisible by 6,” means, “evenly divisible by 6,” that is, the number is an integral multiple of 6.)

Because there is only a finite number of cases, the conjecture can be proved by simply showing it to be true for all the integers between 1 and 20. Table 2.1 is the proof.

Number	Divisible by 6	Divisible by 3
1	no	
2	no	
3	no	
4	no	
5	no	
6	yes: $6 = 1 \times 6$	yes: $6 = 2 \times 3$
7	no	
8	no	
9	no	
10	no	
11	no	
12	yes: $12 = 2 \times 6$	yes: $12 = 4 \times 3$
13	no	
14	no	
15	no	
16	no	
17	no	
18	yes: $18 = 3 \times 6$	yes: $18 = 6 \times 3$
19	no	
20	no	

EXAMPLE 3

Prove the conjecture, “It is not possible to trace all the lines in Figure 2.1 without lifting your pencil and without retracing any lines.”

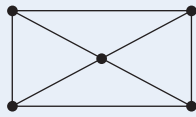


Figure 2.1

There is only a finite number of different ways to trace the lines in the figure. By careful bookkeeping, each of the possibilities can be attempted, and each will fail. In Chapter 7, we will learn a much less tedious way to solve this problem than proof by exhaustion.

PRACTICE 2

- Prove the conjecture “For any positive integer less than or equal to 5, the square of the integer is less than or equal to the sum of 10 plus 5 times the integer.”
- Disprove the conjecture “For any positive integer, the square of the integer is less than or equal to the sum of 10 plus 5 times the integer.”

Direct Proof

In general (where exhaustive proof won't work), how can you prove that $P \rightarrow Q$ is true? The obvious approach is the **direct proof**—assume the hypothesis P and deduce the conclusion Q . A formal proof would require a proof sequence leading from P to Q .

Example 4 shows a formal proof that if two numbers are even (that's the hypothesis P), then their product is even (that's the conclusion Q). Recall that an **even number** is a number that is an integral multiple of 2, for example, 18 is even because $18 = 2(9)$. An **odd number** is 1 more than an integral multiple of 2, for example, $19 = 2(9) + 1$.

EXAMPLE 4

Consider the conjecture

$$x \text{ is an even integer} \wedge y \text{ is an even integer} \rightarrow \text{the product } xy \text{ is an even integer}$$

A complete formal proof sequence might look like the following:

- | | |
|--|---|
| 1. x is an even integer \wedge y is an even integer | hyp |
| 2. $(\forall x)[x \text{ is even integer} \rightarrow$
$(\exists k)(k \text{ an integer} \wedge x = 2k)]$ | number fact (definition
of even integer) |
| 3. x is even integer $\rightarrow (\exists k)(k \text{ an integer} \wedge x = 2k)$ | 2, ui |
| 4. y is even integer $\rightarrow (\exists k)(k \text{ an integer} \wedge y = 2k)$ | 2, ui |
| 5. x is an even integer | 1, sim |
| 6. $(\exists k)(k \text{ is an integer} \wedge x = 2k)$ | 3, 5, mp |
| 7. m is an integer $\wedge x = 2m$ | 6, ei |
| 8. y is an even integer | 1, sim |
| 9. $(\exists k)(k \text{ an integer} \wedge y = 2k)$ | 4, 8, mp |
| 10. n is an integer and $y = 2n$ | 9, ei |
| 11. $x = 2m$ | 7, sim |
| 12. $y = 2n$ | 10, sim |

13. $xy = (2m)(2n)$	11, 12, substitution of equals
14. $xy = 2(2mn)$	13, multiplication fact
15. m is an integer	7, sim
16. n is an integer	10, sim
17. $2mn$ is an integer	15, 16, number fact
18. $xy = 2(2mn) \wedge 2mn$ is an integer	14, 17, con
19. $(\exists k)(k \text{ an integer} \wedge xy = 2k)$	18, eg
20. $(\forall x)((\exists k)(k \text{ an integer} \wedge x = 2k) \rightarrow$ $x \text{ is even integer})$	number fact (definition of even integer)
21. $(\exists k)(k \text{ an integer} \wedge xy = 2k) \rightarrow$ $xy \text{ is even integer}$	20, ui
22. xy is an even integer	19, 21, mp

It is understood that x and y are arbitrary, but this could be stated by expressing the conjecture as

$$(\forall x)(\forall y)(x \text{ is an even integer} \wedge y \text{ is an even integer} \rightarrow \text{the product } xy \text{ is an even integer})$$

Universal generalization can be applied twice to the result we already have in order to put the universal quantifiers on the front. ●

We'll never again do a proof like the one in Example 4, and you won't have to either! A much more informal proof would be perfectly acceptable in most circumstances.

EXAMPLE 5

Following is an informal direct proof that the product of two even integers is even.

Let $x = 2m$ and $y = 2n$, where m and n are integers. Then $xy = (2m)(2n) = 2(2mn)$, where $2mn$ is an integer. Thus xy has the form $2k$, where k is an integer, and xy is therefore even.

Notice that we set $x = 2m$ for some integer m (the definition of an even number), but we set $y = 2n$. In the formal proof of Example 4, the restriction on the use of *ei* required that we use a different multiple of 2 for y than we used for x . Informally, if we were also to set $y = 2m$, we would be saying that x and y are the same integers, which is a very special case. ●

The proof in Example 5 does not explicitly state the hypothesis (that x and y are even), and it makes implicit use of the definition of an even integer. Even in informal proofs, however, it is important to identify the hypothesis and the conclusion, not just what they are in words but what they really mean, by applying appropriate definitions. If we do not clearly understand what we have (the hypothesis) or what we want (the conclusion), we cannot hope to build a bridge from one to the other. That's why it is important to know definitions.

PRACTICE 3 Give a direct proof (informal) of the theorem “If an integer is divisible by 6, then twice that integer is divisible by 4.” ■

Contraposition

If you have tried diligently but failed to produce a direct proof of your conjecture $P \rightarrow Q$, and you still feel that the conjecture is true, you might try some variants on the direct proof technique. If you can prove the theorem $Q' \rightarrow P'$, you can conclude $P \rightarrow Q$ by making use of the tautology $(Q' \rightarrow P') \rightarrow (P \rightarrow Q)$. $Q' \rightarrow P'$ is the **contrapositive** of $P \rightarrow Q$, and the technique of proving $P \rightarrow Q$ by doing a direct proof of $Q' \rightarrow P'$ is called **proof by contraposition**. (The contraposition rule of inference in propositional logic (Table 1.14) says that $P \rightarrow Q$ can be derived from $Q' \rightarrow P'$.)

EXAMPLE 6 Prove that if the square of an integer is odd, then the integer must be odd.

The conjecture is $n^2 \text{ odd} \rightarrow n \text{ odd}$. We do a proof by contraposition, and prove $n \text{ even} \rightarrow n^2 \text{ even}$. Let n be even. Then $n^2 = n(n)$ is even by Example 5. ●

EXAMPLE 7 Prove that if $n + 1$ separate passwords are issued to n students, then some student gets ≥ 2 passwords.

Here the conclusion Q has the form $(\exists x)R(x)$, so Q' is $[(\exists x)R(x)]'$, which is equivalent to $(\forall x)[R(x)]'$. The contrapositive $Q' \rightarrow P'$ is, “If every student gets < 2 passwords, then it is false that $n + 1$ passwords were issued.” Suppose every student has < 2 passwords; then every one of the n students has at most 1 password. The total number of passwords issued is at most n , not $n + 1$, so it is false that $n + 1$ passwords were issued. ●

Example 7 is an illustration of the pigeonhole principle, which we will see in Chapter 4.

PRACTICE 4 Write the contrapositive of each statement in Practice 5 of Chapter 1. ■

Practice 7 of Chapter 1 showed that the wffs $A \rightarrow B$ and $B \rightarrow A$ are not equivalent. $B \rightarrow A$ is the **converse** of $A \rightarrow B$. If an implication is true, its converse may be true or false. Therefore, you cannot prove $P \rightarrow Q$ by looking at $Q \rightarrow P$.

EXAMPLE 8 The implication “If $a > 5$, then $a > 2$ ” is true, but its converse, “If $a > 2$, then $a > 5$,” is false. ●

PRACTICE 5 Write the converse of each statement in Practice 5 of Chapter 1. ■

REMINDER

“If and only if” requires two proofs, one in each direction.

Theorems are often stated in the form “ P if and only if Q ,” meaning P if Q and P only if Q , or $Q \rightarrow P$ and $P \rightarrow Q$. To prove such a theorem, you must prove both an implication and its converse. Again, the truth of one does not imply the truth of the other.

EXAMPLE 9

Prove that the product xy is odd if and only if both x and y are odd integers.

We first prove that if x and y are odd, so is xy . A direct proof will work. Suppose that both x and y are odd. Then $x = 2n + 1$ and $y = 2m + 1$, where m and n are integers. Then $xy = (2n + 1)(2m + 1) = 4nm + 2m + 2n + 1 = 2(2nm + m + n) + 1$. This has the form $2k + 1$, where k is an integer, so xy is odd.

Next we prove that if xy is odd, both x and y must be odd, or

$$xy \text{ odd} \rightarrow x \text{ odd and } y \text{ odd}$$

A direct proof would begin with the hypothesis that xy is odd, which leaves us little more to say. A proof by contraposition works well because we’ll get more useful information as hypotheses. So we will prove

$$(x \text{ odd and } y \text{ odd})' \rightarrow (xy \text{ odd})'$$

By De Morgan’s law $(A \wedge B)' \Leftrightarrow A' \vee B'$, we see that this can be written as

$$x \text{ even or } y \text{ even} \rightarrow xy \text{ even} \tag{1}$$

The hypothesis “ x even or y even” breaks down into three cases. We consider each case in turn.

1. x even, y odd: Here $x = 2m$, $y = 2n + 1$, and then $xy = (2m)(2n + 1) = 2(2mn + m)$, which is even.
2. x odd, y even: This works just like case 1.
3. x even, y even: Then xy is even by Example 5.

This completes the proof of (1) and thus of the theorem. ●

The second part of the proof of Example 9 uses **proof by cases**, a form of exhaustive proof. It involves identifying all the possible cases consistent with the given information and then proving each case separately.

Contradiction

In addition to direct proof and proof by contraposition, you might use the technique of **proof by contradiction**. (Proof by contradiction is sometimes called *indirect proof*, but this term more properly means any argument that is not a direct proof.) As we did in Chapter 1, we will let \perp stand for any contradiction, that is, any wff whose truth value is always false. ($A \wedge A'$ would be such a wff.) Once

more, suppose you are trying to prove $P \rightarrow Q$. By constructing a truth table, we see that

$$(P \wedge Q' \rightarrow 0) \rightarrow (P \rightarrow Q)$$

is a tautology, so to prove the theorem $P \rightarrow Q$, it is sufficient to prove $P \wedge Q' \rightarrow 0$. Therefore, in a proof by contradiction you assume that both the hypothesis and the negation of the conclusion are true and then try to deduce some contradiction from these assumptions.

EXAMPLE 10

Let's use proof by contradiction on the statement, "If a number added to itself gives itself, then the number is 0." Let x represent any number. The hypothesis is $x + x = x$ and the conclusion is $x = 0$. To do a proof by contradiction, assume $x + x = x$ and $x \neq 0$. Then $2x = x$ and $x \neq 0$. Because $x \neq 0$, we can divide both sides of the equation $2x = x$ by x and arrive at the contradiction $2 = 1$. Hence, $(x + x = x) \rightarrow (x = 0)$. ●

REMINDER

To prove that something is not true, try proof by contradiction.

Example 10 notwithstanding, a proof by contradiction most immediately comes to mind when you want to prove that something is *not* true. It's hard to prove that something *is not true*; it's much easier to *assume it is true* and obtain a contradiction.

EXAMPLE 11

A well-known proof by contradiction shows that $\sqrt{2}$ is not a rational number. Recall that a **rational number** is one that can be written in the form p/q where p and q are integers, $q \neq 0$, and p and q have no common factors (other than ± 1).

Let us assume that $\sqrt{2}$ is rational. Then $\sqrt{2} = p/q$, and $2 = p^2/q^2$, or $2q^2 = p^2$. Then 2 divides p^2 , so—because 2 is itself indivisible—2 must divide p . This means that 2 is a factor of p ; hence 4 is a factor of p^2 , and the equation $2q^2 = p^2$ can be written as $2q^2 = 4x$, or $q^2 = 2x$. We see from this equation that 2 divides q^2 and hence 2 divides q . At this point, 2 is a factor of q and a factor of p , which contradicts the statement that p and q have no common factors. Therefore $\sqrt{2}$ is not rational. ●

The proof of Example 11 involves more than just algebraic manipulations. It is often necessary to use lots of words in a proof.

PRACTICE 6

Prove by contradiction that the product of odd integers is not even. (We did a direct proof of an equivalent statement in Example 9.) ■

Proof by contradiction can be a valuable technique, but it is easy to think we have done a proof by contradiction when we really haven't. For example, suppose we assume $P \wedge Q'$ and are able to deduce Q without using the assumption Q' . Then we assert $Q \wedge Q'$ as a contradiction. What really happened here is a direct proof of $P \rightarrow Q$, and the proof should be rewritten in this form. Thus in Example 10, we could assume $x + x = x$ and $x \neq 0$, as before. Then we could argue that from

$x + x = x$ we get $2x = x$ and, after subtracting x from both sides, $x = 0$. We then have $x = 0$ and $x \neq 0$, a contradiction. However, in this argument we never made use of the assumption $x \neq 0$; we actually proved directly that $x + x = x$ implies $x = 0$.

Another misleading claim of proof by contradiction occurs when we assume $P \wedge Q'$ and are able to deduce P' without using the assumption P . Then we assert $P \wedge P'$ as a contradiction. What really happened here is a direct proof of $Q' \rightarrow P'$, and we have constructed a proof by contraposition, not a proof by contradiction. In both this case and the previous one, it is not that the proofs are wrong, just that they are not proofs by contradiction.

Table 2.2 summarizes useful proof techniques we have discussed so far.

Proof Technique	Approach to Prove $P \rightarrow Q$	Remarks
Exhaustive proof	Demonstrate $P \rightarrow Q$ for all possible cases.	May be used only to prove a finite number of cases.
Direct proof	Assume P , deduce Q .	The standard approach—usually the thing to try.
Proof by contraposition	Assume Q' , deduce P' .	Use this if Q' as a hypothesis seems to give more ammunition than P would.
Proof by contradiction	Assume $P \wedge Q'$, deduce a contradiction.	Use this when Q says something is not true.

Serendipity

Serendipity means a fortuitous happening, or good luck. While this isn't really a general proof technique, some of the most interesting proofs come from clever observations that we can admire, even if we would never have thought of them ourselves. We'll look at two such proofs, just for fun.

EXAMPLE 12

A tennis tournament has 342 players. A single match involves 2 players. The winner of a match plays the winner of a match in the next round, while losers are eliminated from the tournament. The 2 players who have won all previous rounds play in the final game, and the winner wins the tournament. Prove that the total number of matches to be played is 341.

The hard way to prove this result is to compute $342/2 = 171$ to get the number of matches in the first round, resulting in 171 winners to go on to the second round. For the second round, $171/2 = 85$ plus 1 left over; there are 85 matches and 85 winners, plus the 1 left over, to go on to the third round. The third round has $86/2 = 43$ matches, and so forth. The total number of matches is the sum of $171 + 85 + 43 + \dots$.

The clever observation is to note that each match results in exactly 1 loser, so there must be the same number of matches as losers in the tournament. Because there is only 1 winner, there are 341 losers and therefore 341 matches. ●

EXAMPLE 13

A standard 64-square checkerboard is arranged in 8 rows of 8 squares each. Adjacent squares are alternating colors of red and black. A set of 32 1×2 tiles, each covering 2 squares, will cover the board completely (4 tiles per row, 8 rows). Prove that if the squares at diagonally opposite corners of the checkerboard are removed, the remaining board cannot be covered with 31 tiles.

The hard way to prove this result is to try all possibilities with 31 tiles and see that they all fail. The clever observation is to note that opposing corners are the same color, so the checkerboard with the corners removed has two less squares of one color than of the other. Each tile covers one square of each color, so any set of tiles must cover an equal number of squares of each color and cannot cover the board with the corners removed. ●

Common Definitions

Many of the examples in this section and many of the exercises that follow involve elementary **number theory**, that is, results about integers. It's useful to work in number theory when first starting to construct proofs because many properties of integers, such as what it means to be an even number, are already familiar. The following definitions may be helpful in working some of these exercises.

- A **perfect square** is an integer n such that $n = k^2$ for some integer k .
- A **prime number** is an integer $n > 1$ such that n is not divisible by any integers other than 1 and n .
- A **composite number** n is a nonprime integer; that is, $n = ab$ where a and b are integers with $1 < a < n$ and $1 < b < n$.
- For two numbers x and y , $x < y$ means $y - x > 0$.
- For two integers n and m , n **divides** m , $n | m$, means that m is divisible by n —that is, $m = k(n)$ for some integer k .
- The **absolute value** of a number x , $|x|$, is x if $x \geq 0$ and is $-x$ if $x < 0$.

SECTION 2.1 REVIEW

TECHNIQUES

- Look for a counterexample.
- Construct direct proofs, proofs by contraposition, and proofs by contradiction.

MAIN IDEAS

- Inductive reasoning is used to formulate a conjecture based on experience. Deductive reasoning

is used either to refute a conjecture by finding a counterexample or to prove a conjecture.

- In proving a conjecture about some subject, facts about that subject can be used.
- Under the right circumstances, proof by contraposition or contradiction may work better than a direct proof.

EXERCISES 2.1

1. Write the contrapositive of each statement in Exercise 5 of Section 1.1.
2. Write the converse of each statement in Exercise 5 of Section 1.1.

3. Provide counterexamples to the following statements.
 - a. Every geometric figure with four right angles is a square.
 - b. If a real number is not positive, then it must be negative.
 - c. All people with red hair have green eyes or are tall.
 - d. All people with red hair have green eyes and are tall.
4. Provide counterexamples to the following statements.
 - a. If a and b are integers where $a|b$ and $b|a$, then $a = b$.
 - b. If $n^2 > 0$ then $n > 0$.
 - c. If n is an even number, then $n^2 + 1$ is prime.
 - d. If n is a positive integer, then $n^3 > n!$.
5. Provide a counterexample to the following statement: The number n is an odd integer if and only if $3n + 5$ is an even integer.
6. Provide a counterexample to the following statement: The number n is an even integer if and only if $3n + 2$ is an even integer.
7.
 - a. Find two even integers whose sum is not a multiple of 4.
 - b. What is wrong with the following “proof” that the sum of two even numbers is a multiple of 4?
Let x and y be even numbers. Then $x = 2m$ and $y = 2m$, where m is an integer, so $x + y = 2m + 2m = 4m$, which is an integral multiple of 4.
8.
 - a. Find an example of an odd number x and an even number y such that $x - y = 7$.
 - b. What is wrong with the following “proof” that an odd number minus an even number is always 1?
Let x be odd and y be even. Then $x = 2m + 1$, $y = 2m$, where m is an integer, and $x - y = 2m + 1 - 2m = 1$.

For Exercises 9–46, prove the given statement.

9. If $n = 25, 100,$ or 169 , then n is a perfect square and is a sum of two perfect squares.
10. If n is an even integer, $4 \leq n \leq 12$, then n is a sum of two prime numbers.
11. For any positive integer n less than or equal to 3, $n! < 2^n$.
12. For $2 \leq n \leq 4$, $n^2 \geq 2^n$.
13. The sum of two even integers is even (do a direct proof).
14. The sum of two even integers is even (do a proof by contradiction).
15. The sum of two odd integers is even.
16. The sum of an even integer and an odd integer is odd.
17. An odd integer minus an even integer is odd.
18. If n is an even integer, then $n^2 - 1$ is odd.
19. The product of any two consecutive integers is even.
20. The sum of an integer and its square is even.
21. The square of an even number is divisible by 4.
22. For every integer n , the number $3(n^2 + 2n + 3) - 2n^2$ is a perfect square.
23. If a number x is positive, so is $x + 1$ (do a proof by contraposition).
24. If n is an odd integer, then it is the difference of two perfect squares.
25. The number n is an odd integer if and only if $3n + 5 = 6k + 8$ for some integer k .
26. The number n is an even integer if and only if $3n + 2 = 6k + 2$ for some integer k .

27. For x and y positive numbers, $x < y$ if and only if $x^2 < y^2$.
28. If $x^2 + 2x - 3 = 0$, then $x \neq 2$.
29. If n is an even prime number, then $n = 2$.
30. The sum of three consecutive integers is divisible by 3.
31. If two integers are each divisible by some integer n , then their sum is divisible by n .
32. If the product of two integers is not divisible by an integer n , then neither integer is divisible by n .
33. If n , m , and p are integers and $n \mid m$ and $m \mid p$, then $n \mid p$.
34. If n , m , p , and q are integers and $n \mid p$ and $m \mid q$, then $nm \mid pq$.
35. The square of an odd integer equals $8k + 1$ for some integer k .
36. The sum of the squares of two odd integers cannot be a perfect square. (*Hint*: Use Exercise 35.)
37. The product of the squares of two integers is a perfect square.
38. The difference of two consecutive cubes is odd.
39. For any two numbers x and y , $|x + y| \leq |x| + |y|$.
40. For any two numbers x and y , $|xy| = |x||y|$.
41. The value A is the average of the n numbers x_1, x_2, \dots, x_n . Prove that at least one of x_1, x_2, \dots, x_n is greater than or equal to A .
42. Suppose you were to use the steps of Example 11 to attempt to prove that $\sqrt{4}$ is not a rational number. At what point would the proof not be valid?
43. Prove that $\sqrt{3}$ is not a rational number.
44. Prove that $\sqrt{5}$ is not a rational number.
45. Prove that $\sqrt[3]{2}$ is not a rational number.
46. Prove that $\log_2 5$ is not a rational number ($\log_2 5 = x$ means $2^x = 5$).

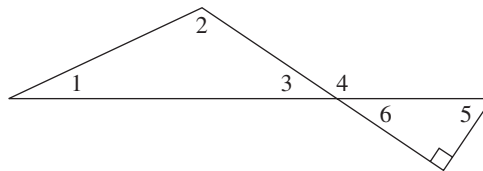
For Exercises 47–72, prove or disprove the given statement.

47. 0 is an even number.
48. 91 is a composite number.
49. 297 is a composite number.
50. 83 is a composite number.
51. The difference between two odd integers is odd.
52. The difference between two even integers is even.
53. The product of any three consecutive integers is even.
54. The sum of any three consecutive integers is even.
55. The sum of an integer and its cube is even.
56. The number n is an even integer if and only if $n^3 + 13$ is odd.
57. The product of an integer and its square is even.
58. Any positive integer can be written as the sum of the squares of two integers.
59. The sum of the square of an odd integer and the square of an even integer is odd.
60. If n is a positive integer that is a perfect square, then $n + 2$ is not a perfect square.
61. For a positive integer n , $n + \frac{1}{n} \geq 2$.

62. If n , m , and p are integers and $n \mid mp$, then $n \mid m$ or $n \mid p$.
63. For every prime number n , $n + 4$ is prime.
64. For every positive integer n , $n^2 + n + 3$ is not prime.
65. For n a positive integer, $n > 2$, $n^2 - 1$ is not prime.
66. For every positive integer n , $2^n + 1$ is prime.
67. For every positive integer n , $n^2 + n + 1$ is prime.
68. For n an even integer, $n > 2$, $2^n - 1$ is not prime.
69. The sum of two rational numbers is rational.
70. The product of two rational numbers is rational.
71. The product of two irrational numbers is irrational.
72. The sum of a rational number and an irrational number is irrational.

For Exercises 73–75, use the following facts from geometry and the accompanying figure.

- The interior angles of a triangle sum to 180° .
- Vertical angles (opposite angles formed when two lines intersect) are the same size.
- A straight angle is 180° .
- A right angle is 90° .



73. Prove that the measure of angle 5 plus the measure of angle 3 is 90° .
74. Prove that the measure of angle 4 is the sum of the measures of angles 1 and 2.
75. If angle 1 and angle 5 are the same size, then angle 2 is a right angle.
76. Prove that the sum of the integers from 1 through 100 is 5050. (*Hint:* Instead of actually adding all the numbers, try to make the same clever observation that the German mathematician Karl Friederick Gauss [1777–1855] made as a schoolchild: Group the numbers into pairs, using 1 and 100, 2 and 99, etc.)

SECTION 2.2 INDUCTION

First Principle of Induction

There is one final proof technique especially useful in computer science. To illustrate how the technique works, imagine that you are climbing an infinitely high ladder. How do you know whether you will be able to reach an arbitrarily high rung? Suppose we make the following two assertions about your climbing abilities:

1. You can reach the first rung.
2. Once you get to a rung, you can always climb to the next one up. (Notice that this assertion is an implication.)

If both statement 1 and the implication of statement 2 are true, then by statement 1 you can get to the first rung and therefore by statement 2 you can get to the second; by statement 2 again, you can get to the third rung; by statement 2 again you can get to the fourth; and so on. You can climb as high as you wish. Both assertions here are necessary. If only statement 1 is true, you have no guarantee of getting beyond the first rung, and if only statement 2 is true, you may never be able to get started. Let's assume that the rungs of the ladder are numbered by positive integers—1, 2, 3, and so on.

Now think of a specific property a number might have. Instead of “reaching an arbitrarily high rung,” we can talk about an arbitrary, positive integer having that property. We use the shorthand notation $P(n)$ to mean that the positive integer n has the property P . How can we use the ladder-climbing technique to prove that for all positive integers n , we have $P(n)$? The two assertions we need to prove are

1. $P(1)$ (1 has property P .)
2. For any positive integer k , $P(k) \rightarrow P(k + 1)$. (If any number has property P , so does the next number.)

If we can prove both assertions 1 and 2, then $P(n)$ holds for any positive integer n , just as you could climb to an arbitrary rung on the ladder.

The foundation for arguments of this type is the first principle of mathematical induction.

● PRINCIPLE FIRST PRINCIPLE OF MATHEMATICAL INDUCTION

1. $P(1)$ is true
 2. $(\forall k)[P(k) \text{ true} \rightarrow P(k + 1) \text{ true}]$
- $$\left. \vphantom{\begin{matrix} 1. \\ 2. \end{matrix}} \right\} \rightarrow P(n) \text{ true for all positive integers } n$$

REMINDER

To prove something true for all $n \geq$ some value, think induction.

The first principle of mathematical induction is an implication. The conclusion is a statement of the form, “ $P(n)$ is true for all positive integers n .” Therefore, whenever we want to prove that something is true for every positive integer n , it is a good bet that mathematical induction is an appropriate proof technique to use.

To know that the conclusion of this implication is true, we show that the two hypotheses, statements 1 and 2, are true. To prove statement 1, we need only show that property P holds for the number 1, usually a trivial task. Statement 2 is also an implication that must hold for all k . To prove this implication, we assume for an arbitrary positive integer k that $P(k)$ is true and show, based on this assumption, that $P(k + 1)$ is true. Therefore $P(k) \rightarrow P(k + 1)$ and, using universal generalization, $(\forall k)[P(k) \rightarrow P(k + 1)]$. You should convince yourself that assuming that property P holds for the number k is not the same as assuming what we ultimately want to prove (a frequent source of confusion when one first encounters proofs of this kind). It is merely the way to proceed with a direct proof that the implication $P(k) \rightarrow P(k + 1)$ is true.

In doing a proof by induction, establishing the truth of statement 1, $P(1)$, is called the **basis**, or **basis step**, for the inductive proof. Establishing the truth of $P(k) \rightarrow P(k + 1)$ is called the **inductive step**. When we assume $P(k)$ to be true

so as to prove the inductive step, $P(k)$ is called the **inductive assumption**, or **inductive hypothesis**.

All the proof methods we have talked about in this chapter are techniques for deductive reasoning—ways to prove a conjecture that perhaps was formulated by inductive reasoning. Mathematical induction is also a *deductive* technique, not a method for inductive reasoning (don't get confused by the terminology here). For the other proof techniques, we can begin with a hypothesis and string facts together until we more or less stumble on a conclusion. In fact, even if our conjecture is slightly incorrect, we might see what the correct conclusion is in the course of doing the proof. In mathematical induction, however, we must know right at the outset the exact form of the property $P(n)$ that we are trying to establish. Mathematical induction, therefore, is not an exploratory proof technique—it can only confirm a correct conjecture.

Proofs by Mathematical Induction

Suppose that the ancestral progenitor Smith married and had two children. Let's call these two children generation 1. Now suppose each of those two children had two children; then in generation 2, there were four offspring. This trend continued from generation unto generation. The Smith family tree therefore looks like Figure 2.2. (This figure looks exactly like Figure 1.1b, where we looked at the possible T–F values for n statement letters.)

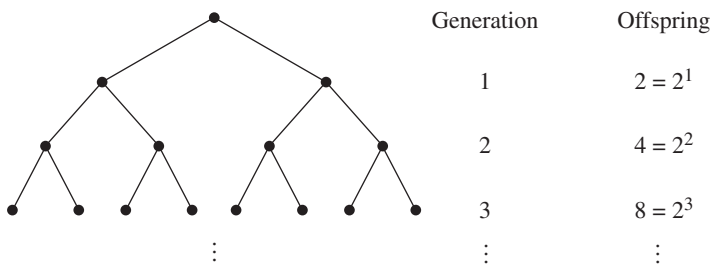


Figure 2.2

It appears that generation n contains 2^n offspring. More formally, if we let $P(n)$ denote the number of offspring at generation n , then we guess that

$$P(n) = 2^n$$

We can use induction to *prove* that our guess for $P(n)$ is correct.

The basis step is to establish $P(1)$, which is the equation

$$P(1) = 2^1 = 2$$

This is true because we are told that Smith had two children. We now assume that our guess is correct for an arbitrary generation k , $k \geq 1$; that is, we assume

$$P(k) = 2^k$$

and try to show that

$$P(k + 1) = 2^{k+1}$$

In this family, each offspring has two children; thus the number of offspring at generation $k + 1$ will be twice the number at generation k , or $P(k + 1) = 2P(k)$. By the inductive assumption, $P(k) = 2^k$, so

$$P(k + 1) = 2P(k) = 2(2^k) = 2^{k+1}$$

and so indeed

$$P(k + 1) = 2^{k+1}$$

This completes our proof. Now that we have set our mind at ease about the Smith clan, we can apply the inductive proof technique to less obvious problems.

EXAMPLE 14

Prove that the equation

$$1 + 3 + 5 + \cdots + (2n - 1) = n^2 \quad (1)$$

is true for any positive integer n . Here the property $P(n)$ is equation (1). (Notice that $P(n)$ is a property of n , or—in language from Chapter 1—a unary predicate. It is a statement about n , expressed here as an equation. Thus it is incorrect to write something like $P(n) = 1 + 3 + 5 + \cdots + (2n - 1)$.)

The left side of this equation is the sum of all the odd integers from 1 to $2n - 1$. The right side is a formula for the value of this sum. Although we can verify the truth of this equation for any particular value of n by substituting that value for n , we cannot substitute all possible positive integer values. Thus a proof by exhaustion does not work. A proof by mathematical induction is appropriate.

The basis step is to establish $P(1)$, which is equation (1) when n has the value 1. When we substitute 1 for n in the left side of equation (1), we get the sum of all the odd integers starting at 1 and ending at $2(1) - 1 = 1$. The sum of all the odd numbers from 1 to 1 equals 1. When we substitute 1 for n in the formula on the right side of this equation, we get $(1)^2$. Therefore

$$P(1): \quad 1 = 1^2$$

This is certainly true. For the inductive hypothesis, we assume $P(k)$ for an arbitrary positive integer k , which is equation (1) when n has the value k , or

$$P(k): \quad 1 + 3 + 5 + \cdots + (2k - 1) = k^2 \quad (2)$$

(Note that $P(k)$ is *not* the equation $(2k - 1) = k^2$, which is true only for $k = 1$.) Using the inductive hypothesis, we want to show $P(k + 1)$, which is equation (1) when n has the value $k + 1$, or

$$P(k + 1): \quad 1 + 3 + 5 + \cdots + [2(k + 1) - 1] \stackrel{?}{=} (k + 1)^2 \quad (3)$$

(The question mark over the equals sign is to remind us that this is the fact we want to prove, as opposed to something we already know.)

The key to an inductive proof is to find a way to relate what we want to show— $P(k + 1)$, equation (3)—to what we have assumed— $P(k)$, equation (2). The left side of $P(k + 1)$ can be rewritten to show the next-to-last term:

$$1 + 3 + 5 + \cdots + (2k - 1) + [2(k + 1) - 1]$$

This expression contains the left side of equation (2) as a subexpression. Because we have assumed $P(k)$ to be true, we can substitute the right side of equation (2) for this subexpression. Thus,

$$\begin{aligned} 1 + 3 + 5 + \cdots + [2(k + 1) - 1] &= 1 + 3 + 5 + \cdots + (2k - 1) + [2(k + 1) - 1] \\ &= k^2 + [2(k + 1) - 1] \\ &= k^2 + [2k + 2 - 1] \\ &= k^2 + 2k + 1 \\ &= (k + 1)^2 \end{aligned}$$

Therefore,

$$1 + 3 + 5 + \cdots + [2(k + 1) - 1] = (k + 1)^2$$

which verifies $P(k + 1)$ and proves that equation (1) is true for any positive integer n . ●

Table 2.3 summarizes the three steps necessary for a proof using the first principle of induction.

TABLE 2.3	
To Prove by First Principle of Induction	
Step 1	Prove base case.
Step 2	Assume $P(k)$.
Step 3	Prove $P(k + 1)$.

Any “summation” induction problem works exactly the same way. Write the summation including the next-to-last term, and you will find the left side of the $P(k)$ equation and can use the inductive hypothesis. After that, it’s just a question of algebra.

EXAMPLE 15

Prove that

$$1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$$

for any $n \geq 1$.

Again, induction is appropriate. $P(1)$ is the equation

$$1 + 2 = 2^{1+1} - 1 \quad \text{or} \quad 3 = 2^2 - 1$$

REMINDER

To prove $P(k) \rightarrow P(k + 1)$, you have to discover the $P(k)$ case within the $P(k + 1)$ case.

which is true. We take $P(k)$

$$1 + 2 + 2^2 + \cdots + 2^k = 2^{k+1} - 1$$

as the inductive hypothesis and try to establish $P(k + 1)$:

$$1 + 2 + 2^2 + \cdots + 2^{k+1} \stackrel{?}{=} 2^{k+1+1} - 1$$

Rewriting the sum on the left side of $P(k + 1)$ reveals how the inductive assumption can be used:

$$\begin{aligned} 1 + 2 + 2^2 + \cdots + 2^{k+1} &= 1 + 2 + 2^2 + \cdots + 2^k + 2^{k+1} \\ &= 2^{k+1} - 1 + 2^{k+1} && \text{(from the inductive hypothesis } P(k)) \\ &= 2(2^{k+1}) - 1 && \text{(adding like terms)} \\ &= 2^{k+1+1} - 1 \end{aligned}$$

Therefore,

$$1 + 2 + 2^2 + \cdots + 2^{k+1} = 2^{k+1+1} - 1$$

which verifies $P(k + 1)$ and completes the proof. ●

The following summation formula is the most important one because it occurs so often in the analysis of algorithms. If you don't remember anything else, you should remember this formula for the sum of the first n positive integers.

PRACTICE 7 Prove that for any positive integer n ,

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

Not all proofs by induction involve formulas with sums. Other algebraic identities about the positive integers, as well as nonalgebraic assertions like the number of offspring in generation n of the Smith family, can be proved by induction.

EXAMPLE 16

Prove that for any positive integer n , $2^n > n$.

$P(1)$ is the assertion $2^1 > 1$, which is surely true. Now we assume $P(k)$, $2^k > k$, and try to conclude $P(k + 1)$, $2^{k+1} > k + 1$. Now, where is $P(k)$ hidden in here? Aha—we can write the left side of $P(k + 1)$, 2^{k+1} , as $2k \cdot 2$, and there's the left side of $P(k)$. Using the inductive hypothesis $2^k > k$ and multiplying both sides of this inequality by 2, we get $2^k \cdot 2 > k \cdot 2$. We complete the argument

$$2^{k+1} = 2^k \cdot 2 > k \cdot 2 = k + k \geq k + 1 \text{ (because } k \geq 1)$$

or

$$2^{k+1} > k + 1$$



For the first step of the induction process, it may be appropriate to begin at 0 or at 2 or 3 instead of at 1. The same principle applies, no matter where you first hop on the ladder.

EXAMPLE 17

Prove that $n^2 > 3n$ for $n \geq 4$.

Here we should use induction and begin with a basis step of $P(4)$. (Testing values of $n = 1, 2$, and 3 shows that the inequality does not hold for these values.) $P(4)$ is the inequality $4^2 > 3(4)$, or $16 > 12$, which is true. The inductive hypothesis is that $k^2 > 3k$ and that $k \geq 4$, and we want to show that $(k + 1)^2 > 3(k + 1)$.

$$\begin{aligned} (k + 1)^2 &= k^2 + 2k + 1 \\ &> 3k + 2k + 1 && \text{(by the inductive hypothesis)} \\ &\geq 3k + 8 + 1 && \text{(because } k \geq 4\text{)} \\ &= 3k + 9 \\ &> 3k + 3 && \text{(because } 9 > 3\text{)} \\ &= 3(k + 1) \end{aligned}$$

In this proof we used the fact that $3k + 9 > 3k + 3$. Of course, $3k + 9$ is greater than lots of things, but $3k + 3$ is what gives us what we want. In an induction proof, because we know exactly what we want as the result, we can let that guide us as we manipulate algebraic expressions. ●

PRACTICE 8

Prove that for all $n > 1$,

$$2^{n+1} < 3^n$$

EXAMPLE 18

Prove that for any positive integer n , the number $2^{2^n} - 1$ is divisible by 3.

The basis step is to show $P(1)$, that $2^{2(1)} - 1 = 4 - 1 = 3$ is divisible by 3. Clearly this is true.

We assume that $2^{2^k} - 1$ is divisible by 3, which means that $2^{2^k} - 1 = 3m$ for some integer m , or $2^{2^k} = 3m + 1$ (this little rewriting trick is the key to these “divisibility” problems). We want to show that $2^{2^{(k+1)}} - 1$ is divisible by 3.

$$\begin{aligned} 2^{2^{(k+1)}} - 1 &= 2^{2^k + 2} - 1 \\ &= 2^2 \cdot 2^{2^k} - 1 \\ &= 2^2(3m + 1) - 1 && \text{(by the inductive hypothesis)} \\ &= 12m + 4 - 1 \\ &= 12m + 3 \\ &= 3(4m + 1) && \text{where } 4m + 1 \text{ is an integer} \end{aligned}$$

Thus $2^{2^{(k+1)}} - 1$ is divisible by 3. ●

Misleading claims of proof by induction are also possible. When we prove the truth of $P(k + 1)$ without relying on the truth of $P(k)$, we have done a direct proof of $P(k + 1)$ where $k + 1$ is arbitrary. The proof is not invalid, but it

should be rewritten to show that it is a direct proof of $P(n)$ for any n , not a proof by induction.

An inductive proof may be called for when its application is not as obvious as in the above examples. The problem statement may not directly say “prove something about nonnegative integers.” Instead, there is some quantity in the statement to be proved that can take on arbitrary nonnegative integer values.

EXAMPLE 19

A programming language might be designed with the following convention regarding multiplication: A single factor requires no parentheses, but the product “ a times b ” must be written as $(a)b$. So the product

$$a \cdot b \cdot c \cdot d \cdot e \cdot f \cdot g$$

could be written in this language as

$$((((((a)b)c)d)e)f)g$$

or, for example,

$$((a)b)((c)d)(e)f)g$$

depending on the order in which the products are formed. The result is the same in either case.

We want to show that any product of factors can be written with an even number of parentheses. The proof is by induction on the number of factors (this is where the nonnegative integer comes in—it represents the number of factors in any product of factors). For a single factor, there are 0 parentheses, an even number. Assume that for any product of k factors there is an even number of parentheses. Now consider a product P of $k + 1$ factors. P can be thought of as r times s where r has k factors and s is a single factor. By the inductive hypothesis, r has an even number of parentheses. Then we write r times s as $(r)s$. This adds 2 more parentheses to the even number of parentheses in r , giving P an even number of parentheses. ●

Here’s an important observation about the proof in Example 19: There are no algebraic expressions! The entire proof is a verbal argument. For some proofs, we can’t rely on just the crutch of nonverbal mathematical manipulations; we have to use words.

EXAMPLE 20

A “tiling” problem gives a nice illustration of induction in a geometric setting. An *angle iron* is an L-shaped piece that can cover 3 squares on a checkerboard (Figure 2.3a). The problem is to show that for any positive integer n , a $2^n \times 2^n$ checkerboard with one square removed can be tiled—completely covered—by angle irons.

The base case is $n = 1$, which gives a 2×2 checkerboard. Figure 2.3b shows the solution to this case if the upper right corner is removed. Removing any of the other three corners works the same way. Assume that any $2^k \times 2^k$ checkerboard with one square removed can be tiled using angle irons. Now consider a checkerboard with

dimensions $2^{k+1} \times 2^{k+1}$. We need to show that it can be tiled when one square is removed. To relate the $k + 1$ case to the inductive hypothesis, divide the $2^{k+1} \times 2^{k+1}$ checkerboard into four quarters. Each quarter will be a $2^k \times 2^k$ checkerboard, and one will have a missing square (Figure 2.3c). By the inductive hypothesis, this checkerboard can be tiled. Remove a corner from each of the other three checkerboards, as in Figure 2.3d. By the inductive hypothesis, the three boards with the holes removed can be tiled, and one angle iron can tile the three holes. Hence the original $2^{k+1} \times 2^{k+1}$ board with its one hole can be tiled.

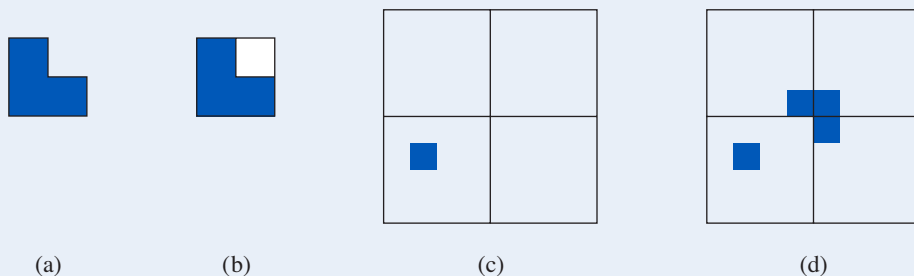


Figure 2.3

Second Principle of Induction

In addition to the first principle of induction, which we have been using,

1. $P(1)$ is true
 2. $(\forall k)[P(k) \text{ true} \rightarrow P(k + 1) \text{ true}]$
- $$\left. \vphantom{\begin{matrix} 1. \\ 2. \end{matrix}} \right\} \rightarrow P(n) \text{ true for all positive integers } n$$

there is a second principle of induction.

PRINCIPLE SECOND PRINCIPLE OF MATHEMATICAL INDUCTION

- 1'. $P(1)$ is true
 - 2'. $(\forall k)[P(r) \text{ true for all } r, 1 \leq r \leq k \rightarrow P(k + 1) \text{ true}]$
- $$\left. \vphantom{\begin{matrix} 1'. \\ 2'. \end{matrix}} \right\} \rightarrow P(n) \text{ true for all positive integers } n$$

These two induction principles differ in statements 2 and 2'. In statement 2, we must be able to prove for an arbitrary positive integer k that $P(k + 1)$ is true based only on the assumption that $P(k)$ is true. In statement 2', we can assume that $P(r)$ is true for all integers r between 1 and an arbitrary positive integer k in order to prove that $P(k + 1)$ is true. This seems to give us a great deal more “ammunition,” so we might sometimes be able to prove the implication in 2' when we cannot prove the implication in 2.

What allows us to deduce $(\forall n)P(n)$ in either case? We will see that the two induction principles themselves, that is, the two methods of proof, are equivalent. In other words, if we accept the first principle of induction as valid, then the second principle of induction is valid, and conversely. In order to prove the equivalence of the two induction principles, we'll introduce another principle, which seems so obvious as to be unarguable.

- PRINCIPLE** **PRINCIPLE OF WELL-ORDERING**
 Every collection of positive integers that contains any members at all has a smallest member.

We will see that the following implications are true:

second principle of induction \rightarrow first principle of induction
 first principle of induction \rightarrow well-ordering
 well-ordering \rightarrow second principle of induction

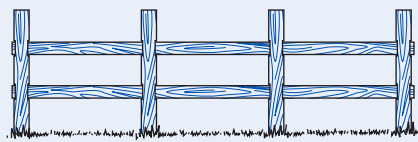
As a consequence, all three principles are equivalent, and accepting any one of them as true means accepting the other two as well.

To prove that the second principle of induction implies the first principle of induction, suppose we accept the second principle as valid reasoning. We then want to show that the first principle is valid; that is, that we can conclude $P(n)$ for all n from statements 1 and 2. If statement 1 is true, so is statement 1'. If statement 2 is true, then so is statement 2', because we can say that we concluded $P(k + 1)$ from $P(r)$ for all r between 1 and k , even though we used only the single condition $P(k)$. (More precisely, statement 2' requires that we prove $P(1) \wedge P(2) \wedge \cdots \wedge P(k) \rightarrow P(k + 1)$, but $P(1) \wedge P(2) \wedge \cdots \wedge P(k) \rightarrow P(k)$, and from statement 2, $P(k) \rightarrow P(k + 1)$, so $P(1) \wedge P(2) \wedge \cdots \wedge P(k) \rightarrow P(k + 1)$.) By the second principle of induction, we conclude $P(n)$ for all n . The proofs that the first principle of induction implies well-ordering and that well-ordering implies the second principle of induction are left as exercises in Section 4.1.

To distinguish between a proof by the first principle of induction and a proof by the second principle of induction, let's look at a rather picturesque example that can be proved both ways.

EXAMPLE 21

Prove that a straight fence with n fence posts has $n - 1$ sections for any $n \geq 1$ (as in Figure 2.4a).



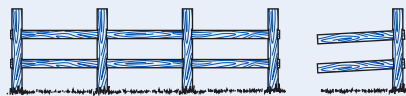
Fence with 4 fenceposts, 3 sections

(a)



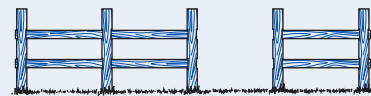
Fence with 1 fencepost, 0 sections

(b)



Fence with last post and last section removed

(c)



Fence with 1 section removed

(d)

Figure 2.4

Let $P(n)$ be the statement that a fence with n fence posts has $n - 1$ sections, and prove $P(n)$ true for all $n \geq 1$.

We'll start with the first principle of induction. For the basis step, $P(1)$ says that a fence with only 1 fence post has 0 sections, which is clearly true (Figure 2.4b). Assume that $P(k)$ is true:

a fence with k fence posts has $k - 1$ sections

and try to prove $P(k + 1)$:

(?) a fence with $k + 1$ fence posts has k sections

Given a fence with $k + 1$ fence posts, how can we relate that to a fence with k fence posts so that we can make use of the inductive hypothesis? We can chop off the last post and the last section (Figure 2.4c). The remaining fence has k fence posts and, by the inductive hypothesis, $k - 1$ sections. Therefore the original fence had k sections.

Now we'll prove the same result using the second principle of induction. The basis step is the same as before. For the inductive hypothesis, we assume

for all r , $1 \leq r \leq k$, a fence with r fence posts has $r - 1$ sections

and try to prove $P(k + 1)$:

(?) a fence with $k + 1$ fence posts has k sections

For a fence with $k + 1$ fence posts, split the fence into two parts by removing one section (Figure 2.4d). The 2 parts of the fence have r_1 and r_2 fence posts, where $1 \leq r_1 \leq k$, $1 \leq r_2 \leq k$, and $r_1 + r_2 = k + 1$. By the inductive hypothesis, the 2 parts have, respectively, $r_1 - 1$ and $r_2 - 1$ sections, so the original fence has

$$(r_1 - 1) + (r_2 - 1) + 1 \text{ sections}$$

(The extra 1 is for the one that we removed.) Simple arithmetic then yields

$$r_1 + r_2 - 1 = (k + 1) - 1 = k \text{ sections}$$

This proves that a fence with $k + 1$ fence posts has k sections, which verifies $P(k + 1)$ and completes the proof using the second principle of induction. ●

Example 21 allowed for either form of inductive proof because we could either reduce the fence at one end or split it at an arbitrary point. The problem of Example 19 is similar.

EXAMPLE 22

We again want to show that any product of factors can be written in this programming language with an even number of parentheses, this time using the second principle of induction. The base case is the same as in Example 19: A single factor has 0 parentheses, an even number. Assume that any product of r factors, $1 \leq r \leq k$, can be written with an even number of parentheses. Then consider a product

P with $k + 1$ factors. P can be written as $(S)T$, a product of two factors S and T , where S has r_1 factors and T has r_2 factors. Then $1 \leq r_1 \leq k$ and $1 \leq r_2 \leq k$, with $r_1 + r_2 = k + 1$. By the inductive hypothesis, S and T each have an even number of parentheses, and therefore so does $(S)T = P$. ●

Most problems do not work equally well with either form of induction; the fence post and the programming language problem were somewhat artificial. Generally, the second principle of induction is called for when the problem “splits” most naturally in the middle instead of growing from the end.

EXAMPLE 23

Prove that for every integer $n \geq 2$, n is a prime number or a product of prime numbers.

We will postpone the decision of whether to use the first or the second principle of induction; the basis step is the same in each case and need not start with 1. Obviously here we should start with 2. $P(2)$ is the statement that 2 is a prime number or a product of primes. Because 2 is a prime number, $P(2)$ is true. Jumping ahead, for either principle we will be considering the number $k + 1$. If $k + 1$ is prime, we are done. If $k + 1$ is not prime, then it is a composite number and can be written as $k + 1 = ab$. Here $k + 1$ has been split into two factors. Maybe neither of these factors has the value k , so an assumption only about $P(k)$ isn't enough. Hence, we'll use the second principle of induction.

So let's start again. We assume that for all r , $2 \leq r \leq k$, $P(r)$ is true— r is prime or the product of primes. Now consider the number $k + 1$. If $k + 1$ is prime, we are done. If $k + 1$ is not prime, then it is a composite number and can be written as $k + 1 = ab$, where $1 < a < k + 1$ and $1 < b < k + 1$. (This is a nontrivial factorization, so neither factor can be 1 or $k + 1$.) Therefore $2 \leq a \leq k$ and $2 \leq b \leq k$. The inductive hypothesis applies to both a and b , so a and b are either prime or the product of primes. Thus, $k + 1 = ab$ is the product of prime numbers. This verifies $P(k + 1)$ and completes the proof by the second principle of induction. ●

The proof in Example 23 is an *existence* proof rather than a *constructive* proof. Knowing that every nonprime number has a factorization as a product of primes does not make it easy to *find* such a factorization. (We will see in Section 2.4 that there is, except for the order of the factors, only one such factorization.) Some encryption systems for passing information in a secure fashion on the Web depend on the difficulty of factoring large numbers into their prime factors (see the discussion on public-key encryption in Section 5.6).

EXAMPLE 24

Prove that any amount of postage greater than or equal to 8 cents can be built using only 3-cent and 5-cent stamps.

Here we let $P(n)$ be the statement that only 3-cent and 5-cent stamps are needed to build n cents worth of postage, and prove that $P(n)$ is true for all $n \geq 8$. The basis step is to establish $P(8)$, which is done by the equation

$$8 = 3 + 5$$

For reasons that will be clear momentarily, we'll also establish two additional cases, $P(9)$ and $P(10)$, by the equations

$$\begin{aligned}9 &= 3 + 3 + 3 \\10 &= 5 + 5\end{aligned}$$

Now we assume that $P(r)$ is true, that is, r can be written as a sum of $3s$ and $5s$, for any r , $8 \leq r \leq k$, and consider $P(k + 1)$. We may assume that $k + 1$ is at least 11, because we have already proved $P(r)$ true for $r = 8, 9$, and 10. If $k + 1 \geq 11$, then $(k + 1) - 3 = k - 2 \geq 8$. Thus $k - 2$ is a legitimate r value, and by the inductive hypothesis, $P(k - 2)$ is true. Therefore $k - 2$ can be written as a sum of $3s$ and $5s$, and adding an additional 3 gives us $k + 1$ as a sum of $3s$ and $5s$. This verifies that $P(k + 1)$ is true and completes the proof. ●

PRACTICE 9

- Why are the additional cases $P(9)$ and $P(10)$ proved separately in Example 24?
- Why can't the first principle of induction be used in the proof of Example 24? ■

REMINDER

Use the second principle of induction when the $k + 1$ case depends on results farther back than k .

As a general rule, the first principle of mathematical induction applies when information about “one position back” is enough, that is, when the truth of $P(k)$ is enough to prove the truth of $P(k + 1)$. The second principle applies when information about “one position back” isn't good enough; that is, you can't prove that $P(k + 1)$ is true just because you know $P(k)$ is true, but you can prove $P(k + 1)$ true if you know that $P(r)$ is true for one or more values of r that are “farther back” than k .

SECTION 2.2 REVIEW

TECHNIQUES

- Ⓜ Use the first principle of induction in proofs.
- Ⓜ Use the second principle of induction in proofs.

MAIN IDEAS

- Mathematical induction is a technique to prove properties of positive integers.

- An inductive proof need not begin with 1.
- Induction can be used to prove statements about quantities whose values are arbitrary nonnegative integers.
- The first and second principles of induction each prove the same conclusion, but one approach may be easier to use than the other in a given situation.

EXERCISES 2.2

- For all positive integers, let $P(n)$ be the equation

$$2 + 6 + 10 + \cdots + (4n - 2) = 2n^2$$

- Write the equation for the base case $P(1)$ and verify that it is true.
- Write the inductive hypothesis $P(k)$.
- Write the equation for $P(k + 1)$.
- Prove that $P(k + 1)$ is true.

2. For all positive integers, let $P(n)$ be the equation

$$2 + 4 + 6 + \cdots + 2n = n(n + 1)$$

- Write the equation for the base case $P(1)$ and verify that it is true.
- Write the inductive hypothesis $P(k)$.
- Write the equation for $P(k + 1)$.
- Prove that $P(k + 1)$ is true.

In Exercises 3–26, use mathematical induction to prove that the statements are true for every positive integer n . [Hint: In the algebra part of the proof, if the final expression you want has factors and you can pull those factors out early, do that instead of multiplying everything out and getting some humongous expression.]

3. $1 + 5 + 9 + \cdots + (4n - 3) = n(2n - 1)$

4. $1 + 3 + 6 + \cdots + \frac{n(n + 1)}{2} = \frac{n(n + 1)(n + 2)}{6}$

5. $4 + 10 + 16 + \cdots + (6n - 2) = n(3n + 1)$

6. $5 + 10 + 15 + \cdots + 5n = \frac{5n(n + 1)}{2}$

7. $1^2 + 2^2 + \cdots + n^2 = \frac{n(n + 1)(2n + 1)}{6}$

8. $1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n + 1)^2}{4}$

9. $1^2 + 3^2 + \cdots + (2n - 1)^2 = \frac{n(2n - 1)(2n + 1)}{3}$

10. $1^4 + 2^4 + \cdots + n^4 = \frac{n(n + 1)(2n + 1)(3n^2 + 3n - 1)}{30}$

11. $1 \cdot 3 + 2 \cdot 4 + 3 \cdot 5 + \cdots + n(n + 2) = \frac{n(n + 1)(2n + 7)}{6}$

12. $1 + a + a^2 + \cdots + a^{n-1} = \frac{a^n - 1}{a - 1}$ for $a \neq 0, a \neq 1$

13. $\frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \cdots + \frac{1}{n(n + 1)} = \frac{n}{n + 1}$

14. $\frac{1}{1 \cdot 3} + \frac{1}{3 \cdot 5} + \frac{1}{5 \cdot 7} + \cdots + \frac{1}{(2n - 1)(2n + 1)} = \frac{n}{2n + 1}$

15. $1^2 - 2^2 + 3^2 - 4^2 + \cdots + (-1)^{n+1}n^2 = \frac{(-1)^{n+1}(n)(n + 1)}{2}$

16. $2 + 6 + 18 + \cdots + 2 \cdot 3^{n-1} = 3^n - 1$

17. $2^2 + 4^2 + \cdots + (2n)^2 = \frac{2n(n + 1)(2n + 1)}{3}$

18. $1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + n \cdot 2^n = (n - 1)2^{n+1} + 2$

19. $1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \cdots + n(n + 1) = \frac{n(n + 1)(n + 2)}{3}$

$$20. 1 \cdot 2 \cdot 3 + 2 \cdot 3 \cdot 4 + \cdots + n(n+1)(n+2) = \frac{n(n+1)(n+2)(n+3)}{4}$$

$$21. \frac{1}{1 \cdot 4} + \frac{1}{4 \cdot 7} + \frac{1}{7 \cdot 10} + \cdots + \frac{1}{(3n-2)(3n+1)} = \frac{n}{3n+1}$$

$$22. 1 \cdot 1! + 2 \cdot 2! + 3 \cdot 3! + \cdots + n \cdot n! = (n+1)! - 1 \text{ where } n! \text{ is the product of the positive integers from 1 to } n.$$

$$23. 1 + 4 + 4^2 + \cdots + 4^n = \frac{4^{n+1} - 1}{3}$$

$$24. 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1} \text{ where } x \text{ is any integer } > 1$$

$$25. 1 + 4 + 7 + 10 + \cdots + (3n-2) = \frac{n(3n-1)}{2}$$

$$26. 1 + [x \cdot 2 - (x-1)] + [x \cdot 3 - (x-1)] + \cdots + [x \cdot n - (x-1)] = \frac{n[xn - (x-2)]}{2}$$

where x is any integer ≥ 1

27. A *geometric progression* (*geometric sequence*) is a sequence of terms where there is an initial term a and each succeeding term is obtained by multiplying the previous term by a *common ratio* r . Prove the formula for the sum of the first n ($n \geq 1$) terms of a geometric sequence where $r \neq 1$:

$$a + ar + ar^2 + \cdots + ar^{n-1} = \frac{a - ar^n}{1 - r}$$

28. An *arithmetic progression* (*arithmetic sequence*) is a sequence of terms where there is an initial term a and each succeeding term is obtained by adding a *common difference* d to the previous term. Prove the formula for the sum of the first n ($n \geq 1$) terms of an arithmetic sequence:

$$a + (a+d) + (a+2d) + \cdots + [a + (n-1)d] = \frac{n}{2}[2a + (n-1)d]$$

29. Using Exercises 27 and 28, find an expression for the value of the following sums.

a. $2 + 2 \cdot 5 + 2 \cdot 5^2 + \cdots + 2 \cdot 5^9$

b. $4 \cdot 7 + 4 \cdot 7^2 + 4 \cdot 7^3 + \cdots + 4 \cdot 7^{12}$

c. $1 + 7 + 13 + \cdots + 49$

d. $12 + 17 + 22 + 27 + \cdots + 92$

30. Prove that

$$(-2)^0 + (-2)^1 + (-2)^2 + \cdots + (-2)^n = \frac{1 - 2^{n+1}}{3}$$

for every positive odd integer n .

31. Prove that $n^2 > n + 1$ for $n \geq 2$.

32. Prove that $n^2 \geq 2n + 3$ for $n \geq 3$.

33. Prove that $n^2 > 5n + 10$ for $n > 6$.

34. Prove that $2^n > n^2$ for $n \geq 5$.

In Exercises 35–40, $n!$ is the product of the positive integers from 1 to n .

35. Prove that $n! > n^2$ for $n \geq 4$.

36. Prove that $n! > n^3$ for $n \geq 6$.
 37. Prove that $n! > 2^n$ for $n \geq 4$.
 38. Prove that $n! > 3^n$ for $n \geq 7$.
 39. Prove that $n! \geq 2^{n-1}$ for $n \geq 1$.
 40. Prove that $n! < n^n$ for $n \geq 2$.
 41. Prove that $(1+x)^n > 1+x^n$ for $n > 1, x > 0$.
 42. Prove that $\left(\frac{a}{b}\right)^{n+1} < \left(\frac{a}{b}\right)^n$ for $n \geq 1$ and $0 < a < b$.
 43. Prove that $1 + 2 + \cdots + n < n^2$ for $n > 1$.
 44. Prove that $1 + \frac{1}{4} + \frac{1}{9} + \cdots + \frac{1}{n^2} < 2 - \frac{1}{n}$ for $n \geq 2$.
 45. a. Try to use induction to prove that

$$1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^n} < 2 \quad \text{for } n \geq 1$$

What goes wrong?

- b. Prove that

$$1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^n} = 2 - \frac{1}{2^n} \quad \text{for } n \geq 1$$

thus showing that

$$1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^n} < 2 \quad \text{for } n \geq 1$$

46. Prove that

$$1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{2^n} \geq 1 + \frac{n}{2} \quad \text{for } n \geq 1$$

(Note that the denominators increase by 1, not by powers of 2.)

For Exercises 47–58, prove that the statements are true for every positive integer.

47. $2^{3n} - 1$ is divisible by 7.
 48. $3^{2n} + 7$ is divisible by 8.
 49. $7^n - 2^n$ is divisible by 5.
 50. $13^n - 6^n$ is divisible by 7.
 51. $2^n + (-1)^{n+1}$ is divisible by 3.
 52. $2^{5n+1} + 5^{n+2}$ is divisible by 27.
 53. $3^{4n+2} + 5^{2n+1}$ is divisible by 14.
 54. $7^{2n} + 16n - 1$ is divisible by 64.
 55. $10^n + 3 \cdot 4^{n+2} + 5$ is divisible by 9.
 56. $n^3 - n$ is divisible by 3.
 57. $n^3 + 2n$ is divisible by 3.
 58. $x^n - 1$ is divisible by $x - 1$ for $x \neq 1$.

59. Prove DeMoivre's Theorem:

$$(\cos \theta + i \sin \theta)^n = \cos n\theta + i \sin n\theta$$

for all $n \geq 1$. *Hint:* Recall the addition formulas from trigonometry:

$$\cos(\alpha + \beta) = \cos \alpha \cos \beta - \sin \alpha \sin \beta$$

$$\sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta$$

60. Prove that

$$\sin \theta + \sin 3\theta + \cdots + \sin(2n - 1)\theta = \frac{\sin^2 n\theta}{\sin \theta}$$

for all $n \geq 1$ and all θ for which $\sin \theta \neq 0$.

61. Use induction to prove that the product of any three consecutive positive integers is divisible by 3.

62. Suppose that exponentiation is defined by the equation

$$x^j \cdot x = x^{j+1}$$

for any $j \geq 1$. Use induction to prove that $x^n \cdot x^m = x^{n+m}$ for $n \geq 1, m \geq 1$.

(*Hint:* Do induction on m for a fixed, arbitrary value of n .)

63. According to Example 20, it is possible to use angle irons to tile a 4×4 checkerboard with the upper right corner removed. Sketch such a tiling.

64. Example 20 does not cover the case of checkerboards that are not sized by powers of 2. Determine whether it is possible to tile a 3×3 checkerboard.

65. Prove that it is possible to use angle irons to tile a 5×5 checkerboard with the upper left corner removed.

66. Find a configuration for a 5×5 checkerboard with one square removed that is not possible to tile; explain why this is not possible.

67. Consider n infinitely long straight lines, none of which are parallel and no three of which have a common point of intersection. Show that for $n \geq 1$, the lines divide the plane into $(n^2 + n + 2)/2$ separate regions.

68. A string of 0s and 1s is to be processed and converted to an even-parity string by adding a parity bit to the end of the string. (For an explanation of the use of parity bits, see Example 30 in Chapter 9.) The parity bit is initially 0. When a 0 character is processed, the parity bit remains unchanged. When a 1 character is processed, the parity bit is switched from 0 to 1 or from 1 to 0. Prove that the number of 1s in the final string, that is, including the parity bit, is always even. (*Hint:* Consider various cases.)

69. What is wrong with the following "proof" by mathematical induction? We will prove that for any positive integer n , n is equal to 1 more than n . Assume that $P(k)$ is true.

$$k = k + 1$$

Adding 1 to both sides of this equation, we get

$$k + 1 = k + 2$$

Thus,

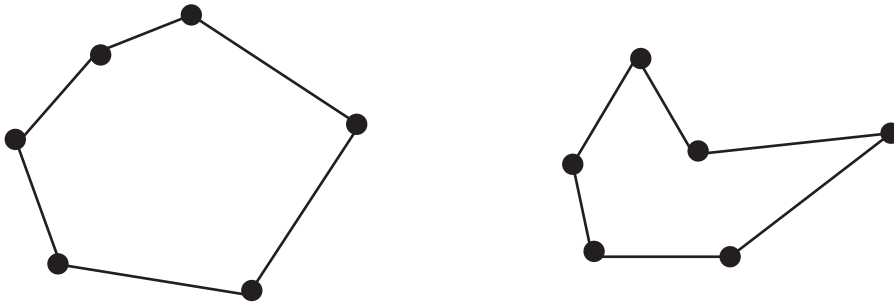
$$P(k + 1) \text{ is true}$$

70. What is wrong with the following "proof" by mathematical induction?

We will prove that all computers are built by the same manufacturer. In particular, we will prove that in any collection of n computers where n is a positive integer, all the computers are built by the same manufacturer. We first prove $P(1)$, a trivial process, because in any collection consisting of

one computer, there is only one manufacturer. Now we assume $P(k)$; that is, in any collection of k computers, all the computers were built by the same manufacturer. To prove $P(k + 1)$, we consider any collection of $k + 1$ computers. Pull one of these $k + 1$ computers (call it HAL) out of the collection. By our assumption, the remaining k computers all have the same manufacturer. Let HAL change places with one of these k computers. In the new group of k computers, all have the same manufacturer. Thus, HAL's manufacturer is the same one that produced all the other computers, and all $k + 1$ computers have the same manufacturer.

71. An obscure tribe has only three words in its language, *moon*, *noon*, and *soon*. New words are composed by juxtaposing these words in any order, as in *soonnoonmoonnoon*. Any such juxtaposition is a legal word.
- Use the first principle of induction (on the number of subwords in the word) to prove that any word in this language has an even number of o's.
 - Use the second principle of induction (on the number of subwords in the word) to prove that any word in this language has an even number of o's.
72. A *simple closed polygon* consists of n points in the plane joined in pairs by n line segments; each point is the endpoint of exactly 2 line segments. Following are two examples.



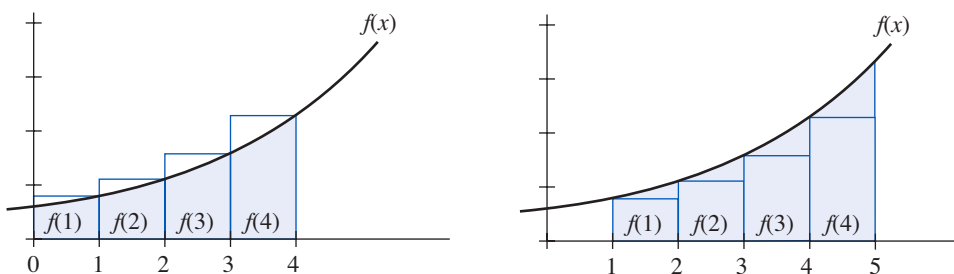
- Use the first principle of induction to prove that the sum of the interior angles of an n -sided simple closed polygon is $(n - 2)180^\circ$ for all $n \geq 3$.
 - Use the second principle of induction to prove that the sum of the interior angles of an n -sided simple closed polygon is $(n - 2)180^\circ$ for all $n \geq 3$.
73. The Computer Science club is sponsoring a jigsaw puzzle contest. Jigsaw puzzles are assembled by fitting 2 pieces together to form a small block, adding a single piece to a block to form a bigger block, or fitting 2 blocks together. Each of these moves is considered a step in the solution. Use the second principle of induction to prove that the number of steps required to assemble an n -piece jigsaw puzzle is $n - 1$.
74. OurWay Pizza makes only two kinds of pizza, pepperoni and vegetarian. Any pizza of either kind comes with an even number of breadsticks (not necessarily the same even number for both kinds). Any order of 2 or more pizzas must include at least 1 of each kind. When the delivery driver goes to deliver an order, he or she puts the completed order together by combining 2 suborders—picking up all the pepperoni pizzas from 1 window and all the vegetarian pizzas from another window. Prove that for a delivery of n pizzas, $n \geq 1$, there are an even number of breadsticks included.
75. Consider propositional wffs that contain only the connectives \wedge , \vee , and \rightarrow (no negation) and where wffs must be parenthesized when joined by a logical connective. Count each statement letter, connective, or parenthesis as one symbol. For example, $((A) \wedge (B)) \vee ((C) \wedge (D))$ is such a wff, with 19 symbols. Prove that any such wff has an odd number of symbols.
76. In any group of k people, $k \geq 1$, each person is to shake hands with every other person. Find a formula for the number of handshakes, and prove the formula using induction.

77. Prove that any amount of postage greater than or equal to 2 cents can be built using only 2-cent and 3-cent stamps.
78. Prove that any amount of postage greater than or equal to 12 cents can be built using only 4-cent and 5-cent stamps.
79. Prove that any amount of postage greater than or equal to 14 cents can be built using only 3-cent and 8-cent stamps.
80. Prove that any amount of postage greater than or equal to 42 cents can be built using only 4-cent and 15-cent stamps.
81. Prove that any amount of postage greater than or equal to 64 cents can be built using only 5-cent and 17-cent stamps.
82. Your bank ATM delivers cash using only \$20 and \$50 bills. Prove that you can collect, in addition to \$20, any multiple of \$10 that is \$40 or greater.

Exercises 83–84 require familiarity with ideas from calculus. Exercises 1–26 give exact formulas for the sum of terms in a sequence that can be expressed as $\sum_{m=1}^n f(m)$. Sometimes it is difficult to find an exact expression for this summation, but if the value of $f(m)$ increases monotonically, integration can be used to find upper and lower bounds on the value of the summation. Specifically,

$$\int_0^n f(x) dx \leq \sum_{m=1}^n f(m) \leq \int_1^{n+1} f(x) dx$$

Using the following figure, we can see (on the left) that $\int_0^n f(x) dx$ underestimates the value of the summation while (on the right) $\int_1^{n+1} f(x) dx$ overestimates it.



83. Show that $\int_0^n 2x dx \leq \sum_{m=1}^n 2m \leq \int_1^{n+1} 2x dx$ (see Exercise 2).

84. Show that $\int_0^n x^2 dx \leq \sum_{m=1}^n m^2 \leq \int_1^{n+1} x^2 dx$ (see Exercise 7).

SECTION 2.3 MORE ON PROOF OF CORRECTNESS

In Section 1.6, we explained the use of a formal logic system to prove mathematically the correctness of a program. Assertions or predicates involving the program variables are inserted at the beginning, at the end, and at intermediate points between the program statements. Then proving the correctness of any particular program statement s_i involves proving that the implication represented by the Hoare triple

$$\{Q\} s_i \{R\} \quad (1)$$

is true. Here Q and R are assertions known, respectively, as the precondition and postcondition for the statement. The program is provably correct if all such implications for the statements in the program are true.

In Chapter 1, we discussed rules of inference that give conditions under which implication (1) is true when s_i is an assignment statement and when s_i is a conditional statement. Now we will use a rule of inference that gives conditions under which implication (1) is true when s_i is a loop statement. We have deferred consideration of loop statements until now because mathematical induction is used in applying this rule of inference.

Loop Rule

Suppose that s_i is a loop statement in the form

```
while condition  $B$  do
   $P$ 
end while
```

where B is a condition that is either true or false and P is a program segment. When this statement is executed, condition B is evaluated. If B is true, program segment P is executed and then B is evaluated again. If B is still true, program segment P is executed again, then B is evaluated again, and so forth. If condition B ever evaluates to false, the loop terminates.

The form of implication (1) that can be used when s_i is a loop statement imposes (like the assignment rule did) a relationship between the precondition and the postcondition. The precondition Q holds before the loop is entered; strangely enough, one requirement is that Q must continue to hold after the loop terminates (which means that we should look for a Q that we want to be true when the loop terminates). In addition, B' —the condition for loop termination—must be true then as well. Thus (1) will have the form

$$\{Q\} s_i \{Q \wedge B'\} \quad (2)$$

EXAMPLE 25

Consider the following pseudocode function, which is supposed to return the value $x * y$ for nonnegative integers x and y .

Product (nonnegative integer x ; nonnegative integer y)

Local variables:

integers i, j

$i = 0$

$j = 0$

while $i \neq x$ **do**

$j = j + y$

$i = i + 1$

end while

 // j now has the value $x * y$

 return j

end function *Product*

This function contains a loop; the condition B for continued loop execution is $i \neq x$. The condition B' for loop termination is therefore $i = x$. When the loop terminates, it is claimed in the comment that j has the value $x * y$. Thus, on loop termination, we want

$$Q \wedge B' = Q \wedge (i = x)$$

and we also want

$$j = x * y$$

To have both

$$Q \wedge (i = x) \text{ and } j = x * y$$

Q must be the assertion

$$j = i * y$$

(Notice that Q is a predicate, that is, it states a relationship between variables in the program. It is never part of an equation such as $Q = j$.) To match the form of (2), the assertion $j = i * y$ would have to be true before the loop statement. This is indeed the case because right before the loop statement, $i = j = 0$.

It would seem that for this example we have a candidate assertion Q for implication (2), but we do not yet have the rule of inference that allows us to say when (2) is a true implication. (Remember that we discovered our Q by “wishful thinking” about the correct operation of the function code.) ●

Assertion Q must be true before the loop is entered. If implication (2) is to hold, Q must remain true after the loop terminates. Because it may not be known exactly when the loop will terminate, Q must remain true after each iteration through the loop, which will include the final iteration. Q represents a predicate, or relation, among the values of the program variables. If this relation holds among the values of the program variables before a loop iteration executes and holds among the values after the iteration executes, then the *relation* among these variables is unaffected by the action of the loop iteration, even though the values themselves may be changed. Such a relation is called a **loop invariant**.

The **loop rule of inference** allows the truth of (2) to be inferred from an implication stating that Q is a loop invariant. Again, for Q to be a loop invariant it must be the case that if Q is true and condition B is true, so that another loop iteration is executed, then Q remains true after that iteration, which can be expressed by the Hoare triple $\{Q \wedge B\} P \{Q\}$. The rule is formally stated in Table 2.4.

From	Can Derive	Name of Rule	Restrictions on Use
$\{Q \wedge B\} P \{Q\}$	$\{Q\} s_i \{Q \wedge B'\}$	loop	s_i has the form while condition B do P end while

To use this rule of inference, we must find a *useful* loop invariant Q —one that asserts what we want and expect to have happen—and then prove the implication

$$\{Q \wedge B\} P \{Q\}$$

Here is where induction comes into play. We denote by $Q(n)$ the statement that a proposed loop invariant Q is true after n iterations of the loop. Because we do not necessarily know how many iterations the loop may execute (that is, how long condition B remains true), we want to show that $Q(n)$ is true for all $n \geq 0$. (The value of $n = 0$ corresponds to the assertion upon entering the loop, after zero loop iterations.)

EXAMPLE 26

Consider again the pseudocode function of Example 25. In that example, we guessed that Q is the relation

$$j = i * y$$

To use the loop rule of inference, we must prove that Q is a loop invariant.

The quantities x and y remain unchanged throughout the function, but values of i and j change within the loop. We let i_n and j_n denote the values of i and j , respectively, after n iterations of the loop. Then $Q(n)$ is the statement $j_n = i_n * y$.

We prove by induction that $Q(n)$ holds for all $n \geq 0$. $Q(0)$ is the statement

$$j_0 = i_0 * y$$

which, as we noted in Example 25, is true, because after zero iterations of the loop, when we first get to the loop statement, both i and j have been assigned the value 0. (Formally, the assignment rule could be used to prove that these conditions on i and j hold at this point.)

Assume $Q(k)$: $j_k = i_k * y$

Show $Q(k + 1)$: $j_{k+1} = i_{k+1} * y$

Between the time j and i have the values j_k and i_k and the time they have the values j_{k+1} and i_{k+1} , one iteration of the loop takes place. In that iteration, j is changed by adding y to the previous value, and i is changed by adding 1. Thus,

$$j_{k+1} = j_k + y \quad (3)$$

$$i_{k+1} = i_k + 1 \quad (4)$$

Then

$$\begin{aligned} j_{k+1} &= j_k + y && \text{(by (3))} \\ &= i_k * y + y && \text{(by the inductive hypothesis)} \\ &= (i_k + 1)y \\ &= i_{k+1} * y && \text{(by (4))} \end{aligned}$$

We have proved that Q is a loop invariant.

The loop rule of inference allows us to infer that after the loop statement is exited, the condition $Q \wedge B'$ holds, which in this case becomes

$$j = i * y \wedge i = x$$

Therefore at this point the statement

$$j = x * y$$

is true, which is exactly what the function is intended to compute. ●

Example 26 illustrates that loop invariants say something stronger about the program than we actually want to show; what we want to show is the special case of the loop invariant on termination of the loop. Finding the appropriate loop invariant requires working backward from the desired conclusion, as in Example 25.

We did not, in fact, prove that the loop in this example actually does terminate. What we proved was **partial correctness**—the program produces the correct answer, given that execution does terminate. Because x is a nonnegative integer and i is an integer that starts at 0 and is then incremented by 1 at each pass through the loop, we know that eventually $i = x$ will become true.

PRACTICE 10 Show that the following function returns the value $x + y$ for nonnegative integers x and y by proving the loop invariant $Q: j = x + i$ and evaluating Q when the loop terminates.

```
Sum (nonnegative integer  $x$ ; nonnegative integer  $y$ )
Local variables:
integers  $i, j$ 
   $i = 0$ 
   $j = x$ 
  while  $i \neq y$  do
     $j = j + 1$ 
     $i = i + 1$ 
  end while
  //  $j$  now has the value  $x + y$ 
  return  $j$ 
end function Sum
```

■

The two functions of Example 25 and Practice 10 are somewhat unrealistic; after all, if we wanted to compute $x * y$ or $x + y$, we could no doubt do it with a single program statement. However, the same techniques apply to more meaningful computations, such as the Euclidean algorithm.

Euclidean Algorithm

The **Euclidean algorithm** was described by the Greek mathematician Euclid over 2300 years ago, although it may have been known even earlier. At any rate, it is one of the oldest known algorithms. This algorithm finds the greatest common divisor of two positive integers a and b with $a > b$. The **greatest common divisor** of a and b , denoted by $\text{gcd}(a, b)$, is the largest integer n such that $n | a$ and $n | b$. For example, $\text{gcd}(12, 18)$ is 6 and $\text{gcd}(420, 66) = 6$.

First, let's dispense with two trivial cases of the $\text{gcd}(a, b)$ that do not require the Euclidean algorithm.

- i. $\text{gcd}(a, a) = a$. Clearly $a | a$ and no larger integer divides a .
- ii. $\text{gcd}(a, 0) = a$. Again, $a | a$ and no larger integer divides a , but also $a | 0$ because 0 is a multiple of a : $0 = 0(a)$

The Euclidean algorithm works by a succession of divisions. To find $\text{gcd}(a, b)$, assuming that $a > b$, you first divide a by b , getting a quotient and a remainder. More formally, at this point $a = q_1b + r_1$, where $0 \leq r_1 < b$. Next you divide the divisor, b , by the remainder, r_1 , getting $b = q_2r_1 + r_2$, where $0 \leq r_2 < r_1$. Again divide the divisor, r_1 , by the remainder, r_2 , getting $r_1 = q_3r_2 + r_3$, where $0 \leq r_3 < r_2$. Clearly, there is a looping process going on, with the remainders getting successively smaller. The process terminates when the remainder is 0, at which point the greatest common divisor is the last divisor used.

EXAMPLE 27

To find $\text{gcd}(420, 66)$ the following divisions are performed:

$$\begin{array}{r} 6 \\ 66 \overline{)420} \\ \underline{396} \\ 24 \end{array} \qquad \begin{array}{r} 2 \\ 24 \overline{)66} \\ \underline{48} \\ 18 \end{array} \qquad \begin{array}{r} 1 \\ 18 \overline{)24} \\ \underline{18} \\ 6 \end{array} \qquad \begin{array}{r} 3 \\ 6 \overline{)18} \\ \underline{18} \\ 0 \end{array}$$

The answer is 6, the divisor used when the remainder became 0. ●

A pseudocode version of the algorithm follows, given in the form of a function to return $\text{gcd}(a, b)$.

ALGORITHM *EUCLIDEAN ALGORITHM*

```
GCD (positive integer  $a$ ; positive integer  $b$ )
//  $a > b$ 
Local variables:
integers  $i, j$ 
     $i = a$ 
     $j = b$ 
```

```

while  $j \neq 0$  do
  compute  $i = qj + r, 0 \leq r < j$ 
   $i = j$ 
   $j = r$ 
end while
//  $i$  now has the value  $\gcd(a, b)$ 
return  $i$ ;
end function GCD

```

We intend to prove the correctness of this function, but we will need one additional fact first, namely,

$$(\forall \text{ integers } a, b, q, r)[(a = qb + r) \rightarrow (\gcd(a, b) = \gcd(b, r))] \quad (5)$$

To prove (5), assume that $a = qb + r$ and suppose that c divides both a and b so that $a = q_1c$ and $b = q_2c$. Then

$$r = a - qb = q_1c - qq_2c = c(q_1 - qq_2)$$

so that c divides r as well. Therefore anything that divides a and b also divides b and r . Now suppose d divides both b and r so that $b = q_3d$ and $r = q_4d$. Then

$$a = qb + r = qq_3d + q_4d = d(qq_3 + q_4)$$

so that d divides a as well. Therefore anything that divides b and r also divides a and b . Because (a, b) and (b, r) have identical divisors, they must have the same greatest common divisor.

EXAMPLE 28

Prove the correctness of the Euclidean algorithm.

Using function *GCD*, we will prove the loop invariant Q : $\gcd(i, j) = \gcd(a, b)$ and evaluate Q when the loop terminates. We use induction to prove $Q(n)$: $\gcd(i_n, j_n) = \gcd(a, b)$ for all $n \geq 0$. $Q(0)$ is the statement

$$\gcd(i_0, j_0) = \gcd(a, b)$$

which is true because when we first get to the loop statement, i and j have the values a and b , respectively.

Assume $Q(k)$: $\gcd(i_k, j_k) = \gcd(a, b)$

Show $Q(k + 1)$: $\gcd(i_{k+1}, j_{k+1}) = \gcd(a, b)$

By the assignment statements within the loop body, we know that

$$\begin{aligned} i_{k+1} &= j_k \\ j_{k+1} &= r_k \end{aligned}$$

Then

$$\begin{aligned} \gcd(i_{k+1}, j_{k+1}) &= \gcd(j_k, r_k) \\ &= \gcd(i_k, j_k) && \text{by (5)} \\ &= \gcd(a, b) && \text{by the inductive hypothesis} \end{aligned}$$

Q is therefore a loop invariant. At loop termination, $\gcd(i, j) = \gcd(a, b)$ and $j = 0$, so $\gcd(i, 0) = \gcd(a, b)$. But $\gcd(i, 0)$ is i , so $i = \gcd(a, b)$. Therefore function GCD is correct. ●

Making Safer Software

Proof of correctness seeks to verify that a given computer program or segment of a program meets its specifications. As we have seen, this approach relies on formal logic to prove that if a certain relationship (the precondition) holds among the program variables before a given statement is executed, then after execution another relationship (the postcondition) holds. Because of the labor-intensive nature of proof of correctness, its use is typically reserved for critical sections of code in important applications.

The B method is a set of tools that does two things:

1. Supports formal project specification by means of an abstract model of the system to be developed. This support includes both automatic generation of lemmas that must be proven in order to guarantee that the model reflects the system requirements and automatic proof tools to prove each lemma or flag it for human verification assistance.
2. Translates the abstract model into a code-ready design, again using lemmas to ensure that the design matches the abstract model. The final level can then be translated into code, often using the Ada programming language, described by its proponents as “the language designed for building systems that really matter.”

One of the most interesting applications of proof of correctness, based on the B method, is the development of software for the Paris Météor train. This is part of the Paris metro train system designed to carry up to 40,000 passengers per hour and per direction with an interval between trains as low as 85 seconds on peak hours. The safety-critical part of the software includes the running and stopping of every train, opening and

closing of doors, electrical traction power, routes, speed of trains, and alarms from passengers. By the end of the project, 27,800 lemmas had been proven, with 92% proven automatically (with no human intervention). But here is the amazing part: the number of bugs in the Ada code found by testing on the host computer, the target computer, on site, and after the system was put into operation was—0. Zero, nada, none. Very impressive indeed.

Other formal method systems have been used for critical software projects, such as

- Development of a left ventricular assist device that helps the heart pump blood in those with congestive heart failure. The eventual goal is an artificial heart.
- “Conflict detection and resolution algorithms” for safety in air traffic control
- Development of the Tokeneer ID Station software to perform biometric verification of a human seeking access to a secure computing environment. Tokeneer is a hypothetical system promoted by NSA (National Security Agency) as a challenge problem for security researchers.


Formal Verification of Large Software Systems, Yin, X., and Knight, J., Proceedings of the NASA Formal Methods Symposium, April 13–15, 2010, Washington D.C., USA.

<http://libre.adacore.com/academia/projects-single/echo>
<http://shemesh.larc.nasa.gov/fm/fm-atm-cdr.html>

“Météor: A Successful Application of B in a Large Project,” Behm, P., Benoit, P., Faivre, A., and Meynadier, J., *World Congress On Formal Methods in the Development of Computing Systems*, Toulouse, France, 1999, vol. 1709, pp. 369–387.

SECTION 2.3 REVIEW

TECHNIQUES

-  Verify the correctness of a program segment that includes a loop statement.
- Compute $\text{gcd}(a, b)$ using Euclid's algorithm.

MAIN IDEAS

- A loop invariant, proved by induction on the number of loop iterations, can be used to prove correctness of a program loop.
- The classic Euclidean algorithm for finding the greatest common divisor of two positive integers is provably correct.

EXERCISES 2.3

1. Let $Q: x^2 > x + 1$ where x is a positive integer. Assume that Q is true after the k th iteration of the following while loop; prove that Q is true after the next iteration.

```

while ( $x > 5$ ) and ( $x < 40$ ) do
   $x = x + 1$ 
end while

```

2. Let $Q: x! > 3^x$ where x is a positive integer. Assume that Q is true after the k th iteration of the following while loop; prove that Q is true after the next iteration.

```

while ( $x > 10$ ) and ( $x < 30$ ) do
   $x = x + 2$ 
end while

```

In Exercises 3–6, prove that the pseudocode program segment is correct by proving the given loop invariant Q and evaluating Q at loop termination.

3. Function to return the value of $x!$ for $x \geq 1$.

```

Factorial (positive integer  $x$ )
Local variables:
integers  $i, j$ 
   $i = 2$ 
   $j = 1$ 
  while  $i \neq x + 1$  do
     $j = j * i$ 
     $i = i + 1$ 
  end while
  //  $j$  now has the value  $x!$ 
  return  $j$ 
end function Factorial

```

$$Q: j = (i - 1)!$$

4. Function to return the value of x^2 for $x \geq 1$.

```

Square (positive integer  $x$ )
Local variables:
integers  $i, j$ 
   $i = 1$ 
   $j = 1$ 
  while  $i \neq x$  do
     $j = j + 2i + 1$ 
     $i = i + 1$ 
  end while

```

```

//j now has the value  $x^2$ 
return j
end function Square

```

$$Q: j = i^2$$

5. Function to return the value of x^y for $x, y \geq 1$.

```

Power (positive integer  $x$ , positive integer  $y$ )
Local variables:
integers  $i, j$ 
   $i = 1$ 
   $j = x$ 
  while  $i \neq y$  do
     $j = j * x$ 
     $i = i + 1$ 
  end while
//j now has the value  $x^y$ 
return j
end function Power

```

$$Q: j = x^i$$

6. Function to compute and write out quotient q and remainder r when x is divided by y , $x \geq 0, y \geq 1$.

```

Divide (nonnegative integer  $x$ ; positive integer  $y$ );
Local variables:
nonnegative integers  $q, r$ 
   $q = 0$ 
   $r = x$ 
  while  $r \geq y$  do
     $q = q + 1$ 
     $r = r - y$ 
  end while
// $q$  and  $r$  are now the quotient and remainder
write("The quotient is"  $q$  "and the remainder is"  $r$ )
end function Divide

```

$$Q: x = q * y + r$$

For Exercises 7–12 use the Euclidean algorithm to find the greatest common divisor of the given numbers.

7. (308, 165)
8. (2420, 70)
9. (735, 90)
10. (8370, 465)
11. (1326, 252)
12. (1018215, 2695)
13. Following is the problem posed at the beginning of this chapter.

The nonprofit organization at which you volunteer has received donations of 792 bars of soap and 400 bottles of shampoo. You want to create packages to distribute to homeless shelters such that each package contains the same number of shampoo bottles and each package contains the same number of bars of soap. How many packages can you create?

Explain why the solution to this problem is the $\text{gcd}(792, 400)$.

14. Concerning the question posed in Exercise 13:
- Use the Euclidean algorithm to find the number of packages.
 - How many bottles of shampoo are in each package?
 - How many bars of soap are in each package?

In Exercises 15–21, prove that the program segment is correct by finding and proving the appropriate loop invariant Q and evaluating Q at loop termination.

15. Function to return the value $x * y^n$ for $n \geq 0$.

```

Computation (integer  $x$ ; integer  $y$ ; nonnegative integer  $n$ )
Local variables:
integers  $i, j$ 
   $i = 0$ 
   $j = x$ 
  while  $i \neq n$  do
     $j = j * y$ 
     $i = i + 1$ 
  end while
  //  $j$  now has the value  $x * y^n$ 
  return  $j$ 
end function Computation

```

16. Function to return the value $x - y$ for $x, y \geq 0$.

```

Difference (nonnegative integer  $x$ ; nonnegative integer  $y$ )
Local variables:
integers  $i, j$ 
   $i = 0$ 
   $j = x$ 
  while  $i \neq y$  do
     $j = j - 1$ 
     $i = i + 1$ 
  end while
  //  $j$  now has the value  $x - y$ 
  return  $j$ 
end function Difference

```

17. Function to return the value $(x + 1)^2$ for $x \geq 1$.

```

IncrementSquare (positive integer  $x$ )
Local variables:
integers  $i, j$ 
   $i = 1$ 
   $j = 4$ 
  while  $i \neq x$  do
     $j = j + 2i + 3$ 
     $i = i + 1$ 
  end while
  //  $j$  now has the value  $(x + 1)^2$ 
  return  $j$ 
end function IncrementSquare

```

18. Function to return the value 2^n for $n \geq 1$.

```

TwosPower (positive integer  $n$ )
Local variables:
integers  $i, j$ 
   $i = 1$ 
   $j = 2$ 
  while  $i \neq n$  do
     $j = j * 2$ 
     $i = i + 1$ 
  end while
  //  $j$  now has the value  $2^n$ 
  return  $j$ 
end function TwosPower

```

19. Function to return the value $x * n!$ for $n \geq 1$.

```

AnotherOne (integer  $x$ ; positive integer  $n$ )
Local variables:
integers  $i, j$ 
   $i = 1$ 
   $j = x$ 
  while  $i \neq n$  do
     $j = j * (i + 1)$ 
     $i = i + 1$ 
  end while
  //  $j$  now has the value  $x * n!$ 
  return  $j$ 
end function AnotherOne

```

20. Function to return the value of the polynomial $a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ at a given value of x .

```

Polly (real  $a_n; \dots; \text{real } a_0; \text{real } x$ )
Local variables:
integers  $i, j$ 
   $i = n$ 
   $j = a_n$ 
  while  $i \neq 0$  do
     $j = j * x + a_{i-1}$ 
     $i = i - 1$ 
  end while
  //  $j$  now has value of the polynomial evaluation
  return  $j$ 
end function Polly

```

21. Function to return the maximum value from the first n entries $a[1], a[2], \dots, a[n]$, $n \geq 1$, in an array of distinct integers.

```

ArrayMax (integers  $n, a[1], a[2], \dots, a[n]$ )
Local variables:
integers  $i, j$ 
   $i = 1$ 

```

```

j = a[1]
while i ≠ n do
    i = i + 1
    if a[i] > j then j = a[i]
end while
//j now has the value of the largest array element
return j
end function ArrayMax

```

22. Following are four functions intended to return the value $a[1] + a[2] + \cdots + a[n]$ for $n \geq 1$ (the sum of the first n entries in an array of integers). For those that do not produce correct results, explain what goes wrong. For those that do produce correct results, do a proof of correctness.

a. *ArraySumA* (integers $n, a[1], a[2], \dots, a[n]$)

Local variables:

integers i, j

$i = 0$

$j = 0$

while $i \leq n$ **do**

$i = i + 1$

$j = j + a[i]$

end while

//j now has the value $a[1] + a[2] + \cdots + a[n]$

return j

end function *ArraySumA*

b. *ArraySumB* (integers $n, a[1], a[2], \dots, a[n]$)

Local variables:

integers i, j

$i = 1$

$j = 0$

while $i \leq n$ **do**

$j = j + a[i]$

$i = i + 1$

end while

//j now has the value $a[1] + a[2] + \cdots + a[n]$

return j

end function *ArraySumB*

c. *ArraySumC* (integers $n, a[1], a[2], \dots, a[n]$)

Local variables:

integers i, j

$i = 0$

$j = 0$

while $i \leq n$ **do**

$j = j + a[i]$

$i = i + 1$

end while

//j now has the value $a[1] + a[2] + \cdots + a[n]$

return j

end function *ArraySumC*

```

d. ArraySumD (integers  $n, a[1], a[2], \dots, a[n]$ )
Local variables:
integers  $i, j$ 
 $i = 1$ 
 $j = a[1]$ 
while  $i \leq n$  do
     $j = j + a[i + 1]$ 
     $i = i + 1$ 
end while
// $j$  now has the value  $a[1] + a[2] + \dots + a[n]$ 
return  $j$ 
end function ArraySumD

```

Exercises 23–28 concern a variation of the Euclidean algorithm more suited to computer implementation. The original GCD algorithm relies on repeated integer divisions to compute remainders. The following variation, called the *binary GCD algorithm*, also uses divisions, but only by 2, plus subtraction and testing for parity (oddness or evenness). Given that the numbers are stored in the computer in binary form, these operations are simple and often done using built-in circuits. Testing for even/odd can be done using bitwise conjunction of N & 1, which results in 1 if and only if N is odd. Given an even N , division by 2 is easily accomplished by a 1-bit right shift operation, where all bits are shifted one place to the right (the rightmost bit disappears), and the leftmost bit is set to 0. (Multiplication by 2 is a 1-bit left shift.) Subtraction involves the 2's complement. As a result, the binary gcd algorithm, which avoids regular division, runs faster than the Euclidean algorithm, even though it does more (but simpler) steps. The binary GCD algorithm relies on three facts:

1. If both a and b are even, then $\gcd(a, b) = 2\gcd(a/2, b/2)$.
2. If a is even and b is odd, then $\gcd(a, b) = \gcd(a/2, b)$.
3. If a and b are both odd, and $a \geq b$, then $\gcd(a, b) = \gcd((a - b)/2, b)$.

23. To prove that if both a and b are even, then $\gcd(a, b) = 2\gcd(a/2, b/2)$, let a and b be even integers. Then 2 is a common factor of both a and b , so 2 is a factor of $\gcd(a, b)$. Let $2c = \gcd(a, b)$. Then

$$a = n(2c) \text{ and } b = m(2c)$$

$$a/2 = nc \text{ and } b/2 = mc$$

so $c \mid a/2$ and $c \mid b/2$. Finish this proof by showing that $c = \gcd(a/2, b/2)$.

24. To prove that if a is even and b is odd, then $\gcd(a, b) = \gcd(a/2, b)$, note that because b is odd, 2 is not a factor of b , hence not a factor of $\gcd(a, b)$. Therefore all contribution to $\gcd(a, b)$ comes from b and $a/2$, and $\gcd(a, b) = \gcd(a/2, b)$. Write an equation for $\gcd(a, b)$ when a is odd and b is even.

25. We want to prove that if a and b are both odd, and $a \geq b$, then $\gcd(a, b) = \gcd((a - b)/2, b)$. If a and b are both odd and $a \geq b$, then $\gcd(a, b) = \gcd(a - b, b)$ because, from the regular Euclidean algorithm, $\gcd(a, b)$ begins with $a = qb + r$, $0 \leq r < b$ and $\gcd(a - b, b)$ begins with $a - b = (q - 1)b + r$, $0 \leq r < b$. The next step in either case is to divide b by r , so the two final answers will be the same. Finish this proof by showing that $\gcd(a - b, b) = \gcd((a - b)/2, b)$.

26. To find $\gcd(420, 66)$ using the binary GCD algorithm takes the following steps (compare with Example 27). You can do these steps in your head!

420	66	Fact 1	Save the 2 factor to multiply at the end
210	33	Fact 2	
105	33	Fact 3	
36	33	Fact 2	
18	33	Fact 2	
9	33	Fact 3	[Because $\gcd(a, b) = \gcd(b, a)$, it doesn't matter whether the larger number is first or second or whether the even number is first or second.]
9	12	Fact 2	
9	6	Fact 2	
9	3	Fact 3	
3	3	Fact 3	
0	3		

One number is now 0, so the other number is a factor in the gcd, therefore $\gcd(420, 66) = 2 \cdot 3 = 6$ (the factor of 2 comes from the very first step).

Use the binary GCD algorithm to find $\gcd(24, 20)$.

27. Use the binary GCD algorithm to find $\gcd(308, 165)$ [see Exercise 7].
 28. Use the binary GCD algorithm to find $\gcd(2420, 70)$ [see Exercise 8].

SECTION 2.4 NUMBER THEORY

In Section 2.1 we proved several elementary number theory results, such as “The product of two even integers is even.” These proofs relied on basic definitions and the standard proof techniques (direct proof, proof by contraposition, and proof by contradiction). Now that we have some additional ammunition, we can prove more number theory results. Number theory is fun because conjectures can be stated easily—after all, only integers are involved—yet sometimes it can be quite difficult to prove. As an extreme case, **Fermat's last theorem** states that there are no positive integers x , y , and z for which

$$x^n + y^n = z^n$$

for any integer $n > 2$. (There are solutions for $n = 2$, such as $3^2 + 4^2 = 5^2$.) Pierre de Fermat stated this result around 1637 but—although many false “proofs” were published in the interim—it took until 1995 before a proof was found, using very complicated mathematics, by Dr. Andrew Wiles of Princeton University. We are interested in number theory, however, because of its usefulness in computer security (see Section 5.6).

REMINDER

A prime number is an integer > 1 that is divisible only by itself and 1.

The Fundamental Theorem of Arithmetic

We'll start by expanding on a result we proved using the second principle of mathematical induction in Example 23, Section 2.2: For every integer $n \geq 2$, n is a prime number or a product of prime numbers.

In fact, a stronger statement can be made.

● **THEOREM THE FUNDAMENTAL THEOREM OF ARITHMETIC**

For every integer $n \geq 2$, n is a prime number or can be written uniquely (ignoring ordering) as a product of prime numbers.

The new part is that there is only one way to factor a composite number into prime factors if we ignore the order in which we write the factors. Thus we consider

$$2(3)(3) = 3(2)(3)$$

to be the same factorization of 18. We intuitively accept the uniqueness idea—how else could you factor 18 into prime factors? A formal proof requires quite a bit of preparation, and it begins with revisiting the idea of the greatest common divisor of two positive integers.

The Euclidean algorithm to find $\gcd(a, b)$ was given in Section 2.3. It turns out that if a and b are positive integers, then $\gcd(a, b)$ can always be written as a **linear combination** of a and b ; that is,

$$\gcd(a, b) = ia + jb \text{ for some integers } i \text{ and } j$$

EXAMPLE 29

In Section 2.3 we learned, using the Euclidean algorithm, that $\gcd(420, 66) = 6$. And 6 can be written as a linear combination of 420 and 66:

$$6 = 3(420) - 19(66)$$

Although it's easy to verify in Example 29 that $3(420) - 19(66)$ indeed has the value 6, the coefficient values of 3 and -19 seem mysterious. They are not just pulled out of the air, however; in fact they are derived from the successive divisions done by the Euclidean algorithm.

EXAMPLE 30

The successive divisions performed by the Euclidean algorithm in finding $\gcd(420, 66)$ can be written as follows (see Example 27 in Section 2.3):

$$420 = 6 \cdot 66 + 24$$

$$66 = 2 \cdot 24 + 18$$

$$24 = 1 \cdot 18 + 6$$

$$18 = 3 \cdot 6 + 0$$

Rewriting the first three equations from the bottom up,

$$\begin{aligned}6 &= 24 - 1 \cdot 18 \\18 &= 66 - 2 \cdot 24 \\24 &= 420 - 6 \cdot 66\end{aligned}$$

Now we use these equations in a series of substitutions:

$$\begin{aligned}6 &= 24 - 1 \cdot 18 = 24 - 1 \cdot (66 - 2 \cdot 24) \quad (\text{substituting for } 18) \\&= 3 \cdot 24 - 66 \\&= 3 \cdot (420 - 6 \cdot 66) - 66 \quad (\text{substituting for } 24) \\&= 3 \cdot 420 - 19 \cdot 66\end{aligned}$$

which reveals the linear combination of 420 and 66 that gives the value 6. ●

The Euclidean algorithm gives us a way to express $\gcd(a, b)$ as a linear combination of a and b , but there is another way to characterize this linear combination.

● **THEOREM ON GCD(a, b)**
Given positive integers a and b , $\gcd(a, b)$ is the linear combination of a and b that has the smallest positive value.

To prove this result, we need to make use of the principle of well-ordering that we mentioned in Section 2.2, namely, that every collection of positive integers that contains any members at all has a smallest member. The collection we have in mind consists of all positive linear combinations of a and b , and certainly such numbers exist (as a trivial example, $1 \cdot a + 1 \cdot b$). By the principle of well-ordering, there is a least such number $c = ia + jb$ where i and j are integers. The theorem claims that $c = \gcd(a, b)$, that is, that $c | a$, $c | b$, and c is the largest integer that divides both a and b .

To prove that $c | a$, we'll use a proof by contradiction. Suppose $c \nmid a$. Then when we divide a by c there is a nonzero remainder

$$a = mc + r \quad \text{with } m \text{ an integer and } 0 < r < c$$

Rewriting this equation,

$$\begin{aligned}r &= a - mc \\&= a - m(ia + jb) \\&= (1 - mi)a - (mj)b\end{aligned}$$

which makes r a positive linear combination of a and b , that is, r is a member of our collection, but $r < c$, which is a contradiction because c was the least member of this collection. Therefore $c | a$. In the same way we can show that $c | b$. Consequently c is a common divisor of a and b ; by Practice 11, c is the greatest common divisor of a and b , which completes the proof of the theorem.

PRACTICE 11

- a. Prove that if d is a positive integer such that $d|a$ and $d|b$, then $d|c$, where $c = ia + jb$.
 b. Prove that if $d|c$, then $c \geq d$. ■

● **DEFINITION** **RELATIVELY PRIME**

Two integers a and b are **relatively prime** if $\gcd(a, b) = 1$.

From the theorem on $\gcd(a, b)$, it follows that a and b are relatively prime if and only if there exist integers i and j such that

$$ia + jb = 1$$

PRACTICE 12 The integers 21 and 16 are relatively prime. Find i and j such that $i(21) + j(16) = 1$. ■

Recall that a prime number is an integer $p > 1$ that is not divisible by any integers other than 1 and p . If a is an integer that is a multiple of a prime p , then clearly $p|p$ and $p|a$, so $\gcd(a, p) = p$. But if a is not a multiple of p , then $\gcd(a, p) = 1$ because nothing else divides p . Therefore all integers are either multiples of p or are relatively prime to p .

Suppose that p is a prime number that divides the product ab of integers a and b . Because p is “irreducible,” p must divide either a or b . More formally, if p does not divide a , that is, a is not a multiple of p , then a is relatively prime to p , $\gcd(a, p) = 1$, and there exist integers i and j such that

$$1 = ia + jp$$

Multiplying this equation by b ,

$$b = (ia)b + (jp)b = i(ab) + (jp)b$$

Because $p|ab$, ab can be written as kp , where k is an integer, so the previous equation becomes

$$b = i(kp) + (jp)b = (ik + jb)p \quad ik + jb \text{ an integer}$$

so that $p|b$. This proves the following theorem.

● **THEOREM** **ON DIVISION BY PRIME NUMBERS**

Let p be a prime number such that $p|ab$ where a and b are integers. Then either $p|a$ or $p|b$.

PRACTICE 13 Extend the theorem on division by prime numbers as follows: Let p be a prime number such that $p|a_1a_2 \dots a_k$ where each a_i is an integer. Then $p|a_j$ for some j , $1 \leq j \leq k$. (*Hint:* You want to prove that this is true for every positive integer k —What proof technique should you use?) ■

Finally we are ready to prove that the factorization of a composite number $n > 2$ into prime factors is unique (save for ordering). If n is a composite number, then n can be written as a product of primes:

$$n = p_1 p_2 \cdots p_r \quad \text{where } p_1 \leq p_2 \leq \cdots \leq p_r \text{ and each } p_i \text{ is a prime number}$$

Now suppose that n can also be written as

$$n = q_1 q_2 \cdots q_s \quad \text{where } q_1 \leq q_2 \leq \cdots \leq q_s \text{ and each } q_i \text{ is a prime number}$$

Then

$$p_1 p_2 \cdots p_r = q_1 q_2 \cdots q_s$$

We are assuming that these two representations are different, but they might still have some factors in common on both sides of the equation; let's assume these have been divided out. Then

$$p_1 | p_1 p_2 \cdots p_r$$

so

$$p_1 | q_1 q_2 \cdots q_s$$

By Practice 13, $p_1 | q_i$ for some i , $1 \leq i \leq s$. However, q_i is a prime number, divisible only by itself and 1, which would mean that $p_1 = q_i$. This is a contradiction because we already eliminated common factors.

EXAMPLE 31

To find the unique factorization of 825 as a product of primes, we can start by simply dividing 825 by successively larger primes (2, 3, 5, and so on).

$$2 \nmid 825 \\ 825 = 3 \cdot 275 = 3 \cdot 5 \cdot 55 = 3 \cdot 5 \cdot 5 \cdot 11 = 3 \cdot 5^2 \cdot 11$$

Similarly,

$$455 = 5 \cdot 7 \cdot 13$$

From these factorizations we can see that $\gcd(825, 455) = 5$. Decomposing two positive integers into their respective prime number factorizations is another way (besides the Euclidean algorithm) to determine their greatest common divisor. ●

PRACTICE 14 Find the unique factorization of 1176 as a product of primes. ■

PRACTICE 15 Find $\gcd(420, 66)$ by unique factorization into products of primes (see Example 27). ■

We have now completed the proof of the fundamental theorem of arithmetic. Note, however, that this is an *existence* result. It says that for any integer $n \geq 2$ that is not prime, there exists a unique factorization as a product of primes. But this does not tell us

- a. how to decide whether n is prime.
- b. if n is not prime, how to find the prime factors of n .

Neither of these problems has an efficient algorithmic solution. The approach in Example 31 of dividing a number n by successively larger primes accomplishes both tasks—if there are prime factors of n , they will be discovered and if there are none, then n is prime. However, this approach becomes very labor intensive when n is large.

More on Prime Numbers

Anything with a title as imposing as the fundamental theorem of arithmetic must be fairly important. We can use this theorem to discover several more results about prime numbers.

Given a positive integer n , suppose we test for prime factors by dividing n by successively larger primes. Clearly we can stop with the largest prime less than or equal to n , but in fact we can stop with the largest prime less than or equal to \sqrt{n} . If n can be factored in a nontrivial way as $n = st$, then s and t cannot both be greater than \sqrt{n} because then their product would be greater than n . Therefore one of s and t , let's say s , must be less than or equal to \sqrt{n} . By the fundamental theorem of arithmetic, s is either prime or can be written as a product of primes. In either case, there is a prime factor less than or equal to \sqrt{n} , which proves the following theorem.

THEOREM ON SIZE OF PRIME FACTORS

If n is a composite number, then it has a prime factor less than or equal to \sqrt{n} .

EXAMPLE 32

Given $n = 1021$, let's find the prime factors of n or determine that n is prime. The value of $\sqrt{1021}$ is just less than 32. So the primes we need to test are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31. None divides 1021, so 1021 is prime. ●

How many prime numbers are there? An infinite number.

THEOREM ON INFINITY OF PRIMES (EUCLID)

There is an infinite number of prime numbers.

Proof: Assume that there is a finite number of primes, listed as p_1, p_2, \dots, p_k . Consider the number $s = p_1 p_2 \cdots p_k + 1$. The integer s is greater than any of the primes p_1, \dots, p_k , which we assumed is the total list of primes, therefore s is not prime. Thus s is composite and, by the fundamental theorem of arithmetic, s can be factored as a product of (some of) the prime numbers. Suppose that p_j is one of the prime factors of s , that is, $s = p_j(m)$ for some integer m . Then

$$1 = s - p_1 p_2 \cdots p_k = p_j(m) - p_1 \cdots p_k = p_j(m - p_1 \cdots p_{j-1} p_{j+1} \cdots p_k)$$

Therefore $p_j | 1$, which is a contradiction. *End of Proof*

Although Euclid's theorem says that there is always another prime number ahead as we march through the positive integers, the distribution of primes among the integers is erratic. Contrary to what one might think, the primes do not get farther and farther apart. Even among the small primes, 23 and 29 are farther apart than 29 and 31.

EXAMPLE 33

In Example 11, we did a proof by contradiction that $\sqrt{2}$ is not a rational number. The same argument works for $\sqrt{3}$ and $\sqrt{5}$ (2, 3, and 5 are all prime numbers).

We can generalize this result from a single prime to any integer x that is the product of an odd number of primes. Assume that by the fundamental theorem of arithmetic, $x = p_1 p_2 \cdots p_{2k+1}$ (In this example, we are not using any exponents, so some of these primes could be identical; that is, $75 = 3 \cdot 5 \cdot 5$). Again doing a proof by contradiction, assume that $\sqrt{p_1 p_2 \cdots p_{2k+1}} = a/b$ where a and b are integers, $b \neq 0$, and a and b are relatively prime. Then

$$p_1 p_2 \cdots p_{2k+1} = \frac{a^2}{b^2}$$

$$a^2 = p_1 p_2 \cdots p_{2k+1} b^2$$

By the fundamental theorem, a can be written as a product of one or more primes, but a^2 will add another factor of each of a 's primes, resulting in a product of an even number of primes. Similarly, b^2 is the product of an even number of primes. Therefore $p_1 p_2 \cdots p_{2k+1} b^2$ is the product of an odd number and an even number (which gives an odd number) of prime factors. Contradiction: a^2 has an even number of prime factors while $p_1 p_2 \cdots p_{2k+1} b^2$ has an odd number of prime factors, yet by the fundamental theorem, factorization is unique. ●

The search for prime numbers and information about primes has generated much interest. As of June 2013, the largest known prime number is $2^{57,885,161} - 1$, which is a number with 17,425,170 decimal digits. If we consider that for a reasonable type size it takes about 1 inch to print 10 digits, then it would take more than 27 miles just to print out a number of this size!

One of the oldest conjectures about prime numbers—still unsolved—is *Goldbach's conjecture*, formulated in 1742: Every even integer greater than 2 is the sum of two prime numbers.

Euler Phi Function

DEFINITION EULER PHI FUNCTION

For n an integer, $n \geq 2$, the **Euler phi function** of n , $\varphi(n)$, is the number of positive integers less than or equal to n and relatively prime to n . ($\varphi(n)$ is pronounced “fee” of n .)

Notice that n will never be relatively prime to n , so this definition could have been stated as, “the number of positive integers less than n and relatively prime to n ,” but it turns out to be convenient to include equality.

EXAMPLE 34

The first few values of $\varphi(n)$, together with the numbers that give those values, are

$$\begin{aligned}\varphi(2) &= 1 \text{ (the number 1)} \\ \varphi(3) &= 2 \text{ (the numbers 1, 2)} \\ \varphi(4) &= 2 \text{ (the numbers 1, 3)} \\ \varphi(5) &= 4 \text{ (the numbers 1, 2, 3, 4)} \\ \varphi(6) &= 2 \text{ (the numbers 1, 5)} \\ \varphi(7) &= 6 \text{ (the numbers 1, 2, 3, 4, 5, 6)}\end{aligned}$$

PRACTICE 16

If p is a prime number, prove that $\varphi(p) = p - 1$.

For small n , it is easy to compute $\varphi(n)$ by a brute-force approach of just trying values $< n$ to find how many are relatively prime to n . But there is actually a formula to compute $\varphi(n)$, which we'll derive now.

EXAMPLE 35

Using the fundamental theorem of arithmetic, write the positive integer n in its factored form as a product of primes, where if the same prime p occurs m times, it is written as p^m . Suppose, for example, that

$$n = p_1^{m_1} p_2^{m_2} p_3^{m_3}$$

To compute $\varphi(n)$, we'll count all the positive integers $\leq n$, of which there are n , and throw out those that are not relatively prime to n . Now let A_i , $1 \leq i \leq 3$, be defined as the collection of all positive integral multiples of p_i that are $\leq n$; these numbers share a common factor of p_i with n and so are not relatively prime to n . The integral multiples of p_i that are $\leq n$ are $p_i, 2p_i, 3p_i, \dots, n$. How many numbers are in this list? Exactly the number of times you can divide n by p_i , or n/p_i . So, denoting the size of A_i by $|A_i|$, we know that $|A_i| = n/p_i$. If we combine A_1, A_2 , and A_3 , that will be all integers $\leq n$ that are not relatively prime to n . How many such integers are there? We can't just add $|A_1| + |A_2| + |A_3| = n/p_1 + n/p_2 + n/p_3$ because there could be some numbers that appear in more than one of the three collections, and would therefore be counted twice. Numbers that appear in both A_i and A_j , $i \neq j$, are integral multiples of both p_i and p_j , so there will be $n/(p_i p_j)$ of them. We'll subtract those for all i, j combinations. But by doing this, we have subtracted any numbers in all three collections three times, which means they now are not counted at all, so we have to add back the $n/(p_1 p_2 p_3)$ numbers that are in all three collections.² Therefore

²This lengthy discussion is an instance of the principle of inclusion and exclusion, discussed in Chapter 4.

$$\begin{aligned}
\varphi(n) &= n - \left(\frac{n}{p_1} + \frac{n}{p_2} + \frac{n}{p_3} - \frac{n}{p_1 p_2} - \frac{n}{p_2 p_3} - \frac{n}{p_1 p_3} + \frac{n}{p_1 p_2 p_3} \right) \\
&= n \left(1 - \frac{1}{p_1} - \frac{1}{p_2} - \frac{1}{p_3} + \frac{1}{p_1 p_2} + \frac{1}{p_2 p_3} + \frac{1}{p_1 p_3} - \frac{1}{p_1 p_2 p_3} \right) \\
&= n \left(\frac{p_1 p_2 p_3 - p_2 p_3 - p_1 p_3 - p_1 p_2 + p_3 + p_1 + p_2 - 1}{p_1 p_2 p_3} \right) \quad (\text{adding fractions}) \\
&= n \left(\frac{(p_1 - 1)(p_2 - 1)(p_3 - 1)}{p_1 p_2 p_3} \right) \quad (\text{check this by multiplying out the numerator}) \\
&= \frac{n}{p_1 p_2 p_3} (p_1 - 1)(p_2 - 1)(p_3 - 1) \\
&= p_1^{m_1 - 1} p_2^{m_2 - 1} p_3^{m_3 - 1} \varphi(p_1) \varphi(p_2) \varphi(p_3) \quad (1)
\end{aligned}$$

Equation (1) expresses $\varphi(n)$ in terms of the Euler phi function of its prime factors, which are known to us (see Practice 16). ●

Equation (1) in Example 34 gives the formula for $\varphi(n)$ where n has 3 distinct prime factors. It is easy to extend this equation to the more general case where n has an arbitrary number of prime factors. If

$$n = p_1^{m_1} p_2^{m_2} \cdots p_k^{m_k}$$

then

$$\varphi(n) = p_1^{m_1 - 1} p_2^{m_2 - 1} \cdots p_k^{m_k - 1} [\varphi(p_1) \varphi(p_2) \cdots \varphi(p_k)] \quad (2)$$

EXAMPLE 36

For $n = 133848 = 2^3 \cdot 3^2 \cdot 11 \cdot 13^2$,

$$\varphi(n) = 2^2 \cdot 3 \cdot 13 [\varphi(2) \varphi(3) \varphi(11) \varphi(13)] = 4 \cdot 3 \cdot 13 [1 \cdot 2 \cdot 10 \cdot 12] = 37440 \quad \bullet$$

Equation (2) requires that we know the prime factorization of n , so it does not avoid the difficulty we noted earlier of factoring large values of n .

PRACTICE 17

For $n = 3^4 \cdot 5 \cdot 7^2$, compute $\varphi(n)$. ■

SECTION 2.4 REVIEW

TECHNIQUES

- W** Write the $\gcd(a, b)$ as a linear combination of a and b .
- Test whether a given positive integer is prime or, if not, find its prime factorization.
 - Compute the Euler phi function $\varphi(n)$ for a positive integer n .

MAIN IDEAS

- Every integer ≥ 2 is prime or can be uniquely factored into prime numbers (fundamental theorem of arithmetic).

- Two integers a and b are relatively prime if a linear combination of a and b can be found that equals 1.
- If n is not prime, it has a prime factor no greater than \sqrt{n} .
- An infinite number of prime numbers exist.
- There is no efficient algorithm to decide whether a positive integer n is prime, or to find the prime factors of n if n is not itself prime.
- Given n written as a product of primes, there is a formula to compute $\varphi(n)$, the number of positive integers $\leq n$ and relatively prime to n .

EXERCISES 2.4

Exercises 1–6 refer to Exercises 7–12 in Section 2.3

1. Write $\gcd(308, 165)$ as a linear combination of 308 and 165.
2. Write $\gcd(2420, 70)$ as a linear combination of 2420 and 70.
3. Write $\gcd(735, 90)$ as a linear combination of 735 and 90.
4. Write $\gcd(8370, 465)$ as a linear combination of 8370 and 465.
5. Write $\gcd(1326, 252)$ as a linear combination of 1326 and 252.
6. Write $\gcd(1018215, 2695)$ as a linear combination of 1018215 and 2695.

For Exercises 7–12, test whether n is prime and, if not, find its decomposition as a product of primes.

7. $n = 1729$
8. $n = 1789$
9. $n = 1171$
10. $n = 1177$
11. $n = 8712$
12. $n = 29575$

Exercises 13–18 refer to Exercises 7–12 in Section 2.3

13. Find $\gcd(308, 165)$ by unique factorization into products of primes.
14. Find $\gcd(2420, 70)$ by unique factorization into products of primes.
15. Find $\gcd(735, 90)$ by unique factorization into products of primes.
16. Find $\gcd(8370, 465)$ by unique factorization into products of primes.
17. Find $\gcd(1326, 252)$ by unique factorization into products of primes.
18. Find $\gcd(1018215, 2695)$ by unique factorization into products of primes.
19. The least common multiple of two positive integers a and b , $\text{lcm}(a, b)$ is the smallest integer n such that $a \mid n$ and $b \mid n$. Like the $\gcd(a, b)$, the $\text{lcm}(a, b)$ can be found from the prime factorizations of a and b . Describe (in words) the \gcd and the lcm in terms of the prime factors of a and b .
20. Prove that for any positive integers a and b , $a \cdot b = \gcd(a, b) \cdot \text{lcm}(a, b)$. (*Hint*: consider both a and b in their factored form as a product of primes.)

For Exercises 21–24, find the gcd and lcm of the two numbers given.

21. $a = 2^2 \cdot 3 \cdot 11$, $b = 2 \cdot 3 \cdot 11^2 \cdot 13$

22. $a = 2^4 \cdot 5^2 \cdot 7^3$, $b = 5 \cdot 7^2$

23. $a = 3 \cdot 5^3 \cdot 11^2$, $b = 3^2 \cdot 5 \cdot 11 \cdot 17$

24. $a = 5 \cdot 11 \cdot 23^2$, $b = 5^3 \cdot 11^3$

25. Prove that for any positive integers a and b , $\gcd(a, b) = \gcd(a, a + b)$.

26. Prove that $\gcd(n, n + 1) = 1$ for all positive integers n .

27. Find an example where $n \mid ab$ but $n \nmid a$ and $n \nmid b$. Does this violate the theorem of division by prime numbers?

28. The division of a full circle into 360° probably dates back to the early Persian calendar from around 700 B.C. that used 360 days in a year, so one day represented a rotation of $1/360$ of other stars around the North Star. But it was also chosen because it is divisible by so many factors, avoiding the need to deal with fractions. Find the distinct nontrivial (but not necessarily prime) factors of 360.

29. Prove that there exist three consecutive odd positive integers that are prime numbers.

30. Prove that for any positive integer n , there exist n consecutive composite numbers. [Hint: Start with $(n + 1)! + 2$]

For Exercises 31–34, find $\varphi(n)$ together with the numbers that give those values.

31. $n = 8$

32. $n = 9$

33. $n = 10$

34. $n = 11$

35. By Practice 16, if p is prime then $\varphi(p) = p - 1$. Prove that this is an “if and only if” condition by proving that if $\varphi(n) = n - 1$ for a positive integer $n > 1$, then n is prime.

36. For any prime number p and any positive integer k , $\varphi(p^k) = p^{k-1}\varphi(p)$. Although this result follows directly from Equation (2) in this section, give a direct proof using the definition of the Euler phi function.

37. Compute $\varphi(2^4)$ and state the numbers being counted. (Hint: See Exercise 36.)

38. Compute $\varphi(3^3)$ and state the numbers being counted. (Hint: See Exercise 36.)

For Exercises 39–42, compute $\varphi(n)$.

39. $n = 117612 = 2^2 \cdot 3^5 \cdot 11^2$

40. $n = 233206 = 2 \cdot 17 \cdot 19^3$

41. $n = 1795625 = 5^4 \cdot 13^2 \cdot 17$

42. $n = 1,690,541,699 = 7^4 \cdot 11^3 \cdot 23^2$

43. If p and q are prime numbers with $p \neq q$, then $\varphi(pq) = \varphi(p)\varphi(q)$. Although this result follows directly from Equation (2) in this section, give a direct proof using the definition of the Euler phi function.

44. Prove that if r and s are relatively prime, then $\varphi(rs) = \varphi(r)\varphi(s)$.

45. Prove that for n and m positive integers, $\varphi(n^m) = n^{m-1}\varphi(n)$.

46. Except for $n = 2$, all the values of $\varphi(n)$ in Example 34 are even numbers. Prove that $\varphi(n)$ is even for all $n > 2$.

47. A particular class of prime numbers is known as Mersenne primes, named for a French monk and mathematician of the seventeenth century who studied them. Mersenne primes are numbers of the form $2^p - 1$ where p is a prime, but not all numbers of this form are primes. For example, $2^{11} - 1 = 23 \cdot 89$ is not

prime. The largest known prime number as of June 2013 is $2^{57,885,161} - 1$, a Mersenne prime. There happens to be a particularly efficient algorithm for testing numbers of the form $2^p - 1$ for primality, which is why almost all of the largest known primes are Mersenne primes. In recent years, most of these Mersenne primes have been discovered (and verified) by the GIMPS (Great Internet Mersenne Prime Search) distributed computing project, a worldwide group of volunteers who collaborate over the Internet to test for Mersenne primes.

Find the first 4—the 4 smallest—Mersenne primes.

48. Goldbach's conjecture states that every even integer greater than 2 is the sum of two prime numbers. Verify Goldbach's conjecture for
- $n = 8$
 - $n = 14$
 - $n = 28$
49. A perfect number is a positive integer n that equals the sum of all divisors less than n . For example, 6 is a perfect number because $6 = 1 + 2 + 3$. Perfect numbers are related to Mersenne primes (see Exercise 47) in that if p is a prime and $2^p - 1$ is a prime, then $2^{p-1}(2^p - 1)$ is a perfect number. (This result was proved by Euclid around 300 B.C.). For example, $6 = 2^1(2^2 - 1)$.
- Prove that 28 is a perfect number by writing it as the sum of its divisors.
 - Write 28 in the form $2^{p-1}(2^p - 1)$.
50. a. Prove that 496 is a perfect number (see Exercise 49) by writing it as the sum of its divisors.
b. Write 496 in the form $2^{p-1}(2^p - 1)$.
51. An algorithm exists to find all prime numbers less than some given positive integer n . This method, called the Sieve of Eratosthenes, was discovered by Eratosthenes, a student of Plato. To carry out this algorithm, list all integers from 1 through $n - 1$. Then make repeated passes through the list, on the first pass crossing out all multiples of 2 that are greater than 2. On the second pass cross out all multiples of 3 that are greater than 3. On the next pass, cross out all multiples of 5 that are greater than 5, and so forth for all primes less than \sqrt{n} . The numbers remaining when this process terminates are the primes less than n . Use the Sieve of Eratosthenes to find all prime numbers less than 100.
52. a. Compute the square of 11.
b. Compute the square of 111.
c. Prove that any n -digit number consisting of all 1's, when squared, produces the number $123 \dots (n - 1)n(n - 1) \dots 321$. A number that reads the same forward and backward is called a *palindrome*.
53. *Sudoku* puzzles are popular number-based puzzles. A game consists of a 9×9 grid made up of nine 3×3 blocks. Each row and each column of the game must contain exactly one of the 9 digits 1 – 9; furthermore, each 3×3 block must contain exactly one of the digits 1 through 9. Following is a sample puzzle, used by permission from Web Sudoku at <http://www.websudoku.com>, where you can generate puzzles at any of four levels of difficulty. Try completing this puzzle.

			2	1	7			
	4				3			2
2		1			4	5	8	
9					5	8	7	
1		8		2		4		9
	6	3	9					5
	8	2	3			7		1
3			4				5	
			8	7	2			

CHAPTER 2 REVIEW

TERMINOLOGY

absolute value (p. 107)	first principle of mathematical induction (p. 111)	odd number (p. 101)
basis step (p. 111)	fundamental theorem of arithmetic (p. 144)	partial correctness (p. 132)
composite number (p. 107)	greatest common divisor (p. 133)	perfect square (p. 107)
contrapositive (p. 103)	inductive assumption (p. 112)	prime number (p. 107)
converse (p. 103)	inductive hypothesis (p. 112)	proof by cases (p. 104)
counterexample (p. 99)	inductive reasoning (p. 99)	proof by contradiction (p. 104)
deductive reasoning (p. 99)	inductive step (p. 111)	proof by contraposition (p. 103)
direct proof (p. 101)	linear combination (p. 144)	proof by exhaustion (p. 100)
divides (p. 107)	loop invariant (p. 130)	rational number (p. 105)
Euclidean algorithm (p. 133)	loop rule of inference (p. 131)	relatively prime (p. 146)
Euler phi function (p. 149)	n factorial (p. 99)	second principle of mathematical induction (p. 118)
even number (p. 101)	number theory (p. 107)	well-ordering principle (p. 119)
Fermat's last theorem (p. 143)		

SELF-TEST

Answer the following true-false questions without looking back in the chapter.

Section 2.1

1. A conjecture can never be proved merely by proving a finite number of cases.
2. A proof by contradiction of $P \rightarrow Q$ begins by assuming both P and Q' .
3. In the statement of the theorem, "twice an odd integer is even," an existential quantifier is understood.
4. To prove the conjecture, "If Laramie is the capital, then Wyoming is the state," it is sufficient to prove, "If Wyoming is the state, then Laramie is the capital."
5. To prove, "A if and only if B," requires a proof of $A \rightarrow B$ and a proof of $B \rightarrow A$.

Section 2.2

1. Induction is an appropriate proof technique for proving a statement about all the positive integers.
2. The basis step of an inductive proof requires proving a property true for $n = 1$.
3. If the truth of $P(k + 1)$ depends on the truth of other previous values besides $P(k)$, then the second principle of induction should be used.
4. The key to a proof by the first principle of induction is to see how the truth of P at the value $k + 1$ depends on the truth of P at value k .
5. The equation $k^3 = k^2(k+1)^2/4$ is the inductive hypothesis in an inductive proof of the statement

$$1^3 + 2^3 + \cdots + n^3 = n^2(n + 1)^2/4$$

Section 2.3

1. A loop invariant remains true until the loop is exited, at which point it becomes false.
2. Partial correctness of a loop statement in a program means that the loop behaves correctly for some input values but not for others.
3. The second principle of induction is used to prove loop invariants because the loop can be executed an arbitrary number of times.
4. If a loop statement has the form

```

while (condition B)
    P
end while

```

then the loop invariant Q will be B' .

5. When computing the $\text{gcd}(42, 30)$ by the Euclidean algorithm, the computation of dividing 30 by 12 is carried out.

Section 2.4

1. $\text{gcd}(a, b)$ can always be written as a linear combination of a and b .
2. Two integers a and b are relatively prime if there exist integers i and j such that $ia + jb = p$ where p is a prime number.
3. If a positive integer n is not a prime number, then it has at least one prime factor $> \sqrt{n}$.
4. $\varphi(n)$ is a prime number for any integer $n \geq 2$.
5. $\varphi(p) = p - 1$ for any prime number p .

ON THE COMPUTER

For Exercises 1–5, write a computer program that produces the desired output from the given input.

- Input:* Number n of terms in a geometric progression (see Exercise 27, Section 2.2), the initial term a , and the common ratio r
Output: Sum of the first n terms using

 - iteration
 - formula of Exercise 27, Section 2.2
- Input:* Number n of terms in an arithmetic progression (see Exercise 28, Section 2.2), the initial term a , and the common difference d
Output: Sum of the first n terms using

 - iteration
 - formula of Exercise 28, Section 2.2
- Input:* Number n
Output: Sum of the first n cubes using

 - iteration, using only multiplication and addition; output the number of multiplications and additions used
 - formula of Exercise 8, Section 2.2, using only multiplication, addition, and division; output the number of multiplications, additions, and divisions used
- Input:* None
Output: Table showing every integer n , $8 \leq n \leq 100$, as the sum of $3s$ and $5s$ (see Example 24)
- Input:* Value for n
Output: Value for $\varphi(n)$
- The formula $4^n < n!$ is true for all $n \geq N$. Write a program to determine N and then prove the result by induction.
- The formula $2^n > n^3$ is true for all $n \geq N$. Write a program to determine N and then prove the result by induction.

Recursion, Recurrence Relations, and Analysis of Algorithms

CHAPTER OBJECTIVES

After studying this chapter, you will be able to:

- Understand recursive definitions of sequences, collections of objects, and operations on objects.
- Write recursive definitions for certain sequences, collections of objects, and operations on objects.
- Understand how recursive algorithms execute.
- Write recursive algorithms to generate sequences defined recursively.
- Find closed-form solutions for certain types of recurrence relations.
- Analyze algorithms by counting the number of executions of a basic unit of work, either directly or by solving a recurrence relation.

You are serving on the city council's Board of Land Management, which is considering a proposal by a private contractor to manage a chemical disposal site. The material to be stored at the site degrades to inert matter at the rate of 5% per year. The contractor claims that, at this rate of stabilization, only about one-third of the original active material will remain at the end of 20 years.

Question: Is the contractor's estimate correct?

It is possible to check this estimate by doing some brute-force calculations: If there is this much initially, then there will be that much next year, so much the following year, and so on through the 20 years. But a quick and elegant solution can be obtained by solving a recurrence relation; recurrence relations are discussed in Section 3.2.

Section 3.1 explores recursion, which is closely related to mathematical induction (discussed in the previous chapter) and is important in expressing many definitions and even algorithms. Some sequences defined recursively can also be defined by a formula. Finding such a formula involves solving a recurrence relation; solution methods for several types of recurrence relations are developed in Section 3.2. Recurrence relations are an important tool in the analysis of algorithms, which mathematically determines the amount of work a particular algorithm must do. Analysis of algorithms is the topic of Section 3.3.

SECTION 3.1 RECURSIVE DEFINITIONS

A definition in which the item being defined appears as part of the definition is called a **recursive definition**. At first this seems like nonsense—how can we define something in terms of itself? This works because there are two parts to a recursive definition:

1. A basis, where some simple cases of the item being defined are explicitly given
2. An inductive or recursive step, where new cases of the item being defined are given in terms of previous cases

Part 1 gives us a place to start by providing some simple, concrete cases; part 2 allows us to construct new cases from these simple ones and then to construct still other cases from these new ones, and so forth. (This seems analogous to proofs by mathematical induction. In a proof by induction, there is a basis step, namely, to show that $P(1)$ —or P at some other initial value—holds, and there is an inductive step where the truth of $P(k + 1)$ is deduced from the truth of P at previous values. This similarity is why the term **inductive definition** is sometimes used instead of recursive definition.)

Recursion is an important idea that can be used to define sequences of objects, more general collections of objects, and operations on objects. (The Prolog predicate *in-food-chain* of Section 1.5 was defined recursively.) Even algorithms can be recursive.

Recursively Defined Sequences

A **sequence** S (an **infinite sequence**) is a list of objects that are enumerated in some order; there is a first such object, then a second, and so on. $S(k)$ denotes the k th object in the sequence. The list goes on forever, so a sequence therefore consists of

$$S(1), S(2), \dots, S(k), \dots$$

Subscript notation is often used to denote the elements in a sequence, as in

$$S_1, S_2, \dots, S_k, \dots$$

The letter S is just a “dummy variable,” so a sequence could also be denoted by

$$a_1, a_2, \dots, a_k, \dots \quad \text{or} \quad w_1, w_2, \dots, w_k, \dots$$

and so forth.¹

A sequence is defined recursively by explicitly naming the first value (or the first few values) in the sequence and then defining later values in the sequence in terms of earlier values.

¹A more formal definition of a sequence is given in Chapter 5, Example 27.

EXAMPLE 1 The sequence S is defined recursively by

1. $S(1) = 2$
2. $S(n) = 2S(n - 1)$ for $n \geq 2$

By statement 1, $S(1)$, the first object in S , is 2. Then by statement 2, the second object in S is $S(2) = 2S(1) = 2(2) = 4$. By statement 2 again, $S(3) = 2S(2) = 2(4) = 8$. Continuing in this fashion, we can see that S is the sequence

$$2, 4, 8, 16, 32, \dots$$

A rule like that of statement 2 in Example 1, which defines a sequence value in terms of one or more earlier values, is called a **recurrence relation**.

PRACTICE 1 The sequence T is defined recursively as follows:

1. $T(1) = 1$
2. $T(n) = T(n - 1) + 3$ for $n \geq 2$

Write the first five values in the sequence T .

EXAMPLE 2 The **Fibonacci sequence** of numbers, introduced in the thirteenth century by an Italian merchant and mathematician, is defined recursively by

$$\begin{aligned} F(1) &= 1 \\ F(2) &= 1 \\ F(n) &= F(n - 2) + F(n - 1) \text{ for } n > 2 \end{aligned}$$

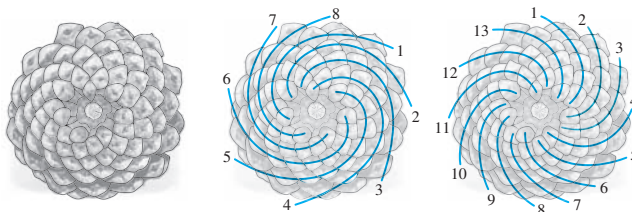
Here the first two values of the sequence are given, and the recurrence relation defines the n th value for $n > 2$ in terms of the two preceding values. It's best to think of the recurrence relation in its most general form, which says that F at any value—except 1 and 2—is the sum of F at the two previous values.

PRACTICE 2 Write the first eight values of the Fibonacci sequence.

The Fibonacci sequence is famous because of its many interesting properties. Here is a small list (without proofs):

- a. Every positive integer can be written uniquely as a sum of 1 or more distinct, nonconsecutive Fibonacci numbers. For example $11 = 3 + 8$, where $3 = F(4)$ and $8 = F(6)$.
- b. $\gcd(F(p), F(q)) = F(\gcd(p, q))$. For example, if $p = 6$ and $q = 9$, then $F(6) = 8$, $F(9) = 34$, and $\gcd(8, 34) = 2$. Also, $\gcd(6, 9) = 3$ and $F(3) = 2$.
- c. Every two consecutive Fibonacci numbers are relatively prime, that is, their greatest common divisor equals 1. As a result, the Euclidean algorithm to find $\gcd(a, b)$ does the maximum amount of work when a and b are two consecutive Fibonacci numbers.

Other mathematical properties of the Fibonacci sequence are given in Example 3 and in the exercises at the end of this section. But it's not only mathematicians who are interested in the Fibonacci sequence. Fibonacci numbers often occur in nature. The number of petals on a daisy is often a Fibonacci number. Viewing a pine cone from its base, the seeds appear to be arranged in clockwise and counterclockwise spirals. Counting the number of each kind of spiral often gives two consecutive Fibonacci numbers (here 8 and 13). The same is true for seeds in flowers such as sunflowers, or for spirals on pineapples.



And in the worlds of art and architecture, the golden ratio is thought to create aesthetically pleasing proportions. The *golden ratio* is

$$\frac{1 + \sqrt{5}}{2} \approx 1.6180339$$

and is the value approached by the ratio of two consecutive Fibonacci numbers $F(n + 1)/F(n)$ for larger and larger values of n .

EXAMPLE 3

Prove that in the Fibonacci sequence

$$F(n + 4) = 3F(n + 2) - F(n) \text{ for all } n \geq 1$$

Because we want to prove something true for all $n \geq 1$, it is natural to think of a proof by induction. And because the value of $F(n)$ depends on both $F(n - 1)$ and $F(n - 2)$, the second principle of induction should be used. For the basis step of the inductive proof, we'll prove two cases, $n = 1$ and $n = 2$. For $n = 1$, by substituting 1 for n in the equation we want to prove, we get

$$F(5) = 3F(3) - F(1)$$

or (using values computed in Practice 2)

$$5 = 3(2) - 1$$

which is true. For $n = 2$,

$$F(6) = 3F(4) - F(2)$$

or

$$8 = 3(3) - 1$$

which is also true. Assume that for all r , $1 \leq r \leq k$,

$$F(r + 4) = 3F(r + 2) - F(r).$$

Now show the case for $k + 1$, where $k + 1 \geq 3$. (We've already proved the case for $n = 1$ and the case for $n = 2$.) Thus we want to show

$$F(k + 1 + 4) \stackrel{?}{=} 3F(k + 1 + 2) - F(k + 1)$$

or

$$F(k + 5) \stackrel{?}{=} 3F(k + 3) - F(k + 1)$$

From the recurrence relation for the Fibonacci sequence, we have

$$F(k + 5) = F(k + 3) + F(k + 4) \quad (F \text{ at any value is the sum of } F \text{ at the two previous values})$$

and by the inductive hypothesis, with $r = k - 1$ and $r = k$, respectively,

$$F(k + 3) = 3F(k + 1) - F(k - 1)$$

and

$$F(k + 4) = 3F(k + 2) - F(k)$$

Therefore

$$\begin{aligned} F(k + 5) &= F(k + 3) + F(k + 4) \\ &= [3F(k + 1) - F(k - 1)] + [3F(k + 2) - F(k)] \\ &= 3[F(k + 1) + F(k + 2)] - [F(k - 1) + F(k)] \\ &= 3F(k + 3) - F(k + 1) \quad (\text{using the recurrence relation again}) \end{aligned}$$

This completes the inductive proof. ●

PRACTICE 3 In the inductive proof of Example 3, why is it necessary to prove $n = 2$ as a special case? ■

EXAMPLE 4 The formula

$$F(n + 4) = 3F(n + 2) - F(n) \text{ for all } n \geq 1$$

of Example 3 can also be proved without induction, using just the recurrence relation from the definition of Fibonacci numbers. The recurrence relation

$$F(n + 2) = F(n) + F(n + 1)$$

can be rewritten as

$$F(n + 1) = F(n + 2) - F(n) \quad (1)$$

Then

$$\begin{aligned}
 F(n + 4) &= F(n + 3) + F(n + 2) \\
 &= F(n + 2) + F(n + 1) + F(n + 2) && \text{(rewriting } F(n + 3)\text{)} \\
 &= F(n + 2) + [F(n + 2) - F(n)] + F(n + 2) && \text{(rewriting } F(n + 1)\text{)} \\
 & && \text{using (1)} \\
 &= 3F(n + 2) - F(n)
 \end{aligned}$$

Recursively Defined Sets

The objects in a sequence are ordered—there is a first object, a second object, and so on. A set of objects is a collection of objects on which no ordering is imposed. Some sets can be defined recursively.

EXAMPLE 5

In Section 1.1 we noted that certain strings of statement letters, logical connectives, and parentheses, such as $(A \wedge B)' \vee C$, are considered legitimate, while other strings, such as $\wedge \wedge A''B$, are not legitimate. The syntax for arranging such symbols constitutes the definition of the set of propositional well-formed formulas, and it is a recursive definition.

1. Any statement letter is a wff.
2. If P and Q are wffs, so are $(P \wedge Q)$, $(P \vee Q)$, $(P \rightarrow Q)$, (P') and $(P \leftrightarrow Q)$.²

Using the rules of precedence for logical connectives established in Section 1.1, we can omit parentheses when doing so causes no confusion. Thus we write $(P \vee Q)$ as $P \vee Q$, or (P') as P' ; the new expressions are technically not wffs by the definition just given, but they unambiguously represent wffs.

By beginning with statement letters and repeatedly using rule 2, any propositional wff can be built. For example, A , B , and C are all wffs by rule 1. By rule 2,

$$(A \wedge B) \text{ and } (C')$$

are both wffs. By rule 2 again,

$$((A \wedge B) \rightarrow (C'))$$

is a wff. Applying rule 2 yet again, we get the wff

$$(((A \wedge B) \rightarrow (C'))')$$

Eliminating some pairs of parentheses, we can write this wff as

$$((A \wedge B) \rightarrow C)'$$

²Sometimes there is a final rule added to the effect that there are no applicable rules besides those already given, which means that if something can't be generated using the rules already given, then it does not belong to the set being described. We'll assume that when we stop writing rules, there are no more applicable rules!

PRACTICE 4 Show how to build the wff $((A \vee (B')) \rightarrow C)$ from the definition in Example 5. ■

PRACTICE 5 A recursive definition for the set of people who are ancestors of James could have the following basis:

James's parents are ancestors of James.

Give the inductive step. ■

Strings of symbols drawn from a finite “alphabet” set are objects that are commonly encountered in computer science. Computers store data as **binary strings**, strings from the alphabet consisting of 0s and 1s; compilers view program statements as strings of *tokens*, such as key words and identifiers. The collection of all finite-length strings of symbols from an alphabet, usually called strings *over* an alphabet, can be defined recursively (see Example 6). Many sets of strings with special properties also have recursive definitions.

EXAMPLE 6 The set of all (finite-length) strings of symbols over a finite alphabet A is denoted by A^* . The recursive definition of A^* is

1. The **empty string** λ (the string with no symbols) belongs to A^* .
2. Any single member of A belongs to A^* .
3. If x and y are strings in A^* , so is xy , the **concatenation** of strings x and y .

Parts 1 and 2 constitute the basis, and part 3 is the recursive step of this definition. Note that for any string x , $x\lambda = \lambda x = x$. ■

PRACTICE 6 If $x = 1011$ and $y = 001$, write the strings xy , yx , and $yx\lambda x$. ■

PRACTICE 7 Give a recursive definition for the set of all binary strings that are **palindromes**, strings that read the same forward and backward. ■

EXAMPLE 7 Suppose that in a certain programming language, identifiers can be alphanumeric strings of arbitrary length but must begin with a letter. A recursive definition for the set of such strings is

1. A single letter is an identifier.
2. If A is an identifier, so is the concatenation of A and any letter or digit.

A more symbolic notation for describing sets of strings that are recursively defined is called **Backus–Naur form**, or **BNF**, originally developed to define the

programming language ALGOL. In BNF notation, items that are defined in terms of other items are enclosed in angle brackets, while specific items that are not further broken down do not appear in brackets. The vertical line $|$ denotes a choice, with the same meaning as the English word *or*. The BNF definition of an identifier is

$$\begin{aligned} \langle \text{identifier} \rangle &::= \langle \text{letter} \rangle \mid \langle \text{identifier} \rangle \langle \text{letter} \rangle \mid \langle \text{identifier} \rangle \langle \text{digit} \rangle \\ \langle \text{letter} \rangle &::= a \mid b \mid c \mid \cdots \mid z \\ \langle \text{digit} \rangle &::= 1 \mid 2 \mid \cdots \mid 9 \end{aligned}$$

Thus the identifier `me2` is built from the definition by a sequence of choices such as

$\langle \text{identifier} \rangle$	can be	$\langle \text{identifier} \rangle \langle \text{digit} \rangle$
	which can be	$\langle \text{identifier} \rangle 2$
	which can be	$\langle \text{identifier} \rangle \langle \text{letter} \rangle 2$
	which can be	$\langle \text{identifier} \rangle e 2$
	which can be	$\langle \text{letter} \rangle e 2$
	which can be	<code>me2</code>

As a further connection between recursion and induction, there is a form of induction called **structural induction** that can be applied to recursively defined sets. Suppose we have a recursively defined set S and there is some property $P(x)$ that may or may not hold for x a member of S . If we can prove

1. Property P holds for all members of S described in the basis.
2. If property P holds for some members of S , then it holds for new members of S constructed from these members using the recursive step.

then property P holds for all members of S .

EXAMPLE 8

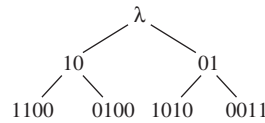
A set S of strings is defined recursively by

1. λ belongs to S .
2. If x belongs to S , so do $1x0$ and $0x1$.

We can use structural induction to prove that every string in S consists of an equal number of 0s and 1s. The basis, rule 1, identifies only a single string in S , namely λ , which consists of an equal number of 0s and 1s (zero 0s and zero 1s). Assume that string x consists of an equal number of 0s and 1s. Using rule 2, the two new strings that can be constructed from x each add a single 1 and a single 0, so the number of 0s and the number of 1s has each been increased by 1 and they are still equal. By structural induction, every string in S has an equal number of 0s and 1s.

Notice that not all strings with an equal number of 0s and 1s belong to S . For example there is no way to generate the string `1001` using the given rules. ●

Ordinary mathematical induction proves properties about integer values, and the integers are ordered: $1, 2, \dots, k, k + 1, \dots$. A set, however, isn't necessarily ordered. If we consider the set S defined in Example 8, it looks like



and structural induction helps us deal with this “spread” of values in the set.

Recursively Defined Operations

Certain operations performed on objects can be defined recursively, as in Examples 9 and 10.

EXAMPLE 9

A recursive definition of the exponentiation operation a^n on a nonzero real number a , where n is a nonnegative integer, is

1. $a^0 = 1$
2. $a^n = (a^{n-1})a$ for $n \geq 1$

EXAMPLE 10

A recursive definition for multiplication of two positive integers m and n is

1. $m(1) = m$
2. $m(n) = m(n-1) + m$ for $n \geq 2$

PRACTICE 8

Let x be a string over some alphabet. Give a recursive definition for the operation x^n (concatenation of x with itself n times) for $n \geq 1$.

In Section 1.1, we defined the operation of logical disjunction on two statement letters. This definition can serve as the basis step for a recursive definition of the disjunction of n statement letters, $n \geq 2$:

1. $A_1 \vee A_2$ defined as in Section 1.1
 2. $A_1 \vee \cdots \vee A_n = (A_1 \vee \cdots \vee A_{n-1}) \vee A_n$ for $n > 2$
- (2)

Using this definition, we can generalize the associative property of disjunction (tautological equivalence 2a) to say that in a disjunction of n statement letters, grouping by parentheses is unnecessary because all such groupings are equivalent to the general expression for the disjunction of n statement letters. In symbolic form, for any n with $n \geq 3$ and any p with $1 \leq p \leq n-1$,

$$(A_1 \vee \cdots \vee A_p) \vee (A_{p+1} \vee \cdots \vee A_n) \Leftrightarrow A_1 \vee \cdots \vee A_n$$

This equivalence can be proved by induction on n . For $n = 3$,

$$\begin{aligned} A_1 \vee (A_2 \vee A_3) &\Leftrightarrow (A_1 \vee A_2) \vee A_3 && \text{(by equivalence 2a)} \\ &= A_1 \vee A_2 \vee A_3 && \text{(by equation (2))} \end{aligned}$$

Assume that for $n = k$ and $1 \leq p \leq k - 1$,

$$(A_1 \vee \cdots \vee A_p) \vee (A_{p+1} \vee \cdots \vee A_k) \Leftrightarrow A_1 \vee \cdots \vee A_k$$

Then for $n = k + 1$ and $1 \leq p \leq k$,

$$\begin{aligned} & (A_1 \vee \cdots \vee A_p) \vee (A_{p+1} \vee \cdots \vee A_{k+1}) \\ &= (A_1 \vee \cdots \vee A_p) \vee [(A_{p+1} \vee \cdots \vee A_k) \vee A_{k+1}] && \text{(by equation (2))} \\ &\Leftrightarrow [(A_1 \vee \cdots \vee A_p) \vee (A_{p+1} \vee \cdots \vee A_k)] \vee A_{k+1} && \text{(by equivalence 2a)} \\ &\Leftrightarrow (A_1 \vee \cdots \vee A_k) \vee A_{k+1} && \text{(by inductive hypothesis)} \\ &= A_1 \vee \cdots \vee A_{k+1} && \text{(by equation (2))} \end{aligned}$$

Recursively Defined Algorithms

Example 1 gives a recursive definition for a sequence S . Suppose we want to write a computer program to evaluate $S(n)$ for some positive integer n . We can use either of two approaches. If we want to find $S(12)$, for example, we can begin with $S(1) = 2$ and then compute $S(2)$, $S(3)$, and so on, much as we did in Example 1, until we finally get to $S(12)$. This approach no doubt involves iterating through some sort of loop. A pseudocode function S that uses this iterative algorithm follows. The basis, where $n = 1$, is handled in the first clause of the **if** statement; the value 2 is returned. The **else** clause, for $n > 1$, does some initializing and then goes into the **while** loop that computes larger values of the sequence until the correct upper limit is reached. You can trace the execution of this algorithm for a few values of n to convince yourself that it works.

ALGORITHM

```

S(positive integer n)
//function that iteratively computes the value S(n)
//for the sequence S of Example 1
Local variables:
integer i      //loop index
CurrentValue //current value of function S

if n = 1 then
    return 2
else
    i = 2
    CurrentValue = 2
    while i <= n do
        CurrentValue = 2 * CurrentValue
        i = i + 1
    end while

    //CurrentValue now has the value S(n)
    return CurrentValue
end if
end function S

```

The second approach to computing $S(n)$ uses the recursive definition of S directly. Following is a version of the *recursive algorithm*, written again as a pseudocode function.

ALGORITHM

```
S(positive integer  $n$ )
//function that recursively computes the value  $S(n)$ 
//for the sequence  $S$  of Example 1

  if  $n = 1$  then
    return 2
  else
    return  $2 * S(n - 1)$ 
  end if
end function  $S$ 
```

The body of this function consists of a single **if-then-else** statement. To understand how the function works, let's trace the execution to compute the value of $S(3)$. The function is first invoked with an input value of $n = 3$. Because n is not 1, execution is directed to the **else** clause. At this point, activity on computing $S(3)$ must be suspended until the value of $S(2)$ is known. Any known information relevant to the computation of $S(3)$ is stored within computer memory on a stack, to be retrieved when the computation can be completed. (A stack is a collection of data where any new item goes on top of the stack, and only the item on top of the stack at any given time can be accessed or removed from the stack. A stack is thus a LIFO—last in, first out—structure.) The function is invoked again with an input value of $n = 2$. Again, the **else** clause is executed, and computation of $S(2)$ is suspended, with relevant information stored on the stack, while the function is invoked again with $n = 1$ as input.

This time the first clause of the **if** statement applies, and the functional value, 2, can be computed directly. This final invocation of the function is now complete, and its value of 2 is returned to the second-to-last invocation, which can now remove any information relevant to the $n = 2$ case from the stack, compute $S(2)$, and return the result to the previous (initial) invocation. Finally, this original invocation of S is able to empty the stack and complete its calculation, returning the value of $S(3)$.

What are the relative advantages of iterative and recursive algorithms for doing the same task? In this example, the recursive version is certainly shorter because it does not have to manage a loop computation. Describing the execution of the recursive version makes it sound more complex than the iterative version, but all steps are carried out automatically. One need not be aware of what is happening internally except to note that a long series of recursive invocations can use a lot of memory by storing information relevant to previous invocations on the stack. If too much memory is consumed, a “stack overflow” can result. Besides using more memory, recursive algorithms can require many more computations and can run more slowly than nonrecursive ones (see Exercise 3 in On the Computer at the end of this chapter).

Nonetheless, recursion provides a natural way to think about many problems, some of which would have very complex nonrecursive solutions. The problem of computing values for a sequence that has itself been defined recursively is well-suited to a recursive solution. Many programming languages support recursion.

PRACTICE 9 Write the body of a recursive function to compute $T(n)$ for the sequence T defined in Practice 1. ■

EXAMPLE 11 In Example 10, a recursive definition was given for multiplying two positive integers m and n . A recursive pseudocode function for multiplication based on this definition follows. ●

ALGORITHM

```

Product(positive integer  $m$ ; positive integer  $n$ )
//Function that recursively computes the product of  $m$  and  $n$ 

    if  $n = 1$  then
        return  $m$ ;
    else
        return Product( $m, n - 1$ ) +  $m$ 
    end if
end function Product

```

REMINDER

Think of a recursive algorithm whenever you could solve the problem from solutions to smaller versions of the problem.

A recursive algorithm invokes itself with “smaller” input values. Suppose a problem can be solved by solving smaller versions of the same problem, and the smaller versions eventually become trivial cases that are easily handled. Then a recursive algorithm can be useful, even if the original problem was not stated recursively.

To convince ourselves that a given recursive algorithm works, we don’t have to start with a particular input and go down through smaller and smaller cases to the trivial case and then back up again. We did this when discussing the computation of $S(3)$, but that was just to illustrate the mechanics of a recursive computation. Instead, we can verify the trivial case (like proving the base case in an induction proof) and verify that if the algorithm works correctly when invoked on smaller input values, then it indeed solves the problem for the original input values (this is similar to proving $P(k + 1)$ from the assumption $P(k)$ in an inductive proof).

EXAMPLE 12 One of the most common tasks in data processing is to sort a list L of n items into increasing or decreasing numerical or alphabetical order. (The list might consist of customer names, for example, and in sorted order “Valdez, Juanita” should come after “Tucker, Joseph.”) The **selection sort** algorithm—a simple but not particularly efficient sorting algorithm—is described in pseudocode in the accompanying box.

This function sorts the first j items in L into increasing order; when the function is initially invoked, j has the value n (thus, the first invocation ultimately sorts the entire list). The recursive part of the algorithm lies within the **else** clause; the algorithm examines the section of the list under consideration and finds the location i such that $L[i]$ is the maximum value. It then exchanges $L[i]$ and $L[j]$, after which the maximum value occurs at position j , the last position in the part of the list being considered. $L[j]$ is now correct and should never change again, so this

process is repeated on the list $L[1]$ through $L[j - 1]$. If this part of the list is sorted correctly, then the entire list will be sorted correctly. Whenever j has the value 1, the part of the list being considered consists of only one entry, which must be in the right place. The entire list is sorted at that point. ●

ALGORITHM SELECTIONSORT

```

SelectionSort(list  $L$ ; positive integer  $j$ )
//recursively sorts the items from 1 to  $j$  in list  $L$  into increasing order

    if  $j = 1$  then
        sort is complete, write out the sorted list
    else
        find the index  $i$  of the maximum item in  $L$  between 1 and  $j$ 
        exchange  $L[i]$  and  $L[j]$ 
        SelectionSort( $L, j - 1$ )
    end if
end function SelectionSort

```

Other recursive sorting algorithms are discussed in the exercises of Section 3.3.

EXAMPLE 13

Now that we have sorted our list, another common task is to search the list for a particular item. (Is Juanita Valdez already a customer?) An efficient search technique for a sorted list is the recursive **binary search algorithm**, which is described here in pseudocode.

ALGORITHM BINARYSEARCH

```

BinarySearch(list  $L$ ; positive integer  $i$ ; positive integer  $j$ ; itemtype  $x$ )
//searches sorted list  $L$  from  $L[i]$  to  $L[j]$  for item  $x$ 

    if  $i > j$  then
        write("not found")
    else
        find the index  $k$  midway between  $i$  and  $j$ 
        if  $x =$  midpoint item  $L[k]$  then
            write("found")
        else
            if  $x <$  midpoint item  $L[k]$  then
                BinarySearch( $L, i, k - 1, x$ )
            else
                BinarySearch( $L, k + 1, j, x$ )
            end if
        end if
    end if
end function BinarySearch

```

This algorithm searches the section of list L between $L[i]$ and $L[j]$ for item x ; initially i and j have the values 1 and n , respectively. The first clause of the major **if** statement is the basis step that says x cannot be found in an empty list, one where the first index exceeds the last index. In the major **else** clause, the midpoint item in a section of the list must be found. (If the section contains an odd number of items, there is indeed a midpoint item; if the section contains an even number of items, it is sufficient to take as the “midpoint” item the one at the end of the first half of the list section.) Comparing x with the midpoint item either locates x or indicates which half of the list to search next, the half before the midpoint or the half after the midpoint. ●

EXAMPLE 14

Let’s apply the binary search algorithm to the list

3, 7, 8, 10, 14, 18, 22, 34

where the target item x is the number 25. The initial list is not empty, so the midpoint item is located and determined to have the value 10. Then x is compared with the midpoint item. Because $x > 10$, the search is invoked on the second half of the list, namely, the items

14, 18, 22, 34

Again, this list is nonempty, and the midpoint item is 18. Because $x > 18$, the second half of this list is searched, namely, the items

22, 34

In this nonempty list, the midpoint item is 22. Because $x > 22$, the search continues on the second half of the list, namely,

34

This is a one-element list, with the midpoint item being the only item. Because $x < 34$, a search is begun on the “first half” of the list; but the first half is empty. The algorithm terminates at this point with the information that x is not in the list.

This execution requires four comparisons in all; x is compared, in turn, to 10, 18, 22, and 34. ●

PRACTICE 10

In a binary search of the list in Example 14, name the elements against which x is compared if x has the value 8. ■


We have now seen a number of recursive definitions. Table 3.1 summarizes their features.

TABLE 3.1

Recursive Definitions	
What Is Being Defined	Characteristics
Recursive sequence	The first one or two values in the sequence are known; later items in the sequence are defined in terms of earlier items.
Recursive set	A few specific items are known to be in the set; other items in the set are built from combinations of items already in the set.
Recursive operation	A “small” case of the operation gives a specific value; other cases of the operation are defined in terms of smaller cases.
Recursive algorithm	For the smallest values of the arguments, the algorithm behavior is known; for larger values of the arguments, the algorithm invokes itself with smaller argument values.

SECTION 3.1 REVIEW

TECHNIQUES

- Generate values in a sequence defined recursively.
- Prove properties of the Fibonacci sequence.
-  Recognize objects in a recursively defined collection of objects.
- Give recursive definitions for particular sets of objects.
- Give recursive definitions for certain operations on objects.
- Write recursive algorithms to generate sequences defined recursively.

MAIN IDEAS

- Recursive definitions can be given for sequences of objects, sets of objects, and operations on objects where basis information is known and new information depends on already known information.
- Recursive algorithms provide a natural way to solve certain problems by invoking the same task on a smaller version of the problem.

EXERCISES 3.1

For Exercises 1–12, write the first five values in the sequence.

1. $S(1) = 10$

$$S(n) = S(n - 1) + 10 \text{ for } n \geq 2$$

2. $C(1) = 5$

$$C(n) = 2C(n - 1) + 5 \text{ for } n \geq 2$$

3. $A(1) = 2$

$$A(n) = \frac{1}{A(n - 1)} \text{ for } n \geq 2$$

4. $B(1) = 1$

$$B(n) = B(n - 1) + n^2 \text{ for } n \geq 2$$

5. $S(1) = 1$

$$S(n) = S(n - 1) + \frac{1}{n} \text{ for } n \geq 2$$

6. $T(1) = 1$

$$T(n) = nT(n - 1) \text{ for } n \geq 2$$

7. $P(1) = 1$
 $P(n) = n^2 P(n - 1) + (n - 1)$ for $n \geq 2$
8. $A(1) = 2$
 $A(n) = nA(n - 1) + n$ for $n \geq 2$
9. $M(1) = 2$
 $M(2) = 2$
 $M(n) = 2M(n - 1) + M(n - 2)$ for $n > 2$
10. $D(1) = 3$
 $D(2) = 5$
 $D(n) = (n - 1)D(n - 1) + (n - 2)D(n - 2)$ for $n > 2$
11. $W(1) = 2$
 $W(2) = 3$
 $W(n) = W(n - 1)W(n - 2)$ for $n > 2$
12. $T(1) = 1$
 $T(2) = 2$
 $T(3) = 3$
 $T(n) = T(n - 1) + 2T(n - 2) + 3T(n - 3)$ for $n > 3$

In Exercises 13–18, prove the given property of the Fibonacci numbers directly from the definition.

13. $F(n + 1) + F(n - 2) = 2F(n)$ for $n \geq 3$
14. $F(n) = 5F(n - 4) + 3F(n - 5)$ for $n \geq 6$
15. $F(n) = 3F(n - 3) + 2F(n - 4)$ for $n \geq 5$
16. $[F(n + 1)]^2 = [F(n)]^2 + F(n - 1)F(n + 2)$ for $n \geq 2$
17. $F(n + 3) = 2F(n + 1) + F(n)$ for $n \geq 1$
18. $F(n + 6) = 4F(n + 3) + F(n)$ for $n \geq 1$

In Exercises 19–22, prove the given property of the Fibonacci numbers for all $n \geq 1$. (*Hint:* The first principle of induction will work.)

19. $F(1) + F(2) + \cdots + F(n) = F(n + 2) - 1$
20. $F(2) + F(4) + \cdots + F(2n) = F(2n + 1) - 1$
21. $F(1) + F(3) + \cdots + F(2n - 1) = F(2n)$
22. $[F(1)]^2 + [F(2)]^2 + \cdots + [F(n)]^2 = F(n)F(n + 1)$

In Exercises 23–26, prove the given property of the Fibonacci numbers using the second principle of induction.

23. Exercise 17
24. Exercise 18
25. $F(n) < 2^n$ for $n \geq 1$
26. $F(n) > \left(\frac{3}{2}\right)^{n-1}$ for $n \geq 6$

27. Write a pseudocode recursive algorithm for a function to compute $F(n)$, the n th Fibonacci number.
28. Walk through your recursive algorithm from Exercise 27 to compute $F(6)$.
- How many times is the function invoked?
 - How many times is $F(4)$ computed?
 - How many times is $F(3)$ computed?
 - How many times is $F(2)$ computed?

Exercises 29 and 30 concern a proof of correctness of the following iterative algorithm for a function to compute $F(n)$, the n th Fibonacci number.

```

F(positive integer  $n$ )
//function that iteratively computes the value of
//the  $n$ th Fibonacci number
Local variables:
positive integer  $i$            //loop index
positive integers  $p, q, r$     //terms in Fibonacci sequence

if  $n = 1$  then
    return 1
else
    if  $n = 2$  then
        return 1
    else
         $i = 2$ 
         $p = 1$            // $p$  = lagging term in Fibonacci sequence
         $q = 1$            // $q$  = leading term in Fibonacci sequence
        while  $i < n$  do
             $r = p + q$     //form the next term as the
                        //sum of the two previous terms
             $p = q$        //bump up  $p$ 
             $q = r$        //bump up  $q$ 
             $i = i + 1$ 
        end while
        // $q$  now has the value  $F(n)$ 
        return  $q$ 
    end if
end if
end function  $F$ 

```

29. a. In the iterative Fibonacci algorithm, the condition B for loop continuation is $i < n$, so B' is $i \geq n$, but what is the exact value of i when the loop terminates?
- b. When the loop exits, you want $q = F(n)$; what do you want for the value of p at that point?
30. a. Write the loop invariant Q for the iterative Fibonacci algorithm.
- b. Prove that Q is a loop invariant.

31. The values p and q are defined as follows:

$$p = \frac{1 + \sqrt{5}}{2} \quad \text{and} \quad q = \frac{1 - \sqrt{5}}{2}$$

a. Prove that $1 + p = p^2$ and $1 + q = q^2$.

b. Prove that

$$F(n) = \frac{p^n - q^n}{p - q}$$

c. Use part (b) to prove that

$$F(n) = \frac{\sqrt{5}}{5} \left(\frac{1 + \sqrt{5}}{2} \right)^n - \frac{\sqrt{5}}{5} \left(\frac{1 - \sqrt{5}}{2} \right)^n$$

is a closed-form solution for the Fibonacci sequence.

32. The *Lucas sequence* is defined by

$$L(1) = 1$$

$$L(2) = 3$$

$$L(n) = L(n - 1) + L(n - 2) \text{ for } n \geq 2$$

a. Write the first five terms of the sequence.

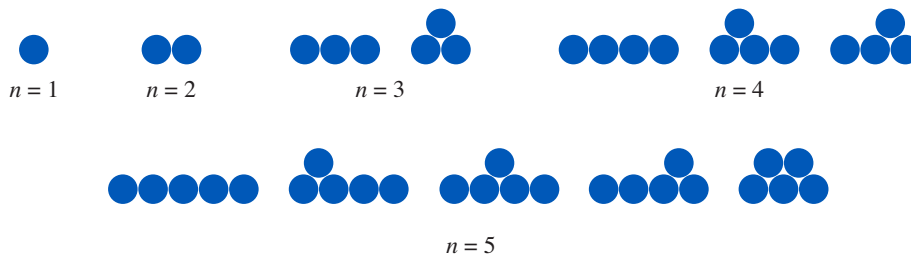
b. Prove that $L(n) = F(n + 1) + F(n - 1)$ for $n \geq 2$ where F is the Fibonacci sequence.

For Exercises 33–36, decide whether the sequences described are subsequences of the Fibonacci sequence, that is, whether their members are some or all of the members, in the right order, of the Fibonacci sequence.³

33. The sequence $A(n)$, where $A(n) = 1 +$ (the sum of the first n terms of the Fibonacci sequence), $n \geq 1$. The first four values are 2, 3, 5, 8, which—so far—form a subsequence of the Fibonacci sequence.

34. The sequence $B(n)$, where $B(n) = (n - 1)2^{n-2} + 1$, $n \geq 1$. The first four values are 1, 2, 5, 13, which—so far—form a subsequence of the Fibonacci sequence.

35. The sequence $C(n)$, where $C(n)$ is the number of ways in which n coins can be arranged in horizontal rows with all the coins in each row touching and every coin above the bottom row touching two coins in the row below it, $n \geq 1$. The first five values are 1, 1, 2, 3, 5, which—so far—form a subsequence of the Fibonacci sequence.



36. The sequence $D(n)$, where $D(n)$ describes the number of ways to paint the floors on an n -story building where each floor is painted yellow or blue and no two adjacent floors can be blue (although adjacent floors

³Exercises 33–36 are taken from “Mathematical Recreations” by Ian Stewart, *Scientific American*, May 1995.

can be yellow), $n \geq 1$. The first four values are 2, 3, 5, 8, which—so far—form a subsequence of the Fibonacci sequence. For example, $D(3) = 5$ because a three-story building can be painted

Y	Y	Y	B	B
Y	Y	B	Y	Y
Y	B	Y	B	Y

(Hint: think about a recursive expression for $D(n + 1)$.)

- 37 a. The original problem posed by Fibonacci concerned pairs of rabbits. Two rabbits do not breed until they are 2 months old. After that, each pair of rabbits produces a new pair each month. No rabbits ever die. Let $R(n)$ denote the number of rabbit pairs at the end of n months if you start with a single rabbit pair. Show that $R(n)$ is the Fibonacci sequence.
- b. Write 27 and 62 as the sum of distinct nonconsecutive Fibonacci numbers.
38. a. The sequence of *Catalan numbers* is defined recursively by

$$C(0) = 1$$

$$C(1) = 1$$

$$C(n) = \sum_{k=1}^n C(k-1)C(n-k) \text{ for } n \geq 2$$

Compute the values of $C(2)$, $C(3)$, and $C(4)$ using this recurrence relation.

- b. Frank and Jody are both candidates for president of the County Council. The number of votes cast equals $2n$, where n votes are cast for Frank and n for Jody. Votes are counted sequentially. The *ballot problem* asks: In how many ways can the votes be counted so that Jody's total is never ahead of Frank's total? The answer, as it turns out, is $C(n)$, the n th Catalan number. For example, if $n = 5$, one possible counting sequence that meets this requirement is

FFJFFJFFJJ

Using $n = 3$, write down all the satisfactory counting sequences and compare the result to the Catalan number $C(3)$.

39. A sequence is recursively defined by

$$S(1) = 2$$

$$S(2) = 2$$

$$S(3) = 6$$

$$S(n) = 3S(n-3) \text{ for } n \geq 3$$

Prove that $S(n)$ is an even number for $n \geq 1$.

40. A sequence is recursively defined by

$$T(5) = 6$$

$$T(6) = 10$$

$$T(n) = 2T(n-2) + 2 \text{ for } n \geq 7$$

Prove that $T(n) \geq 2n$ for $n \geq 7$.

41. A sequence is recursively defined by

$$S(0) = 1$$

$$S(1) = 1$$

$$S(n) = 2S(n - 1) + S(n - 2) \text{ for } n \geq 2$$

- a. Prove that $S(n)$ is an odd number for $n \geq 0$.
- b. Prove that $S(n) < 6S(n - 2)$ for $n \geq 4$.

42. A sequence is recursively defined by

$$T(0) = 1$$

$$T(1) = 2$$

$$T(n) = 2T(n - 1) + T(n - 2) \text{ for } n \geq 2$$

Prove that $T(n) \leq \left(\frac{5}{2}\right)^n$ for $n \geq 0$.

43. Write a recursive definition for a geometric progression with initial term a and common ratio r (see Exercise 27, Section 2.2.).
44. Write a recursive definition for an arithmetic progression with initial term a and common difference d (see Exercise 28, Section 2.2.).
45. In an experiment, a certain colony of bacteria initially has a population of 50,000. A reading is taken every 2 hours, and at the end of every 2-hour interval, there are 3 times as many bacteria as before.
 - a. Write a recursive definition for $A(n)$, the number of bacteria present at the beginning of the n th time period.
 - b. At the beginning of which interval are there 1,350,000 bacteria present?
46. An amount of \$500 is invested in an account paying 1.2% interest compounded annually.
 - a. Write a recursive definition for $P(n)$, the amount in the account at the beginning of the n th year.
 - b. After how many years will the account balance exceed \$570?
47. A set T of numbers is defined recursively by
 1. 2 belongs to T .
 2. If x belongs to T , so does $x + 3$ and $2 * x$.

Which of the following numbers belong to T ?

 - a. 6 b. 7 c. 19 d. 12

48. A set M of numbers is defined recursively by

1. 2 and 3 belong to M .
2. If x and y belong to M , so does $x * y$.

Which of the following numbers belong to M ?

- a. 6 b. 9 c. 16 d. 21 e. 26 f. 54 g. 72 h. 218

49. A set S of strings of characters is defined recursively by

1. a and b belong to S .
2. If x belongs to S , so does xb .

Which of the following strings belong to S ?

- a. a b. ab c. aba d. $aaab$ e. $bbbbbb$

50. A set W of strings of symbols is defined recursively by

1. a , b , and c belong to W .
2. If x belongs to W , so does $a(x)c$.

Which of the following strings belong to W ?

- a. $a(b)c$ b. $a(a(b)c)c$ c. $a(abc)c$ d. $a(a(a(a)c)c)c$ e. $a(aacc)c$

51. A set S of integers is defined recursively by

1. 0 and 3 belong to S .
2. If x and y belong to S , so does $x + y$.

Use structural induction to prove that every integer in S is a multiple of 3.

52. A set T of strings is defined recursively by

1. pqq belongs to T .
2. If x and y belong to T , so do $pxqq$, $qqxp$, and xy .

Use structural induction to prove that every string in T has twice as many q 's as p 's.

53. Give a recursive definition for the set of all unary predicate wffs in x .

54. Give a recursive definition for the set of all well-formed formulas of integer arithmetic, involving integers together with the arithmetic operations of $+$, $-$, $*$, and $/$.

55. Give a recursive definition for the set of all odd integers.

56. Give a recursive definition for the set of all strings of well-balanced parentheses.

57. Give a recursive definition for the set of all binary strings containing an odd number of 0s.

58. Give a recursive definition for the set of all binary strings containing an even number of 1s.

59. Give a recursive definition for the set of all binary strings ending with 0.

60. Give a recursive definition for the set of all binary strings with an equal number of 0s and 1s.

61. Use BNF notation to define the set of positive integers.

62. Use BNF notation to define the set of decimal numbers, which consist of an optional sign ($+$ or $-$), followed by one or more digits, followed by a decimal point, followed by zero or more digits.

63. Give a recursive definition for x^R , the reverse of the string x .

64. Give a recursive definition for $|x|$, the length of the string x .

65. Give a recursive definition for the factorial operation $n!$ for $n \geq 1$.

66. Give a recursive definition for the addition of two nonnegative integers m and n .
67. a. Write a recursive definition for the operation of taking the maximum of n integers $a_1, \dots, a_n, n \geq 2$.
 b. Write a recursive definition for the operation of taking the minimum of n integers $a_1, \dots, a_n, n \geq 2$.
68. a. Give a recursive definition for the conjunction of n statement letters in propositional logic, $n \geq 2$.
 b. Write a generalization of the associative property of conjunction (tautological equivalence 2b of Section 1.1) and use induction to prove it.
69. Let A and B_1, B_2, \dots, B_n be statement letters. Prove the finite extension of the distributive equivalences of propositional logic:

$$A \vee (B_1 \wedge B_2 \wedge \cdots \wedge B_n) \Leftrightarrow (A \vee B_1) \wedge (A \vee B_2) \wedge \cdots \wedge (A \vee B_n)$$

and

$$A \wedge (B_1 \vee B_2 \vee \cdots \vee B_n) \Leftrightarrow (A \wedge B_1) \vee (A \wedge B_2) \vee \cdots \vee (A \wedge B_n)$$

for $n \geq 2$.

70. Let B_1, B_2, \dots, B_n be statement letters. Prove the finite extension of De Morgan's laws:

$$(B_1 \vee B_2 \vee \cdots \vee B_n)' \Leftrightarrow B'_1 \wedge B'_2 \wedge \cdots \wedge B'_n$$

and

$$(B_1 \wedge B_2 \wedge \cdots \wedge B_n)' \Leftrightarrow B'_1 \vee B'_2 \vee \cdots \vee B'_n$$

for $n \geq 2$.

In Exercises 71–76, write the body of a recursive function to compute $S(n)$ for the given sequence S .

71. 1, 3, 9, 27, 81, ...

72. 2, 1, 1/2, 1/4, 1/8, ...

73. 1, 2, 4, 7, 11, 16, 22, ...

74. 2, 4, 16, 256, ...

75. $a, b, a + b, a + 2b, 2a + 3b, 3a + 5b, \dots$

76. $p, p - q, p + q, p - 2q, p + 2q, p - 3q, \dots$

77. What value is returned by the following recursive function *Mystery* for an input value of n ?

Mystery (positive integer n)

if $n = 1$ **then**

 return 1

else

 return *Mystery*($n - 1$) + 1

end if

end function *Mystery*

78. The following recursive function is initially invoked with an i value of 1. L is a list (array) of 10 integers. What does the function do?

$g(\text{list } L; \text{ positive integer } i; \text{ integer } x)$

if $i > 10$ **then**

 return 0

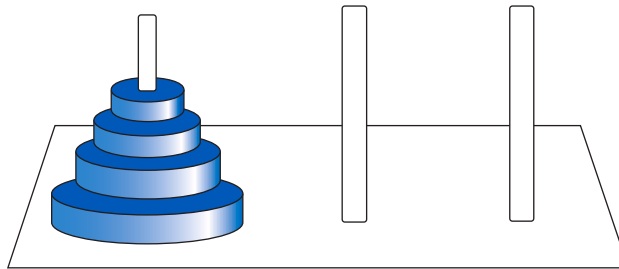
else

```

if  $L[i] = x$  then
    return 10
else
    return  $g(L, i + 1, x)$ 
end if
end if
end function  $g$ 

```

79. Informally describe a recursive algorithm to reverse the entries in a list of items.
80. Informally describe a recursive algorithm to compute the sum of the digits of a positive integer.
81. Informally describe a recursive algorithm to compute the greatest common divisor of two positive integers a and b where $a > b$. (*Hint*: The solution is based on the Euclidean algorithm, discussed in Section 2.3. In particular, make use of expression (5) on page 134.)
82. The famous Towers of Hanoi puzzle involves 3 pegs with n disks of varying sizes stacked in order from the largest (on the bottom) to the smallest (on the top) on 1 of the pegs. The puzzle requires that the disks end up stacked the same way on a different peg; only one disk at a time can be moved to another peg, and no disk can ever be stacked on top of a smaller disk. Informally describe a recursive algorithm to solve the Towers of Hanoi puzzle.



83. Simulate the execution of algorithm *SelectionSort* on the following list L ; write the list after every exchange that changes the list.
- 4, 10, -6, 2, 5
84. Simulate the execution of algorithm *SelectionSort* on the following list L ; write the list after every exchange that changes the list.
- 9, 0, 2, 6, 4
85. The binary search algorithm is used with the following list; x has the value “Chicago.” Name the elements against which x is compared.
- Boston, Charlotte, Indianapolis, New Orleans, Philadelphia, San Antonio, Yakima
86. The binary search algorithm is used with the following list; x has the value “flour.” Name the elements against which x is compared.
- butter, chocolate, eggs, flour, shortening, sugar
87. Do a proof of correctness for the iterative function given in this section to compute $S(n)$ of Example 1, $S(n) = 2^n$.
88. The *Online Encyclopedia of Integer Sequences* (OEIS) was originated and maintained for many years by Neil Sloane, a mathematician at AT&T who has also written several books about sequences. The OEIS Foundation now manages the database, which contains more than 200,000 sequences of integers that have

been submitted and studied by many people. (See oeis.org). There is even a YouTube movie about the OEIS! *Recaman's sequence* (number A005132 in the OEIS catalog) is a recursive sequence defined as follows:

$$a(1) = 1$$

For $n > 1$,

$$a(n) = \begin{cases} a(n-1) - n & \text{if that number is positive and not already in the sequence,} \\ \text{otherwise} \\ a(n-1) + n \end{cases}$$

- Confirm that the first few terms of this sequence are 1, 3, 6, 2, 7, 13.
- It has been conjectured that every nonnegative integer will eventually appear in this sequence. Find the index of this sequence at which the following numbers appear: 10, 12, 23.

SECTION 3.2 RECURRENCE RELATIONS

We developed two algorithms, one iterative and one recursive, to compute a value $S(n)$ for the sequence S of Example 1. However, there is a still easier way to compute $S(n)$. Recall that

$$S(1) = 2 \tag{1}$$

$$S(n) = 2S(n-1) \text{ for } n \geq 2 \tag{2}$$

Because

$$S(1) = 2 = 2^1$$

$$S(2) = 4 = 2^2$$

$$S(3) = 8 = 2^3$$

$$S(4) = 16 = 2^4$$

and so on, we can see that

$$S(n) = 2^n \tag{3}$$

Using Equation (3), we can plug in a value for n and compute $S(n)$ without having to compute—either explicitly, or, through recursion, implicitly—all the lower values of S first. An equation such as (3), where we can substitute a value and get the output value back directly, is called a **closed-form solution** to the recurrence relation (2) subject to the basis step (1). Finding a closed-form solution is called **solving** the recurrence relation.

Recurrence relations can be used to describe a variety of things, from chemical degradation (see the opening problem for this chapter) to the amount in an interest-bearing account, from the growth of species to the spread of a computer virus. Clearly, it is nice to find a closed-form solution to a recurrence relation whenever possible.

Linear First-Order Recurrence Relations

Expand, Guess, and Verify

One technique for solving recurrence relations is an “expand, guess, and verify” approach that repeatedly uses the recurrence relation to expand the expression for the n th term until the general pattern can be guessed. Finally the guess is verified by mathematical induction.

EXAMPLE 15

Consider again the basis step and recurrence relation for the sequence S of Example 1:

$$S(1) = 2 \quad (4)$$

$$S(n) = 2S(n - 1) \text{ for } n \geq 2 \quad (5)$$

REMINDER

Don't get hung up on " n " and " $n - 1$ " in the recurrence relation. Think of it as "S at some value is 2 times S at the previous value."

Let's pretend we don't already know the closed-form solution and use the expand, guess, and verify approach to find it. Beginning with $S(n)$, we expand by using the recurrence relation repeatedly. Keep in mind that the recurrence relation is a recipe that says S at any value can be replaced by two times S at the previous value. We apply this recipe to S at the values $n, n - 1, n - 2$, and so on:

$$\begin{aligned} S(n) &= 2S(n - 1) \\ &= 2[2S(n - 2)] = 2^2S(n - 2) \\ &= 2^2 [2S(n - 3)] = 2^3S(n - 3) \end{aligned}$$

By looking at the developing pattern, we guess that after k such expansions, the equation has the form

$$S(n) = 2^k S(n - k)$$

This expansion of S values in terms of lower S values must stop when $n - k = 1$, that is, when $k = n - 1$. At that point,

$$\begin{aligned} S(n) &= 2^{n-1} S[n - (n - 1)] \\ &= 2^{n-1} S(1) = 2^{n-1}(2) = 2^n \end{aligned}$$

which expresses the closed-form solution.

We are not yet done, however, because we guessed at the general pattern. We now confirm our closed-form solution by induction on the value of n . The statement we want to prove is therefore $S(n) = 2^n$ for $n \geq 1$.

For the basis step, $S(1) = 2^1$. This is true by equation (4). We assume that $S(k) = 2^k$. Then

$$\begin{aligned} S(k + 1) &= 2S(k) && \text{(by equation (5))} \\ &= 2(2^k) && \text{(by the inductive hypothesis)} \\ &= 2^{k+1} \end{aligned}$$

This proves that our closed-form solution is correct. ●

PRACTICE 11

Find a closed-form solution for the recurrence relation, subject to the basis step, for sequence T :

1. $T(1) = 1$
2. $T(n) = T(n - 1) + 3$ for $n \geq 2$

(Hint: Expand, guess, and verify.) ■

A Solution Formula

Some types of recurrence relations have known solution formulas. A recurrence relation for a sequence $S(n)$ is **linear** if the earlier values of S appearing in the definition occur only to the first power. The most general linear recurrence relation has the form

$$S(n) = f_1(n)S(n-1) + f_2(n)S(n-2) + \cdots + f_k(n)S(n-k) + g(n)$$

where the f_i 's and g can be expressions involving n . The recurrence relation has **constant coefficients** if the f_i 's are all constants. It is **first-order** if the n th term depends only on term $n-1$. Linear first-order recurrence relations with constant coefficients therefore have the form

$$S(n) = cS(n-1) + g(n) \quad (6)$$

Finally, a recurrence relation is **homogeneous** if $g(n) = 0$ for all n .

We will find the solution formula for equation (6), the general linear first-order recurrence relation with constant coefficients, subject to the basis that $S(1)$ is known. We will use the expand, guess, and verify approach. The work here is a generalization of what was done in Example 15. Repeatedly applying equation (6) and simplifying, we get

$$\begin{aligned} S(n) &= cS(n-1) + g(n) \\ &= c[cS(n-2) + g(n-1)] + g(n) \\ &= c^2S(n-2) + cg(n-1) + g(n) \\ &= c^2[cS(n-3) + g(n-2)] + cg(n-1) + g(n) \\ &= c^3S(n-3) + c^2g(n-2) + cg(n-1) + g(n) \\ &\quad \vdots \end{aligned}$$

After k expansions, the general form appears to be

$$S(n) = c^kS(n-k) + c^{k-1}g(n-(k-1)) + \cdots + cg(n-1) + g(n)$$

If the sequence has a base value at 1, then the expansion terminates when $n-k=1$ or $k=n-1$, at which point

$$\begin{aligned} S(n) &= c^{n-1}S(1) + c^{n-2}g(2) + \cdots + cg(n-1) + g(n) \\ &= c^{n-1}S(1) + c^{n-2}g(2) + \cdots + c^1g(n-1) + c^0g(n) \end{aligned} \quad (7)$$

We can use **summation notation** to write part of this expression more compactly. The uppercase Greek letter sigma, Σ , stands for summation. The notation

$$\sum_{i=p}^q (\text{expression})$$

says to substitute into the expression successive values of i , the **index of summation**, from the lower limit p to the upper limit q , and then sum the results. (See Appendix B for further discussion of summation notation.) Thus, for example,

$$\sum_{i=1}^n (2i-1) = 1 + 3 + 5 + \cdots + (2n-1)$$

In Example 14, Section 2.2, we proved by induction that the value of this summation is n^2 .

In summation notation, Equation (7) becomes

$$S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i)$$

Induction can be used, much as was done in Example 15, to verify that this formula is the solution to recurrence relation (6) (see Exercise 26).

Therefore, the solution to the recurrence relation (6) is

$$S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i) \quad (8)$$

This is not yet a closed-form solution, however, because we must find an expression for the summation. Usually it is either trivial to find the sum or we found its value in Section 2.2 using mathematical induction. (If we can't find an expression for the summation, we are really no better off than before. We must iterate through the summation to find the desired value as opposed to iterating through the recurrence relation to get the desired value.)

The work we've done here gives a general solution—Equation (8)—once and for all for any recurrence relation of the form shown in (6); this work *need not be repeated*. All that is necessary is to match your problem to equation (6) to find the value for c and the formula for $g(n)$ and then plug these results into the expression in (8). The $g(n)$ in Equation (6) is the usual notation for a function of n ; although we will study functions formally in Chapter 5, you can think of $g(n)$ as giving a “recipe” for what to do with its argument n . If, for example,

$$g(n) = 2n$$

then g doubles whatever its argument value is:

$$g(3) = 2(3) = 6 \quad g(27) = 2(27) = 54 \quad \text{and} \quad g(i) = 2i$$

This last value, $2i$, would be used in Equation (8) if $g(n) = 2n$.

EXAMPLE 16

The sequence $S(n)$ of Example 15,

$$\begin{aligned} S(1) &= 2 \\ S(n) &= 2S(n-1) \text{ for } n \geq 2 \end{aligned}$$

is a linear, first-order, homogeneous recurrence relation with constant coefficients. In other words, it matches equation (6) with $c = 2$ and $g(n) = 0$. Because $g(n) = 0$, the g function evaluates to 0 no matter what the argument is. From formula (8), the closed-form solution is

$$S(n) = 2^{n-1}(2) + \sum_{i=2}^n 0 = 2^n$$

which agrees with our previous result. ●

You now have a choice of two alternative ways to solve a linear, first-order recurrence relation with constant coefficients. Table 3.2 summarizes these approaches.

TABLE 3.2

To Solve Recurrence Relations of the Form $S(n) = cS(n - 1) + g(n)$ Subject to Basis $S(1)$

Method	Steps
Expand, guess, verify	<ol style="list-style-type: none"> 1. Repeatedly use the recurrence relation until you can guess a pattern. 2. Decide what that pattern will be when $n - k = 1$. 3. Verify the resulting formula by induction.
Solution formula	<ol style="list-style-type: none"> 1. Match your recurrence relation to the form $S(n) = cS(n - 1) + g(n)$ to find c and $g(n)$. 2. Use c, $g(n)$, and $S(1)$ in the formula $S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i)$ 3. Evaluate the resulting summation to get the final expression.

EXAMPLE 17

Find a closed-form solution to the recurrence relation

$$S(n) = 2S(n - 1) + 3 \text{ for } n \geq 2$$

subject to the basis step

$$S(1) = 4$$

We'll use the solution formula method. Comparing our recurrence relation

$$S(n) = 2S(n - 1) + 3$$

with the general form $S(n) = cS(n - 1) + g(n)$, we see that

$$c = 2 \quad g(n) = 3$$

The fact that $g(n) = 3$ says that g has a constant value of 3 no matter what the value of its argument; in particular, $g(i) = 3$. Substituting into the general solution form

$$S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i)$$

we get

$$\begin{aligned}
 S(n) &= 2^{n-1}(4) + \sum_{i=2}^n 2^{n-i}(3) \\
 &= 2^{n-1}(2^2) + 3 \sum_{i=2}^n 2^{n-i} \\
 &= 2^{n+1} + 3[2^{n-2} + 2^{n-3} + \dots + 2^1 + 2^0] \\
 &= 2^{n+1} + 3[2^{n-1} - 1] \quad (\text{from Example 15, Section 2.2})
 \end{aligned}$$

REMINDER

When expanding, be sure to pick up all the pieces of the recurrence relation recipe, like the + 3 in this example.

So the value of $S(5)$, for example, is $2^6 + 3(2^4 - 1) = 64 + 3(15) = 109$. Alternatively, by the expand, guess, and verify technique, we expand

$$\begin{aligned} S(n) &= 2S(n-1) + 3 \\ &= 2[2S(n-2) + 3] + 3 = 2^2S(n-2) + 2 \cdot 3 + 3 \\ &= 2^2[2S(n-3) + 3] + 2 \cdot 3 + 3 = 2^3S(n-3) + 2^2 \cdot 3 + 2 \cdot 3 + 3 \\ &\quad \vdots \end{aligned}$$

The general pattern seems to be

$$S(n) = 2^k S(n-k) + 2^{k-1} \cdot 3 + 2^{k-2} \cdot 3 + \cdots + 2^2 \cdot 3 + 2 \cdot 3 + 3$$

which, when $n - k = 1$ or $k = n - 1$, becomes

$$\begin{aligned} S(n) &= 2^{n-1}S(1) + 2^{n-2} \cdot 3 + 2^{n-3} \cdot 3 + \cdots + 2^2 \cdot 3 + 2 \cdot 3 + 3 \\ &= 2^{n-1}(4) + 3[2^{n-2} + 2^{n-3} + \cdots + 2^2 + 2 + 1] \\ &= 2^{n+1} + 3[2^{n-1} - 1] \quad (\text{from Example 15, Section 2.2}) \end{aligned}$$

Finally, we must prove by induction that $S(n) = 2^{n+1} + 3[2^{n-1} - 1]$.

Base case: $n = 1$: $S(1) = 4 = 2^2 + 3[2^0 - 1]$, true

Assume $S(k) = 2^{k+1} + 3[2^{k-1} - 1]$

Show $S(k+1) = 2^{k+2} + 3[2^k - 1]$

$$\begin{aligned} S(k+1) &= 2S(k) + 3 && \text{(by the recurrence relation)} \\ &= 2(2^{k+1} + 3[2^{k-1} - 1]) + 3 && \text{(by the inductive hypothesis)} \\ &= 2^{k+2} + 3 \cdot 2^k - 6 + 3 && \text{(multiplying out)} \\ &= 2^{k+2} + 3[2^k - 1] \end{aligned}$$

Practice 12

Rework Practice 11 using Equation (8).

EXAMPLE 18

Find a closed-form solution to the recurrence relation

$$T(n) = T(n-1) + (n+1) \text{ for } n \geq 2$$

subject to the basis step

$$T(1) = 2$$

Using the solution formula method and comparing the recurrence relation with the general form from Equation (6), $S(n) = cS(n-1) + g(n)$, we find that $c = 1$ and $g(n) = n + 1$. We'll substitute into the solution equation (8)

$$S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i), \text{ where } g(i) \text{ will be } i + 1.$$

$$\begin{aligned}
 T(n) &= (1)^{n-1}(2) + \sum_{i=2}^n (1)^{n-i}(i+1) \\
 &= 2 + \sum_{i=2}^n (i+1) \\
 &= 2 + (3 + 4 + \cdots + (n+1)) \\
 &= \frac{(n+1)(n+2)}{2} - 1 \qquad \text{(from Practice 7, Section 2.2)}
 \end{aligned}$$

EXAMPLE 19

Consider the problem of reading data from a computer disk drive.⁴ The circular drive is organized as a series of concentric tracks, divided into sectors. Each sector contains a block of data (Figure 3.1).

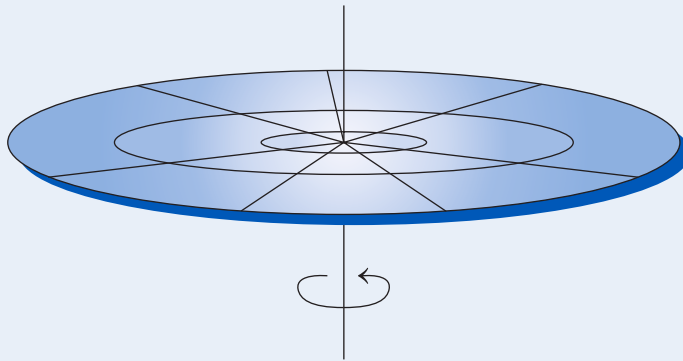


Figure 3.1

The time to read a particular block of data into memory has three components:

1. *Seek time*—the time to position the read head over the proper track. This time varies depending on the relative position of the read head and the proper track when the read request is generated. In the best case, the read head is already over the proper track and the seek time is 0. At the worst case, assuming there are n tracks, the read head might be over track 1 and have to move to track n , which would be $n - 1$ units, where a *unit* is the distance between adjacent tracks. We can assume that the seek time would be some multiple of the number of units.
2. *Latency time*—the time for the proper sector to rotate underneath the read head. This time also varies, depending on whether the correct sector is just coming under the read head (minimum latency time) or whether it has just gone by and a full rotation must occur (maximum latency time).
3. *Transfer time*—the time to read a block once it is positioned under the read head, usually a constant amount of time for all blocks.

The problem is to find the average seek time, actually the average number $A(n)$ of units. The assumptions are that there are n tracks, the read head is positioned over some track i , and the read head is equally likely to have to move to any track j .

⁴This example is based on work found in “Research Problem for Undergraduate Students which Spans Hardware Issues, Mathematical Methods and Programming: Average Seek Time of a Computer Disk,” by Jan Plaza, <http://faculty.plattsburgh.edu/jan.plaza/teaching/papers/seektime.html>

Table 3.3 shows the number of units in going from one track to another, where a row is the source track and a column is the destination track. For example, if the read head is currently over track 3 and must end up over track n , then $n - 3$ units are required, as shown by the entry in row 3, column n . The entry in row n , column 3 is the same because it takes the same number of units to move in the opposite direction.

TABLE 3.3

Source track \ Destination track	1	2	3	...	$n - 1$	n
1	0	1	2	...	$n - 2$	$n - 1$
2	1	0	1	...	$n - 3$	$n - 2$
3	2	1	0	...	$n - 4$	$n - 3$
...			
$n - 1$	$n - 2$	$n - 3$	$n - 4$...	0	1
n	$n - 1$	$n - 2$	$n - 3$...	1	0

Table 3.3 illustrates the n^2 different possible track moves. We find the average number $A(n)$ of units for these n^2 cases by computing the total number of units shown in the table, $T(n)$, and then dividing by n^2 . To compute $T(n)$, note that $T(n) = T(n - 1) +$ (the total of the last row plus the last column) and that the last row plus the last column contribute

$$2[1 + 2 + 3 + \cdots + (n - 1)] = 2 \left[\frac{(n - 1)n}{2} \right] \quad (\text{using Practice 7, Section 2.2})$$

$$= (n - 1)n$$

so that

$$T(n) = T(n - 1) + (n - 1)n$$

The base case is $T(1) = 0$ (no seek time for a 1-track disk). This is a linear, first-order recurrence relation with constant coefficients. We can solve it using Equation (8), where $c = 1$ and $g(n) = (n - 1)n$. The solution is

$$T(n) = 0 + \sum_{i=2}^n (i - 1)i$$

$$= 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \cdots + (n - 1)n$$

$$= \frac{(n - 1)n(n + 1)}{3} \quad (\text{from Exercise 19, Section 2.2})$$

Therefore the average number of units is

$$A(n) = \frac{(n - 1)n(n + 1)}{3} / n^2 = \frac{n^3 - n^2 + n^2 - n}{3n^2} = \frac{n^3 - n}{3n^2} = \frac{n^2 - 1}{3n} = \frac{n}{3} - \frac{1}{3n}$$

Because the best case is 0 and the worst case is $n - 1$, we might have expected the average case to be close to $n/2$, but in fact it is slightly less than $n/3$. ●

EXAMPLE 20

Not every recurrence relation fits the pattern of Equation (6). Consider the recurrence relation

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2nT(n-1) \text{ for } n \geq 2 \end{aligned}$$

Although this is a linear, first-order recurrence relation, it does not have constant coefficients. Equation (8) does not apply. To find a closed-form solution, we have to go back to the expand, guess, and verify technique.

$$\begin{aligned} T(n) &= 2nT(n-1) \\ &= 2n[2(n-1)T(n-2)] = 2^2n(n-1)T(n-2) \\ &= 2^2n(n-1)[2(n-2)T(n-3)] = 2^3n(n-1)(n-2)T(n-3) \end{aligned}$$

In general, it seems that

$$T(n) = 2^k n(n-1)(n-2) \dots (n-(k-1))T(n-k)$$

When $n-k=1$, then $k=n-1$ and

$$T(n) = 2^{n-1}n(n-1)(n-2) \dots (2)T(1) = 2^{n-1}n(n-1)(n-2) \dots (2)(1) = 2^{n-1}n!$$

This is our guess at a closed-form solution, which we verify by induction on n .

$$\text{Base case, } T(1): \quad T(1) = 2^{1-1}1! = 2^0(1) = 1, \text{ true}$$

$$\text{Assume } T(k): \quad T(k) = 2^{k-1}k!$$

$$\text{Show } T(k+1): \quad T(k+1) = 2^k(k+1)!$$

$$\begin{aligned} T(k+1) &= 2(k+1)T(k) && \text{(by the recurrence relation)} \\ &= 2(k+1)2^{k-1}k! && \text{(by the inductive hypothesis)} \\ &= 2^k(k+1)! \end{aligned}$$

Therefore our closed-form solution guess was correct. ●

Linear Second-Order Recurrence Relations

In a first-order recurrence relation, the n th term depends only on the previous term. In a **second-order recurrence relation**, the n th term depends on the two previous terms. Linear second-order homogeneous recurrence relations with constant coefficients therefore have the form

$$S(n) = c_1S(n-1) + c_2S(n-2) \tag{9}$$

The Fibonacci sequence is an example (Exercise 37 asks for a solution):

$$\begin{aligned} F(1) &= 1 \\ F(2) &= 1 \\ F(n) &= F(n-1) + F(n-2) \text{ for } n > 2 \end{aligned}$$

In such a sequence, we must have two “base case” values, that is, two known values of the sequence in order to generate subsequent values.

We’d like to find a general solution formula for recurrence relations like (9). If we drop the second term, we of course have a linear first-order homogeneous recurrence relation with constant coefficients:

$$S(n) = c_1 S(n - 1)$$

From Equation (8), we know that the solution to this recurrence relation has the form

$$S(n) = c_1^{n-1} S(1)$$

Let’s express this solution as

$$S(n) = pr^{n-1} \tag{10}$$

where r (that is, c_1) is the solution (root) of the linear equation

$$t - c_1 = 0 \tag{11}$$

and p (that is, $S(1)$) satisfies equation (10) for the initial condition $n = 1$:

$$S(1) = pr^{1-1} = pr^0 = p$$

This viewpoint suggests a way in which we might conjecture a solution to equation (9). Since we now have two terms in the equation itself, let’s add a second term to (10) and represent a potential solution as

$$S(n) = pr_1^{n-1} + qr_2^{n-1} \tag{12}$$

where r_1 and r_2 are two distinct roots of (extending (11) to a quadratic equation)

$$t^2 - c_1 t - c_2 = 0 \tag{13}$$

The p and q will have to be chosen to satisfy the two initial conditions:

$$\begin{aligned} S(1) &= pr_1^{1-1} + qr_2^{1-1} = p + q \\ S(2) &= pr_1^{2-1} + qr_2^{2-1} = pr_1 + qr_2 \end{aligned}$$

or, simplifying,

$$\begin{aligned} p + q &= S(1) \\ pr_1 + qr_2 &= S(2) \end{aligned} \tag{14}$$

Of course, this is just a wild leap of speculation on our part, so we must now verify that Equation (12) is a closed-form solution to recurrence relation (9).

We are trying to prove that

$$S(n) = pr_1^{n-1} + qr_2^{n-1}$$

(where r_1 , r_2 , p , and q are as described) is a solution to

$$S(n) = c_1 S(n - 1) + c_2 S(n - 2)$$

for all $n \geq 1$. The “for all $n \geq 1$ ” phrase suggests a proof by mathematical induction. Because $S(n)$ has to “reach back” two values to compute the current value, we should use the second principle of induction.

Base Cases:

When $n = 1$, the proposed solution gives

$$S(1) = pr_1^{1-1} + qr_2^{1-1} = p + q$$

When $n = 2$, the proposed solution gives

$$S(2) = pr_1^{2-1} + qr_2^{2-1} = pr_1 + qr_2$$

Both are trivially true because we chose p and q to meet these very conditions.

Assume that for all r , $1 \leq r \leq k$, $S(r) = pr_1^{r-1} + qr_2^{r-1}$. Show that $S(k+1) = pr_1^k + qr_2^k$. Before we proceed, note that because r_1 and r_2 are solutions of the equation $t^2 - c_1t - c_2 = 0$, it is true that

$$\begin{aligned} r_1^2 - c_1r_1 - c_2 &= 0 & \text{or} & & r_1^2 &= c_1r_1 + c_2 \\ r_2^2 - c_1r_2 - c_2 &= 0 & \text{or} & & r_2^2 &= c_1r_2 + c_2 \end{aligned} \quad (15)$$

Now

$$\begin{aligned} S(k+1) &= c_1S(k) + c_2S(k-1) && \text{(by the recurrence relation)} \\ &= c_1(pr_1^{k-1} + qr_2^{k-1}) + c_2(pr_1^{k-2} + qr_2^{k-2}) && \text{(by the inductive} \\ &= pr_1^{k-2}(c_2 + c_1r_1) + qr_2^{k-2}(c_2 + c_1r_2) && \text{hypothesis, applied twice)} \\ &= pr_1^{k-2}r_1^2 + qr_2^{k-2}r_2^2 && \text{(by Equation (15))} \\ &= pr_1^k + qr_2^k \end{aligned}$$

which is the desired result. This confirms that Equation (12) is a solution to Equation (8).

The key to the solution is the quadratic equation

$$t^2 - c_1t - c_2 = 0$$

which is called the **characteristic equation** of the recurrence relation

$$S(n) = c_1S(n-1) + c_2S(n-2)$$

EXAMPLE 21

Solve the recurrence relation

$$S(n) = 2S(n-1) + 3S(n-2) \text{ for } n \geq 3$$

subject to the initial conditions

$$S(1) = 3$$

$$S(2) = 1$$

In this recurrence relation, $c_1 = 2$ and $c_2 = 3$. To find the closed-form solution, we form the characteristic equation

$$t^2 - 2t - 3 = 0$$

which has roots $r_1 = 3$, $r_2 = -1$. Equation (12) gives the solution form:

$$S(n) = p3^{n-1} + q(-1)^{n-1}$$

where from Equation (14) p and q satisfy

$$\begin{aligned} p + q &= 3 \\ p(3) + q(-1) &= 1 \end{aligned}$$

Solving this system of equations results in $p = 1$, $q = 2$. Therefore the closed-form solution is

$$S(n) = 3^{n-1} + 2(-1)^{n-1}$$

Practice 13

- Using the base cases and the recurrence relation, compute the first five terms of the sequence $S(n)$ of Example 21.
- Check that the closed-form solution formula in Example 21 produces the correct first five terms. ■

Practice 14

Solve the recurrence relation

$$T(n) = 6T(n-1) - 5T(n-2) \text{ for } n \geq 3$$

subject to the initial conditions

$$\begin{aligned} T(1) &= 5 \\ T(2) &= 13 \end{aligned}$$

Although it would seem at this point that we have the solution method in hand for any linear second-order homogeneous recurrence relations with constant coefficients, such is not the case. Consider the system of Equation (14):

$$\begin{aligned} p + q &= S(1) \\ pr_1 + qr_2 &= S(2) \end{aligned}$$

We can solve the first equation for p —

$$p = S(1) - q$$

—and then substitute in the second equation to solve for q :

$$\begin{aligned} [S(1) - q]r_1 + qr_2 &= S(2) \\ q(r_2 - r_1) &= S(2) - S(1)r_1 \\ q &= \frac{S(2) - S(1)r_1}{r_2 - r_1} \end{aligned}$$

Now what happens if the characteristic equation $t^2 - c_1t - c_2 = 0$ happens to have one repeated root, that is, $r_1 = r_2$? Oops—we can't solve this system of equations. The solution form when the characteristic equation has a repeated root r looks like

$$S(n) = pr^{n-1} + q(n-1)r^{n-1} \text{ for all } n \geq 1$$

where p and q satisfy the equations

$$\begin{aligned} p &= S(1) \\ pr + qr &= S(2) \end{aligned}$$

This can be proved by induction in a manner similar to the distinct roots case (see Exercise 44).

EXAMPLE 22

Solve the recurrence relation

$$S(n) = 8S(n-1) - 16S(n-2) \text{ for } n \geq 3$$

subject to the initial conditions

$$\begin{aligned} S(1) &= 1 \\ S(2) &= 12 \end{aligned}$$

In this recurrence relation, $c_1 = 8$ and $c_2 = -16$. To find the closed-form solution, we form the characteristic equation

$$\begin{aligned} t^2 - 8t + 16 &= 0 \\ (t - 4)^2 &= 0 \end{aligned}$$

which has a repeated root $r = 4$. The solution is

$$S(n) = p4^{n-1} + q(n-1)4^{n-1}$$

where

$$\begin{aligned} p &= 1 \\ p(4) + q(4) &= 12 \end{aligned}$$

Solving this system of equations, $p = 1$ and $q = 2$, so the solution is

$$S(n) = 4^{n-1} + 2(n-1)4^{n-1} = (2n-1)4^{n-1}$$

Table 3.4 summarizes the solution steps for a linear second-order homogeneous recurrence relation with constant coefficients:

TABLE 3.4	
To Solve Recurrence Relations of the Form $S(n) = c_1S(n - 1) + c_2S(n - 2)$ Subject to Initial Conditions $S(1)$ and $S(2)$	
1. Solve the characteristic equation $t^2 - c_1t - c_2 = 0$	
2. If the characteristic equation has distinct roots r_1 and r_2 , the solution is	
	$S(n) = pr_1^{n-1} + qr_2^{n-1}$
where	
	$p + q = S(1)$ $pr_1 + qr_2 = S(2)$
3. If the characteristic equation has a repeated root r , the solution is	
	$S(n) = pr^{n-1} + q(n-1)r^{n-1}$
where	
	$p = S(1)$ $pr + qr = S(2)$

The proofs for case 2 and case 3 are unchanged if the roots of the characteristic equation turn out to be complex numbers. In other words, the solution formulas still work.

Divide-and-Conquer Recurrence Relations

Still another recurrence relation form occurs when the value of $S(n)$ depends not on the previous term or on the two previous terms, but on the value halfway back in the sequence, $S\left(\frac{n}{2}\right)$

EXAMPLE 23

Consider the sequence with the following values:

$$S(1) = 2, S(2) = 4, S(4) = 8, S(8) = 16, S(16) = 32, \dots$$

We are looking at only selected terms of the sequence, namely $S(n)$ where n is a power of 2. For these terms, we can see that $S(n) = 2S\left(\frac{n}{2}\right)$ •

Such recurrence relations will occur in the analysis of certain “divide-and-conquer” algorithms, algorithms that solve a problem by breaking it into smaller versions, each half the size of the original (see the next section). Hence such recurrence relations are called **divide-and-conquer recurrence relations**. The general form is

$$S(n) = cS\left(\frac{n}{2}\right) + g(n) \text{ for } n \geq 2, n = 2^m \quad (16)$$

where c is a constant and g can be an expression involving n . It's convenient to look only at values of n that are powers of 2 because then cutting n in half over and over always results in an integer. As we will see in the next section, this is not a significant restriction.

To solve a divide-and-conquer recurrence relation, we go back to the expand, guess, and verify approach. Also, the solution will involve the logarithm function; for a review of the logarithm function and its properties, see Appendix C.

EXAMPLE 24

Solve the recurrence relation

$$C(n) = 1 + C\left(\frac{n}{2}\right) \text{ for } n \geq 2, n = 2^m$$

subject to the basis step

$$C(1) = 1$$

Expanding, we get

$$\begin{aligned} C(n) &= 1 + C\left(\frac{n}{2}\right) \\ &= 1 + \left(1 + C\left(\frac{n}{4}\right)\right) \\ &= 1 + 1 + \left(1 + C\left(\frac{n}{8}\right)\right) \\ &\quad \vdots \end{aligned}$$

and the general term seems to be

$$C(n) = k + C\left(\frac{n}{2^k}\right)$$

The process stops when $n/2^k = 1$ or $2^k = n$, which means $k = \log_2 n$. (We'll omit the base 2 notation from now on— $\log n$ will mean $\log_2 n$. See Appendix C for a brief discussion of logarithms.) Then

$$C(n) = \log n + C(1) = 1 + \log n$$

Now we will use induction to verify that $C(n) = 1 + \log n$ for all $n \geq 1$, $n = 2^m$. This is a somewhat different form of induction, because the only values of interest are powers of 2. We still take 1 as the basis step for the induction, but then we prove that if our statement is true for a value k , it is true for $2k$. The statement will then be true for 1, 2, 4, 8, ..., that is, for all nonnegative integer powers of 2, which is just what we want.

For the base case,

$$C(1) = 1 + \log 1 = 1 + 0 = 1, \text{ true}$$

Assume that $C(k) = 1 + \log k$. Then

$$\begin{aligned}
 C(2k) &= 1 + C(k) && \text{(by the recurrence relation)} \\
 &= 1 + 1 + \log k && \text{(by the inductive hypothesis)} \\
 &= 1 + \log 2 + \log k && (\log 2 = 1) \\
 &= 1 + \log 2k && \text{(property of logarithms)}
 \end{aligned}$$

This calculation completes the inductive proof. ●

We'd like to find a closed-form solution for (16) subject to the basis that $S(1)$ is known. We could use the expand, guess, and verify approach to find the general solution, but instead we will do a transformation on (16) to convert it to a first-order recurrence relation with constant coefficients, use the solution formula we already have for such a recurrence relation, and then reverse the transformation. Figure 3.2 shows this round-about approach.

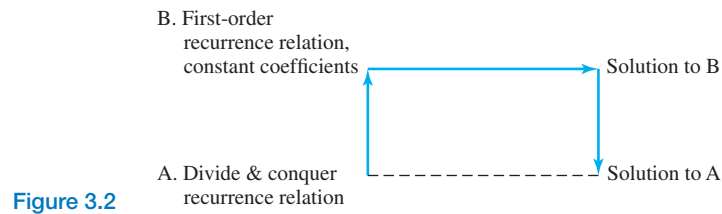


Figure 3.2

Equation (16) assumes that $n = 2^m$ with $n \geq 2$. From this it follows that $m = \log n$ and $m \geq 1$. Substituting 2^m for n in equation (16) results in

$$S(2^m) = cS(2^{m-1}) + g(2^m) \quad (17)$$

Now, letting $T(m)$ represent $S(2^m)$ in Equation (17), we get

$$T(m) = cT(m-1) + g(2^m) \text{ for } m \geq 1 \quad (18)$$

Equation (18) is a linear, first-order equation with constant coefficients; from Equation (8), we obtain the solution

$$T(m) = c^{m-1}T(1) + \sum_{i=2}^m c^{m-i}g(2^i) \quad (19)$$

subject to the basis condition that $T(1)$ is known. Because Equation (18) holds for $m = 1$, we know that

$$T(1) = cT(0) + g(2)$$

Making this substitution in (19) results in

$$T(m) = c^m T(0) + \sum_{i=1}^m c^{m-i} g(2^i) \quad (20)$$

Now reversing the substitution $T(m) = S(2^m)$, (20) becomes

$$S(2^m) = c^m S(2^0) + \sum_{i=1}^m c^{m-i} g(2^i)$$

Finally, letting $2^m = n$ or $m = \log n$, we get

$$S(n) = c^{\log n} S(1) + \sum_{i=1}^{\log n} c^{(\log n)-i} g(2^i) \quad (21)$$

REMINDER

In the summation part of the general solution formula, c is raised to the $(\log n) - i$ power, not $(\log n) - 1$

Equation (21) thus represents the solution for the recurrence relation (16). As before, to use this general solution you need only match your recurrence relation to (16) to determine c and $g(n)$, then substitute into Equation (21). Again as before, $g(n)$, gives a recipe for what to do with an argument n ; in Equation (21), the argument is 2^i . If you can evaluate the resulting summation, you will then have a closed-form solution. Table 3.5 outlines the solution steps.

TABLE 3.5

To Solve Recurrence Relations of the Form $S(n) = cS\left(\frac{n}{2}\right) + g(n)$ for $n \geq 2$, $n = 2^m$ Subject to Initial Condition $S(1)$

1. Match your recurrence relation to the form

$$S(n) = cS\left(\frac{n}{2}\right) + g(n)$$

to find c and $g(n)$.

2. Use c , $g(n)$ and $S(1)$ in the formula

$$S(n) = c^{\log n} S(1) + \sum_{i=1}^{\log n} c^{(\log n)-i} g(2^i)$$

3. Evaluate the resulting summation to get the final expression.

EXAMPLE 25

The recurrence relation

$$C(1) = 1$$

$$C(n) = 1 + C\left(\frac{n}{2}\right) \text{ for } n \geq 2, n = 2^m$$

matches Equation (16), with $c = 1$ and $g(n) = 1$. Because $g(n) = 1$, the g function evaluates to 1 no matter what the argument is. The solution, according to formula (21), is

$$\begin{aligned} C(n) &= 1^{\log n} C(1) + \sum_{i=1}^{\log n} 1^{(\log n)-i} (1) \\ &= 1 + (\log n)(1) = 1 + \log n \end{aligned}$$

which agrees with our previous result from Example 24. ●

EXAMPLE 26 Solve the recurrence relation

$$T(1) = 3$$

$$T(n) = 2T\left(\frac{n}{2}\right) + 2n$$

This is a match for Equation (16), where $c = 2$ and $g(n) = 2n$. Therefore $g(2^i) = 2(2^i)$. Substituting into Equation (21)—the solution of Equation (16)—gives the following result, where we use the fact that $2^{\log n} = n$.

$$\begin{aligned} T(n) &= 2^{\log n}T(1) + \sum_{i=1}^{\log n} 2^{\log n - i}2(2^i) \\ &= 2^{\log n}(3) + \sum_{i=1}^{\log n} 2^{\log n + 1} \\ &= n(3) + (2^{\log n + 1})\log n \\ &= 3n + (2^{\log n} \cdot 2)\log n \\ &= 3n + 2n \log n \end{aligned}$$

Practice 15 Show that the solution to the recurrence relation

$$S(1) = 1$$

$$S(n) = 2S\left(\frac{n}{2}\right) + 1 \text{ for } n \geq 2, n = 2^m$$

is $2n - 1$. (*Hint:* See Example 15 in Section 2.2 and note that $2^{\log n} = n$.)

SECTION 3.2 REVIEW

TECHNIQUES

- Solve recurrence relations by the expand, guess, and verify technique.
- Solve linear, first-order recurrence relations with constant coefficients by using a solution formula.
- Solve linear, second-order homogeneous recurrence relations with constant coefficients by using the characteristic equation.

- Solve divide-and-conquer recurrence relations by using a solution formula.

MAIN IDEA

- Certain recurrence relations have closed-form solutions.

EXERCISES 3.2

In Exercises 1–12, solve the recurrence relation subject to the basis step.

1. $S(1) = 5$
 $S(n) = S(n - 1) + 5$ for $n \geq 2$
2. $B(1) = 5$
 $B(n) = 3B(n - 1)$ for $n \geq 2$

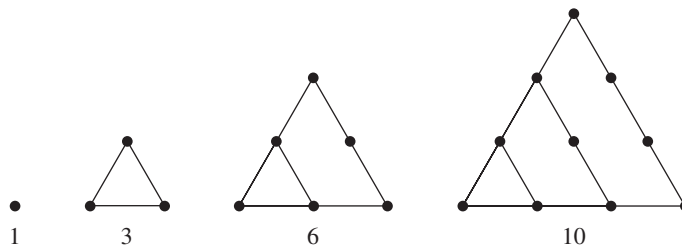
3. $F(1) = 2$
 $F(n) = 2F(n - 1) + 2^n$ for $n \geq 2$
4. $T(1) = 1$
 $T(n) = 2T(n - 1) + 1$ for $n \geq 2$
(Hint: See Example 15 in Section 2.2.)
5. $A(1) = 1$
 $A(n) = A(n - 1) + n$ for $n \geq 2$
(Hint: See Practice 7 in Section 2.2.)
6. $S(1) = 1$
 $S(n) = S(n - 1) + (2n - 1)$ for $n \geq 2$
(Hint: See Example 14 in Section 2.2.)
7. $T(1) = 1$
 $T(n) = T(n - 1) + n^2$ for $n \geq 2$
(Hint: See Exercise 7 in Section 2.2.)
8. $P(1) = 2$
 $P(n) = 2P(n - 1) + n2^n$ for $n \geq 2$
(Hint: See Practice 7 in Section 2.2.)
9. $F(1) = 1$
 $F(n) = nF(n - 1)$ for $n \geq 2$
10. $S(1) = 1$
 $S(n) = nS(n - 1) + n!$ for $n \geq 2$
11. $A(1) = 1$
 $A(n) = 2(n - 1)A(n - 1)$ for $n \geq 2$
(Hint: $0!$ is defined to equal 1.)
12. $P(1) = 2$
 $P(n) = 3(n + 1)P(n - 1)$ for $n \geq 2$
13. At the beginning of this chapter the contractor claimed:

The material to be stored at the chemical disposal site degrades to inert matter at the rate of 5% per year. Therefore only about one-third of the original active material will remain at the end of 20 years.

- a. Write a recurrence relation $T(n)$ for the amount of active material at the beginning of year n . Assume that $T(1) = X$, a specific but unknown amount.
 - b. Solve the recurrence relation.
 - c. Compute $T(21)$ to check the contractor's claim; note that the end of 20 years is the beginning of the 21st year.
14. A colony of bats is counted every 2 months. The first four counts are 1200, 1800, 2700, and 4050.
 - a. Assuming that this growth rate continues, write a recurrence relation for the number of bats at count n .
 - b. Solve the recurrence relation.
 - c. What will the 12th count be?
 15. Spam e-mail containing a virus is sent to 1,000 e-mail addresses. After 1 second, a recipient machine broadcasts 10 new spam e-mails containing the virus, after which the virus disables itself on that machine.

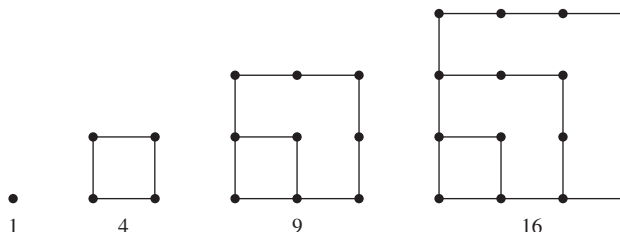
- a. Write a recurrence relation for the number of e-mails sent at the start of the n th second.
 - b. Solve the recurrence relation.
 - c. How many e-mails are sent at the end of 20 seconds (that is, at the beginning of the 21st second)?
16. Total natural gas consumption in the state of New Jersey was 614,908 million cubic feet in 2008 and 653,459 million cubic feet in 2010.
- a. Assuming a constant annual percentage growth rate r , write a recurrence relation (in terms of r) for the total natural gas consumption in New Jersey in year n .
 - b. Solve the recurrence relation (in terms of r).
 - c. Using the given data, compute the value of r .
 - d. What will be the total natural gas consumption in New Jersey in the year 2020?
17. A loan of \$5,000 is charged a 12% annual interest rate. An \$80 payment is made each month.
- a. Write a recurrence relation for the loan balance remaining at the beginning of month n .
 - b. Solve the recurrence relation. (See Exercise 27 of Section 2.2 for the formula for the sum of a geometric sequence.)
 - c. How much is left of the loan balance at the beginning of the 19th month?
18. In an account that pays 3% annually, \$1,000 is deposited. At the end of each year, an additional \$100 is deposited into the account.
- a. Write a recurrence relation for the amount in the account at the beginning of year n .
 - b. Solve the recurrence relation. (See Exercise 27 of Section 2.2 for the formula for the sum of a geometric sequence.)
 - c. What is the account worth at the beginning of the 8th year?
19. The shellfish population in a bay is estimated to have a count of about 1,000,000. Studies show that pollution reduces this population by about 2% per year, while other hazards are judged to reduce the population by about 10,000 per year.
- a. Write a recurrence relation for the shellfish population at the beginning of year n .
 - b. Solve the recurrence relation. (See Exercise 27 of Section 2.2 for the formula for the sum of a geometric sequence.)
 - c. What is the approximate shellfish population at the beginning of year 10?
20. A certain protected species normally doubles its population each month. The initial population is 20, but by the beginning of the next month, 1 specimen has died of an infection. In successive months, the infection kills 2, then 4, then 8, and so forth.
- a. Write a recurrence relation for the size of the population at the beginning of month n .
 - b. Solve this recurrence relation.
 - c. What is the size of the population at the beginning of month 7?
21. A computer virus that spreads by way of e-mail messages is planted in 3 machines the first day. Each day, each infected machine from the day before infects 5 new machines. By the end of the second day, a software solution has been found to counteract the virus, and 1 machine is clean at that point. Each day thereafter, 6 times as many machines are clean as were clean the day before.
- a. Write a recurrence relation for the total number of infected machines on day n .
 - b. Solve this recurrence relation.
 - c. How many days will it be before the effects of the virus are completely gone?
22. This problem concerns the Towers of Hanoi puzzle (see Exercise 82 in Section 3.1).
- a. Based on the recursive algorithm of Exercise 82 in Section 3.1, find a recurrence relation $M(n)$ for the number of disk moves required to solve the Towers of Hanoi puzzle for n disks.

- b. Solve this recurrence relation. (*Hint*: See Exercise 15 in Section 2.2.)
- c. Go through the steps of the solution algorithm for $n = 3$ and record the number of disk moves required. Compare this number with the result from part (b) with $n = 3$.
- d. The mythical origin of the Towers of Hanoi puzzle concerns 64 golden disks that a group of monks are moving from one tower to another. When their task is complete, the world will end. Assuming that the monks can move 1 disk per second, calculate the number of years to complete the task.
23. Early members of the Pythagorean Society defined *figurate numbers* to be the number of dots in certain geometrical configurations. The first few *triangular numbers* are 1, 3, 6, and 10:



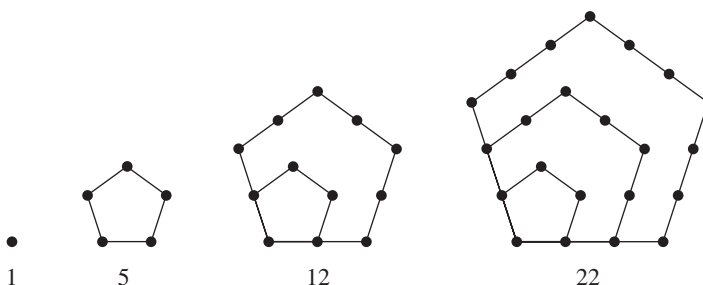
Find and solve a recurrence relation for the n th triangular number. (*Hint*: See Practice 7 in Section 2.2.)

24. The first few *square numbers* (see the previous Exercise) are 1, 4, 9, and 16:



Find and solve a recurrence relation for the n th square number. (*Hint*: See Example 14 in Section 2.2.)

25. The first few *pentagonal numbers* (see Exercise 23) are 1, 5, 12, and 22:



Find and solve a recurrence relation for the n th pentagonal number. (*Hint*: See Exercise 28 of Section 2.2 for the formula for the sum of an arithmetic sequence.)

26. Use induction to verify that equation (8) of this section is the solution to the recurrence relation (6) subject to the basis condition that $S(1)$ is known.

In Exercises 27–34, solve the recurrence relation subject to the initial conditions.

27. $T(1) = 5$
 $T(2) = 11$
 $T(n) = 5T(n - 1) - 6T(n - 2)$ for $n \geq 3$

$$28. \begin{aligned} A(1) &= 7 \\ A(2) &= 18 \\ A(n) &= 6A(n-1) - 8A(n-2) \text{ for } n \geq 3 \end{aligned}$$

$$29. \begin{aligned} S(1) &= 4 \\ S(2) &= -2 \\ S(n) &= -S(n-1) + 2S(n-2) \text{ for } n \geq 3 \end{aligned}$$

$$30. \begin{aligned} P(1) &= 5 \\ P(2) &= 17 \\ P(n) &= 7P(n-1) - 12P(n-2) \text{ for } n \geq 3 \end{aligned}$$

$$31. \begin{aligned} F(1) &= 8 \\ F(2) &= 16 \\ F(n) &= 6F(n-1) - 5F(n-2) \text{ for } n \geq 3 \end{aligned}$$

$$32. \begin{aligned} T(1) &= -1 \\ T(2) &= 7 \\ T(n) &= -4T(n-1) - 3T(n-2) \text{ for } n \geq 3 \end{aligned}$$

$$33. \begin{aligned} B(1) &= 3 \\ B(2) &= 14 \\ B(n) &= 4B(n-1) - 4B(n-2) \text{ for } n \geq 3 \end{aligned}$$

$$34. \begin{aligned} F(1) &= -10 \\ F(2) &= 40 \\ F(n) &= -10F(n-1) - 25F(n-2) \text{ for } n \geq 3 \end{aligned}$$

In Exercises 35 and 36, solve the recurrence relation subject to the initial conditions; the solutions involve complex numbers.

$$35. \begin{aligned} A(1) &= 8 \\ A(2) &= 8 \\ A(n) &= 2A(n-1) - 2A(n-2) \text{ for } n \geq 3 \end{aligned}$$

$$36. \begin{aligned} S(1) &= 4 \\ S(2) &= -8 \\ S(n) &= -4S(n-1) - 5S(n-2) \text{ for } n \geq 3 \end{aligned}$$

37. Solve the Fibonacci recurrence relation

$$\begin{aligned} F(1) &= 1 \\ F(2) &= 1 \\ F(n) &= F(n-1) + F(n-2) \text{ for } n > 2 \end{aligned}$$

Compare your answer with Exercise 31 of Section 3.1.

38. Find a closed-form solution for the Lucas sequence

$$\begin{aligned} L(1) &= 1 \\ L(2) &= 3 \\ L(n) &= L(n-1) + L(n-2) \text{ for } n \geq 3 \end{aligned}$$

39. Houses in a new development go on sale initially for an average price of \$200,000. At the beginning of month 2, the average sale price has risen to \$250,000. At the beginning of each succeeding month, the average price increase is half what it was the previous month.
- Write and solve a recurrence relation for $M(n)$, the average sale price at the beginning of month n .
 - At the beginning of which month is the average price within \$2,000 of \$300,000?
40. A contaminated soil site is tested monthly for the presence of a particular microorganism. Initially, 950 microorganisms per cubic foot of soil are found; at the beginning of month 2, there are 1,000 organisms per cubic foot. Left untreated, the growth rate of this microorganism increases by 25% per month.
- Write and solve a recurrence relation for $O(n)$, the number of organisms present per cubic foot at the beginning of month n .
 - At the end of what month does the number of organisms first exceed 5,000 per cubic foot?
41. Prove that the number of binary strings of length n with no two consecutive 0s is given by the Fibonacci sequence term $F(n + 2)$. (*Hint*: Write a recurrence relation; consider strings of length n that end in 1 and those that end in 0.)
- Find a recurrence relation for the number of binary strings of length n that have two consecutive 1s.
 - How many binary strings of length 4 have two consecutive 1s? What are these strings?
43. Consider the recurrence relation $S(n) = c_1S(n - 1)$ as a linear second-order homogeneous recurrence relation with constant coefficients where $c_2 = 0$. Solve this recurrence relation using its characteristic equation, and prove that the solution is the same as that of Equation (8).
44. Prove that

$$S(n) = pr^{n-1} + q(n - 1)r^{n-1}$$

where

$$\begin{aligned} p &= S(1) \\ pr + qr &= S(2) \end{aligned}$$

is a solution to the recurrence relation $S(n) = c_1S(n - 1) + c_2S(n - 2)$ for all $n \geq 1$ if r is a repeated root of the characteristic equation.

In Exercises 45–48, solve the recurrence relation subject to the basis step. (*Hint*: See Example 15 in Section 2.2, and note that $2^{\log n} = n$.)

45. $P(1) = 1$

$$P(n) = 2P\left(\frac{n}{2}\right) + 3 \text{ for } n \geq 2, n = 2^m$$

46. $T(1) = 3$

$$T(n) = T\left(\frac{n}{2}\right) + n \text{ for } n \geq 2, n = 2^m$$

47. $S(1) = 1$

$$S(n) = 2S\left(\frac{n}{2}\right) + n \text{ for } n \geq 2, n = 2^m$$

48. $P(1) = 1$

$$P(n) = 2P\left(\frac{n}{2}\right) + n^2 \text{ for } n \geq 2, n = 2^m$$

SECTION 3.3 ANALYSIS OF ALGORITHMS

The General Idea

Often more than one algorithm can perform the same task. Because we assume that all these algorithms perform correctly, we need some other basis of comparison to decide which algorithm to use in a given situation. Several criteria could be used to judge which is the “best” algorithm. We might ask, for example, which is easiest to understand, which makes the most efficient use of machine memory when implemented as computer code, or which runs most efficiently. One might expect that to judge whether an algorithm is “running efficiently” means standing by with a stopwatch while the computer code executes. But the stopwatch approach may tell us more about the speed of the processor than it does the inherent efficiency of the algorithm. Even timing the code for competing algorithms on the same processor and using the same input data can give a misleading picture of what might happen when the input data set is larger.

Instead, we evaluate the time efficiency of an algorithm by estimating the number of operations it must perform. We count only the operations that are basic to the task at hand, not “housekeeping” operations that make just a small contribution to the total work required.

The study of the efficiency of algorithms, that is, the number of operations they perform, is called **analysis of algorithms**. Various techniques for algorithm analysis have been developed. Sometimes a rather straightforward analysis can be done simply by inspecting the algorithm.

EXAMPLE 27

Following is an algorithm to write out, for each student on a grade roster, the sum of m quiz grades minus the lowest grade. The outer loop goes through each of the n students; the inner loop goes through the quizzes for the current student. For each student, the successive quiz grades are added and the lowest grade is ultimately subtracted from the sum of all the grades. These additions and subtractions seem fundamental to how the algorithm works, so we will count the work contributed by these arithmetic operations.

```

for  $i = 1$  to  $n$  do
     $low = roster[i].quiz[1]$ 
     $sum = roster[i].quiz[1]$ 

    for  $j = 2$  to  $m$  do
         $sum = sum + roster[i].quiz[j]$            //A
        if  $roster[i].quiz[j] < low$  then
             $low = roster[i].quiz[j]$ 
        end if
    end for
     $sum = sum - low$                                //S
    write(“Total for student”,  $i$ , “is”,  $sum$ )
end for

```

Subtraction occurs at the line marked //S, which is executed once for each pass through the outer loop (once for each student), a total of n times. Addition,

however, occurs at the line marked *//A*. This is done within the inner loop, which executes $m - 1$ times for each student, that is, for each of the n passes of the outer loop. The total number of additions is therefore $n(m - 1)$. The total number of arithmetic operations is $n + n(m - 1) = nm$. Of course the value of this expression depends on n (the number of students) and m (the number of quizzes). The quantities n and m measure the amount of input data; virtually any algorithm's work will change with the input size.

The algorithm also does some "housekeeping" work. Values are assigned to variables, comparisons are done (to find the lowest quiz grade for each student), and for the loop indices i and j have to be incremented. But the number of times these operations are done also depends on the number of times through the loops, so their effect might be to multiply the nm result by some constant factor. In comparing algorithms A and B , we're usually looking for bigger differences than just a constant multiple, which is why we ignore the housekeeping details. ●

Now suppose the task is to search a sorted list of n words or numbers for a particular target value x . We already have one algorithm to perform this task, the binary search algorithm from Section 3.1. Another algorithm for the same task, the **sequential search algorithm**, simply compares x with each entry in the list in turn until either x is found or the list is exhausted. (This algorithm actually works on any list, sorted or not.) A pseudocode description of the sequential search algorithm is given in the following box.

ALGORITHM *SEQUENTIALSEARCH*

```

SequentialSearch(list  $L$ ; integer  $n$ ; itemtype  $x$ )
//searches a list  $L$  of  $n$  items for item  $x$ 
Local variable:
integer  $i$  //marks position in the list
     $i = 1$ 
    while  $L[i] \neq x$  and  $i < n$  do
         $i = i + 1$ 
    end while
    if  $L[i] = x$  then
        write("Found")
    else
        write("Not found")
    end if
end function SequentialSearch

```

Both binary search and sequential search work by comparing elements from the list with the target value x until a match is found. In fact it is difficult to imagine how any possible search algorithm could avoid such comparisons, so these comparisons will be the basic operation to count in analyzing these two algorithms.

The sequential search algorithm does the maximum amount of work (number of comparisons) when x is the last item in the list or when x does not appear in the list at all. In either case, all elements are compared to x , so n comparisons are done. This is the “worst case” for this algorithm, and the work depends on the size n of the input (the length of the list). The minimum amount of work is done when x is the very first item in the list; only one comparison is made. This is the “best case” for this algorithm. (In the algorithm of Example 27, the same number of arithmetic operations is always done; there is no best or worst case.)

There are many possibilities between the best case and the worst case. If x falls in the exact middle of the list, the search would require roughly $n/2$ comparisons. It would be helpful to obtain some measure of the “average” amount of work done. This measure would require some way to describe the average list being searched and the average relationship of a target item x to that list. Exercises 35 and 36 in this section explore some aspects of average case analysis for the sequential search algorithm. For most algorithms, however, average behavior is very difficult to determine. To compare the efficiency of algorithms, therefore, we often content ourselves with the worst-case count of the number of operations required.

EXAMPLE 28

Given a long string of text characters, can we find the first instance of a particular substring or “pattern” within the text? This problem has a number of important applications, such as

- looking for specific strings in an HTML document that might be governed by a style rule of a Cascading Style Sheet.
- using the UNIX `grep` command to search a file for a specified string, or the “Find” command in any text editor or word processor.
- looking for specific gene sequences within a strand of DNA.

DNA is a long molecule that is basically a chemically bonded chain of smaller molecules called nucleotides. There are four nucleotides, abbreviated as A , C , G , and T . Thus a section of DNA might be represented as the sequence

...TAATCATGGTCATAGCTGTTTCCTGTGTGAAATTG...

DNA is stored within the cells of living organisms on chromosomes; various sections of these chromosomes are identified as genes. Genes, through their DNA “instructions,” create proteins that control specific functions or traits within the organism (hair color, blood type, and so on). Thus our entire genetic code requires only four symbols! “Mapping the human genome”, that is, determining the entire DNA sequence of humans, was a huge scientific undertaking, essentially completed in 2003, although the work of identifying specific genes and their particular function is ongoing. It is known, for example, that the disease cystic fibrosis is caused by a mutation in a particular gene (DNA sequence) that is composed of about 230,000 nucleotides.

The most intuitive (although not the most efficient) string pattern-matching algorithm compares the pattern (a string of length m) against the text (a string of length n with $n \geq m$), starting with the first character of the text and marching through the pattern.

$$\begin{array}{cccccccc}
 T_1 & T_2 & T_3 & \dots & T_m & T_{m+1} & \dots & T_n \\
 | & | & | & & | & & & \\
 P_1 & P_2 & P_3 & \dots & P_m & & &
 \end{array}$$

If all m characters match, the pattern has been found. If there is a mismatch at some point between the text and the pattern, the pattern slides over one character in the text, and the matching process begins again.

$$\begin{array}{cccccccc}
 T_1 & T_2 & T_3 & \dots & T_m & T_{m+1} & \dots & T_n \\
 & | & | & & & | & & \\
 & P_1 & P_2 & P_3 & \dots & P_m & &
 \end{array}$$

The last segment of text where the pattern could possibly occur is the last m elements of the text. This segment begins at T_{n-m+1} , as shown below.

$$\begin{array}{cccccccc}
 T_1 & T_2 & T_3 & \dots & T_{n-m+1} & T_{n-m+2} & \dots & T_{n-m+m} = T_n \\
 & & & & | & | & & | \\
 & & & & P_1 & P_2 & \dots & P_m
 \end{array}$$

For example, if the text is 23 characters long and the pattern is 5 characters long, then the last piece of text that can possibly hold the pattern is $T_{19}T_{20}T_{21}T_{22}T_{23}$.

The unit of work in this algorithm is comparisons between a text character and a pattern character. The best case occurs when the pattern is found as the first m characters of the text, which requires m comparisons. The worst case is when the pattern does not occur in the text at all, and the pattern “window” slides all the way over to T_{n-m+1} . From each of those $n - m + 1$ starting points, the pattern fails to be found, but the worst case is if the pattern almost matches and only the last character fails. For example, consider the text and pattern shown:

Text: *TTTTTTTTTTTTTT*

Pattern: *TTTTTS*

This example requires m comparisons (the first $m - 1$ are matches and only the m th comparison fails) at each of the $n - m + 1$ starting positions, making the total number of comparisons $m(n - m + 1)$. ●

Analysis Using Recurrence Relations

In this section we will analyze algorithms that are defined recursively. Because much of the activity of a recursive algorithm takes place “out of sight” in the many invocations that can occur, an analysis using the direct counting technique of Example 27 won’t work. Analysis of recursive algorithms usually involves solving a recurrence relation.

EXAMPLE 29

We can recast the sequential search algorithm from an iterative version (repeating some action over and over in a loop) to a recursive version. The base case says that you check whether you have run off the end of the list and if not, you check for the target x at the first position in the list. If you find it, fine, if not, then you invoke the algorithm again on the rest of the list. Here is pseudocode for the recursive function to search the list from $L[i]$ to $L[n]$; the function is invoked initially with $i = 1$.

ALGORITHM *SEQUENTIALSEARCHRECURSIVE*

```

SequentialSearchRecursive(list L; integer i, n; itemtype x)
//searches list L from L[i] to L[n] for item x
if  $i > n$  then
    write("not found")
else
    if  $L[i] = x$  then
        write("found")
    else
        SequentialSearchRecursive(L, i + 1, n, x)
    end if
end if
end function SequentialSearchRecursive

```

Figure 3.3 gives a visual representation of the recursive sequential search algorithm. Each time the algorithm is invoked, the new list to be searched is only 1 element shorter than the previous list, so in the worst case the algorithm has to work quite hard.

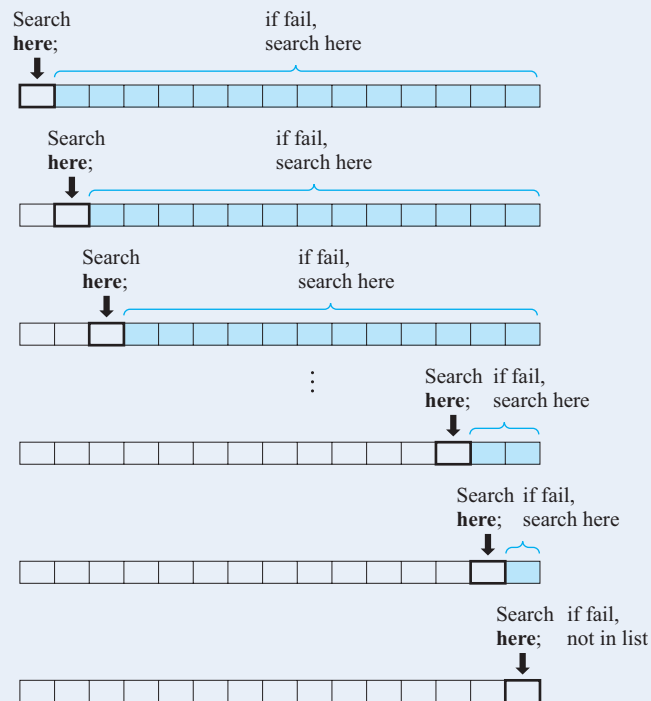


Figure 3.3

Let $C(n)$ represent the maximum number of comparisons required for an n -element list. This is a symbolic expression for an answer we are assuming we don't already know, but will learn by solving the recurrence relation. By this notation, $C(n - 1)$ symbolically represents the maximum number of comparisons required to search the rest of the list after the first position. The recurrence relation is

$$\begin{aligned} C(1) &= 1 && \text{(1 comparison to search a 1-element list)} \\ C(n) &= 1 + C(n - 1) \text{ for } n \geq 2 && \text{(1 comparison against the first element,} \\ &&& \text{then however many comparisons} \\ &&& \text{are required for the rest of the list)} \end{aligned}$$

This is a first-order, linear recurrence relation with constant coefficients. By Equation (8) in Section 3.2, the solution is

$$C(n) = (1)^{n-1}(1) + \sum_{i=2}^n (1)^{n-i}(1) = 1 + (n - 1) = n$$

This agrees with our previous analysis for the worst case. ●

EXAMPLE 30

Now let's do a worst-case analysis of the binary search algorithm. Recall that binary search is a recursive algorithm that operates on a list that is sorted in increasing order. It first does one comparison of the target with the midpoint value of the list. If this comparison fails, then the process is repeated on the right half or the left half of the list, depending on whether the target value is greater than or less than the midpoint value. Figure 3.4 illustrates one possible worst-case path.

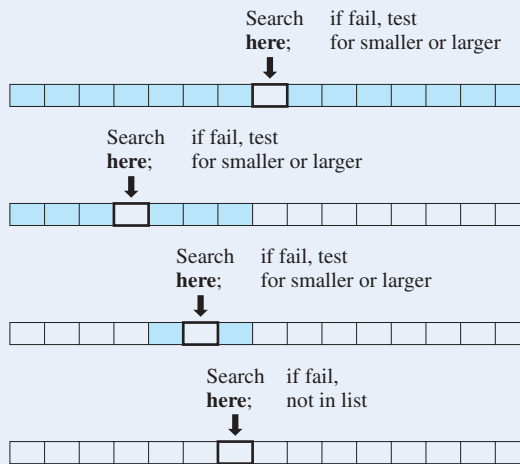


Figure 3.4

Binary search is a **divide-and-conquer algorithm**, where the problem is decomposed recursively into significantly smaller subproblems. If the original list is n elements long, then half the list is at worst $n/2$ elements long. (In the 8-element list of Example 14, for instance, when 10 is the midpoint value, the right “half” of the list has 4 elements but the left “half” has only 3.) Cutting the list in half makes much faster progress than reducing the list by one element, as in sequential search, so we expect the worst case of binary search to require less work.

Let $C(n)$ stand for the maximum number of comparisons required to do a binary search on an n -element list. Then $C\left(\frac{n}{2}\right)$ stands for the maximum number of comparisons required to search a list half that size. If we are to keep cutting the list in half, it is convenient to consider only the case where we get an integer value each time we cut in half, so we will assume that $n = 2^m$ for some $m \geq 0$. The recurrence relation for $C(n)$ is

$$C(1) = 1 \quad \text{(1 comparison to search a 1-element list)}$$

$$C(n) = 1 + C\left(\frac{n}{2}\right) \text{ for } n \geq 2, n = 2^m \quad \text{(1 comparison against the middle element, then however many comparisons are required for half the list)}$$

This recurrence relation was solved in the previous section (Examples 24 and 25). The solution is

$$C(n) = 1 + \log n$$

By the preceding example, the maximum number of comparisons required to do a binary search on an n -element ordered list, with $n = 2^m$, is $1 + \log n$. In Example 14, n was 8, and four comparisons ($1 + \log 8$) were required in the worst case (x not in the list). A sequential search would require eight comparisons. Because

$$1 + \log n < n \text{ for } n = 2^m, n \geq 4$$

binary search is almost always more efficient than sequential search. However, the sequential search algorithm does have one big advantage—if the list being searched is *unsorted*, the sequential search algorithm works, but the binary search algorithm does not. If we first sort the list and then use the binary search algorithm, we must then consider the work involved in sorting the list. Exercises 13–34 at the end of this section ask you to count the operations required to sort a list by several different algorithms.

Practice 16

Fill in the following table for the worst-case number of comparisons required for sequential search and binary search on a list of the indicated size.

n	Sequential Search	Binary Search
64		
1024		
32768		

Although we have computed the work for binary search only on a list of size n where n is a power of 2, this gives us a range for the work required for values of n

that fall between powers of 2. That's why limiting n to powers of 2 in our analysis is not particularly significant.

Upper Bound (Euclidean Algorithm)

The Euclidean algorithm, as presented in Section 2.3, uses a while loop to do successive divisions to find $\gcd(a, b)$ for positive integers a and b , $a > b$. To analyze the Euclidean algorithm, we must first decide on the operation we are counting. Because the Euclidean algorithm does repeated divisions, we'll take the division operation as our unit of work. Because $a > b$, we can take a as a measure of the size of the input values (which we usually denote by n). We want to find $E(a)$, where this denotes the amount of work (the number of divisions) required to find $\gcd(a, b)$ in the worst case.

A recursive version of the Euclidean algorithm can also be written (see Exercise 81, Section 3.1); the key to the recursive version is to recognize that $\gcd(a, b)$ involves finding $\gcd(b, r)$, where r is the remainder on dividing a by b . We've just seen a case where the operations of a recursive algorithm (binary search) could be expressed neatly as a recurrence relation where the input size gets halved after each operation. A recurrence relation would express $E(a)$ in terms of E at smaller values. But what are these smaller values? To find $\gcd(a, b)$ we find $\gcd(b, r)$, so it is clear that the input size is getting smaller, but in what way? Consider Example 27 of Section 2.3, where to find $\gcd(420, 66)$ the following divisions were performed:

$$\begin{array}{r} 6 \\ 66 \overline{)420} \\ \underline{396} \\ 24 \end{array} \quad \begin{array}{r} 2 \\ 24 \overline{)66} \\ \underline{48} \\ 18 \end{array} \quad \begin{array}{r} 1 \\ 18 \overline{)24} \\ \underline{18} \\ 6 \end{array} \quad \begin{array}{r} 3 \\ 6 \overline{)18} \\ \underline{18} \\ 0 \end{array}$$

Here the successive values being divided are 420, 66, 24, 18. The change from 420 to 66 is much larger than cutting in half, while the change from 24 to 18 is less.

In fact, we won't find a recurrence relation or an exact expression for $E(a)$. But we will at least find an *upper bound* for $E(a)$. An **upper bound** is a ceiling on the amount of work an algorithm does; the algorithm can require *no more steps* than the upper bound, but it may not require that many.

To find this upper bound, we will show that if $i > j$ and i is divided by j with remainder r , then $r < i/2$. There are two cases:

1. If $j \leq i/2$, then $r < i/2$ because $r < j$.
2. If $j > i/2$, then $i = 1 * j + (i - j)$; in other words, the quotient is 1 and r is $i - j$, which is $< i/2$.

In the Euclidean algorithm, the remainder r at any step becomes the dividend (the number being divided) two steps later. So the successive dividends are at least halved every two divisions. The value a can be halved $\log n$ times, therefore at most $2 \log n$ divisions are done. Thus

$$E(a) \leq 2 \log a \tag{1}$$

The value of $2 \log a$ for $a = 420$ is almost 18, whereas it took only 4 divisions to find $\gcd(420, 66)$. Evidently this upper bound estimate is rather loose, like saying that every student in this class is under 12 feet in height. An improved (that is, lower) upper bound is derived in Exercises 37–40 at the end of this section.

Of Trees ... and Pancakes

Of Trees ...

Mapping the evolutionary “tree of life” has been the subject of research since Charles Darwin. Until recently, this research sought to find similarities between species based on structural properties such as skeletons, but today scientists search for similarities in DNA and other genetic evidence. This field of research, called phylogenetics, can involve aligning the molecular sequences of many thousands of species, and the work becomes an enormous computational problem. Researchers at the University of Texas have developed a software package called SATé (and a new and improved SATé-II)—Simultaneous Alignment and Tree Estimation—that uses a divide-and-conquer algorithm. Huge data sets are divided into small data sets, alignments are found for the small sets, and then the results are combined to determine an overall alignment (and a likely tree) for the full data set. The resulting full alignment isn’t foolproof, and the software repeats this process many times, creating new alignments and trees. A statistical “maximum likelihood” method selects the best result by comparison with known answers. This approach has been proven to produce results comparable to other, slower methods, or to produce more accurate results in the same amount of time.

Sequence alignment and evolutionary tree-building tools have applications to areas other than tracing the path of historical evolution. For example, the Centers for Disease Control use them to detect how a newly emerging virus differs from previous viruses in order to plan the best counterattack.

<http://www.tacc.utexas.edu/news/feature-stories/2012/tree-of-life>

<http://www.ncbi.nlm.nih.gov/pubmed/22139466>

... and Pancakes

A problem posed in the *American Mathematical Monthly* in 1975 by Jacob Goodman concerned a waiter in a café where the cook produced a stack of pancakes of varying sizes. The waiter, on the way to delivering the stack to the customer, attempted to arrange the pancakes in order by size, with the largest on the bottom. The only action available was to stick a spatula into the stack at some point and flip the entire stack above that point. The question is: What is the maximum number of flips ever needed for any stack of n pancakes? This number, P_n , is known as the *nth* pancake number.

Here’s a fairly simple algorithm to arrange the pancakes. Put the spatula under the largest pancake, and flip. This puts the largest pancake on top. Put the spatula at the bottom of the unordered section (in this case at the bottom) and flip. This puts the largest pancake on the bottom, where it belongs. Repeat with the rest of the pancakes. Each pancake therefore requires two flips, which would give a total of $2n$ flips required. But the last two pancakes require at most one flip; if they are already in order, no flips are needed, and if they are out of order, only one flip is needed. So this algorithm requires at most $2(n - 2) + 1 = 2n - 3$ flips in the worst case, which means that $P_n \leq 2n - 3$. Are there other algorithms that require fewer flips in the worst case?

A faculty member at Harvard University posed this question to his class; several days later, a sophomore from the class came to his office with a better algorithm. This algorithm, which requires at most $(5n + 5)/3$ flips, was published in the journal *Discrete Mathematics* in 1979. The authors were William Gates (the student) and Christos Papadimitriou.

Yes, THAT William Gates.

From SCHNEIDER/GERSTING. *Invitation to Computer Science*, 6/E. © 2013 South-Western, a part of Cengage Learning, Inc. Reproduced by permission.
www.cengage.com/permissions

SECTION 3.3 REVIEW

TECHNIQUE

- Do a worst-case analysis of an algorithm either directly from the algorithm description or from a recurrence relation.

- Analysis of recursive algorithms often leads to recurrence relations.
- Lacking an exact expression for the number of operations an algorithm performs, it may be possible to find an upper bound.

MAIN IDEAS

- Analysis of an algorithm estimates the number of basic operations that the algorithm performs, which is dependent on the size of the input.

EXERCISES 3.3

1. Modify the algorithm of Example 27 so that in addition to dropping the student's lowest quiz grade, the highest quiz grade is counted twice (like the old version, your new algorithm should do no operations besides addition and subtraction).
2. What is the total number of arithmetic operations done in the algorithm of Exercise 1?
3. The following algorithm adds all the entries in a square $n \times n$ array A . Analyze this algorithm where the work unit is the addition operation.

```

sum = 0
for i = 1 to n do
  for j = 1 to n do
    sum = sum + A[i,j]
  end for
end for
write ("Total of all array elements is", sum)

```

4. The following algorithm adds all the entries in the "upper triangular" part of a square $n \times n$ array A . Analyze this algorithm where the work unit is the addition operation.

```

sum = 0
for k = 1 to n do
  for j = k to n do
    sum = sum + A[k,j]
  end for
end for
write ("Total of all upper triangular array elements is", sum)

```

5. Analyze the following algorithm where the work unit is the output statement. Assume that $n = 2^m$ for some positive integer m .

```

integer j, k
for k = 1 to n do
  j = n;
  while j ≥ 2 do
    write j
    j = j/2
  end while
end for

```

6. Analyze the following algorithm where the work unit is the output statement. (*Hint*: One of the exercises in Section 2.2 might be helpful).

```

integer i
real d, x;
for i = 1 to n do
  d = 1.0/i;
  x = i;
  while x > 0 do
    write x
    x = x - d;
  end while
end for

```

Exercises 7 and 8 involve $n! = n(n-1)(n-2)\cdots 1$.

7. a. Write the body of an iterative function to compute $n!$ for $n \geq 1$.
 b. Analyze this function where the work unit is the multiplication operation.
8. a. Write a recursive function to compute $n!$ for $n \geq 1$.
 b. Write a recurrence relation for the work done by this function where multiplication is the unit of work.
 c. Solve the recurrence relation of part b.
 d. Compare your answer in part c to your result in Exercise 7b.

Exercises 9 and 10 involve evaluating a polynomial $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$ for a specific value of x .

9. A straightforward algorithm to evaluate a polynomial is given by the following function:

```

Poly(real an, real an-1, ..., real a0, real c, integer n)
//evaluates polynomial anxn + an-1xn-1 + ... + a0    for x = c
Local variables:
integer i
real sum = a0
real product = 1

  for i = 1 to n do
    product = product * c
    sum = sum + ai * product
  end for
  return sum
end function Poly

```

- a. Walk through this algorithm to compute the value of $2x^3 - 7x^2 + 5x - 14$ for $x = 4$.
 b. The algorithm involves both additions and multiplications; analyze this algorithm where those operations are the work units.
10. An alternative to the polynomial evaluation algorithm in Exercise 9 is an algorithm called *Horner's method*. Horner's method relies on an alternative expression for a polynomial, for example

$$2x^3 - 7x^2 + 5x - 14 = -14 + x(5 + x(-7 + x(2)))$$

```

Horner(real  $a_n$ , real  $a_{n-1}$ , ..., real  $a_0$ , real  $c$ , integer  $n$ )
//evaluates polynomial  $a_n x^n + a_{n-1} x^{n-1} + \dots + a_0$  for  $x = c$ 
//using Horner's method
Local variables:
integer  $i$ 
real  $result = a_n$ 

    for  $i = 1$  to  $n$  do
         $result = result * c + a_{n-i}$ 
    end for
    return  $result$ 
end function Horner

```

- a. Walk through this algorithm to compute the value of $2x^3 - 7x^2 + 5x - 14$ for $x = 4$.
 - b. Analyze this algorithm where addition and multiplication operations are the work units.
 - c. In evaluating a polynomial of degree $n = 98$ for some value of x , how many operations have been saved by using Horner's method over the method of Exercise 9?
11. For the algorithm of Example 27, count the total number of assignments and comparisons done in the best case (least work) and the worst case (most work); describe each of these cases.
 12. a. Write a function to convert a binary string $b_n b_{n-1} \dots b_1 b_0$ to its decimal equivalent.
 - b. Test your function on the binary string 10011
 - c. Describe the worst case for this algorithm and find the number of multiplications and additions done in this case.
 - d. Describe the best case for this algorithm and find the number of multiplications and additions done in this case.

Exercises 13 and 14 relate to a recursive sorting algorithm called *BubbleSort*.

13. Algorithm *BubbleSort* works by making repeated passes through a list; on each pass, adjacent elements that are out of order are exchanged. At the end of pass 1, the maximum element has "bubbled up" to the end of the list and does not participate in subsequent passes. The following algorithm is called initially with $j = n$.

```

BubbleSort(list  $L$ ; integer  $j$ )
//recursively sorts the items from 1 to  $j$  in list  $L$  into increasing order

    if  $j = 1$  then
        sort is complete, write out the sorted list
    else
        for  $i = 1$  to  $j - 1$  do
            if  $L[i] > L[i + 1]$  then
                exchange  $L[i]$  and  $L[i + 1]$ 
            end if
        end for
        BubbleSort( $L, j - 1$ )
    end if
end function BubbleSort

```

- a. Walk through algorithm *BubbleSort* to sort the list 5, 6, 3, 4, 8, 2.
 - b. Write a recurrence relation for the number of comparisons of list elements done by this algorithm to sort an n -element list.
 - c. Solve this recurrence relation.
14. In algorithm *BubbleSort*, suppose we include exchanges of list elements as a work unit, in addition to comparisons between list elements.
- a. Describe the worst case and find the number of comparisons and exchanges done in this case.
 - b. Describe the best case and find the number of comparisons and exchanges done in this case.
 - c. Assume that on the average exchanges between elements must be done about half the time. Find the number of comparisons and exchanges done in this case.

Exercises 15–18 refer to the recursive algorithm *SelectionSort* of Section 3.1.

15. In one part of algorithm *SelectionSort*, the index of the maximum item in a list must be found. This requires comparisons between list elements. In an n -element (unsorted) list, how many such comparisons are needed in the worst case to find the maximum element? How many such comparisons are needed in the average case?
16. Defining the basic operation as the comparison of list elements and ignoring the amount of work required to exchange list elements, write a recurrence relation for the amount of work done by selection sort on an n -element list. (*Hint*: Use the result from Exercise 15.)
17. Solve the recurrence relation of Exercise 16.
18. Assume that the exchange of $L[i]$ and $L[j]$ takes place even if $i = j$. Write an expression for the total number of comparisons and exchanges done to sort an n -element list.

Exercises 19–24 relate to a recursive sorting algorithm called *MergeSort*, which is described as follows: A one-element list is already sorted; no further work is required. Otherwise, split the list in half, sort each half using *MergeSort* (this is the recursive part), and then merge the two halves back into one sorted list.

19. The merge part of algorithm *MergeSort* requires comparing elements from each of two sorted lists to see which goes next into the combined, sorted list. When one list runs out of elements, the remaining elements from the other list can be added without further comparisons. Given the following pairs of lists, perform a merge and count the number of comparisons to merge the two lists into one.
- a. 6, 8, 9 and 1, 4, 5
 - b. 1, 5, 8 and 2, 3, 4
 - c. 0, 2, 3, 4, 7, 10 and 1, 8, 9
20. Under what circumstances will the maximum number of comparisons take place while merging two sorted lists? If the lengths of the lists are r and s , what is the maximum number of comparisons?
21. Write a recurrence relation for the number of comparisons between list elements done by algorithm *MergeSort* in the worst case. Assume that $n = 2^m$.
22. Solve the recurrence relation of Exercise 21.
23. Use the results of Exercises 18 and 22 to compare the worst-case behavior of *SelectionSort* (counting comparisons and exchanges) and *MergeSort* (counting comparisons) for $n = 4, 8, 16,$ and 32 (use a calculator or spreadsheet).
24. Use the results of Exercises 14 and 22 to compare the worst-case behavior of *BubbleSort* (counting comparisons and exchanges) and *MergeSort* (counting comparisons) for $n = 4, 8, 16,$ and 32 (use a calculator or spreadsheet).

Exercises 25–34 relate to a recursive sorting algorithm called *QuickSort*, which is described as follows: A one-element list is already sorted; no further work is required. Otherwise, take the first element in the list, call it the

pivot element, then walk through the original list to create two new sublists, L_1 and L_2 . L_1 consists of all elements that are less than the pivot element and L_2 consists of all elements that are greater than the pivot element. Put the pivot element between L_1 and L_2 . Sort each of L_1 and L_2 using *QuickSort* (this is the recursive part). Eventually all lists will consist of 1 element sublists separated by previous pivot elements, and at this point the entire original list is in sorted order. This is a little confusing, so here is an example, where pivot elements are shown in brackets:

Original list: 6, 2, 1, 7, 9, 4, 8
 After 1st pass: 2, 1, 4, [6], 7, 9, 8
 After 2nd pass: 1, [2], 4, [6], [7], 9, 8
 After 3rd pass: 1, [2], 4, [6], [7], 8, [9] Sorted

25. Illustrate *QuickSort* as above using the list 9, 8, 3, 13.
26. Illustrate *QuickSort* as above using the list 8, 4, 10, 5, 9, 6, 14, 3, 1, 12, 11.
27. How many comparisons between list elements are required for pass 1 of *QuickSort* in the example list?
28. How many comparisons between list elements are required for pass 1 of *QuickSort* on an n -element list?
29. Suppose that for each pass, each pivot element splits its sublist into two equal-length lists, each approximately half the size of the sublist (which is actually very difficult to achieve). Write a recurrence relation for the number of comparisons between list elements in this case.
30. Solve the recurrence relation of Exercise 29.
31. Suppose that for each pass, each pivot element splits its sublist (which has k elements) into one empty list and one list of size $k - 1$. Write a recurrence relation for the number of comparisons between list elements in this case.
32. Solve the recurrence relation of Exercise 31.
33. Unlike the situation described in Exercise 29 where each pivot element splits the sublist in half for the next pass, the situation described in Exercise 31 can easily occur. Describe a characteristic of the original list that would cause this to happen.
34. Exercise 29 describes the best case of *QuickSort* and Exercise 31 describes the worst case of *QuickSort* with respect to comparisons between list elements.
 - a. To which sorting algorithm (*SelectionSort*, *BubbleSort*, *MergeSort*) is the best case of *QuickSort* comparable in the number of comparisons required?
 - b. To which sorting algorithm (*SelectionSort*, *BubbleSort*, *MergeSort*) is the worst case of *QuickSort* comparable in the number of comparisons required?

Exercises 35 and 36 refer to algorithm *SequentialSearch*. It is not hard to do an average case analysis of the sequential search algorithm under certain assumptions. Given an n -element list and a target value x for which we are searching, the basic operation is a comparison of list elements to x , hence an analysis should count how many times such an operation is performed “on the average.” The definition of “average” is shaped by our assumptions.

35. Assume that x is in the list and is equally to be found at any of the n positions in the list. Fill in the rest of the table giving the number of comparisons for each case.

Position at Which x Occurs	Number of Comparisons
1	1
2	
3	
\vdots	
n	

Find the average number of comparisons by adding the results from the table and dividing by n . (*Hint*: See Practice 7 of Section 2.2—we told you that you should remember this!)

36. Find the average number of comparisons under the assumption that x is equally likely to be at any of the n positions in the list or not in the list.

Exercises 37–40 concern a better upper bound for the number of divisions required by the Euclidean algorithm in finding $\gcd(a, b)$. Assume that a and b are positive integers with $a > b$.

37. Suppose that m divisions are required to find $\gcd(a, b)$. Prove by induction that for $m \geq 1$, it is true that $a \geq F(m + 2)$ and $b \geq F(m + 1)$, where $F(n)$ is the Fibonacci sequence. (*Hint*: To find $\gcd(a, b)$, after the first division the algorithm computes $\gcd(b, r)$.)
38. Suppose that m divisions are required to find $\gcd(a, b)$, with $m \geq 4$. Prove that

$$\left(\frac{3}{2}\right)^{m+1} < F(m + 2) \leq a$$

(*Hint*: Use the result of Exercise 37 here and Exercise 26 of Section 3.1.)

39. Suppose that m divisions are required to find $\gcd(a, b)$, with $m \geq 4$. Prove that $m < (\log_{1.5} a) - 1$. (*Hint*: Use the result of Exercise 38.)
40. a. Compute $\gcd(89, 55)$ and count the number of divisions required.
 b. Compute the upper bound on the number of divisions required for $\gcd(89, 55)$ using Equation (1).
 c. Compute the upper bound on the number of divisions required for $\gcd(89, 55)$ using the result of Exercise 39.
 d. The eighteenth-century French mathematician Gabriel Lamé proved that an upper bound on the number of division done by the Euclidean algorithm to find $\gcd(a, b)$ where $a > b$ is 5 times the number of decimal digits in b . Compute the upper bound on the number of divisions required for $\gcd(89, 55)$ using Lamé's theorem.

CHAPTER 3 REVIEW

TERMINOLOGY

analysis of algorithms (p. 203)	empty string (p. 163)	selection sort algorithm (p. 168)
Backus–Naur form (BNF) (p. 163)	Fibonacci sequence (p. 159)	sequence (infinite sequence) (p. 158)
binary search algorithm (p. 169)	first-order recurrence relation (p. 182)	sequential search algorithm (p. 204)
binary string (p. 163)	homogeneous recurrence relation (p. 182)	solving a recurrence relation (p. 180)
characteristic equation of a recurrence relation (p. 190)	index of summation (p. 182)	structural induction (p. 164)
closed–form solution (p. 180)	inductive definition (p. 158)	summation notation (p. 182)
concatenation (p. 163)	linear recurrence relation (p. 182)	upper bound (p. 210)
constant coefficient recurrence relation (p. 182)	palindrome (p. 163)	
divide-and-conquer algorithm (p. 208)	recurrence relation (p. 159)	
divide-and-conquer recurrence relation (p. 193)	recursive definition (p. 158)	
	second-order recurrence relation (p. 188)	

SELF-TEST

Answer the following true-false questions without looking back in the chapter.

Section 3.1

1. A sequence defined by

$$S(1) = 7$$

$$S(n) = 3S(n - 1) + 2 \quad \text{for } n \geq 2$$

contains the number 215.

2. A collection T of numbers is defined recursively by

1. 6 and 8 belong to T

2. If X and Y belong to T , so does $X + 2Y$

Every even number ≥ 18 belongs to T .

3. Recursive algorithms are valuable primarily because they run more efficiently than iterative algorithms.

4. In the recursive algorithm *SelectionSort*, changing one line of the algorithm to

“find the index i of the minimum item in L between 1 and j ”

sorts the list L in decreasing order.

5. In applying the binary search algorithm to the list

2, 5, 7, 10, 14, 20

where $x = 8$ is the target item, x is never compared to 5.

Section 3.2

1. A closed-form solution to a recurrence relation is obtained by applying mathematical induction to the recurrence relation.

2. $S(n) = 2S(n - 1) + 3S(n - 2) + 5n$ is a linear, first-order recurrence relation with constant coefficients.

3. $S(n) = c^{n-1}S(1) + \sum_{i=2}^n c^{n-i}g(i)$ is a closed-form

solution to any linear first-order recurrence relation with constant coefficients.

4. The solution to the recurrence relation $S(n) = c_1S(n - 1) + c_2S(n - 2)$ involves solving the characteristic equation $t^2 - c_1t - c_2 = 0$.

5. Divide-and-conquer algorithms lead to recurrence relations that are not first-order.

Section 3.3

1. Analysis of an algorithm generally finds the amount of work done in the worst case because it is too difficult to analyze an average case.

2. In the worst case, the string pattern-matching algorithm requires $n + m$ comparisons, where $n =$ the text size and $m =$ the pattern size.

3. Binary search is more efficient than sequential search on a sorted list of more than three elements.

4. The recursive version of the sequential search algorithm is a divide-and-conquer algorithm.

5. An upper bound for the Euclidean algorithm gives a ceiling on the number of divisions required to find $\text{gcd}(a, b)$.

ON THE COMPUTER

For Exercises 1–7, write a computer program that produces the desired output from the given input.

1. *Input:* Binary string

Output: Message indicating whether the input string is a palindrome (see Practice 7)

Algorithm: Use recursion.

2. *Input:* String of characters x and a positive integer n

Output: Concatenation of n copies of x

Algorithm: Use recursion.

(Some programming languages provide built-in string manipulation capabilities, such as concatenation.)

3. *Input:* Positive integer n

Output: n th value in the Fibonacci sequence using

a. iteration.

b. recursion.

Now insert a counter in each version to indicate the total number of addition operations done. Run each version for various values of n and, on a single graph, plot the number of additions as a function of n for each version.

4. *Input:* Two positive integers a and b with $a > b$
Output: $\text{gcd}(a, b)$ using
 - a. the iterative version of the Euclidean algorithm
 - b. a recursive version of the Euclidean algorithm
5. *Input:* Unsorted list of 10 integers
Output: Input list sorted in increasing order
Algorithm: Use the recursive selection sort of Example 12.
6. *Input:* Sorted list of 10 integers and an integer x
Output: Message indicating whether x is in the list
Algorithm: Use the binary search algorithm of Example 13.
7. *Input:* Text string, pattern string
Output: Location of beginning of pattern string in text string, or a message that the pattern string is not found within the text string
Algorithm: See Example 28.
8. The value $(1 + \sqrt{5})/2$, known as the *golden ratio*, is related to the Fibonacci sequence by
$$\lim_{n \rightarrow \infty} \frac{F(n+1)}{F(n)} = \frac{1 + \sqrt{5}}{2}$$
Verify this limit by computing $F(n+1)/F(n)$ for $n = 10, 15, 25, 50,$ and 100 and comparing the result with the golden ratio.
9. Compare the work done by sequential search and binary search on an ordered list of n entries by computing n and $1 + \log n$ for values of n from 1 to 100. Present the results in graphic form.

This page intentionally left blank

Sets, Combinatorics, and Probability

CHAPTER OBJECTIVES

After studying this chapter, you will be able to:

- Use the notation of set theory.
- Find the power set of a finite set.
- Find the union, intersection, difference, complement, and Cartesian product of sets.
- Identify binary and unary operations on a set.
- Prove set identities.
- Recognize that not all sets are countable.
- Apply the multiplication principle and the addition principle to solve counting problems.
- Use decision trees to solve counting problems.
- Use the principle of inclusion and exclusion to find the number of elements in the union of sets.
- Use the pigeonhole principle to decide when certain events must occur.
- Use the formulas for permutations and combinations of r objects, with and without repetition, from a set of n distinct objects.
- Find the number of distinct permutations of n objects that are not all distinct.
- Generate all permutations of n distinct objects, and all combinations of r out of n distinct objects.
- Find the probability of an event given that all outcomes are equally likely, that a probability distribution has been assigned to the outcomes, or that another event has already occurred.
- Compute the expected value of a quantity with an assigned probability distribution.
- Use the binomial theorem to expand $(a + b)^n$.

You survey the 87 computer users who subscribe to your electronic newsletter in preparation for the release of your new software product. The results of your survey reveal that 68 have a Windows-based system available to them, 34 have a Linux system available, and 30 have access to a Mac. In addition, 19 have access to both Windows and Linux systems, 11 have access to both Linux systems and Macs, and 23 can use both Macs and Windows.

Question: How many of your subscribers have access to all three types of systems?

This is an example of a counting problem; you want to count the number of elements in a certain collection or set—the set of all subscribers with access to all three systems. A formula that easily solves this counting problem is developed in Section 4.3.

Set theory is one of the cornerstones of mathematics. Many concepts in mathematics and computer science can be conveniently expressed in the language of sets. Operations can be performed on sets to generate new sets. Although most sets of interest to computer scientists are finite or countable, there are sets with so many members that they cannot be enumerated. Set theory is discussed in Section 4.1.

It is often of interest to count the number of elements in a finite set. This may not be a trivial task. Section 4.2 provides some ground rules for counting the number of elements in a set consisting of the outcomes of an event. Counting the elements in such a set can be made manageable by breaking the event down into a sequence of subevents or into disjoint subevents that have no outcomes in common. Some specialized counting principles appear in Section 4.3. Section 4.4 provides formulas for counting the number of ways to arrange objects in a set and to select objects from a set, as well as algorithms to generate all the possible arrangements or selections. Section 4.5 discusses the binomial theorem, an algebraic result that can also be viewed as a consequence of the counting formulas. Finally, Section 4.6 extends “counting” to the more general idea of probability.

SECTION 4.1 SETS

Definitions are important in any science because they contribute to precise communication. However, if we look up a word in the dictionary, the definition is expressed using other words, which are defined using still other words, and so on. Thus, we have to have a starting point for definitions where the meaning is taken to be understood; our starting point in this discussion will be the idea of a set, a term that we will not formally define. Instead, we will simply use the intuitive idea that a set is a collection of objects. Usually all of the objects in a set share some common property (aside from that of belonging to the same set!); any object that has the property is a member of the set, and any object that does not have the property is not a member. (This is consistent with our use of the word *set* in Section 3.1, where we talked about the set of propositional well-formed formulas, the set of all strings of symbols from a finite alphabet, and the set of identifiers in some programming language.)

Notation

We use capital letters to denote sets and the symbol \in to denote membership in a set. Thus $a \in A$ means that object a is a member, or element, of set A , and $b \notin A$ means that object b is not an element of set A . Braces are used to indicate a set.

EXAMPLE 1 If $A = \{\text{violet, chartreuse, burnt umber}\}$, then $\text{chartreuse} \in A$ and $\text{magenta} \notin A$. ●

No ordering is imposed on the elements in a set; therefore $\{\text{violet, chartreuse, burnt umber}\}$ is the same as $\{\text{chartreuse, burnt umber, violet}\}$. Also, each element of a set is listed only once; it is redundant to list it again.

Two sets are **equal** if they contain the same elements. (In a definition, “if” really means “if and only if”; thus two sets are equal if and only if they contain the same elements.) Using predicate logic notation,

$$A = B \text{ means } (\forall x)[(x \in A \rightarrow x \in B) \wedge (x \in B \rightarrow x \in A)]$$

In describing a particular set, we have to identify its elements. For a **finite set** (one with n elements for some nonnegative integer n), we might do this by simply listing all the elements, as in set A of Example 1. Although it is impossible to list all elements of an **infinite set** (one that is not finite), for some infinite sets we can indicate a pattern for listing elements indefinitely. Thus, we might write $\{2, 4, 6, \dots\}$ to express the set S of all positive even integers. (Although this is a common practice, the danger exists that the reader will not see the pattern that the writer has in mind.) S can also be defined recursively by giving an explicit member of S and then describing other members of S in terms of already known members. For example,

1. $2 \in S$
2. If $n \in S$, then $(n + 2) \in S$

But the clearest way to describe this particular set S is to describe the characterizing property of the set elements in words and write

$$S = \{x \mid x \text{ is a positive even integer}\}$$

read as “the set of all x such that x is a positive even integer.”

We’ve now given three ways to describe a set:

1. List (or partially list) its elements.
2. Use recursion to describe how to generate the set elements.
3. Describe a property P that characterizes the set elements.

Later in this section we’ll see that there are sets for which the first approach won’t work; often the second approach is difficult to use. The third method is usually the best choice.

The notation for a set S whose elements are characterized as having property P is $\{x \mid P(x)\}$. Property P here is a *unary predicate*; this term was introduced in Chapter 1. For any given x , $P(x)$ is either true or false. In fact, the formal logic notation of Chapter 1 again comes to the rescue to clarify what we mean by a characterizing property of a set’s elements:

$$S = \{x \mid P(x)\} \text{ means } (\forall x)[(x \in S \rightarrow P(x)) \wedge (P(x) \rightarrow x \in S)]$$

In words, every element of S has property P and everything that has property P is an element of S .

PRACTICE 1 Describe each of the following sets by listing its elements.

- a. $\{x \mid x \text{ is an integer and } 3 < x \leq 7\}$
- b. $\{x \mid x \text{ is a month with exactly 30 days}\}$
- c. $\{x \mid x \text{ is the capital of the United States}\}$

PRACTICE 2 Describe each of the following sets by giving a characterizing property.

- a. $\{1, 4, 9, 16\}$
- b. $\{\text{the butcher, the baker, the candlestick maker}\}$
- c. $\{2, 3, 5, 7, 11, 13, 17, \dots\}$

It is convenient to name certain standard sets so that we can refer to them easily. We will use

\mathbb{N} = set of all nonnegative integers (note that $0 \in \mathbb{N}$)

\mathbb{Z} = set of all integers

\mathbb{Q} = set of all rational numbers

\mathbb{R} = set of all real numbers

\mathbb{C} = set of all complex numbers

Sometimes we will also want to talk about the set with no elements (the **empty set**, or **null set**), denoted by \emptyset or $\{\}$. For example, if $S = \{x \mid x \in \mathbb{N} \text{ and } x < 0\}$, then $S = \emptyset$. Note that \emptyset , the set with no elements, is not the same as $\{\emptyset\}$, which is a set with a single element where the single element is the empty set.

EXAMPLE 2

Suppose that a set A is described as

$$A = \{x \mid (\exists y)(y \in \{0, 1, 2\} \text{ and } x = y^3)\}$$

Because y is not a free variable here, this is still of the form $A = \{x \mid P(x)\}$. The members of A can be found by letting y assume each of the values 0, 1, and 2 and then taking the cube of each such value. Therefore $A = \{0, 1, 8\}$. For

$$B = \{x \mid x \in \mathbb{N} \text{ and } (\exists y)(y \in \mathbb{N} \text{ and } x \leq y)\}$$

choosing $y = 0$ gives $x = 0$; choosing $y = 1$ gives $x = 0$ or 1; choosing $y = 2$ gives $x = 0, 1, \text{ or } 2$; and so on. In other words, B consists of all nonnegative integers that are less than or equal to some nonnegative integer, which means that $B = \mathbb{N}$. For the set

$$C = \{x \mid x \in \mathbb{N} \text{ and } (\forall y)(y \in \mathbb{N} \rightarrow x \leq y)\}$$

0 is the only nonnegative integer that is less than or equal to every nonnegative integer, so $C = \{0\}$. •

PRACTICE 3 Describe each set.

a. $A = \{x \mid x \in \mathbb{N} \text{ and } (\forall y)(y \in \{2, 3, 4, 5\} \rightarrow x \geq y)\}$

b. $B = \{x \mid (\exists y)(\exists z)(y \in \{1, 2\} \text{ and } z \in \{2, 3\} \text{ and } x = y + z)\}$ ■

Relationships Between Sets

For $A = \{2, 3, 5, 12\}$ and $B = \{2, 3, 4, 5, 9, 12\}$, every member of A is also a member of B . When this happens, A is said to be a *subset* of B .

PRACTICE 4 Complete the definition: A is a **subset** of B if

$$(\forall x)(x \in A \rightarrow \underline{\hspace{2cm}}) \quad \text{■}$$

If A is a subset of B , we denote the relationship by $A \subseteq B$. If $A \subseteq B$ but $A \neq B$ (there is at least one element of B that is not an element of A), then A is a **proper subset** of B , denoted by $A \subset B$.

PRACTICE 5 Use formal logic notation to define $A \subset B$.

EXAMPLE 3 Let

$$\begin{aligned} A &= \{1, 7, 9, 15\} \\ B &= \{7, 9\} \\ C &= \{7, 9, 15, 20\} \end{aligned}$$

Then the following statements (among others) are all true:

$$\begin{aligned} B &\subseteq C & 15 &\in C \\ B &\subseteq A & \{7, 9\} &\subseteq B \\ B &\subset A & \{7\} &\subset A \\ A &\not\subseteq C & \emptyset &\subseteq C \end{aligned}$$

The last statement ($\emptyset \subseteq C$) is true because the statement $(\forall x)(x \in \emptyset \rightarrow x \in C)$ is true because $x \in \emptyset$ is always false.

REMINDER

Be sure you understand the difference between the symbols \in (element of) and \subseteq (subset of).

PRACTICE 6 Let

$$\begin{aligned} A &= \{x \mid x \in \mathbb{N} \text{ and } x \geq 5\} \\ B &= \{10, 12, 16, 20\} \\ C &= \{x \mid (\exists y)(y \in \mathbb{N} \text{ and } x = 2y)\} \end{aligned}$$

Which of the following statements are true?

- | | |
|---------------------------------|--|
| a. $B \subseteq C$ | g. $\{12\} \in B$ |
| b. $B \subset A$ | h. $\{12\} \subseteq B$ |
| c. $A \subseteq C$ | i. $\{x \mid x \in \mathbb{N} \text{ and } x < 20\} \not\subseteq B$ |
| d. $26 \in C$ | j. $5 \subseteq A$ |
| e. $\{11, 12, 13\} \subseteq A$ | k. $\{\emptyset\} \subseteq B$ |
| f. $\{11, 12, 13\} \subset C$ | l. $\emptyset \notin A$ |

Suppose that $B = \{x \mid P(x)\}$ and that $A \subseteq B$. Because every element of A is also an element of B , and P is a property characterizing all elements of B , then every element in A also has property $P(x)$. The elements of A “inherit” property P . In fact, to prove that $A \subseteq B$, we pick an arbitrary $x \in A$ and show that $P(x)$

holds. If A is a proper subset of B , A 's elements will usually have some additional characterizing property not shared by all elements of B . (This is the same notion of “inheritance” that prevails when a child type, or subtype, or derived type is defined in an object-oriented programming language. The child type inherits all of the properties and operations from the parent type with the addition of specialized local properties or operations as needed.)

EXAMPLE 4

Let

$$B = \{x \mid x \text{ is a multiple of } 4\}$$

and let

$$A = \{x \mid x \text{ is a multiple of } 8\}$$

Then we have $A \subseteq B$. To prove it, let $x \in A$; note that x is a completely arbitrary member of A . We must show that x satisfies the characterizing property of B ; in other words, we must show that x is a multiple of 4. Because we have $x \in A$, x satisfies the characterizing property of A ; that is, x is a multiple of 8 and thus we can write $x = m \cdot 8$ for some integer m . This equation can be written as $x = m \cdot 2 \cdot 4$ or $x = k \cdot 4$, where $k = 2m$, so k is an integer. This shows that x is a multiple of 4, and therefore $x \in B$.

There are numbers (like 12) that are multiples of 4 but not multiples of 8, so $A \subset B$. Another way to describe A is

$$A = \{x \mid x = k \cdot 4 \text{ and } k \text{ is an even number}\}$$

In this form it is clear that A 's elements have inherited the characterizing property of B —being a multiple of 4—but that there is an additional restriction that makes A less general than B . ●

PRACTICE 7 | Let

$$A = \{x \mid x \in \mathbb{R} \text{ and } x^2 - 4x + 3 = 0\}$$

$$B = \{x \mid x \in \mathbb{N} \text{ and } 1 \leq x \leq 4\}$$

Prove that $A \subset B$. ■

We know that A and B are equal sets if they have the same elements. We can restate this equality in terms of subsets: $A = B$ if and only if $A \subseteq B$ and $B \subseteq A$. Proving set inclusion in both directions is the usual way to establish the equality of two sets.

EXAMPLE 5

We will prove that $\{x \mid x \in \mathbb{N} \text{ and } x^2 < 15\} = \{x \mid x \in \mathbb{N} \text{ and } 2x < 7\}$.

Let $A = \{x \mid x \in \mathbb{N} \text{ and } x^2 < 15\}$ and $B = \{x \mid x \in \mathbb{N} \text{ and } 2x < 7\}$. To show that $A = B$, we show $A \subseteq B$ and $B \subseteq A$. For $A \subseteq B$, we must choose an arbitrary member of A —that is, anything satisfying the characterizing property of A —and show that it also satisfies the characterizing property of B . Let $x \in A$. Then x is a nonnegative integer satisfying the inequality $x^2 < 15$. The nonnegative integers with squares less than 15 are 0, 1, 2, and 3, so these integers are the members of A . The double of each of these nonnegative integers is a number less than 7. Hence, each member of A is a member of B , and $A \subseteq B$.

Now we show $B \subseteq A$. Any member of B is a nonnegative integer whose double is less than 7. These numbers are 0, 1, 2, and 3, each of which has a square less than 15, so $B \subseteq A$. ●

Sets of Sets

For a set S , we can form a new set whose elements are all of the subsets of S . This new set is called the **powerset** of S , $\wp(S)$.

EXAMPLE 6

For $S = \{0, 1\}$, $\wp(S) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$. Note that the members of the power set of a set are themselves sets. ●

REMINDER

To find $\wp(S)$, start with \emptyset . Then add sets taking 1 element from S at a time, then 2 elements at a time, then 3 at a time, and so forth.

For any set S , $\wp(S)$ will always have at least \emptyset and S itself as members, since $\emptyset \subseteq S$ and $S \subseteq S$ are always true.

PRACTICE 8 For $A = \{1, 2, 3\}$, what is $\wp(A)$? ■

In Practice 8, A has 3 elements and $\wp(A)$ has 8 elements. Try finding $\wp(S)$ for other sets S until you can guess the answer to the following practice problem.

PRACTICE 9 If S has n elements, then $\wp(S)$ has _____ elements. (Does your answer work for $n = 0$, too?) ■

There are several ways we can show that for a set S with n elements, $\wp(S)$ will have 2^n elements. The following proof uses induction. For the basis step of the induction, we let $n = 0$. The only set with 0 elements is \emptyset . The only subset of \emptyset is \emptyset , so $\wp(\emptyset) = \{\emptyset\}$, a set with $1 = 2^0$ elements. We assume that for any set with k elements, the power set has 2^k elements.

Now let S have $k + 1$ elements and put one of these elements, call it x , aside. The remaining set has k elements, so by our inductive assumption, its power set has 2^k elements. Each of these elements is also a member of $\wp(S)$. The only members of $\wp(S)$ not counted by this procedure are those including element x . All the subsets including x can be found by taking all those subsets not including x (of which there are 2^k) and throwing in the x ; thus, there will be 2^k subsets including x . Altogether, there are 2^k subsets without x and 2^k subsets with x , or $2^k + 2^k = 2 \cdot 2^k = 2^{k+1}$ subsets. Therefore, $\wp(S)$ has 2^{k+1} elements.

Analogy with the truth tables of Section 1.1 is another way to show that $\wp(S)$ has 2^n elements for a set S with n elements. There we had n statement letters and showed that there were 2^n true-false combinations among these letters. But we can also think of each true-false combination as representing a particular subset, with T indicating membership and F indicating nonmembership in that subset. (For example, the row of the truth table with all statement letters F corresponds to the empty set.) Thus, the number of true-false combinations among n statement letters equals the number of subsets of a set with n elements; both are 2^n .

Binary and Unary Operations

By itself a set is not very interesting until we do something with its elements. For example, we can perform several arithmetic operations on elements of the set \mathbb{Z} . We might subtract two integers, or we might take the negative of an integer. Subtraction acts on two integers; it is a *binary* operation on \mathbb{Z} . Negation acts on one integer; it is a *unary* operation on \mathbb{Z} .

To see exactly what is involved in a binary operation, let's look at subtraction more closely. For any two integers x and y , $x - y$ produces an answer and only one answer, and that answer is always an integer. Finally, subtraction is performed on an **ordered pair** of numbers. For example, $7 - 5$ does not produce the same result as $5 - 7$. An ordered pair is denoted by (x, y) , where x is the first component of the ordered pair and y is the second component. Order is important in an ordered pair; thus, the sets $\{1, 2\}$ and $\{2, 1\}$ are equal, but the ordered pairs $(1, 2)$ and $(2, 1)$ are not. You are probably familiar with ordered pairs used as coordinates to locate a point in the plane. The point $(1, 2)$ is different from the point $(2, 1)$. Two ordered pairs (x, y) and (u, v) are equal only when it is the case that $x = u$ and $y = v$.

PRACTICE 10 Given that $(2x - y, x + y) = (7, -1)$, solve for x and y .

PRACTICE 11 Let $S = \{3, 4\}$. List all the ordered pairs (x, y) of elements of S .

We will generalize the properties of subtraction on the integers to define a binary operation \circ on a set S . The symbol \circ is merely a placeholder; in any specific discussion, it will be replaced by the appropriate operation symbol, such as a subtraction sign.

• **DEFINITION BINARY OPERATION**

\circ is a **binary operation** on a set S if for every ordered pair (x, y) of elements of S , $x \circ y$ exists, is unique, and is a member of S .

In other words, if \circ is a binary operation on S , then for any two values x and y in S , $x \circ y$ produces one and only one answer, and that answer belongs to S . That the value $x \circ y$ always exists and is unique. It is described by saying that the binary operation \circ is **well-defined**. The property that $x \circ y$ always belongs to S is described by saying that S is **closed** under the operation \circ . Uniqueness does not mean that the result of a binary operation occurs only once; it means that for a given x and y , there is only one result. For subtraction, there are many x and y values such that $x - y = 7$, but for a given x and y , like $x = 5$ and $y = 2$, there is only one answer for $x - y$.

EXAMPLE 7

Addition, subtraction, and multiplication are all binary operations on \mathbb{Z} . For example, when we perform addition on the ordered pair of integers (x, y) , $x + y$ exists and is a unique integer. •

EXAMPLE 8

The logical operations of conjunction, disjunction, implication, and equivalence are binary operations on the set of propositional wffs. If P and Q are propositional wffs, then $P \wedge Q$, $P \vee Q$, $P \rightarrow Q$, and $P \leftrightarrow Q$ are unique propositional wffs. •

A candidate \circ for an operation can fail to be a binary operation on a set S in any of three ways: (1) There are elements $x, y \in S$ for which $x \circ y$ does not exist; (2) there are elements $x, y \in S$ for which $x \circ y$ gives more than one result; or (3) there are elements $x, y \in S$ for which $x \circ y$ does not belong to S .

EXAMPLE 9

Division is not a binary operation on \mathbb{Z} because $x \div 0$ does not exist. •

EXAMPLE 10

Define $x \circ y$ on \mathbb{N} by

$$x \circ y = \begin{cases} 1 & \text{if } x \geq 5 \\ 0 & \text{if } x < 5 \end{cases}$$

Then, by the first part of the definition for \circ , $5 \circ 1 = 1$, but by its second part, $5 \circ 1 = 0$. Thus, \circ is not well-defined on \mathbb{N} because the result of $5 \circ 1$ is not unique. •

EXAMPLE 11

Subtraction is not a binary operation on \mathbb{N} because \mathbb{N} is not closed under subtraction. (For example, $1 - 10 \notin \mathbb{N}$.) •

For $\#$ to be a **unary operation** on a set S , it must be true that for any $x \in S$, $x\#$ is well-defined and S is closed under $\#$; in other words, for any $x \in S$, $x\#$ exists, is unique, and is a member of S . We do not have a unary operation if any of these conditions is not met.

EXAMPLE 12

Let $x\#$ be defined by $x\# = -x$ so that $x\#$ is the negative of x . Then $\#$ is a unary operation on \mathbb{Z} but not on \mathbb{N} because \mathbb{N} is not closed under $\#$. ●

EXAMPLE 13

The logical connective of negation is a unary operation on the set of propositional wffs. If P is a propositional wff, then P' is a unique propositional wff. ●

From these examples it is clear that whether \circ (or $\#$) is a binary (or unary) operation can depend not only on its definition but also on the set involved.

PRACTICE 12

Which of the following candidates are neither binary nor unary operations on the given sets? Why not?

- $x \circ y = x \div y$; $S =$ set of all positive integers
- $x \circ y = x \div y$; $S =$ set of all positive rational numbers
- $x \circ y = x^y$; $S = \mathbb{R}$
- $x \circ y =$ maximum of x and y ; $S = \mathbb{N}$
- $x\# = \sqrt{x}$; $S =$ set of all positive real numbers
- $x\# =$ solution to equation $(x\#)^2 = x$; $S = \mathbb{C}$

So far, all our binary operations have been defined by means of a description or an equation. Suppose S is a finite set, $S = \{x_1, x_2, \dots, x_n\}$. Then a binary operation \circ on S can be defined by an $n \times n$ table, where element i, j (i th row and j th column) denotes $x_i \circ x_j$.

EXAMPLE 14

Let $S = \{2, 5, 9\}$, and let \circ be defined by the table

\circ	2	5	9
2	2	2	9
5	5	9	2
9	5	5	9

Thus, $2 \circ 5 = 2$ and $9 \circ 2 = 5$. Inspecting the table, we see that \circ is a binary operation on S . ●

Operations on Sets

Most of the operations we have seen operate on numbers, but we can also operate on sets. Given an arbitrary set S , we can define some binary and unary

operations on the set $\wp(S)$. S in this case is called the **universal set** or the **universe of discourse**. The universal set defines the context of the objects being discussed. If $S = \mathbb{Z}$, for example, then all subsets will contain only integers.

A binary operation on $\wp(S)$ must act on any two subsets of S to produce a unique subset of S . There are at least two natural ways in which this can happen.

EXAMPLE 15

Let S be the set of all students at Silicon U. Then the members of $\wp(S)$ are sets of students. Let A be the set of computer science majors, and let B be the set of business majors. Both A and B belong to $\wp(S)$. A new set of students can be defined that consists of everybody who is majoring in either computer science or business (or both); this set is called the *union* of A and B . Another new set can be defined that consists of everybody who is majoring in both computer science and business. This set (which might be empty) is called the *intersection* of A and B . ●

● **DEFINITION UNION AND INTERSECTION OF SETS**

Let $A, B \in \wp(S)$. The **union** of A and B , denoted by $A \cup B$, is $\{x \mid x \in A \text{ or } x \in B\}$. The **intersection** of A and B , denoted by $A \cap B$, is $\{x \mid x \in A \text{ and } x \in B\}$.

EXAMPLE 16

Let $A = \{1, 3, 5, 7, 9\}$ and $B = \{3, 5, 6, 10, 11\}$. Here we may consider A and B as members of $\wp(\mathbb{N})$. Then $A \cup B = \{1, 3, 5, 6, 7, 9, 10, 11\}$ and $A \cap B = \{3, 5\}$. Both $A \cup B$ and $A \cap B$ are members of $\wp(\mathbb{N})$. ●

PRACTICE 13

Let $A, B \in \wp(S)$ for any set S . Is it always the case that $A \cap B \subseteq A \cup B$? ■

We can use *Venn diagrams* (named for the nineteenth-century British mathematician John Venn) to visualize the binary operations of union and intersection. The shaded areas in Figures 4.1 and 4.2 illustrate the set that results from performing the binary operation on the two given sets.

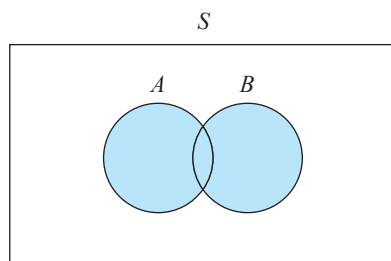


Figure 4.1 $A \cup B$

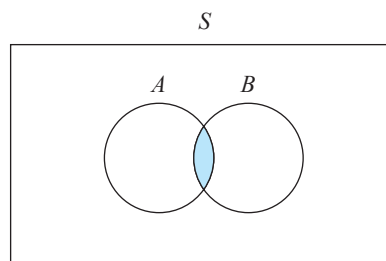


Figure 4.2 $A \cap B$

We will define one unary operation on $\wp(S)$.

● **DEFINITION** **COMPLEMENT OF A SET**

For a set $A \in \wp(S)$, the **complement** of A , A' , is $\{x \mid x \in S \text{ and } x \notin A\}$.

PRACTICE 14 Illustrate A' in a Venn diagram. ■

EXAMPLE 17

In Section 1.1 we discussed the use of logical connectives to formulate Web search queries. A query such as

“used cars” AND (Ford OR Buick) AND NOT trucks

is asking the search engine to return a set of pages (or more properly, a set of links to pages). If

U = set of used car pages

F = set of Ford pages

B = set of Buick pages

T = set of truck pages

then

$$U \cap (F \cup B) \cap T'$$

represents the set of Web pages that is the desired result of the query. ●

Another binary operation on sets A and B in $\wp(S)$ is **set difference**: $A - B = \{x \mid x \in A \text{ and } x \notin B\}$. This operation can be rewritten as $A - B = \{x \mid x \in A \text{ and } x \in B'\}$ and, finally, as $A - B = A \cap B'$.

PRACTICE 15 Illustrate $A - B$ in a Venn diagram. ■

Two sets A and B such that $A \cap B = \emptyset$ are said to be **disjoint**. Thus, $A - B$ and $B - A$, for example, are disjoint sets.

EXAMPLE 18

Let

$$A = \{x \mid x \text{ is an even nonnegative integer}\}$$

$$B = \{x \mid (\exists y)(y \in \mathbb{N} \text{ and } x = 2y + 1)\}$$

$$C = \{x \mid (\exists y)(y \in \mathbb{N} \text{ and } x = 4y)\}$$

be subsets of \mathbb{N} . Because B represents the set of nonnegative odd integers, A and B are disjoint sets. Also, every nonnegative integer is either even or odd, so $A \cup B = \mathbb{N}$. These two facts also tell us that $A' = B$. Every multiple of 4 is an even number, so C is a subset of A , from which it follows that $A \cup C = A$. C is in fact a proper subset of A , and $A - C = \{x \mid (\exists y)(y \in \mathbb{N} \text{ and } x = 4y + 2)\}$. ●

PRACTICE 16 | Let

$$A = \{1, 2, 3, 5, 10\}$$

$$B = \{2, 4, 7, 8, 9\}$$

$$C = \{5, 8, 10\}$$

be subsets of $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Find

- a. $A \cup B$ b. $A - C$ c. $B' \cap (A \cup C)$

We will define one final operation using elements of $\wp(S)$.

● **DEFINITION** **CARTESIAN PRODUCT**

Let A and B be subsets of S . The **Cartesian product (cross product)** of A and B , denoted by $A \times B$, is defined by

$$A \times B = \{(x, y) \mid x \in A \text{ and } y \in B\}$$

Thus, the Cartesian product of two sets A and B is the set of all ordered pairs whose first component comes from A and whose second component comes from B . The cross product is not a binary operation on $\wp(S)$. Although it acts on an ordered pair of members of $\wp(S)$ and gives a unique result, the resulting set is not, in general, a subset of S . The elements are not members of S but ordered pairs of members of S . So the resulting set is not a member of $\wp(S)$. The closure property for a binary operation fails to hold.

Because we will often be interested in the cross product of a set with itself, we will abbreviate $A \times A$ as A^2 ; in general, we use A^n to mean the set of all ordered n -tuples (x_1, x_2, \dots, x_n) of elements of A .

PRACTICE 17 | Let $A = \{1, 2\}$ and $B = \{3, 4\}$.

- a. Find $A \times B$. b. Find $B \times A$. c. Find A^2 . d. Find A^3 .

Set Identities

There are many set equalities involving the operations of union, intersection, difference, and complementation that are true for all subsets of a given set S . Because they are independent of the particular subsets used, these equalities are called set identities. Some basic set identities follow. The names and forms of these identities are very similar to the tautological equivalences of Section 1.1 (check back and compare). We will see in Chapter 8 that this similarity is not a coincidence.

BASIC SET IDENTITIES

1a. $A \cup B = B \cup A$	1b. $A \cap B = B \cap A$	(commutative properties)
2a. $(A \cup B) \cup C = A \cup (B \cup C)$	2b. $(A \cap B) \cap C = A \cap (B \cap C)$	(associative properties)
3a. $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	3b. $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	(distributive properties)
4a. $A \cup \emptyset = A$	4b. $A \cap S = A$	(identity properties)
5a. $A \cup A' = S$	5b. $A \cap A' = \emptyset$	(complement properties)

(Note that 2a allows us to write $A \cup B \cup C$ with no need for parentheses; 2b allows us to write $A \cap B \cap C$.)

EXAMPLE 19

Let's prove identity 3a. We might draw Venn diagrams for each side of the equation and see that they look the same. However, identity 3a is supposed to hold for all subsets A , B , and C , and whatever one picture we draw cannot be completely general. Thus, if we draw A and B disjoint, that's a special case, but if we draw A and B not disjoint, that doesn't take care of the case where A and B are disjoint. To do a proof by Venn diagrams requires a picture for each possible case, and the more sets involved (A , B , and C in this problem), the more cases there are. To avoid drawing a picture for each case, let's prove set equality by proving set inclusion in each direction. Thus, we want to prove

$$A \cup (B \cap C) \subseteq (A \cup B) \cap (A \cup C)$$

and also

$$(A \cup B) \cap (A \cup C) \subseteq A \cup (B \cap C)$$

To show that $A \cup (B \cap C) \subseteq (A \cup B) \cap (A \cup C)$, we let x be an arbitrary member of $A \cup (B \cap C)$. Then we can proceed as follows:

$$\begin{aligned} x \in A \cup (B \cap C) &\rightarrow x \in A \text{ or } x \in (B \cap C) \\ &\rightarrow x \in A \text{ or } (x \in B \text{ and } x \in C) \\ &\rightarrow (x \in A \text{ or } x \in B) \text{ and } (x \in A \text{ or } x \in C) \\ &\rightarrow x \in (A \cup B) \text{ and } x \in (A \cup C) \\ &\rightarrow x \in (A \cup B) \cap (A \cup C) \end{aligned}$$

To show that $(A \cup B) \cap (A \cup C) \subseteq A \cup (B \cap C)$, we reverse the above argument. ●

Once we have proved the set identities in this list, we can use them to prove other set identities. Just as the tautological equivalences of propositional logic represent recipes or patterns for transforming wffs, the set identities represent patterns for transforming set expressions. And, as with tautologies, the set identity can be applied only when the set expression exactly matches the pattern.

EXAMPLE 20

We can use the basic set identities to prove

$$[A \cup (B \cap C)] \cap ([A' \cup (B \cap C)] \cap (B \cap C)') = \emptyset$$

for A , B , and C any subsets of S . In the following proof, the number to the right is that of the basic set identity used to validate each step. The first step uses identity 2b because the expression

$$[A \cup (B \cap C)] \cap ([A' \cup (B \cap C)] \cap (B \cap C)')$$

matches the right side of 2b, $A \cap (B \cap C)$ where A is $[A \cup (B \cap C)]$, B is $[A' \cup (B \cap C)]$, and C is $(B \cap C)'$

$$\begin{aligned} & [A \cup (B \cap C)] \cap ([A' \cup (B \cap C)] \cap (B \cap C)') \\ &= ([A \cup (B \cap C)] \cap [A' \cup (B \cap C)]) \cap (B \cap C)' && (2b) \\ &= ((B \cap C) \cup A) \cap ((B \cap C) \cup A') \cap (B \cap C)' && (1a \text{ twice}) \\ &= [(B \cap C) \cup (A \cap A')] \cap (B \cap C)' && (3a) \\ &= [(B \cap C) \cup \emptyset] \cap (B \cap C)' && (5b) \\ &= (B \cap C) \cap (B \cap C)' && (4a) \\ &= \emptyset && (5b) \end{aligned}$$

REMINDER

You must match the pattern of a set identity in order to use it. In the set identities, A , B , and C can represent any sets.

The **dual** for each set identity in our list also appears in the list. The dual is obtained by interchanging \cup and \cap and interchanging S and \emptyset . The dual of the identity in Example 20 is

$$[A \cap (B \cup C)] \cup ([A' \cap (B \cup C)] \cup (B \cup C)') = S$$

which we could prove true by replacing each basic set identity used in the proof of Example 20 with its dual. Because this method always works, any time we have proved a set identity by using the basic identities, we have also proved its dual.

PRACTICE 19

- a. Using the basic set identities, establish the set identity

$$[C \cap (A \cup B)] \cup [(A \cup B) \cap C'] = A \cup B$$

(A , B , and C are any subsets of S .)

- b. State the dual identity that you now know is true.

Table 4.1 summarizes the approaches to proving set identities.

TABLE 4.1	
Method	Comment
Draw a Venn diagram	Not a good plan because no one diagram fits all cases and it will not prove the general identity.
Establish set inclusion in each direction	Take an arbitrary member of one side and show it belongs to the other side, and conversely.
Use already proved identities	Be sure to match the pattern of the identity you want to use.

Countable and Uncountable Sets

In a finite set S , we can always designate one element as the first member, s_1 , another element as the second member, s_2 , and so forth. If there are k elements in the set, then these can be listed in the order we have selected:

$$s_1, s_2, \dots, s_k$$

This list represents the entire set S . The number of elements in a finite set is the **cardinality** of the set, so this would be a set of cardinality k , denoted by $|S| = k$.

If the set is infinite, we may still be able to select a first element s_1 , a second element s_2 , and so forth, so that the list

$$s_1, s_2, s_3, \dots$$

represents all elements of the set. Every element of the set will eventually appear in this list. Such an infinite set is said to be **denumerable**. Both finite and denumerable sets are **countable** sets because we can count, or enumerate, all of their elements. Being countable does not mean that we can state the total number of elements in the set; rather, it means that we can say, “Here is a first one,” “Here is a second one,” and so on, through the set. There are, however, infinite sets that are **uncountable**. In an uncountable set, the set is so big that there is no way to count out the elements and get the whole set in the process. Before we prove that uncountable sets exist, let’s look at some denumerable (countably infinite) sets.

EXAMPLE 21

The set \mathbb{N} is denumerable.

To prove denumerability, we need only exhibit a counting scheme. For the set \mathbb{N} of nonnegative integers, it is clear that

$$0, 1, 2, 3, \dots$$

is an enumeration that will eventually include every member of the set. ●

PRACTICE 20

Prove that the set of even positive integers is denumerable. ■

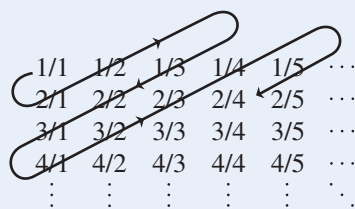
EXAMPLE 22

The set \mathbb{Q}^+ of positive rational numbers is denumerable.

We assume that each positive rational number is written as a fraction of positive integers. We can write all such fractions having the numerator 1 in one row, all those having the numerator 2 in a second row, and so on:

$$\begin{array}{cccccc} 1/1 & 1/2 & 1/3 & 1/4 & 1/5 & \cdots \\ 2/1 & 2/2 & 2/3 & 2/4 & 2/5 & \cdots \\ 3/1 & 3/2 & 3/3 & 3/4 & 3/5 & \cdots \\ 4/1 & 4/2 & 4/3 & 4/4 & 4/5 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

To show that the set of all fractions in this array is denumerable, we will thread an arrow through the entire array, beginning with $1/1$; following the arrow gives an enumeration of the set. Thus the fraction $1/3$ is the fourth member in this enumeration:



Therefore the set represented by the array is denumerable. Note that our path through the array must “spread out” from one corner. If we begin to follow just the first row or just the first column, for example, we will never finish it to get on to other rows (or columns).

To obtain an enumeration of \mathbb{Q}^+ , we use the enumeration of the set shown but eliminate any fractions not in lowest terms. This avoids the problem of listing both $1/2$ and $2/4$, for example, which represent the same positive rational. The enumeration of \mathbb{Q}^+ thus begins with

$$1/1, 2/1, 1/2, 1/3, 3/1, 4/1, \dots$$

For example, we have eliminated $2/2$, which reduces to $1/1$.

PRACTICE 21

What is the 11th fraction in the above enumeration? What is the 11th positive rational? ■

Now let’s show that there is an uncountable (not countable) infinite set. The proof technique that seems appropriate to prove that set A does *not* have property B is to assume that A does have property B and look for a contradiction. The proof in Example 23 is a very famous proof by contradiction known as **Cantor’s diagonalization method**, after Georg Cantor, the nineteenth-century German mathematician known as the “father of set theory.”

EXAMPLE 23

We will show that the set of all the real numbers between 0 and 1 is uncountable.

We will write such numbers in decimal form; thus any member of the set can be written as

$$0.d_1d_2d_3 \dots$$

Now let us assume that our set is countable. Therefore some enumeration of the set exists. A number such as $0.24999999 \dots$ can be written in alternative form as $0.2500000 \dots$ (see Exercise 102 for an explanation of why these are alternative representations of the same number). To avoid writing the same element twice in our enumeration, we will choose (arbitrarily) to always use the former representation and not the latter. We can depict an enumeration of the set as follows, where d_{ij} is the j th decimal digit in the i th number in the enumeration:

$$\begin{array}{r} 0.d_{11}d_{12}d_{13} \dots \\ 0.d_{21}d_{22}d_{23} \dots \\ 0.d_{31}d_{32}d_{33} \dots \\ \vdots \quad \vdots \quad \vdots \quad \ddots \end{array}$$

We now construct a real number $p = 0.p_1p_2p_3 \dots$ as follows: p_i is always chosen to be 5 if $d_{ii} \neq 5$ and 6 if $d_{ii} = 5$. Thus p is a real number between 0 and 1. For instance, if the enumeration begins with

$$\begin{array}{r} 0.342134 \dots \\ 0.257001 \dots \\ 0.546122 \dots \\ 0.716525 \dots \end{array}$$

then $d_{11} = 3$, $d_{22} = 5$, $d_{33} = 6$, and $d_{44} = 5$, so $p_1 = 5$, $p_2 = 6$, $p_3 = 5$, and $p_4 = 6$. Thus p begins with $0.5656 \dots$.

If we compare p with the enumeration of the set, p differs from the first number at the first decimal digit, from the second number at the second decimal digit, from the third number at the third decimal digit, and so on.

$$\begin{array}{r} 0.\textcircled{3}42134\dots \\ 0.2\textcircled{5}7001\dots \\ 0.54\textcircled{6}122\dots \\ 0.716\textcircled{5}25\dots \\ \vdots \end{array}$$

Therefore p does not agree with any of the representations in the enumeration. Furthermore, because p contains no 0s to the right of the decimal, it is not the alternative representation of any of the numbers in the enumeration. Therefore p is a real number between 0 and 1 different from any other number in the enumeration, yet the enumeration was supposed to include all members of the set. Here, then, is the contradiction, and the set of all real numbers between 0 and 1 is indeed uncountable. (You can see why this proof is called a “diagonalization method.”) ●

Although it is interesting and perhaps surprising to learn that there are uncountable sets, we are usually concerned with countable sets. A computer, of course, can manage only finite sets. In the rest of this chapter, we too, limit our attention to finite sets and various ways to count their elements.

SECTION 4.1 REVIEW

TECHNIQUES

- Describe sets by a list of elements and by a characterizing property.
- Prove that one set is a subset of another.
- Find the power set of a set.
- Check that the required properties for a binary or unary operation are satisfied.
- Form new sets by taking the union, intersection, complement, and cross product of sets.
- Prove set identities by showing set inclusion in each direction or using the basic set identities.
- Demonstrate the denumerability of certain sets.
- Use the Cantor diagonalization method to prove that certain sets are uncountable.

MAIN IDEAS

- Sets are unordered collections of objects that can be related (equal sets, subsets, etc.) or combined (union, intersection, etc.).
- Certain standard sets have their own notation.
- The power set of a set with n elements has 2^n elements.
- Basic set identities exist (in dual pairs) and can be used to prove other set identities; once an identity is proved in this manner, its dual is also true.
- Countable sets can be enumerated, and uncountable sets exist.

EXERCISES 4.1

- Let $S = \{2, 5, 17, 27\}$. Which of the following expressions are true?
 - $5 \in S$
 - $2 + 5 \in S$
 - $\emptyset \in S$
 - $S \in S$
- Let $B = \{x \mid x \in \mathbb{Q} \text{ and } -1 < x < 2\}$. Which of the following expressions are true?
 - $0 \in B$
 - $-1 \in B$
 - $-0.84 \in B$
 - $\sqrt{2} \in B$
- How many different sets are described here? What are they?

$\{2, 3, 4\}$	\emptyset
$\{x \mid x \text{ is the first letter of cat, bat, or apple}\}$	$\{x \mid x \text{ is the first letter of cat, bat, and apple}\}$
$\{x \mid x \in \mathbb{N} \text{ and } 2 \leq x \leq 4\}$	$\{2, a, 3, b, 4, c\}$
$\{a, b, c\}$	$\{3, 4, 2\}$
- How many different sets are described here? What are they?
 - $\{x \mid x = F(n) \wedge n \in \{5, 6, 7\}\}$ [$F(n)$ is a Fibonacci number]
 - $\{x \mid x \mid 24\}$ [x divides 24]
 - $\{1, 2, 3, 4\}$
 - $\{5, 8, 13\}$
 - $\{x \mid x \in \mathbb{N} \wedge 0 < x \leq 4\}$
 - $\{x \mid x \in \varphi(5)\}$ [$\varphi(n)$ is the Euler phi function]
 - $\{12, 2, 6, 24, 8, 3, 1, 4\}$
 - $\{x \mid x \text{ is a digit in the decimal equivalent of the Roman numeral MCCXXXIV}\}$

5. Describe each of the following sets by listing its elements:
- $\{x \mid x \in \mathbb{N} \text{ and } x^2 < 25\}$
 - $\{x \mid x \in \mathbb{N} \text{ and } x \text{ is even and } 2 < x < 11\}$
 - $\{x \mid x \text{ is one of the first three U.S. presidents}\}$
 - $\{x \mid x \in \mathbb{R} \text{ and } x^2 = -1\}$
 - $\{x \mid x \text{ is one of the New England states}\}$
 - $\{x \mid x \in \mathbb{Z} \text{ and } |x| < 4\}$ ($|x|$ denotes the absolute value function)
6. Describe each of the following sets by listing its elements:
- $\{x \mid x \in \mathbb{N} \text{ and } x^2 - 5x + 6 = 0\}$
 - $\{x \mid x \in \mathbb{R} \text{ and } x^2 = 7\}$
 - $\{x \mid x \in \mathbb{N} \text{ and } x^2 - 2x - 8 = 0\}$
7. Describe each of the following sets by giving a characterizing property:
- $\{1, 2, 3, 4, 5\}$
 - $\{1, 3, 5, 7, 9, 11, \dots\}$
 - $\{\text{Melchior, Gaspar, Balthazar}\}$
 - $\{0, 1, 10, 11, 100, 101, 110, 111, 1000, \dots\}$
8. Describe each of the following sets:
- $\{x \mid x \in \mathbb{N} \text{ and } (\exists q)(q \in \{2, 3\} \text{ and } x = 2q)\}$
 - $\{x \mid x \in \mathbb{N} \text{ and } (\exists y)(\exists z)(y \in \{0, 1\} \text{ and } z \in \{3, 4\} \text{ and } y < x < z)\}$
 - $\{x \mid x \in \mathbb{N} \text{ and } (\forall y)(y \text{ even} \rightarrow x \neq y)\}$
9. Given the description of a set A as $A = \{2, 4, 8, \dots\}$, do you think $16 \in A$?
10. What is the cardinality of each of the following sets?
- $S = \{a, \{a, \{a\}\}\}$
 - $S = \{\{a\}, \{\{a\}\}\}$
 - $S = \{\emptyset\}$
 - $S = \{a, \{\emptyset\}, \emptyset\}$
 - $S = \{\emptyset, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset, \{\emptyset\}\}\}\}$

11. Let

$$A = \{2, 5, 7\}$$

$$B = \{1, 2, 4, 7, 8\}$$

$$C = \{7, 8\}$$

Which of the following statements are true?

- $5 \subseteq A$
- $C \subseteq B$
- $\emptyset \in A$
- $7 \in B$
- $\{2, 5\} \subseteq A$
- $\emptyset \subseteq C$

12. Let

$$A = \{x \mid x \in \mathbb{N} \text{ and } 1 < x < 50\}$$

$$B = \{x \mid x \in \mathbb{R} \text{ and } 1 < x < 50\}$$

$$C = \{x \mid x \in \mathbb{Z} \text{ and } |x| \geq 25\}$$

Which of the following statements are true?

- a. $A \subseteq B$ e. $\sqrt{3} \in B$
 b. $17 \in A$ f. $\{0, 1, 2\} \subseteq A$
 c. $A \subseteq C$ g. $\emptyset \in B$
 d. $-40 \in C$ h. $\{x | x \in \mathbb{Z} \text{ and } x^2 > 625\} \subseteq C$

13. Let

$$\begin{aligned} R &= \{1, 3, \pi, 4.1, 9, 10\} & T &= \{1, 3, \pi\} \\ S &= \{\{1\}, 3, 9, 10\} & U &= \{\{1, 3, \pi\}, 1\} \end{aligned}$$

Which of the following statements are true? For those that are not, why not?

- a. $S \subseteq R$ e. $\{1\} \subseteq T$
 b. $1 \in R$ f. $\{1\} \subseteq S$
 c. $1 \in S$ g. $T \subset R$
 d. $1 \subseteq U$

14. Let

$$\begin{aligned} R &= \{1, 3, \pi, 4.1, 9, 10\} & T &= \{1, 3, \pi\} \\ S &= \{\{1\}, 3, 9, 10\} & U &= \{\{1, 3, \pi\}, 1\} \end{aligned}$$

Which of the following statements are true? For those that are not, why not?

- a. $\{1\} \in S$ e. $T \notin R$
 b. $\emptyset \subseteq S$ f. $T \subseteq R$
 c. $T \subseteq U$ g. $S \subseteq \{1, 3, 9, 10\}$
 d. $T \in U$

15. Let

$$A = \{a, \{a\}, \{\{a\}\}\} \quad B = \{a\} \quad C = \{\emptyset, \{a, \{a\}\}\}$$

Which of the following statements are true? For those that are not, where do they fail?

- a. $B \subseteq A$ f. $\{a, \{a\}\} \in A$
 b. $B \in A$ g. $\{a, \{a\}\} \subseteq A$
 c. $C \subseteq A$ h. $B \subseteq C$
 d. $\emptyset \subseteq C$ i. $\{\{a\}\} \subseteq A$
 e. $\emptyset \in C$

16. Let

$$A = \{\emptyset, \{\emptyset, \{\emptyset\}\}\} \quad B = \emptyset \quad C = \{\emptyset\} \quad D = \{\emptyset, \{\emptyset\}\}$$

Which of the following statements are true? For those that are not, where do they fail?

- a. $C \subseteq A$ f. $C = B$
 b. $C \in A$ g. $C \subseteq D$
 c. $B \in A$ h. $C \in D$
 d. $B \subseteq A$ i. $D \subseteq A$
 e. $B \in C$

17. Let

$$A = \{(x, y) \mid (x, y) \text{ lies within 3 units of the point } (1, 4)\}$$

and

$$B = \{(x, y) \mid (x - 1)^2 + (y - 4)^2 \leq 25\}$$

Prove that $A \subset B$.

18. Let

$$A = \{x \mid x \in \mathbb{R} \text{ and } x^2 - 4x + 3 < 0\}$$

and

$$B = \{x \mid x \in \mathbb{R} \text{ and } 0 < x < 6\}$$

Prove that $A \subset B$.

19. Program QUAD finds and prints solutions to quadratic equations of the form $ax^2 + bx + c = 0$. Program EVEN lists all the even integers from $-2n$ to $2n$. Let Q denote the set of values output by QUAD and E denote the set of values output by EVEN.
- Show that for $a = 1$, $b = -2$, $c = -24$, and $n = 50$, $Q \subseteq E$.
 - Show that for the same values of a , b , and c , but a value for n of 2, $Q \not\subseteq E$.
20. Let $A = \{x \mid \cos(x/2) = 0\}$ and $B = \{x \mid \sin x = 0\}$. Prove that $A \subseteq B$.
21. Which of the following statements are true for all sets A , B , and C ?
- If $A \subseteq B$ and $B \subseteq A$, then $A = B$.
 - $\{\emptyset\} = \emptyset$
 - $\{\emptyset\} = \{0\}$
 - $\emptyset \in \{\emptyset\}$
 - $\emptyset \subseteq A$
22. Which of the following statements are true for all sets A , B , and C ?
- $\emptyset \in A$
 - $\{\emptyset\} = \{\{\emptyset\}\}$
 - If $A \subset B$ and $B \subseteq C$, then $A \subset C$.
 - If $A \neq B$ and $B \neq C$, then $A \neq C$.
 - If $A \in B$ and $B \notin C$, then $A \notin C$.
23. Prove that if $A \subseteq B$ and $B \subseteq C$, then $A \subseteq C$.
24. Prove that if $A' \subseteq B'$, then $B \subseteq A$.
25. Prove that for any integer $n \geq 2$, a set with n elements has $n(n - 1)/2$ subsets that contain exactly two elements.
26. Prove that for any integer $n \geq 3$, a set with n elements has $n(n - 1)(n - 2)/6$ subsets that contain exactly three elements. (*Hint:* Use Exercise 25.)
27. Find $\wp(S)$ for $S = \{a\}$.
28. Find $\wp(S)$ for $S = \{a, b\}$.
29. Find $\wp(S)$ for $S = \{1, 2, 3, 4\}$. How many elements do you expect this set to have?
30. Find $\wp(S)$ for $S = \{\emptyset\}$.

31. Find $\wp(S)$ for $S = \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}$.
32. Find $\wp(\wp(S))$ for $S = \{a, b\}$.
33. What can be said about A if $\wp(A) = \{\emptyset, \{x\}, \{y\}, \{x, y\}\}$?
34. What can be said about A if $\wp(A) = \{\emptyset, \{a\}, \{\{a\}\}$?
35. Prove that if $\wp(A) = \wp(B)$, then $A = B$.
36. Prove that if $A \subseteq B$, then $\wp(A) \subseteq \wp(B)$.
37. Solve for x and y .
- a. $(y, x + 2) = (5, 3)$ b. $(2x, y) = (16, 7)$ c. $(2x - y, x + y) = (-2, 5)$
38. a. Recall that ordered pairs must have the property that $(x, y) = (u, v)$ if and only if $x = u$ and $y = v$. Prove that $\{\{x\}, \{x, y\}\} = \{\{u\}, \{u, v\}\}$ if and only if $x = u$ and $y = v$. Therefore, although we know that $(x, y) \neq \{x, y\}$, we can define the ordered pair (x, y) as the set $\{\{x\}, \{x, y\}\}$.
- b. Show by an example that we cannot define the ordered triple (x, y, z) as the set $\{\{x\}, \{x, y\}, \{x, y, z\}\}$.
39. Which of the following candidates are binary or unary operations on the given sets? For those that are not, where do they fail?
- a. $x \circ y = x + 1$; $S = \mathbb{N}$
- b. $x \circ y = x + y - 1$; $S = \mathbb{N}$
- c. $x \circ y = \begin{cases} x - 1 & \text{if } x \text{ is odd} \\ x & \text{if } x \text{ is even} \end{cases} \quad S = \mathbb{Z}$
- d. $x\# = \ln x$; $S = \mathbb{R}$
40. Which of the following candidates are binary or unary operations on the given sets? For those that are not, where do they fail?
- a. $x\# = x^2$; $S = \mathbb{Z}$
- b.
- | | | | | |
|---|---|---|---|-------------------|
| o | 1 | 2 | 3 | $S = \{1, 2, 3\}$ |
| 1 | 1 | 2 | 3 | |
| 2 | 2 | 3 | 4 | |
| 3 | 3 | 4 | 5 | |
- c. $x \circ y =$ that fraction, x or y , with the smaller denominator; $S =$ set of all fractions.
- d. $x \circ y =$ that person, x or y , whose name appears first in an alphabetical sort; $S =$ set of 10 people with different names.
41. Which of the following candidates are binary or unary operations on the given sets? For those that are not, where do they fail?
- a. $x \circ y = \begin{cases} 1/x & \text{if } x \text{ is positive} \\ 1/(-x) & \text{if } x \text{ is negative} \end{cases} \quad S = \mathbb{R}$
- b. $x \circ y = xy$ (concatenation); $S =$ set of all finite-length strings of symbols from the set $\{p, q, r\}$
- c. $x\# = \lfloor x \rfloor$ where $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x ; $S = \mathbb{R}$
- d. $x \circ y = \min(x, y)$; $S = \mathbb{N}$
42. Which of the following candidates are binary or unary operations on the given sets? For those that are not, where do they fail?
- a. $x \circ y =$ greatest common multiple of x and y ; $S = \mathbb{N}$
- b. $x \circ y = x + y$; $S =$ the set of Fibonacci numbers

- c. $x\# =$ the string that is the reverse of x ; $S =$ set of all finite-length strings of symbols from the set $\{p, q, r\}$
 d. $x \circ y = x + y$; $S = \mathbb{R} - \mathbb{Q}$
43. How many different unary operations can be defined on a set with n elements? (*Hint*: Think about filling in a table.)
44. How many different binary operations can be defined on a set with n elements? (*Hint*: Think about filling in a table.)
45. We have written binary operations in *infix* notation, where the operation symbol appears between the two operands, as in $A + B$. Evaluation of a complicated arithmetic expression is more efficient when the operations are written in *postfix* notation, where the operation symbol appears after the two operands, as in $AB+$. Many compilers change expressions in a computer program from infix to postfix form. One way to produce an equivalent postfix expression from an infix expression is to write the infix expression with a full set of parentheses, move each operator to replace its corresponding right parenthesis, and then eliminate all left parentheses. (Parentheses are not required in postfix notation.) Thus,

$$A * B + C$$

becomes, when fully parenthesized,

$$((A * B) + C)$$

and the postfix notation is $AB * C+$. Rewrite each of the following expressions in postfix notation:

- a. $(A + B) * (C - D)$
 b. $A ** B - C * D$ (**denotes exponentiation)
 c. $A * C + B / (C + D * B)$
46. Evaluate the following postfix expressions (see Exercise 45):
 a. $2\ 4\ * \ 5\ +$ b. $5\ 1\ + \ 2 \ / \ 1\ -$ c. $3\ 4\ + \ 5\ 1\ - \ *$
47. Let

$$A = \{p, q, r, s\}$$

$$B = \{r, t, v\}$$

$$C = \{p, s, t, u\}$$

be subsets of $S = \{p, q, r, s, t, u, v, w\}$. Find

- a. $B \cap C$ b. $A \cup C$ c. C' d. $A \cap B \cap C$
48. Let

$$A = \{p, q, r, s\}$$

$$B = \{r, t, v\}$$

$$C = \{p, s, t, u\}$$

be subsets of $S = \{p, q, r, s, t, u, v, w\}$. Find

- a. $B - C$ b. $(A \cup B)'$ c. $A \times B$ d. $(A \cup B) \cap C'$

49. Let

$$A = \{2, 4, 5, 6, 8\}$$

$$B = \{1, 4, 5, 9\}$$

$$C = \{x \mid x \in \mathbb{Z} \text{ and } 2 \leq x < 5\}$$

be subsets of $S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Find

a. $A \cup B$ e. $A - B$

b. $A \cap B$ f. A'

c. $A \cap C$ g. $A \cap A'$

d. $B \cup C$

50. Let

$$A = \{2, 4, 5, 6, 8\}$$

$$B = \{1, 4, 5, 9\}$$

$$C = \{x \mid x \in \mathbb{Z} \text{ and } 2 \leq x < 5\}$$

be subsets of $S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Find

a. $(A \cap B)'$ e. $(C' \cup B)'$

b. $C - B$ f. $B \times C$

c. $(C \cap B) \cup A'$ g. $(B - A) \cup C$

d. $(B - A)' \cap (A - B)$

51. Let

$$A = \{a, \{a\}, \{\{a\}\}\}$$

$$B = \{\emptyset, \{a\}, \{a, \{a\}\}\}$$

$$C = \{a\}$$

be subsets of $S = \{\emptyset, a, \{a\}, \{\{a\}\}, \{a, \{a\}\}\}$. Find

a. $A \cap C$ d. $\emptyset \cap B$ f. $A' \cap B$

b. $B \cap C'$ e. $(B \cup C) \cap A$ g. $\{\emptyset\} \cap B$

c. $A \cup B$

52. Let

$$A = \{x \mid x \text{ is the name of a former president of the United States}\}$$

$$B = \{\text{Adams, Hamilton, Jefferson, Grant}\}$$

$$C = \{x \mid x \text{ is the name of a state}\}$$

Find

a. $A \cap B$ b. $A \cap C$ c. $B \cap C$

53. Let $S = A \times B$ where $A = \{2, 3, 4\}$ and $B = \{3, 5\}$. Which of the following statements are true?

- a. $A \subseteq S$ d. $(5, 4) \in S$
 b. $3 \in S$ e. $\emptyset \subseteq S$
 c. $(3, 3) \in S$ f. $\{(2, 5)\} \subseteq S$

54. Let

$$A = \{x \mid x \text{ is a word that appears before } \textit{dog} \text{ in an English language dictionary}\}$$

$$B = \{x \mid x \text{ is a word that appears after } \textit{canary} \text{ in an English language dictionary}\}$$

$$C = \{x \mid x \text{ is a word of more than four letters}\}$$

Which of the following statements are true?

- a. $B \subseteq C$
 b. $A \cup B = \{x \mid x \text{ is a word in an English language dictionary}\}$
 c. $\textit{cat} \in B \cap C'$
 d. $\textit{bamboo} \in A - B$

55. Consider the following subsets of \mathbb{Z} :

$$A = \{x \mid (\exists y)(y \in \mathbb{Z} \text{ and } y \geq 4 \text{ and } x = 3y)\}$$

$$B = \{x \mid (\exists y)(y \in \mathbb{Z} \text{ and } x = 2y)\}$$

$$C = \{x \mid x \in \mathbb{Z} \text{ and } |x| \leq 10\}$$

Using set operations, describe each of the following sets in terms of A , B , and C .

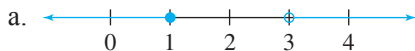
- a. set of all odd integers
 b. $\{-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10\}$
 c. $\{x \mid (\exists y)(y \in \mathbb{Z} \text{ and } y \geq 2 \text{ and } x = 6y)\}$
 d. $\{-9, -7, -5, -3, -1, 1, 3, 5, 7, 9\}$
 e. $\{x \mid (\exists y)(y \in \mathbb{Z} \text{ and } y \geq 5 \text{ and } x = 2y + 1)\} \cup \{x \mid (\exists y)(y \in \mathbb{Z} \text{ and } y \leq -5 \text{ and } x = 2y - 1)\}$

56. Let

$$A = \{x \mid x \in \mathbb{R} \text{ and } 1 < x \leq 3\}$$

$$B = \{x \mid x \in \mathbb{R} \text{ and } 2 \leq x \leq 5\}$$

Using set operations, describe each of the sets shown in terms of A and B .



57. Consider the following subsets of the set of all students:

A = set of all computer science majors

B = set of all physics majors

C = set of all science majors

D = set of all female students

Using set operations, describe each of the following sets in terms of A , B , C , and D :

- set of all students not majoring in science
- set of all female physics majors
- set of all students majoring in both computer science and physics
- set of all male computer science majors

58. Consider the following subsets of the set of all students:

A = set of all computer science majors

B = set of all physics majors

C = set of all science majors

D = set of all female students

Using set operations, describe each of the following sets in terms of A , B , C , and D :

- set of all male students who are not physics majors
 - set of all science majors who are not computer science majors
 - set of all students who are female or science majors
 - set of all students who are science majors but are neither computer science nor physics majors
59. Write the set expression for the desired results of the Web search query for sites pertaining to dogs that are not retrievers (Exercise 31, Section 1.1). Assume that D = set of dog pages, R = set of retriever pages.
60. Write the set expression for the desired results of the Web search query for sites pertaining to oil paintings by Van Gogh or Rembrandt but not Vermeer (Exercise 32, Section 1.1). Assume that O = set of oil painting pages, G = set of Van Gogh pages, R = set of Rembrandt pages, V = set of Vermeer pages.
61. Write the set expression for the desired results of the Web search query for sites pertaining to novels or plays about AIDS (Exercise 33, Section 1.1). Assume that N = set of novel pages, P = set of play pages, A = set of AIDS pages.
62. Write the set expression for the desired results of the Web search query for sites pertaining to coastal wetlands in Louisiana but not in Alabama (Exercise 34, Section 1.1). Assume that C = set of coastal wetlands pages, L = set of Louisiana pages, A = set of Alabama pages.
63. Which of the following statements are true for all sets A , B , and C ?
- | | |
|-------------------------------|---------------------------------------|
| a. $A \cup A = A$ | d. $(A')' = A$ |
| b. $B \cap B = B$ | e. $A - B = (B - A)'$ |
| c. $(A \cap B)' = A' \cap B'$ | f. $(A - B) \cap (B - A) = \emptyset$ |
64. Which of the following statements are true for all sets A , B , and C ?
- | | |
|---|---|
| a. If $A \cap B = \emptyset$, then $A \subset B$. | d. $\emptyset \cap \{\emptyset\} = \emptyset$ |
| b. $B \times A = A \times B$ | e. $(A - B) \cup (B - C) = A - C$ |
| c. $\emptyset \times A = \emptyset$ | f. $(A - C) \cap (A - B) = A - (B \cup C)$ |

65. Which of the following statements are true for all sets A , B , and C ?
- $A \cup (B \times C) = (A \cup B) \times (A \cup C)$
 - $A \times (B \cap C) = (A \times B) \cap (A \times C)$
 - $A \times \emptyset = \emptyset$
66. Which of the following statements are true for all sets A , B , and C ?
- $\wp(A) \times \wp(A) = \wp(A^2)$
 - $A \times (B \times C) = (A \times B) \times C$
 - $\wp(A \times B) = \wp(A) \times \wp(B)$
67. For each of the following statements, find general conditions on sets A and B to make the statement true:
- $A \cup B = A$
 - $A \cap B = A$
 - $A \cup \emptyset = \emptyset$
 - $B - A = \emptyset$
 - $A \cup B \subseteq A \cap B$
 - $A \times B = B \times A$
68. For any finite set S , $|S|$ denotes the number of elements in S . If $|A| = 3$ and $|B| = 4$, find
- $|A \times B|$
 - $|A^2|$
 - $|B^2|$
 - the maximum possible value for $|A \cap B|$
 - the minimum possible value for $|A \cup B|$
69. Prove that $(A \cap B) \subseteq A$ where A and B are arbitrary sets.
70. Prove that $A \subseteq (A \cup B)$ where A and B are arbitrary sets.
71. Prove that $\wp(A) \cap \wp(B) = \wp(A \cap B)$ where A and B are arbitrary sets.
72. Prove that $\wp(A) \cup \wp(B) \subseteq \wp(A \cup B)$ where A and B are arbitrary sets.
73. Prove that if $A \cup B = A - B$, then $B = \emptyset$. (*Hint*: Do a proof by contradiction.)
74. Prove that if $(A - B) \cup (B - A) = A \cup B$, then $A \cap B = \emptyset$. (*Hint*: Do a proof by contradiction.)
75. Prove that if $C \subseteq B - A$, then $A \cap C = \emptyset$.
76. Prove that if $(A - B) \cup B = A$, then $B \subseteq A$.
77. Prove that $A \subseteq B$ if and only if $A \cap B' = \emptyset$.
78. Prove that $(A \cap B) \cup C = A \cap (B \cup C)$ if and only if $C \subseteq A$.

Exercises 79 and 80 refer to a binary operation on sets called the *symmetric difference*, which is defined by $A \oplus B = (A - B) \cup (B - A)$.

79. a. Draw a Venn diagram to illustrate $A \oplus B$.
 b. For $A = \{3, 5, 7, 9\}$ and $B = \{2, 3, 4, 5, 6\}$, what is $A \oplus B$?
 c. Prove that $A \oplus B = (A \cup B) - (A \cap B)$ for arbitrary sets A and B .
80. a. For an arbitrary set A , what is $A \oplus A$? What is $\emptyset \oplus A$?
 b. Prove that $A \oplus B = B \oplus A$ for arbitrary sets A and B .
 c. For any sets A , B , and C , prove that $(A \oplus B) \oplus C = A \oplus (B \oplus C)$.
81. Verify the basic set identities on page 234 by showing set inclusion in each direction. (We have already done 3a and 4a.)
82. A and B are subsets of a set S . Prove the following set identities by showing set inclusion in each direction.
- $(A \cup B)' = A' \cap B'$
 - $(A \cap B)' = A' \cup B'$
- } De Morgan's laws

- c. $A \cup (B \cap A) = A$
 d. $(A \cap B')' \cup B = A' \cup B$
 e. $(A \cap B) \cup (A \cap B') = A$
 f. $[A \cap (B \cup C)]' = A' \cup (B' \cap C')$
83. A , B , and C are subsets of a set S . Prove the following set identities using the basic set identities listed in this section. Give a reason for each step. State the dual of each of these identities.
- a. $(A \cup B) \cap (A \cup B') = A$
 b. $([(A \cap C) \cap B] \cup [(A \cap C) \cap B']) \cup (A \cap C)' = S$
 c. $(A \cup C) \cap [(A \cap B) \cup (C' \cap B)] = A \cap B$
84. A is a subset of a set S . Prove the following set identities:
- a. $A \cup A = A$ d. $A \cup S = S$
 b. $A \cap A = A$ e. $(A')' = A$
 c. $A \cap \emptyset = \emptyset$
85. A , B , and C are subsets of a set S . Prove the following set identities by using previously proved identities, including those in Exercises 82–84. Give a reason for each step.
- a. $A \cap (B \cup A') = B \cap A$
 b. $(A \cup B) - C = (A - C) \cup (B - C)$
 c. $(A - B) - C = (A - C) - B$
86. A , B , and C are subsets of a set S . Prove the following set identities by using previously proved identities, including those in Exercises 82–84. Give a reason for each step.
- a. $[(A' \cup B') \cap A']' = A$
 b. $(A - B) - C = (A - C) - (B - C)$
 c. $A - (A - B) = A \cap B$
 d. $(A \cup B) - (A \cap B) = (A - B) \cup (B - A)$
87. The operation of set union can be defined as an n -ary operation for any integer $n \geq 2$.
- a. Give a definition similar to that for the union of two sets for $A_1 \cup A_2 \cup \cdots \cup A_n$.
 b. Give a recursive definition for $A_1 \cup A_2 \cup \cdots \cup A_n$.
88. Using the recursive definition of set union from Exercise 87(b), prove the generalized associative property of set union, which is that for any n with $n \geq 3$ and any p with $1 \leq p \leq n - 1$,

$$(A_1 \cup A_2 \cup \cdots \cup A_p) \cup (A_{p+1} \cup A_{p+2} \cup \cdots \cup A_n) = A_1 \cup A_2 \cup \cdots \cup A_n$$

89. The operation of set intersection can be defined as an n -ary operation for any integer $n \geq 2$.
- a. Give a definition similar to that for the intersection of two sets for $A_1 \cap A_2 \cap \cdots \cap A_n$.
 b. Give a recursive definition for $A_1 \cap A_2 \cap \cdots \cap A_n$.
90. Using the recursive definition of set intersection from Exercise 89(b), prove the generalized associative property of set intersection, which is that for any n with $n \geq 3$ and any p with $1 \leq p \leq n - 1$,

$$(A_1 \cap A_2 \cap \cdots \cap A_p) \cap (A_{p+1} \cap A_{p+2} \cap \cdots \cap A_n) = A_1 \cap A_2 \cap \cdots \cap A_n$$

91. Prove that for subsets A_1, A_2, \dots, A_n and B of a set S , the following generalized distributive properties hold, where $n \geq 2$. (See Exercises 87 and 89.)
- $B \cup (A_1 \cap A_2 \cap \dots \cap A_n) = (B \cup A_1) \cap (B \cup A_2) \cap \dots \cap (B \cup A_n)$
 - $B \cap (A_1 \cup A_2 \cup \dots \cup A_n) = (B \cap A_1) \cup (B \cap A_2) \cup \dots \cup (B \cap A_n)$
92. Prove that for subsets A_1, A_2, \dots, A_n of a set S , the following generalized De Morgan's laws hold, where $n \geq 2$. (See Exercises 82, 87, and 89.)
- $(A_1 \cup A_2 \cup \dots \cup A_n)' = A_1' \cap A_2' \cap \dots \cap A_n'$
 - $(A_1 \cap A_2 \cap \dots \cap A_n)' = A_1' \cup A_2' \cup \dots \cup A_n'$
93. The operations of set union and set intersection can be extended to apply to an infinite family of sets. We may describe the family as the collection of all sets A_i , where i takes on any of the values of a fixed set I . Here, I is called the *index set* for the family. The union of the family, $\bigcup_{i \in I} A_i$, is defined by

$$\bigcup_{i \in I} A_i = \{x \mid x \text{ is a member of some } A_i\}$$

The intersection of the family, $\bigcap_{i \in I} A_i$, is defined by

$$\bigcap_{i \in I} A_i = \{x \mid x \text{ is a member of each } A_i\}.$$

- Let $I = \{1, 2, 3, \dots\}$, and for each $i \in I$, let A_i be the set of real numbers in the interval $(-1/i, 1/i)$. What is $\bigcup_{i \in I} A_i$? What is $\bigcap_{i \in I} A_i$?
 - Let $I = \{1, 2, 3, \dots\}$, and for each $i \in I$, let A_i be the set of real numbers in the interval $[-1/i, 1/i]$. What is $\bigcup_{i \in I} A_i$? What is $\bigcap_{i \in I} A_i$?
94. According to our use of the word "set," if C is a subset of the universal set S , then every element of S either does or does not belong to C . In other words, the probability of a member x of S being a member of C is either 1 (x is a member of C) or 0 (x is not a member of C). C is a *fuzzy set* if every $x \in S$ has a probability p , $0 \leq p \leq 1$, of being a member of C . The probability p associated with x is an estimate of the likelihood that x may belong to C when the actual composition of C is unknown. Set operations can be done on fuzzy sets as follows: If element x has probability p_1 of membership in C and probability p_2 of membership in D , then the probability of x being a member of $C \cup D$, $C \cap D$, and C' is, respectively, $\max(p_1, p_2)$, $\min(p_1, p_2)$, and $1 - p_1$. (If we consider the statements $x \in C$ and $x \in D$ as propositional wffs A and B , respectively, with certain probabilistic truth values, then the probability of $x \in C \cup D$ is the probability that $A \vee B$ is true. The rules for fuzzy set operations then parallel the rules for *fuzzy logic*, discussed in Exercise 54, Section 1.1.)
- Let S be a set of possible disease-causing agents, $S = \{\text{genetics, virus, nutrition, bacteria, environment}\}$. The fuzzy sets AIDS and ALZHEIMERS are defined as AIDS = {genetics, 0.2; virus, 0.8; nutrition, 0.1; bacteria, 0.4; environment, 0.3} and ALZHEIMERS = {genetics, 0.7; virus, 0.4; nutrition, 0.3; bacteria, 0.3; environment, 0.4}.
- Find the fuzzy set AIDS \cup ALZHEIMERS.
 - Find the fuzzy set AIDS \cap ALZHEIMERS.
 - Find the fuzzy set (AIDS)'.

Exercises 95 and 96 complete the proof, begun in Section 2.2, that the second principle of induction, the first principle of induction, and the principle of well-ordering are all equivalent.

95. The *principle of well-ordering* says that every nonempty set of positive integers has a smallest member. Prove that the first principle of mathematical induction, that is,

$$\left. \begin{array}{l} 1. P(1) \text{ is true} \\ 2. (\forall k)[P(k) \text{ true} \rightarrow P(k+1) \text{ true}] \end{array} \right\} \rightarrow P(n) \text{ true for all positive integers } n$$

implies the principle of well-ordering. (*Hint*: Assume that the first principle of mathematical induction is valid, and use proof by contradiction to show that the principle of well-ordering is valid. Let T be a nonempty subset of the positive integers that has no smallest member. Let $P(n)$ be the property that every member of T is greater than n .)

96. Prove that the principle of well-ordering (see Exercise 95) implies the second principle of mathematical induction. *Hint*: Assume that the principle of well-ordering is valid, and let P be a property for which

- 1.' $P(1)$ is true
- 2.' $(\forall k)[P(r)$ true for all $r, 1 \leq r \leq k \rightarrow P(k + 1)$ true]

Let T be the subset of the positive integers defined by

$$T = \{t \mid P(t) \text{ is not true}\}$$

Show that T is the empty set.

97. Prove that the set of odd positive integers is denumerable.
98. Prove that the set \mathbb{Z} of all integers is denumerable.
99. Prove that the set of all finite-length strings of the letter a is denumerable.
100. Prove that the set of all finite-length binary strings is denumerable.
101. Prove that the set $\mathbb{Z} \times \mathbb{Z}$ is denumerable.
102. In Example 23, the claim was made that $0.24999999 \dots$ is the same number as $0.25000000 \dots$. The first representation is a nonterminating decimal, and a calculus-type argument can be made that “in the limit” these are the same values. Here is a slightly different argument:
- a. Let $n = 0.249999 \dots$.
 Compute $100n$ by multiplying both sides of this equation by 100.
 Subtract n from $100n$ to give a value for $99n$.
 Solve the resulting equation for n .
 - b. Let $m = 0.250000 \dots$.
 Compute $100m$ by multiplying both sides of this equation by 100.
 Subtract m from $100m$ to give a value for $99m$.
 Solve the resulting equation for m .
 - c. Compare the values of n and m .
103. Use Cantor’s diagonalization method to show that the set of all infinite sequences of positive integers is not countable.
104. Use Cantor’s diagonalization method to show that the set of all infinite strings of the letters $\{a, b\}$ is not countable.
105. Explain why the union of any two denumerable sets is denumerable.
106. Explain why any subset of a countable set is countable.
107. Sets can have sets as elements (see Exercise 13, for example). Let B be the set defined as follows:

$$B = \{S \mid S \text{ is a set and } S \notin S\}$$

Argue that both $B \in B$ and $B \notin B$ are true. This contradiction is called *Russell’s paradox*, after the famous philosopher and mathematician Bertrand Russell, who stated it in 1901. (A carefully constructed axiomatization of set theory puts some restrictions on what can be called a set. All ordinary sets are still sets, but peculiar sets that get us into trouble, like B in this exercise, seem to be avoided.)

SECTION 4.2 COUNTING

Combinatorics is the branch of mathematics that deals with counting. Counting questions are important whenever we have finite resources (How much storage does a particular database consume? How many users can a given computer configuration support?) or whenever we are interested in efficiency (How many computations does a particular algorithm involve?).

Counting questions often boils down to finding how many members there are in some finite set, that is, what is the cardinality of the set. This seemingly trivial question can be difficult to answer. We have already answered some “how many” questions—How many rows are there in a truth table with n statement letters, and how many subsets are there in a set with n elements? (Actually, as we’ve noted, these questions can be thought of as the same question.)

Multiplication Principle

We solved the truth table question by drawing a tree of possibilities. This tree suggests a general principle that can be used to solve many counting problems. Before we state the general principle, we look at another tree example.

EXAMPLE 24

A child is allowed to choose one jellybean out of two jellybeans, one red and one black, and one gummy bear out of three gummy bears, yellow, green, and white. How many different sets of candy can the child have?

We can solve this problem by breaking the task of choosing candy into two sequential tasks of choosing the jellybean and then choosing the gummy bear. The tree of Figure 4.3 shows that there are $2 \times 3 = 6$ possible outcomes: $\{R, Y\}$, $\{R, G\}$, $\{R, W\}$, $\{B, Y\}$, $\{B, G\}$, and $\{B, W\}$.

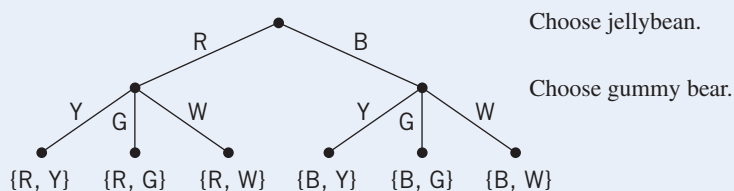


Figure 4.3

In this problem the sequence of events could be reversed; the child could choose the gummy bear first and the jellybean second, resulting in the tree of Figure 4.4, but the number of outcomes is the same ($3 \times 2 = 6$). Thinking of a sequence of successive events helps us solve the problem, but the sequencing is not a part of the problem since the set $\{R, Y\}$ is the same as the set $\{Y, R\}$.

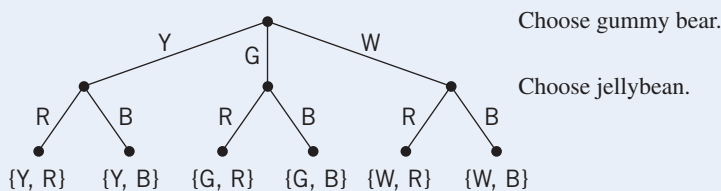


Figure 4.4

Both these trees are “balanced” in the sense that the second level has a fixed number of outcomes regardless of the outcome at the previous level. ●

Example 24 illustrates that the total number of outcomes for a sequence of events can be obtained by multiplying the number of outcomes for the first event by the number of outcomes for the second. This idea is summarized in the *multiplication principle*.

● **PRINCIPLE** **MULTIPLICATION PRINCIPLE**

If there are n_1 possible outcomes for a first event and n_2 possible outcomes for a second event, there are $n_1 \times n_2$ possible outcomes for the sequence of the two events.

REMINDER

Use the multiplication principle when there is a sequence of events.

The multiplication principle can be extended by induction to apply to a sequence of any finite number of events. (See Exercise 73 at the end of this section.) The multiplication principle is useful whenever we want to count the total number of possible outcomes for a task that can be broken down into a sequence of successive subtasks.

EXAMPLE 25

The last part of your telephone number contains four digits. How many such four-digit numbers are there?

We can construct four-digit numbers by performing a sequence of subtasks: choose the first digit, then the second, the third, and finally the fourth. The first digit can be any one of the 10 digits from 0 to 9, so there are 10 possible outcomes for the first subtask. Likewise, there are 10 different possibilities each for the second digit, the third, and the fourth. Using the multiplication principle, we multiply the number of outcomes for each subtask in the sequence. Therefore there are $10 \cdot 10 \cdot 10 \cdot 10 = 10,000$ different numbers. ●

If an element cannot be used again—that is, if repetitions are not allowed—the number of possible outcomes for successive events will be affected.

EXAMPLE 26

Referring to Example 25, how many four-digit numbers are there if the same digit cannot be used twice?

Again we have the sequence of subtasks of selecting the four digits, but no repetitions are allowed. There are 10 choices for the first digit, but only 9 choices for the second because we can't use what we used for the first digit, and so on. There are $10 \cdot 9 \cdot 8 \cdot 7 = 5040$ different numbers. ●

EXAMPLE 27

- How many ways are there to choose three officers from a club of 25 people?
- How many ways are there to choose three officers from a club of 25 people if someone can hold more than one office?

In (a), there are three successive subtasks with no repetitions. The first subtask, choosing the first officer, has 25 possible outcomes. The second subtask has 24 outcomes, the third 23 outcomes. The total number of outcomes is $25 \cdot 24 \cdot 23 = 13,800$. In (b), the same three subtasks are done in succession, but repetitions are allowed. The total number of outcomes is $25 \cdot 25 \cdot 25 = 15,625$. ●

EXAMPLE 28

When you order pizza at your favorite pizza place, you have the choice of thin crust, regular crust, or deep-dish; small, medium, or large; and pepperoni, sausage, barbeque, extra cheese, or vegetable. How many different pizzas can be ordered?

Again, there is a sequence of tasks: choose the crust, choose the size, and choose the topping. The total number of outcomes is $3 \cdot 3 \cdot 5 = 45$. ●

PRACTICE 22

If a man has four suits, eight shirts, and five ties, how many outfits can he put together? ■

EXAMPLE 29

For any finite set S , $|S|$ denotes the number of elements in S . If A and B are finite sets, then

$$|A \times B| = |A| \cdot |B|$$

$A \times B$ consists of all ordered pairs with first component from A and second component from B . Forming such ordered pairs can be thought of as the sequence of tasks of choosing the first component, for which there are $|A|$ outcomes, and then choosing the second component, for which there are $|B|$ outcomes. The result follows from the multiplication principle. ●

Addition Principle

Suppose we want to select a dessert from three pies and four cakes. In how many ways can this be done? There are two events, one with three outcomes (choosing a pie) and one with four outcomes (choosing a cake). However, we are not doing a sequence of two events here, since we are getting only one dessert, which must be chosen from the two disjoint sets of possibilities. The number of different outcomes is the total number of choices we have, $3 + 4 = 7$. This illustrates the *addition principle*.

● **PRINCIPLE** **ADDITION PRINCIPLE**

If A and B are disjoint events with n_1 and n_2 possible outcomes, respectively, then the total number of possible outcomes for event “ A or B ” is $n_1 + n_2$.

The addition principle can be extended by induction to the case of any finite number of disjoint events. (See Exercise 74 at the end of this section.) The addition principle is useful whenever we want to count the total number of possible outcomes for a task that can be broken down into disjoint cases.

EXAMPLE 30

A customer wants to purchase a vehicle from a dealer. The dealer has 23 autos and 14 trucks in stock. How many selections does the customer have?

The customer wants to choose a car or truck. These are disjoint events; choosing an auto has 23 outcomes and choosing a truck has 14. By the addition principle, choosing a vehicle has $23 + 14 = 37$ outcomes. Notice the requirement that the outcomes for events A and B be disjoint sets. Thus, if a customer wanted to purchase a vehicle from a dealer who had 23 autos, 14 trucks, and 17 red vehicles in stock, we could not conclude that the customer had $23 + 14 + 17$ choices! ●

REMINDER

Use the addition principle only when the events are disjoint—have no common outcomes.

EXAMPLE 31

Let A and B be disjoint finite sets. Then $|A \cup B| = |A| + |B|$.

Finding $|A \cup B|$ can be done by the disjoint cases of counting the number of elements in A , $|A|$, and the number of elements in B , $|B|$. By the addition principle, we sum these two numbers. ●

EXAMPLE 32

If A and B are finite sets, then

$$|A - B| = |A| - |A \cap B|$$

and

$$|A - B| = |A| - |B| \text{ if } B \subseteq A$$

To prove the first equality, note that

$$\begin{aligned} (A - B) \cup (A \cap B) &= (A \cap B') \cup (A \cap B) \\ &= A \cap (B' \cup B) \\ &= A \cap S \\ &= A \end{aligned}$$

so that $A = (A - B) \cup (A \cap B)$. Also, $A - B$ and $A \cap B$ are disjoint sets; therefore, by Example 31,

$$|A| = |(A - B) \cup (A \cap B)| = |A - B| + |A \cap B|$$

or

$$|A - B| = |A| - |A \cap B|$$

The second equation follows from the first, because if $B \subseteq A$ then $A \cap B = B$. ●

Using the Principles Together

Frequently the addition principle is used in conjunction with the multiplication principle.

EXAMPLE 33

Referring to Example 24, suppose we want to find how many different ways the child can *choose* the candy, rather than the number of sets of candy the child can have. Then choosing a red jellybean followed by a yellow gummy bear is not the same as choosing a yellow gummy bear followed by a red jellybean. We can consider two disjoint cases—choosing jellybeans first or choosing gummy bears first. Each of these cases (by the multiplication principle) has six outcomes, so (by the addition principle) there are $6 + 6 = 12$ possible ways to choose the candy. ●

EXAMPLE 34

How many four-digit numbers begin with a 4 or a 5?

We can consider the two disjoint cases—numbers that begin with 4 and numbers that begin with 5. Counting the numbers that begin with 4, there is 1 outcome for the subtask of choosing the first digit, then 10 possible outcomes for the subtasks of choosing each of the other three digits. Hence, by the multiplication principle there are $1 \cdot 10 \cdot 10 \cdot 10 = 1000$ ways to get a four-digit number beginning with 4. The same reasoning shows that there are 1000 ways to get a four-digit number beginning with 5. By the addition principle, there are $1000 + 1000 = 2000$ total possible outcomes. ●

PRACTICE 23

If a woman has 7 blouses, 5 skirts, and 9 dresses, how many different outfits does she have? ■

Often a counting problem can be solved in more than one way. Although the possibility of a second solution might seem confusing, it provides an excellent way to check our work; if two different ways of looking at the problem produce the same answer, it increases our confidence that we have analyzed the problem correctly.

EXAMPLE 35

Consider the problem of Example 34 again. We can avoid using the addition principle by thinking of the problem as four successive subtasks, where the first subtask, choosing the first digit, has two possible outcomes—choosing a 4 or choosing a 5. Then there are $2 \cdot 10 \cdot 10 \cdot 10 = 2000$ possible outcomes. ●

EXAMPLE 36

How many three-digit integers (numbers between 100 and 999 inclusive) are even?

One solution notes that an even number ends in 0, 2, 4, 6, or 8. Taking these as separate cases, the number of three-digit integers ending in 0 can be found by choosing the three digits in turn. There are 9 choices, 1 through 9, for the first digit; 10 choices, 0 through 9, for the second digit; and 1 choice for the third digit, 0. By the multiplication principle, there are 90 numbers ending in 0. Similarly, there are 90 numbers ending in 2, 4, 6, and 8, so by the addition principle, there are $90 + 90 + 90 + 90 + 90 = 450$ numbers.

Another solution takes advantage of the fact that there are only 5 choices for the third digit. By the multiplication principle, there are $9 \cdot 10 \cdot 5 = 450$ numbers.

For this problem, there is a third solution of the “serendipity” type we discussed in Section 2.1. There are $999 - 100 + 1 = 900$ three-digit integers in the range specified. Half are even and half are odd, so 450 of them must be even. ●

EXAMPLE 37

Suppose the last four digits of a telephone number must include at least one repeated digit. How many such numbers are there?

Although it is possible to do this problem by using the addition principle directly, it is difficult because there are so many disjoint cases to consider. For example, if the first two digits are alike but the third and fourth are different, there are $10 \cdot 1 \cdot 9 \cdot 8$ ways this can happen. If the first and third digit are alike but the second

and fourth are different, there are $10 \cdot 9 \cdot 1 \cdot 8$ ways this can happen. If the first two digits are alike and the last two are also alike but different from the first two, there are $10 \cdot 1 \cdot 9 \cdot 1$ such numbers. Obviously, there are many other possibilities.

Instead, we solve the problem by noting that numbers with repetitions and numbers with no repetitions are disjoint sets whose union equals all four-digit numbers. By Example 31 we can find the number with repetitions by subtracting the number with no repetitions (5040, according to Example 26) from the total number (10,000, according to Example 25). Therefore, there are 4960 numbers with repetitions. ●

EXAMPLE 38

A computer (or tablet, or camera, or cell phone) that wishes to connect to the Internet must have an IP (Internet Protocol) address assigned to it. This allows the device to be “found” over the Internet, much as a postal address allows a building to be “found” via regular mail. The version of IP known as IPv4 uses an address that is a 32-bit, or 4-byte, number (1 byte equals 8 bits). The first part of the address, called the netid, identifies the network that the machine is part of and the rest, called the hostid, identifies the machine itself. Note that this is a hierarchical addressing scheme. A router trying to decide where to send a data packet looks at the netid to determine the network. The hostid bytes need never be consulted until the data packet has reached the correct network. U.S. postal addresses are hierarchical in the opposite order, with the most specific information first.

How many different IPv4 addresses are there? Each of the 32 bits can be set to 0 or 1, so by the multiplication principle, there are $2 \cdot 2 \cdot 2 \cdots 2 = 2^{32}$ different bit patterns. Looking at a more abstract view, assume that a particular IP address uses 16 bits for the netid and 16 bits for the hostid. Again using the multiplication principle, this would give $2^{16} \cdot 2^{16} =$ (again) 2^{32} unique IP addresses. This number is roughly 4.3 billion, which seems large enough to satisfy the world’s needs. But no — the pool of IPv4 addresses allotted to some regions of the world began to run out in 2011 and more would do so in another year or two. Hence the switch to IPv6.

An IPv6 address is 128 bits, divided into 64 bits for the network prefix that identifies a particular network and the last 64 bits for the interface ID that identifies the unique node on the network. While the gross structure of an IPv6 address therefore sounds just like an IPv4 address only bigger, there are details that make the IPv6 scheme more efficient. And exactly how big is the pool of IPv6 addresses? Using the same reasoning as before, there are 2^{128} unique addresses. This number is roughly 3.4×10^{38} , or 340 trillion trillion trillion, enough, it is said, for every star in the known universe to have the equivalent of its very own IPv4 internet.

The World IPv6 Launch occurred on June 6, 2012, but it was not like turning on a switch. Many major companies already supported IPv6 and IPv4 will continue to be supported over a few years of transition. ●

Decision Trees

Trees such as those in Figures 4.3 and 4.4 illustrate the number of outcomes of an event based on a series of possible choices. Such trees are called **decision trees**. We will see in Chapter 6 how decision trees are used in analyzing algorithms, but for now we use them to solve additional counting problems. The trees of Figures 4.3 and 4.4 led to the multiplication principle because the number of

outcomes at any one level of the tree is the same throughout that level. In Figure 4.4, for example, level 2 of the tree shows two outcomes for each of the 3 branches formed at level 1. Less regular decision trees can still be used to solve counting problems where the multiplication principle does not apply.

EXAMPLE 39

Tony is pitching pennies. Each toss results in heads (H) or tails (T). How many ways can he toss the coin 5 times without having 2 heads in a row?

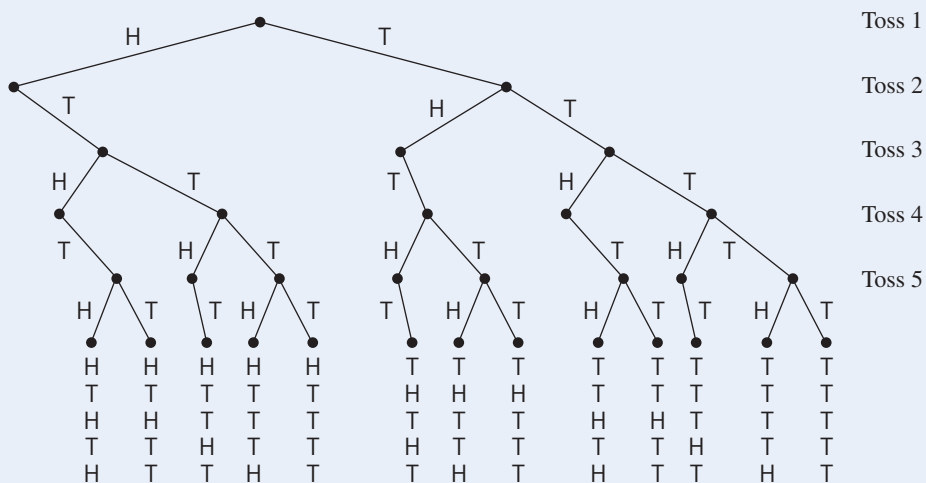


Figure 4.5

Figure 4.5 shows the decision tree for this problem. Each coin toss has 2 outcomes; the left branch is labeled H for heads, the right branch is labeled T for tails. Whenever an H appears on a branch, the next level can only contain a right (T) branch. There are 13 possible outcomes, shown at the bottom of the tree.

PRACTICE 24 Explain why the multiplication principle does not apply to Example 39.

PRACTICE 25 Draw the decision tree for the number of strings of X 's, Y 's, and Z 's with length 3 that do not have a Z following a Y .

SECTION 4.2 REVIEW

TECHNIQUE

W Use the multiplication principle, the addition principle, and decision trees for counting the number of objects in a finite set.

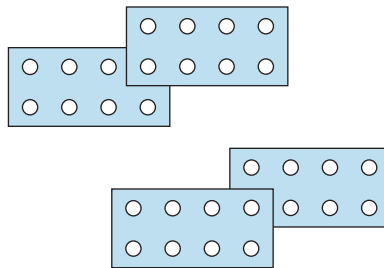
MAIN IDEAS

- The multiplication principle is used to count the number of possible outcomes for a sequence of events, each of which has a fixed number of outcomes.
- The addition principle is used to count the number of possible outcomes for disjoint events.
- The multiplication and addition principles are often used together.
- Decision trees can be used to count the number of outcomes for a sequence of events where the number of outcomes for a given event is not constant but depends on the outcome of the preceding event.

EXERCISES 4.2

1. A frozen yogurt shop allows you to choose one flavor (vanilla, strawberry, lemon, cherry, or peach), one topping (chocolate shavings, crushed toffee, or crushed peanut brittle), and one condiment (whipped cream or shredded coconut). How many different desserts are possible?
2. In Exercise 1, how many dessert choices do you have if you are allergic to strawberries and chocolate?
3. A video game is begun by making selections from each of 3 menus. The first menu (number of players) has 4 selections, the second menu (level of play) has 8, and the third menu (speed) has 6. In how many configurations can the game be played?
4. A multiple choice exam has 20 questions each with 4 possible answers and 10 additional questions each with 5 possible answers. How many different answer sheets are possible?
5. A user's password to access a computer system consists of 3 letters followed by 2 digits. How many different passwords are possible?
6. On the computer system of Exercise 5, how many passwords are possible if uppercase and lowercase letters can be distinguished?
7. A telephone conference call is being placed from Central City to Booneville by way of Cloverdale. There are 45 trunk lines from Central City to Cloverdale and 13 from Cloverdale to Booneville. How many different ways can the call be placed?
8. A , B , C , and D are nodes on a computer network. There are 2 paths between A and C , 2 between B and D , 3 between A and B , and 4 between C and D . Along how many routes can a message from A to D be sent?
9. How many nine-digit Social Security numbers are possible?
10. An apartment building purchases a new lock system for its 175 units. A lock is opened by punching in a two-digit code. Has the apartment management made a wise purchase?
11. A palindrome is a string of characters that reads the same forward and backward. How many five-letter English language palindromes are possible?
12. How many three-digit numbers less than 600 can be made using the digits 8, 6, 4, and 2?
13. A binary logical connective can be defined by giving its truth table. How many different binary logical connectives are there?
14. Three seats on the county council are to be filled, each with someone from a different party. There are 4 candidates running from the Concerned Environmentalist party, 3 from the Limited Development party, and 2 from the Friends of the Spotted Newt party. In how many ways can the seats be filled?
15. In the original BASIC programming language, an identifier must be either a single letter or a letter followed by a single digit. How many identifiers are possible?
16. A president and vice-president must be chosen for the executive committee of an organization. There are 17 volunteers from the Eastern Division and 24 volunteers from the Western Division. If both officers must come from the same division, in how many ways can the officers be selected?
17. A dinner special allows you to select from 5 appetizers, 3 salads, 4 entrees, and 3 beverages. How many different dinners are there?
18. In Exercise 17, how many different dinners are there if you may have an appetizer or a salad but not both?
19. A new car can be ordered with a choice of 10 exterior colors; 7 interior colors; automatic or three-speed or five-speed transmission; with or without air conditioning; with or without power steering; and with or without the option package that contains the power door lock and the rear-window defroster. How many different cars can be ordered?
20. In Exercise 19, how many different cars can be ordered if the option package comes only on a car with an automatic transmission?

21. In one state, automobile license plates must have two digits (no leading zeros) followed by one letter followed by a string of two to four digits (leading zeros are allowed). How many different plates are possible?
22. A Hawaiian favorite fast food is the “loco moco,” invented at a Hilo restaurant. It consists of a bed of rice under a meat patty with egg on top, the whole thing smothered in brown gravy. The rice can be white or brown, the egg can be fried, scrambled, or poached, and the meat can be hamburger, Spam, Portuguese sausage, bacon, turkey, hot dog, salmon, or mahi. How many different loco mocos can be ordered?
23. A customer at a fast-food restaurant can order a hamburger with or without mustard, ketchup, pickle, or onion; a fish sandwich with or without lettuce, tomato, or tartar sauce; and a choice of 3 kinds of soft drinks or 2 kinds of milk shakes. How many different orders are possible if a customer can order at most 1 hamburger, 1 fish sandwich, and 1 beverage but can order less?
24. How many unique ways are there to stack two 2×4 Lego bricks of the same color? Two stacks that look the same if you merely rotate them are considered to be the same arrangement. Here is one such stack, the original on top and the same stack after rotating 180° .



25. What is the value of *Count* after the following pseudocode has been executed?

```

Count = 0
for i = 1 to 5 do
  for Letter = 'A' to 'C' do
    Count = Count + 1
  end for
end for

```

26. What is the value of *Result* after the following pseudocode has been executed?

```

Result = 0
for Index = 20 down to 10 do
  for Inner = 5 to 10 do
    Result = Result + 2
  end for
end for

```

Exercises 27–32 concern the set of three-digit integers (numbers between 100 and 999 inclusive).

27. How many are divisible by 5?
28. How many are divisible by 4?
29. How many are not divisible by 5?
30. How many are divisible by 4 or 5?
31. How many are divisible by 4 and 5?
32. How many are divisible by neither 4 nor 5?

Exercises 33–42 concern the set of binary strings of length 8 (each character is either the digit 0 or the digit 1).

33. How many such strings are there?
34. How many begin and end with 0?
35. How many begin or end with 0?
36. How many have 1 as the second digit?
37. How many begin with 111?
38. How many contain exactly one 0?
39. How many begin with 10 or have a 0 as the third digit?
40. How many are palindromes? (See Exercise 11.)
41. How many contain exactly seven 1s?
42. How many contain two or more 0s?

In Exercises 43–48, 2 dice are rolled, 1 black and 1 white.

43. How many different rolls are possible? (Note that a 4-black, 1-white result and a 1-black, 4-white result are two different outcomes.)
44. How many rolls result in “snake eyes” (both dice showing 1)?
45. How many rolls result in doubles (both dice showing the same value)?
46. How many rolls result in a total of 7 or 11?
47. How many rolls occur in which neither die shows the value 4?
48. How many rolls occur in which the value on the white die is greater than the value on the black die?

In Exercises 49–54, a customer is ordering a new desktop computer system. The choices are 21-inch, 23-inch, or 24-inch monitor (optional); 1 TB or 2 TB hard drive; 6 GB or 8 GB of RAM; 16X DVD or Blu-ray disk optical drive; Intel, AMD, or NVIDIA video card; inkjet, laser, or laser color printer; 1-, 2-, or 3-year warranty.

49. How many different machine configurations are possible?
50. How many different machines can be ordered with a 2 TB hard drive?
51. How many different machines can be ordered with a 21-inch monitor and an inkjet printer?
52. How many different machines can be ordered if the customer decides not to get a new monitor?
53. How many different machines can be ordered with a 21-inch monitor, a 1 TB hard drive, and an inkjet printer?
54. How many different machines can be ordered if the customer does not want a 3-year warranty?

Exercises 55–58 refer to the Konami code, a sequence of 10 keystrokes using 6 different characters: $\uparrow\uparrow\downarrow\downarrow\leftarrow\rightarrow\leftarrow\rightarrow$ B A. This code was inadvertently left by the developer in the first release of a video game; when entered on the video game console as the title screen is open, it surreptitiously adds assets to the player’s on-screen avatar. This code or versions of it have been retained in many video games.

55. How many 10-character codes can be created using these 6 characters?
56. How many 10-character codes can be created if the BA sequence must be the last 2 characters?
57. How many 10-character codes can be created if only 2 down-arrow characters can be used and they must be paired together wherever they appear?
58. How many 10-character codes can be created if only the arrow characters are used?

In Exercises 59–68, a hand consists of 1 card drawn from a standard 52-card deck with flowers on the back and 1 card drawn from a standard 52-card deck with birds on the back. A standard deck has 13 cards from each of 4 suits (clubs, diamonds, hearts, spades). The 13 cards have face value 2 through 10, jack, queen, king, or ace. Each face value is a “kind” of card. The jack, queen, and king are “face cards.”

59. How many different hands are possible? (Note that a flower-ace-of-spades, bird-queen-of-hearts and a flower-queen-of-hearts, bird-ace-of-spades are two different outcomes.)
60. How many hands consist of a pair of aces?
61. How many hands contain all face cards?
62. How many hands contain exactly 1 king?
63. How many hands consist of two of a kind (2 aces, 2 jacks, and so on)?
64. How many hands have a face value of 5 (aces count as 1, face cards count as 10)?
65. How many hands have a face value of less than 5 (aces count as 1, face cards count as 10)?
66. How many hands do not contain any face cards?
67. How many hands contain at least 1 face card?
68. How many hands contain at least 1 king?
69. Draw a decision tree to find the number of binary strings of length 4 that do not have consecutive 0s. (Compare your answer with the one for Exercise 41 of Section 3.2.)
70. Draw a decision tree (use teams A and B) to find the number of ways the NBA playoffs can happen, where the winner is the first team to win 4 out of 7.
71. Voting on a certain issue is conducted by having everyone put a red, blue, or green slip of paper into a hat. Then the slips are pulled out one at a time. The first color to receive two votes wins. Draw a decision tree to find the number of ways in which the balloting can occur.
72. In Example 39, prove the following facts, where $C(n)$ = the total number of nodes in the decision tree at level n , $H(n)$ = the total number of nodes at level n resulting from an H toss, $T(n)$ = the total number of nodes at level n resulting from a T toss.¹
 - a. $C(n) = H(n) + T(n)$
 - b. $H(n) = T(n - 1)$
 - c. $T(n) = H(n - 1) + T(n - 1)$
 - d. $H(n) = H(n - 2) + T(n - 2)$
 - e. $C(n) = C(n - 2) + C(n - 1)$ for $n \geq 3$
 - f. $C(n) = F(n + 1)$ where $F(n)$ is the n th Fibonacci number
73. Use mathematical induction to extend the multiplication principle to a sequence of m events for any integer m , $m \geq 2$.
74. Use mathematical induction to extend the addition principle to m disjoint events for any integer m , $m \geq 2$.
75. Consider the product of n factors, $x_1 \cdot x_2 \cdot \cdots \cdot x_n$. Such an expression can be fully parenthesized to indicate the order of multiplication in a number of ways. For example, if $n = 4$, there are five ways to parenthesize:

$$\begin{aligned}
 &x_1 \cdot (x_2 \cdot (x_3 \cdot x_4)) \\
 &x_1 \cdot ((x_2 \cdot x_3) \cdot x_4) \\
 &(x_1 \cdot x_2) \cdot (x_3 \cdot x_4) \\
 &(x_1 \cdot (x_2 \cdot x_3)) \cdot x_4 \\
 &((x_1 \cdot x_2) \cdot x_3) \cdot x_4
 \end{aligned}$$

¹This problem was suggested by Mr. Tracy Castile, a former University of Hawaii at Hilo student.

- a. Write a recurrence relation for $P(n)$, the number of ways to parenthesize a product of n factors, $n \geq 1$. Assume that $P(1) = 1$. (*Hint*: Note that for $n > 2$, the last multiplication to be performed can occur in any of $n - 1$ positions.)
- b. Prove that

$$P(n) = C(n - 1)$$

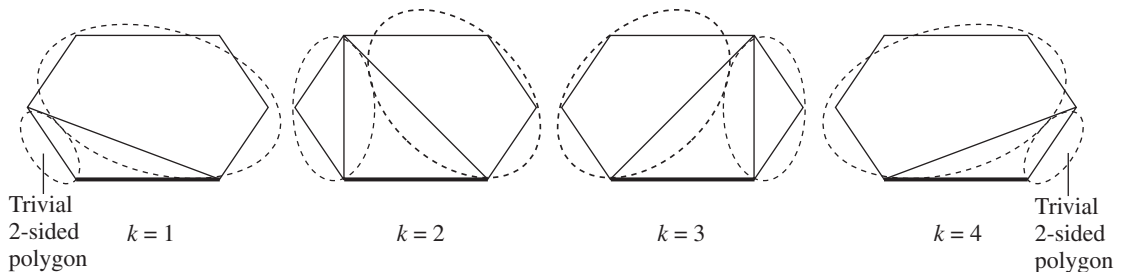
where $C(0), C(1), \dots$ is the sequence of Catalan numbers (see Exercise 38 of Section 3.1).

76. A *simple closed convex polygon* consists of n points in the plane joined in pairs by n line segments; each point is the endpoint of exactly 2 line segments, and any line connecting 2 nonadjacent points lies wholly within the polygon.
- a. Show that an $(n + 2)$ -sided simple closed convex polygon can be triangulated (divided into triangular regions) using $n - 1$ lines. (The figures show two different triangulations of a 6-sided polygon, where $n = 4$).



(*Hint*: Use a pair of straight lines to shave off 2 corners; consider the cases where n is even and the cases where n is odd.)

- b. Write a recurrence relation for $T(n)$, the number of different triangulations of an $(n + 2)$ -sided polygon. Assume that $T(0) = 1$. (*Hint*: Fix one edge of the polygon as the base of a triangle whose tip rotates around the polygon, as shown. Use the sides of the triangle to divide the polygon into 2 polygonal sections with $(k + 1)$ sides and $(n - k + 2)$ sides.)



- c. Prove that $T(n) = C(n)$, where $C(0), C(1), \dots$ is the sequence of Catalan numbers (see Exercise 38 of Section 3.1).

SECTION 4.3

PRINCIPLE OF INCLUSION AND EXCLUSION; PIGEONHOLE PRINCIPLE

In this section we discuss two more counting principles that can be used to solve combinatorics problems.

Principle of Inclusion and Exclusion

To develop the principle of inclusion and exclusion, we first note that if A and B are any subsets of a universal set S , then $A - B$, $B - A$, and $A \cap B$ are mutually disjoint sets (see Figure 4.6). For example, if $x \in A - B$, then $x \notin B$, therefore $x \notin B - A$ and $x \notin A \cap B$.

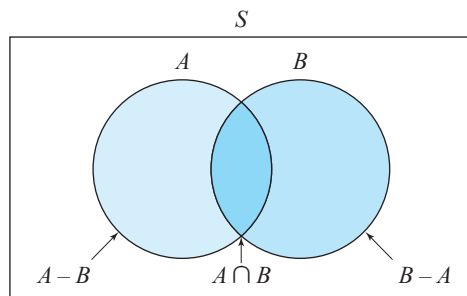


Figure 4.6

Also, something can be said about the union of these three sets.

PRACTICE 26 What is another name for the set $(A - B) \cup (B - A) \cup (A \cap B)$? ■

From Example 31 (extended to three disjoint finite sets),

$$|(A - B) \cup (B - A) \cup (A \cap B)| = |A - B| + |B - A| + |A \cap B| \quad (1)$$

From Example 32,

$$|A - B| = |A| - |A \cap B|$$

and

$$|B - A| = |B| - |A \cap B|$$

Using these expressions in Equation (1), along with the result of Practice 26, we get

$$|A \cup B| = |A| - |A \cap B| + |B| - |A \cap B| + |A \cap B|$$

or

$$|A \cup B| = |A| + |B| - |A \cap B| \quad (2)$$

Equation (2) is the two-set version of the principle of inclusion and exclusion. The name derives from the fact that when counting the number of elements in the union of A and B , we must “include” (count) the number of elements in A and the number of elements in B , but we must “exclude” (subtract) those elements in $A \cap B$ to avoid counting them twice.

PRACTICE 27 How does Equation (2) relate to Example 31 of Section 4.2?

EXAMPLE 40

A pollster queries 35 voters, all of whom support referendum 1, referendum 2, or both, and finds that 14 voters support referendum 1 and 26 support referendum 2. How many voters support both?

If we let A be the set of voters supporting referendum 1 and B be the set of voters supporting referendum 2, then we know that

$$|A \cup B| = 35 \quad |A| = 14 \quad |B| = 26$$

From Equation (2),

$$\begin{aligned} |A \cup B| &= |A| + |B| - |A \cap B| \\ 35 &= 14 + 26 - |A \cap B| \\ |A \cap B| &= 14 + 26 - 35 = 5 \end{aligned}$$

so 5 voters support both.

Equation (2) can easily be extended to three sets, as follows:

$$\begin{aligned} |A \cup B \cup C| &= |A \cup (B \cup C)| \\ &= |A| + |B \cup C| - |A \cap (B \cup C)| \\ &= |A| + |B| + |C| - |B \cap C| - |(A \cap B) \cup (A \cap C)| \\ &= |A| + |B| + |C| - |B \cap C| - (|A \cap B| + |A \cap C| - |A \cap B \cap C|) \\ &= |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C| \end{aligned}$$

Therefore the three-set version of the principle of inclusion and exclusion is

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C| \quad (3)$$

PRACTICE 28 Justify each of the equalities used in deriving Equation (3).

In addition to the formal derivation of Equation (3) that we just did, a sort of geometric argument for $|A \cup B \cup C|$ is suggested by Figure 4.7 on the next page. When we add $|A| + |B| + |C|$, we are counting each of $|A \cap B|$, $|A \cap C|$, and $|B \cap C|$ twice, so we must throw each of them away once. When we add $|A| + |B| + |C|$, we are counting $|A \cap B \cap C|$ three times, but in subtracting $|A \cap B|$, $|A \cap C|$, $|B \cap C|$ we have thrown it away three times, so we must add it back once.

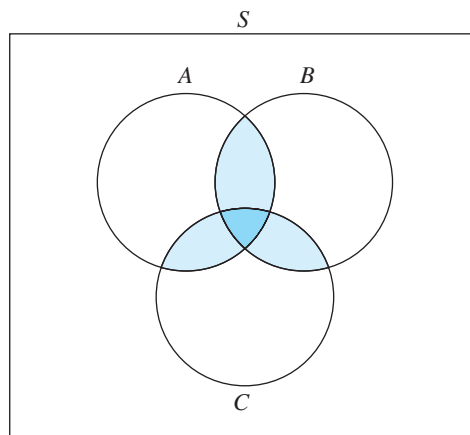


Figure 4.7

EXAMPLE 41

A group of students plans to order pizza. If 13 will eat sausage topping, 10 will eat pepperoni, 12 will eat extra cheese, 4 will eat both sausage and pepperoni, 5 will eat both pepperoni and extra cheese, 7 will eat both sausage and extra cheese, and 3 will eat all three toppings, how many students are in the group?

Let

$A = \{\text{students who will eat sausage}\}$

$B = \{\text{students who will eat pepperoni}\}$

$C = \{\text{students who will eat extra cheese}\}$

Then $|A| = 13$, $|B| = 10$, $|C| = 12$, $|A \cap B| = 4$, $|B \cap C| = 5$, $|A \cap C| = 7$, and $|A \cap B \cap C| = 3$. From Equation (3),

$$|A \cup B \cup C| = 13 + 10 + 12 - 4 - 5 - 7 + 3 = 22$$

We can also solve this problem by filling in all the pieces in a Venn diagram. Working from the middle outward, we know there are 3 people in $|A \cap B \cap C|$ (Figure 4.8a). We also know the number in each of $|A \cap B|$, $|B \cap C|$, and $|A \cap C|$, so with a little subtraction we can fill in more pieces (Figure 4.8b). We also know the size of A , B , and C , allowing us to complete the picture (Figure 4.8c). Now the total number of students, 22, is obtained by adding up all the numbers.

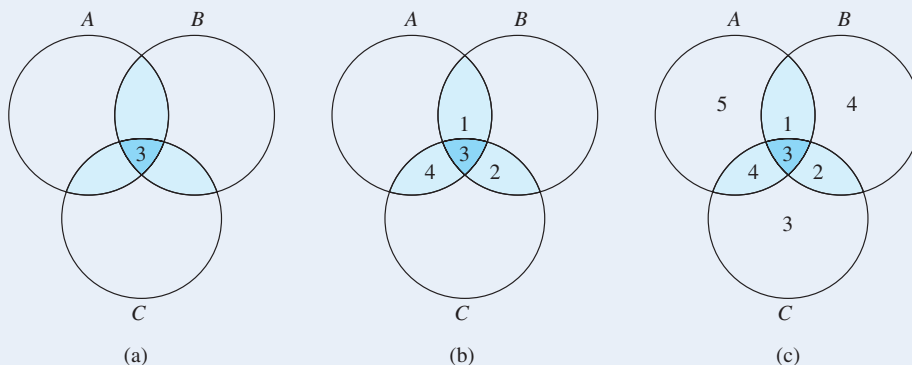


Figure 4.8

Although we are about to generalize Equation (3) to an arbitrary number of sets, the Venn diagram approach gets too complicated to draw with more than three sets.

EXAMPLE 42

A produce stand sells only broccoli, carrots, and okra. One day the stand served 207 people. If 114 people purchased broccoli, 152 purchased carrots, 25 purchased okra, 64 purchased broccoli and carrots, 12 purchased carrots and okra, and 9 purchased all three, how many people purchased broccoli and okra?

Let

$$A = \{\text{people who purchased broccoli}\}$$

$$B = \{\text{people who purchased carrots}\}$$

$$C = \{\text{people who purchased okra}\}$$

Then $|A \cup B \cup C| = 207$, $|A| = 114$, $|B| = 152$, $|C| = 25$, $|A \cap B| = 64$, $|B \cap C| = 12$, and $|A \cap B \cap C| = 9$. From Equation (3),

$$\begin{aligned} |A \cup B \cup C| &= |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C| \\ 207 &= 114 + 152 + 25 - 64 - |A \cap C| - 12 + 9 \\ |A \cap C| &= 114 + 152 + 25 - 64 - 12 + 9 - 207 = 17 \end{aligned}$$

In Equation (2), we add the number of elements in the single sets and subtract the number of elements in the intersection of two sets. In Equation (3), we add the number of elements in the single sets, subtract the number of elements in the intersection of two sets, and add the number of elements in the intersection of three sets. This seems to suggest a pattern: If we have n sets, we should add the number of elements in the single sets, subtract the number of elements in the intersection of two sets, add the number of elements in the intersection of three sets, subtract the number of elements in the intersection of four sets, and so on. This leads us to the general form of the principle of inclusion and exclusion:

● **PRINCIPLE OF INCLUSION AND EXCLUSION**

Given the finite sets A_1, \dots, A_n , $n \geq 2$, then

$$\begin{aligned} |A_1 \cup \dots \cup A_n| &= \sum_{1 \leq i \leq n} |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| \\ &\quad + \sum_{1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k| \\ &\quad - \dots + (-1)^{n+1} |A_1 \cap \dots \cap A_n| \end{aligned} \quad (4)$$

In Equation (4) the notation

$$\sum_{1 \leq i < j \leq n} |A_i \cap A_j|$$

for example, says to add together the number of elements in all the intersections of the form $A_i \cap A_j$ where i and j can take on any values between 1 and n as long as $i < j$.

For $n = 3$, this gives $|A_1 \cap A_2|$ ($i = 1, j = 2$), $|A_1 \cap A_3|$ ($i = 1, j = 3$), and $|A_2 \cap A_3|$ ($i = 2, j = 3$). This agrees with Equation (3), where $A_1 = A$, $A_2 = B$, and $A_3 = C$.

To prove the general form of the principle of inclusion and exclusion, we use mathematical induction. Although the idea of the proof is straightforward, the notation is rather messy. The base case, $n = 2$, is just Equation (2). We assume that Equation (4) is true for $n = k$ and show that it is true for $n = k + 1$. We write

$$\begin{aligned} & |A_1 \cup \cdots \cup A_{k+1}| \\ &= |(A_1 \cup \cdots \cup A_k) \cup A_{k+1}| \\ &= |(A_1 \cup \cdots \cup A_k)| + |A_{k+1}| \\ &\quad - |(A_1 \cup \cdots \cup A_k) \cap A_{k+1}| \quad (\text{by Equation (2)}) \\ &= \sum_{1 \leq i \leq k} |A_i| - \sum_{1 \leq i < j \leq k} |A_i \cap A_j| + \sum_{1 \leq i < j < m \leq k} |A_i \cap A_j \cap A_m| \\ &\quad - \cdots + (-1)^{k+1} |A_1 \cap \cdots \cap A_k| + |A_{k+1}| \\ &\quad - |(A_1 \cap A_{k+1}) \cup \cdots \cup (A_k \cap A_{k+1})| \end{aligned}$$

(by the inductive hypothesis and the distributive property)

$$\begin{aligned} &= \sum_{1 \leq i \leq k+1} |A_i| - \sum_{1 \leq i < j \leq k} |A_i \cap A_j| + \sum_{1 \leq i < j < m \leq k} |A_i \cap A_j \cap A_m| \\ &\quad - \cdots + (-1)^{k+1} |A_1 \cap \cdots \cap A_k| \\ &\quad - \left(\sum_{1 \leq i \leq k} |A_i \cap A_{k+1}| - \sum_{1 \leq i < j \leq k} |A_i \cap A_j \cap A_{k+1}| + \cdots + \right. \\ &\quad \left. + (-1)^k \sum_{1 \leq i < j < \cdots < m \leq k} \underbrace{|(A_i \cap A_{k+1}) \cap (A_j \cap A_{k+1}) \cap \cdots \cap (A_m \cap A_{k+1})|}_{k-1 \text{ terms}} \right. \\ &\quad \left. + (-1)^{k+1} |A_1 \cap \cdots \cap A_{k+1}| \right) \end{aligned}$$

(by combining terms ① from above and using the inductive hypothesis on the k sets $A_1 \cap A_{k+1}, A_2 \cap A_{k+1}, \dots, A_k \cap A_{k+1}$)

$$\begin{aligned} &= \sum_{1 \leq i \leq k+1} |A_i| - \sum_{1 \leq i < j \leq k+1} |A_i \cap A_j| + \sum_{1 \leq i < j < m \leq k+1} |A_i \cap A_j \cap A_m| \\ &\quad - \cdots - (-1)^{k+1} |A_1 \cap \cdots \cap A_{k+1}| \end{aligned}$$

(by combining like-numbered terms from above)

$$\begin{aligned} &= \sum_{1 \leq i \leq k+1} |A_i| - \sum_{1 \leq i < j \leq k+1} |A_i \cap A_j| + \sum_{1 \leq i < j < m \leq k+1} |A_i \cap A_j \cap A_m| \\ &\quad - \cdots + (-1)^{k+2} |A_1 \cap \cdots \cap A_{k+1}| \end{aligned}$$

This completes the proof of Equation (4). A different proof of the principle of inclusion and exclusion can be found in Exercise 23 of Section 4.5.

Pigeonhole Principle

The **pigeonhole principle** acquired its quaint name from the following idea: If more than k pigeons fly into k pigeonholes, then at least 1 hole will end up with more than 1 pigeon. Although this seems immediately obvious, we can belabor the point. Suppose each of the k pigeonholes contains at most 1 pigeon. Then there are at most k pigeons, not the more-than- k pigeons that supposedly flew in.

Now we'll state the pigeonhole principle in a less picturesque way.

- **PRINCIPLE PIGEONHOLE PRINCIPLE**
If more than k items are placed into k bins, then at least 1 bin contains more than 1 item.

By cleverly choosing items and bins, a number of interesting counting problems can be solved (see Example 7 of Chapter 2).

EXAMPLE 43 How many people must be in a room to guarantee that 2 people have last names that begin with the same initial?

There are 26 letters of the alphabet (bins). If there are 27 people, then there are 27 initials (items) to put into the 26 bins, so at least 1 bin will contain more than 1 last initial. ●

PRACTICE 29 How many times must a single die be rolled in order to guarantee getting the same value twice? ■

EXAMPLE 44 Prove that if 51 positive integers between 1 and 100 are chosen, then one of them must divide another.

Let the integers be n_1, \dots, n_{51} . Each integer $n_i \geq 2$ can be written as a product of prime numbers (fundamental theorem of arithmetic), every prime number except 2 is odd, and the product of odd numbers is odd. Therefore for each i , $n_i = 2^{k_i}b_i$, where $k_i \geq 0$ and b_i is an odd number. Furthermore, $1 \leq b_i \leq 99$, and there are 50 odd integers between 1 and 99 inclusive, but there are 51 b values. By the pigeonhole principle, $b_i = b_j$ for some i and j , so $n_i = 2^{k_i}b_i$ and $n_j = 2^{k_j}b_i$. If $k_i \leq k_j$, then n_i divides n_j ; otherwise, n_j divides n_i . ●

SECTION 4.3 REVIEW

TECHNIQUES

- Use the principle of inclusion and exclusion to find the number of elements in the union of sets.
- Use the pigeonhole principle to find the minimum number of elements to guarantee two with a duplicate property.

MAIN IDEA

- The principle of inclusion and exclusion and the pigeonhole principle are additional counting mechanisms for sets.

EXERCISES 4.3

1. All the guests at a dinner party drink coffee or tea; 13 guests drink coffee, 10 drink tea, and 4 drink both coffee and tea. How many people are guests at the dinner party?
2. In a group of 42 tourists, everyone speaks English or French; there are 35 English speakers and 18 French speakers. How many speak both English and French?
3. After serving 137 customers, a cafeteria notes at the end of the day that 56 orders of green beans were sold, 38 orders of beets were sold, and 17 customers purchased both green beans and beets. How many customers bought neither beans nor beets?
4. A bike show will be held for mountain bikes and road bikes. Of the 24 people who register for the show, 17 will bring road bikes and 5 will bring both road bikes and mountain bikes. How many will bring mountain bikes?
5. Quality control in a factory pulls 40 parts with paint, packaging, or electronics defects from an assembly line. Of these, 28 had a paint defect, 17 had a packaging defect, 13 had an electronics defect, 6 had both paint and packaging defects, 7 had both packaging and electronics defects, and 10 had both paint and electronics defects. Did any part have all three types of defect?
6. In a group of 24 people who like rock, country, and classical music, 14 like rock, 17 like classical, 11 like both rock and country, 9 like rock and classical, 13 like country and classical, and 8 like rock, country, and classical. How many like country?
7. Nineteen different mouthwash products make the following claims: 12 claim to freshen breath, 10 claim to prevent gingivitis, 11 claim to reduce plaque, 6 claim to both freshen breath and reduce plaque, 5 claim to both prevent gingivitis and freshen breath, and 5 claim to both prevent gingivitis and reduce plaque.
 - a. How many products make all three claims?
 - b. How many products claim to freshen breath but do not claim to reduce plaque?
8. From the 83 students who want to enroll in CS 320, 32 have completed CS 120, 27 have completed CS 180, and 35 have completed CS 215. Of these, 7 have completed both CS 120 and CS 180, 16 have completed CS 180 and CS 215, and 3 have completed CS 120 and CS 215. Two students have completed all three courses. The prerequisite for CS 320 is completion of one of CS 120, CS 180, or CS 215. How many students are not eligible to enroll?
9. A survey of 150 college students reveals that 83 own automobiles, 97 own bikes, 28 own motorcycles, 53 own a car and a bike, 14 own a car and a motorcycle, 7 own a bike and a motorcycle, and 2 own all three.
 - a. How many students own a bike and nothing else?
 - b. How many students do not own any of the three?
10. Among a bank's 214 customers with checking or savings accounts, 189 have checking accounts, 73 have regular savings accounts, 114 have money market savings accounts, and 69 have both checking and regular savings accounts. No customer is allowed to have both regular savings and money market savings accounts.
 - a. How many customers have both checking and money market savings accounts?
 - b. How many customers have a checking account but no savings account?
11. At the beginning of this chapter you surveyed the 87 computer users who subscribe to your electronic newsletter in preparation for the release of your new software product.

The results of your survey reveal that of the 87 subscribers, 68 have a Windows-based system available to them, 34 have a Linux system available, and 30 have access to a Mac. In addition, 19 have access to both Windows and Linux systems, 11 have access to both Linux systems and Macs, and 23 can use both Macs and Windows.

Use the principle of inclusion and exclusion to determine how many subscribers have access to all three types of systems.

12. You are developing a new bath soap, and you hire a public opinion survey group to do some market research for you. The group claims that in its survey of 450 consumers, the following criteria were named as important factors in purchasing bath soap:

Odor	425
Lathering ease	397
Natural ingredients	340
Odor and lathering ease	284
Odor and natural ingredients	315
Lathering ease and natural ingredients	219
All three factors	147

Should you have confidence in these results? Why or why not?

13. a. How many integers n , $1 \leq n \leq 100$, are multiples of either 2 or 5?
 b. How many integers n , $1 \leq n \leq 100$, are not multiples of either 2 or 5?
14. How many integers n , $1 \leq n \leq 1000$, are not multiples of either 3 or 7?
15. a. Write the expression for $|A \cup B \cup C \cup D|$ from Equation (4).
 b. Write an expression for the number of terms in the expansion of $|A_1 \cup \cdots \cup A_n|$ given by Equation (4).
16. Patrons of a local bookstore can sign up for advance notification of new book arrivals in genres of interest. In the first month of this service, 32 sign up for mysteries, 34 for spy novels, 18 for westerns, and 41 for science fiction. Of these, 17 sign for both mysteries and spy novels, 8 for both mysteries and westerns, 19 for mysteries and science fiction, 5 for spy novels and westerns, 20 for spy novels and science fiction, and 12 for westerns and science fiction. In addition, 2 sign up for mysteries, spy novels and westerns, 11 for mysteries, spy novels and science fiction, 6 for mysteries, westerns, and science fiction, and 5 for spy novels, westerns, and science fiction. Finally, 2 people sign up for all four categories. How many people signed up for service in the first month?
17. How many cards must be drawn from a standard 52-card deck to guarantee 2 cards of the same suit?
18. How many cards must be drawn from a standard 52-card deck to guarantee a black card?
19. If 12 cards are drawn from a standard deck, must at least 2 of them be of the same denomination (type)?
20. How many cards must be drawn from a standard 52-card deck to guarantee 2 queens?
21. A computerized dating service has a list of 50 men and 50 women. Names are selected at random; how many names must be chosen to guarantee one name of each gender?
22. A computerized housing service has a list of 50 men and 50 women. Names are selected at random; how many names must be chosen to guarantee two names of the same gender?
23. How many people must be in a group to guarantee that 2 people in the group have the same birthday (don't forget leap year)?
24. In a group of 25 people, must there be at least 3 who were born in the same month?
25. Prove that if four numbers are chosen from the set $\{1, 2, 3, 4, 5, 6\}$, at least one pair must add up to 7. (*Hint*: Find all the pairs of numbers from the set that add to 7.)
26. How many numbers must be selected from the set $\{2, 4, 6, 8, 10, 12, 14, 16, 18, 20\}$ to guarantee that at least one pair adds up to 22? (See the hint for Exercise 25).
27. Let n be a positive number. Show that in any set of $n + 1$ numbers, there are at least two with the same remainder when divided by n .

SECTION 4.4 PERMUTATIONS AND COMBINATIONS

Permutations

Example 26 in Section 4.2 discussed the problem of counting all possibilities for the last four digits of a telephone number with no repeated digits. In this problem, the number 1259 is not the same as the number 2951 because the order of the four digits is important. An ordered arrangement of objects is called a **permutation**. Each of these numbers is a permutation of 4 distinct objects chosen from a set of 10 distinct objects (the digits). How many such permutations are there? The answer, found by using the multiplication principle, is $10 \cdot 9 \cdot 8 \cdot 7$ —there are 10 choices for the first digit, then 9 for the next digit because repetitions are not allowed, 8 for the next digit, and 7 for the fourth digit. The number of permutations of r distinct objects chosen from n distinct objects is denoted by $P(n, r)$. Therefore the solution to the problem of the four-digit number without repeated digits can be expressed as $P(10, 4)$.

A formula for $P(n, r)$ can be written using the factorial function. For a positive integer n , **n factorial** is defined as $n(n - 1)(n - 2) \cdots 1$ and denoted by $n!$; also, $0!$ is defined to have the value 1. From the definition of $n!$, we see that

$$n! = n(n - 1)!$$

and that for $r < n$,

$$\begin{aligned} \frac{n!}{(n - r)!} &= \frac{n(n - 1) \cdots (n - r + 1)(n - r)!}{(n - r)!} \\ &= n(n - 1) \cdots (n - r + 1) \end{aligned}$$

Using the factorial function,

$$\begin{aligned} P(10, 4) &= 10 \cdot 9 \cdot 8 \cdot 7 \\ &= \frac{10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1} = \frac{10!}{6!} = \frac{10!}{(10 - 4)!} \end{aligned}$$

In general, $P(n, r)$ is given by the formula

$$P(n, r) = \frac{n!}{(n - r)!} \text{ for } 0 \leq r \leq n$$

EXAMPLE 45

The value of $P(7, 3)$ is

$$\frac{7!}{(7 - 3)!} = \frac{7!}{4!} = \frac{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{4 \cdot 3 \cdot 2 \cdot 1} = 7 \cdot 6 \cdot 5 = 210$$

EXAMPLE 46

Three somewhat special cases that can arise when computing $P(n, r)$ are the two “boundary conditions” $P(n, 0)$ and $P(n, n)$, and also $P(n, 1)$. According to the formula,

$$P(n, 0) = \frac{n!}{(n-0)!} = \frac{n!}{n!} = 1$$

This formula can be interpreted as saying that there is only one ordered arrangement of zero objects—the empty set.

$$P(n, 1) = \frac{n!}{(n-1)!} = n$$

This formula reflects the fact that there are n ordered arrangements of 1 object. (Each arrangement consists of the 1 object, so this merely counts how many ways to get the 1 object.)

$$P(n, n) = \frac{n!}{(n-n)!} = \frac{n!}{0!} = n!$$

This formula states that there are $n!$ ordered arrangements of n distinct objects, which merely reflects the multiplication principle— n choices for the first object, $n-1$ choices for the second object, and so on, with 1 choice for the n th object. ●

EXAMPLE 47

The number of permutations of 3 objects, say a , b , and c , is given by $P(3, 3) = 3! = 3 \cdot 2 \cdot 1 = 6$. The 6 permutations of a , b , and c are

$$abc, acb, bac, bca, cab, cba$$

EXAMPLE 48

How many three-letter words (not necessarily meaningful) can be formed from the word “compiler” if no letters can be repeated? Here the arrangement of letters matters, and we want to know the number of permutations of 3 distinct objects taken from 8 objects. The answer is $P(8, 3) = 8!/5! = 336$. ●

Note that we could have solved Example 48 just by using the multiplication principle—there are 8 choices for the first letter, 7 for the second, and 6 for the third, so the answer is $8 \cdot 7 \cdot 6 = 336$. $P(n, r)$ simply gives us a new way to think about the problem, as well as a compact notation.

EXAMPLE 49

Ten athletes compete in an Olympic event. Gold, silver, and bronze medals are awarded; in how many ways can the awards be made?

This problem is essentially the same as the one in Example 48. Order matters; given 3 winners A, B, and C, the arrangement A–gold, B–silver, C–bronze is different than the arrangement C–gold, A–silver, B–bronze. So we want the number of ordered arrangements of 3 objects from a pool of 10, or $P(10, 3)$. Using the formula for $P(n, r)$, $P(10, 3) = 10!/7! = 10 \cdot 9 \cdot 8 = 720$. ●

PRACTICE 30 In how many ways can a president and vice-president be selected from a group of 20 people? ■

PRACTICE 31 In how many ways can 6 people be seated in a row of 6 chairs? ■

Counting problems can have other counting problems as subtasks.

EXAMPLE 50

A library has 4 books on operating systems, 7 on programming, and 3 on data structures. Let's see how many ways these books can be arranged on a shelf, given that all books on the same subject must be together.

We can think of this problem as a sequence of subtasks. First we consider the subtask of arranging the 3 subjects. There are $3!$ outcomes to this subtask, that is, $3!$ different orderings of subject matter. The next subtasks are arranging the books on operating systems ($4!$ outcomes), then arranging the books on programming ($7!$ outcomes), and finally arranging the books on data structures ($3!$ outcomes). Thus, by the multiplication principle, the final number of arrangements of all the books is $(3!)(4!)(7!)(3!) = 4,354,560$. ●

Combinations

Sometimes we want to select r objects from a set of n objects, but we don't care how they are arranged. Then we are counting the number of **combinations** of r distinct objects chosen from n distinct objects, denoted by $C(n, r)$. For each such combination, there are $r!$ ways to permute the r chosen objects. By the multiplication principle, the number of permutations of r distinct objects chosen from n objects is the product of the number of ways to choose the objects, $C(n, r)$, multiplied by the number of ways to arrange the objects chosen, $r!$. Thus,

$$C(n, r) \cdot r! = P(n, r)$$

or

$$C(n, r) = \frac{P(n, r)}{r!} = \frac{n!}{r!(n-r)!} \text{ for } 0 \leq r \leq n$$

Other notations for $C(n, r)$ are

$${}_n C_r, \quad C_r^n, \quad \binom{n}{r}$$

EXAMPLE 51

The value of $C(7, 3)$ is

$$\begin{aligned} \frac{7!}{3!(7-3)!} &= \frac{7!}{3!4!} = \frac{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1}{3 \cdot 2 \cdot 1 \cdot 4 \cdot 3 \cdot 2 \cdot 1} \\ &= \frac{7 \cdot 6 \cdot 5}{3 \cdot 2 \cdot 1} = 7 \cdot 5 = 35 \end{aligned}$$

From Example 45, the value of $P(7, 3)$ is 210, and $C(7, 3) \cdot (3!) = 35(6) = 210 = P(7, 3)$. ●

EXAMPLE 52

The special cases for $C(n, r)$ are $C(n, 0)$, $C(n, 1)$, and $C(n, n)$. The formula for $C(n, 0)$,

$$C(n, 0) = \frac{n!}{0!(n-0)!} = 1$$

reflects the fact that there is only one way to choose zero objects from n objects: Choose the empty set.

$$C(n, 1) = \frac{n!}{1!(n-1)!} = n$$

Here the formula indicates that there are n ways to select 1 object from n objects.

$$C(n, n) = \frac{n!}{n!(n-n)!} = 1$$

Here we see that there is only one way to select n objects from n objects, and that is to choose all of the objects. ●

In the formula for $C(n, r)$, suppose n is held fixed and r is increased. Then $r!$ increases, which tends to make $C(n, r)$ smaller, but $(n-r)!$ decreases, which tends to make $C(n, r)$ larger. For small values of r , the increase in $r!$ is not as great as the decrease in $(n-r)!$, and so $C(n, r)$ increases from 1 to n to larger values. At some point, however, the increase in $r!$ overcomes the decrease in $(n-r)!$, and the values of $C(n, r)$ decrease back down to 1 by the time $r = n$, as we calculated in Example 52. Figure 4.9a illustrates the rise and fall of the values of $C(n, r)$ for a fixed n . For $P(n, r)$, as n is held fixed and r is increased, $n-r$ and therefore $(n-r)!$ decreases, so $P(n, r)$ increases. Values of $P(n, r)$ for $0 \leq r \leq n$ thus increase from 1 to n to $n!$, as we calculated in Example 46. See Figure 4.9b; note the difference in the vertical scale of Figures 4.9a and 4.9b.

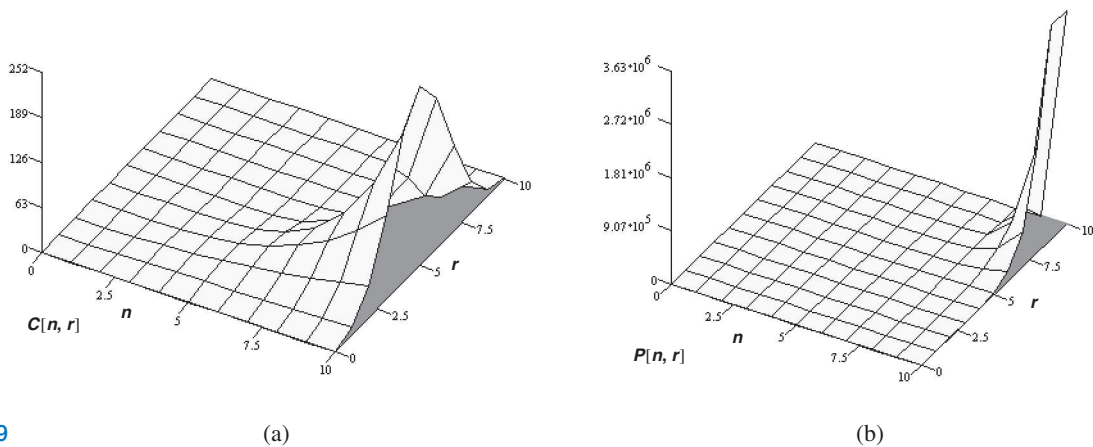


Figure 4.9

EXAMPLE 53

How many 5-card poker hands are possible with a 52-card deck? Here order does not matter because we simply want to know which cards end up in the hand. We want the number of ways to choose 5 objects from a pool of 52, which is a combinations problem. The answer is $C(52, 5) = 52!/(5!47!) = 2,598,960$. ●

Unlike earlier problems, the answer to Example 53 cannot easily be obtained by applying the multiplication principle. Thus, $C(n, r)$ gives us a way to solve new problems.

EXAMPLE 54

Ten athletes compete in an Olympic event; 3 will be declared winners. How many sets of winners are possible?

Here, as opposed to Example 49, there is no order to the 3 winners, so we are simply choosing 3 objects out of 10. This is a combinations problem, not a permutations problem. The result is $C(10, 3) = 10!/(3!7!) = 120$. Notice that there are fewer ways to choose 3 winners (a combinations problem) than to award gold, silver, and bronze medals to 3 winners (a permutations problem—Example 49).

PRACTICE 32

How many committees of 3 are possible from a group of 12 people?

REMINDER

In a counting problem, first ask yourself if order matters. If it does, it's a permutations problem. If not, it's a combinations problem.

Remember that the distinction between permutations and combinations lies in whether the objects are to be merely selected or both selected and ordered. If ordering is important, the problem involves permutations; if ordering is not important, the problem involves combinations. For example, Practice 30 is a permutations problem—2 people are to be selected and ordered, the first as president, the second as vice-president—whereas Practice 32 is a combinations problem—3 people are selected but not ordered.

In solving counting problems, $C(n, r)$ can be used in conjunction with the multiplication principle or the addition principle.

EXAMPLE 55

A committee of 8 students is to be formed from a class consisting of 19 freshmen and 34 sophomores.

- How many committees of 3 freshmen and 5 sophomores are possible?
- How many committees with exactly 1 freshman are possible?
- How many committees with at most 1 freshman are possible?
- How many committees with at least 1 freshman are possible?

Because the ordering of the individuals chosen is not important, these are combinations problems.

For part (a), we have a sequence of two subtasks, selecting freshmen and selecting sophomores. The multiplication principle should be used. (Thinking of a sequence of subtasks may seem to imply ordering, but it just sets up the levels of the decision tree, the basis for the multiplication principle. There is no ordering of the students.) Because there are $C(19, 3)$ ways to choose the freshmen and $C(34, 5)$ ways to choose the sophomores, the answer is

$$C(19, 3) \cdot C(34, 5) = \frac{19!}{3!16!} \cdot \frac{34!}{5!29!} = (969)(278,256)$$

For part (b), we again have a sequence of subtasks: selecting the single freshman and then selecting the rest of the committee from among the sophomores. There are $C(19, 1)$ ways to select the single freshman and $C(34, 7)$ ways to select the remaining 7 members from the sophomores. By the multiplication principle, the answer is

$$C(19, 1) \cdot C(34, 7) = \frac{19!}{1!(19-1)!} \cdot \frac{34!}{7!(34-7)!} = 19(5,379,616)$$

For part (c), we get at most 1 freshman by having exactly 1 freshman or by having 0 freshmen. Because these are disjoint events, we use the addition principle. The number of ways to select exactly 1 freshman is the answer to part (b). The number of ways to select 0 freshmen is the same as the number of ways to select the entire 8-member committee from among the 34 sophomores, $C(34, 8)$. Thus the answer is

$$C(19, 1) \cdot C(34, 7) + C(34, 8) = \text{some big number}$$

We can attack part (d) in several ways. One way is to use the addition principle, thinking of the disjoint possibilities as exactly 1 freshman, exactly 2 freshmen, and so on, up to exactly 8 freshmen. We could compute each of these numbers and then add them. However, it is easier to do the problem by counting all the ways the committee of 8 can be selected from the total pool of 53 people and then eliminating (subtracting) the number of committees with 0 freshmen (all sophomores). Thus the answer is

$$C(53, 8) - C(34, 8)$$

REMINDER

“At least” counting problems are often best solved by subtraction.

The factorial function grows large quickly. A number like $100!$ cannot be computed on most calculators (or on most computers unless double-precision arithmetic is used), but expressions like

$$\frac{100!}{25!75!}$$

can nevertheless be computed by first canceling common factors.

Eliminating Duplicates

We mentioned earlier that counting problems can often be solved in different ways. Unfortunately, it is also easy to find so-called solutions that sound eminently reasonable but are incorrect. Usually they are wrong because they count something more than once (or sometimes they overlook counting something entirely).

EXAMPLE 56

Consider again part (d) of Example 55, the number of committees with at least 1 freshman. A bogus solution to this problem goes as follows: Think of a sequence of two subtasks, choosing a freshman and then choosing the rest of the committee.

There are $C(19, 1)$ ways to choose 1 freshman. Once a freshman has been selected, that guarantees that at least 1 freshman will be on the committee, so we are free to choose the remaining 7 members of the committee from the remaining 52 people without any restrictions, giving us $C(52, 7)$ choices. By the multiplication principle, this gives $C(19, 1) \cdot C(52, 7)$. However, this is a bigger number than the correct answer.

The problem is this: Suppose Derek and Felicia are both freshmen. In one of the choices we have counted, Derek is the one guaranteed freshman, and we pick the rest of the committee in such a way that Felicia is on it along with 6 others. But we have also counted the option of making Felicia the guaranteed freshman and having Derek and the same 6 others be the rest of the committee. This is the same committee as before, and we have counted it twice. ●

PRACTICE 33 A committee of 2 to be chosen from 4 math majors and 3 physics majors must include at least 1 math major. Compute the following 2 values.

- $C(7, 2) - C(3, 2)$ (correct solution: all committees minus those with no math majors)
- $C(4, 1) \cdot C(6, 1)$ (bogus solution: choose 1 math major and then choose the rest of the committee)

The expression $C(4, 1) \cdot C(6, 1) - C(4, 2)$ also gives the correct answer because $C(4, 2)$ is the number of committees with 2 math majors, and these are the committees counted twice in $C(4, 1) \cdot C(6, 1)$. ■

EXAMPLE 57

- How many distinct permutations can be made from the characters in the word FLORIDA?
- How many distinct permutations can be made from the characters in the word MISSISSIPPI?

Part (a) is a simple problem of the number of ordered arrangements of seven distinct objects, which is $7!$. However, the answer to part (b) is not $11!$ because the 11 characters in MISSISSIPPI are not all distinct. This means that $11!$ counts some of the same arrangements more than once (the same arrangement meaning that we cannot tell the difference between $MIS_1S_2ISSIPPI$ and $MIS_2S_1ISSIPPI$.)

Consider any one arrangement of the characters. The four S 's occupy certain positions in the string. Rearranging the S 's within those positions would result in no distinguishable change, so our one arrangement has $4!$ look-alikes. In order to avoid overcounting, we must divide $11!$ by $4!$ to take care of all the ways of moving the S 's around. Similarly, we must divide by $4!$ to take care of the four I 's and by $2!$ to take care of the two P 's. The number of distinct permutations is thus

$$\frac{11!}{4!4!2!}$$

In general, suppose there are n objects of which a set of n_1 are indistinguishable from each other, another set of n_2 are indistinguishable from each other, and so on, down to n_k objects that are indistinguishable from each other. The number of distinct permutations of the n objects is

$$\frac{n!}{(n_1!)(n_2!) \cdots (n_k!)}$$

PRACTICE 34 How many distinct permutations are there of the characters in the word MONGOOSES? ■

Permutations and Combinations with Repetitions

Our formulas for $P(n, r)$ and $C(n, r)$ assume that we arrange or select r objects out of the n available using each object only once. Therefore $r \leq n$. Suppose, however, that the n objects are available for reuse as many times as desired. For example, we construct words using the 26 letters of the alphabet; the words may be as long as desired with letters used repeatedly. Or we may draw cards from a deck, replacing a card after each draw; we may draw as many cards as we like with cards used repeatedly. We can still talk about permutations or combinations of r objects out of n , but with repetitions allowed, r might be greater than n .

Counting the number of permutations of r objects out of n distinct objects with repetition is easy. We have n choices for the first object and, because we can repeat that object, n choices for the second object, n choices for the third, and so on. Hence, the number of permutations of r objects out of n distinct objects with repetition allowed is n^r .

To determine the number of combinations of r objects out of n distinct objects with repetition allowed, we use a rather clever idea.

EXAMPLE 58

A jeweler designing a pin has decided to use five stones chosen from a supply of diamonds, rubies, and emeralds. How many sets of stones are possible?

Because we are not interested in any ordered arrangement of the stones, this is a combinations problem rather than a permutations problem. We want the number of combinations of five objects out of three objects with repetition allowed. The pin might consist of 1 diamond, 3 rubies, and 1 emerald, for instance, or 5 diamonds. We can represent these possibilities by representing the stones chosen by 5 asterisks and placing markers between the asterisks to represent the distribution among the three types of gem, diamonds, rubies, and emeralds. For example, we could represent the choice of 1 diamond, 3 rubies, and 1 emerald by

*|***|*

while the choice of 5 diamonds, 0 rubies, and 0 emeralds would be represented by

*****||

Although we wrote the asterisks and markers in a row, there is no ordering implied. We are just looking at seven slots holding the five gems and the two markers, and the different choices are represented by which of the seven slots are occupied by asterisks. We therefore count the number of ways to choose five items out of seven, which is $C(7, 5)$ or

$$\frac{7!}{5!2!}$$

In general, if we use the same scheme to represent a combination of r objects out of n distinct objects with repetition allowed, there must be $n - 1$ markers to indicate the number of copies of each of the n objects. This gives $r + (n - 1)$ slots to fill, and we want to know the number of ways to select r of these. Therefore we want

$$C(r + n - 1, r) = \frac{(r + n - 1)!}{r!(r + n - 1 - r)!} = \frac{(r + n - 1)!}{r!(n - 1)!}$$

This agrees with the result in Example 58, where $r = 5$, $n = 3$.

PRACTICE 35

Six children get one lollipop each from among a selection of red, yellow, and green lollipops. How many sets of lollipops are possible? (We do not care which child gets which.) ■

We have discussed a number of counting techniques in this chapter. Table 4.2 summarizes the techniques you can apply in various circumstances, although there may be several legitimate ways to solve any one counting problem.

TABLE 4.2	
You Want to Count the Number of ...	Technique to Try
Subsets of an n -element set	Use formula 2^n .
Outcomes of successive events	Multiply the number of outcomes for each event.
Outcomes of disjoint events	Add the number of outcomes for each event.
Outcomes given specific choices at each step	Draw a decision tree and count the number of paths.
Elements in overlapping sections of related sets	Use principle of inclusion and exclusion formula.
Ordered arrangements of r out of n distinct objects	Use $P(n, r)$ formula.
Ways to select r out of n distinct objects	Use $C(n, r)$ formula.
Ways to select r out of n distinct objects with repetition allowed	Use $C(r + n - 1, r)$ formula.

Generating Permutations and Combinations

In a certain county, lottery ticket numbers consist of a sequence (a permutation) of the 9 digits 1, 2, ..., 9. The ticket printing company may or may not know that $9! = 362,880$ distinct ticket numbers are possible, but it certainly needs a way to

generate all possible ticket numbers. Or the county council (a group of 12 members) wants to form a subcommittee of 4 members but wants to pick the combination of council members it feels can best work together. The council could ask someone to generate all $C(12, 4) = 495$ potential subcommittees and examine the membership of each one. We see that in some situations, simply counting the number of permutations or combinations is not enough; it is useful to be able to list all the permutations or combinations.

EXAMPLE 59

Example 47 asked for the number of permutations of the three objects a , b , and c . The answer is given by the formula $P(3,3) = 3! = 6$. However Example 47 went on to list the six permutations:

$$abc, acb, bac, bca, cab, cba$$

This list presents the six permutations using **lexicographical ordering**, that is, the order in which they would be found in a dictionary if they were legitimate words. Thus abc precedes acb because although both words begin with the same first character, for the second character, b precedes c . If we had three integers, say 4, 6, and 7, instead of three alphabetical characters, the lexicographical ordering of all six permutations would present values in increasing numerical order:

$$467, 476, 647, 674, 746, 764$$
PRACTICE 36

Arrange the following list of permutations in lexicographical order:

$$scary, yarsc, scyra, cysar, scrya, yares$$

Words that are close in lexicographical order have the maximum number of matching leftmost characters or, equivalently, differ in the fewest rightmost characters. We use this characteristic to develop a process to generate all permutations of the integers $\{1, \dots, n\}$ in lexicographical order.

EXAMPLE 60

Consider the set $\{1, 2, 3, 4, 5\}$. The smallest numerical value (the first permutation) is given by the increasing order of all the integers, namely,

$$12345$$

To generate the next number in lexicographical order, we want to retain as many of the leftmost digits as possible. Clearly we can't keep the leftmost four digits because this also determines the fifth digit. To keep the leftmost three digits, $123--$, we must be able to rearrange the remaining two digits to represent a larger value than they do now. Reading 12345 from right to left, we find in the last two digits that $4 < 5$, which means we can reverse the 4 and the 5 to get

$$12354$$

which is the next permutation in the list. That's all that can be done with the last two digits; in particular, since 54 is a decreasing sequence, we can't use these two values to generate anything larger.

For the next number, we keep 12 – – – and consider how to arrange the last three digits. Reading 12354 from right to left, we find in the last three digits that $3 < 5$, but we know that everything from 5 to the right is a decreasing sequence. The next permutation should replace 3 with the next largest value to its right. Reading from right to left in the number 12354, the first value larger than 3, in this case 4, is the least value larger than 3. Swapping 3 and 4 gives 12453, which puts 4 in the correct order; the digits to the right are now in descending order, so reversing them gives

12435

which is the next permutation. ●

EXAMPLE 61

To continue Example 60, let's jump ahead. Suppose we have just generated permutation

25431

and we want the next permutation. Reading from right to left, everything increases until we get to 2, where we have $2 < 5$. Starting again from right to left, we stop at the first (and smallest) value greater than 2, which is 3. Swapping 2 and 3 gives 35421, giving the correct first digit. The digits after 3 are in descending order, so reversing them involves swapping 5 and 1, and also swapping 4 and 2, giving the next permutation

31245

From the preceding examples, we can construct an algorithm to generate all permutations of the integers from 1 to n in lexicographical order.

ALGORITHM *PERMUTATION GENERATOR*

```

PermGenerator(integer  $n \geq 2$ )
//generates in lexicographical order all permutations
//of the integers in the set  $\{1, \dots, n\}$ 
Local variables:
integers  $i, j$  //indices of permutation elements
integer  $k$  //for loop counter
integers  $d_1, d_2, \dots, d_n$  //left to right elements of a permutation

```

```

//create and write out smallest permutation
for  $k = 1$  to  $n$  do
     $d_k = k$ 
end for
write  $d_1d_2\dots d_n$ 

//create and write out remaining permutations
for  $k = 2$  to  $n!$  do
    //look right to left for first break in increasing sequence
     $i = n - 1$ 
     $j = n$ 
    while  $d_i > d_j$  do //still increasing right to left
         $i = i - 1$ 
         $j = j - 1$ 
    end while
    //now  $d_i < d_j$ , need to replace  $d_i$  with next largest integer

    //look right to left for smallest value greater than  $d_i$ 
     $j = n$ 
    while  $d_i > d_j$  do
         $j = j - 1$ 
    end while
    //now  $d_j$  is smallest value  $> d_i$ 

    swap  $d_i$  and  $d_j$ 

    //reverse the digits to the right of index  $i$ 
     $i = i + 1$ 
     $j = n$ 
    while  $i < j$  do
        swap  $d_i$  and  $d_j$ 
         $i = i + 1$ 
         $j = j - 1$ 
    end while

    write  $d_1d_2\dots d_n$ 
end for
end function PermGenerator

```

PRACTICE 37

Walk through the steps in the algorithm that generate the next permutation following 51432.

Another algorithm for generating (not in lexicographical order) all permutations of the integers $\{1, \dots, n\}$ is suggested in Exercise 7 of On the Computer at the end of this chapter. Both of these algorithms can also be used to generate all

permutations of any n distinct elements; simply assign each of the n elements a unique integer from 1 to n , generate the permutations of the integers, and then reverse the assignment.

Our second problem is to generate the $C(n, r)$ combinations of r distinct integers chosen from $\{1, \dots, n\}$. Such a combination does not involve order, it is merely a subset of r elements. Nonetheless we will represent the subset $\{3, 5, 7\}$ as the sequence 357, and generate the subsets in lexicographical order. Once we generate 357, we can't also generate 375 or 753 or any of the other permutations of the elements in this set. Each legitimate representation is an increasing sequence.

EXAMPLE 62

Consider the lexicographical ordering of the combinations of 4 integers from $\{1, \dots, 7\}$. If the combination

$$2346$$

has just been generated, then the next combination would be

$$2347$$

obtained by incrementing the last digit of the sequence. However, in 2347, the last digit is already at its maximum allowable value. Moving to the left, the 4 can be bumped up to 5, but then the last digit has to be reduced to its minimum value, which is 6 (one more than 5). Therefore,

$$2356$$

is the next combination. The next two values are

$$2357, 2367$$

at which point both 7 and 6 are at their maximum values. The 3 can be bumped up, but the two digits to its right have to be reset to their lowest possible values. The next few values are

$$2456, 2457, 2467, 2567 \dots$$

Based on the ideas of Example 62, given a combination sequence the algorithm should bump up the rightmost digit that is not at its maximum allowable value. The sequence of digits to the right of the newly incremented digit v should have the values $v + 1$, $v + 2$, and so on. The initial (smallest) combination is $12 \dots r$.

ALGORITHM COMBINATION GENERATOR

```

CombGenerator(integer  $n \geq 2$ , integer  $r \geq 1$ )
//generates in lexicographical order all combinations
//of  $r$  integers from the set  $\{1, \dots, n\}$ 
Local variables:
integers  $i, j$            //indices of combination elements
integer  $k$                //for loop counter
integer  $max$              //maximum allowable value for a digit
integers  $d_1, d_2, \dots, d_r$  //left to right elements of a combination

//create and write out smallest combination
for  $k = 1$  to  $r$  do
     $d_k = k$ 
end for
write  $d_1d_2 \dots d_r$ 

//create and write out remaining combinations
for  $k = 2$  to  $C(n, r)$  do
    //look right to left for first non-max value
     $max = n$ 
     $i = r$ 
    while  $d_i = max$  do //look left
         $i = i - 1$ 
         $max = max - 1$ 
    end while
    //now  $d_i < max$ , need to increment  $d_i$ 

     $d_i = d_i + 1$ 

    //reset values right of  $d_i$ 
    for  $j = i + 1$  to  $r$  do
         $d_j = d_{j-1} + 1$ 
    end for
    write  $d_1d_2 \dots d_r$ 
end for
end function CombGenerator

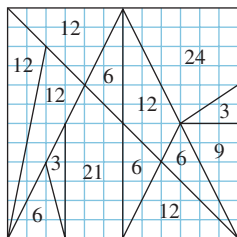
```

PRACTICE 38 Using this algorithm, find the next combination of five items from $\{1, \dots, 9\}$ after 24589. ■

Archimedes and the Stomachion

Archimedes was one of the greatest mathematicians of the ancient world. He lived about 287–212 B.C.E. in Syracuse (a Greek city-state in current-day Sicily). It is thought that he studied for a time in Alexandria (Egypt) with students of Euclid. At any rate, he was greatly interested in geometry, and he thought his most important work was the discovery of a formula for the volume of a sphere. He also approximated the value of π and grasped the ideas of integral calculus by approximating the area under a curve by breaking it up into a series of rectangles. His contributions to mechanics, engineering, physics, and astronomy are also amazing. Some of his inventions helped defend the city of Syracuse from Roman invaders.

Archimedes wrote a treatise about the Stomachion, which is a puzzle apparently known even before Archimedes' time. The puzzle consists of 14 polygons that fill a 12×12 square. It is constructed by marking off unit intervals along the edges of the square and connecting them to create a grid. Each intersection of two grid lines, or of a grid line with the square's border, is called a lattice point. The polygons are formed by lines that connect certain lattice points. The 14 pieces and their areas (each an integer) are shown here, and these areas add up to 144, the area of the square. (A modern result, Pick's theorem, says that the area of such a polygon is given by the formula $A = I + B/2 - 1$, where I = the number of lattice points enclosed within the polygon and B = the number of lattice points on the border of the polygon. You can test this theorem on the diagram here, which is from the Stomachion construction Web page.)



That Archimedes' work on the Stomachion is available today, at least in part, is quite a story. No original writings of Archimedes exist today, but copies were made throughout the ages. One such copy is an ancient parchment (goatskin) manuscript that was written sometime in the 900s. It survived for 300 centuries but in 1229 it was torn apart, washed, folded, and written over as a prayer book (early recycling!). Then it disappeared for hundreds of years, to be discovered in the library of a Greek Orthodox monastery in Constantinople in 1906 by a Danish classics scholar who noted the written-over characters and deciphered what he could, enough to recognize that it was a work by Archimedes. It contained information on the Stomachion. The object of the puzzle, it was thought, was to use the pieces to construct interesting shapes (a bird, an elephant, etc.), much like the better-known tangram puzzle. Pretty trivial stuff to occupy the mind of Archimedes, it seemed.

The parchment prayer book again disappeared after World War I until it was put up for auction by a French family in 1998. Purchased by an anonymous American buyer for \$2 million, it was handed over to scientists for further study. By this time, the parchment was in terrible shape, with missing pages, holes and tears, and covered with mold, to say nothing of the overwriting. Modern technology has revealed more of the original text, which brought about a new interpretation of the Stomachion puzzle.

Dr. Reviel Netz, a mathematics historian at Stanford, has concluded that Archimedes was trying to count how many ways the pieces could be combined to form the original square. This is essentially a combinatorics problem. Is it a simple problem? No, as it turns out. Dr. Netz posed this problem to two husband-and-wife teams of combinatorics experts, who worked six weeks to prove that the number is 17,152. If you ignore issues of symmetry, such as rotations and reflections, the answer is 536, which was confirmed via a computer program written by the computer scientist Dr. William Cutler.

Combinatorics had been considered a modern branch of mathematics, of much interest in computer science but supposedly unknown to the ancient Greeks. Archimedes' Palimpsest (a "palimpsest" is a written-over manuscript) gives combinatorics a much earlier foundation.

"Archimedes," *Encyclopedia Britannica*,
<http://www.britannica.com/EBchecked/topic/32808/Archimedes>
"In Archimedes' Puzzle, a New Eureka Moment," *The New York Times*, December 14, 2003,

<http://www.nytimes.com/2003/12/14/us/in-archimedes-puzzle-a-new-eureka-moment.html?pagewanted=all&src=pm>
Reading Between the Lines, Smithsonian.com,
<http://www.smithsonianmag.com/science-nature/archimedes.html?c=y&page=1>
Stomachion construction,
<http://www.math.nyu.edu/~crrorres/Archimedes/Stomachion/construction.html>
A Tour of Archimedes' Stomachion, Chung, F., and Graham, R.,
<http://www.math.ucsd.edu/~fan/stomach/>

SECTION 4.4 REVIEW

TECHNIQUES

- Find the number of permutations of r distinct objects chosen from n distinct objects.
- Find the number of combinations of r distinct objects chosen from n distinct objects.
- W Use permutations and combinations in conjunction with the multiplication principle and the addition principle.
- Find the number of distinct permutations of n objects that are not all distinct.
- Find the number of permutations of r objects out of n distinct objects when objects may be repeated.
- Find the number of combinations of r objects out of n distinct objects when objects may be repeated.

- Generate all permutations of the integers $\{1, \dots, n\}$ in lexicographical order.
- Generate all combinations of r integers from the set $\{1, \dots, n\}$.

MAIN IDEAS

- There are formulas for counting various permutations and combinations of objects.
- Care must be taken when analyzing a counting problem to avoid counting the same thing more than once or not counting some things at all.
- Algorithms exist to generate all permutations of n objects and all combinations of r out of n objects.

EXERCISES 4.4

1. Compute the value of the following expressions.
 - a. $P(7, 2)$
 - b. $P(8, 5)$
2. Compute the value of the following expressions.
 - a. $P(6, 4)$
 - b. $P(n, n - 1)$
3. How many batting orders are possible for a 9-man baseball team?
4. The 14 teams in the local Little League are listed in the newspaper. How many listings are possible?
5. How many different ways can 10 flavors of ice cream be arranged in an ice cream store display case?
6. How many different ways are there to arrange 6 candidate names on a ballot?
7. How many permutations of the characters in COMPUTER are there? How many of the permutations end in a vowel?
8. In how many ways can 6 people be seated in a circle of 6 chairs? Only relative positions in the circle can be distinguished.
9. In how many ways can first, second, and third prize in a pie-baking contest be given to 15 contestants?
10. a. Stock designations on an exchange are limited to 3 letters. How many different designations are there?
b. How many different designations are there if letters cannot be repeated?
11. In how many different ways can 19 people be seated in a row?
12. In how many different ways can 11 men and 8 women be seated in a row?
13. In how many different ways can 11 men and 8 women be seated in a row if the men all sit together and the women all sit together?
14. In how many different ways can 11 men and 8 women be seated in a row if no 2 women are to sit together?
15. In how many different ways can 11 men and 8 women be seated around a circular table? (Only relative positions in the circle can be distinguished.)
16. In how many different ways can 11 men and 8 women be seated around a circular table if no 2 women are to sit together? (Only relative positions in the circle can be distinguished.)

17. Compute the value of the following expressions.
 - a. $C(10, 7)$
 - b. $C(9, 2)$
 - c. $C(8, 6)$
18. Compute $C(n, n - 1)$. Explain why $C(n, n - 1) = C(n, 1)$.
19. Quality control wants to test 25 microprocessor chips from the 300 manufactured each day. How many different batches of test chips are possible?
20. A soccer team carries 18 players on the roster; 11 players make a team. How many different teams are possible?
21. How many juries of 5 men and 7 women can be formed from a panel of 17 men and 23 women?
22. How many different sets of 4 novels and 3 plays can be created from a collection of 21 novels and 11 plays?

Exercises 23–26 deal with the following situation: Of a company's personnel, 7 people work in design, 14 in manufacturing, 4 in testing, 5 in sales, 2 in accounting, and 3 in marketing. A committee of 6 people is to be formed to meet with upper management.

23. How many committees with 1 member from each department are possible?
24. How many committees with exactly 2 members from manufacturing are possible?
25. How many committees with no representative from accounting and exactly 1 representative from marketing are possible?
26. How many committees with at least 2 representatives from manufacturing are possible?

Exercises 27–32 concern a 5-card hand from a standard 52-card deck. A standard deck has 13 cards from each of 4 suits (clubs, diamonds, hearts, spades). The 13 cards have face value 2 through 10, jack, queen, king, or ace. Each face value is a "kind" of card. The jack, queen, and king are "face cards."

27. How many hands contain 4 queens?
28. How many hands contain all diamonds?
29. How many hands contain 3 spades and 2 hearts?
30. How many hands contain cards from all 4 suits?
31. How many hands consist of all face cards?
32. How many hands contain exactly 2 spades and exactly 2 hearts?

Exercises 33–42 concern 5-card poker hands from a standard 52-card deck.

33. How many hands contain a royal straight flush (that is, the 10, jack, queen, king, ace of one suit)?
34. How many hands contain a straight flush (that is, 5 consecutive cards of the same suit, where aces can be low or high) that is not a royal straight flush (see Exercise 33)?
35. How many hands contain four of a kind (such as 4 jacks plus a fifth card)?
36. How many hands contain a full house (that is, three of a kind plus a pair of another kind)?
37. How many hands contain a flush (that is, 5 cards of the same suit) that is not a straight flush or a royal straight flush (see Exercises 33 and 34)?
38. How many hands contain a straight (that is, 5 consecutive cards, where aces can be low or high) that is not a straight flush or a royal straight flush (see Exercises 33 and 34)?
39. How many hands contain three of a kind (that is, exactly 3 cards of the same kind plus 2 other cards that are not a pair)?
40. How many hands contain 2 pairs (that is, 2 pairs of 2 different kinds plus a fifth card of some third kind)?
41. How many hands contain 1 pair (that is, exactly 2 cards of the same kind)?

42. If the joker is added to the deck and functions as a fifth ace (of any suit), how many hands contain a royal straight flush (that is, the 10, jack, queen, king, ace of one suit)?

For Exercises 43–48, 14 copies of a code module are to be executed in parallel on identical processors organized into two communicating clusters, A and B . Cluster A contains 16 processors and cluster B contains 32 processors.

43. How many different sets of processors can be used?
44. How many different sets of processors can be used if all modules must execute on cluster B ?
45. How many different sets of processors can be used if 8 modules are to be processed on cluster A and 6 on cluster B ?
46. How many different sets of processors can be used if cluster A has 3 failed processors and cluster B has 2 failed processors?
47. How many different sets of processors can be used if exactly 2 modules are to execute on cluster B ?
48. How many different sets of processors can be used if all modules are to be executed either on cluster A or on cluster B ?

For Exercises 49–52, a set of 4 coins is selected from a box containing 5 dimes and 7 quarters.

49. Find the number of sets of 4 coins.
50. Find the number of sets in which 2 coins are dimes and 2 are quarters.
51. Find the number of sets composed of all dimes or all quarters.
52. Find the number of sets with 3 or more quarters.

Exercises 53–56 concern a computer network with 60 switching nodes.

53. The network is designed to withstand the failure of any 2 nodes. In how many ways can such a failure occur?
54. In how many ways can 1 or 2 nodes fail?
55. If 1 node has failed, in how many ways can 7 nodes be selected without encountering the failed node?
56. If 2 nodes have failed, in how many ways can 7 nodes be selected to include exactly 1 failed node?

In Exercises 57–60, a congressional committee of 3 is to be chosen from a set of 5 Democrats, 3 Republicans, and 4 independents.

57. How many committees are possible?
58. How many committees with at least 1 independent are possible?
59. How many committees that do not include both Democrats and Republicans are possible?
60. How many committees with at least 1 Democrat and at least 1 Republican are possible?

In Exercises 61–66, the states of California, Arizona, New Mexico, Utah, and Nevada each send a team of 6 delegates to the Southwestern States annual conference. A subcommittee of 9 is to be formed to discuss water rights.

61. How many committees are possible?
62. How many committees with no delegates from New Mexico are possible?
63. How many committees with exactly 1 delegate from New Mexico are possible?
64. How many committees with at most 1 delegate from New Mexico are possible?

65. How many committees with at least 2 delegates from Nevada are possible?
66. How many committees with at most 4 delegates total from Arizona and California are possible?

In Exercises 67–70, a hostess wishes to invite 6 dinner guests from a list of 14 friends.

67. How many sets of guests are possible?
68. How many sets of guests are possible if 6 friends are boring and 6 friends are interesting, and the hostess wants to have at least 1 of each?
69. How many sets of guests are possible if 2 of the friends dislike each other and neither will come if the other is present?
70. How many sets of guests are possible if 2 of the friends are very fond of each other and one won't come without the other?
71. Twenty-five people, including Simon and Yuan, are candidates to serve on a committee of 5. How many committees that include Simon or Yuan are possible?
72. A student must select 5 classes for the next semester from among 12, but one of the classes must be either American history or English literature. How many sets of classes are possible?
73. How many 5-card hands from a standard 52-card deck contain exactly 4 aces and exactly 1 club?
74. How many 5-card hands from a standard 52-card deck contain exactly 3 jacks and exactly 2 hearts?
75. How many distinct permutations of the characters in ERROR are there? (Remember that the various R's cannot be distinguished from one another.)
76. How many distinct permutations of the characters in LOLLAPALOOZA are there?
77. a. How many distinct permutations of the characters in the word HAWAIIAN are there?
b. How many of the permutations begin with H?
78. a. How many distinct permutations of the characters in the word APALACHICOLA are there?
b. How many of the permutations have both L's together?
79. A bookstore displays a shelf of 5, 3, and 4 copies, respectively, of the top 3 bestsellers. How many distinguishable arrangements of these books are there if books with the same title are not distinguishable?
80. The United Group for Divisive Action uses secret code words that are permutations of 5 characters. You learn that there are only 10 code words. What can you say about repeated characters in the code words?
81. At a dinner party for 5, a tray of 5 servings of appetizers is prepared. An appetizer could be escargots, egg rolls, or nachos. How many different trays could the kitchen produce?
82. A florist has a large number of roses, carnations, lilies, and snapdragons in stock. How many different bouquets of one dozen flowers can be made?
83. A cheese shop carries a large stock of 34 kinds of cheese. By the end of the day, 48 cheese sales have been made, and the items sold must be restocked. How many different restocking orders are possible?
84. One "game package" consists of 12 bingo cards. How many different game packages are there if there are 15 kinds of cards and repetitions are allowed?
85. A hardware shipping order contains 6 items, where each item is either a gallon of paint, a hammer, or a drill.
 - a. How many different shipping orders are possible?
 - b. How many different shipping orders are possible if no paint is shipped?
 - c. How many different shipping orders are possible if each order must contain at least 1 gallon of paint, 1 hammer, and 1 drill?

86. At a birthday party, a mother prepares a plate of cookies for 8 children. There are plenty of chocolate chip, peanut butter, and oatmeal cookies, but each child gets only 1 cookie.
- How many different plates can be prepared?
 - How many different plates can be prepared if at least 1 of each kind of cookie is given out?
 - How many different plates can be prepared if no one likes oatmeal cookies?
 - How many different plates can be prepared if 2 children insist on getting peanut butter?
 - How many different plates can be prepared if the dog got into the kitchen and ate all the chocolate chip cookies except 2?
87. On Halloween, 10 identical apples are distributed to 7 children.
- How many distributions are possible? (*Hint*: One possible distribution is that child 1 gets 3 apples, child 2 gets 0 apples, child 3 gets 2 apples, child 4 gets 0 apples, child 5 and child 6 get 1 apple each, and child 7 gets 3 apples. Although the problem says apples are distributed to children, think of assigning a child's name to each apple; a child's name can go to more than 1 apple.)
 - How many distributions are possible if each child is to receive at least 1 apple?
88. Eight identical antique pie safes are sold at a furniture auction to 3 bidders.
- In how many ways can the pie safes be distributed among the bidders? (See the hint for Exercise 87.)
 - In how many ways can the pie safes be distributed if bidder A gets only 1 pie safe?
89. How many distinct nonnegative integer solutions are there to the equation

$$x_1 + x_2 + x_3 + x_4 = 10$$

where the solution

$$x_1 = 3, x_2 = 1, x_3 = 4, x_4 = 2$$

and the solution

$$x_1 = 4, x_2 = 2, x_3 = 3, x_4 = 1$$

are distinct? (*Hint*: Think of this problem as distributing 10 pennies to 4 children; then see the hint in Exercise 87.)

90. How many distinct nonnegative integer solutions are there to the equation

$$x_1 + x_2 + x_3 = 7$$

in which $x_1 \geq 3$? (See the hint for Exercise 89.)

91. Prove that for $n \geq 1$, $P(n, n) = P(n, n - 1)$. (The proof does not require induction, even though it sounds like a very likely candidate for induction.)
92. Prove that for $n \geq 2$, $P(n, 1) + P(n, 2) = n^2$.
93. Prove that for any n and r with $0 \leq r \leq n$, $C(n, r) = C(n, n - r)$. Explain why this is intuitively true.
94. Prove that for any n and r with $0 \leq r \leq n$, $C(n, 2) = C(r, 2) + C(n - r, 2) + r(n - r)$.
95. Prove the identity

$$C(n, r)C(r, k) = C(r, k)C(n - k, r - k) \text{ for } r \leq n \text{ and } k \leq r$$

Give a combinatorial argument.

96. Prove Vandermonde's identity:

$$C(n + m, r) = \sum_{k=0}^r C(n, k)C(m, r - k)$$

(Hint: use a combinatorial argument, picking r elements from the union of two disjoint sets of size n and m , respectively.)

97. The recurrence relation for the sequence of Catalan numbers is

$$C(0) = 1$$

$$C(1) = 1$$

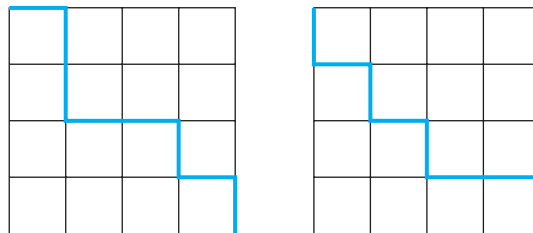
$$C(n) = \sum_{k=1}^n C(k - 1)C(n - k) \quad n \geq 2$$

Although we will not prove it, a closed-form solution to this recurrence relation is

$$C(n) = \frac{1}{n + 1} C(2n, n)$$

(Note that $C(n)$ denotes a value in the Catalan sequence and $C(2n, n)$ denotes the number of combinations of n objects from $2n$ objects.) Compute $C(2)$, $C(3)$, and $C(4)$ using this formula and compare the results with the recurrence relation results (see Exercise 38 of Section 3.1).

98. a. A turtle begins at the upper left corner of an $n \times n$ grid and makes his way to the lower right corner. Along the way, he can move only right or down. The accompanying figure shows two possible paths in a 4×4 grid. How many possible paths can the turtle take?



(Hint: Each path can be described by a sequence of R 's (right moves) and D 's (down moves). Find the number of ways to distribute the R 's in such a sequence.)

b. Relate the answer to part (a) to the sequence of Catalan numbers (see Exercise 97).

99. Arrange the following permutations of the numbers $\{1, \dots, 6\}$ in lexicographical order:

163542, 345621, 643125, 634521, 163452, 356421

100. Arrange the following permutations of the numbers $\{1, \dots, 5\}$ in reverse lexicographical order:

32541, 35142, 53124, 42531, 32154, 42315

In Exercises 101–104, use algorithm permutation generator to generate the next permutation after the given permutation in the set of all permutations of the numbers $\{1, \dots, 7\}$.

101. 7431652

102. 4365127

103. 3675421

104. 2756431

105. In generating all combinations of five items from the set $\{1, \dots, 9\}$, find the next five values in the list after 24579.

106. In generating all combinations of four items from the set $\{1, \dots, 6\}$, find the next five values in the list after 1234.

107. Describe an algorithm to generate all permutations of the integers $\{1, \dots, n\}$ in reverse lexicographical order.

108. Describe an algorithm to generate all permutations of r elements from the set $\{1, \dots, n\}$.

SECTION 4.5 BINOMIAL THEOREM

The expression for squaring a binomial is a familiar one:

$$(a + b)^2 = a^2 + 2ab + b^2$$

This is a particular case of raising a binomial to a nonnegative integer power n . The formula for $(a + b)^n$ involves combinations of n objects. Before we prove this formula, we'll look at a historically interesting array of numbers that suggests a fact we will need in the proof.

Pascal's Triangle

Pascal's triangle is named for the seventeenth-century French mathematician Blaise Pascal (for whom the programming language Pascal was also named), although it was apparently known several centuries earlier. Row n of the triangle ($n \geq 0$) consists of all the values $C(n, r)$ for $0 \leq r \leq n$. Thus the triangle looks like this:

						Row	
			$C(0, 0)$			0	
		$C(1, 0)$	$C(1, 1)$			1	
	$C(2, 0)$	$C(2, 1)$	$C(2, 2)$			2	
	$C(3, 0)$	$C(3, 1)$	$C(3, 2)$	$C(3, 3)$		3	
	$C(4, 0)$	$C(4, 1)$	$C(4, 2)$	$C(4, 3)$	$C(4, 4)$	4	
	$C(5, 0)$	$C(5, 1)$	$C(5, 2)$	$C(5, 3)$	$C(5, 4)$	$C(5, 5)$	5
			\vdots			\vdots	
$C(n, 0)$	$C(n, 1)$...		$C(n, n - 1)$	$C(n, n)$	n

If we compute the numerical values of the expressions, we see that Pascal's triangle has the form

$$\begin{array}{cccccc}
 & & & & & 1 & & & & \\
 & & & & & 1 & & 1 & & \\
 & & & & 1 & 2 & & 1 & & \\
 & & 1 & 3 & 3 & 1 & & & & \\
 & 1 & 4 & 6 & 4 & 1 & & & & \\
 1 & 5 & 10 & 10 & 5 & 1 & & & & \\
 & & & & & & & & & \vdots
 \end{array}$$

Observing this figure, it is clear that the outer edges are all 1s. But it also seems that any element not on the outer edge can be obtained by adding together the two elements directly above it in the preceding row (for example, the first 10 in row five is below the first 4 and the 6 of row four). If this relationship is indeed always true, it means that

$$C(n, k) = C(n - 1, k - 1) + C(n - 1, k) \text{ for } 1 \leq k \leq n - 1 \quad (1)$$

Equation (1) is known as **Pascal's formula**.

To prove Pascal's formula, we begin with the right side:

$$\begin{aligned}
 C(n - 1, k - 1) + C(n - 1, k) &= \frac{(n - 1)!}{(k - 1)![n - 1 - (k - 1)]!} + \frac{(n - 1)!}{k!(n - 1 - k)!} \\
 &= \frac{(n - 1)!}{(k - 1)!(n - k)!} + \frac{(n - 1)!}{k!(n - 1 - k)!} \\
 &= \frac{k(n - 1)!}{k!(n - k)!} + \frac{(n - 1)!(n - k)}{k!(n - k)!}
 \end{aligned}$$

(multiplying the first term by k/k and the second term by $(n - k)/(n - k)$)

$$= \frac{k(n - 1)! + (n - 1)!(n - k)}{k!(n - k)!}$$

(adding fractions)

$$= \frac{(n - 1)![k + (n - k)]}{k!(n - k)!}$$

(factoring the numerator)

$$\begin{aligned}
 &= \frac{(n - 1)!(n)}{k!(n - k)!} \\
 &= \frac{n!}{k!(n - k)!} \\
 &= C(n, k)
 \end{aligned}$$

Another, less algebraic way to prove Pascal's formula involves a counting argument; hence it is called a **combinatorial proof**. We want to compute $C(n, k)$, the number of ways to choose k objects from n objects. There are two disjoint categories of such choices—item 1 is one of the k objects or it is not. If item 1 is one of the k objects, then the remaining $k - 1$ objects must come from the remaining $n - 1$ objects exclusive of item 1, and there are $C(n - 1, k - 1)$ ways for this to happen. If item 1 is not one of the k objects, then all k objects must come from the remaining $n - 1$ objects, and there are $C(n - 1, k)$ ways for this to happen. The total number of outcomes is the sum of the number of outcomes from these two disjoint cases.

Once we have Pascal's formula for our use, we can develop the formula for $(a + b)^n$, known as the **binomial theorem**.

Binomial Theorem and Its Proof

In the expansion of $(a + b)^2$, $a^2 + 2ab + b^2$, the coefficients are 1, 2, and 1, which is row 2 in Pascal's triangle.

PRACTICE 39

Compute the expansion for $(a + b)^3$ and $(a + b)^4$ and compare the coefficients with rows 3 and 4 of Pascal's triangle.

Looking at the coefficients in the expansion of $(a + b)^2$, $(a + b)^3$, and $(a + b)^4$ suggests a general result, which is that the coefficients in the expansion of $(a + b)^n$ look like row n in Pascal's triangle. This is indeed the binomial theorem.

THEOREM BINOMIAL THEOREM

For every nonnegative integer n ,

$$\begin{aligned}(a + b)^n &= C(n, 0)a^n b^0 + C(n, 1)a^{n-1}b^1 + C(n, 2)a^{n-2}b^2 \\ &\quad + \cdots + C(n, k)a^{n-k}b^k + \cdots + C(n, n-1)a^1b^{n-1} + C(n, n)a^0b^n \\ &= \sum_{k=0}^n C(n, k)a^{n-k}b^k\end{aligned}$$

REMINDER

The expansion of $(a + b)^n$ starts with $a^n b^0$. From there the power of a goes down and the power of b goes up, but for each term the powers of a and b add up to n . The coefficients are all of the form $C(n, \text{the power of } b)$.

Because the binomial theorem is stated “for every nonnegative integer n ,” a proof by induction seems appropriate. For the basis step, $n = 0$, the theorem states

$$(a + b)^0 = C(0, 0)a^0 b^0$$

which is

$$1 = 1$$

Since this is certainly true, the basis step is satisfied.

As the inductive hypothesis, we assume that

$$(a + b)^k = C(k, 0)a^k b^0 + C(k, 1)a^{k-1}b^1 + \cdots + C(k, k-1)a^1 b^{k-1} + C(k, k)a^0 b^k$$

Now consider

$$\begin{aligned} (a + b)^{k+1} &= (a + b)^k(a + b) = (a + b)^k a + (a + b)^k b \\ &= [C(k, 0)a^k b^0 + C(k, 1)a^{k-1}b^1 + \cdots + C(k, k-1)a^1 b^{k-1} \\ &\quad + C(k, k)a^0 b^k]a + [C(k, 0)a^k b^0 + C(k, 1)a^{k-1}b^1 \\ &\quad + \cdots + C(k, k-1)a^1 b^{k-1} + C(k, k)a^0 b^k]b \end{aligned}$$

(by the inductive hypothesis)

$$\begin{aligned} &= C(k, 0)a^{k+1}b^0 + C(k, 1)a^k b^1 + \cdots + C(k, k-1)a^2 b^{k-1} \\ &\quad + C(k, k)a^1 b^k + C(k, 0)a^k b^1 + C(k, 1)a^{k-1}b^2 \\ &\quad + \cdots + C(k, k-1)a^1 b^k + C(k, k)a^0 b^{k+1} \\ &= C(k, 0)a^{k+1}b^0 + [C(k, 0) + C(k, 1)]a^k b^1 + [C(k, 1) + C(k, 2)]a^{k-1}b^2 \\ &\quad + \cdots + [C(k, k-1) + C(k, k)]a^1 b^k + C(k, k)a^0 b^{k+1} \end{aligned}$$

(collecting like terms)

$$\begin{aligned} &= C(k, 0)a^{k+1}b^0 + C(k+1, 1)a^k b^1 + C(k+1, 2)a^{k-1}b^2 \\ &\quad + \cdots + C(k+1, k)a^1 b^k + C(k, k)a^0 b^{k+1} \end{aligned}$$

(using Pascal's formula)

$$\begin{aligned} &= C(k+1, 0)a^{k+1}b^0 + C(k+1, 1)a^k b^1 + C(k+1, 2)a^{k-1}b^2 \\ &\quad + \cdots + C(k+1, k)a^1 b^k + C(k+1, k+1)a^0 b^{k+1} \end{aligned}$$

(because $C(k, 0) = 1 = C(k+1, 0)$ and $C(k, k) = 1 = C(k+1, k+1)$)

This completes the inductive proof of the binomial theorem.

The binomial theorem also has a combinatorial proof. Writing $(a + b)^n$ as $(a + b)(a + b) \cdots (a + b)$ (n factors), we know that the answer (using the distributive law of numbers) is the sum of all values obtained by multiplying each term in a factor by a term from every other factor. For example, using b as the term from k factors and a as the term from the remaining $n - k$ factors produces the expression $a^{n-k}b^k$. Using b from a different set of k factors and a from the $n - k$ remaining factors also produces $a^{n-k}b^k$. How many such terms are there? There are $C(n, k)$ different ways to select k factors from which to use b ; hence there are $C(n, k)$ such terms. After adding these terms together, the coefficient of $a^{n-k}b^k$ is $C(n, k)$. As k ranges from 0 to n , the result of summing the terms is the binomial theorem.

Because of its use in the binomial theorem, the expression $C(n, r)$ is also known as a **binomial coefficient**.

Applying the Binomial Theorem

EXAMPLE 63

Using the binomial theorem, we can write the expansion of $(x - 3)^4$. To match the form of the binomial theorem, think of this expression as $(x + (-3))^4$ so that b equals -3 . Remember that a negative number raised to a power is positive for an even power, negative for an odd power. Thus

$$\begin{aligned}(x - 3)^4 &= C(4, 0)x^4(-3)^0 + C(4, 1)x^3(-3)^1 + C(4, 2)x^2(-3)^2 \\ &\quad + C(4, 3)x^1(-3)^3 + C(4, 4)x^0(-3)^4 \\ &= x^4 + 4x^3(-3) + 6x^2(9) + 4x(-27) + 81 \\ &= x^4 - 12x^3 + 54x^2 - 108x + 81\end{aligned}$$

PRACTICE 40

Expand $(x + 1)^5$ using the binomial theorem.

The binomial theorem tells us that term $k + 1$ in the expansion of $(a + b)^n$ is $C(n, k)a^{n-k}b^k$. This allows us to find individual terms in the expansion without computing the entire expression.

PRACTICE 41

What is the fifth term in the expansion of $(x + y)^7$?

By using various values for a and b in the binomial theorem, certain identities can be obtained.

EXAMPLE 64

Let $a = b = 1$ in the binomial theorem. Then

$$(1 + 1)^n = C(n, 0) + C(n, 1) + \cdots + C(n, k) + \cdots + C(n, n)$$



or

$$2^n = C(n, 0) + C(n, 1) + \cdots + C(n, k) + \cdots + C(n, n) \quad (2)$$

This result says that the sum of all the entries in row n of Pascal's triangle equals 2^n . Actually, Equation (2) can be proved on its own using a combinatorial proof. The number $C(n, k)$, the number of ways to select k items from a set of n items, can be thought of as the number of k -element subsets of an n -element set. The right side of Equation (2) therefore represents the total number of all the subsets (of all sizes) of an n -element set. But we already know that the number of such subsets is 2^n .

SECTION 4.5 REVIEW

TECHNIQUE

-  Use the binomial theorem to expand a binomial.
-  Use the binomial theorem to find a particular term in the expansion of a binomial.

MAIN IDEAS

- The binomial theorem provides a formula for expanding a binomial without multiplying it out.
- The coefficients of a binomial raised to a nonnegative integer power are combinations of n items as laid out in row n of Pascal's triangle.

EXERCISES 4.5

1. Expand the expression using the binomial theorem.
 - a. $(a + b)^5$
 - b. $(x + y)^6$
 - c. $(a + 2)^5$
 - d. $(a - 4)^4$
2. Expand the expression using the binomial theorem.
 - a. $(2x + 3y)^3$
 - b. $(3x - 1)^5$
 - c. $(2p - 3q)^4$
 - d. $(3x + \frac{1}{2})^5$

In Exercises 3–10, find the indicated term in the expansion.

3. The fourth term in $(a + b)^{10}$
4. The seventh term in $(x - y)^{12}$
5. The sixth term in $(2x - 3)^9$
6. The fifth term in $(3a + 2b)^7$
7. The last term in $(x - 3y)^8$
8. The last term in $(ab + 3x)^6$
9. The third term in $(4x - 2y)^5$
10. The fourth term in $(3x - \frac{1}{2})^8$
11. Use the binomial theorem (more than once) to expand $(a + b + c)^3$.
12. Expand $(1 + 0.1)^5$ in order to compute $(1.1)^5$.
13. What is the coefficient of x^3y^4 in the expansion of $(2x - y + 5)^8$?
14. What is the coefficient of $x^5y^2z^2$ in the expansion of $(x + y + 2z)^9$?
15. Prove that

$$C(n + 2, r) = C(n, r) + 2C(n, r - 1) + C(n, r - 2) \text{ for } 2 \leq r \leq n$$

(Hint: Use Pascal's formula.)

16. Prove that

$$C(k, k) + C(k + 1, k) + \cdots + C(n, k) = C(n + 1, k + 1) \text{ for } 0 \leq k \leq n$$

(Hint: Use induction on n for a fixed, arbitrary k , as well as Pascal's formula.)

17. Use the binomial theorem to prove that

$$C(n, 0) - C(n, 1) + C(n, 2) - \cdots + (-1)^n C(n, n) = 0$$

18. Use the binomial theorem to prove that

$$C(n, 0) + C(n, 1)2 + C(n, 2)2^2 + \cdots + C(n, n)2^n = 3^n$$

19. Use the binomial theorem to prove that

$$C(n, n) + C(n, n-1)2 + C(n, n-2)2^2 + \cdots + C(n, 1)2^{n-1} + C(n, 0)2^n = 3^n$$

20. Prove the result of Exercise 19 directly from Exercise 18.

21. (Requires calculus)

a. Expand $(1+x)^n$.

b. Differentiate both sides of the equation from part (a) with respect to x to obtain

$$n(1+x)^{n-1} = C(n, 1) + 2C(n, 2)x + 3C(n, 3)x^2 + \cdots + nC(n, n)x^{n-1}$$

c. Prove that

$$C(n, 1) + 2C(n, 2) + 3C(n, 3) + \cdots + nC(n, n) = n2^{n-1}$$

d. Prove that

$$C(n, 1) - 2C(n, 2) + 3C(n, 3) - 4C(n, 4) + \cdots + (-1)^{n-1}nC(n, n) = 0$$

22. (Requires calculus)

a. Prove that

$$\frac{2^{n+1} - 1}{n+1} = C(n, 0) + \frac{1}{2}C(n, 1) + \frac{1}{3}C(n, 2) + \cdots + \frac{1}{n+1}C(n, n)$$

b. Prove that

$$\frac{1}{n+1} = C(n, 0) - \frac{1}{2}C(n, 1) + \frac{1}{3}C(n, 2) + \cdots + (-1)^n \frac{1}{n+1}C(n, n)$$

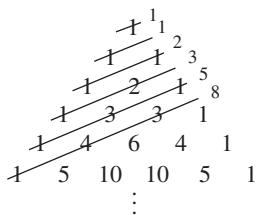
(Hint: Integrate both sides of the equation from part (a) of Exercise 21.)

23. The general form of the principle of inclusion and exclusion is

$$\begin{aligned} |A_1 \cup \cdots \cup A_n| &= \sum_{1 \leq i \leq n} |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| \\ &\quad + \sum_{1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k| \\ &\quad - \cdots + (-1)^{n+1} |A_1 \cap \cdots \cap A_n| \end{aligned} \tag{1}$$

This exercise provides an alternate to the inductive proof given in Section 4.3 of the principle of inclusion and exclusion. Equation (1) is correct if for any x in $\{A_1 \cup \dots \cup A_n\}$, x is counted exactly once by the right side of the equation.

- a. Suppose x is an element of k of the n sets $\{A_1, \dots, A_n\}$. Let B equal the set of A_i s of which x is a member. Then x is counted once in the right side of (1) for each of the intersections that include only sets from B . Show that in the intersections of m sets from $\{A_1, \dots, A_n\}$, $1 \leq m \leq k$, there are $C(k, m)$ that include only sets from B .
 - b. Using the result of part (a), write a sum of terms that represents the number of times x is counted in the right side of (1).
 - c. Use Exercise 17 to show that this sum of terms equals 1.
24. Pascal's triangle has many interesting properties. If you follow diagonal paths through the triangle and sum the values on each path, the result is the Fibonacci sequence (see (a) below). The values along the diagonals are easier to understand if the rows of the triangle are written one row per line with each row beginning one position to the right of the previous row (see (b) below). Then the diagonals are the table columns.



(a)

	0	1	2	3	4	5	6	7	8
0	1								
1		1	1						
2			1	2	1				
3				1	3	3	1		
4					1	4	6	4	1
5						1	5	10	10
6							1	6	15
7								1	7
8									1
	1	1	2	3	5	8	13	21	34

(b)

- a. Prove that the values in column n , $n \geq 2$, read from bottom to top, are given by the expression $\sum_{k=0}^{n/2} C(n - k, k)$ if n is even and by $\sum_{k=0}^{(n-1)/2} C(n - k, k)$ if n is odd.
- b. Prove that the sum of the values in column n , $n \geq 0$, equals $F(n + 1)$.

SECTION 4.6 PROBABILITY

Introduction to Finite Probability

Probability is an extension of the combinatorics (counting) ideas we have already been using. If some action can produce Y different outcomes and X of those Y outcomes are of special interest, we may want to know how likely it is that one of the X outcomes will occur. Probability had its beginnings in gaming or gambling, and we have pretty good intuition for simple cases.

EXAMPLE 65

What is the probability of

- getting “heads” when a coin is tossed?
- getting a 3 with a roll of a die?
- drawing either the ace of clubs or the queen of diamonds from a standard deck of cards?

For (a), tossing a coin results in 2 different possible results, heads or tails, but only the heads result is of interest. The probability of getting heads is “one out of two” or $1/2$. For (b), the roll of a standard 6-sided die has 6 possible results—any of the numbers 1, 2, 3, 4, 5, or 6 might come up. The 3 is exactly one of these possibilities, so the probability of rolling a 3 is $1/6$. For (c), because a standard card deck contains 52 cards, the action of drawing one card has 52 possible results. Two of these are successful results (ace of clubs or queen of diamonds), so the probability of succeeding in this task is $2/52$ or $1/26$. ●

On closer examination, our intuitive answers rely on certain assumptions. We assume that the coin is a “balanced” coin, equally likely to come up heads or tails. We assume that the die is not “loaded” and that the deck of cards is not “stacked.” In other words, we assume that each of the possible outcomes is equally likely; otherwise our probability of success would be different. (Loaded dice and stacked decks are the motivation for many Western movie gunfights.)

The set of all possible outcomes of an action is called the **sample space** S of the action. Any subset of the sample space is called an **event**. If S is a finite set of equally likely outcomes, then the **probability** $P(E)$ of event E is defined to be

$$P(E) = \frac{|E|}{|S|}$$

(Remember that $|A|$ denotes the size of a finite set A .)

EXAMPLE 66

Two coins, A and B , are tossed at the same time; each coin is a fair coin, equally likely to come up heads (H) or tails (T). The sample space of this action is $S = \{HH, HT, TH, TT\}$. Here HT denotes that A comes up heads and B comes up tails, while TH denotes that A comes up tails and B comes up heads; these are two different outcomes. Let the event E be the set $\{HH\}$. The probability of E —that is, the probability of having both coins come up heads²—is

$$P(E) = \frac{|E|}{|S|} = \frac{|\{HH\}|}{|\{HH, HT, TH, TT\}|} = \frac{1}{4} = 0.25$$

PRACTICE 42

Find the probability of drawing an ace from a standard deck of cards. ■

²An event, while technically a set, is also used to describe the action whose outcomes are the members of this set. Here the event E is the set $\{HH\}$, but we also speak of the event as the action of tossing two coins and having both come up heads.

Because events are sets, they can be combined using set operations. Suppose that E_1 and E_2 are two events from the same sample space S . If we are interested in the outcomes in either E_1 or E_2 or both, this will be the event $E_1 \cup E_2$. If we are interested in the outcomes in both E_1 and E_2 , this will be the event $E_1 \cap E_2$. And if we are interested in all the outcomes that are not in E_1 , this will be E_1' .

EXAMPLE 67

Employees from testing, development, and marketing participate in a drawing in which one employee name is chosen. There are 5 employees in testing (2 men and 3 women), 23 in development (16 men and 7 women), and 14 in marketing (6 men and 8 women).

The sample space has 42 names, that is, $|S| = 42$. Let W be the event that a name drawn belongs to a woman. Then $|W| = 3 + 7 + 8 = 18$. Therefore, the probability $P(W)$ that the name drawn belongs to a woman is $|W|/|S| = 18/42 = 3/7$. Let M be the event that a name drawn belongs to someone from marketing. Then $|M| = 14$. Thus, the probability $P(M)$ that the name drawn belongs to someone from marketing is $|M|/|S| = 14/42 = 1/3$. The event that the name drawn belongs to a woman in marketing is $W \cap M$. Because there are 8 women in marketing, $|W \cap M| = 8$, and the probability $P(W \cap M)$ that the name drawn belongs to a woman from marketing is $8/42 = 4/21$. Finally, the event that a name drawn belongs to either a woman or to someone from marketing is $W \cup M$, and $|W \cup M| = 3 + 7 + 14 = 24$. Hence $P(W \cup M) = 24/42 = 4/7$. ●

PRACTICE 43

In Example 67, what is the probability of drawing the name of a male from development? Of drawing a name from testing or development? ■

Probability involves finding the size of sets, either of the sample space or of the event of interest. Therefore many of our previous counting techniques come into play. We may need to use the addition or multiplication principles, the principle of inclusion and exclusion, or the formula for the number of combinations of r things from n objects. (In Example 67, we could have found the size of the union of the women and the marketing people by using the principle of inclusion and exclusion: $|W \cup M| = |W| + |M| - |W \cap M| = 18 + 14 - 8 = 24$.)

EXAMPLE 68

At a party, each card in a standard deck is torn in half and both halves are placed in a box. Two guests each draw a half-card from the box. What is the probability that they draw two halves of the same card?

There are $52 \cdot 2 = 104$ half-cards in the box. The size of the sample space is the number of ways to pick two objects from 104, that is, $|S| = C(104, 2)$. Let H be the event that the halves match. There are 52 ways that the halves can match, so $|H| = 52$. The probability is therefore

$$P(H) = \frac{|H|}{|S|} = \frac{52}{C(104, 2)} = \frac{52}{\frac{104!}{2!102!}} = \frac{52}{\frac{104 \cdot 103}{2}} = \frac{52}{52 \cdot 103} = \frac{1}{103} \cong 0.0097$$
●

Using our definition

$$P(E) = \frac{|E|}{|S|}$$

we can make some observations about probability for any events E_1 and E_2 from a sample space S of equally likely outcomes (Table 4.3). These observations are also called **probability axioms**.

TABLE 4.3		
	Observation	Justification
1.	$0 \leq P(E_1) \leq 1$	$E_1 \subseteq S$ so $0 \leq E_1 $ and $ E_1 \leq S $
2.	The probability of an impossibility is 0	$E_1 = \emptyset$ so $ E_1 = 0$
3.	The probability of a “sure thing” is 1	$E_1 = S$ so $ E_1 = S $
4.	$P(E_1') = 1 - P(E_1)$	$ E_1' = S - E_1 $
5.	$P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2)$	See following discussion
6.	If E_1 and E_2 are disjoint events, then $P(E_1 \cup E_2) = P(E_1) + P(E_2)$	Follows from observation 5

Observation 5 requires a bit of explanation. From the principle of inclusion and exclusion,

$$|E_1 \cup E_2| = |E_1| + |E_2| - |E_1 \cap E_2|$$

So

$$P(E_1 \cup E_2) = \frac{|E_1 \cup E_2|}{|S|} = \frac{|E_1| + |E_2| - |E_1 \cap E_2|}{|S|} = \frac{|E_1|}{|S|} + \frac{|E_2|}{|S|} - \frac{|E_1 \cap E_2|}{|S|} = P(E_1) + P(E_2) - P(E_1 \cap E_2)$$

Probability Distributions

If an action produces outcomes that are not all equally likely, one way to handle the situation is by introducing an appropriate number of repetitions of some of the outcomes.

EXAMPLE 69

Suppose a fair die is rolled. There are 6 possible outcomes, so $|S| = 6$. Let T be the event of rolling a 3; there is only one successful outcome, so $|T| = 1$. Therefore the probability of rolling a 3, just as in Example 65b, is

$$P(T) = \frac{|T|}{|S|} = \frac{1}{6} \cong 0.167$$

The probability of rolling a 4 is the same.

Now suppose the die is loaded so that a 4 comes up three times more often than a 1, 2, 3, 5, or 6. We can describe the set of outcomes for the loaded die by


$$\{1, 2, 3, 4_1, 4_2, 4_3, 5, 6\}$$

The size of the sample space is now $|S| = 8$ and the probability of rolling a 3 is now

$$P(T) = \frac{|T|}{|S|} = \frac{1}{8} = 0.125$$

This is a lower probability than before because the loaded die is not as likely to come up with a 3. However, if F is the event of rolling a 4, then there are three successful outcomes from the sample space. Therefore, the probability of rolling a 4 is now

$$P(F) = \frac{|F|}{|S|} = \frac{3}{8} = 0.375$$

which is higher than before because the loaded die is more likely to come up with a 4. 

Another way to look at problems where not all outcomes are equally likely is to assign a **probability distribution** to the sample space. Rather than artificially enlarging the sample space by creating duplicates of outcomes that occur more frequently, simply consider each distinct outcome in the original sample space as an event and assign it a probability. If there are k different outcomes in the sample space and each outcome x_i is assigned a probability $p(x_i)$, the following rules apply:

1. $0 \leq p(x_i) \leq 1$
2. $\sum_{i=1}^k p(x_i) = 1$

The first equation must hold because any probability value must fall within this range. The second equation must hold from observation 6 in Table 4.3; the union of all of these k disjoint outcomes is the sample space S , and the probability of S is 1.

Now consider some event $E \subseteq S$. The **probability of event E** is then given by

$$P(E) = \sum_{x_i \in E} p(x_i) \tag{1}$$

In other words, we can add up all the probabilities for the individual outcomes in E . This also follows from observation 6 in Table 4.3; E is the union of all its distinct outcomes. The definition of $P(E)$ as $|E|/|S|$ when the outcomes are equally likely is a special case of this definition where $p(x_i) = 1/|S|$ for each x_i in E .

EXAMPLE 70

For the loaded die of Example 69, the appropriate probability distribution is

x_i	1	2	3	4	5	6
$p(x_i)$	1/8	1/8	1/8	3/8	1/8	1/8

As in Example 69, the probability of rolling a 3 is $1/8$ and the probability of rolling a 4 is $3/8$. Let E be the event that a 2 or a 4 is rolled. These are disjoint outcomes, so by Equation (1), $P(E) = p(2) + p(4) = 1/8 + 3/8 = 4/8 = 0.5$. ●

PRACTICE 44

The sample space $S = \{a, b, c\}$. Assume $p(a) = 0.2$ and $p(b) = 0.3$.

- What is $p(c)$?
- What is the probability of getting an outcome of a or c ?

Conditional Probability

A fair coin is tossed twice. The sample space is

$$\{HH, HT, TH, TT\}$$

The probability of getting two tails is clearly $1/4$, but let us belabor this conclusion. Let E_1 be the event that the first toss results in T , so $E_1 = \{TH, TT\}$; let E_2 be the event that the second toss results in T , so $E_2 = \{HT, TT\}$. Then getting two tails is the event $E_1 \cap E_2 = \{TT\}$. The desired probability is

$$P(\text{two tails}) = \frac{|E_1 \cap E_2|}{|S|} = \frac{1}{4}$$

Suppose, however, that we already know that the first toss resulted in T . Does this fact change the probability of getting two tails? Surely so, because we already have half of what we want. The outcome of interest is still $E_1 \cap E_2 = \{TT\}$, but the sample space is now limited to that meeting the condition that E_1 has indeed occurred. That is, because we are assuming that event E_1 has occurred, our sample space now becomes E_1 itself, namely $\{TH, TT\}$. Let $E_2|E_1$ denote the event that E_2 occurs *given that* E_1 has already occurred. Then

$$P(E_2|E_1) = \frac{|E_1 \cap E_2|}{|E_1|} = \frac{1}{2}$$

In terms of probabilities, $P(E_1 \cap E_2) = 1/4$, $P(E_1) = 2/4$, and

$$\frac{P(E_1 \cap E_2)}{P(E_1)} = \frac{1/4}{2/4} = 1/2 = P(E_2|E_1)$$

This suggests the following definition.

● **DEFINITION** **CONDITIONAL PROBABILITY**

Given events E_1 and E_2 , the **conditional probability** of E_2 given E_1 , $P(E_2 | E_1)$, is

$$P(E_2 | E_1) = \frac{P(E_1 \cap E_2)}{P(E_1)}$$

EXAMPLE 71

In a drug study of a group of patients, 17% responded positively to compound A , 34% responded positively to compound B , and 8% responded positively to both. The probability that a patient responded positively to compound B given that he or she responded positively to A is

$$P(B|A) = \frac{P(A \cap B)}{P(A)} = \frac{0.08}{0.17} \cong 0.47$$

PRACTICE 45 In the problem of tossing a fair coin twice, what is the probability of getting two heads given that at least one of the tosses results in heads? (*Hint*: Let E_2 be the event of two heads and E_1 be the event of at least one head.)

If $P(E_2 | E_1) = P(E_2)$, then E_2 is just as likely to happen whether E_1 happens or not. In this case E_1 and E_2 are said to be **independent events** and we have

$$P(E_2 | E_1) = \frac{P(E_1 \cap E_2)}{P(E_1)} = P(E_2)$$

or

$$P(E_1 \cap E_2) = P(E_1) \cdot P(E_2) \quad (2)$$

Equation (2) can be extended to any finite number of independent events and can also be used to test whether events are independent.

EXAMPLE 72

The events of tossing a coin and coming up heads one time (E_1) and heads the next (E_2) are independent events because

$$\begin{aligned} P(E_1 \cap E_2) &= 1/4 \\ P(E_1) &= 1/2, P(E_2) = 1/2 \end{aligned}$$

so Equation (2) is satisfied. If we toss a fair coin repeatedly and the coin lands 4 or 5 or 6 times in a row heads up, we may feel that tails is “due to come up,” in other words, that at the next toss the coin has a better than 50% probability of coming up tails, but in fact that’s not the case. It is true that the probability of getting longer and longer runs of heads decreases from $1/4$ (two heads) to $1/8$ (three heads) to $1/16$ (four heads), and so forth, yet on each successive toss, the probability of getting a head is still $1/2$.

REMINDER

$P(E_1 \cup E_2) = P(E_1) + P(E_2)$ only when E_1 and E_2 are disjoint events. $P(E_1 \cap E_2) = P(E_1) \cdot P(E_2)$ only when E_1 and E_2 are independent events.

At this point we have what might be called an addition rule and a multiplication rule for probability, loosely related to the addition principle and the multiplication principle for counting. If we want the probability of event E_1 or event E_2 , that is, $P(E_1 \cup E_2)$, we can add the respective probabilities—but only if the events are disjoint. If we want the probability of event E_1 and event E_2 , that is, $P(E_1 \cap E_2)$, we can multiply the respective probabilities—but only if the events are independent.

Bayes' Theorem

Bayes' theorem allows us to squeeze an additional probability out of a certain set of known probabilities. Before we state the theorem, let's look at an example.

EXAMPLE 73

A grocery store receives an order from Supplier A that consists of 57% lettuce and 43% spinach. It also receives an order from Supplier B that consists of 39% lettuce and 61% spinach. Before the orders are unloaded, a clerk randomly selects an order box and pulls out a package of produce to show to the produce manager. Later the grocer is notified that spinach from Supplier B is contaminated. If the clerk pulled out a package of spinach, what is the probability that it came from Supplier B ?

Let E_1 be the event that the package came from Supplier A and E_2 be the event that the package came from Supplier B . Let F be the event that the package was spinach. The sample space looks something like

$\{AL1, AL2, AL3, \dots, AS1, AS2, AS3, \dots, BL1, BL2, BL3, \dots, BS1, BS2, BS3, \dots\}$

E_1 and E_2 are disjoint events and $E_1 \cup E_2 = S$.

We know that

$$P(E_1) = 1/2 \quad (\text{equally likely that either box was chosen})$$

$$P(E_2) = 1/2$$

$$P(F|E_1) = 43/100 \quad (\text{percentage of spinach in Supplier } A \text{ order})$$

$$P(F|E_2) = 61/100 \quad (\text{percentage of spinach in Supplier } B \text{ order})$$

and we want

$$P(E_2|F) \quad (\text{probability the package came from Supplier } B \text{ given that it was spinach})$$

Although we know that the probability of the clerk choosing the Supplier B order box is 0.5, we suspect that $P(E_2|F)$ is greater than 0.5 because the item was spinach and Supplier B has a higher percentage of spinach than Supplier A . It turns out that we can compute this probability by sufficient fiddling with the probabilities we do have.

From the definition of conditional probability,

$$P(E_2|F) = \frac{P(F \cap E_2)}{P(F)} \text{ or } P(F \cap E_2) = P(E_2|F)P(F)$$

$$P(F|E_2) = \frac{P(E_2 \cap F)}{P(E_2)} \text{ or } P(E_2 \cap F) = P(F|E_2)P(E_2)$$

Because $F \cap E_2 = E_2 \cap F$, $P(F \cap E_2) = P(E_2 \cap F)$ and therefore

$$P(F \cap E_2) = P(F|E_2)P(E_2) \text{ [and a similar equation for } P(F \cap E_1)\text{]}$$

and

$$P(E_2|F) = \frac{P(F \cap E_2)}{P(F)} = \frac{P(F|E_2)P(E_2)}{P(F)}$$

F is another event in (that is, subset of) S , and

$$F = F \cap S = F \cap (E_1 \cup E_2) = (F \cap E_1) \cup (F \cap E_2)$$

so F is the union of disjoint events, and

$$\begin{aligned} P(F) &= P(F \cap E_1) + P(F \cap E_2) = P(F|E_1)P(E_1) + P(F|E_2)P(E_2) \\ &= \frac{43}{100} \cdot \frac{1}{2} + \frac{61}{100} \cdot \frac{1}{2} = \frac{43 + 61}{200} = \frac{104}{200} \end{aligned}$$

Finally,

$$P(E_2|F) = \frac{P(F|E_2)P(E_2)}{P(F)} = \frac{(61/100)(1/2)}{104/200} = \frac{61/200}{104/200} = 61/104 \cong 0.587$$

As we suspected, the probability that the spinach came from the contaminated batch is greater than 0.5. ●

The general statement of Bayes' theorem (see Exercise 89) follows.

● **THEOREM BAYES' THEOREM**

Let E_1, \dots, E_n be disjoint events from a sample space S whose union equals S . If F is another event from S , then the probability of event E_i , $1 \leq i \leq n$, given event F , is

$$P(E_i|F) = \frac{P(F|E_i)P(E_i)}{\sum_{k=1}^n P(F|E_k)P(E_k)}$$

PRACTICE 46

In Example 73, what is the probability that the package came from Supplier A if it was lettuce? ■

Expected Value

A student takes three tests; the set of grades received is $S = \{g_1, g_2, g_3\}$. The student computes the average test grade $A(g)$ by

$$A(g) = \frac{g_1 + g_2 + g_3}{3}$$

This assumes that the three tests are equally weighted. If we write

$$A(g) = \frac{1}{3}(g_1 + g_2 + g_3) = g_1\left(\frac{1}{3}\right) + g_2\left(\frac{1}{3}\right) + g_3\left(\frac{1}{3}\right)$$

we can also see that $A(g)$ is the sum of the product of each test grade times the amount of its contribution to the total grade. If the last test counts twice as much as the other two, then the “weighted average” grade would be

$$A(g) = \frac{1}{4}(g_1 + g_2 + 2g_3) = g_1\left(\frac{1}{4}\right) + g_2\left(\frac{1}{4}\right) + g_3\left(\frac{2}{4}\right)$$

If we consider S as the sample space and assign the probability distribution

x_i	g_1	g_2	g_3
$p(x_i)$	1/4	1/4	2/4

then

$$A(g) = \sum_{i=1}^3 g_i p(g_i) \tag{3}$$

We want to take the weighted average idea of Equation (3) and make it a bit more general. For the test grades, the sample space S consisted of numerical values. If the values in the sample space are not numerical, we may find a function X that associates a numerical value (a real number) with each element in the sample space. Such a function is called a **random variable**.³ Given a sample space $S = \{x_1, x_2, \dots, x_n\}$ to which a random variable X and a probability distribution p have been assigned, the **expected value** or **weighted average** of the random variable is

$$E(X) = \sum_{i=1}^n X(x_i)p(x_i)$$

³The term “random variable” is a misnomer because X is neither random nor a variable—it’s a function that associates with each value x_i in S a real number $X(x_i)$

EXAMPLE 74

A fair coin is tossed three times. The sample space S is

$$S = \{HHH, HHT, HTH, HTT, THH, THT, TTH, TTT\}$$

Let the random variable X assign to each outcome in S the number of heads in that outcome, which will be an integer value between 0 and 3. Because this is a fair coin, each member of S occurs with equal probability, which determines the probability distribution. Hence we can write

x_i	HHH	HHT	HTH	HTT	THH	THT	TTH	TTT
$X(x_i)$	3	2	2	1	2	1	1	0
$p(x_i)$	1/8	1/8	1/8	1/8	1/8	1/8	1/8	1/8

The expected value of X , that is, the expected number of heads in three tosses, is

$$\begin{aligned} E(X) &= \sum_{i=1}^8 X(x_i)p(x_i) \\ &= 3(1/8) + 2(1/8) + 2(1/8) + 1(1/8) + 2(1/8) + 1(1/8) + 1(1/8) + 0(1/8) \\ &= 12(1/8) = 3/2 = 1.5 \end{aligned}$$

This seems intuitively correct; because the coin is fair, we would expect to get heads about half the time, or 1.5 times out of 3. (Of course, we really can't get half a head, but if we get heads about half the time, then we expect to get 4 heads out of 8 tosses, or 64 heads out of 128 tosses, and so forth.) Note how the expected value is a "predictor" of future outcomes.

Now suppose the coin is weighted in such a way that it is three times more likely to come up heads. In other words, the probability of a head is $3/4$ while the probability of a tail is $1/4$. The elements in the sample space are no longer equally likely, but we can compute their probability distribution. We know that the successive tosses are independent events, so the probability of each outcome in S can be obtained by multiplying the probability of each toss. The probability of HTT, for example, is

$$\left(\frac{3}{4}\right)\left(\frac{1}{4}\right)\left(\frac{1}{4}\right) = \frac{3}{64}$$

The new table looks like

x_i	HHH	HHT	HTH	HTT	THH	THT	TTH	TTT
$X(x_i)$	3	2	2	1	2	1	1	0
$p(x_i)$	27/64	9/64	9/64	3/64	9/64	3/64	3/64	1/64

and the new expected value for X is

$$\begin{aligned}
 E(X) &= \sum_{i=1}^8 X(x_i)p(x_i) \\
 &= 3(27/64) + 2(9/64) + 2(9/64) + 1(3/64) + 2(9/64) + 1(3/64) + 1(3/64) + 0(1/64) \\
 &= 144/64 = 2.25
 \end{aligned}$$

The expected number of heads is now higher because the coin is much more likely than before to come up heads. ●

PRACTICE 47

Given the following table for a sample space, a random variable X , and a probability distribution p , find the expected value for X .

x_i	x_1	x_2	x_3	x_4
$X(x_i)$	5	2	3	7
$p(x_i)$	2/8	3/8	2/8	1/8

Expected value has a property called **linearity**. If X_1 and X_2 are two random variables on the same sample space S and a and b are real numbers, then

$$E(X_1 + X_2) = E(X_1) + E(X_2) \quad (4)$$

$$E(aX_1 + b) = aE(X_1) + b \quad (5)$$

Keep in mind that for any x_i in S , $X_1(x_i)$ and $X_2(x_i)$ are both numerical values that can be added, so the random variable $X_1 + X_2$ just means that $(X_1 + X_2)(x_i) = X_1(x_i) + X_2(x_i)$. Similarly, if a and b are real numbers, then $aX_1 + b$ just means that $(aX_1 + b)(x_i) = aX_1(x_i) + b$. Then Equation (4) is true because

$$\begin{aligned}
 E(X_1 + X_2) &= \sum_{i=1}^n (X_1 + X_2)(x_i)p(x_i) = \sum_{i=1}^n (X_1(x_i) + X_2(x_i))p(x_i) \\
 &= \sum_{i=1}^n X_1(x_i)p(x_i) + \sum_{i=1}^n X_2(x_i)p(x_i) = E(X_1) + E(X_2)
 \end{aligned}$$

Equation (4) extends to any finite sum of random variables. Equation (5) is true because (note that $\sum_{i=1}^n p(x_i) = 1$)

$$\begin{aligned}
 E(aX_1 + b) &= \sum_{i=1}^n (aX_1 + b)(x_i)p(x_i) = \sum_{i=1}^n (aX_1(x_i) + b)p(x_i) \\
 &= a \sum_{i=1}^n X_1(x_i)p(x_i) + b \sum_{i=1}^n p(x_i) = aE(X_1) + b(1)
 \end{aligned}$$

Use of linearity can sometimes simplify the calculation of an expected value (see Exercise 94).

Binomial Distributions

Consider an event that has only two possible outcomes, success or failure. The probability of success is p and the probability of failure $q = 1 - p$. Our well-known coin toss would fit this description because there are only two possible outcomes. Such an event is called a **Bernoulli trial** or **Bernoulli experiment** after the eighteenth-century Swiss mathematician Jacob (or James) Bernoulli. However, a single Bernoulli trial is not of much interest; instead we want to talk about a finite series of Bernoulli trials, each of which has the same probability p of success (hence the same probability $q = 1 - p$ of failure). Because the probability of success does not vary, the various trials are mutually independent events.

Without specifically describing the sample space S , we can define a random variable X as the number of successful outcomes that occur in the n trials. X can range from 0 (no successful outcomes) to n (all successful outcomes). We can determine the probability of k successful outcomes out of n trials as follows:

Assume the k successes (and $(n - k)$ failures) occur in some specific pattern. For example if $k = 3$ and $n = 4$, one pattern would look like $S-F-S-S$. The probability of a success is p , the probability of a failure is q . Because the n trials are mutually independent, we can multiply their probabilities, giving $p^k q^{n-k}$. But this is only one pattern of k successes; how many patterns are there? Exactly the number of ways we can select k out of n items, $C(n, k)$. So the probability of k successful outcomes out of n trials is

$$P(k) = C(n, k)p^k q^{n-k}$$

We have therefore determined a probability distribution for the various values of X , as shown in this table:

$X = k$	0	1	2	...	k	...	n
$P(k)$	$C(n, 0)p^0 q^n$	$C(n, 1)pq^{n-1}$	$C(n, 2)p^2 q^{n-2}$		$C(n, k)p^k q^{n-k}$		$C(n, n)p^n q^0$

Look at the values in this probability distribution. They are the terms in the expansion of $(q + p)^n$. Hence this is called a **binomial distribution**.

Notice that because $q + p = 1$, the sum of the probability distribution terms equals $1^n = 1$. This makes sense because the various values of X are all disjoint so the probability of their union is the sum of their individual probabilities. But the union covers all possible outcomes, so its probability is 1.

EXAMPLE 75

A fair coin is tossed three times, with heads being considered a success, tails being considered a failure. Here $n = 3$ and $p = q = 1/2$. The binomial distribution is

k	0	1	2	3
$P(k)$	$(1/2)^3 = 1/8$	$3(1/2)(1/2)^2 = 3/8$	$3(1/2)^2(1/2) = 3/8$	$(1/2)^3 = 1/8$

Note that this table contains the same information as the table in Example 74, except that here we don't care about the order in which the heads occur.

To compute $E(X)$,

$$E(X) = 0(1/8) + 1(3/8) + 2(3/8) + 3(1/8) = 12/8 = 1.5$$

again agreeing with the result in Example 74. ●

PRACTICE 48 From the expected value in Example 75, we “expect” to get 100 heads if we toss a fair coin 200 times. Find the actual probability of getting 100 heads out of 200 trials. ■

Average Case Analysis of Algorithms

Previously, we have done primarily worst-case analysis of algorithms (Section 3.3). Expected value may help give an average case analysis of an algorithm, i.e., tell us the expected “average” amount of work performed by an algorithm. As in any algorithm analysis, the first step is to identify a suitable “unit of work” based on the nature of the algorithm. Then let the sample space S be the set of all possible inputs to the algorithm. We’ll assume that S is finite; while there may be an infinite number of distinct input values, we can group together those with the same work unit characteristics. Let the random variable X assign to each member of S the number of work units required to execute the algorithm on that input. And let p be a probability distribution on S (here is where we make assumptions about what constitutes “average” input). Then

$$E(X) = \sum_{i=1}^n X(x_i)p(x_i)$$

gives the expected number of work units.

EXAMPLE 76

Consider the sequential search algorithm (Section 3.3). The work unit is the number of comparisons of a target element x against the n elements in a list. Assume that the target element is in the list and is equally likely to be any of the n elements of the list (see Exercise 35 of Section 3.3). This assumption gives the table

x_i	L_1	L_2	\cdots	L_n
$X(x_i)$	1	2	\cdots	n
$p(x_i)$	$1/n$	$1/n$	\cdots	$1/n$

Then

$$\begin{aligned} E(X) &= \sum_{i=1}^n X(x_i)p(x_i) \\ &= \sum_{i=1}^n i \left(\frac{1}{n} \right) = \frac{1}{n} \sum_{i=1}^n i = \frac{1}{n} (1 + 2 + \cdots + n) = \frac{1}{n} \frac{n(n+1)}{2} = \frac{n+1}{2} \end{aligned}$$

The average number of comparisons to find a target in the list, with a uniform probability distribution, is a little more than half the length of the list. ●

SECTION 4.6 REVIEW

TECHNIQUES

- W Compute the probability of an event when all outcomes of an action are equally likely.
 - Compute the probability of an event when a probability distribution has been assigned to the sample space.
- W Compute the conditional probability of an event given that another event has already occurred.
 - Determine whether two events are independent.
 - Given a sample space with a random variable x and a probability distribution, compute the expected value $E(X)$.

MAIN IDEAS

- An event is a subset of the set of all possible outcomes of some action.
- For equally likely outcomes, the probability of an event is the ratio of the number of outcomes in the event to the number of all possible outcomes.
- In the conditional probability of event E_2 given that event E_1 has already taken place, the sample space is reduced to E_1 .
- Events E_1 and E_2 are independent if and only if the conditional probability of E_2 given E_1 is the same as the probability of E_2 .
- In the simplest form of Bayes' theorem, the probability of event A given event B can be computed from the probability of A , the probability of B , and the probability of B given A .
- Given a sample space to which a random variable and a probability distribution have been assigned, the expected value of the random variable is a predictor of its future value.
- An average case analysis of an algorithm is the expected value of work units over the sample space of all inputs; the probability distribution reflects the assumptions being made about "average" input.

EXERCISES 4.6

Exercises 1–6 concern three coins tossed at the same time, each equally likely to come up heads or tails.

1. What is the size of the sample space?
2. What is the probability of getting 1 head and 2 tails?
3. What is the probability of getting all tails?
4. What is the probability that no coin comes up heads?
5. What is the probability of getting all tails or all heads?
6. What is the probability of getting all tails and all heads?

In Exercises 7–14, a pair of fair dice is rolled.

7. What is the size of the sample space?
8. What is the probability of getting "snake eyes" (two 1s)?
9. What is the probability of getting doubles (the same number on each die)?
10. What is the probability of getting a 1 on at least one die?
11. What is the probability of getting a total of 7 on the two dice?
12. What is the probability of getting two consecutive values, such as 3–4, on the two dice?
13. What is the probability of getting a total on the two dice greater than 10?
14. What is the probability of getting a total on the two dice that is an odd number?

Exercises 15–18 concern 3 people participating in a race, with each participant equally likely to either finish the race or drop out of the race.

15. What is the size of the sample space?
16. What is the probability of exactly 1 participant finishing the race?
17. What is the probability of no one finishing the race?
18. What is the probability of at least 2 of the participants finishing the race?

Exercises 19–24 concern a single card drawn from a standard 52-card deck. A standard deck has 13 cards from each of 4 suits (clubs, diamonds, hearts, spades). The 13 cards have face value 2 through 10, jack, queen, king, or ace. Each face value is a “kind” of card. The jack, queen, and king are “face cards.”

19. What is the probability of drawing a diamond?
20. What is the probability of drawing a queen?
21. What is the probability of drawing the queen of diamonds?
22. What is the probability of drawing a queen or a diamond?
23. What is the probability of drawing a black card?
24. What is the probability of drawing a card with a value less than 4 (aces count low)?

Exercises 25–36 concern 2-card hands from a standard 52-card deck. A standard deck has 13 cards from each of 4 suits (clubs, diamonds, hearts, spades). The 13 cards have face value 2 through 10, jack, queen, king, or ace. Each face value is a “kind” of card. The jack, queen, and king are “face cards.”

25. What is the size of the sample space?
26. What is the probability that both cards are the same suit?
27. What is the probability that neither card is a spade?
28. What is the probability that both cards are spades?
29. What is the probability that exactly 1 card is a spade?
30. What is the probability that at least 1 card is a spade?
31. What is the probability that both cards are face cards?
32. What is the probability that exactly 1 card is a face card?
33. What is the probability that both cards are spade face cards?
34. What is the probability that both cards are either face cards or spades?
35. How does the answer to Exercise 30 relate to the answers for Exercises 28 and 29?
36. How does the answer to Exercise 30 relate to the answer for Exercise 27?

Exercises 37–40 concern possible games in the Hoosier lottery, the Indiana state lottery.

37. In the Daily 3 game, three numbers between 0 and 9 will be drawn in succession (repetitions allowed). The player marks three numbers on a game card and has a choice of how to play, straight (the player’s numbers will match the three numbers drawn in exact order) or box (the player’s numbers will match the three drawn in any order).
 - a. What is the size of the sample space?
 - b. What is the probability of a straight?
 - c. What is the probability of a box if 3 distinct numbers are drawn?
 - d. What is the probability of a box if 2 of the numbers drawn are the same?

38. In the Daily4 game, four numbers between 0 and 9 will be drawn in succession (repetitions allowed). The player marks four numbers on a game card and has a choice of how to play, straight (the player's numbers will match the four numbers drawn in exact order) or box (the player's numbers will match the four drawn in any order).
- What is the size of the sample space?
 - What is the probability of a straight?
 - What is the probability of a box if four distinct numbers are drawn?
 - What is the probability of a box if two of the numbers drawn are the same?
 - What is the probability of a box if two distinct pairs of numbers are drawn?
 - What is the probability of a box if three of the numbers drawn are the same?
39. In the Cash5 game, five different numbers between 1 and 39 will be drawn in succession. The player marks five different numbers between 1 and 39 on a game card.
- What is the size of the sample space?
 - What is the probability of matching all five numbers in any order?
 - What is the probability of matching exactly four of the five numbers in any order?
 - What is the probability of matching exactly three of the five numbers in any order?
40. In the Powerball game, five different numbers between 1 and 59 will be drawn in succession, and then one number (the Powerball number) between 1 and 35 will be drawn. The player marks five different numbers between 1 and 59 and one number between 1 and 35 on a game card.
- What is the size of the sample space?
 - What is the probability of matching all five numbers in any order plus matching the Powerball number?
 - What is the probability of matching none of the five numbers but matching the Powerball number?

Exercises 41–50 concern 5-card poker hands from a standard 52-card deck. A standard deck has 13 cards from each of 4 suits (clubs, diamonds, hearts, spades). The 13 cards have face value 2 through 10, jack, queen, king, or ace. Each face value is a “kind” of card. The jack, queen, and king are “face cards.” (See Exercises 33–41 of Section 4.4. for the definitions of terms.)

- What is the probability of a royal straight flush?
- What is the probability of a straight flush that is not a royal straight flush?
- What is the probability of four of a kind?
- What is the probability of a full house?
- What is the probability of a flush that is not a straight flush or a royal straight flush?
- What is the probability of a straight that is not a straight flush or a royal straight flush?
- What is the probability of three of a kind?
- What is the probability of two pairs?
- What is the probability of a pair?
- Explain why, in poker,
 - a straight flush beats a full house.
 - four of a kind beats a straight.

Exercises 51–56 relate to the well-known “birthday problem.” Consider a room containing n persons, each of whom has a birthday equally likely to fall on day 1–365 of the year (ignoring leap years).

51. What is the size of the sample space of all possible assignments of birthdays to people?
52. Let E be the event that no 2 persons in the room have the same birthday. Find an expression for $P(E)$.
53. Let B be the event that 2 or more persons in the room share the same birthday. Find an expression for $P(B)$.
54. Let C be the event that exactly 2 persons in the room share the same birthday. Find an expression for $P(C)$.
55. Which has the larger probability, B or C , and why?
56. Use a spreadsheet or calculator to determine that 23 is the minimum number of people in the room to give a probability of least $1/2$ that 2 or more people share the same birthday.

Exercises 57–62 refer to the gambling game of roulette. A roulette wheel contains 18 black slots and 18 red slots that are numbered (not sequentially) 1–36. There are two green slots numbered 0 and 00. The dealer spins the wheel and spins a ball into the wheel in the opposite direction. Bets are made on which slot the ball will fall into as the wheel slows to a stop. The ball is equally likely to fall into any slot.

57. What is the size of the sample space?
58. What is the probability that the ball lands in a red slot (a “red” bet)?
59. What is the probability that the ball lands in a specific numbered slot (a “straight up” bet)?
60. What is the probability that the ball lands on one of three specific numbers (a “street” bet)?
61. What is the probability that the ball lands on one of four specific numbers (a “corner” bet)?
62. What is the probability that the ball lands on an odd number three times in a row?
63. A woman in Lake Havasu City, Arizona, gave birth to twin girls on September 22, 2006. This happened to be the same date as the woman’s birthday—and the same date as her mother’s birthday. What is the probability that three generations share the same birthday?
64. The NCAA men’s Division I college basketball championship (known as “March Madness”) is a single-elimination tournament held in March of each year. The tournament begins with 68 teams. The structure of the tournament is as follows; the loser of each game is eliminated:

Name of the Round	Number of Teams	Number of Games
The First Four	8	4
Round of 64	64	32
Round of 32	32	16
Sweet Sixteen	16	8
Elite Eight	8	4
Final Four	4	2
Championship Game	2	1

What is the probability of correctly picking the championship team? (Assume that all teams are equally likely to win their respective games.)

65. Bridge is a card game played by four players with a standard 52-card deck. All 52 cards are dealt to the four players at the beginning of each hand. In 1963, women playing bridge in Kankakee, Illinois, discovered that after the deal, each player held the 13 cards of a single suit. What is the probability of one player being dealt an entire suit?
66. Referring to Exercise 65, what is the probability of each of the four players being dealt an entire suit?
67. E_1 and E_2 are events from the same sample space; $P(E_1) = 0.37$, $P(E_2) = 0.45$, and $P(E_1 \cap E_2) = 0.14$.
- Find the probability that E_2 does not occur.
 - Find the probability that either E_1 or E_2 occurs.
 - Find the probability that neither E_1 nor E_2 occurs.
68. An 8-letter password is automatically generated from the 26 lowercase letters of the English alphabet. Each letter is equally likely to be used, and letters can be repeated.
- What is the size of the sample space?
 - Find the probability that the word does not contain an “e.”
 - Find the probability that the word contains at least one “e.”
 - Find the probability that the word contains a single “e.”
 - Find the probability that the word contains both a single “h” and a single “x.”
 - Find the probability that the word contains either a single “h” or a single “x.”
69. A loaded die has the following probability distribution:

x_i	1	2	3	4	5	6
$p(x_i)$	0.2	0.05	0.1	0.2	0.3	0.15

When the die is rolled, let E_1 be the event that the number rolled is odd, let E_2 be the event that the number rolled is 3 or 6, and let E_3 be the event that the number rolled is 4 or more.

- Find $P(E_1)$.
 - Find $P(E_2)$.
 - Find $P(E_3)$.
 - Find $P(E_2 \cap E_3)$.
 - Find $P(E_1 \cup E_3)$.
70. A congressional race has a Democratic, a Republican, and an Independent candidate in a district where past voting patterns indicate that a Democrat is twice as likely to be elected as a Republican, and that a Republican is four times as likely to be elected as an Independent.
- Find the appropriate probability distribution.
 - What is the probability that a Democrat will be elected?
 - What is the probability that a Republican will not be elected?
71. At a certain school, 72% of the students play one or more sports. The percentage of students who play one or more sports and who graduate is 67%. Find the probability that a student graduates given that the student plays one or more sports.

72. On a particular manufacturing job, the probability that there will be a shortage of copper is 0.37. The probability that there will be a shortage of both copper and aluminum is 0.28. Find the probability of a shortage of aluminum given a shortage of copper.
73. In Example 71,
- What is the probability that a patient responds positively to compound A given that he or she responds positively to B ?
 - What is the probability that a patient responds positively to either compound A or compound B ?
 - What is the probability that a patient does not respond positively to either compound?
74. A food magazine surveys the preferences of its readers who like Asian food. The findings follow.

47% like Thai food (E_1)

39% like Indian food (E_2)

78% like Chinese food (E_3)

23% like both Thai and Indian food

31% like both Indian and Chinese food

29% like both Thai and Chinese food

- Extend observation 5 in Table 4.3 to the case of three events, E_1 , E_2 , and E_3 .
- Find the probability that a reader likes all three types of food.
- Find the probability that a reader likes Chinese food given that the reader likes Indian food.

For Exercises 75–80, a student takes a true–false quiz with four questions, each equally likely to be either T or F .

75. What is the probability of getting exactly one question wrong?
76. What is the probability of getting Question 1 correct?
77. What is the probability of getting three or more questions correct?
78. What is the probability of getting the first two questions correct?
79. What is the probability of getting the first two questions correct given that the answer to Question 1 is correct?
80. What is the probability of getting all four questions correct given that the answers to the first two questions are correct?

For Exercises 81–88, a family has 3 children; boys and girls are equally likely offspring.

81. What is the probability that the oldest child is a boy?
82. What is the probability that the youngest two children are girls?
83. What is the probability of 2 boys and 1 girl?
84. What is the probability of no girls?
85. What is the probability of at least 1 girl?
86. What is the probability of 3 girls?
87. What is the probability of 3 girls given that the first 2 are girls?
88. What is the probability of at least 1 boy and at least 1 girl given that there is at least 1 boy?

89. Prove Bayes' theorem. The proof parallels what was done in Example 73. Let E_1, \dots, E_n be disjoint events from a sample space S whose union equals S . If F is another event from S , then Bayes' theorem says that the probability of event E_i , $1 \leq i \leq n$, given event F , is

$$P(E_i|F) = \frac{P(F|E_i)P(E_i)}{\sum_{k=1}^n P(F|E_k)P(E_k)}$$

- a. Use the definition of $P(E_i|F)$ and $P(F|E_i)$ to prove that

$$P(E_i|F) = \frac{P(F|E_i)P(E_i)}{P(F)}$$

- b. Prove that

$$P(F) = \sum_{k=1}^n P(F \cap E_k)$$

- c. Use the definition of $P(F|E_i)$ and the result from part (b) to prove that

$$P(F) = \sum_{k=1}^n P(F|E_i)P(E_i)$$

- d. Using parts (a) and (c), prove Bayes' theorem.

90. Toys for boys and for girls are donated to a benefit event by two groups. The Lakeville Do-Gooders donated 5 toys for boys and 7 for girls. The Southside Champions Club donated 6 toys for boys and 5 for girls. The master of ceremonies pulls the first toy out of a bin and it's a toy for a boy. Find the probability that it was donated by the Do-Gooders.
91. An online pharmacy sells an over-the-counter drug, medication X , that is used for a variety of purposes. The pharmacy has data that say that 18% of its customers are HIV positive, 9% of its HIV-positive customers buy medication X , and 3% of the customers who are not HIV positive buy medication X . Find the probability that a customer who buys medication X is HIV positive. The pharmacy can use this data to shape its marketing/advertising plan.
92. Of the high blood pressure patients in a particular clinic, 62% are treated with medication X , the remainder with medication Y . It is known that 1.4% of patients using medication X suffer from fainting spells, as do 2.9% of the patients using medication Y . A patient known by the clinic to have high blood pressure suffers a fainting spell, but she does not remember which medication she is on. Which medication is she most likely to be taking? (*Hint*: Let E_1 and E_2 —treated with X and treated with Y , respectively—be events in the sample space of all patients with high blood pressure in the clinic, and let F be the event of fainting; find $P(X|F)$ and $P(Y|F)$.)
93. a. A fair die is rolled once. Let the random variable X equal the value that comes up. Find the expected value of X , $E(X)$.
- b. The die is now "loaded" so that a 2 comes up twice as often as any other number. Find the new expected value of X .
- c. Your answer to part (b) should be (greater than, less than) your answer to part (a). Explain why.

94. Two fair dice are rolled. The sample space S contains the 36 combinations of two numbers. For each member (r, s) of S , the random variable $X(r, s) = r + s$.
- Write a table showing the values for X and the probability of those values; instead of 36 columns each with probability $1/36$, do a column for each distinct value of X and show the probability of that value.
 - Find the expected value of the sum of the numbers that come up when two fair dice are rolled.
 - Find the expected value of the sum of the numbers that come up when two fair dice are rolled. This time let the sample space S consist of the ordered pairs (r, s) that can appear on the two dice. Use two different random variables over this sample space, where $X_1 =$ the value of the first component of the ordered pair and $X_2 =$ the value of the second component of the ordered pair. Make use of the linearity property from Equation (4).

95. At a gambling casino, a ball will be drawn from a bin containing 43 red balls, 27 green balls, and 8 blue balls. A player marks a game card with the color he or she believes will be picked. The prize money for guessing the correct color is

Red	\$3.00
Green	\$6.00
Blue	\$10.00

The price of the game card is \$5.00. Find the expected value of the prize money.

96. A directory on a computer's hard disk contains 12 files, 3 of which have viruses. If a file with a virus is selected, the virus is detected and another file is then selected. Find the expected number of files that must be selected in order to get a virus-free file.
97. Bit strings are sent across a computer network in packets of length 10. The probability of a bit getting corrupted (that is, a 0 gets changed to a 1 or vice versa) is 0.01. These bit errors are independent.
- Find the probability that in a single packet there are no bit errors. (*Hint:* It's OK to call a bit error a "success.")
 - Find the probability that there are no more than two bit errors.
 - Find the probability that there is at least one bit error.
98. Of the items produced in a certain manufacturing facility, 5% are defective. If 8 items are chosen at random, find the probability that
- 1 is defective.
 - 2 are defective.
 - none is defective.
 - at least 1 is defective.
 - at most 1 is defective.
99. A student has failed to study for a true–false test and guesses at every one of the 10 questions. If the passing grade is 8 correct answers, what is the probability that the student will pass the quiz?
100. A baseball player has a probability $p = 0.04$ of hitting a home run for each at bat. Find the minimum number of at bats he must take so that there is at least an 80% probability of hitting a home run (that is, at least 1 home run).
101. Find the average number of comparisons to search for a target element x using the sequential search algorithm under the assumption that x is equally likely to be at any of the n positions in the list or not in the list.
102. Find the average number of comparisons to search for a target element x using the sequential search algorithm under the assumption that x is not in the list 80% of the time, but if x is in the list it is equally likely to be at any of the n positions.

CHAPTER 4 REVIEW

TERMINOLOGY

- | | | |
|--|--------------------------------------|---|
| addition principle (p. 254) | decision tree (p. 257) | permutation (p. 272) |
| Bayes' theorem (p. 309) | denumerable set (p. 236) | pigeonhole principle (p. 269) |
| Bernoulli trial (p. 313) | disjoint sets (p. 232) | power set (p. 227) |
| Bernoulli experiment (p. 313) | dual of a set identity (p. 235) | principle of inclusion and exclusion (p. 267) |
| binary operation (p. 229) | empty set (p. 224) | probability axioms (p. 304) |
| binomial coefficient (p. 297) | equal sets (p. 223) | probability distribution (p. 305) |
| binomial distribution (p. 313) | event (p. 302) | probability of an event (p. 302) |
| binomial theorem (p. 296) | expected value (p. 310) | probability of an event E (p. 305) |
| Cantor's diagonalization method (p. 237) | finite set (p. 223) | proper subset (p. 225) |
| cardinality of a set (p. 236) | independent events (p. 307) | random variable (p. 310) |
| Cartesian product (cross product) of sets (p. 233) | infinite set (p. 223) | sample space (p. 302) |
| closed set under an operation (p. 229) | intersection of sets (p. 231) | set difference (p. 232) |
| combination (p. 274) | lexicographical ordering (p. 281) | subset (p. 224) |
| combinatorial proof (p. 296) | linearity of expected value (p. 312) | unary operation (p. 230) |
| combinatorics (p. 252) | multiplication principle (p. 253) | uncountable set (p. 236) |
| complement of a set (p. 232) | n factorial (p. 272) | union of sets (p. 231) |
| conditional probability (p. 307) | null set (p. 224) | universal set (p. 231) |
| countable set (p. 236) | ordered pair (p. 228) | universe of discourse (p. 231) |
| | Pascal's formula (p. 295) | weighted average (p. 310) |
| | Pascal's triangle (p. 294) | well-defined operation (p. 229) |

SELF-TEST

Answer the following true–false questions.

Section 4.1

- The empty set is a proper subset of every set.
- If A and B are disjoint sets, then $(A - B) \cup (B - A) = A \cup B$.
- If a set has n elements, then its power set has 2^n elements.
- If a binary operation \circ on a set S is well-defined, then $x \circ y \in S$ for all x and y in S .
- Cantor's diagonalization method is a way to prove that certain sets are denumerable.

Section 4.2

- According to the multiplication principle, the number of outcomes for a sequence of tasks is the product of the number of outcomes for each separate task.
- The addition principle finds the total number of branches of a decision tree.

- The addition principle requires the tasks at hand to have disjoint sets of outcomes.
- The multiplication principle says that the number of elements in $A \times B$ equals the number of elements in A times the number of elements in B .
- Any problem that requires a decision tree for its solution cannot be solved by the multiplication principle.

Section 4.3

- The principle of inclusion and exclusion requires that A and B be disjoint sets in order to find the number of elements in $A \cup B$.
- The principle of inclusion and exclusion applied to two sets says that the number of elements in the union minus the number of elements in the intersection is the sum of the number of elements in each set.

3. The principle of inclusion and exclusion applies to the union of any number of sets as long as at least one of them is finite.
4. The pigeonhole principle is a way to count the number of elements in the union of disjoint sets, or “pigeonholes.”
5. The pigeonhole principle guarantees that if there are 8 people in a room, at least 2 must have been born on the same day of the week.
2. Pascal’s formula says that an “interior” number in Pascal’s triangle is the sum of the two numbers directly above it in the triangle.
3. In the expansion of a binomial to the n th power, the k th term is found in row k of Pascal’s triangle.
4. A combinatorial argument is one that is based on counting techniques.
5. The coefficient of the seventh term in the expansion of $(a + b)^{12}$ is given by the expression $C(12, 6)$.

Section 4.4

1. A permutation is an ordered arrangement of objects.
2. The number of combinations of r objects out of n , $r > 1$, is fewer than the number of permutations of r objects out of n .
3. To find the number of ways a subset of r objects can be selected from n objects, use the formula $P(n, r)$.
4. The number of permutations of the letters in a word with three sets of repeated letters is $n!/3$.
5. The formula $C(r + n - 1, r)$ computes the number of combinations of r objects out of n objects where objects may be used repeatedly.

Section 4.5

1. Pascal’s triangle consists of rows that represent the number of ways to arrange r out of n objects for various r .

Section 4.6

1. The probability of an event always falls in the range between 0 and 1.
2. In a sample space with n equally likely outcomes, the probability distribution is $1/n$ for each outcome.
3. To find the probability of several events occurring, multiply the probabilities of the individual events.
4. A random variable is a variable whose value is randomly assigned using a random number generator.
5. If E_1 and E_2 are disjoint events whose union equals the sample space, then Bayes’ theorem allows you to derive the conditional probability $P(E_1 | F)$ if you know $P(F | E_1)$, $P(F | E_2)$, $P(E_1)$ and $P(E_2)$.

ON THE COMPUTER

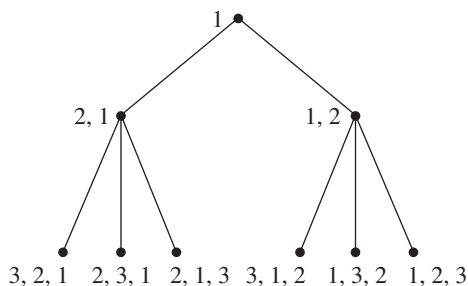
For Exercises 1–10, write a computer program that produces the desired output from the given input.

1. *Input:* Elements in a finite set S
Output: Elements in $\wp(S)$
Algorithm: Use recursion.
2. *Input:* Arithmetic expression in postfix notation (see Exercise 45 in Section 4.1)
Output: Value of the expression
3. *Input:* Arithmetic expression in infix notation (see Exercise 45 in Section 4.1)
Output: Postfix form of the expression
Do this problem in two ways:
 - a. Assume that the input is fully parenthesized.
 - b. Do not assume that the input is fully parenthesized, but apply the proper order of precedence of operators within the program (order of precedence of operators is parenthesized expressions first, then exponentiation, then multiplication and division, then addition and subtraction).
4. *Input:* Values for n and r , $0 \leq r \leq n$
Output: Value of $P(n, r)$
5. *Input:* Values for n and r , $0 \leq r \leq n$
Output: Value of $C(n, r)$
6. *Input:* Value for n
Output: All values of $C(n, r)$, $0 \leq r \leq n$

7. *Input:* Value for n

Output: All permutations of the integers $1, \dots, n$

Here is an outline for an alternative to the algorithm given in Section 4.4 to generate the $n!$ permutations of the integers $\{1, \dots, n\}$. Use a recursive algorithm. Once a permutation A of the integers $1, \dots, k - 1$ exists, permutations of the integers $1, \dots, k$ can be obtained by inserting integer k into every possible position in A . Every time $k = n$, any permutation so obtained can be written out. Initiate the process by sending 1 to an empty permutation list. For the case $n = 3$, for example, this algorithm successively traverses the branches of the tree and prints out the leaves.



8. *Input:* Values for a, b , and n

Output: Value of $(a + b)^n$

- Use the binomial theorem to compute your result.
- Compute $a + b$ and raise this value to the n th power; compare your answer with that of part (a).

9. *Input:* Values for a, b, n , and $r, 1 \leq r \leq n + 1$

Output: r th term in the expansion of $(a + b)^n$

10. *Input:* A random variable and a probability distribution for a finite sample space.

Output: The expected value of the random variable.

11. Write a program that allows the user to enter a value for $n, 1 \leq n \leq 10$, and then queries the user for the values needed on the right side of Equation (4) of Section 4.3 (the principle of inclusion and exclusion) and computes the value of

$$|A_1 \cup \dots \cup A_n|$$

12. Write a program to generate a given number of rows of Pascal's triangle. Do this problem in two ways.

- Use the definition of Pascal's triangle (and perhaps use your answer to compute Exercise 5 as a function).
- Use recursion and Pascal's formula.

13. Benford's law, also called the first-digit law, states that in many (but not all) large numerical data sets, the first digit is not equally likely to be 1 through 9. In fact, the probability that the first digit equals 1, $p(1)$, is about 30%, and the probability for each successive value of the first digit goes down until $p(9)$ is about 4.6%. The formula for Benford's law is

$$p(d) = \log_{10} \left(1 + \frac{1}{d} \right)$$

Evidence based on Benford's law is admissible in court and has been used to help detect fraudulent data in accounting, economics, scientific research, and other areas.

- Use the given formula to compute the probability of occurrence in the first digit of digits 1–9.
- Write a program to generate the first 200 Fibonacci numbers and determine whether the first digits follow Benford's law.

This page intentionally left blank

Relations, Functions, and Matrices

CHAPTER OBJECTIVES

After studying this chapter, you will be able to:

- Identify ordered pairs related by a binary relation.
- Test a binary relation for the reflexive, symmetric, transitive, and antisymmetric properties.
- Find the reflexive, symmetric, and transitive closures of a binary relation.
- Recognize partial orderings and construct Hasse diagrams for them.
- Recognize an equivalence relation on a set and describe how it partitions the set into equivalence classes.
- Draw a PERT chart from a task table.
- Find the minimum time-to-completion and a critical path in a PERT chart.
- Extend a partial ordering on a finite set to a total ordering by doing a topological sort.
- Understand the entity-relationship model and the relational model for an enterprise.
- Perform restrict, project, and join operations in a relational database.
- Create relational database queries in the languages of relational algebra, SQL, and relational calculus.
- Determine whether a binary relation is a function.
- Test a function for the onto and one-to-one properties.
- Create composite functions.
- Decide whether a function has an inverse function and what the inverse function is.
- Manipulate cycle notation for permutation functions.
- Compute the number of functions, onto functions, and one-to-one functions from one finite set to another.
- Understand order of magnitude as a relative measure of function rate of growth.
- Build a hash table using a modulo hash function.
- Encode and decode messages using RSA public key encryption.
- Use the mod function to compute check digits for various identification codes.
- Perform matrix arithmetic on matrices of appropriate dimensions.
- Solve systems of linear equations using Gaussian elimination.
- Perform Boolean arithmetic operations on Boolean matrices of appropriate dimensions.

Your company has developed a program for use on a small parallel processing machine. According to the technical documentation, the program executes processes P1, P2, and P3 in parallel; these processes all need results from process P4, so they must wait for Process P4 to complete execution before they begin. Processes P7 and P10 execute in parallel but must wait until processes P1, P2, and P3 have finished. Process P4 requires results from P5 and P6 before it can begin execution. P5 and P6 execute in parallel. Processes P8 and P11 execute in parallel, but P8 must wait for Process P7 to complete and P11 must wait for process P10 to complete. Process P9 must wait for results from P8 and P11. You have been assigned to convert the software for use on a single processor machine.

Question: In what order should the processes be executed?

Here various pairs of processes are related to one another by a “prerequisite” relation. This is a special case of a binary relation, a relationship between pairs of elements within a set. We will study the various properties of binary relations in Section 5.1. One type of binary relation is called a partial ordering; elements related by a partial ordering can be represented graphically. Another type of binary relation is an equivalence relation; elements related by an equivalence relation can be grouped into classes.

A topological sort extends a partial ordering to a total ordering. For a partial ordering of prerequisite tasks, a corresponding total ordering identifies the sequential order in which the tasks would have to be done, which is the solution to the parallel processing conversion problem. Topological sorting is presented in Section 5.2.

A generalization of a binary relation forms the basis for relational databases, considered in Section 5.3. Using operations of restrict, project, and join on the relations in a database, we can make various queries of the database.

A function is a special kind of binary relation. Functions, as well as relations, describe a number of real-world situations. Functions can also have special properties, as discussed in Section 5.4. Order of magnitude, presented in Section 5.5, provides a way to compare the growth rate of two functions and is useful in the analysis of algorithms. A simple function called the modulo function has a surprising number of applications, ranging from encryption algorithms for computer security to the basis for artistic design patterns. Some of these applications are mentioned in Section 5.6.

In Section 5.7, we consider matrices and develop an arithmetic for manipulating them. Matrices provide a mechanism for solving systems of linear equations. We will later use matrices to represent relations and graphs.

SECTION 5.1 RELATIONS

Binary Relations

If we learn that two people, Henrietta and Horace, are related, we understand that there is some family connection between them—that (Henrietta, Horace) stands out from other ordered pairs of people because there is a relationship (cousins, sister and brother, or whatever) that Henrietta and Horace satisfy. The mathematical

analogue is to distinguish certain ordered pairs of objects from other ordered pairs because the components of the distinguished pairs satisfy some relationship that the components of the other pairs do not.

EXAMPLE 1

Remember (Section 4.1) that the Cartesian product of a set S with itself, $S \times S$ or S^2 , is the set of all ordered pairs of elements of S . Let $S = \{1, 2, 3\}$; then

$$S \times S = \{(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)\}$$

If we were interested in the relationship of equality, then $(1, 1)$, $(2, 2)$, $(3, 3)$ would be the distinguished elements of $S \times S$, that is, the only ordered pairs whose components are equal. If we were interested in the relationship of one number being less than another, we would choose $(1, 2)$, $(1, 3)$, and $(2, 3)$ as the distinguished ordered pairs of $S \times S$. ●

In Example 1, we could pick out the distinguished ordered pairs (x, y) by saying that $x = y$ or that $x < y$. Similarly, the notation $x \rho y$ indicates that the ordered pair (x, y) satisfies a relation ρ . The relation ρ may be defined in words or by an equation or simply by listing the ordered pairs that satisfy ρ .

EXAMPLE 2

Let $S = \{1, 2, 4\}$. On the set $S \times S = \{(1, 1), (1, 2), (1, 4), (2, 1), (2, 2), (2, 4), (4, 1), (4, 2), (4, 4)\}$, a relation ρ can be defined by $x \rho y$ if and only if $x = y/2$, abbreviated $x \rho y \leftrightarrow x = y/2$. Thus $(1, 2)$ and $(2, 4)$ satisfy ρ . Alternatively, the same ρ could be defined by saying that $\{(1, 2), (2, 4)\}$ is the set of ordered pairs satisfying ρ . ●

As in Example 2, one way to define the binary relation ρ is to specify a subset of $S \times S$. Formally, this is the definition of a binary relation on a set.

● **DEFINITION BINARY RELATION ON A SET S**

A **binary relation on a set S** is a subset of $S \times S$ (a set of ordered pairs of elements of S).

Now that we know that a binary relation ρ is a subset, we see that

$$x \rho y \leftrightarrow (x, y) \in \rho$$

Generally, a binary relation is defined by describing the relation rather than by listing the ordered pairs. The description gives a characterizing property of elements of the relation; that is, it is a binary predicate satisfied by certain ordered pairs. A binary relation implies a yes/no result—an ordered pair either does or does not satisfy the binary predicate and either does or does not belong to the relation.

EXAMPLE 3

Let $S = \{1, 2\}$. Then $S \times S = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$. Let ρ on S be given by the description $x \rho y \leftrightarrow x + y$ is odd. Then $(1, 2) \in \rho$ and $(2, 1) \in \rho$. The ordered pair $(1, 1) \notin \rho$ because $1 + 1$ is not odd. Similarly $(2, 2) \notin \rho$. ●

EXAMPLE 4

Let $S = \{1, 2\}$. Then $S \times S = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$. If ρ is defined on S by $\rho = \{(1, 1), (2, 1)\}$, then $1 \rho 1$ and $2 \rho 1$ hold, but not, for instance, $1 \rho 2$. Here ρ seems to have no obvious verbal description. ●

In this section we will be concerned almost exclusively with binary relations on a single set, but more generally, relations can be defined on multiple sets.

● **DEFINITION RELATIONS ON MULTIPLE SETS**

Given two sets S and T , a **binary relation from S to T** (also called a **binary relation on $S \times T$**) is a subset of $S \times T$. Given n sets S_1, S_2, \dots, S_n , $n > 2$, an **n -ary relation on $S_1 \times S_2 \times \cdots \times S_n$** is a subset of $S_1 \times S_2 \times \cdots \times S_n$.

EXAMPLE 5

Let $S = \{1, 2, 3\}$ and $T = \{2, 4, 7\}$. Then the set

$$\{(1, 2), (2, 4), (2, 7)\}$$

consists of elements from $S \times T$. It is a binary relation from S to T . ●

PRACTICE 1

For each of the following binary relations ρ on \mathbb{N} , decide which of the given ordered pairs belong to ρ .

- $x \rho y \leftrightarrow x = y + 1$; $(2, 2), (2, 3), (3, 3), (3, 2)$
- $x \rho y \leftrightarrow x$ divides y ; $(2, 4), (2, 5), (2, 6)$
- $x \rho y \leftrightarrow x$ is odd; $(2, 3), (3, 4), (4, 5), (5, 6)$
- $x \rho y \leftrightarrow x > y^2$; $(1, 2), (2, 1), (5, 2), (6, 4), (4, 3)$

If ρ is a binary relation on S , then ρ will consist of a set of ordered pairs of the form (s_1, s_2) . A given first component s_1 or second component s_2 can be paired in various ways in the relation. The relation is **one-to-one** if each first component and each second component appears only once in the relation. The relation is **one-to-many** if some first component s_1 appears more than once; that is, one s_1 is paired with more than one second component. It is **many-to-one** if some second component s_2 is paired with more than one first component. Finally, it is **many-to-many** if at least one s_1 is paired with more than one second component and at least one s_2 is paired with more than one first component. Figure 5.1 illustrates these four possibilities. Note that not all values in S need be components in ordered pairs of ρ .

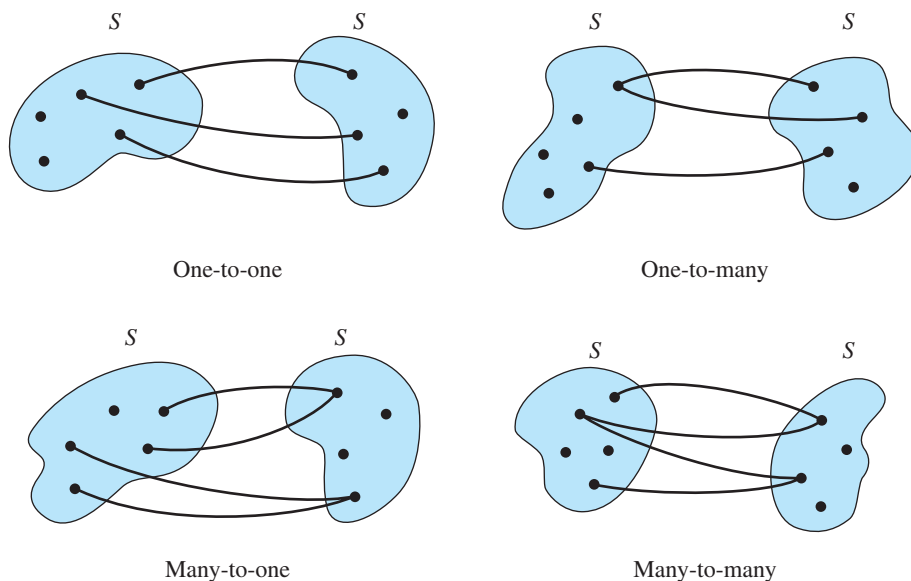


Figure 5.1

These ideas extend to relations from a set S to a set T . The relation of Example 5 is one-to-many because the first component 2 appears more than once; 2 from set S is associated with both 4 and 7 from set T .

PRACTICE 2 Identify each of these relations on S , where $S = \{2, 5, 7, 9\}$, as one-to-one, one-to-many, many-to-one, or many-to-many.

- $\{(5, 2), (7, 5), (9, 2)\}$
- $\{(2, 5), (5, 7), (7, 2)\}$
- $\{(7, 9), (2, 5), (9, 9), (2, 7)\}$

Suppose B is the set of all binary relations on a given set S . If ρ and σ belong to B , then they are subsets of $S \times S$. As such, we can perform set operations of union, intersection, and complementation that result in new subsets of $S \times S$, that is, new binary relations, which we will denote by $\rho \cup \sigma$, $\rho \cap \sigma$, and ρ' , respectively. Thus

$$\begin{aligned} x(\rho \cup \sigma)y &\leftrightarrow x\rho y \text{ or } x\sigma y \\ x(\rho \cap \sigma)y &\leftrightarrow x\rho y \text{ and } x\sigma y \\ x\rho'y &\leftrightarrow \text{not } x\rho y \end{aligned}$$

PRACTICE 3 Let ρ and σ be two binary relations on \mathbb{N} defined by $x\rho y \leftrightarrow x = y$ and $x\sigma y \leftrightarrow x < y$. Give verbal descriptions for parts (a), (b), and (c); give a set description for part (d).

- What is the relation $\rho \cup \sigma$?
- What is the relation ρ' ?
- What is the relation σ' ?
- What is the relation $\rho \cap \sigma$?

The following facts about the operations \cup , \cap and $'$ on relations are immediate consequences of the basic set identities found in Section 4.1. The set S^2 (which is, after all, a subset of S^2) is being viewed here as a binary relation on S .

$$\begin{array}{ll} \text{1a. } \rho \cup \sigma = \sigma \cup \rho & \text{1b. } \rho \cap \sigma = \sigma \cap \rho \\ \text{2a. } (\rho \cup \sigma) \cup \gamma = \rho \cup (\sigma \cup \gamma) & \text{2b. } (\rho \cap \sigma) \cap \gamma = \rho \cap (\sigma \cap \gamma) \\ \text{3a. } \rho \cup (\sigma \cap \gamma) = (\rho \cup \sigma) \cap (\rho \cup \gamma) & \text{3b. } \rho \cap (\sigma \cup \gamma) = (\rho \cap \sigma) \cup (\rho \cap \gamma) \\ \text{4a. } \rho \cup \emptyset = \rho & \text{4b. } \rho \cap S^2 = \rho \\ \text{5a. } \rho \cup \rho' = S^2 & \text{5b. } \rho \cap \rho' = \emptyset \end{array}$$

REMINDER

Reflexive—Every x is related to itself.

Symmetric—If x is related to y , then y is related to x .

Transitive—If x is related to y and y is related to z , then x is related to z .

Properties of Relations

A binary relation on a set S may have certain properties. For example, the relation ρ of equality on S , $(x, y) \in \rho \leftrightarrow x = y$, has three properties: (1) for any $x \in S$, $x = x$, or $(x, x) \in \rho$; (2) for any $x, y \in S$, if $x = y$ then $y = x$, or $(x, y) \in \rho \rightarrow (y, x) \in \rho$; and (3) for any $x, y, z \in S$, if $x = y$ and $y = z$, then $x = z$, or $[(x, y) \in \rho \text{ and } (y, z) \in \rho] \rightarrow (x, z) \in \rho$. These three properties make the equality relation reflexive, symmetric, and transitive.

DEFINITION REFLEXIVE, SYMMETRIC, AND TRANSITIVE RELATIONS

Let ρ be a binary relation on a set S . Then

$$\begin{array}{ll} \rho \text{ is reflexive means} & (\forall x)(x \in S \rightarrow (x, x) \in \rho) \\ \rho \text{ is symmetric means} & (\forall x)(\forall y)(x \in S \wedge y \in S \wedge (x, y) \in \rho \rightarrow (y, x) \in \rho) \\ \rho \text{ is transitive means} & (\forall x)(\forall y)(\forall z)(x \in S \wedge y \in S \wedge z \in S \wedge \\ & (x, y) \in \rho \wedge (y, z) \in \rho \rightarrow (x, z) \in \rho) \end{array}$$

EXAMPLE 6

Consider the relation \leq on the set \mathbb{N} . This relation is reflexive because for any nonnegative integer x , $x \leq x$. It is also a transitive relation because for any nonnegative integers x , y , and z , if $x \leq y$ and $y \leq z$, then $x \leq z$. However, \leq is not symmetric; $3 \leq 4$ does not imply $4 \leq 3$. In fact, for any $x, y \in \mathbb{N}$, if both $x \leq y$ and $y \leq x$, then $x = y$. This characteristic is described by saying that \leq is antisymmetric. ●

DEFINITION ANTISYMMETRIC RELATION

Let ρ be a binary relation on a set S . Then ρ is **antisymmetric** means

$$(\forall x)(\forall y)(x \in S \wedge y \in S \wedge (x, y) \in \rho \wedge (y, x) \in \rho \rightarrow x = y)$$

EXAMPLE 7

Let $S = \wp(\mathbb{N})$. Define a binary relation ρ on S by $A \rho B \leftrightarrow A \subseteq B$. Then ρ is reflexive because every set is a subset of itself. Also, ρ is transitive because if A is a subset of B and B is a subset of C , then A is a subset of C . Finally, ρ is antisymmetric because if A is a subset of B and B is a subset of A , then A and B are equal sets. ●

REMINDER

Antisymmetric—If x is related to y and y is related to x , then $x = y$.

All four relational properties involve the implication connective. The universal quantifiers mean that the implications must be true for arbitrary choices of variables. Recall that to prove an implication true, we assume that the antecedent is true and prove that the consequent must also be true. For the reflexive property, the antecedent just chooses an arbitrary element in S ; the consequent says that this element must be related to itself. For a relation ρ on a set to be reflexive, then, every element in the set must be related to itself, which specifies certain ordered pairs that must belong to ρ .

However, in the symmetric, transitive, and antisymmetric properties, the antecedent does not say only that the elements are in S . To prove that a relation is symmetric, for example, we must show that if x and y are arbitrary elements in S and if in addition x is related to y , then it must be the case that y is related to x . This says that if certain ordered pairs are found in ρ , then certain other ordered pairs must also be in ρ in order for ρ to be symmetric. In other words, knowledge of the set S is critical to determining whether reflexivity holds, while to determine the other properties, it is sufficient just to look at the ordered pairs in ρ .

At any rate, the question of whether a given relation on a set S has a certain property requires a yes or no answer. The property either holds or it doesn't.

PRACTICE 4 Let $S = \{1, 2, 3\}$.

- If a relation ρ on S is reflexive, what ordered pairs must belong to ρ ?
- If a relation ρ on S is symmetric, what ordered pairs must belong to ρ ? (This is a trick question; see the answer at the back of the book.)
- If a relation ρ on S is symmetric and if $(a, b) \in \rho$, then what other ordered pair must belong to ρ ?
- If a relation ρ on S is antisymmetric and if (a, b) and (b, a) belong to ρ , what must be true?
- Is the relation $\rho = \{(1, 2)\}$ on S transitive? (*Hint*: Remember the truth table for implication.) ■

The properties of symmetry and antisymmetry for binary relations are not precisely opposites. Antisymmetric does not mean “not symmetric.” A relation is not symmetric if some (x, y) belongs to the relation but (y, x) does not. More formally, not symmetric means

$$\begin{aligned} & ((\forall x)(\forall y)[x \in S \wedge y \in S \wedge (x, y) \in \rho \rightarrow (y, x) \in \rho])' \\ & \leftrightarrow (\exists x)(\exists y)[x \in S \wedge y \in S \wedge (x, y) \in \rho \rightarrow (y, x) \notin \rho]' \\ & \leftrightarrow (\exists x)(\exists y)[(x \in S \wedge y \in S \wedge (x, y) \in \rho)' \vee (y, x) \in \rho]' \\ & \leftrightarrow (\exists x)(\exists y)[(x \in S \wedge y \in S \wedge (x, y) \in \rho) \wedge (y, x) \notin \rho] \end{aligned}$$

Relations can therefore be symmetric and not antisymmetric, antisymmetric and not symmetric, both, or neither.

The equality relation on a set S is both symmetric and antisymmetric. However, the equality relation on S (or a subset of this relation) is the only relation having both these properties. To illustrate, suppose ρ is a symmetric and antisymmetric relation on S , and let $(x, y) \in \rho$. By symmetry, it follows that $(y, x) \in \rho$. But by antisymmetry, $x = y$. Thus, only equal elements can be related. The relation $\rho = \{(1, 2), (2, 1), (1, 3)\}$ on the set $S = \{1, 2, 3\}$ is neither symmetric— $(1, 3)$ belongs but $(3, 1)$ does not—nor antisymmetric— $(1, 2)$ and $(2, 1)$ belong, but $1 \neq 2$.

PRACTICE 5 Test each binary relation on the given set S for reflexivity, symmetry, antisymmetry, and transitivity.

- $S = \mathbb{N}$; $x \rho y \leftrightarrow x + y$ is even
- $S = \mathbb{Z}^+$ (positive integers); $x \rho y \leftrightarrow x$ divides y
- $S =$ set of all lines in the plane; $x \rho y \leftrightarrow x$ is parallel to y or x coincides with y
- $S = \mathbb{N}$; $x \rho y \leftrightarrow x = y^2$
- $S = \{0, 1\}$; $x \rho y \leftrightarrow x = y^2$
- $S = \{x \mid x \text{ is a person living in Peoria}\}$; $x \rho y \leftrightarrow x$ is older than y
- $S = \{x \mid x \text{ is a student in your class}\}$; $x \rho y \leftrightarrow x$ sits in the same row as y
- $S = \{1, 2, 3\}$; $\rho = \{(1, 1), (2, 2), (3, 3), (1, 2), (2, 1)\}$

EXAMPLE 8

The discussion on recursion in Prolog (Section 1.5) noted that a recursive rule should be used when the predicate being described is one that is inherited from one object to the next. The predicate *in-food-chain* used there has this property because

$$\text{in-food-chain}(x, y) \wedge \text{in-food-chain}(y, z) \rightarrow \text{in-food-chain}(x, z)$$

Now we see that this is simply the transitive property.

Closures of Relations

If a relation ρ on a set S fails to have a certain property, we may be able to extend ρ to a relation ρ^* on S that does have that property. By “extend,” we mean that the new relation ρ^* will contain all the ordered pairs in ρ plus the additional ordered pairs needed for the desired property to hold. Thus $\rho \subseteq \rho^*$. If ρ^* is the smallest such set, then ρ^* is called the closure of ρ with respect to that property.

DEFINITION CLOSURE OF A RELATION

A binary relation ρ^* on a set S is the **closure of a relation** ρ on S with respect to property P if

- ρ^* has property P .
- $\rho \subseteq \rho^*$.
- ρ^* is a subset of any other relation on S that includes ρ and has property P .

We can look for the **reflexive closure**, the **symmetric closure**, and the **transitive closure** of a relation on a set. Of course, if the relation already has a property, it is its own closure with respect to that property.

EXAMPLE 9

Let $S = \{1, 2, 3\}$ and $\rho = \{(1, 1), (1, 2), (1, 3), (3, 1), (2, 3)\}$. Then ρ is not reflexive, not symmetric, and not transitive. The closure of ρ with respect to reflexivity is

$$\rho^* = \{(1, 1), (1, 2), (1, 3), (3, 1), (2, 3), (2, 2), (3, 3)\}$$

This relation is reflexive and contains ρ . Furthermore, any reflexive relation on S would have to contain the new ordered pairs we've added— $(2, 2)$ and $(3, 3)$ —so no smaller reflexive relation can exist; that is, any reflexive relation containing ρ must have the relation above as a subset.

The closure of ρ with respect to symmetry is

$$\rho^* = \{(1, 1), (1, 2), (1, 3), (3, 1), (2, 3), (2, 1), (3, 2)\}$$

Here it is also clear that we have added just those new pairs required— $(2, 1)$ and $(3, 2)$ —for the relation to be symmetric.

For both reflexive closure and symmetric closure, we only had to inspect the ordered pairs already in ρ to find out what ordered pairs we needed to add (assuming we knew what the set S was). The reflexive or symmetric closure of the relation could be found in one step. Transitive closure may require a series of steps. Inspecting the ordered pairs in our example ρ , we see that we need to add $(3, 2)$ (because of $(3, 1)$ and $(1, 2)$), $(3, 3)$ (because of $(3, 1)$ and $(1, 3)$), and $(2, 1)$ (because of $(2, 3)$ and $(3, 1)$). This gives the relation

$$\{(1, 1), (1, 2), (1, 3), (3, 1), (2, 3), (3, 2), (3, 3), (2, 1)\}$$

However, this relation is still not transitive. Because of the new pair $(2, 1)$ and the old pair $(1, 2)$, we need to add $(2, 2)$. This gives the relation

$$\{(1, 1), (1, 2), (1, 3), (3, 1), (2, 3), (3, 2), (3, 3), (2, 1), (2, 2)\}$$

which is transitive and also the smallest transitive relation containing ρ . It is the transitive closure of ρ . ●

As in Example 9, one way to find the transitive closure of a relation is to inspect the ordered pairs in the original relation, add new pairs if necessary, inspect the resulting relation, add new pairs if necessary, and so on, until a transitive relation is achieved. This is a rather ad hoc procedure, and we will give a better algorithm in Chapter 7. There we will also see that the transitive closure of a binary relation is related to “reachability in a directed graph,” which has many applications.

PRACTICE 6 Does it make sense to look for the antisymmetric closure of a relation on a set? Why or why not? ■

PRACTICE 7 Find the reflexive, symmetric, and transitive closure of the relation

$$\{(a, a), (b, b), (c, c), (a, c), (a, d), (b, d), (c, a), (d, a)\}$$

on the set $S = \{a, b, c, d\}$. ■

For the rest of this section we will concentrate on two types of binary relations that are characterized by which properties (reflexivity, symmetry, antisymmetry, and transitivity) they satisfy.

Partial Orderings

DEFINITION PARTIAL ORDERING

A binary relation on a set S that is reflexive, antisymmetric, and transitive is called a **partial ordering** on S .

From previous examples and Practice 5, we have the following instances of partial orderings:

- On \mathbb{N} , $x \rho y \leftrightarrow x \leq y$.
- On $\wp(\mathbb{N})$, $A \rho B \leftrightarrow A \subseteq B$.
- On \mathbb{Z}^+ , $x \rho y \leftrightarrow x$ divides y .
- On $\{0, 1\}$, $x \rho y \leftrightarrow x = y^2$.

If ρ is a partial ordering on S , then the ordered pair (S, ρ) is called a **partially ordered set** (also known as a **poset**). We will denote an arbitrary, partially ordered set by (S, \leq) ; in any particular case, \leq has some definite meaning such as “less than or equal to,” “is a subset of,” “divides,” and so on. (The symbol for a generic partial ordering, \leq , is designed to resemble the inequality symbol \leq , which, as we’ve just noted, is a partial ordering on the set \mathbb{N} or on any other set in which a less-than-or-equal-to relation makes sense.)

Let (S, \leq) be a partially ordered set, and let $A \subseteq S$. Then \leq is a set of ordered pairs of elements of S , some of which may be ordered pairs of elements of A . If we select from \leq the ordered pairs of elements of A , this new set is called the **restriction** of \leq to A and is a partial ordering on A . (Do you see why the three required properties still hold?) For instance, once we know that the relation “ x divides y ” is a partial ordering on \mathbb{Z}^+ , we automatically know that “ x divides y ” is a partial ordering on $\{1, 2, 3, 6, 12, 18\}$.

We want to introduce some terminology about partially ordered sets. Let (S, \leq) be a partially ordered set. If $x \leq y$, then either $x = y$ or $x \neq y$. If $x \leq y$ but $x \neq y$, we write $x < y$ and say that x is a **predecessor** of y or y is a **successor** of x . A given y may have many predecessors, but if $x < y$ and there is no z with $x < z < y$, then x is an **immediate predecessor** of y .

PRACTICE 8 Consider the relation “ x divides y ” on $\{1, 2, 3, 6, 12, 18\}$.

- a. Write the ordered pairs (x, y) of this relation.
- b. Write all the predecessors of 6.
- c. Write all the immediate predecessors of 6.

If S is finite, we can visually depict a partially ordered set (S, \leq) by using a **Hasse diagram**. Each of the elements of S is represented by a dot, called a

node, or **vertex**, of the diagram. If x is an immediate predecessor of y , then the node for y is placed above the node for x and the two nodes are connected by a straight-line segment.

EXAMPLE 10

Consider $\wp(\{1, 2\})$ under the relation of set inclusion. This is a partially ordered set, a restriction of the partially ordered set $(\wp(\mathbb{N}), \subseteq)$. The elements of $\wp(\{1, 2\})$ are \emptyset , $\{1\}$, $\{2\}$, and $\{1, 2\}$. The binary relation \subseteq consists of the following ordered pairs:

$$(\emptyset, \emptyset), (\{1\}, \{1\}), (\{2\}, \{2\}), (\{1, 2\}, \{1, 2\}), (\emptyset, \{1\}),$$

$$(\emptyset, \{2\}), (\emptyset, \{1, 2\}), (\{1\}, \{1, 2\}), (\{2\}, \{1, 2\})$$

The Hasse diagram of this partially ordered set appears in Figure 5.2. Note that although \emptyset is not an immediate predecessor of $\{1, 2\}$, it is a predecessor of $\{1, 2\}$ (shown in the diagram by the chain of upward line segments connecting \emptyset with $\{1, 2\}$).

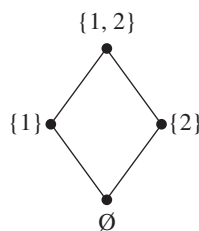


Figure 5.2

REMINDER

Two nodes in a Hasse diagram should never be joined by a horizontal line.

PRACTICE 9

Draw the Hasse diagram for the relation “ x divides y ” on $\{1, 2, 3, 6, 12, 18\}$.

The Hasse diagram of a partially ordered set conveys all the information about the partial ordering. We can reconstruct the set of ordered pairs making up the partial ordering just by looking at the diagram. The lines in the diagram tell us immediate (predecessor, successor) pairs. We can fill in the rest by using the reflexive and transitive properties. Thus, given the Hasse diagram in Figure 5.3 of a partial ordering \leq on a set $\{a, b, c, d, e, f\}$, we can conclude that \leq is the set

$$\{(a, a), (b, b), (c, c), (d, d), (e, e), (f, f), (a, b), (a, c), (a, d), (a, e), (d, e)\}$$

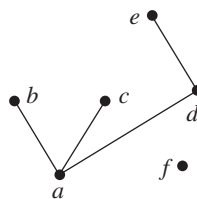


Figure 5.3

Two elements of S may be unrelated in a partial ordering of S . In Example 10, $\{1\}$ and $\{2\}$ are unrelated; so are 2 and 3, and 12 and 18 in Practice 9. In Figure 5.3, f is not related to any other element. A partial ordering in which every element of the set is related to every other element is called a **total ordering**, or **chain**. The Hasse diagram for a total ordering on a four-element set looks like Figure 5.4. The relation \leq on \mathbb{N} is a total ordering, although we can't draw a Hasse diagram because \mathbb{N} is an infinite set.



Figure 5.4

Again, let (S, \leq) be a partially ordered set. If there is a $y \in S$ with $y \leq x$ for all $x \in S$, then y is a **least element** of the partially ordered set. A least element, if it exists, is unique. To show its uniqueness, assume that y and z are both least elements. Then $y \leq z$ because y is least and $z \leq y$ because z is least; by antisymmetry, $y = z$. An element $y \in S$ is **minimal** if there is no $x \in S$ with $x < y$. In the Hasse diagram, a least element is below all others, while a minimal element has no elements below it. Similar definitions apply for greatest element and maximal elements.

PRACTICE 10 Define **greatest element** and **maximal element** in a partially ordered set (S, \leq) . ■

EXAMPLE 11 In the partially ordered set of Practice 9, 1 is both least and minimal; 12 and 18 are both maximal, but there is no greatest element. ●

A least element is always minimal and a greatest element is always maximal, but the converses are not true (see Example 11). In a totally ordered set, however, a minimal element is the least element and a maximal element is the greatest element.

PRACTICE 11 Draw the Hasse diagram for a partially ordered set with four elements in which there are two minimal elements but no least element, two maximal elements but no greatest element, and each element is related to exactly two other elements. ■

Partial orderings satisfy the properties of reflexivity, antisymmetry, and transitivity. Another type of binary relation, which we study next, satisfies a different set of properties.

Equivalence Relations

DEFINITION EQUIVALENCE RELATION

A binary relation on a set S that is reflexive, symmetric, and transitive is called an **equivalence relation** on S .

REMINDER

A partial ordering is anti-symmetric; an equivalence relation is symmetric.

We have already come across the following examples of equivalence relations:

On any set S , $x \rho y \leftrightarrow x = y$.

On \mathbb{N} , $x \rho y \leftrightarrow x + y$ is even.

On the set of all lines in the plane, $x \rho y \leftrightarrow x$ is parallel to y or coincides with y .

On $\{0, 1\}$, $x \rho y \leftrightarrow x = y^2$.

On $\{x \mid x \text{ is a student in your class}\}$, $x \rho y \leftrightarrow x$ sits in the same row as y .

On $\{1, 2, 3\}$, $\rho = \{(1, 1), (2, 2), (3, 3), (1, 2), (2, 1)\}$.

We can illustrate an important feature of an equivalence relation on a set by looking at $S = \{x \mid x \text{ is a student in your class}\}$, $x \rho y \leftrightarrow$ “ x sits in the same row as y .” Let’s group together all those students in set S who are related to one another. We come up with Figure 5.5. We have partitioned the set S into subsets in such a way that everyone in the class belongs to one and only one subset.

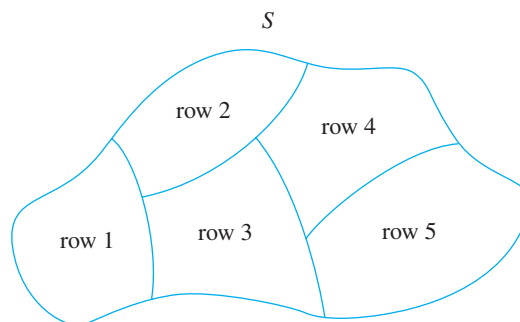


Figure 5.5

DEFINITION PARTITION OF A SET

A **partition** of a set S is a collection of nonempty disjoint subsets of S whose union equals S .

Any equivalence relation, as we will see, partitions the set on which it is defined. The subsets making up the partition, often called the **blocks** of the partition, are formed by grouping together related elements, like the students in the classroom.

For ρ an equivalence relation on a set S and $x \in S$, we let $[x]$ denote the set of all members of S to which x is related, called the **equivalence class** of x . Thus,

$$[x] = \{y \mid y \in S \wedge x \rho y\}$$

(Because ρ is symmetric, we could just as well have said that $[x] = \{y \mid y \in S \wedge y \rho x\}$.)

EXAMPLE 12

In the case where $x\rho y \leftrightarrow$ “ x sits in the same row as y ,” suppose that John, Chuck, Jose, Judy, and Ted all sit in row 3. Then $[\text{John}] = \{\text{John, Chuck, Jose, Judy, Ted}\}$. Also $[\text{John}] = [\text{Ted}] = [\text{Judy}]$, and so on. These are not distinct classes, but the same class with multiple names. An equivalence class can take its name from any of the elements in it. ●

Now we state the result about equivalence relations and partitions. For some practice with formal theorems and proofs, we give this result as a formal theorem, then analyze the structure of the proof and complete part of the proof.

● **THEOREM ON EQUIVALENCE RELATIONS AND PARTITIONS**

An equivalence relation ρ on a set S determines a partition of S , and a partition of a set S determines an equivalence relation on S .

Partial Proof: The theorem makes two separate statements:

- a. An equivalence relation on S determines a partition of S .
- b. A partition of S determines an equivalence relation on S .

To prove part (a), we must show that the distinct equivalence classes of members of S under equivalence relation ρ satisfy the definition of a partition. To satisfy the definition of a partition, we must show that

- i. the union of these distinct classes equals S .
- ii. the distinct classes are disjoint.

To prove part (a. i), we must show something about the union of the distinct equivalence classes formed by ρ . Equivalence classes are sets of elements of S , so their union is a set; let's denote this set by U . We must show that $U = S$, which is a set equality. To prove this set equality, we will prove set inclusion in each direction; in other words,

1. $U \subseteq S$
2. $S \subseteq U$

For this part of the proof, we are finally down to two small statements that are easy to prove, as follows:

a.i.1: Let $x \in U$. Then x belongs to an equivalence class. Every equivalence class is a subset of S , so $x \in S$.

a.i.2: Let $x \in S$. Then $x\rho x$ (reflexivity of ρ); thus, $x \in [x]$, and every member of S belongs to some equivalence class, hence to the union of classes U .

This completes the proof of part (a.i). For part (a.ii), let $[x]$ and $[z]$ be two equivalence classes. We want to show that distinct classes are disjoint, or

$$[x] \neq [z] \rightarrow [x] \cap [z] = \emptyset \quad (\text{a.ii})$$

If we assume that $[x] \neq [z]$, we must then show that $[x] \cap [z]$ does *not* contain anything, which might be hard to do. So instead, we'll prove the contrapositive of a. ii:

$$[x] \cap [z] \neq \emptyset \rightarrow [x] = [z] \quad (\text{contrapositive of a.ii})$$

Therefore, we assume that $[x] \cap [z] \neq \emptyset$ and that there is a $y \in S$ such that $y \in [x] \cap [z]$. What does this tell us?

$$\begin{array}{ll} y \in [x] \cap [z] & (\text{assumption}) \\ y \in [x], y \in [z] & (\text{definition of } \cap) \\ x \rho y, z \rho y & (\text{definition of } [x] \text{ and } [z]) \\ x \rho y, y \rho z & (\text{symmetry of } \rho) \\ x \rho z & (\text{transitivity of } \rho) \end{array}$$

Now we can show that $[x] = [z]$ by proving set inclusion in each direction:

3. $[z] \subseteq [x]$
4. $[x] \subseteq [z]$

To show (3), $[z] \subseteq [x]$, let $q \in [z]$ (we know $[z] \neq \emptyset$ because $y \in [z]$.) Then

$$\begin{array}{ll} z \rho q & (\text{definition of } [z]) \\ x \rho z & (\text{from above}) \\ x \rho q & (\text{transitivity of } \rho) \\ q \in [x] & (\text{definition of } [x]) \\ [z] \subseteq [x] & (\text{definition of } \subseteq) \end{array}$$

Practice 12 asks for a proof of (4), $[x] \subseteq [z]$. Once this proof is supplied, it completes (3) and (4), which leads to the conclusion $[x] = [z]$. This completes the proof of the contrapositive of part (a. ii) and therefore proves part (a. ii), which in turn completes the proof of part (a). Whew!

Practice 13 asks for a proof of part b.

End of Partial Proof

PRACTICE 12 For the foregoing argument, supply the proof that $[x] \subseteq [z]$. ■

PRACTICE 13 Prove part (b) of the theorem. Given a partition of a set S , define a relation ρ by

$$x \rho y \leftrightarrow x \text{ is in the same block of the partition as } y$$

and show that ρ is an equivalence relation on S , that is, show that ρ is reflexive, symmetric, and transitive. ■

EXAMPLE 13 The equivalence relation on \mathbb{N} given by

$$x \rho y \leftrightarrow x + y \text{ is even}$$

partitions \mathbb{N} into two equivalence classes. If x is an even number, then for any even number y , $x + y$ is even and $y \in [x]$. All even numbers form one class. If x is an odd number and y is any odd number, $x + y$ is even and $y \in [x]$. All odd numbers form the second class. The partition can be pictured as in Figure 5.6. Notice again that an equivalence class may have more than one name, or representative. In this example, $[2] = [8] = [1048]$, and so on; $[1] = [17] = [947]$, and so on.

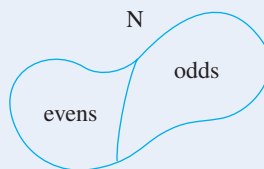


Figure 5.6

PRACTICE 14 For each of the following equivalence relations, describe the corresponding equivalence classes.

- On the set of all lines in the plane, $x\rho y \leftrightarrow x$ is parallel to y or x coincides with y .
- On the set \mathbb{N} , $x\rho y \leftrightarrow x = y$.
- On $\{1, 2, 3\}$, $\rho = \{(1, 1), (2, 2), (3, 3), (1, 2), (2, 1)\}$.

Partitioning a set into equivalence classes is helpful because it is often convenient to go up one level of abstraction and treat the classes themselves as entities. We will conclude this section with two important examples where this is the case (you actually saw the first example somewhere around the fourth grade).

EXAMPLE 14

Let $S = \{a/b \mid a, b \in \mathbb{Z}, b \neq 0\}$. S is therefore the set of all fractions. The fractions $1/2$ and $2/4$ are different fractions—they have different numerators and different denominators, but they are said to be “equivalent.” Formally, a/b is equivalent to c/d , denoted by $a/b \sim c/d$, if and only if $ad = bc$. We will show that the binary relation \sim on S is indeed an equivalence relation. First, $a/b \sim a/b$ because $ab = ba$. Also, if $a/b \sim c/d$ then $ad = bc$, or $cb = da$ and $c/d \sim a/b$. Hence, \sim is reflexive and symmetric. To show that \sim is transitive, let $a/b \sim c/d$ and $c/d \sim e/f$. Then $ad = bc$ and $cf = de$. Multiplying the first equation by f and the second by b , we get $adf = bcf$ and $bcf = bde$. Therefore, $adf = bde$, or $af = be$ (why is it legitimate to divide by d here?). Thus, $a/b \sim e/f$, and \sim is transitive. Some sample equivalence classes of S formed by this equivalence relation are

$$\left[\frac{1}{2}\right] = \left\{ \dots, \frac{-3}{-6}, \frac{-2}{-4}, \frac{-1}{-2}, \frac{1}{2}, \frac{2}{4}, \frac{3}{6}, \dots \right\}$$

$$\left[\frac{3}{10}\right] = \left\{ \dots, \frac{-9}{-30}, \frac{-6}{-20}, \frac{-3}{-10}, \frac{3}{10}, \frac{6}{20}, \frac{9}{30}, \dots \right\}$$

The set \mathbb{Q} of rational numbers can be regarded as the set of all equivalence classes of S . A single rational number, such as $[1/2]$, has many fractions representing it, although we customarily use the reduced fractional representation. When we add two rational numbers, such as $[1/2] + [3/10]$, we look for representatives

from the classes having the same denominator and add those representatives. Our answer is the class to which the resulting sum belongs, and we usually name the class by using a reduced fraction. Thus, to add $[1/2] + [3/10]$, we represent $[1/2]$ by $5/10$ and $[3/10]$ by $3/10$. The sum of $5/10$ and $3/10$ is $8/10$, and $[8/10]$ is customarily named $[4/5]$. This procedure is so familiar that it is generally written as $1/2 + 3/10 = 4/5$; nonetheless, equivalence classes of fractions are being manipulated by means of representatives. ●

EXAMPLE 15

We will define a binary relation of **congruence modulo 4** on the set \mathbb{Z} of integers. An integer x is congruent modulo 4 to y , symbolized by $x \equiv_4 y$, or $x \equiv y \pmod{4}$, if $x - y$ is an integral multiple of 4. Congruence modulo 4 is an equivalence relation on \mathbb{Z} . To construct the equivalence classes, note that $[0]$, for example, will contain all integers differing from 0 by a multiple of 4, such as 4, 8, -12 , and so on. The distinct equivalence classes are

$$[0] = \{\dots, -8, -4, 0, 4, 8, \dots\}$$

$$[1] = \{\dots, -7, -3, 1, 5, 9, \dots\}$$

$$[2] = \{\dots, -6, -2, 2, 6, 10, \dots\}$$

$$[3] = \{\dots, -5, -1, 3, 7, 11, \dots\}$$
 ●

There is nothing special about the choice of 4 in Example 15; we can give a definition for **congruence modulo n** for any positive integer n .

● **DEFINITION CONGRUENCE MODULO n**

For integers x and y and positive integer n ,

$$x \equiv y \pmod{n} \text{ if } x - y \text{ is an integral multiple of } n$$

This binary relation is an equivalence relation on \mathbb{Z} for any positive integer n (see Exercise 46). This equivalence relation and the resulting equivalence classes can be used for integer arithmetic on a computer. An integer is stored as a sequence of bits (0s and 1s) within a single memory location. Each computer allocates a fixed number of bits for a single memory location (this number varies depending on the architecture of the computer—how its memory space is laid out). The larger the integer, the more bits required to represent it. Therefore each machine has some limit on the size of the integers it can store. Suppose that $n - 1$ is the maximum size and that x and y are integer values with $0 \leq x \leq n - 1$, $0 \leq y \leq n - 1$. If the sum $x + y$ exceeds the maximum size, it cannot be stored. As an alternative, the computer may perform **addition modulo n** and find the remainder r when $x + y$ is divided by n .

The equation

$$x + y = qn + r, 0 \leq r < n$$

symbolizes this division, where q is the quotient and r is the remainder. This equation may be written as

$$(x + y) - r = qn$$

which shows that $(x + y) - r$ is an integral multiple of n , or that $(x + y) \equiv r \pmod{n}$. The integer r may not be $x + y$, but it is in the equivalence class $[x + y]$, and since $0 \leq r < n$, it is also in the range of integers that can be stored. (The system may or may not issue an *integer overflow* message if $x + y$ is too large to store and addition modulo n must be used.) The situation is analogous to your car's odometer, which records mileage modulo 100,000; when mileage reaches 102,758, for example, it is displayed on the odometer as 2,758.

PRACTICE 15 What are the equivalence classes corresponding to the relation of congruence modulo 5 on \mathbb{Z} ?

PRACTICE 16 If 4 is the maximum integer that can be stored on a (micromicro) computer, what will be stored for the value $3 + 4$ if addition modulo 5 is used?

Table 5.1 summarizes important features of partial orderings and equivalence relations.

Type of Binary Relation	Reflexive	Symmetric	Antisymmetric	Transitive	Important Feature
Partial ordering	Yes	No	Yes	Yes	Predecessors and successors
Equivalence relation	Yes	Yes	No	Yes	Determines a partition

SECTION 5.1 REVIEW

TECHNIQUES

- Test an ordered pair for membership in a binary relation.
- Ⓜ Test a binary relation for reflexivity, symmetry, antisymmetry, and transitivity.
- Find the reflexive, symmetric, and transitive closure of a relation.

- Draw the Hasse diagram for a finite partially ordered set.
- Find least, minimal, greatest, and maximal elements in a partially ordered set.
- Ⓜ Find the equivalence classes associated with an equivalence relation.

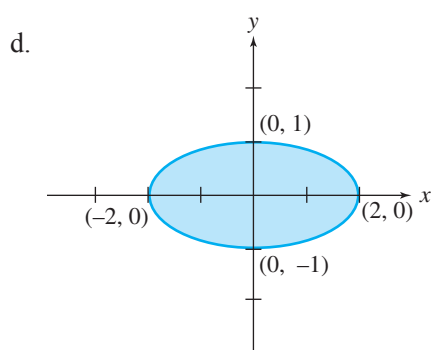
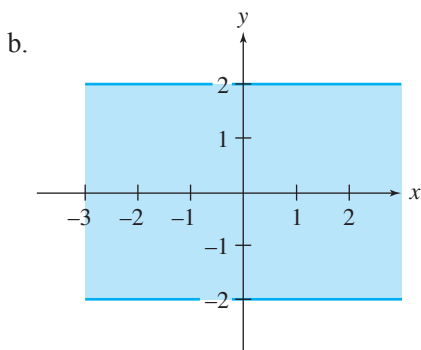
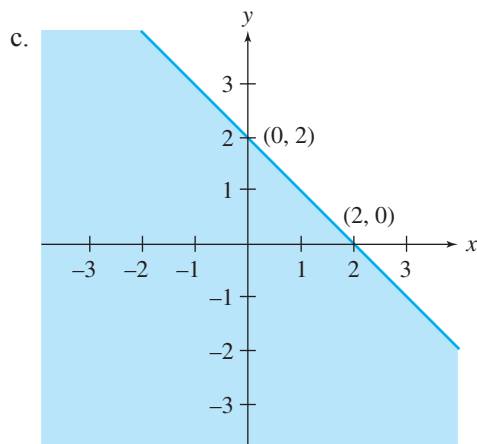
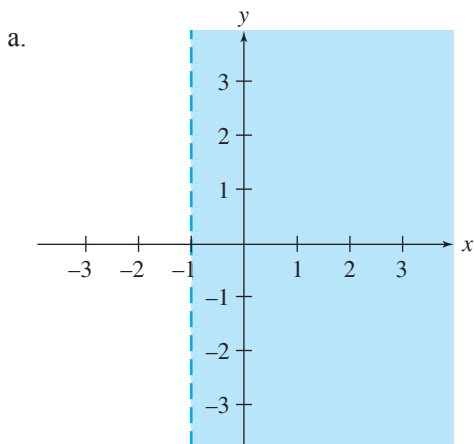
MAIN IDEAS

- A binary relation on a set S is formally a subset of $S \times S$; the distinctive relationship satisfied by the relation's members often has a verbal description as well.
- Operations on binary relations on a set include union, intersection, and complementation.
- Binary relations can have properties of reflexivity, symmetry, transitivity, and antisymmetry.
- Finite partially ordered sets can be represented graphically.
- An equivalence relation on a set S determines a partition of S , and conversely. The blocks of the partition are equivalence classes, which may themselves be treated as entities.

EXERCISES 5.1

- For each of the following binary relations ρ on \mathbb{N} , decide which of the given ordered pairs belong to ρ .
 - $x\rho y \leftrightarrow x + y < 7$; (1, 3), (2, 5), (3, 3), (4, 4)
 - $x\rho y \leftrightarrow x = y + 2$; (0, 2), (4, 2), (6, 3), (5, 3)
 - $x\rho y \leftrightarrow 2x + 3y = 10$; (5, 0), (2, 2), (3, 1), (1, 3)
 - $x\rho y \leftrightarrow y$ is a perfect square; (1, 1), (4, 2), (3, 9), (25, 5)
- For each of the following binary relations ρ on \mathbb{Z} , decide which of the given ordered pairs belong to ρ .
 - $x\rho y \leftrightarrow x|y$; (2, -6), (3, 5), (8, 4), (4, 8)
 - $x\rho y \leftrightarrow x$ and y are relatively prime; (5, 8), (9, 16), (6, 8), (8, 21)
 - $x\rho y \leftrightarrow \gcd(x, y) = 7$; (28, 14), (7, 7), (10, 5), (21, 14)
 - $x\rho y \leftrightarrow x^2 + y^2 = z^2$ for some integer z ; (1, 0), (3, 9), (2, 2), (-3, 4)
 - $x\rho y \leftrightarrow x$ is a number from the Fibonacci sequence; (4, 3), (7, 6), (7, 12), (20, 20)
- Decide which of the given items satisfy the relation.
 - ρ a binary relation on \mathbb{Z} , $x\rho y \leftrightarrow x = -y$; (1, -1), (2, 2), (-3, 3), (-4, -4)
 - ρ a binary relation on \mathbb{N} , $x\rho y \leftrightarrow x$ is prime; (19, 7), (21, 4), (33, 13), (41, 16)
 - ρ a binary relation on \mathbb{Q} , $x\rho y \leftrightarrow x \leq 1/y$; (1, 2), (-3, -5), (-4, 1/2), (1/2, 1/3)
 - ρ a binary relation on $\mathbb{N} \times \mathbb{N}$, $(x, y)\rho(u, v) \leftrightarrow x + u = y + v$; ((1, 2), (3, 2)), ((4, 5), (0, 1))
- Decide which of the given items satisfy the relation.
 - ρ a binary relation on $S \times C$, where $S = \{\text{states in the United States}\}$, $C = \{\text{cities in the United States}\}$, $x\rho y \leftrightarrow y$ is the capital of x ; (Indiana, Indianapolis), (Illinois, Chicago), (Kansas, Kansas City), (Kentucky, Louisville), (North Dakota, Bismarck)
 - ρ a binary relation on $A \times P$, where $A = \{\text{artists}\}$, $P = \{\text{paintings}\}$, $x\rho y \leftrightarrow x$ painted y ; (DaVinci, Mona Lisa), (Grant Wood, American Gothic), (Remington, Ridden Down), (Picasso, Blue Dancers), (van Gogh, Starry Night)
 - ρ a binary relation on $C \times M$, where $C = \{\text{composers}\}$, $M = \{\text{music}\}$, $x\rho y \leftrightarrow x$ composed y ; (Bernstein, West Side Story), (Presley, Blue Suede Shoes), (Gershwin, Rhapsody in Blue), (Beethoven, Moonlight Sonata), (Rogers and Hammerstein, Phantom of the Opera)
 - ρ a binary relation on $A \times B$, where $A = \{\text{authors}\}$, $B = \{\text{books}\}$, $x\rho y \leftrightarrow x$ wrote y ; (Hemingway, The Old Man and the Sea), (Sawyer, Huckleberry Finn) (Poe, Moby Dick), (Orwell, 1984), (Tolstoy, Crime and Punishment)
- For each of the following binary relations on \mathbb{R} , draw a figure to show the region of the plane it describes.
 - $x\rho y \leftrightarrow y \leq 2$
 - $x\rho y \leftrightarrow x = y - 1$
 - $x\rho y \leftrightarrow x^2 + y^2 \leq 25$
 - $x\rho y \leftrightarrow x \geq y$

6. For each of the accompanying figures, give the binary relation on \mathbb{R} that describes the shaded area.



7. Identify each relation on \mathbb{N} as one-to-one, one-to-many, many-to-one, or many-to-many.

- $\rho = \{(1, 2), (1, 4), (1, 6), (2, 3), (4, 3)\}$
- $\rho = \{(9, 7), (6, 5), (3, 6), (8, 5)\}$
- $\rho = \{(12, 5), (8, 4), (6, 3), (7, 12)\}$
- $\rho = \{(2, 7), (8, 4), (2, 5), (7, 6), (10, 1)\}$

8. Identify each of the following relations on S as one-to-one, one-to-many, many-to-one, or many-to-many.

- $S = \mathbb{N}$
 $x\rho y \leftrightarrow x = y + 1$
- $S =$ set of all women in Vicksburg
 $x\rho y \leftrightarrow x$ is the daughter of y
- $S = \wp(\{1, 2, 3\})$
 $A\rho B \leftrightarrow |A| = |B|$
- $S = \mathbb{R}$
 $x\rho y \leftrightarrow x = 5$

9. Let ρ and σ be binary relations on \mathbb{N} defined by $x\rho y \leftrightarrow$ “ x divides y ,” $x\sigma y \leftrightarrow 5x \leq y$. Decide which of the given ordered pairs satisfy the following relations.

- $\rho \cup \sigma$; $(2, 6), (3, 17), (2, 1), (0, 0)$
- $\rho \cap \sigma$; $(3, 6), (1, 2), (2, 12)$
- ρ' ; $(1, 5), (2, 8), (3, 15)$
- σ' ; $(1, 1), (2, 10), (4, 8)$

10. Both ρ and σ are binary relations from P to C where $P = \{\text{people in the United States}\}$, $C = \{\text{cities in the United States}\}$, $x\rho y \leftrightarrow x$ lives in y , and $x\sigma y \leftrightarrow x$ works in y . Describe each of the following relations.
- $\rho \cap \sigma$
 - $\rho \cup \sigma$
 - $\rho \cap \sigma'$
 - $\rho' \cap \sigma$
11. Let $S = \{1, 2, 3\}$. Test the following binary relations on S for reflexivity, symmetry, antisymmetry, and transitivity.
- $\rho = \{(1, 3), (3, 3), (3, 1), (2, 2), (2, 3), (1, 1), (1, 2)\}$
 - $\rho = \{(1, 1), (3, 3), (2, 2)\}$
 - $\rho = \{(1, 1), (1, 2), (2, 3), (3, 1), (1, 3)\}$
 - $\rho = \{(1, 1), (1, 2), (2, 3), (1, 3)\}$
12. Let $S = \{0, 1, 2, 4, 6\}$. Test the following binary relations on S for reflexivity, symmetry, antisymmetry, and transitivity.
- $\rho = \{(0, 0), (1, 1), (2, 2), (4, 4), (6, 6), (0, 1), (1, 2), (2, 4), (4, 6)\}$
 - $\rho = \{(0, 1), (1, 0), (2, 4), (4, 2), (4, 6), (6, 4)\}$
 - $\rho = \{(0, 1), (1, 2), (0, 2), (2, 0), (2, 1), (1, 0), (0, 0), (1, 1), (2, 2)\}$
 - $\rho = \{(0, 0), (1, 1), (2, 2), (4, 4), (6, 6), (4, 6), (6, 4)\}$
 - $\rho = \emptyset$
13. Test the following binary relations on the given sets S for reflexivity, symmetry, antisymmetry, and transitivity.
- $S = \mathbb{Q}$
 $x\rho y \leftrightarrow |x| \leq |y|$
 - $S = \mathbb{Z}$
 $x\rho y \leftrightarrow x - y$ is an integral multiple of 3
 - $S = \mathbb{N}$
 $x\rho y \leftrightarrow x \cdot y$ is even
 - $S = \mathbb{N}$
 $x\rho y \leftrightarrow x$ is odd
 - $S =$ set of all squares in the plane
 $S_1\rho S_2 \leftrightarrow$ length of side of $S_1 =$ length of side of S_2
14. Test the following binary relations on the given sets S for reflexivity, symmetry, antisymmetry, and transitivity.
- $S =$ set of all finite-length strings of characters
 $x\rho y \leftrightarrow$ number of characters in $x =$ number of characters in y
 - $S = \{0, 1, 2, 3, 4, 5\}$
 $x\rho y \leftrightarrow x + y = 5$
 - $S = \wp(\{1, 2, 3, 4, 5, 6, 7, 8, 9\})$
 $A\rho B \leftrightarrow |A| = |B|$
 - $S = \wp(\{1, 2, 3, 4, 5, 6, 7, 8, 9\})$
 $A\rho B \leftrightarrow |A| \neq |B|$
 - $S = \mathbb{N} \times \mathbb{N}$
 $(x_1, y_1)\rho(x_2, y_2) \leftrightarrow x_1 \leq x_2$ and $y_1 \geq y_2$

15. Which of the binary relations of Exercise 13 are equivalence relations? For each equivalence relation, describe the associated equivalence classes.
16. Which of the binary relations of Exercise 14 are equivalence relations? For each equivalence relation, describe the associated equivalence classes.
17. Test the following binary relations on the given sets S for reflexivity, symmetry, antisymmetry, and transitivity.
- $S = \mathbb{Z}$
 $x\rho y \leftrightarrow x + y$ is a multiple of 5
 - $S = \mathbb{Z}$
 $x\rho y \leftrightarrow x < y$
 - $S =$ set of all finite-length binary strings
 $x\rho y \leftrightarrow x$ is a prefix of y
 - $S =$ set of all finite-length binary strings
 $x\rho y \leftrightarrow x$ has the same number of 1s as y
18. Test the following binary relations on the given sets S for reflexivity, symmetry, antisymmetry, and transitivity.
- $S = \mathbb{Z}$
 $x\rho y \leftrightarrow x = ky$ for some integer k
 - $S = \mathbb{Z}$
 $x\rho y \leftrightarrow$ there is a prime number p such that $p|x$ and $p|y$
 - $S = \wp(\{1, 2, 3, 4, 5, 6, 7, 8, 9\})$
 $A\rho B \leftrightarrow A \cap B = \emptyset$
 - $S = \wp(\{1, 2, 3, 4, 5, 6, 7, 8, 9\})$
 $A\rho B \leftrightarrow A = B'$
19. Let S be the set of people in the United States. Test the following binary relations on S for reflexivity, symmetry, antisymmetry, and transitivity.
- $x\rho y \leftrightarrow x$ is at least as tall as y .
 - $x\rho y \leftrightarrow x$ is taller than y .
 - $x\rho y \leftrightarrow x$ is the same height as y .
 - $x\rho y \leftrightarrow x$ is a child of y .
20. Let S be the set of people in the United States. Test the following binary relations on S for reflexivity, symmetry, antisymmetry, and transitivity.
- $x\rho y \leftrightarrow x$ is the husband of y .
 - $x\rho y \leftrightarrow x$ is the spouse of y .
 - $x\rho y \leftrightarrow x$ has the same parents as y .
 - $x\rho y \leftrightarrow x$ is the brother of y .
21. For each case, think of a set S and a binary relation ρ on S (different from any in the examples or problems) satisfying the given conditions.
- ρ is reflexive and symmetric but not transitive.
 - ρ is reflexive and transitive but not symmetric.
 - ρ is not reflexive or symmetric but is transitive.
 - ρ is reflexive but neither symmetric nor transitive.

22. Let ρ and σ be binary relations on a set S .
- If ρ and σ are reflexive, is $\rho \cup \sigma$ reflexive? Is $\rho \cap \sigma$ reflexive?
 - If ρ and σ are symmetric, is $\rho \cup \sigma$ symmetric? Is $\rho \cap \sigma$ symmetric?
 - If ρ and σ are antisymmetric, is $\rho \cup \sigma$ antisymmetric? Is $\rho \cap \sigma$ antisymmetric?
 - If ρ and σ are transitive, is $\rho \cup \sigma$ transitive? Is $\rho \cap \sigma$ transitive?
23. Find the reflexive, symmetric, and transitive closure of each of the relations in Exercise 11.
24. Find the reflexive, symmetric, and transitive closure of each of the relations in Exercise 12.
25. Given the following binary relation

$S =$ set of all cities in the country

$x\rho y \leftrightarrow$ Take-Your-Chance Airlines flies directly from x to y

describe in words what the transitive closure relation would be.

26. Two additional properties of a binary relation ρ are defined as follows:

ρ is *irreflexive* means: $(\forall x)(x \in S \rightarrow (x, x) \notin \rho)$

ρ is *asymmetric* means: $(\forall x)(\forall y)(x \in S \wedge y \in S \wedge (x, y) \in \rho \rightarrow (y, x) \notin \rho)$

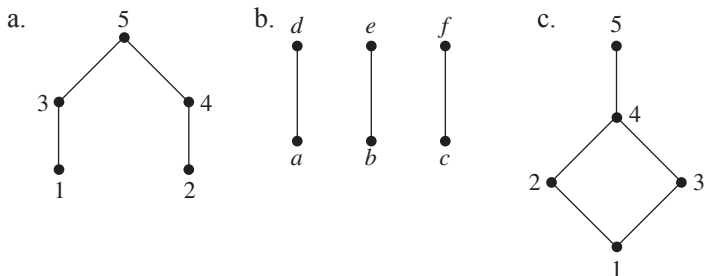
- Give an example of a binary relation ρ on set $S = \{1, 2, 3\}$ that is neither reflexive nor irreflexive.
 - Give an example of a binary relation ρ on set $S = \{1, 2, 3\}$ that is neither symmetric nor asymmetric.
 - Prove that if ρ is an asymmetric relation on a set S , then ρ is irreflexive.
 - Prove that if ρ is an irreflexive and transitive relation on a set S , then ρ is asymmetric.
 - Prove that if ρ is a nonempty, symmetric, and transitive relation on a set S , then ρ is not irreflexive.
27. Does it make sense to look for the irreflexive closure of a relation? (See Exercise 26.) Why or why not?
28. Does it make sense to look for the asymmetric closure of a relation? (See Exercise 26.) Why or why not?
29. Let S be an n -element set. How many different binary relations can be defined on S ? (*Hint*: Recall the formal definition of a binary relation.)
30. Let ρ be a binary relation on a set S . For $A \subseteq S$, define

$$\#A = \{x \mid x \in S \wedge (\forall y)(y \in A \rightarrow x\rho y)\}$$

$$A\# = \{x \mid x \in S \wedge (\forall y)(y \in A \rightarrow y\rho x)\}$$

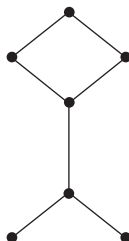
- Prove that if ρ is symmetric, then $\#A = A\#$.
 - Prove that if $A \subseteq B$ then $\#B \subseteq \#A$ and $B\# \subseteq A\#$.
 - Prove that $A \subseteq (\#A)\#$.
 - Prove that $A \subseteq \#(A\#)$.
31. Draw the Hasse diagram for the following partial orderings.
- $S = \{a, b, c\}$
 $\rho = \{(a, a), (b, b), (c, c), (a, b), (b, c), (a, c)\}$
 - $S = \{a, b, c, d\}$
 $\rho = \{(a, a), (b, b), (c, c), (d, d), (a, b), (a, c)\}$
 - $S = \{\emptyset, \{a\}, \{a, b\}, \{c\}, \{a, c\}, \{b\}\}$
 $A\rho B \leftrightarrow A \subseteq B$

32. For Exercise 31, name any least elements, minimal elements, greatest elements, and maximal elements.
33. Let (S, \leq) be a partially ordered set, and let $A \subseteq S$. Prove that the restriction of \leq to A is a partial ordering on A .
34. a. Draw the Hasse diagram for the partial ordering “ x divides y ” on the set $\{2, 3, 5, 7, 21, 42, 105, 210\}$. Name any least elements, minimal elements, greatest elements, and maximal elements. Name a totally ordered subset with four elements.
 b. Draw the Hasse diagram for the partial ordering “ x divides y ” on the set $\{3, 6, 9, 18, 54, 72, 108, 162\}$. Name any least elements, minimal elements, greatest elements, and maximal elements. Name any unrelated elements.
35. Draw the Hasse diagram for each of the two partially ordered sets.
 a. $S = \{1, 2, 3, 5, 6, 10, 15, 30\}$ b. $S = \wp(\{1, 2, 3\})$
 $x \rho y \leftrightarrow x$ divides y $A \rho B \leftrightarrow A \subseteq B$
 What do you notice about the structure of these two diagrams?
36. For each Hasse diagram of a partial ordering in the accompanying figure, list the ordered pairs that belong to the relation.



37. Let (S, ρ) and (T, σ) be two partially ordered sets. A relation μ on $S \times T$ is defined by $(s_1, t_1) \mu (s_2, t_2) \leftrightarrow s_1 \rho s_2$ and $t_1 \sigma t_2$. Show that μ is a partial ordering on $S \times T$.
38. Let ρ be a binary relation on a set S . Then a binary relation called the inverse of ρ , denoted by ρ^{-1} , is defined by $x \rho^{-1} y \leftrightarrow y \rho x$.
- For $\rho = \{(1, 2), (2, 3), (5, 3), (4, 5)\}$ on the set \mathbb{N} , what is ρ^{-1} ?
 - Prove that if ρ is a reflexive relation on a set S , then ρ^{-1} is reflexive.
 - Prove that if ρ is a symmetric relation on a set S , then ρ^{-1} is symmetric.
 - Prove that if ρ is an antisymmetric relation on a set S , then ρ^{-1} is antisymmetric.
 - Prove that if ρ is a transitive relation on a set S , then ρ^{-1} is transitive.
 - Prove that if ρ is an irreflexive relation on a set S (see Exercise 26), then ρ^{-1} is irreflexive.
 - Prove that if ρ is an asymmetric relation on a set S (see Exercise 26), then ρ^{-1} is asymmetric.
39. Prove that if a binary relation ρ on a set S is reflexive and transitive, then the relation $\rho \cap \rho^{-1}$ is an equivalence relation (see Exercise 38 for the definition of ρ^{-1}).
40. a. Let (S, ρ) be a partially ordered set. Then ρ^{-1} can be defined as in Exercise 38. Show that (S, ρ^{-1}) is a partially ordered set, called the *dual* of (S, ρ) .

- b. If (S, ρ) is a finite, partially ordered set with the Hasse diagram shown, draw the diagram of the dual of (S, ρ) .



- c. Let (S, ρ) be a totally ordered set and let $X = \{(x, x) | x \in S\}$. Show that the set difference $\rho^{-1} - X$ equals the set ρ' .
41. A computer program is to be written that will generate a dictionary or the index for a book. We will assume a maximum length of n characters per word. Thus, we are given a set S of words of length at most n , and we want to produce a linear list of these words arranged in alphabetical order. There is a natural total ordering \leq on alphabetic characters ($a < b$, $b < c$, etc.), and we will assume our words contain only alphabetic characters. We want to define a total ordering \leq on S called a *lexicographical ordering* that will arrange the members of S alphabetically. The idea is to compare two words X and Y character by character, passing over equal characters. If at any point the X character alphabetically precedes the corresponding Y character, then X precedes Y ; if all characters in X are equal to the corresponding Y characters but we run out of characters in X before characters in Y , then X precedes Y . Otherwise, Y precedes X .

Formally, let $X = (x_1, x_2, \dots, x_j)$ and $Y = (y_1, y_2, \dots, y_k)$ be members of S with $j \leq k$. Let β (for blank) be a new symbol, and fill out X with $k - j$ blanks on the right. X can now be written (x_1, x_2, \dots, x_k) . Let β precede any alphabetical character. Then $X \leq Y$ if

$$x_1 \neq y_1 \text{ and } x_1 \leq y_1$$

or

$$\begin{aligned} x_1 = y_1, x_2 = y_2, \dots, x_m = y_m (m \leq k) \\ x_{m+1} \neq y_{m+1} \text{ and } x_{m+1} \leq y_{m+1} \end{aligned}$$

Otherwise, $Y \leq X$.

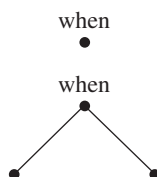
Note that because the ordering \leq on alphabetical characters is a total ordering, if $Y \leq X$ by “otherwise,” then there exists $m \leq k$ such that $x_1 = y_1, x_2 = y_2, \dots, x_m = y_m, x_{m+1} \neq y_{m+1}$ and $y_{m+1} \leq x_{m+1}$.

Show that \leq on S as defined above is a total ordering.

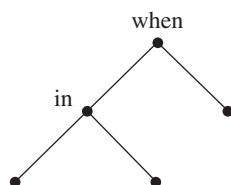
42. Apply the total ordering described in Exercise 41 to the words *boo*, *bug*, *be*, *bah*, and *bugg*. Note why each word precedes the next.
43. Exercise 41 discusses a total ordering on a set of words of length at most n that will produce a linear list in alphabetical order. Suppose we want to generate a list of all the distinct words in a text (for example, a compiler must create a symbol table of variable names). As in Exercise 41, we assume that the words contain only alphabetical characters because there is a natural precedence relation already existing ($a < b$, $b < c$, and so on). If numeric or special characters are involved, they must be assigned a

precedence relation with alphabetical characters (the collating sequence must be determined). If we list words alphabetically, it is a fairly quick procedure to decide whether a word currently being processed is new, but to fit the new word into place, all successive words must be moved one unit down the line. If the words are listed in the order in which they are processed, new words are simply tacked onto the end and no rearranging is necessary, but each word being processed has to be compared with each member of the list to determine if it is new. Thus, both logical linear lists have disadvantages.

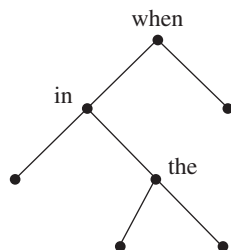
We describe a structure called a *binary search tree*; using this structure, a search process called a *binary tree search* can usually determine quickly whether a word is new and, if it is, no juggling is required to fit it into place, thus eliminating the disadvantages of both linear list structures described earlier. Suppose we want to process the phrase “when in the course of human events.” The first word in the text is used to label the first node of a graph. Once a node is labeled, it drops down a left and right arc, putting two unlabeled nodes below the one just labeled.



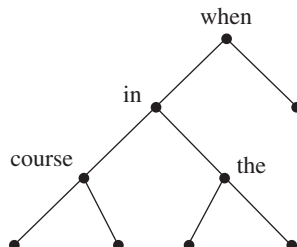
When the next word in the text is processed, it is compared with the first node. When the word being processed alphabetically precedes the label of a node, the left arc is taken; when the word follows the label alphabetically, the right arc is taken. The word becomes the label of the first unlabeled node it reaches. (If the word equals a node label, it is a duplicate, so the next word in the text is processed.) This procedure continues for the entire text. Thus,



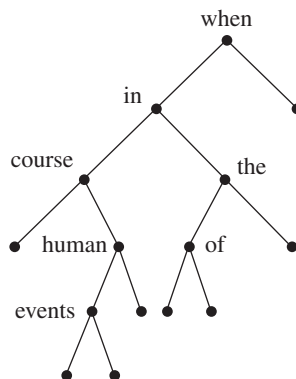
then



then



until finally



By traversing the nodes of this graph in the proper order (described by always processing the left nodes below a node first, then the node, then the right nodes below it), an alphabetical listing “course, events, human, in, of, the, when” is produced.

- a. This type of graph is called a *tree*. (Unlabeled nodes and arcs to unlabeled nodes usually are not shown.) Turned upside down, the graph can be viewed as the Hasse diagram of a partial ordering \leq . What would be the least element? Would there be a greatest element? Which of the following ordered pairs would belong to \leq : (in, of), (the, of), (in, events), (course, of)?

Here the tree structure contains more information than the partial ordering, because we are interested in not only whether a word w_1 precedes a word w_2 in the partial ordering sense but also whether w_2 is to the left or right of w_1 .

- b. Build a binary search tree for the phrase “Old King Cole was a merry old soul.” Eliminate unlabeled nodes. Considering the upside-down graph as the Hasse diagram of a partial ordering, name the maximal elements.
44. The alphabetical ordering defined in Exercise 41 can be applied to words of any finite length. If we define A^* to be the set of all finite-length “words” (strings of characters, not necessarily meaningful) from the English alphabet, then the alphabetical ordering on A^* has all words composed only of the letter A preceding all other words. Thus, all the words in the infinite list

$$a, aa, aaa, aaaa, \dots$$

precede words such as “ b ” or “ $aaaaaab$.” Therefore this list does not enumerate A^* , because we can never count up to any words with any characters other than a . However, the set A^* is denumerable. Write a partial enumeration of A^* by ordering words by length (all words of length 1 precede all words of length 2, and so on) and then alphabetically ordering words of the same length.

45. a. For the equivalence relation $\rho = \{(a, a), (b, b), (c, c), (a, c), (c, a)\}$, what is the set $[a]$? Does it have any other names?
- b. For the equivalence relation $\rho = \{(1, 1), (2, 2), (1, 2), (2, 1), (1, 3), (3, 1), (3, 2), (2, 3), (3, 3), (4, 4), (5, 5), (4, 5), (5, 4)\}$ what is the set $[3]$? What is the set $[4]$?
46. Prove that for any positive integer n , congruence modulo n is an equivalence relation on the set \mathbb{Z} .
47. For the equivalence relation of congruence modulo 2 on the set \mathbb{Z} , what is the set $[1]$?
48. For the equivalence relation of congruence modulo 5 on the set \mathbb{Z} , what is the set $[-3]$?
49. Assume that $x \equiv y \pmod{n}$ and $z \equiv w \pmod{n}$ for some positive integer n . Prove that
- $x + z \equiv y + w \pmod{n}$
 - $x - z \equiv y - w \pmod{n}$
 - $x \cdot z \equiv y \cdot w \pmod{n}$
 - $x^s \equiv y^s \pmod{n}$ for $s \geq 1, n \geq 2$

50. Let p be a prime number. Prove that $x^2 \equiv y^2 \pmod{p}$ if and only if $x \equiv y \pmod{p}$ or $x \equiv -y \pmod{p}$.
51. a. Given the partition $\{1, 2\}$ and $\{3, 4\}$ of the set $S = \{1, 2, 3, 4\}$, list the ordered pairs in the corresponding equivalence relation.
 b. Given the partition $\{a, b, c\}$ and $\{d, e\}$ of the set $S = \{a, b, c, d, e\}$, list the ordered pairs in the corresponding equivalence relation.
52. Let S be the set of all books in the library. Let ρ be a binary relation on S defined by $x\rho y \leftrightarrow$ “the color of x ’s cover is the same as the color of y ’s cover.” Show that ρ is an equivalence relation on S and describe the resulting equivalence classes.
53. Let $S = \mathbb{N}$ and let ρ be a binary relation on S defined by $x\rho y \leftrightarrow x^2 - y^2$ is even. Show that ρ is an equivalence relation on S and describe the resulting equivalence classes.
54. Let $S = \mathbb{R}$ and let ρ be a binary relation on S defined by $x\rho y \leftrightarrow x - y$ is an integer.
 a. Show that ρ is an equivalence relation on S .
 b. List 5 values that belong to $[1.5]$.
55. Let $S = \mathbb{N} \times \mathbb{N}$ and let ρ be a binary relation on S defined by $(x, y)\rho(z, w) \leftrightarrow y = w$. Show that ρ is an equivalence relation on S and describe the resulting equivalence classes.
56. Let $S = \mathbb{N} \times \mathbb{N}$ and let ρ be a binary relation on S defined by $(x, y)\rho(z, w) \leftrightarrow x + y = z + w$. Show that ρ is an equivalence relation on S and describe the resulting equivalence classes.
57. Let S be the set of all binary strings of length 8 and let ρ be a binary relation on S defined by $x\rho y \leftrightarrow$ x starts with the same bit value (0 or 1) as y and x ends with the same bit value (0 or 1) as y .
 a. Show that ρ is an equivalence relation on S .
 b. How many strings are in the set S ?
 c. Into how many equivalence classes does ρ partition S ? Explain your answer.
 d. How many strings are in each equivalence class?
58. The documentation for the Java programming language recommends that when a boolean “equals method” is defined for an object, it should be an equivalence relation. That is, if ρ is defined by $x\rho y \leftrightarrow x.\text{equals}(y)$ for all objects in the class, then ρ should be an equivalence relation. In a graphics application, a programmer creates an object called a point, consisting of two coordinates in the plane. The programmer defines an equals method as follows: If p and q are any two points in the plane, then

$$p.\text{equals}(q) \leftrightarrow \text{the distance from } p \text{ to } q \text{ is } \leq c$$

where c is a small positive number that depends on the resolution of the computer display. Is the programmer’s equals method an equivalence relation? Justify your answer.

59. Let S be the set of all propositional wffs with n statement letters. Let ρ be a binary relation on S defined by $P\rho Q \leftrightarrow$ “ $P \leftrightarrow Q$ is a tautology.” Show that ρ is an equivalence relation on S and describe the resulting equivalence classes. (We have used the notation $P \Leftrightarrow Q$ for $P\rho Q$.)
60. Given two partitions π_1 and π_2 of a set S , π_1 is a *refinement* of π_2 if each block of π_1 is a subset of a block of π_2 . Show that refinement is a partial ordering on the set of all partitions of S .

Exercises 61–72 all deal with partitions on a set.

61. Let P_n denote the total number of partitions of an n -element set, $n \geq 1$. The numbers P_n are called Bell numbers. Compute the following Bell numbers.
 a. P_1
 b. P_2
 c. P_3
 d. P_4

62. From Exercise 61, you might be looking for a closed-form formula giving the value of P_n . Although Bell numbers have been extensively studied, no closed-form formula has been found. Bell numbers can be computed via a recurrence relation. Let P_0 have the value 1. Prove that for $n \geq 1$,

$$P_n = \sum_{k=0}^{n-1} C(n-1, k)P_k$$

(Hint: Use a combinatorial proof instead of an inductive proof. Let x be a fixed but arbitrary member of a set with n elements. In each term of the sum, $n - k$ represents the size of the partition block that contains x .)

63. Use the formula of Exercise 62 to compute P_1 , P_2 , P_3 , and P_4 , and compare your answers to those in Exercise 61.
64. Use the formula of Exercise 62 to compute P_5 and P_6 .
65. Let $S(n, k)$ denote the number of ways to partition a set of n elements into k blocks. The numbers $S(n, k)$ are called *Stirling numbers*.
- Find $S(3, 2)$.
 - Find $S(4, 2)$.
66. Prove that for all $n \geq 1$, $S(n, k)$ satisfies the recurrence relation

$$\begin{aligned} S(n, 1) &= 1 \\ S(n, n) &= 1 \\ S(n+1, k+1) &= S(n, k) + (k+1)S(n, k+1) \text{ for } 1 \leq k \leq n \end{aligned}$$

(Hint: Use a combinatorial proof instead of an inductive proof. Let x be a fixed but arbitrary member of a set with $n + 1$ elements, and put x aside. Partition the remaining set of n elements. A partition of the original set could be obtained either by adding $\{x\}$ as a separate block or by putting x in one of the existing blocks.)

67. Use the formula of Exercise 66 to rework Exercise 65.
68. The recurrence relation of Exercise 66 is similar to Pascal's formula, Equation (1) of Section 4.5. Use this relation to compute the numeric values in the first five rows of *Stirling's triangle*, which begins

$$\begin{array}{ccc} & & S(1, 1) \\ & & S(2, 1) \quad S(2, 2) \\ & S(2, 1) \quad S(3, 2) \quad S(3, 3) \\ & & \vdots \end{array}$$

69. Prove that

$$P_n = \sum_{k=1}^n S(n, k)$$

70. Use the formula of Exercise 69 and Stirling's triangle (Exercise 68) to compute P_1 , P_2 , P_3 , and P_4 .
71. Find the number of ways to distribute 4 different-colored marbles among 3 identical containers so that no container is empty.
72. Find the number of ways in which 5 different jobs can be assigned to 3 identical processors so that each processor gets at least 1 job.

73. Binary relations on a set S are ordered pairs of elements of S . More generally, an n -ary relation on a set S is a set of ordered n -tuples of elements of S . Decide which of the given items satisfy the relation.

a. ρ a unary relation on \mathbb{Z} , $x \in \rho \leftrightarrow x$ is a perfect square

$$25, 39, 49, 62$$

b. ρ a ternary relation on \mathbb{N} , $(x, y, z) \in \rho \leftrightarrow x^2 + y^2 = z^2$

$$(1, 1, 2), (3, 4, 5), (0, 5, 5), (8, 6, 10)$$

c. ρ a 4-ary relation on \mathbb{Z} , $(x, y, z, w) \in \rho \leftrightarrow y = |x|$ and $w \geq x + z^2$

$$(-4, 4, 2, 0), (5, 5, 1, 5), (6, -6, 6, 45), (-6, 6, 0, -2)$$

74. A ternary relation ρ is defined on the set $S = (2, 4, 6, 8)$ by $(x, y, z) \in \rho \leftrightarrow x + y = z$. List the 3-tuples that belong to ρ .

75. If x is a real number, $x \neq 0$, then a number y such that $x \cdot y = 1$ is called the multiplicative inverse of x . Given positive integers x and n , a positive integer y such that $x \cdot y \equiv 1 \pmod{n}$ is called the *modular multiplicative inverse of x modulo n* . But

$$x \cdot y \equiv 1 \pmod{n}$$

$$\leftrightarrow x \cdot y - 1 = kn \text{ where } k \text{ is an integer}$$

$$\leftrightarrow xy - kn = 1$$

$$\leftrightarrow 1 \text{ is a linear combination of } x \text{ and } n$$

$$\leftrightarrow \gcd(x, n) = 1$$

$$\leftrightarrow x \text{ and } n \text{ are relatively prime}$$

Thus, if x and n are not relatively prime, the modular inverse of x does not exist. If they are relatively prime, the modular inverse of x is the positive coefficient of x in the linear combination of x and n that equals 1.

Use the Euclidean algorithm to find the modular multiplicative inverse of 21 modulo 25 (note that 21 and 25 are relatively prime).

76. Use the Euclidean algorithm to find the modular multiplicative inverse of 68 modulo 15 (see Exercise 75).

SECTION 5.2 TOPOLOGICAL SORTING

If ρ is a partial ordering on a set S , then some elements of S are predecessors of other elements. If S is a set of tasks that are to be done, then the idea of x as a predecessor of y can be interpreted literally to mean that task x must be done before task y . Thus partial orderings and Hasse diagrams are natural ways to represent problems in task scheduling.

EXAMPLE 16

Ernie and his brothers run a woodworking shop in the hills of New Hampshire that manufactures rocking chairs with padded cushion seats. The manufacturing process can be broken down into a number of tasks, some of which have certain other tasks as prerequisites. The following table shows the manufacturing tasks for a rocking chair, the prerequisite tasks, and the number of hours required to perform each task.

Task	Prerequisite Tasks	Hours to Perform
1. Selecting wood	None	3.0
2. Carving rockers	1	4.0
3. Carving seat	1	6.0
4. Carving back	1	7.0
5. Carving arms	1	3.0
6. Selecting fabric	None	1.0
7. Sewing cushion	6	2.0
8. Assembling back and seat	3, 4	2.0
9. Attaching arms	5, 8	2.0
10. Attaching rockers	2, 8	3.0
11. Varnishing	9, 10	5.0
12. Adding cushion	7, 11	0.5

We can define a partial ordering on the set of tasks by

$$x \leq y \leftrightarrow \text{task } x = \text{task } y \text{ or task } x \text{ is a prerequisite to task } y$$

It is easy to see that this relation is reflexive, antisymmetric, and transitive. Also,

$$x < y \leftrightarrow \text{task } x \text{ is a prerequisite to task } y$$

In the Hasse diagram for this partial ordering, the nodes are tasks; we'll add to each node the information about the time to perform the task. Also, as is traditional, we'll orient the diagram so that if $x < y$, then x is to the left of y rather than below y . Thus the entire diagram runs from left to right rather than from bottom to top. Such a diagram for task scheduling is often called a **PERT(program evaluation and review technique)** chart, first developed for tracking the construction of Navy submarines but useful for managing any complex project with a number of subtasks. The PERT chart for manufacturing rocking chairs is shown in Figure 5.7, with task numbers substituted for task names and arrows pointing to a task from its prerequisite task(s). The numbers in parentheses indicate the time required to perform the task.

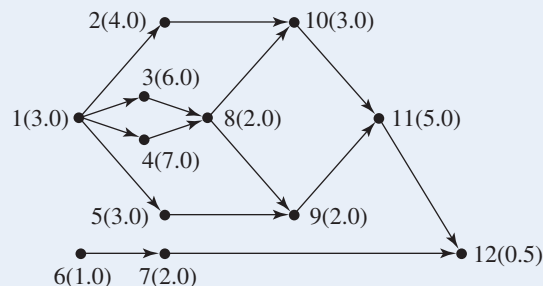


Figure 5.7

PRACTICE 17 Construct the PERT chart for building a house from the following task table.

Task	Prerequisite Tasks	Days to Perform
1. Clearing lot	None	4
2. Pouring pad	1	3
3. Doing framing	2	7
4. Shingling roof	3	6
5. Adding outside siding	3	4
6. Installing plumbing and wiring	4, 5	6
7. Hanging windows and doors	3	5
8. Installing wallboard	6	5
9. Painting interior	7, 8	5

A project represented by a PERT chart must begin with the tasks at the leftmost edge of the PERT chart and end with the tasks at the rightmost edge. An upper limit on the time required to complete the project can be obtained by adding the times for performing each task, but this does not take into account the fact that perhaps some tasks can be performed in parallel, such as tasks 2 through 5 in Example 16. To obtain the minimum time required to complete the project, we can move through the chart from left to right, computing for each node the minimum time to complete the work from the beginning through the work at that node. If a node x has multiple nodes as prerequisites, all the prerequisite tasks must be completed before we can begin work on x ; thus we must add the time for task x to the maximum completion time of the prerequisite nodes.

EXAMPLE 17 Let's compute the time for completing each task in Example 16.

$$\begin{aligned}
 \text{Task 1:} & \quad 3.0 \\
 \text{Task 2:} & \quad 3.0 + 4.0 = 7.0 \\
 \text{Task 3:} & \quad 3.0 + 6.0 = 9.0 \\
 \text{Task 4:} & \quad 3.0 + 7.0 = 10.0 \\
 \text{Task 5:} & \quad 3.0 + 3.0 = 6.0 \\
 \text{Task 6:} & \quad 1.0 \\
 \text{Task 7:} & \quad 1.0 + 2.0 = 3.0 \\
 \text{Task 8:} & \quad \max(\text{time to complete task 3, time to complete task 4}) \\
 & \quad + \text{time to perform task 8} \\
 & \quad = \max(9.0, 10.0) + 2.0 = 10.0 + 2.0 = 12.0
 \end{aligned}$$

$$\begin{aligned}
 \text{Task 9:} & \quad \max(\text{time to complete task 5, time to complete task 8}) \\
 & \quad + \text{time to perform task 9} \\
 & \quad = \max(6.0, 12.0) + 2.0 = 12.0 + 2.0 = 14.0 \\
 \text{Task 10:} & \quad \max(\text{time to complete task 2, time to complete task 8}) \\
 & \quad + \text{time to perform task 10} \\
 & \quad = \max(7.0, 12.0) + 3.0 = 12.0 + 3.0 = 15.0 \\
 \text{Task 11:} & \quad \max(\text{time to complete task 9, time to complete task 10}) \\
 & \quad + \text{time to perform task 11} \\
 & \quad = \max(14.0, 15.0) + 5.0 = 15.0 + 5.0 = 20.0 \\
 \text{Task 12:} & \quad \max(\text{time to complete task 7, time to complete task 11}) \\
 & \quad + \text{time to perform task 12} \\
 & \quad = \max(3.0, 20.0) + 0.5 = 20.0 + 0.5 = 20.5
 \end{aligned}$$

Therefore the minimum number of hours to manufacture a rocking chair is 20.5. From node 12, we can travel back in the chart, selecting at each point of multiple prerequisites the node that contributed the maximum value. This gives the sequence of nodes

$$12, 11, 10, 8, 4, 1$$

or, reversing this sequence,

$$1, 4, 8, 10, 11, 12$$

The sum of the times to perform each task in this sequence is 20.5. If any of these tasks takes longer to perform than its allotted time, the entire project will take longer than 20.5 hours. This sequence of nodes is a **critical path** through the PERT chart—performing these tasks in the allotted time is critical to completing the entire project on time. ●

The critical path in a PERT chart represents the minimum time to completion of the entire project. If a task not on the critical path takes longer than its allotted time to perform, then the critical path may shift to include this node, because it then becomes the bottleneck slowing down completion of the total project. In a complex project, the critical path must continually be recomputed to determine where best to allocate resources to move the project forward.

PRACTICE 18

Compute the minimum time to completion and the nodes on the critical path for the house-building project of Practice 17. ■

Given a partial ordering ρ on a finite set, there is always a total ordering σ that is an extension of ρ , meaning that if $x\rho y$, then $x\sigma y$. The process of **topological sorting** finds such a total ordering from a partial ordering. This is indeed a sorting process in the sense that the objects end up being totally ordered, but since they must be partially ordered to begin with, it is a very specialized sorting process.

Recall that in a finite partially ordered set, an element is minimal if it has no predecessors. In a finite nonempty partially ordered set, at least one minimal

element must exist. To see this, let x belong to the set. If x is not minimal, then there is a y in the set with $y\rho x$, $y \neq x$. If y is not minimal, then there is a z in the set with $z\rho y$, $z \neq y$, and so on. Because the set is finite, this process cannot go on indefinitely, so one such element must be minimal. A minimal element in a Hasse diagram has no elements below it; a minimal element in a PERT chart has no elements to its left.

The accompanying pseudocode algorithm for topological sorting operates on a partially ordered set (S, ρ) . Minimal elements (picked at random if there is a choice of minimal elements at any stage) are repeatedly removed from the ordered set until the set is empty. Each removal of a minimal element leaves a finite partially ordered set, so that another minimal element may be found.

ALGORITHM *TOPOLOGICALSORT*

```

TopSort(finite set  $S$ ; partial ordering  $\rho$  on  $S$ )
//find a total ordering on  $S$  that is an extension of  $\rho$ 
Local variable
integer  $i$  //enumerates tasks in total ordering
 $i = 1$ 
while  $S \neq \emptyset$ 
    pick a minimal element  $x_i$  from  $S$ ;
     $S = S - \{x_i\}$ 
     $i = i + 1$ 
end while
// $x_1 < x_2 < x_3 < \dots < x_n$  is now a total ordering that extends  $\rho$ 
write( $x_1, x_2, x_3, \dots, x_n$ )
end function TopSort

```

The ordering $x_1 < x_2 < x_3 < \dots < x_n$ produced by this algorithm is a total ordering. To see that it is an extension of ρ , suppose that $x_i \rho x_j$. Then x_i precedes x_j and x_i must be chosen as a minimal element and removed from the set before x_j can be chosen as a minimal element. Therefore $i < j$ and $x_i < x_j$.

EXAMPLE 18

One topological sort of the partial ordering of Example 16 is

6, 1, 7, 2, 3, 5, 4, 8, 10, 9, 11, 12

In Figure 5.7, either 6 or 1 is minimal and may be chosen as the first element. If 6 is chosen and removed from the set, then, as shown in Figure 5.8, either 1 or 7 is minimal. If 1 is then chosen and removed from the set (Figure 5.9), then 2, 3, 4, 5, and 7 are all minimal and any one can be chosen next. The process continues until all nodes have been chosen. If Ernie's brothers all move to the city and he is left to build rocking chairs alone, the topological sort gives an order in which he can perform tasks sequentially.

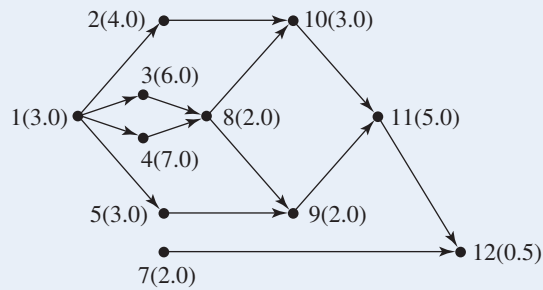


Figure 5.8

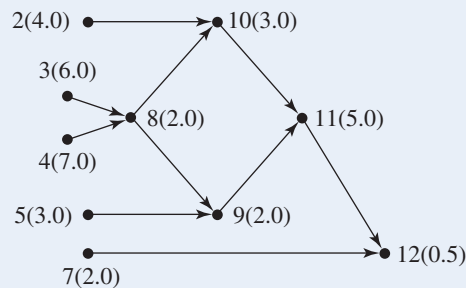


Figure 5.9

PRACTICE 19 Find another topological sort for the partial ordering of Example 16.

PRACTICE 20 Find a topological sort for the partial ordering of Practice 17.

The algorithm given here for topological sorting is still somewhat imprecise, as we have not given a mechanical method for finding a minimal element. Another algorithm will be described in Section 7.4.

SECTION 5.2 REVIEW

TECHNIQUES

- W Construct a PERT chart from a task table.
 - Find the critical path in a PERT chart.
- W Do a topological sort on a partially ordered set.

MAIN IDEAS

- PERT charts are diagrams of partially ordered sets representing tasks and prerequisites among tasks.
- A topological sort extends a partial ordering on a finite set to a total ordering.

EXERCISES 5.2

1. The following tasks are required in order to assemble a bicycle. As the manufacturer, you must write a list of sequential instructions for the buyer to follow. Will the sequential order given below work? Give another sequence that could be used.

Task	Prerequisite Tasks
1. Tightening frame fittings	None
2. Attaching handle bars to frame	1
3. Attaching gear mechanism	1
4. Mounting tire on wheel assembly	None
5. Attaching wheel assembly to frame	1, 4
6. Installing brake mechanism	2, 3, 5
7. Adding pedals	6
8. Attaching seat	1
9. Adjusting seat height	7, 8

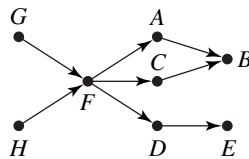
2. Give a list of tasks and prerequisite tasks for cooking and serving a hamburger.
 3. Construct a PERT chart from the following task table.

Task	Prerequisite Tasks	Time to Perform
<i>A</i>	<i>E</i>	3
<i>B</i>	<i>C, D</i>	5
<i>C</i>	<i>A</i>	2
<i>D</i>	<i>A</i>	6
<i>E</i>	None	2
<i>F</i>	<i>A, G</i>	4
<i>G</i>	<i>E</i>	4
<i>H</i>	<i>B, F</i>	1

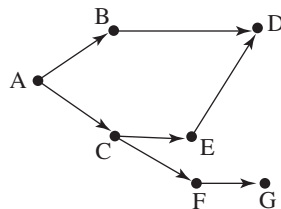
4. Construct a PERT chart from the following task table.

Task	Prerequisite Tasks	Time to Perform
1	2	4
2	3	2
3	8	5
4	3	2
5	4, 7	2
6	5	1
7	3	3
8	None	5

5. Compute the minimum time to completion and the nodes on the critical path for the problem in Exercise 3.
6. Compute the minimum time to completion and the nodes on the critical path for the problem in Exercise 4.
7. For the problem in Exercise 3, great improvements in productivity have knocked down the time to perform task *D* from 6 units to 1 unit. Recompute the minimum time to completion and the nodes on the critical path.
8. For the problem in Exercise 4, an extra quality-control step has been added to task 4, which now requires 4 units of time to perform. Recompute the minimum time to completion and the nodes on the critical path.
9. Do a topological sort on the partially ordered set shown.



10. Do a topological sort on the partially ordered set shown.



11. Find a topological sort for the problem in Exercise 3.
12. Find a topological sort for the problem in Exercise 4.
13. Given the following task chart for fixing an Asian dinner, find a total ordering in which the tasks can be performed sequentially.

Task	Prerequisite Tasks
1. Chop onions	9
2. Wash lettuce	11
3. Make dressing	11
4. Do stir fry	10
5. Toss salad	2, 3
6. Cut up chicken	None
7. Grate ginger	9
8. Chop bok choy	9
9. Marinate chicken	6
10. Heat wok	1, 7, 8, 11
11. Prepare rice	None

14. A U.S. journalist, on being posted to a bureaucratic foreign country, was faced with the following tasks before she could begin work.

Task	Prerequisite Tasks
1. Obtain a residence permit from the Public Security Bureau	2, 3, 7
2. Obtain a health certificate from the local hospital	None
3. Obtain a journalist work card from the Foreign Ministry	None
4. Obtain a customs certificate from the Customs Office	1, 3, 9
5. Post an announcement in the local newspaper about the presence of her news company in the country	None
6. Obtain a journalist visa from the Public Security Bureau	2, 3, 7
7. Obtain a foreign journalist housing contract from the local housing authority	None
8. Pick up her shipment of belongings from the United States	1, 4, 6
9. Obtain a news organization permit from the Foreign Ministry.	5

Find a total ordering in which the tasks can be performed sequentially.

15. Recall the problem posed at the beginning of this chapter:

Your company has developed a program for use on a small parallel processing machine. According to the technical documentation, the program executes processes P1, P2, and P3 in parallel; these processes all need results from process P4, so they must wait for Process P4 to complete execution before they begin. Processes P7 and P10 execute in parallel but must wait until Processes P1, P2, and P3 have finished. Process P4 requires results from P5 and P6 before it can begin execution. P5 and P6 execute in parallel. Processes P8 and P11 execute in parallel but P8 must wait for Process P7 to complete, and Process P11 must wait for P10 to complete. Process P9 must wait for results from P8 and P11. You have been assigned to convert the software for use on a single processor machine.

Use a topological sort to determine the order in which the processes should be executed sequentially.

16. Given the following task chart for conducting a water quality study, find a total ordering in which the tasks can be performed sequentially.

Task	Prerequisite Tasks
1. Plan schedule	7
2. Collect new data	1, 10
3. Assemble team	7
4. Collect prior water quality data	6
5. Obtain equipment	7
6. Identify streams	None
7. Decide on readings needed	4, 8
8. Review federal and state guidelines	None
9. Write report	11
10. Distribute equipment	3, 5
11. Analyze new data	2

SECTION 5.3 RELATIONS AND DATABASES

A **database** is a storehouse of associated information about some enterprise. The user of a database can certainly retrieve some specific fact stored in the database. But a well-designed database is more than simply a list of facts. The user can perform queries on the database to retrieve information not contained in any single fact. The whole becomes more than the sum of its parts.

To design a useful and efficient computerized database, it is necessary to model or represent the enterprise with which the database is concerned. A **conceptual model** attempts to capture the important features and workings of the enterprise. Considerable interaction with those who are familiar with the enterprise may be required to obtain all the information necessary to formulate the model.

Entity-Relationship Model

One high-level representation of an enterprise is the **entity-relationship model**. In this model, important objects, or **entities**, in the enterprise are identified, together with their relevant attributes or properties. Then the relationships between these various entities are noted. This information is represented graphically by an **entity-relationship diagram**, or **E-R diagram**. In an E-R diagram, rectangles denote entity sets, ellipses denote attributes, and diamonds denote relationships.

EXAMPLE 19

The Pet Lovers of America Club (PLAC) wants to set up a database. PLAC has bought mailing lists from commercial sources, and it is interested in people who own pets and in some basic information about those pets, such as the name, type of pet (dog, cat, and so on), and the breed.

Figure 5.10 shows an E-R diagram for the PLAC enterprise. This diagram says that persons and pets are the entities. Persons have the attributes of *Name*, *Address*, *City*, and *State*. Pets have the attributes of *PetName*, *PetType*, and *Breed*. The diagram also shows that persons own pets. Thinking of the entities as sets, the Person set and the Pet set, the relationship “owns” is a binary relation from Person to Pet—the ownership relation is captured by (person, pet) ordered pairs. The “1” and “N” on the connecting lines indicate that this binary relation is one-to-many; that is, in this particular enterprise, one person can own many pets, but no pet has multiple owners. (Pets with multiple owners would result in a many-to-many relation.) Also, in this example, some persons may own no pets, and some pets may have no owners.

The fact that no pet has multiple owners is one of the “business rules” of the enterprise. Such business rules are important to identify when designing a database, because they can determine various features of the database, as we will see.

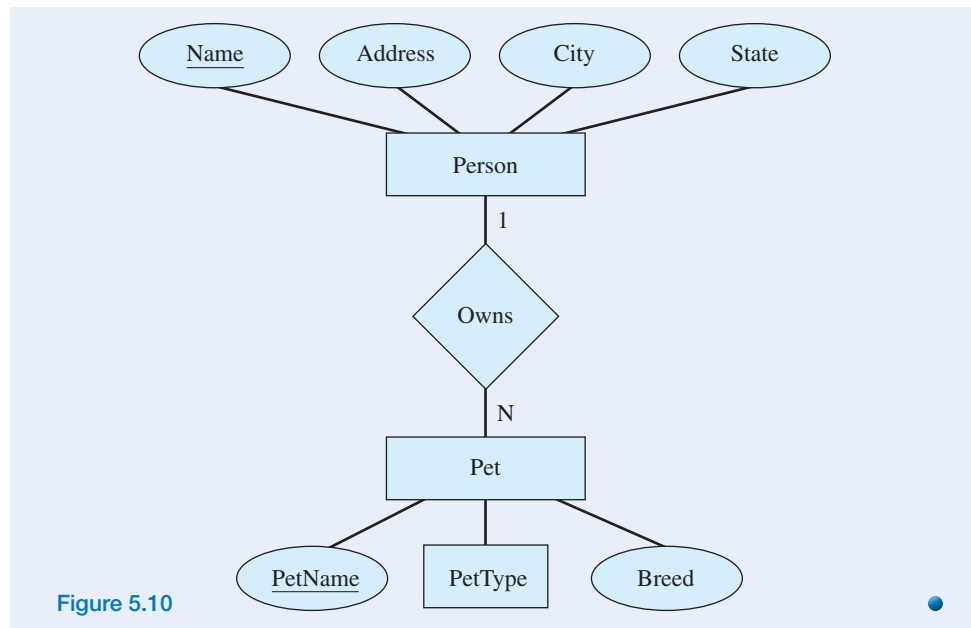


Figure 5.10

Relational Model

Another representation of an enterprise, called a **relational model**, can be developed from the E-R model. Both the entity sets and the relationships of the E-R model become relations (in the mathematical sense) in the relational model. The relations are described by tables. A **relational database** consists of collections of such tables.

An entity set table is named for the entity set. Each row in the table contains the values of the n attributes for a specific instance of that entity set. Thus the relational table may be thought of as a set of n -tuples (rows), and an individual row is called a **tuple**. True to the idea of a set, no duplicate tuples exist, and no ordering of the tuples is assumed. The ordering of the attributes is unimportant, except that consistency must be maintained; that is, each column in the table contains values for a specific attribute in all of the tuples. The number of attributes (columns) is called the **degree of the relation**. The number of n -tuples (rows) is called the **cardinality of the relation**; it is the cardinality (in the set-theoretic sense) of the set of rows.

More formally, a database relation is a subset of $D_1 \times D_2 \times \cdots \times D_n$, where D_i is the domain from which attribute A_i takes its values. This means that the database use of the word *relation* is consistent with our definition of an n -ary relation on multiple sets (page 330). Beyond the data in the table itself, additional information, sometimes called **metadata**—data about data—is needed to specify the domain for each attribute. Is the domain the set of all possible strings, or do these strings have to follow a specific format? Is the domain the set of integers or does it have to be the set of integers within some specific range? Does the attribute represent a date? If so, the domain has to specify the specific date format to be used, for example, March 13, 2014 or 3/13/14 or 3/13/2014, and so forth. In addition, each attribute domain D_i is assumed to contain a special NULL value (empty

value), so a given tuple could have a NULL value for one or more of its attributes. Any tuple (x_1, x_2, \dots, x_n) in a relation table must satisfy the n -ary predicate of the form $(\forall x_i)(x_i \in D_i)$.

EXAMPLE 20

The Person relation in the PLAC database might contain the following data.

Person			
<u>Name</u>	<u>Address</u>	<u>City</u>	<u>State</u>
Patrick, Tom	2425 Samset	Sarasota	FL
Smith, Mary	1121 Ridge Rd.	Rockville	IL
Collier, Jon	429 Via Rivio	Venice	IL
Jones, Kate	345 Forest St.	Cleveland	OH
Smith, Bob	1201 45th St.	Falls City	MA
White, Janet	110 Toledo Rd.	Brookville	GA
Garcia, Maria	24 E. 56th St.	New York City	NY

The four attributes for each tuple are *Name*, *Address*, *City*, and *State*. The meta-data specify that the domain for the Name attribute is the set of strings of the form LastName, FirstName; the domain for the State attribute is the set of legitimate two-character state abbreviations. The Pet relation could be

Pet		
<u>PetName</u>	<u>PetType</u>	<u>Breed</u>
Spot	Dog	Hound
Twinkles	Cat	Siamese
Lad	Dog	Collie
Lassie	Dog	Collie
Mohawk	Fish	Moorish idol
Tweetie	Bird	Canary
Tiger	Cat	Shorthair

Because there are no duplicate tuples in a relation, giving the value of all n attributes of a tuple clearly distinguishes that tuple from all others. However, there may be a minimal subset of the attributes that can be used to uniquely identify each tuple. This subset is called the **primary key** of the relation; if the subset consists of more than one attribute, then it is a **composite primary key**. In the table describing the relation (and in the E-R diagram), the primary key is underlined in the row of attribute names. No component of the primary key should ever have a NULL (empty) value. This **entity integrity** constraint merely confirms that each

tuple must have a primary key value in order to distinguish that tuple and that all attribute values of the primary key are needed in order to identify a tuple uniquely.

Another business rule of the PLAC enterprise is that all people have unique names; therefore *Name* is sufficient to identify each tuple and was chosen as the primary key in the Person relation. Note that for the Person relation as shown in this example, *State* could not serve as a primary key because there are two tuples with *State* value “IL.” However, just because *Name* has unique values in this instance does not preclude the possibility of duplicate names. It is the business rule that determines that names will be unique. (There is no business rule that says that addresses or cities are unique, so neither of these attributes can serve as the primary key, even though there happen to be no duplicates in the Person relation shown.)

The assumption of unique names is a somewhat simplistic business rule. The primary key in a relation involving people is often an identifying number that is a unique attribute. This used to be a Social Security number, but due to privacy concerns, institutions now often generate a local unique identifier such as a student ID number or employee ID number, or they use a driver’s license number. Because *PetName* is the primary key in the Pet relation of Example 20, we can surmise the even more surprising business rule that in the PLAC enterprise, all pets have unique names. A more realistic scenario would call for creating a unique attribute for each pet, sort of a pet Social Security number, to be used as the primary key. This key would have no counterpart in the real enterprise, so the database user would never need to see it; such a key is called a **blind key** or **surrogate key**. Blind keys are often generated automatically by the database system using a simple sequential numbering scheme.

An attribute in one relation (called the “child” relation) may have the same domain as the primary key attribute in another relation (called the “parent” relation). Such an attribute is called a **foreign key** (of the child relation) into the parent relation. A relation for a relationship (that is, for a diamond in the E-R diagram) between entities uses foreign keys to establish connections between those entities. There will be one foreign key in the relationship relation for each entity participating in the relationship.

EXAMPLE 21

The PLAC enterprise has identified the following instance of the Owns relationship. The *Name* attribute of Owns is a foreign key into the Person relation where *Name* is a primary key; *PetName* of Owns is a foreign key into the Pet relation, where *PetName* is a primary key. The first tuple establishes the Owns relationship between Bob Smith and Spot; that is, it indicates that Bob Smith owns Spot.

Owns	
<i>Name</i>	<i>PetName</i>
Smith, Bob	Spot
Smith, Mary	Twinkles
Jones, Kate	Lad
Jones, Kate	Lassie
Collier, Jon	Tweetie
White, Janet	Tiger

Persons who do not own pets are not represented in Owns, nor are pets with no owners. The primary key of Owns is *PetName*. Recall the business rule that no pet has multiple owners. If any pet could have multiple owners, the composite primary key *Name/PetName* would have to be used. *Name* alone cannot serve as the primary key because people can have more than one pet (for example, Jones, Kate, does not identify a unique tuple.) ●

In a one-to-one or one-to-many relationship such as our example, a separate relationship table (like Owns), while not incorrect, is also not necessary.

EXAMPLE 22

Because *PetName* in the Owns relation is a foreign key into the Pet relation, the two relations can be combined (using an operation called *outer join over PetName*) to form the PetOwner relation.

PetOwner			
<i>Name</i>	<u><i>PetName</i></u>	<i>PetType</i>	<i>Breed</i>
Smith, Bob	Spot	Dog	Hound
Smith, Mary	Twinkles	Cat	Siamese
Jones, Kate	Lad	Dog	Collie
Jones, Kate	Lassie	Dog	Collie
NULL	Mohawk	Fish	Moorish idol
Collier, Jon	Tweetie	Bird	Canary
White, Janet	Tiger	Cat	Shorthair

This PetOwner relation can replace both the Owns relation and the Pet relation with no loss of information. PetOwner contains a tuple with a NULL value for *Name*. This tuple does not violate entity integrity because *Name* is not a component of the primary key but instead is still a foreign key into Person. ●

Operations on Relations

Two unary operations that can be performed on relations are *restrict* and *project*. The **restrict** operation creates a new relation made up of those tuples of the original relation that satisfy certain conditions. The **project** operation creates a new relation made up of certain attributes from the original relation, eliminating any duplicate tuples. The restrict and project operations can be thought of in terms of subsets. The restrict operation creates a subset of the rows that satisfy certain conditions; the project operation creates a subset of the columns that represent certain attributes.

EXAMPLE 23

The operation

Restrict PetOwner **where** $PetType = \text{“Dog”}$ **giving** DogOwner

results in the relation DogOwner.

DogOwner			
Name	PetName	PetType	Breed
Smith, Bob	Spot	Dog	Hound
Jones, Kate	Lad	Dog	Collie
Jones, Kate	Lassie	Dog	Collie

The operation

Project PetOwner **over** $(Name, PetType)$ **giving** Preference

results in the relation Preference.

Preference	
Name	PetType
Smith, Bob	Dog
Smith, Mary	Cat
Jones, Kate	Dog
NULL	Fish
Collier, Jon	Bird
White, Janet	Cat

PRACTICE 21 Write the relation that results from the operation

Project Person **over** $(Name, State)$ **giving** Locale

Because relations are sets of n -tuples, the binary operations of union, intersection, and set difference can be applied to two relations with the same basic structure. Thus in our example, two different tables containing information about pet owners, both laid out with the same structure, could be intersected to produce a relation containing all the common 4-tuples.

Another binary operation, **join**, can be performed on two relations with a common attribute (column). Theoretically, this operation initially forms the Cartesian product of all n -tuples (rows) in the first relation with all k -tuples (rows) in the second relation. It views the result as a set of $(n + k)$ -tuples and then restricts to the subset of those where the common attribute has the same value, writing the

result as a set of $(n + k - 1)$ -tuples (the common attribute is written only once). Join is therefore not really a separate operation but is defined as the result of doing a Cartesian product followed by a restrict.

EXAMPLE 24

The operation

Join Person and PetOwner over Name giving Listing

results in the Listing relation.

Listing						
Name	Address	City	State	PetName	PetType	Breed
Smith, Mary	1121 Ridge Rd.	Rockville	IL	Twinkles	Cat	Siamese
Collier, Jon	429 Via Rivio	Venice	IL	Tweetie	Bird	Canary
Jones, Kate	345 Forest St.	Cleveland	OH	Lad	Dog	Collie
Jones, Kate	345 Forest St.	Cleveland	OH	Lassie	Dog	Collie
Smith, Bob	1201 45th St.	Falls City	MA	Spot	Dog	Hound
White, Janet	110 Toledo Rd.	Brookville	GA	Tiger	Cat	Shorthair

The restrict, project, and join operations can be applied in various combinations to formulate queries that the user wishes to perform on the database. For example, suppose the query is

Give the names of all cats whose owners live in Illinois. (1)

If the only existing relations are Person and PetOwner, the following sequence of operations will produce a relation that answers this query:

Restrict PetOwner where PetType = “Cat” giving Results1

Results1			
Name	PetName	PetType	Breed
Smith, Mary	Twinkles	Cat	Siamese
White, Janet	Tiger	Cat	Shorthair

Restrict Person where State = “IL” giving Results2

Results2			
Name	Address	City	State
Smith, Mary	1121 Ridge Rd.	Rockville	IL
Collier, Jon	429 Via Rivio	Venice	IL

Join Results2 and Results1 over *Name* giving Results3

Results3						
<i>Name</i>	<i>Address</i>	<i>City</i>	<i>State</i>	<i>PetName</i>	<i>PetType</i>	<i>Breed</i>
Smith, Mary	1121 Ridge Rd.	Rockville	IL	Twinkles	Cat	Siamese

Project Results3 over *PetName* giving FinalResults

FinalResults
<i>PetName</i>
Twinkles

This query could also be performed by first doing the join operation of Example 24 followed by the restrict and project operations, but the join table would be much larger.

EXAMPLE 25

Relational algebra is a theoretical relational database language in which the restrict, project, and join operations can be combined. The relational algebra equivalent of the sequence of operations we did to find the names of cats whose owners live in Illinois would be the statement

project (join(restrict PetOwner where *PetType* = “Cat”) and (restrict Person where *State* = “IL”) over *Name*) over *PetName* giving Final_Results. (2)

SQL is an international standard relational database language; the preceding query would appear as the following SQL statement, where the lines are numbered only for discussion purposes:

1. **SELECT** *PetName*
2. **FROM** PetOwner, Person
3. **WHERE** *PetOwner. Name* = *Person. Name*
4. **AND** *PetType* = “Cat”
5. **AND** *State* = “IL”;

(3)

SQL’s **SELECT** statement can actually perform relational algebra restricts, projects, and joins, as shown here. Lines 4 and 5 represent the two restrict operations. Line 2 represents the Cartesian product between the two relations and line 3 identifies the common attribute. Therefore lines 2 and 3 together represent the join. Line 1 represents the project operation. AND, OR, and NOT connectives are also available. ●

Instead of using the relational algebra approach, in which the restrict, project, and join operations are used to process a query, we can use the relational calculus approach. In **relational calculus**, instead of specifying the operations to be done

in order to process a query, we give a set-theoretic description of the desired result of the query. We specify what we want, not how to get it. Sounds like Prolog (see Section 1.5). In fact, the description of the set may involve notation from predicate logic; remember that predicate logic is also called predicate calculus, hence the name relational calculus. Relational algebra and relational calculus are equivalent in their expressive power; that is, any query that can be formulated in one language can be formulated in the other.

EXAMPLE 26

The relational calculus expression for the query asking for the names of all cats whose owners live in Illinois is

Range of x is PetOwner
 Range of y is Person
 $\{x.PetName \mid x.PetType = \text{“Cat” and}$
 $\text{exists } y(y.Name = x.Name \text{ and } y.State = \text{“IL”})\}$ (4)

Here “Range of x is PetOwner” specifies the relation from which the tuple x may be chosen, and “Range of y is Person” specifies the relation from which the tuple y may be chosen. (The use of the term *range* is unfortunate. We are really talking about domain in the same sense we talked about the domain of an interpretation in predicate logic—the pool of potential values.) The notation “exists y ” stands for the existential quantifier ($\exists y$).

Expressions (1) through (4) all represent the same query expressed in English language, relational algebra, SQL, and relational calculus, respectively.

PRACTICE 22

Using the relations Person and PetOwner, express the following query in relational algebra, SQL, and relational calculus form:

Give the names of all cities where dog owners live.

NULL Values and Three-valued Logic

The value of an attribute in a particular tuple may be unknown, in which case the attribute is assigned a NULL value. For example, we might have the tuple

Bruno, Dog, NULL

in the Pet table if Bruno is a dog of unknown breed. (Note that Bruno’s breed might be unknown in some absolute sense, or it might simply be unknown to the person entering the data.)

Because NULL means “unknown value,” any comparisons between a NULL value and any other value must result in NULL. Thus

“Poodle” = NULL

results in NULL; since the NULL value is unknown, it is also unknown whether it has the value “Poodle.”

REMINDER

Any comparison involving a NULL value results in NULL.

Ordinary comparisons (does $2 = 2$? does $2 = 5$?) result in True or False values, but when a NULL value is involved, the result, as we have seen, will be NULL. This introduces a *three-valued logic* where expressions can have values of True, False, or NULL. Truth tables can be written for three-valued logic (see Exercise 53 of Section 1.1).

A	B	$A \wedge B$
T	T	T
T	F	F
T	N	N
F	T	F
F	F	F
F	N	F
N	T	N
N	F	F
N	N	N

A	B	$A \vee B$
T	T	T
T	F	T
T	N	T
F	T	T
F	F	F
F	N	N
N	T	T
N	F	N
N	N	N

A	A'
T	F
F	T
N	N

Most database management systems follow these rules of three-valued logic until a final truth value decision must be made, and then a NULL value gets set to False. But this can have unexpected consequences. For example, if Bruno is added to the Pet table and the following SQL query is executed

```
SELECT PetName
FROM Pet
WHERE PetType = "Dog"
AND NOT (Breed = "Collie");
```

we may expect to see Bruno's name in the resulting relation, since Bruno's breed is not "Collie." But as Bruno's attribute values are compared with the criteria specified in the SQL statement, we get

```
PetType = "Dog" AND NOT (Breed = "Collie")
"Dog" = "Dog" AND NOT (NULL = "Collie")
True AND NOT NULL
True AND NULL
NULL
```

which then is set to False, so Bruno does not satisfy this query. On second thought, because Bruno's breed is NULL, he might actually be a collie; this query result reflects the fact that it cannot be said with certainty that Bruno is a noncollie dog.

But consider the SQL query

```
SELECT PetName
FROM Pet
WHERE PetType = "Dog"
AND Breed = NULL;
```

Surely this describes Bruno. However, remember that the result of any comparison involving NULL is NULL, so

```
PetType = "Dog" AND Breed = NULL
"Dog" = "Dog" AND NULL = NULL
True AND NULL
NULL
```

which then is set to False. Contrary to intuition, Bruno does not satisfy this query either. The only true fact about Bruno is that he is a dog.

The SQL query

```
SELECT PetName
FROM Pet
WHERE PetType = "Dog"
AND Breed IS NULL;
```

is a completely different query from the preceding one. The WHERE clause is asking whether the Breed attribute for any tuple has the value NULL. This query would produce the following result because Bruno is the only tuple in the Pet table with a NULL value for Breed.

IsNull
PetName
Bruno

Database Integrity

New information must be added to a database from time to time, obsolete information deleted, and changes or updates made to existing information. In other words, the database will be subjected to **add**, **delete**, and **modify** operations. An add operation can be carried out by creating a second relation table with the new information and performing a set union of the existing table and the new table. Delete can be accomplished by creating a second relation table with the tuples to be deleted and performing a set difference that subtracts the new table from the existing table. Modify can be achieved by a delete (of the old tuple) followed by an add (of the modified tuple).

These operations must be carried out so that the information in the database remains in a correct and consistent state that agrees with the business rules. Enforcing three “integrity rules” will help. **Data integrity** requires that the values for an attribute do indeed come from that attribute’s domain. In our example, for instance, values for the State attribute of Person must be legitimate two-letter state abbreviations (or the NULL value). **Entity integrity**, as we discussed earlier, requires that no component of a primary key value be NULL. These integrity constraints clearly affect the tuples that can be added to a relation.

Referential integrity requires that any values for foreign keys of child relations into parent relations either be NULL or have values that match values in the corresponding primary keys of the parent relations. The referential integrity constraint affects both add and delete operations (and therefore modify operations).

For instance, we could not add a tuple to *PetOwner* with a non-NULL *Name* value that does not exist in the *Person* relation, because this would violate the *Owns* relation as a binary relation on $\text{Person} \times \text{Pet}$. Also, if the Bob Smith tuple is deleted from the *Person* relation, then the Bob Smith tuple must be deleted from the *PetOwner* relation or the *Name* value “Bob Smith” changed to NULL (a business rule must specify which is to occur) so that *PetOwner*’s foreign key *Name* does not violate referential integrity. This prevents the inconsistent state of a reference to Bob Smith in *PetOwner* when Bob Smith no longer exists as a “Person.”

SECTION 5.3 REVIEW

TECHNIQUES

- W Carry out restrict, project, and join operations in a relational database.
- W Formulate relational database queries using relational algebra, SQL, and relational calculus.

MAIN IDEAS

- A relational database uses mathematical relations, described by tables, to model objects and relationships in an enterprise.
- The database operations of restrict, project, and join are operations on relations (sets of tuples).
- Queries on relational databases can be formulated using the restrict, project, and join operations, SQL statements, or notations borrowed from set theory and predicate logic.

EXERCISES 5.3

Exercises 1–4 refer to the *Person*, *Pet*, and *PetOwner* relations of Examples 20 and 22.

1. Consider the following operation:

Restrict *Pet* **where** $\text{PetType} = \text{“Cat”}$ **giving** *Kitties*

- a. Write a query in English that would result in the information contained in *Kitties*.
 - b. What is the cardinality of the relation obtained by performing this operation?
 - c. Write an SQL query to obtain this information.
2. Consider the following operation:

Project *Person* **over** $(\text{Name}, \text{City}, \text{State})$ **giving** *Census*

- a. Write a query in English that would result in the information contained in *Census*.
 - b. What is the degree of the relation obtained by performing this operation?
 - c. Write an SQL query to obtain this information.
3. Write the results of the following operation:

Project *Pet* **over** $(\text{PetName}, \text{Breed})$ **giving** *What Am I*

4. Write the results of the following operation:

Restrict *PetOwner* **where** $\text{PetType} = \text{“Bird”}$ **OR** $\text{PetType} = \text{“Cat”}$ **giving** *SomeOwners*

Exercises 5–28 are all related to the same enterprise.

5. A library maintains a database about its books. Information kept on each author includes the author's name and country of origin. Information kept on each book includes the ISBN, title, publisher, and subject. Authors and books are the entities in this enterprise, and "writes" is a relationship between these entities. Sketch an E-R diagram for the enterprise. In the absence of any business rules, what must be assumed about the binary relation "writes" regarding whether it is one-to-one, one-to-many, and so on?
6. In a relational model of the library database, there is an author relation, a book relation, and a writes relation. Give the table heading for each of the relation tables, underlining the primary key. Business rules state that authors are uniquely identified by name and books are uniquely identified by ISBN. Explain your choice of primary key for the Writes relation table.

For Exercises 7–16, use the following relation tables and write the results of the operations. These tables are sorted by primary key (or in the case of the Writes table, by the first component of the primary key), which is not required but is useful for Exercise 25. Many database systems maintain data in sorted order by primary key using a tree structure (see Section 5.1, Exercise 43).

Author	
<u>Name</u>	Country
Chan, Jimmy	China
East, Jane	U. S.
King, Dorothy	England
Kovalsco, Bert	U.S.
Lau, Won	China
Nkoma, Jon	Kenya
Quercos, Tom	Mexico

Book			
<u>ISBN</u>	Title	Publisher	Subject
0-115-01214-1	Birds of Africa	Lorraine	Nature
0-364-87547-X	Early Tang Paintings	Bellman	Art
0-56-000142-8	Springtime Gardening	Swift-Key	Nature
0-816-35421-9	Springtime Gardening	Harding	Nature
0-816-53705-4	Baskets for Today	Harding	Art
0-816-88506-0	Autumn Annuals	Harding	Nature

Writes	
<i>Name</i>	<i>ISBN</i>
Chan, Jimmy	0-364-87547-X
East, Jane	0-56-000142-8
King, Dorothy	0-816-35421-9
King, Dorothy	0-816-88506-0
Kovalsco, Bert	0-816-53705-4
Lau, Won	0-364-87547-X
Nkoma, Jon	0-115-01214-1

7. **Restrict** Author **where** *Country* = “U. S.” **giving** Results7
8. **Restrict** Writes **where** *Name* = “Dorothy King” **giving** Results8
9. **Restrict** Book **where** *Publisher* = “Bellman” **or** *Publisher* = “Swift Key” **giving** Results9
10. **Restrict** Book **where** *Publisher* = “Harding” and *Subject* = “Art” **giving** Results10
11. **Project** Author **over** *Name* **giving** Results11
12. **Project** Author **over** (*Name*, *Country*) **giving** Results12
13. **Project** Book **over** (*Publisher*, *Subject*) **giving** Results13
14. **Project** Book **over** (*ISBN*, *Title*, *Subject*) **giving** Results14
15. **Join** Book and Writes over *ISBN* **giving** Results15
16. **Join** Author and Writes over *Name* **giving** Results16

For Exercises 17–23, using the relation tables given before Exercise 7, express each query in relational algebra, SQL, and relational calculus forms. Also give the result of each query.

17. Give the titles of all books about art.
18. Give the titles of all books published by Harding.
19. Give the names of all authors who publish with Harding.
20. Give the names of all authors who have written nature books.
21. Give the titles of all books written by U.S. authors.
22. Give the titles, ISBNs, and publishers of all art books whose authors live in the United States.
23. Give the authors’ names and book titles for all art books written by English authors.
24. If the tuple

Fleur, Suzanne NULL

gets added to the Author table, write the results of the SQL query

```
SELECT Name
FROM Author
WHERE Country = “U. S.”
OR Country = NULL;
```

25. Suppose a join operation over some attribute is to be done on two tables of cardinality p and q , respectively.
- The first step is usually to form the Cartesian product of the two relations and then examine the resulting tuples to find those with a common attribute value. How many tuples result from the Cartesian product that then have to be examined to complete the join operation?
 - Now suppose that the two tables have each been sorted on the common attribute. Explain how the join operation can be done more cleverly, avoid the Cartesian product, and examine (read) at most only $(p + q)$ rows.
 - To accomplish a join operation of Author and Writes over *Name*, how many rows must be examined?
 - To accomplish a join operation of Book and Writes over *ISBN*, how many rows must be examined? (See Exercise 26 for why this operation would not be a good idea anyway.)
26. One rule of thumb about good database design is “one fact, one place.” Suppose you try to combine the Book and Writes tables over *ISBN* into a single relation as was done with the PetName and Owns relation. This table would have a heading of the form

<u>ISBN</u>	<u>Title</u>	<u>Publisher</u>	<u>Subject</u>	<u>Name</u>
-------------	--------------	------------------	----------------	-------------

How would the resulting table violate the “one fact, one place” rule? How many tuples have to be updated if the publisher “Bellman” changes its name to “Bellman-Boyd”?

For Exercises 27 and 28, suppose that an additional attribute called *RoyaltyPercent* with a domain of integers between 0 and 100 is added to the Writes relation. The new Writes table appears here. Because the domain of *RoyaltyPercent* is numerical, arithmetic comparisons can be done on a given *RoyaltyPercent* value.

Writes		
<u>Name</u>	<u>ISBN</u>	<u>RoyaltyPercent</u>
Chan, Jimmy	0-364-87547-X	20
East, Jane	0-56-000142-8	100
King, Dorothy	0-816-35421-9	100
King, Dorothy	0-816-88506-0	100
Kovalsco, Bert	0-816-53705-4	100
Lau, Won	0-364-87547-X	80
Nkoma, Jon	0-115-01214-1	100

27. a. Write an SQL query to give the author’s name, the title and ISBN of the book, and the royalty percent for all authors with a royalty percent of less than 100.
 b. Write the results of the query.
28. What database integrity errors would be caused by attempting each of the following actions?
- Adding a tuple in the Writes table: Wilson, Jermain 0-115-01214-1 40
 - Modifying a tuple in the Writes table: Chan, Jimmy 0-364-87547-X Sixty

Exercises 29–36 are all related to the same enterprise.

29. A corporation sponsors a yearly campaign to solicit monetary contributions from its employees for a local charity, and the company decides to use a database to keep track of the data. Employee data already include employee ID, first name, last name, and department. Employees sign a contribution pledge on a particular date, that specifies the total amount they wish to donate and the number of equal biweekly payroll deductions (starting with the next pay period) they want to use to pay off the total. The payroll department needs to know details about each payment, including the contribution pledge the payment is for, the payment date and the amount deducted. An employee can make multiple pledges.

- a. Do you agree that the following “data decomposition” is consistent with the enterprise description? If not, what should be added or what should be removed?

Entity	Attributes				
Employee	<i>EmployeeID</i>	<i>FirstName</i>	<i>LastName</i>	<i>Department</i>	
Contribution	<i>ContributionID</i>	<i>EmployeeID</i>	<i>ContributionDate</i>	<i>TotalAmount</i>	<i>NumberofPayments</i>
Payment	<i>ContributionID</i>	<i>PaymentDate</i>	<i>PaymentAmount</i>		

- b. Identify a primary key for each of the Employee, Contribution, and Payment entities and explain your choice.

30. Draw an E-R diagram based on Exercise 29.

31. A “universal relation” contains all the data values in one relation. The table represents a report that might be distributed to the campaign manager. The universal relation as of 1/16/2014 is shown here.

<i>Employee ID</i>	<i>First Name</i>	<i>Last Name</i>	<i>Department</i>	<i>Contribution ID</i>	<i>Contribution Date</i>	<i>Total Amount</i>	<i>Number of Payments</i>	<i>Payment Date</i>	<i>Payment Amount</i>
1	Mary	Black	Accounting	101	1/1/2013	\$300.00	3	1/15/2013	\$100.00
1	Mary	Black	Accounting	101	1/1/2013	\$300.00	3	1/31/2013	\$100.00
1	Mary	Black	Accounting	101	1/1/2013	\$300.00	3	2/15/2013	\$100.00
1	Mary	Black	Accounting	105	6/1/2013	\$210.00	3	6/15/2013	\$70.00
1	Mary	Black	Accounting	105	6/1/2013	\$210.00	3	6/30/2013	\$70.00
1	Mary	Black	Accounting	105	6/1/2013	\$210.00	3	7/15/2013	\$70.00
2	June	Brown	Payroll	107	6/1/2013	\$300.00	2	6/15/2013	\$150.00
2	June	Brown	Payroll	107	6/1/2013	\$300.00	2	6/30/2013	\$150.00
2	June	Brown	Payroll	108	1/1/2014	\$600.00	12	1/15/2014	\$50.00
3	Kevin	White	Accounting	102	1/1/2013	\$500.00	2	1/15/2013	\$250.00
3	Kevin	White	Accounting	102	1/1/2013	\$500.00	2	1/31/2013	\$250.00
3	Kevin	White	Accounting	109	1/1/2014	\$500.00	2	1/15/2014	\$250.00
4	Kelly	Chen	Payroll	104	4/15/2013	\$100.00	1	4/30/2013	\$100.00
6	Conner	Smith	Sales	103	1/1/2013	\$150.00	2	1/15/2013	\$75.00
6	Conner	Smith	Sales	103	1/1/2013	\$150.00	2	1/31/2013	\$75.00

- a. Given this universal relation, create and populate with data the three relation tables for the three entities described in Exercise 29. Underline the primary key in each table.
 - b. Describe any foreign keys in the relation tables.
 - c. Consider the form of the Employee IDs. This is probably what kind of key?
32. a. If Mary Black moves from the Accounting Department to the Sales Department, how many tuples must be updated in the universal relation?
- b. The three relation tables from Exercise 31 should follow the “one fact, one place” rule (see Exercise 26). With the same change to Mary Black’s department, how many tuples must be updated in the database using the three relation tables of Exercise 31?

Exercises 33–36 make use of the three relation tables from Exercise 31.

33. Write an SQL query to give the employee ID, pay dates, and payment amounts for all pay dates with amounts $>$ \$100. Give the result of the query.
34. Write an SQL query to give the contribution ID, pay date, and payment amount for all payments by Mary Black. Give the result of the query.
35. Write an SQL query to give the first and last names and payment amount of all employees who had a payroll deduction on 1/15/2013. Give the result of the query.
36. Write an SQL query to reproduce the universal relation of Exercise 31 from the three relation tables.

SECTION 5.4 FUNCTIONS

In this section we discuss functions, which are really special cases of binary relations from a set S to a set T . This view of a function is a rather sophisticated one, however, and we will work up to it gradually.

Definition

Function is a common enough word even in nontechnical contexts. A newspaper may have an article on how starting salaries for this year’s college graduates have increased over those for last year’s graduates. The article might say something like, “The salary increase varies depending on the degree program,” or, “The salary increase is a function of the degree program.” It may illustrate this functional relationship with a graph like Figure 5.11. The graph shows that each degree program has some figure for the salary increase associated with it, that no degree program has more than one figure associated with it, and that both the physical sciences and the liberal arts have the same figure, 1.5%.

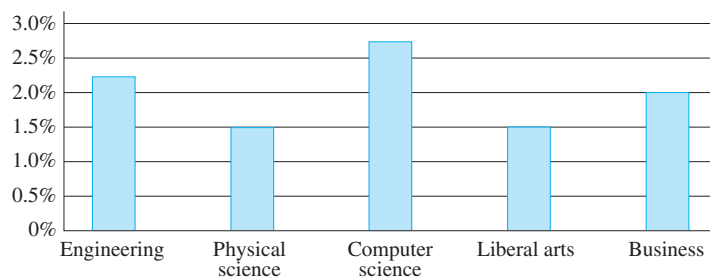


Figure 5.11

Of course, we also use mathematical functions in algebra and calculus. The equation $g(x) = x^3$ expresses a functional relationship between a value for x and the corresponding value that results when the value for x is used in the equation. Thus an x value of 2 has the number $2^3 = 8$ associated with it. (This number is expressed as $g(2) = 8$.) Similarly, $g(1) = 1^3 = 1$, $g(-1) = (-1)^3 = -1$, and so on. For each x value, the corresponding $g(x)$ value is unique. If we were to graph this function on a rectangular coordinate system, the points $(2, 8)$, $(1, 1)$, and $(-1, -1)$ would be points on the graph. If we allow x to take on any real number value, the resulting graph is the continuous curve shown in Figure 5.12.

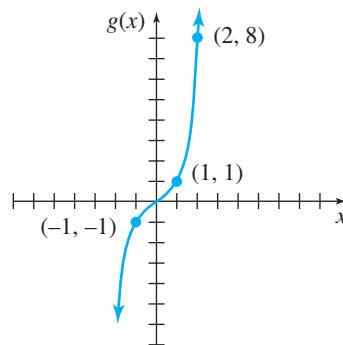


Figure 5.12

The function in the salary increase example could be described as follows. We set the stage by the diagram in Figure 5.13, which indicates that the function always starts with a given degree program and that a particular salary increase is associated with that degree program. The association itself is described by the set of ordered pairs $\{(\text{engineering}, 2.25\%), (\text{physical sciences}, 1.5\%), (\text{computer science}, 2.75\%), (\text{liberal arts}, 1.5\%), (\text{business}, 2.0\%)\}$.

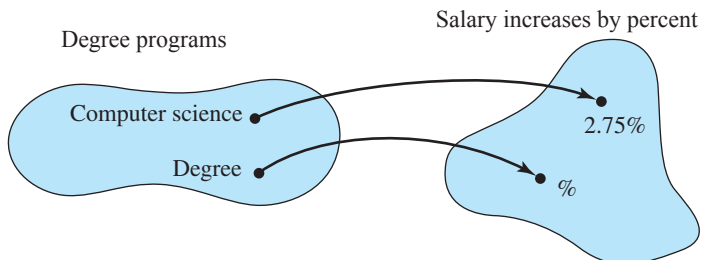


Figure 5.13

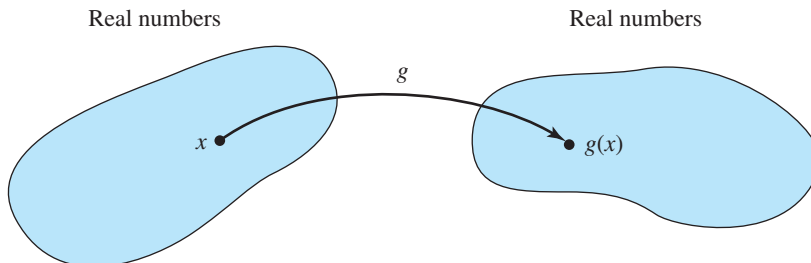


Figure 5.14

For the algebraic example $g(x) = x^3$, Figure 5.14 shows that the function always starts with a given real number and associates a second real number with it.

The association itself is described by $\{(x, g(x)) \mid g(x) = x^3\}$, or simply $g(x) = x^3$. This set includes $(2, 8)$, $(1, 1)$, $(-1, -1)$, but because it is an infinite set, we cannot list all its members; we have to describe them.

From the above examples, we can conclude that there are three parts to a function: (1) a set of starting values, (2) a set from which associated values come, and (3) the association itself. The set of starting values is called the *domain* of the function, and the set from which associated values come is called the *codomain* of the function. Thus both the domain and codomain represent pools from which values may be chosen. (This usage is consistent with our use of the word *domain* when discussing predicate wffs in Section 1.2. There the domain of an interpretation is a pool of values that variables can assume and to which constant symbols may be assigned. Similarly, the domain D_i of an attribute A_i in a database relation, discussed in Section 5.3, is a pool of potential values for the attribute.)

The picture for an arbitrary function f is shown in Figure 5.15. Here f is a function from S to T , symbolized $f: S \rightarrow T$. S is the domain and T is the codomain. The association itself is a set of ordered pairs, each of the form (s, t) where $s \in S$, $t \in T$, and t is the value from T that the function associates with the value s from S ; $t = f(s)$. Hence, the association is a subset of $S \times T$ (a binary relation from S to T). But the important property of this relation is that every member of S must have one and only one T value associated with it, so every $s \in S$ will appear exactly once as the first component of an (s, t) pair. (This property does not prevent a given T value from appearing more than once.)

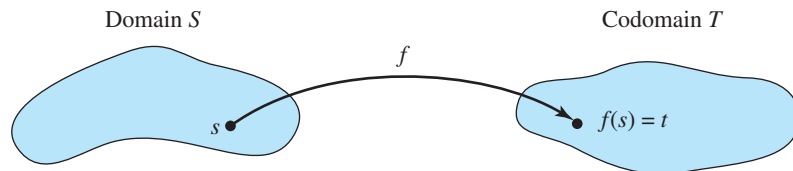


Figure 5.15

We are now ready for the formal definition of a function.

DEFINITIONS TERMINOLOGY FOR FUNCTIONS

Let S and T be sets. A **function (mapping)** f from S to T , $f: S \rightarrow T$, is a subset of $S \times T$ where each member of S appears exactly once as the first component of an ordered pair. S is the **domain** and T is the **codomain** of the function. If (s, t) belongs to the function, then t is denoted by $f(s)$; t is the **image** of s under f , s is a **preimage** of t under f , and f is said to map s to t . For $A \subseteq S$, $f(A)$ denotes $\{f(a) \mid a \in A\}$.

A function from S to T is a subset of $S \times T$ with certain restrictions on the ordered pairs it contains. That is why we spoke of a function as a special kind of binary relation. By the definition of a function, a binary relation that is one-to-many (or many-to-many) cannot be a function. Also, each member of S must be used as a first component.

We have talked a lot about values from the sets S and T , but as our example of salary increases shows, these values are not necessarily numbers, nor is the association itself necessarily described by an equation.

PRACTICE 23 Which of the following formulas are functions from the domain to the codomain indicated? For those that are not, why not?

- $f: S \rightarrow T$ where $S = T = \{1, 2, 3\}$, $f = \{(1, 1), (2, 3), (3, 1), (2, 1)\}$
- $g: \mathbb{Z} \rightarrow \mathbb{N}$ where g is defined by $g(x) = |x|$ (the absolute value of x)
- $h: \mathbb{N} \rightarrow \mathbb{N}$ where h is defined by $h(x) = x - 4$
- $f: S \rightarrow T$ where S is the set of all people in your hometown, T is the set of all automobiles, and f associates with each person the automobile that person owns
- $g: S \rightarrow T$ where $S = \{2013, 2014, 2015, 2016\}$, $T = \{\$20,000, \$30,000, \$40,000, \$50,000, \$60,000\}$, and g is defined by the graph in Figure 5.16.
- $h: S \rightarrow T$ where S is the set of all quadratic polynomials in x with integer coefficients, $T = \mathbb{Z}$, and h is defined by $h(ax^2 + bx + c) = b + c$
- $f: \mathbb{R} \rightarrow \mathbb{R}$ where f is defined by $f(x) = 4x - 1$
- $g: \mathbb{N} \rightarrow \mathbb{N}$ where g is defined by

$$g(x) = \begin{cases} x + 3 & \text{if } x \geq 5 \\ x & \text{if } x \leq 5 \end{cases}$$

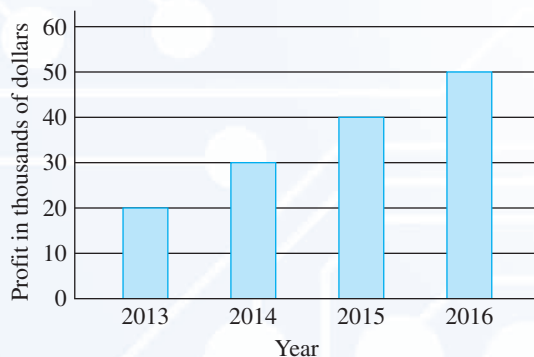


Figure 5.16 Profits of the American Earthworm Corp.

PRACTICE 24 Let $f: \mathbb{Z} \rightarrow \mathbb{Z}$ be defined by $f(x) = x^2$.

- What is the image of -4 ?
- What are the preimages of 9 ?

EXAMPLE 27

When we studied recursive definitions in Section 3.1, we talked about sequences, where a sequence S was written as

$$S(1), S(2), S(3), \dots$$

Changing the notation to

$$f(1), f(2), f(3), \dots$$

we see that a sequence is nothing but a list of functional values for a function f whose domain is the positive integers, and this is how a sequence is often defined.

Indeed, the algorithms we gave for computing the values in such sequences were pseudocode that computes the function.

Also in Section 3.1, we talked about recursive operations such as a^n where a is a fixed nonzero real number and $n \geq 0$. This is also simply a function $f(n) = a^n$ whose domain is \mathbb{N} .

The definition of a function includes functions of more than one variable. We can have a function $f: S_1 \times S_2 \times \cdots \times S_n \rightarrow T$ that associates with each ordered n -tuple of elements (s_1, s_2, \dots, s_n) , $s_i \in S_i$, a unique element of T .

EXAMPLE 28

$f: \mathbb{Z} \times \mathbb{N} \times \{1, 2\} \rightarrow \mathbb{Z}$ is given by $f(x, y, z) = x^y + z$. Then $f(-4, 3, 1) = (-4)^3 + 1 = -64 + 1 = -63$.

EXAMPLE 29

In Section 4.1 we defined a unary operation on a set S as associating a unique member of S , $x^\#$, with each member x of S . This means that a unary operation on S is a function with domain and codomain S . We also defined a binary operation \circ on a set S as associating a unique member of S , $x \circ y$, with every (x, y) pair of elements of S . Therefore a binary operation on S is a function with domain $S \times S$ and codomain S .

Again, domain values and codomain values are not always numbers.

EXAMPLE 30

Let S be the set of all character strings of finite length. Then the association that pairs each string with the number of characters in the string is a function with domain S and codomain \mathbb{N} (we allow the “empty string,” which has zero characters).

EXAMPLE 31

Any propositional wff with n statement letters defines a function with domain $\{T, F\}^n$ and codomain $\{T, F\}$. The domain consists of all n -tuples of T-F values; with each n -tuple is associated a single value of T or F. The truth table for the wff gives the association. For example, if the wff is $A \vee B'$, then the truth table

A	B	B'	A \vee B'
T	T	F	T
T	F	T	T
F	T	F	F
F	F	T	T

says that the image of the 2-tuple (F, T) under this function is F. If we call this function f , then $f(F, T) = F$.

PRACTICE 25

Let the function defined by the wff $A \wedge (B \vee C')$ be denoted by f . What is $f(T, T, F)$? What is $f(F, T, F)$?

The next example defines two functions that are sometimes useful in analyzing algorithms.

EXAMPLE 32

The **floor function** $\lfloor x \rfloor$ associates with each real number x the greatest integer less than or equal to x . The **ceiling function** $\lceil x \rceil$ associates with each real number x the smallest integer greater than or equal to x . Thus $\lfloor 2.8 \rfloor = 2$, $\lceil 2.8 \rceil = 3$, $\lfloor -4.1 \rfloor = -5$, and $\lceil -4.1 \rceil = -4$. Both the floor function and the ceiling function are functions from \mathbb{R} to \mathbb{Z} .

PRACTICE 26

- Sketch a graph of the function $\lfloor x \rfloor$.
- Sketch a graph of the function $\lceil x \rceil$.

EXAMPLE 33

For any integer x and any positive integer n , the **modulo function**, denoted by $f(x) = x \bmod n$, associates with x the nonnegative remainder when x is divided by n . We can write x as $x = qn + r$, $0 \leq r < n$, where q is the quotient and r is the remainder, so the value of $x \bmod n$ is r .

$$25 = 12 \cdot 2 + 1 \text{ so } 25 \bmod 2 = 1$$

$$21 = 3 \cdot 7 + 0 \text{ so } 21 \bmod 7 = 0$$

$$15 = 3 \cdot 4 + 3 \text{ so } 15 \bmod 4 = 3$$

$$-17 = (-4) \cdot 5 + 3 \text{ so } -17 \bmod 5 = 3$$

(it is true that $-17 = (-3)5 + (-2)$ but remember that the remainder must be nonnegative)

Section 5.6 discusses some of the many applications of the modulo function.

The definition of a function $f: S \rightarrow T$ includes three parts—the domain set S , the codomain set T , and the association itself. Is all this necessary? Why can't we simply write an equation, like $g(x) = x^3$, to define a function?

The quickest answer is that not all functional associations can be described by an equation (see Example 30, for instance). But there is more to it—let's limit our attention to situations where an equation can be used to describe the association, such as $g: \mathbb{R} \rightarrow \mathbb{R}$ where $g(x) = x^3$. Even in algebra and calculus, it is common to say "consider the function $g(x) = x^3$," implying that the equation *is* the function. Technically, the equation only describes a way to compute associated values. The function $h: \mathbb{R} \rightarrow \mathbb{R}$ given by $h(x) = x^3 - 3x + 3(x + 5) - 15$ is the same function as g because it contains the same ordered pairs. However, the equation is different in that it says to process any given x value differently.

On the other hand, the function $f: \mathbb{Z} \rightarrow \mathbb{R}$ given by $f(x) = x^3$ is not the same function as g . The domain has been changed, which changes the set of ordered pairs. The graph of $f(x)$ would consist of discrete (separate) points (Figure 5.17). Most of the functions in which we are interested have this feature. Even in situations where one quantity varies continuously with another, in a digital computer we approximate by taking data at discrete, small intervals, much as the graph of $g(x)$ (see Figure 5.12) is approximated by the graph of $f(x)$ (see Figure 5.17).

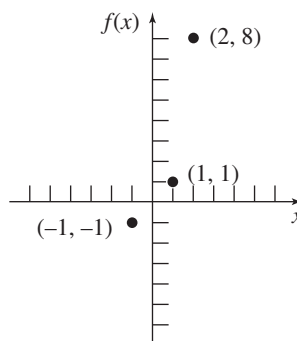


Figure 5.17

Finally, let's look at the function $k: \mathbb{R} \rightarrow \mathbb{C}$ given by $k(x) = x^3$. The equation and domain is the same as for $g(x)$; the codomain has been enlarged, but the change does not affect the ordered pairs. Is this function considered the same function as $g(x)$? It is not, but to see why, we'll have to wait until we discuss the *onto* property of functions. Then we will see that g has the onto property while k does not, so we do not want to consider them the same function.

In summary, a complete definition of a function requires giving its domain, its codomain, and the association, where the association may be given by a verbal description, a graph, an equation, or a collection of ordered pairs.

● **DEFINITION** **EQUAL FUNCTIONS**

Two functions are **equal** if they have the same domain, the same codomain, and the same association of values of the codomain with values of the domain.

Suppose we are trying to show that two functions with the same domain and the same codomain are equal. Then we must show that the associations are the same. This can be done by showing that, given an arbitrary element of the domain, both functions produce the same associated value for that element; that is, they map it to the same place.

PRACTICE 27 Let $S = \{1, 2, 3\}$ and $T = \{1, 4, 9\}$. The function $f: S \rightarrow T$ is defined by $f = \{(1, 1), (2, 4), (3, 9)\}$. The function $g: S \rightarrow T$ is defined by the equation

$$g(n) = \frac{\sum_{k=1}^n (4k - 2)}{2}$$

Prove that $f = g$. ■

Properties of Functions

Onto Functions

Let $f: S \rightarrow T$ be an arbitrary function with domain S and codomain T (Figure 5.18). Part of the definition of a function is that every member of S has an image under f and that all the images are members of T ; the set R of all such images is called the **range** of the function f . Thus, $R = \{f(s) \mid s \in S\}$, or $R = f(S)$. Clearly, $R \subseteq T$; the range R is shaded in Figure 5.19. If it should happen that $R = T$, that is, that the range coincides with the codomain, then the function is called an *onto* function.

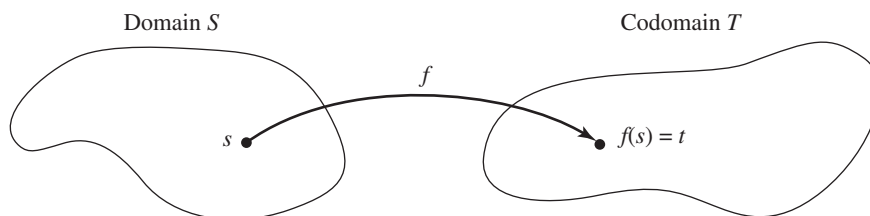


Figure 5.18

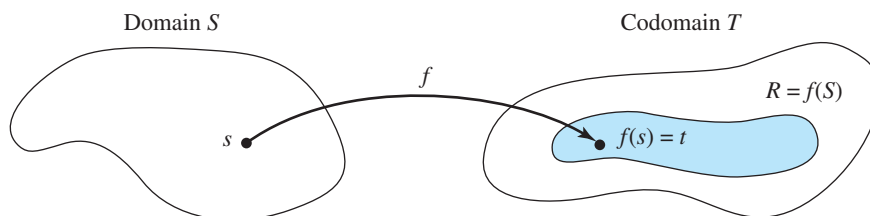


Figure 5.19

DEFINITION ONTO (SURJECTIVE) FUNCTION

A function $f: S \rightarrow T$ is an **onto** or **surjective** function if the range of f equals the codomain of f .

REMINDER

To show that a function is onto, pick an arbitrary element in the codomain and show that it has a preimage in the domain.

In every function with range R and codomain T , $R \subseteq T$. To prove that a given function is onto, we must show that $T \subseteq R$; then it will be true that $R = T$. We must therefore show that an arbitrary member of the codomain is a member of the range, that is, that it is the image of some member of the domain. On the other hand, if we can produce one member of the codomain that is not the image of any member of the domain, then we have proved that the function is not onto.

EXAMPLE 34

The function $g: \mathbb{R} \rightarrow \mathbb{R}$ defined by $g(x) = x^3$ is an onto function. To prove that $g(x)$ is onto, let r be an arbitrary real number, and let $x = \sqrt[3]{r}$. Then x is a real number, so x belongs to the domain of g and $g(x) = (\sqrt[3]{r})^3 = r$. Hence, any member of the codomain is the image under g of a member of the domain. The function $k: \mathbb{R} \rightarrow \mathbb{C}$ given by $k(x) = x^3$ is not onto. There are many complex numbers (i , for example) that cannot be obtained by cubing a real number. Thus, g and k are not equal functions.

EXAMPLE 35

Let $f: \mathbb{Q} \rightarrow \mathbb{Q}$ be defined by $f(x) = 3x + 2$. To test whether f is onto, let $q \in \mathbb{Q}$. We want an $x \in \mathbb{Q}$ such that $f(x) = 3x + 2 = q$. When we solve this equation for x , we find that $x = (q - 2)/3$ is the only possible value and is indeed a member of \mathbb{Q} . Thus, q is the image of a member of \mathbb{Q} under f , and f is onto. However, the function $h: \mathbb{Z} \rightarrow \mathbb{Q}$ defined by $h(x) = 3x + 2$ is not onto because there are many values $q \in \mathbb{Q}$, for example 0, for which the equation $3x + 2 = q$ has no integer solution.

PRACTICE 28

Which of the functions found in Practice 23 are onto functions?

PRACTICE 29

Suppose a function $f: \{T, F\}^n \rightarrow \{T, F\}$ is defined by a propositional wff P (see Example 31).

Give the two conditions on P under each of which f will fail to be an onto function.

One-to-One Functions

The definition of a function guarantees a unique image for every member of the domain. A given member of the range may have more than one preimage, however. In our very first example of a function (salary increases), both physical sciences and liberal arts were preimages of 1.5%. This function was not one-to-one.

DEFINITION ONE-TO-ONE (INJECTIVE) FUNCTION

A function $f: S \rightarrow T$ is **one-to-one**, or **injective**, if no member of T is the image under f of two distinct elements of S .

REMINDER

To show that a function f is one-to-one, assume $f(s_1) = f(s_2)$ and show that $s_1 = s_2$.

The one-to-one idea here is the same as for binary relations in general, as discussed in Section 5.1, except that every element of S must appear as a first component in an ordered pair. To prove that a function is one-to-one, we assume that there are elements s_1 and s_2 of S with $f(s_1) = f(s_2)$ and then show that $s_1 = s_2$. To prove that a function is not one-to-one, we produce a counterexample, an element in the range with two preimages in the domain.

EXAMPLE 36

The function $g: \mathbb{R} \rightarrow \mathbb{R}$ defined by $g(x) = x^3$ is one-to-one because if x and y are real numbers with $g(x) = g(y)$, then $x^3 = y^3$ and $x = y$. The function $f: \mathbb{R} \rightarrow \mathbb{R}$ given by $f(x) = x^2$ is not one-to-one because, for example, $f(2) = f(-2) = 4$. However, the function $h: \mathbb{N} \rightarrow \mathbb{N}$ given by $h(x) = x^2$ is one-to-one because if x and y are nonnegative integers with $h(x) = h(y)$, then $x^2 = y^2$; because x and y are both nonnegative, $x = y$.

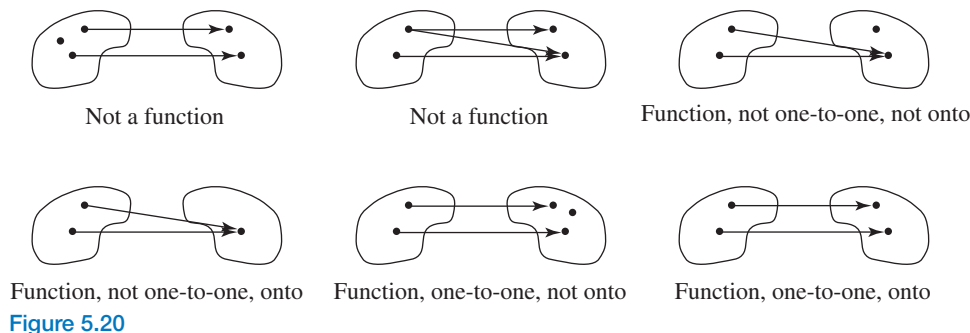
PRACTICE 30

Which of the functions found in Practice 23 are one-to-one functions?

EXAMPLE 37

The floor function and the ceiling function of Example 32 are clearly not one-to-one. This is evident also in the graphs of these functions (Practice 26), which have a number of horizontal sections, indicating that many different domain values in \mathbb{R} are mapped by the function to the same codomain value in \mathbb{Z} . ●

Figure 5.20 gives simple illustrations about functions and their properties. In each case, the domain is on the left and the codomain is on the right.

**Bijections****DEFINITION BIJECTIVE FUNCTION**

A function $f: S \rightarrow T$ is **bijective** (a **bijection**) if it is both one-to-one and onto. ●

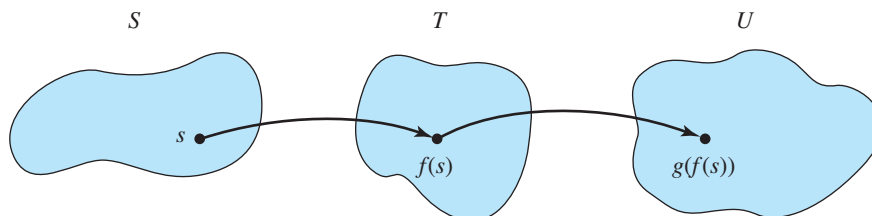
EXAMPLE 38

The function $g: \mathbb{R} \rightarrow \mathbb{R}$ given by $g(x) = x^3$ is a bijection. The function in part (g) of Practice 23 is a bijection. The function $f: \mathbb{R} \rightarrow \mathbb{R}$ given by $f(x) = x^2$ is not a bijection (not one-to-one), and neither is the function $k: \mathbb{R} \rightarrow \mathbb{C}$ given by $k(x) = x^3$ (not onto). ●

Composition of Functions**REMINDER**

To prove that a function is a bijection requires proving two things—onto and one-to-one.

Suppose that f and g are functions with $f: S \rightarrow T$ and $g: T \rightarrow U$. Then for any $s \in S$, $f(s)$ is a member of T , which is also the domain of g . Thus, the function g can be applied to $f(s)$. The result is $g(f(s))$, a member of U (Figure 5.21). Taking an arbitrary member s of S , applying the function f , and then applying the function g to $f(s)$ is the same as associating a unique member of U with s . In short, we have created a function $S \rightarrow U$, called the composition function of f and g and denoted by $g \circ f$ (Figure 5.22).



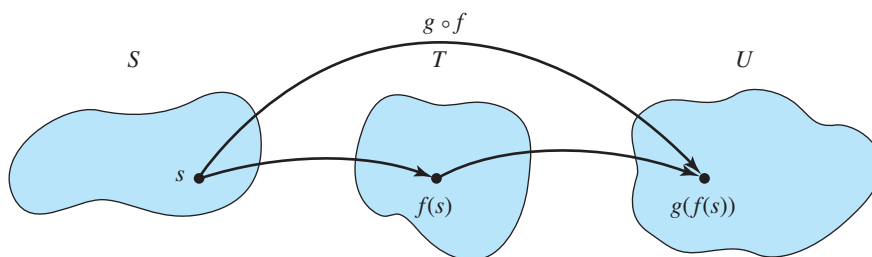


Figure 5.22

● **DEFINITION** **COMPOSITION FUNCTION**

Let $f: S \rightarrow T$ and $g: T \rightarrow U$. Then the **composition function**, $g \circ f$, is a function from S to U defined by $(g \circ f)(s) = g(f(s))$.

Note that the function $g \circ f$ is applied right to left; function f is applied first and then function g .

The diagram in Figure 5.23 also illustrates the definition of the composition function. The corners indicate the domains and codomains of the three functions. The diagram says that, starting with an element of S , if we follow either path $g \circ f$ or path f followed by path g , we get to the same element in U . Diagrams illustrating that alternate paths produce the same effect are called **commutative diagrams**.

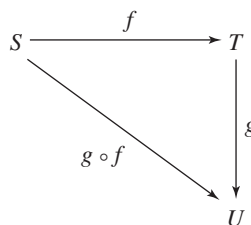


Figure 5.23

It is not always possible to take any two arbitrary functions and compose them; the domains and ranges have to be “compatible.” For example, if $f: S \rightarrow T$ and $g: W \rightarrow Z$, where T and W are disjoint, then $(g \circ f)(s) = g(f(s))$ is undefined because $f(s)$ is not in the domain of g .

PRACTICE 31 Let $f: \mathbb{R} \rightarrow \mathbb{R}$ be defined by $f(x) = x^2$. Let $g: \mathbb{R} \rightarrow \mathbb{R}$ be defined by $g(x) = \lfloor x \rfloor$.

- What is the value of $(g \circ f)(2.3)$?
- What is the value of $(f \circ g)(2.3)$?

From Practice 31 we see that order is important in function composition, which should not be surprising. If you make a deposit in your checking account and then write a large check, the effect is not the same as if you write a large check and later make a deposit! Your bank is very sensitive to these differences.

Function composition preserves the properties of being onto and being one-to-one. Again, let $f: S \rightarrow T$ and $g: T \rightarrow U$, but also suppose that both f and g are onto functions. Then the composition function $g \circ f$ is also onto. Recall that $g \circ f: S \rightarrow U$, so we must pick an arbitrary $u \in U$ and show that it has a preimage under $g \circ f$ in S . Because g is onto, there exists $t \in T$ such that $g(t) = u$. And because f is onto, there exists $s \in S$ such that $f(s) = t$. Then $(g \circ f)(s) = g(f(s)) = g(t) = u$, and $g \circ f$ is an onto function.

PRACTICE 32 Let $f: S \rightarrow T$ and $g: T \rightarrow U$, and assume that both f and g are one-to-one functions.

Prove that $g \circ f$ is a one-to-one function. (*Hint:* Assume that $(g \circ f)(s_1) = (g \circ f)(s_2)$.) ■

We have now proved the following theorem.

● **THEOREM ON COMPOSING TWO BIJECTIONS**
The composition of two bijections is a bijection.

Inverse Functions

Bijjective functions have another important property. Let $f: S \rightarrow T$ be a bijection. Because f is onto, every $t \in T$ has a preimage in S . Because f is one-to-one, that preimage is unique. We can associate with each element t of T a unique member of S , namely, that $s \in S$ such that $f(s) = t$. This association describes a function $g: T \rightarrow S$. The picture for f and g is given in Figure 5.24. The domains and codomains of g and f are such that we can form both $g \circ f: S \rightarrow S$ and $f \circ g: T \rightarrow T$. If $s \in S$, then $(g \circ f)(s) = g(f(s)) = g(t) = s$. Thus, $g \circ f$ maps each element of S to itself. The function that maps each element of a set S to itself, that is, that leaves each element of S unchanged, is called the **identity function** on S and denoted by i_S . Hence, $g \circ f = i_S$.

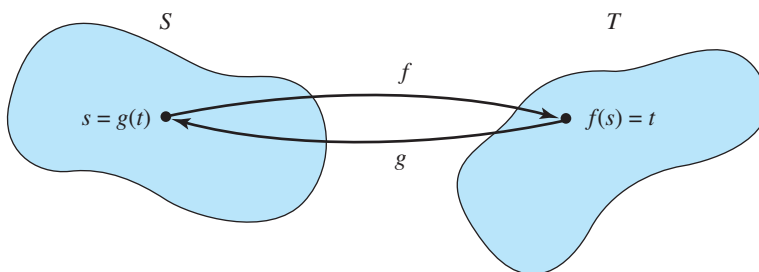


Figure 5.24

PRACTICE 33 Show that $f \circ g = i_T$. ■

We have now seen that if f is a bijection, $f: S \rightarrow T$, then there is a function $g: T \rightarrow S$ with $g \circ f = i_S$ and $f \circ g = i_T$. The converse is also true. To prove the converse, suppose $f: S \rightarrow T$ and there exists $g: T \rightarrow S$ with $g \circ f = i_S$ and $f \circ g = i_T$. We can prove that f is a bijection. To show that f is onto, let $t \in T$. Then $t = i_T(t) = (f \circ g)(t) = f(g(t))$. Because $g: T \rightarrow S$, $g(t) \in S$, and $g(t)$ is the preimage under f of t . To show that f is one-to-one, suppose $f(s_1) = f(s_2)$. Then $g(f(s_1)) = g(f(s_2))$ and $(g \circ f)(s_1) = (g \circ f)(s_2)$ implying $i_S(s_1) = i_S(s_2)$, or $s_1 = s_2$. Thus, f is a bijection.

● **DEFINITION** **INVERSE FUNCTION**

Let f be a function, $f: S \rightarrow T$. If there exists a function $g: T \rightarrow S$ such that $g \circ f = i_S$ and $f \circ g = i_T$, then g is called the **inverse function** of f , denoted by f^{-1} .

We have proved the following theorem.

● **THEOREM** **ON BIJECTIONS AND INVERSE FUNCTIONS**

Let $f: S \rightarrow T$. Then f is a bijection if and only if f^{-1} exists.

Actually, we have been a bit sneaky in talking about *the* inverse function of f . What we have shown is that if f is a bijection, this is equivalent to the existence of *an* inverse function. But it is easy to see that there is only one such inverse function. *When you want to prove that something is unique, the standard technique is to assume that there are two different such things and then obtain a contradiction.* Thus, suppose f has two inverse functions, f_1^{-1} and f_2^{-1} (existence of either means that f is a bijection). Both f_1^{-1} and f_2^{-1} are functions from T to S ; if they are not the same function, then they must act differently somewhere. Assume that there is a $t \in T$ such that $f_1^{-1}(t) \neq f_2^{-1}(t)$. Because f is one-to-one, it follows that $f(f_1^{-1}(t)) \neq f(f_2^{-1}(t))$, or $(f \circ f_1^{-1})(t) \neq (f \circ f_2^{-1})(t)$. But both $f \circ f_1^{-1}$ and $f \circ f_2^{-1}$ are i_T , so $t \neq t$, which is a contradiction. We are therefore justified in speaking of f^{-1} as *the* inverse function of f . If f is a bijection, so that f^{-1} exists, then f is the inverse function for f^{-1} ; therefore, f^{-1} is also a bijection.

PRACTICE 34 $f: \mathbb{R} \rightarrow \mathbb{R}$ given by $f(x) = 3x + 4$ is a bijection.

Describe f^{-1} . ■

We've introduced a lot of terminology about functions. Table 5.2 gives an informal summary of these terms.

TABLE 5.2	
Term	Meaning
function	Mapping from one set to another that associates with each member of the starting set exactly one member of the ending set
domain	Starting set for a function
codomain	Ending set for a function
image	Point that results from a mapping
preimage	Starting point for a mapping
range	Collection of all images of the domain
onto (surjective)	Range is the whole codomain; every codomain element has a preimage
one-to-one (injective)	No two elements in the domain map to the same place
bijection	One-to-one and onto
identity function	Maps each element of a set to itself
inverse function	For a bijection, a new function that maps each codomain element back where it came from

Permutation Functions

Bijections that map a set to itself are given a special name.

● **DEFINITION PERMUTATIONS OF A SET**
 For a given set A , $S_A = \{f \mid f: A \rightarrow A \text{ and } f \text{ is a bijection}\}$. S_A is thus the set of all bijections of set A into (and therefore onto) itself; such functions are called **permutations** of A .

If f and g both belong to S_A , then they each have domain = range = A . Therefore the composition function $g \circ f$ is defined and maps $A \rightarrow A$. Furthermore, because f and g are both bijections, our theorem on composing bijections says that $g \circ f$ is a bijection, a (unique) member of S_A . Thus, function composition is a binary operation on the set S_A .

In Section 4.4 we described a permutation of objects in a set as being an ordered arrangement of those objects. Is this now a new use of the word “permutation”? Not exactly; permutation functions represent ordered arrangements of the objects in the domain. If $A = \{1, 2, 3, 4\}$, one permutation function of A , call it f , is given by $f = \{(1, 2), (2, 3), (3, 1), (4, 4)\}$. We can also describe function f in array form by listing the elements of the domain in a row and, directly beneath, the images of these elements under f . Thus,

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix}$$

The bottom row is an ordered arrangement of the objects in the top row.

A shorter way to describe the permutation f shown in array form is to use *cycle notation* and write $f = (1, 2, 3)$ —understood to mean that f maps each element listed to the one on its right, the last element listed to the first, and an element of the domain not listed to itself. Here 1 maps to 2, 2 maps to 3, and 3 maps to 1. The element 4 maps to itself because it does not appear in the cycle. The cycle $(2, 3, 1)$ also represents f . It says that 2 maps to 3, 3 maps to 1, 1 maps to 2, and 4 maps to itself, the same information as before. Similarly, $(3, 1, 2)$ also represents f .

PRACTICE 35

- a. Let $A = \{1, 2, 3, 4, 5\}$, and let $f \in S_A$ be given in array form by

$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 2 & 3 & 5 & 1 \end{pmatrix}$$

Write f in cycle form.

- b. Let $A = \{1, 2, 3, 4, 5\}$, and let $g \in S_A$ be given in cycle form by $g = (2, 4, 5, 3)$. Write g in array form. ■

If f and g are members of S_A for some set A , then $g \circ f \in S_A$, and the action of $g \circ f$ on any member of A is determined by applying function f and then function g . If f and g are cycles, $g \circ f$ is still computed the same way.

EXAMPLE 39

If $A = \{1, 2, 3, 4\}$ and $f, g \in S_A$ are given by $f = (1, 2, 3)$ and $g = (2, 3)$, then $g \circ f = (2, 3) \circ (1, 2, 3)$. But what does this composition function look like? Let's see what happens to element 1 of A . Working from right to left (first f , then g), $1 \rightarrow 2$ under f and then $2 \rightarrow 3$ under g , so $1 \rightarrow 3$ under $g \circ f$. If we want to write $g \circ f$ as a cycle, we see that it can start with

$$(1, 3)$$

and we next need to see what happens to 3. Under f , $3 \rightarrow 1$ and then under g , $1 \rightarrow 1$ (because 1 does not appear in the cycle notation for g), so $3 \rightarrow 1$ under $g \circ f$. Thus we can close the above cycle, writing it as $(1, 3)$. But what happens to 2 and 4? If we consider 2, $2 \rightarrow 3$ under f and then $3 \rightarrow 2$ under g , so $2 \rightarrow 2$ under $g \circ f$. Similarly, $4 \rightarrow 4$ under f and $4 \rightarrow 4$ under g , so $4 \rightarrow 4$ under $g \circ f$. We conclude that $g \circ f = (1, 3)$. ●

In Example 39, if we were to compute $f \circ g = (1, 2, 3) \circ (2, 3)$, we would get $(1, 2)$. (We already know that order is important in function composition.) If, however, f and g are members of S_A and f and g are **disjoint cycles**—the cycles have no elements in common—then $f \circ g = g \circ f$.

PRACTICE 36 Let $A = \{1, 2, 3, 4, 5\}$. Compute $g \circ f$ and $f \circ g$ for the following cycles in S_A .

- $f = (5, 2, 3)$; $g = (3, 4, 1)$. Write the answers in cycle form.
- $f = (1, 2, 3, 4)$; $g = (3, 2, 4, 5)$. Write the answers in array form.
- $f = (1, 3)$; $g = (2, 5)$. Write the answers in array form.

Let $A = \{1, 2, 3, 4\}$ and consider the cycle $f \in S_A$ given by $f = (1, 2)$. If we compute $f \circ f = (1, 2) \circ (1, 2)$, we see that each element of A is mapped to itself. The permutation that maps each element of A to itself is the identity function on A , i_A , also called the **identity permutation**.

If A is an infinite set, not every permutation of A can be written as a cycle. But even when A is a finite set, not every permutation of A can be written as a cycle; for example, the permutation $g \circ f$ of Practice 36(b) cannot be written as a cycle. However, every permutation on a finite set that is not the identity permutation can be written as a composition of one or more disjoint cycles. The permutation

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 2 & 5 & 1 & 3 \end{pmatrix}$$

of Practice 36(b) is $(1, 4) \circ (3, 5)$ or $(3, 5) \circ (1, 4)$.

PRACTICE 37 Write

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 5 & 1 & 3 & 6 \end{pmatrix}$$

as a composition of disjoint cycles.

Among the permutations of A , some will map certain elements of A to themselves, while others will so thoroughly mix elements around that no element in A is mapped to itself. A permutation on a set that maps no element to itself is called a **derangement**.

EXAMPLE 40 The permutation f on $A = \{1, 2, 3, 4, 5\}$ given in array form by

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 5 & 4 & 1 & 3 \end{pmatrix}$$

is a derangement. Members of S_A that are *not* derangements, if written as a cycle or a product of cycles, will have at least one element of A that is not listed. Thus $g \in S_A$ defined as $g = (1, 4) \circ (3, 5)$ maps 2 to itself, so g is not a derangement. ●

How Many Functions

Suppose S and T are finite sets, say $|S| = m$ and $|T| = n$. What can we say about the number of functions with various properties that map S to T ? First, let's just count the number of functions $f: S \rightarrow T$, assuming no special properties about the functions. The multiplication principle can be used here because we can think of defining a function by assigning an image to each of the m elements of S . This gives us a sequence of m tasks. Each task has n outcomes because each element of S can map to any element in T . Therefore the number of functions is

$$\underbrace{|n \times n \times n \times \cdots \times n|}_{m \text{ factors}} = n^m$$

How many one-to-one functions are there from S to T ? We must have $m \leq n$ or we can't have any one-to-one functions at all. (All the elements of S must be mapped to T , and if $m > n$ there are too many elements in S to allow for a one-to-one mapping. Actually, this is the pigeonhole principle at work.) We can again solve this problem by carrying out the sequence of tasks of assigning an image to each element in S , but this time we cannot use any image we have used before. By the multiplication principle, we have a product that begins with the factors

$$n(n-1)(n-2)\cdots$$

and must contain a total of m factors, so the result is

$$\begin{aligned} n(n-1)(n-2)\cdots [n-(m-1)] &= n(n-1)(n-2)\cdots(n-m+1) \\ &= \frac{n!}{(n-m)!} = P(n,m) \end{aligned}$$

How many onto functions are there from S to T ? This time we must have $m \geq n$ so that there are enough values in the domain to provide preimages for every value in the codomain. (By the definition of a function, an element in S cannot be a preimage of more than one element in T .) Our overall plan is to subtract the number of non-onto functions from the total number of functions, which we know. To count the number of non-onto functions, we'll use the principle of inclusion and exclusion.

Enumerate the elements of set T as t_1, \dots, t_n . For each i , $1 \leq i \leq n$, let A_i denote the set of functions from S to T that do not map anything to element t_i . (These sets are not disjoint, but every non-onto function belongs to at least one such set.) By the principle of inclusion and exclusion, we can write

$$\begin{aligned} |A_1 \cup \cdots \cup A_n| &= \sum_{1 \leq i \leq n} |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| + \sum_{1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k| \\ &\quad - \cdots + (-1)^{n+1} |A_1 \cap \cdots \cap A_n| \end{aligned} \quad (1)$$

For any i , $|A_i|$ is the number of functions that do not map anything to t_i but have no other restrictions. By the multiplication principle, we can count the number of such functions by counting for each of the m domain elements its $n - 1$ possible images. The result is that $|A_i| = (n - 1)^m$. Therefore the first summation in Equation (1) adds together terms that are all of the same size. There is one such term for each distinct individual set A_i out of the n sets, so there are $C(n, 1)$ such terms.

For any i and j , $|A_i \cap A_j|$ is the number of functions that do not map anything to t_i or t_j , leaving $n - 2$ possible images for each of the m elements of S . Thus $|A_i \cap A_j| = (n - 2)^m$. The second summation adds one such term for each distinct group of two sets out of n , so there are $C(n, 2)$ such terms.

A similar result holds for all the intersection terms. If there are k sets in the intersection, then there are $(n - k)^m$ functions in the intersection set and there are $C(n, k)$ distinct groups of k sets to form the intersection. Equation (1) can thus be written as

$$|A_1 \cup \cdots \cup A_n| = C(n, 1)(n - 1)^m - C(n, 2)(n - 2)^m + C(n, 3)(n - 3)^m - \cdots + (-1)^{n+1}C(n, n)(n - n)^m \quad (2)$$

Now the expression on the left of Equation (2) represents the number of all functions that fail to map to at least one of the elements of T , that is, all the non-onto functions. If we subtract the value of this expression from the total number of functions, which we know is n^m , we will have the number of onto functions. Thus the number of onto functions is

$$n^m - C(n, 1)(n - 1)^m + C(n, 2)(n - 2)^m - C(n, 3)(n - 3)^m + \cdots + (-1)^{n-1}C(n, n - 1)[n - (n - 1)]^m + (-1)^n C(n, n)(n - n)^m$$

where we've added the next-to-last term. The last term is zero, so the final answer is

$$n^m - C(n, 1)(n - 1)^m + C(n, 2)(n - 2)^m - C(n, 3)(n - 3)^m + \cdots + (-1)^{n-1} C(n, n - 1)(1)^m$$

We'll summarize these results.

THEOREM ON THE NUMBER OF FUNCTIONS WITH FINITE DOMAINS AND CODOMAINS

If $|S| = m$ and $|T| = n$, then

1. The number of functions $f: S \rightarrow T$ is n^m .
2. The number of one-to-one functions $f: S \rightarrow T$, assuming that $m \leq n$, is

$$\frac{n!}{(n - m)!}$$

3. The number of onto functions $f: S \rightarrow T$, assuming that $m \geq n$, is

$$n^m - C(n, 1)(n - 1)^m + C(n, 2)(n - 2)^m - C(n, 3)(n - 3)^m + \cdots + (-1)^{n-1}C(n, n - 1)(1)^m$$

EXAMPLE 41

Let $S = \{A, B, C\}$ and $T = \{a, b\}$. Find the number of functions from S onto T . Here $m = 3$ and $n = 2$. By our theorem on the number of functions, there are

$$2^3 - C(2, 1)(1)^3 = 8 - 2 \cdot 1 = 6$$

such functions. ●

PRACTICE 38

One of the six onto functions in Example 41 can be illustrated by the following diagram:



Draw diagrams for the remaining five onto functions. ■

If A is a set with $|A| = n$, then the number of permutations of A is $n!$. This number can be obtained by any of three methods:

1. A combinatorial argument (each of the n elements in the domain must map to one of the n elements in the range with no repetitions)
2. Thinking of such functions as permutations on a set with n elements and noting that $P(n, n) = n!$
3. Using result (2) in the previous theorem with $m = n$

We propose to count the number of derangements on A . Our plan is similar to the one we used in counting onto functions. We'll use the principle of inclusion and exclusion to compute the number of permutations that are not derangements and then subtract this value from the total number of permutation functions.

Enumerate the elements of set A as a_1, \dots, a_n . For each i , $1 \leq i \leq n$, let A_i be the set of all permutations that leave a_i fixed. (These sets are not disjoint, but every permutation that is not a derangement belongs to at least one such set.) By the principle of inclusion and exclusion, we can write

$$|A_1 \cup \dots \cup A_n| = \sum_{1 \leq i \leq n} |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| + \sum_{1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k| - \dots + (-1)^{n+1} |A_1 \cap \dots \cap A_n| \quad (3)$$

For any i , $|A_i|$ is the number of permutations that leave a_i fixed. By the multiplication principle we can count the number of such functions by counting for each of the n domain elements, beginning with a_i , its possible images. There is only one choice of where to map a_i because it must map to itself; the next element can map anywhere except to a_i , so there are $n - 1$ outcomes; the next element can map anywhere except the two images already used, so there are $n - 2$ outcomes, and so on. Continuing, there are

$$(1)(n - 1)(n - 2) \cdots (1) = (n - 1)!$$

elements in A_i for each i . Therefore the first summation in equation (3) adds together terms that are all of the same size. The number of such terms equals the number of ways to pick one set A_i out of the n such sets, or $C(n, 1)$.

In the second summation, the terms count the number of permutations on n elements that leave two of those elements fixed. There are

$$(1)(1)(n-2)\cdots(1) = (n-2)!$$

such functions in a given $A_i \cap A_j$ and $C(n, 2)$ ways to choose the two sets out of n . In general, if there are k sets in the intersection, then k elements must be held fixed, so there are $(n-k)!$ functions in the intersection set, and there are $C(n, k)$ ways to choose the k sets to form the intersection. Therefore equation (3) becomes

$$|A_1 \cup \cdots \cup A_n| = C(n, 1)(n-1)! - C(n, 2)(n-2)! + C(n, 3)(n-3)! \\ - \cdots + (-1)^{n+1}C(n, n)(n-n)!$$

This expression represents the number of all possible nonderangement permutations. We subtract this value from the total number of permutation functions, which is $n!$:

$$n! - C(n, 1)(n-1)! + C(n, 2)(n-2)! - C(n, 3)(n-3)! \\ + \cdots + (-1)^n C(n, n)(n-n)!$$

Rewriting this expression,

$$n! - \frac{n!}{1!(n-1)!}(n-1)! + \frac{n!}{2!(n-2)!}(n-2)! - \frac{n!}{3!(n-3)!}(n-3)! \\ + \cdots + (-1)^n \frac{n!}{n!0!} 0! \\ = n! - \frac{n!}{1!} + \frac{n!}{2!} - \frac{n!}{3!} + \cdots + (-1)^n \frac{n!}{n!} \\ = n! \left[1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!} \right] \quad (4)$$

EXAMPLE 42

For $n = 3$, Equation (4) says that the number of derangements is

$$3! \left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} \right) = \frac{3!}{2!} - \frac{3!}{3!} = 3 - 1 = 2$$

Written in array form, the two derangements are

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

Equivalent Sets

DEFINITIONS EQUIVALENT SETS AND CARDINALITY

A set S is **equivalent** to a set T if there exists a bijection $f: S \rightarrow T$. Two sets that are equivalent have the same **cardinality**.

The notion of equivalent sets allows us to extend our definition of cardinality from finite to infinite sets. The cardinality of a finite set is the number of elements in the set. If S is equivalent to T , then all the members of S and T are paired off by f in a one-to-one correspondence. If S and T are finite sets, this pairing off can happen only when S and T are the same size. With infinite sets, the idea of size gets a bit fuzzy, because we can sometimes prove that a given set is equivalent to what seems to be a smaller set. The cardinality of an infinite set is therefore given only in a comparative sense; for example, we may say that an infinite set A has (or does not have) the same cardinality as the set \mathbb{N} .

PRACTICE 39

Describe a bijection $f: \mathbb{Z} \rightarrow \mathbb{N}$, thus showing that \mathbb{Z} is equivalent to \mathbb{N} (\mathbb{Z} and \mathbb{N} have the same cardinality) even though $\mathbb{N} \subset \mathbb{Z}$.

If we have found a bijection between a set S and \mathbb{N} , we have established a one-to-one correspondence between the members of S and the nonnegative integers. We can then name the members of S according to this correspondence, writing s_0 for the value of S associated with 0, s_1 for the value of S associated with 1, and so on. Then the list

$$s_0, s_1, s_2, \dots$$

includes all the members of S . Since this list constitutes an enumeration of S , S is a denumerable set. Conversely, if S is denumerable, then a listing of the members of S exists and can be used to define a bijection between S and \mathbb{N} . Therefore a set is denumerable if and only if it is equivalent to \mathbb{N} .

For finite sets, we know that if S has n elements, then $\wp(S)$ has 2^n elements. Of course, $2^n > n$, and we cannot find a bijection between a set with n elements and a set with 2^n elements. Therefore S and $\wp(S)$ are not equivalent. This result is also true for infinite sets.

THEOREM CANTOR'S THEOREM

For any set S , S and $\wp(S)$ are not equivalent.

Proof: We will do a proof by contradiction and assume that S and $\wp(S)$ are equivalent. Let f be the bijection between S and $\wp(S)$. For any member s of S , $f(s)$ is a member of $\wp(S)$, so $f(s)$ is a set containing some members of S , possibly containing s itself. Now we define a set $X = \{x \in S \mid x \notin f(x)\}$. Because X is a subset of S , it is an element of $\wp(S)$ and therefore must be equal to $f(y)$ for some $y \in S$.

Then y either is or is not a member of X . If $y \in X$, then by the definition of X , $y \notin f(y)$, but since $f(y) = X$, then $y \in X$. On the other hand, if $y \notin X$, then since $X = f(y)$, $y \in f(y)$, and by the definition of X , $y \in X$. In either case, there is a contradiction, and our original assumption is incorrect. Therefore S and $\wp(S)$ are not equivalent. *End of Proof*

The proof of Cantor's theorem depends on the nature of set X , which was carefully constructed to provide the crucial contradiction. In this sense, the proof is similar to the diagonalization method (see Example 23 in Chapter 4) used to prove the existence of an uncountable set. Indeed, the existence of an uncountable set can be shown directly from Cantor's theorem.

EXAMPLE 43

The set \mathbb{N} is, of course, a denumerable set. By Cantor's theorem, the set $\wp(\mathbb{N})$ is not equivalent to \mathbb{N} and is therefore not a denumerable set, although it is clearly infinite.

SECTION 5.4 REVIEW**TECHNIQUES**

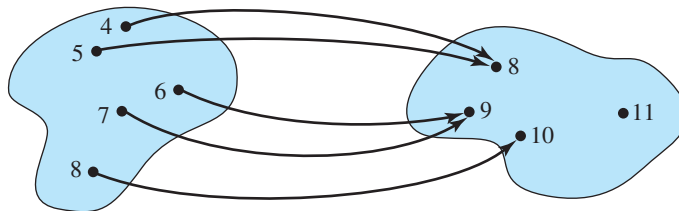
- Test whether a given relation is a function.
- W Test a function for being one-to-one or onto.
- Find the image of an element under function composition.
- W Write permutations of a set in array or cycle form.
- Count the number of functions, one-to-one functions, and onto functions from one finite set to another.

MAIN IDEAS

- The concept of function, especially bijective function, is extremely important.
- Composition of functions preserves bijectiveness.
- The inverse function of a bijection is itself a bijection.
- Permutations are bijections on a set.

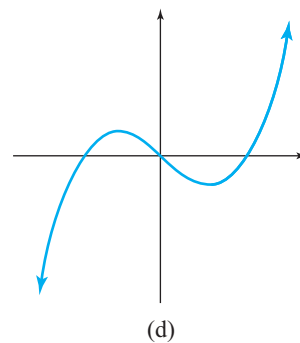
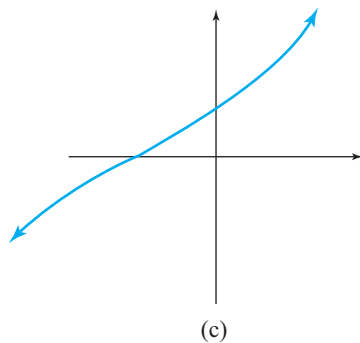
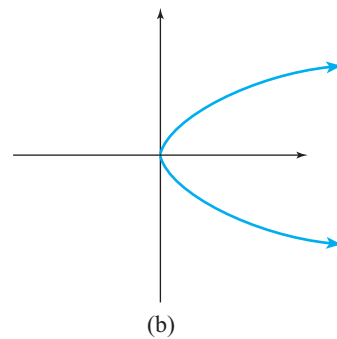
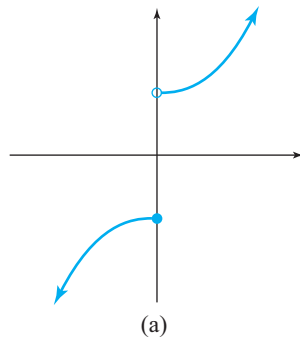
EXERCISES 5.4

1. The accompanying figure represents a function.



- a. What is the domain? What is the codomain? What is the range?
- b. What is the image of 5? of 8?
- c. What are the preimages of 9?
- d. Is this an onto function? Is it one-to-one?

2. The accompanying figure illustrates various binary relations from \mathbb{R} to \mathbb{R} . Which are functions? For those that are functions, which are onto? Which are one-to-one?



3. Using the equation $f(x) = 2x - 1$ to describe the functional association, write the function as a set of ordered pairs if the codomain is R and the domain is
- $S = \{0, 1, 2\}$.
 - $S = \{1, 2, 4, 5\}$.
 - $S = \{\sqrt{7}, 1.5\}$.
4. Using the equation $f(x) = x^2 + 1$ to describe the functional association, write the function as a set of ordered pairs if the codomain is \mathbb{Z} and the domain is
- $S = \{1, 5\}$.
 - $S = \{-1, 2, -2\}$.
 - $S = \{-\sqrt{12}, 3\}$.
5. If $f: \mathbb{Z} \rightarrow \mathbb{Z}$ is defined by $f(x) = 3x$, find $f(A)$ for
- $A = \{1, 3, 5\}$.
 - $A = \{x \mid x \in \mathbb{Z} \text{ and } (\exists y)(y \in \mathbb{Z} \text{ and } x = 2y)\}$.
6. If $f: \mathbb{R} \rightarrow \mathbb{R}$ is defined by $f(x) = x^2$, describe
- $f(\mathbb{N})$.
 - $f(\mathbb{Z})$.
 - $f(\mathbb{R})$.

7. The function $f: \{\text{all English words}\} \rightarrow \mathbb{Z}$. In each case, find $f(S)$.
- $S = \{\text{dog, cat, buffalo, giraffe}\}$, $f(x) = \text{the number of characters in } x$
 - $S = \{\text{goose, geese, moose, Mississippi}\}$, $f(x) = \text{the number of double-letter pairs in } x$
 - $S = \{\text{cheetah, seal, porpoise, koala}\}$, $f(x) = \text{the number of e's in } x$
8. The function $f: \{\text{binary strings}\} \rightarrow \{\text{binary strings}\}$. In each case, find $f(S)$.
- $S = \{000, 1011, 10001\}$, $f(x) = \text{the second bit in } x$
 - $S = \{111, 100, 0111\}$, $f(x) = \text{the binary string that is the sum of the first and last bit}$
 - $S = \{001, 11, 101\}$, $f(x) = \text{the binary string that is equal to } x + 1$
9. True or false:
- An onto function means that every element in the codomain must have a unique preimage.
 - A one-to-one function means that every element in the codomain must have a unique preimage.
 - A one-to-one function means that no two elements in the domain map to the same element in the codomain.
 - An onto function means that $(\text{the range}) \cap (\text{the codomain}) = \emptyset$.
10. True or false:
- If every element in the domain has an image, it must be an onto function.
 - If every element in the codomain has an image, it must be an onto function.
 - If every element in the codomain has a preimage, it must be an onto function.
 - If the domain is the larger than the codomain, it can't be a one-to-one function.
11. Let $S = \{0, 2, 4, 6\}$ and $T = \{1, 3, 5, 7\}$. Determine whether each of the following sets of ordered pairs is a function with domain S and codomain T . If so, is it one-to-one? Is it onto?
- $\{(0, 2), (2, 4), (4, 6), (6, 0)\}$
 - $\{(6, 3), (2, 1), (0, 3), (4, 5)\}$
 - $\{(2, 3), (4, 7), (0, 1), (6, 5)\}$
 - $\{(2, 1), (4, 5), (6, 3)\}$
 - $\{(6, 1), (0, 3), (4, 1), (0, 7), (2, 5)\}$
12. For any bijections in Exercise 11, describe the inverse function.
13. Let $S = \text{the set of all U. S. citizens alive today}$. Which of the following are functions from domain S to the codomain given? Which functions are one-to-one? Which functions are onto?
- Codomain = the alphabet, $f(\text{person}) = \text{initial of person's middle name}$
 - Codomain = the set of dates between January 1 and December 31, $f(\text{person}) = \text{person's date of birth}$
 - Codomain = 9-digit numbers, $f(\text{person}) = \text{person's Social Security number}$
14. Let $S = \text{the set of people at a meeting}$, let $T = \text{the set of all shoes in the room}$. Let $f(x) = \text{the left shoe } x \text{ is wearing}$.
- Is this a function?
 - Is it one-to-one?
 - Is it onto?

15. Which of the following definitions describe functions from the domain to the codomain given? Which functions are one-to-one? Which functions are onto? Describe the inverse function for any bijective function.
- $f: \mathbb{Z} \rightarrow \mathbb{N}$ where f is defined by $f(x) = x^2 + 1$
 - $g: \mathbb{N} \rightarrow \mathbb{Q}$ where g is defined by $g(x) = 1/x$
 - $h: \mathbb{Z} \times \mathbb{N} \rightarrow \mathbb{Q}$ where h is defined by $h(z, n) = z/(n + 1)$
 - $f: \{1, 2, 3\} \rightarrow \{p, q, r\}$ where $f = \{(1, q), (2, r), (3, p)\}$
 - $g: \mathbb{N} \rightarrow \mathbb{N}$ where g is given by $g(x) = 2^x$
 - $h: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ where h is defined by $h(x, y) = (y + 1, x + 1)$
16. Which of the following definitions describe functions from the domain to the codomain given? Which functions are one-to-one? Which functions are onto? Describe the inverse function for any bijective function.
- $f: \mathbb{Z}^2 \rightarrow \mathbb{N}$ where f is defined by $f(x, y) = x^2 + 2y^2$
 - $f: \mathbb{N} \rightarrow \mathbb{N}$ where f is defined by $f(x) = \begin{cases} x/2 & \text{if } x \text{ is even} \\ x + 1 & \text{if } x \text{ is odd} \end{cases}$
 - $g: \mathbb{R} \rightarrow \mathbb{R}$ where g is defined by $g(x) = 1/\sqrt{x + 1}$
 - $f: \mathbb{N} \rightarrow \mathbb{N}$ where f is defined by $f(x) = \begin{cases} x + 1 & \text{if } x \text{ is even} \\ x - 1 & \text{if } x \text{ is odd} \end{cases}$
 - $h: \mathbb{N}^3 \rightarrow \mathbb{N}$ where h is given by $h(x, y, z) = x + y - z$
 - $g: \mathbb{N}^2 \rightarrow \mathbb{N}^3$ where g is defined by $g(x, y) = (y, x, 0)$
17. Let $f: \mathbb{R} \rightarrow \mathbb{R}$ be defined by $f(x) = x^n$, where n is a fixed, positive integer. For what values of n is f bijective?
18. Let $f: \mathbb{R} \rightarrow \mathbb{R}$ be defined by $f(x) = n2^x$, where n is a fixed, positive integer. For what values of n is f an onto function?
19. Let $A = \{x, y\}$ and let A^* be the set of all strings of finite length made up of symbols from A . A function $f: A^* \rightarrow \mathbb{Z}$ is defined as follows: For s in A^* , $f(s) =$ the length of s . Is f one-to-one? Prove or disprove. Is f onto? Prove or disprove.
20. Let $A = \{x, y\}$ and let A^* be the set of all strings of finite length made up of symbols from A . A function $f: A^* \rightarrow \mathbb{Z}$ is defined as follows: For s in A^* , $f(s) =$ the number of x 's minus the number of y 's. Is f one-to-one? Prove or disprove. Is f onto? Prove or disprove.
21. Let $A = \{x, y\}$ and let A^* be the set of all strings of finite length made up of symbols from A . A function $f: A^* \rightarrow A^*$ is defined as follows: For s in A^* , $f(s)$ is the string obtained by writing the characters of s in reverse order. Is f one-to-one? Prove or disprove. Is f onto? Prove or disprove.
22. Let $A = \{x, y\}$ and let A^* be the set of all strings of finite length made up of symbols from A . A function $f: A^* \rightarrow A^*$ is defined as follows: For s in A^* , $f(s) = xs$ (the single-character string x followed by s). Is f one-to-one? Prove or disprove. Is f onto? Prove or disprove.
23. Let P be the power set of $\{a, b, c\}$. A function $f: P \rightarrow \mathbb{Z}$ is defined as follows: For A in P , $f(A) =$ the number of elements in A . Is f one-to-one? Prove or disprove. Is f onto? Prove or disprove.
24. Let P be the power set of $\{a, b\}$ and let S be the set of all binary strings of length 2. A function $f: P \rightarrow S$ is defined as follows: For A in P , $f(A)$ has a 1 in the high-order bit position (left end of string) if and only if a is in A . $f(A)$ has a 1 in the low-order bit position (right end of string) if and only if b is in A . Is f one-to-one? Prove or disprove. Is f onto? Prove or disprove.
25. Let $S = \{x | x \in \mathbb{R} \text{ and } x \geq 1\}$, and $T = \{x | x \in \mathbb{R} \text{ and } 0 < x \leq 1\}$. Find a function $f: S \rightarrow T$ that is a bijection.

26. Let $S = \{a, b, c, d\}$ and $T = \{x, y, z\}$.
- Give an example of a function from S to T that is neither onto nor one-to-one.
 - Give an example of a function from S to T that is onto but not one-to-one.
 - Can you find a function from S to T that is one-to-one?
27. Compute the following values.
- $\lfloor 3.4 \rfloor$
 - $\lceil -0.2 \rceil$
 - $\lfloor 0.5 \rfloor$
28. Compute the following values.
- $\lceil -5 - 1.2 \rceil$
 - $\lceil -5 - \lfloor 1.2 \rfloor \rceil$
 - $\lfloor 2 * 3.7 \rfloor$
 - $\lceil 1 + 1/2 + 1/3 + 1/4 \rceil$
29. What can be said about x if $\lfloor x \rfloor = \lceil x \rceil$?
30. Prove that $\lceil x \rceil + 1 = \lceil x + 1 \rceil$.
31. Prove that $\lfloor x \rfloor = -\lceil -x \rceil$.
32. The ceiling function $f(x) = \lceil x \rceil: \mathbb{R} \rightarrow \mathbb{Z}$. Prove or disprove:
- f is one-to-one
 - f is onto
33. Prove or disprove:
- $\lceil \lfloor x \rfloor \rceil = x$
 - $\lfloor 2x \rfloor = 2\lfloor x \rfloor$
34. Prove or disprove:
- $\lfloor x \rfloor + \lfloor y \rfloor = \lfloor x + y \rfloor$
 - $\lfloor 2x \rfloor = \lfloor x \rfloor + \lfloor x + 1/2 \rfloor$
35. Prove that if $2^k < n < 2^{k+1}$ then $k = \lfloor \log n \rfloor$ and $k + 1 = \lceil \log n \rceil$. (Here $\log n$ means $\log_2 n$.)
36. Prove that if $2^k \leq n < 2^{k+1}$ then $\lfloor \log n \rfloor + 1 = \lceil \log(n + 1) \rceil$. (Here $\log n$ means $\log_2 n$.)
37. Compute the value of the following expressions.
- $31 \bmod 11$
 - $16 \bmod 8$
 - $22 \bmod 6$
 - $-7 \bmod 3$
38. a. List five values x such that $x \bmod 7 = 0$.
 b. List five values x such that $x \bmod 5 = 2$.
39. Prove or disprove: For any integers x and y , $x \bmod 10 + y \bmod 10 = (x + y) \bmod 10$.
40. Prove that $x \equiv y \pmod{n}$ if and only if $x \bmod n = y \bmod n$. (Recall the definition of congruence modulo n from Section 5.1.)
41. Let S be a set and let A be a subset of S . The *characteristic function* of A is a function $c_A: S \rightarrow \{0, 1\}$ with $c_A(x) = 1$ exactly when $x \in A$.
- Let $S = \{1, 2, 3, 4, 5\}$ and $A = \{1, 3, 5\}$. Give the ordered pairs that belong to c_A .
 - Prove that for any set S and any subsets A and B of S , $c_{A \cap B}(x) = c_A(x) \cdot c_B(x)$.

c. Prove that $c_A(x) = 1 - c_{A^c}(x)$.

d. Is it true that for any set S and any subsets A and B of S , $c_{A \cup B}(x) = c_A(x) + c_B(x)$? Prove or give a counterexample.

42. Ackermann's function¹, mapping \mathbb{N}^2 to \mathbb{N} , is a recursive function that grows very rapidly. It is given by

$$A(0, n) = n + 1 \text{ for all } n \in \mathbb{N}$$

$$A(m, 0) = A(m - 1, 1) \text{ for all } m \in \mathbb{N}, m > 0$$

$$A(m, n) = A(m - 1, A(m, n - 1)) \text{ for all } m \in \mathbb{N}, n \in \mathbb{N}, m > 0, n > 0$$

a. Compute (show all steps) the value of $A(1, 1)$.

b. Compute (show all steps) the value of $A(2, 1)$.

c. The value of $A(4, 0) = 13 = 2^{2^2} - 3$, still a small value. But $A(4, 1) = 2^{2^{2^2}} - 3$. Compute this value.

d. Write a likely expression for the value of $A(4, 2)$.

43. Another rapidly growing function is the Smorynski function, which also maps \mathbb{N}^2 to \mathbb{N} . The definition is

$$S(0, n) = n^n \text{ for all } n \in \mathbb{N}$$

$$S(m, n) = S(m - 1, S(m - 1, n)) \text{ for all } m \in \mathbb{N}, n \in \mathbb{N}, m > 0$$

a. How does $S(0, n)$ compare to $A(0, n)$? (See Exercise 40.)

b. Find (show all steps) an expression for the value of $S(1, n)$.

c. A googolplex is a very large number, which if written in standard form (such as 1,000,000 ...), even in 1-point font, would take more room to write than the diameter of the known universe. Look up the definition of the googolplex and write it as $S(m, n)$ for a specific value of m and n .

44. The Dwyer function also maps \mathbb{N}^2 to \mathbb{N} and grows very rapidly, but it has a closed-form definition:

$$D(m, n) = n! \left[\frac{(2m + 1)!}{2^m m!} \right]^n$$

a. Compute the values of $D(1, 1)$, $D(2, 1)$, $D(3, 1)$ and $D(4, 1)$.

b. Verify that $D(2, 1) = (2 \cdot 1 + 3)D(1, 1)$, that $D(3, 1) = (2 \cdot 2 + 3)D(2, 1)$, and that $D(4, 1) = (2 \cdot 3 + 3)D(3, 1)$.

c. Verify that $D(m, 1)$ satisfies the recurrence relation

$$D(m + 1, 1) = (2m + 3)D(m, 1) \text{ with } D(0, 1) = 1$$

(Hint: When you evaluate $D(m + 1, 1)$ and $D(m, 1)$ do not divide the denominator factorial into the numerator factorial. Instead, think of the numerator factorial as a product of even and odd factors.)

d. Find (use a spreadsheet) the smallest value of m for which

$$D(m, 1) < m^m$$

¹This is the most common of several versions of Ackermann's function, all of which are recursive with extremely rapid growth rates. To watch the tedious recursiveness of computations of Ackermann's function, go to <http://www.gfredericks.com/sandbox/arith/ackermann>

45. Let $S = \{1, 2, 3, 4\}$, $T = \{1, 2, 3, 4, 5, 6\}$, and $U = \{6, 7, 8, 9, 10\}$. Also, let $f = \{(1, 2), (2, 4), (3, 3), (4, 6)\}$ be a function from S to T , and let $g = \{(1, 7), (2, 6), (3, 9), (4, 7), (5, 8), (6, 9)\}$ be a function from T to U . Write the ordered pairs in the function $g \circ f$.
46. a. Let $f: \mathbb{R} \rightarrow \mathbb{Z}$ be defined by $f(x) = \lfloor x \rfloor$. Let $g: \mathbb{Z} \rightarrow \mathbb{N}$ be defined by $g(x) = x^2$. What is $(g \circ f)(-4.7)$?
 b. Let f map the set of books into the integers where f assigns to each book the number of words in its title. Let $g: \mathbb{Z} \rightarrow \mathbb{Z}$ be given by $g(x) = 2x$. What is $(g \circ f)$ (this book)?
 c. Let f map strings of alphabetical characters and blank spaces into strings of alphabetical consonants where f takes any string and removes all vowels and all blanks. Let g map strings of alphabetical consonants into integers where g maps a string into the number of characters it contains. What is $(g \circ f)$ (abraham lincoln)?
47. Let $f: \mathbb{N} \rightarrow \mathbb{N}$ be defined by $f(x) = x + 1$. Let $g: \mathbb{N} \rightarrow \mathbb{N}$ be defined by $g(x) = 3x$. Calculate the value of the following expressions.
 a. $(g \circ f)(5)$
 b. $(f \circ g)(5)$
 c. $(g \circ f)(x)$
 d. $(f \circ g)(x)$
 e. $(f \circ f)(x)$
 f. $(g \circ g)(x)$
48. The following functions map \mathbb{R} to \mathbb{R} . Give an equation describing the composition functions $g \circ f$ and $f \circ g$ in each case.
 a. $f(x) = 6x^3$, $g(x) = 2x$
 b. $f(x) = (x - 1)/2$, $g(x) = 4x^2$
 c. $f(x) = \lceil x \rceil$, $g(x) = \lfloor x \rfloor$
49. Let $f: S \rightarrow T$ and $g: T \rightarrow U$ be functions.
 a. Prove that if $g \circ f$ is one-to-one, so is f .
 b. Prove that if $g \circ f$ is onto, so is g .
 c. Find an example where $g \circ f$ is one-to-one but g is not one-to-one.
 d. Find an example where $g \circ f$ is onto but f is not onto.
50. a. Let f be a function, $f: S \rightarrow T$. If there exists a function $g: T \rightarrow S$ such that $g \circ f = i_S$, then g is called a *left inverse* of f . Show that f has a left inverse if and only if f is one-to-one.
 b. Let f be a function, $f: S \rightarrow T$. If there exists a function $g: T \rightarrow S$ such that $f \circ g = i_T$, then g is called a *right inverse* of f . Show that f has a right inverse if and only if f is onto.
 c. Let $f: \mathbb{N} \rightarrow \mathbb{N}$ be given by $f(x) = 3x$. Then f is one-to-one. Find two different left inverse functions for f .
 d. Let $f: \mathbb{N}^+ \rightarrow \mathbb{N}^+$ be given by $f(x) = \left\lceil \frac{x}{2} \right\rceil$. Then f is onto. Find two different right inverse functions for f .
51. For each of the following bijections $f: \mathbb{R} \rightarrow \mathbb{R}$, find f^{-1} .
 a. $f(x) = 2x$
 b. $f(x) = x^3$
 c. $f(x) = (x + 4)/3$
52. Let f and g be bijections, $f: S \rightarrow T$ and $g: T \rightarrow U$. Then f^{-1} and g^{-1} exist. Also, $g \circ f$ is a bijection from S to U . Show that $(g \circ f)^{-1} = f^{-1} \circ g^{-1}$.

53. Let $A = \{1, 2, 3, 4, 5\}$. Write each of the following permutations on A in cycle form.

a. $f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 1 & 5 & 4 & 2 \end{pmatrix}$

b. $f = \{(1, 4), (2, 5), (3, 2), (4, 3), (5, 1)\}$

54. Let $A = \{a, b, c, d\}$. Write each of the following permutations on A in array form.

a. $f = \{(a, c), (b, b), (c, d), (d, a)\}$

b. $f = (c, a, b, d)$

c. $f = (d, b, a)$

d. $f = (a, b) \circ (b, d) \circ (c, a)$

55. Let A be any set and let S_A be the set of all permutations of A . Let $f, g, h \in S_A$. Prove that the functions $h \circ (g \circ f)$ and $(h \circ g) \circ f$ are equal, thereby showing that we can write $h \circ g \circ f$ without parentheses to indicate grouping.

56. Find the composition of the following cycles representing permutations on $A = \{1, 2, 3, 4, 5\}$. Write your answer as a composition of one or more disjoint cycles.

a. $(2, 4, 5, 3) \circ (1, 3)$

b. $(3, 5, 2) \circ (2, 1, 3) \circ (4, 1)$ (By Exercise 55, we can omit parentheses indicating grouping.)

c. $(2, 4) \circ (1, 2, 5) \circ (2, 3, 1) \circ (5, 2)$

57. Find the composition of the following cycles representing permutations on $A = \{1, 2, 3, 4, 5, 6, 7, 8\}$. Write your answer as a composition of one or more disjoint cycles.

a. $(1, 3, 4) \circ (5, 1, 2)$

b. $(2, 7, 8) \circ (1, 2, 4, 6, 8)$

c. $(1, 3, 4) \circ (5, 6) \circ (2, 3, 5) \circ (6, 1)$

d. $(2, 7, 1, 3) \circ (2, 8, 7, 5) \circ (4, 2, 1, 8)$

58. Find the composition of the following cycles representing permutations on \mathbb{N} . Write your answer as a composition of one or more disjoint cycles.

a. $(3, 5, 2) \circ (6, 2, 4, 1) \circ (4, 8, 6, 2)$

b. $(1, 5, 13, 2, 6) \circ (3, 6, 4, 13) \circ (13, 2, 6, 1)$

c. $(1, 2) \circ (1, 3) \circ (1, 4) \circ (1, 5)$

59. Find the composition of the following cycles representing permutations on $A = \{a, b, c, d, e\}$. Write your answer as a composition of one or more disjoint cycles.

a. $(a, d, c, e) \circ (d, c, b) \circ (e, c, a, d) \circ (a, c, b, d)$

b. $(e, b, a) \circ (b, e, d) \circ (d, a)$

c. $(b, e, d) \circ (d, a) \circ (e, a, c) \circ (a, c, b, e)$

60. Find a permutation on an infinite set that cannot be written as a cycle.

61. The function f written in cycle form as $f = (4, 2, 8, 3)$ is a bijection on the set \mathbb{N} . Write f^{-1} in cycle form.

62. The “pushdown store,” or “stack,” is a storage structure that operates much like a set of plates stacked on a spring in a cafeteria. All storage locations are initially empty. An item of data is added to the top of the stack by a “push” instruction, which pushes any previously stored items farther down in the stack. Only the topmost item on the stack is accessible at any moment, and it is fetched and removed from the stack by a “pop” instruction.

Let's consider strings of integers that are an even number of characters in length; half the characters are positive integers, and the other half are zeros. We process these strings through a pushdown store as

follows: As we read from left to right, the push instruction is applied to any nonzero integer, and a zero causes the pop instruction to be applied to the stack, thus printing the popped integer. Thus, processing the string 12030040 results in an output of 2314, and processing 12304000 results in an output of 3421. (A string such as 10020340 cannot be handled by this procedure because we cannot pop two integers from a stack containing only one integer.) Both 2314 and 3421 can be thought of as permutations,

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 1 \end{pmatrix}$$

respectively, on the set $A = \{1, 2, 3, 4\}$.

- a. What permutation of $A = \{1, 2, 3, 4\}$ is generated by applying this procedure to the string 12003400?
 - b. Name a permutation of $A = \{1, 2, 3, 4\}$ that cannot be generated from any string where the digits 1, 2, 3, and 4 appear in order, no matter where the zeros are placed.
63. Let $S = \{2, 4, 6, 8\}$ and $T = \{1, 5, 7\}$.
- a. Find the number of functions from S to T .
 - b. Find the number of onto functions from S to T .
64. Let $S = \{P, Q, R\}$ and $T = \{k, l, m, n\}$.
- a. Find the number of functions from S to T .
 - b. Find the number of one-to-one functions from S to T .
65. a. For $|S| = 2, 3,$ and $4,$ respectively, use the theorem on the number of functions to show that the number of one-to-one functions from S to S equals the number of onto functions from S to S .
- b. Argue that for $|S| = n, f: S \rightarrow S$ is one-to-one if and only if f is onto.
- c. Find an infinite set S and a function $f: S \rightarrow S$ such that f is one-to-one but not onto.
- d. Find an infinite set S and a function $f: S \rightarrow S$ such that f is onto but not one-to-one.
66. Let $A = \{a, b, c, d\}$. How many functions are in S_A ? How many of these functions are derangements? Write all the derangements in array form.
67. Let $|S| = n$. In parts a-e, find the number of
- a. functions from S to S .
 - b. one-to-one functions from S to S .
 - c. functions from S onto S (see Exercise 65).
 - d. permutations from S onto S .
 - e. derangements from S onto S .
 - f. Order the values obtained in parts (a) through (e) from smallest to largest and explain why this ordering is reasonable.
68. a. A system development project calls for five different tasks to be assigned to Maria, Jon, and Suzanne. In how many ways can the assignment be done if each of the three workers must get at least one task?
- b. In how many ways can the projects be assigned if Maria must develop the test plan, which is one of the five tasks, but may do other tasks as well? (*Hint*: Consider the two cases where Maria does and does not do any of the other tasks.)
69. In a programming class of seven students, the instructor wants each student to modify the program from a previous assignment, but no student should work on his or her own program. In how many ways can the instructor assign programs to the students?

70. a. Find a calculus book and look up the Maclaurin series representation for the function e^x .
 b. Use the answer to part (a) to find a series representation for e^{-1} .
 c. Use a calculator to compute an approximate value for e^{-1} to about 5 decimal places.
 d. How can the answer to parts (b) and (c) help you approximate the number of derangements of n objects when n is large, say, $n \geq 10$? (*Hint*: Look at Equation (4) in this section.)
 e. Apply this approach to Exercise 69 and compare the results.
 f. Approximately how many derangements are there of 10 objects?
71. Let f be a function, $f: S \rightarrow T$.
 a. Define a binary relation ρ on S by $x \rho y \leftrightarrow f(x) = f(y)$. Prove that ρ is an equivalence relation.
 b. What can be said about the equivalence classes if f is a one-to-one function?
 c. For $S = T = \mathbb{Z}$ and $f(x) = 3x^2$, what is $[4]$ under the equivalence relation of part (a)?
72. Prove that $S(m, n)$, the number of ways to partition a set of m elements into n blocks, is equal to $1/n!$ times the number of onto functions from a set with m elements to a set with n elements. (*Hint*: Consider Exercise 71.)
73. By the definition of a function f from S to T , f is a subset of $S \times T$ where the image of every $s \in S$ under f is uniquely determined as the second component of the ordered pair (s, t) in f . Now consider any binary relation ρ from S to T . The relation ρ is a subset of $S \times T$ in which some elements of S may not appear at all as first components of an ordered pair and some may appear more than once. We can view ρ as a *nondeterministic function* from a subset of S to T . An $s \in S$ not appearing as the first component of an ordered pair represents an element outside the domain of ρ . For an $s \in S$ appearing once or more as a first component, ρ can select for the image of s any one of the corresponding second components.
 Let $S = \{1, 2, 3\}$, $T = \{a, b, c\}$, and $U = \{m, n, o, p\}$. Let ρ be a binary relation on $S \times T$ and σ be a binary relation on $T \times U$ defined by

$$\rho = \{(1, a), (1, b), (2, b), (2, c), (3, c)\}$$

$$\sigma = \{(a, m), (a, o), (a, p), (b, n), (b, p), (c, o)\}$$

Thinking of ρ and σ as nondeterministic functions from S to T and T to U , respectively, we can form the composition $\sigma \circ \rho$, a nondeterministic function from S to U .

- a. What is the set of possible images of 1 under $\sigma \circ \rho$?
 b. What is the set of possible images of 2 under $\sigma \circ \rho$? of 3?
74. Let f be a function, $f: S \rightarrow T$.
 a. Show that for all subsets A and B of S , $f(A \cap B) \subseteq f(A) \cap f(B)$.
 b. Show that $f(A \cap B) = f(A) \cap f(B)$ for all subsets A and B of S if and only if f is one-to-one.
75. Let \mathcal{C} be a collection of sets, and define a binary relation ρ on \mathcal{C} as follows: For $S, T \in \mathcal{C}$, $S \rho T \leftrightarrow S$ is equivalent to T . Show that ρ is an equivalence relation on \mathcal{C} .
76. Group the following sets into equivalence classes according to the equivalence relation of Exercise 75.

$$A = \{2, 4\}$$

$$B = \mathbb{N}$$

$$C = \{x \mid x \in \mathbb{N} \text{ and } (\exists y)(y \in \mathbb{N} \text{ and } x = 2*y)\}$$

$$D = \{a, b, c, d\}$$

$$E = \emptyset(\{1, 2\})$$

$$F = \mathbb{Q}^+$$

Exercises 77 and 78 involve programming with a functional language. Functional programming languages, as opposed to conventional (procedural) programming languages such as C++, Java, or Python, treat tasks in terms of mathematical functions. A mathematical function such as $f(x) = 2x$ transforms the argument 5 into the result 10. Think of a program as a big function to transform input into output. A functional programming language contains primitive functions as part of the language, and the programmer can define new functions as well. Functional programming languages support function composition, allowing for complex combinations of functions. Using the functional programming language Scheme, we can define the doubling function by

```
(define (double x)
  (* 2 x))
```

The user can then run the program and type

```
(double 5)
```

which produces an immediate output of 10.

77. a. Write a Scheme function to square a number.
b. What is the output from the following user input?

```
(double (square 3))
```

78. Scheme also supports recursion, plus the usual control structures of procedural languages, such as conditional and iterative statements.
a. Given the Scheme function

```
(define (mystery n)
  (cond ((= n 1) 1)
        (else (*n (mystery (- n 1))))))
```

what is the output of

```
(mystery 4)
```

- b. The “mystery” function is better known as _____.

SECTION 5.5 ORDER OF MAGNITUDE

Function Growth

Order of magnitude is a way of comparing the “rate of growth” of different functions. We know, for instance, that if we compute $f(x) = x$ and $g(x) = x^2$ for increasing values of x , the g values will be larger than the f values by an ever increasing amount. This difference in the rate of increase cannot be overcome by simply multiplying the f values by some large constant; no matter how large a constant we choose, the g values will eventually race ahead again. Our experience indicates that the f and g functions seem to behave in fundamentally different ways with respect to their rates of growth. In order to characterize this difference formally, we define a binary relation on functions.

Let S be the set of all functions with domain and codomain the nonnegative real numbers. We can define a binary relation on S by

$$f \rho g \leftrightarrow \text{there exist positive constants } n_0, c_1, \text{ and } c_2 \text{ such that, for all } x \geq n_0, \\ c_1 g(x) \leq f(x) \leq c_2 g(x)$$

EXAMPLE 44

Let f and g be functions in S where $f(x) = 3x^2$ and $g(x) = 200x^2 + 140x + 7$

Let $n_0 = 2$, $c_1 = \frac{1}{100}$, and $c_2 = 1$. Then for $x \geq 2$,

$$\frac{1}{100}(200x^2 + 140x + 7) \leq 3x^2 \leq (1)(200x^2 + 140x + 7)$$

or

$$2x^2 + 1.4x + 0.07 \leq 3x^2 \leq 200x^2 + 140x + 7 \quad (1)$$

Therefore $f \rho g$. ●

PRACTICE 40

- Verify the inequality in Equation (1) for the following values of x : 2, 3, 4, 5. (Use a calculator.)
- In Example 44, can n_0 have the value 1 if c_1 and c_2 remain the same?
- Find a different set of three values n_0 , c_1 , and c_2 that will also work to show that $f \rho g$ in Example 44. ■

The relation ρ is an equivalence relation on S . For example, to prove that $f \rho f$, we can pick $n_0 = c_1 = c_2 = 1$ and have

$$(1)f(x) \leq f(x) \leq (1)f(x)$$

PRACTICE 41

- Prove that ρ is symmetric.
- Prove that ρ is transitive. ■

Given that ρ is an equivalence relation, it partitions S into equivalence classes. If f is in the same class as g , then f is said to have the same order of magnitude as g , denoted by $f = \Theta(g)$ and pronounced “ f is order g ” or sometimes “ f is **big theta** of g .” Because of symmetry, this also means that g is the same order of magnitude as f , or $g = \Theta(f)$. (The notation $f = \Theta(g)$ is a bit of a misuse of the equality symbol because $\Theta(g)$ is not some function identical to f . It is just a shorthand way of saying that $f \in [g]$ under the equivalence relation ρ defined above.)

● **DEFINITION ORDER OF MAGNITUDE**

Let f and g be functions mapping nonnegative reals into nonnegative reals. Then f is the same **order of magnitude** as g , written $f = \Theta(g)$, if there exist positive constants n_0 , c_1 , and c_2 such that for $x \geq n_0$, $c_1g(x) \leq f(x) \leq c_2g(x)$.

We will usually try to find the simplest representative of a given equivalence class. Thus for the functions f and g of Example 44, we would say $f = \Theta(x^2)$ and $g = \Theta(x^2)$. A polynomial is always the order of magnitude of its highest-degree term; lower-order terms and all coefficients can be ignored. This is not surprising, since for large values of x , the highest-degree term will dominate the result.

PRACTICE 42

Prove (by finding appropriate constants that satisfy the definition of order of magnitude) that $f = \Theta(x^2)$ and $g = \Theta(x^2)$ for the functions f and g of Example 44.

To understand more intuitively what these equivalence classes mean, we'll draw some graphs. Let $h(x) \in S$, where $h(x) = x^2$. Figure 5.25 shows the graph of $h(x)$. Now suppose we multiply the h values by the two constants $c_1 = 1/2$ and $c_2 = 2$. The functions $c_1h(x)$ and $c_2h(x)$ are shown as dotted lines in Figure 5.26. These dotted lines form a kind of envelope around the $h(x)$ values, roughly tracing the shape of $h(x)$. Changing the value of the constants changes the width of the envelope but not the basic shape. If $h_1(x)$ is a function with $h_1 = \Theta(h)$, then there is some positive constant n_0 and some envelope around h such that for all domain values to the right of n_0 , the h_1 values must fall within this envelope, as shown in Figure 5.27. Therefore the h_1 values can never stray too far from the h values. The functions h_1 and h are roughly the same size—they are the same order of magnitude.

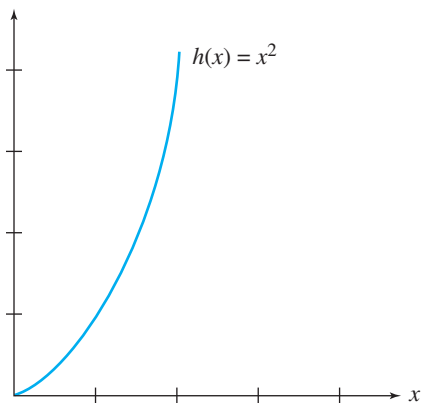


Figure 5.25

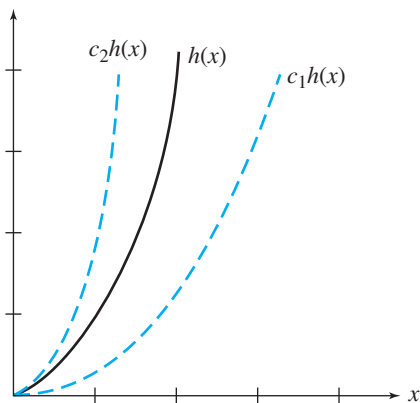


Figure 5.26

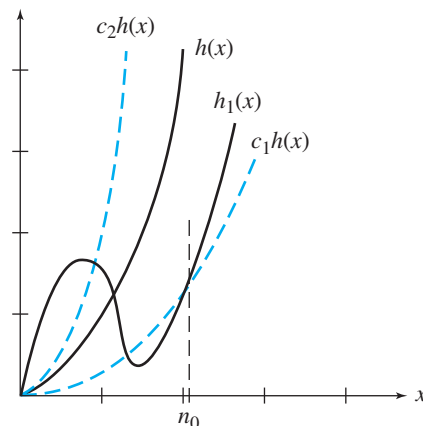


Figure 5.27

EXAMPLE 45

Let $f(x) = x$ and $h(x) = x^2$. Figure 5.28 illustrates that for the constants $c_1 = 1/2$ and $c_2 = 2$, f soon falls below the envelope. Reducing the c_1 constant (lowering the bottom edge of the envelope) only postpones the problem. Formally, we can do a proof by contradiction to show that f is not $\Theta(x^2)$. Suppose $f = \Theta(x^2)$. Then there exist constants n_0 and c_1 with $c_1x^2 \leq f(x)$ for $x \geq n_0$. But this would imply that $c_1x^2 \leq x$ or $c_1x \leq 1$ or $x \leq 1/c_1$ for all $x \geq n_0$. Because c_1 is fixed, we can always choose x large enough so that $x > 1/c_1$, which is a contradiction. Therefore $f(x) = x$ is not $\Theta(x^2)$.

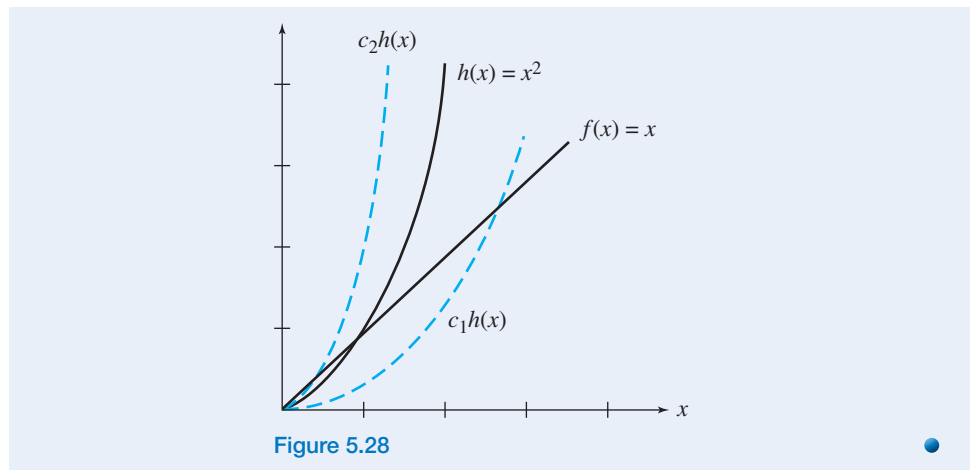


Figure 5.28

If we imagine functions representing various forms of transportation, then functions that are the same order of magnitude (belong to the same equivalence class) represent the same mode of transportation. One class represents travel on foot, another class represents travel by automobile, a third represents travel by air. Speeds within a given mode are about the same; ignoring coefficients and low-order terms amounts to ignoring the difference between walking and running or between a Jeep and a Jaguar or between a Cessna and a Boeing 787. Walking (at any speed) is distinctly different from driving, which is distinctly different from flying.

We can imagine a hierarchy of orders of magnitude. For example, the class $\Theta(x)$ is a lower order of magnitude than the class $\Theta(x)^2$ because functions that are $\Theta(x)$ eventually fall below functions that are $\Theta(x^2)$. Also, the class $\Theta(\log x)$ is a lower order of magnitude than $\Theta(x)$ (see Exercise 15 at the end of this section). In our transportation analogy, walking is slower than driving is slower than flying.

A sort of arithmetic can be developed using order of magnitude. For example, if $f_1(x) = x$ and $f_2(x) = x^2$, then the function $(f_1 + f_2)(x) = f_1(x) + f_2(x) = x + x^2 = \Theta(x^2)$. In general, if $f_1 = \Theta(g_1)$ and $f_2 = \Theta(g_2)$, then $f_1 + f_2 = \Theta(\max(g_1, g_2))$ (see Exercise 8). When expressed in abbreviated form, this leads to somewhat bizarre equations such as $\Theta(x) + \Theta(x^2) = \Theta(x^2)$ or $\Theta(x^2) + \Theta(x^2) = \Theta(x^2)$.

More on Analysis of Algorithms

Order of magnitude is important in analysis of algorithms, which we discussed in Section 3.3. In analyzing an algorithm, we identify the important tasks the algorithm performs. Usually the number of times such tasks must be done in executing the algorithm will depend on the size of the input. For example, searching a list of n elements or sorting a list of n elements will require more work as n increases. Typically, we can express input size as a nonnegative integer, so the functions that express the amount of work will be functions with domain \mathbb{N} . We found in Section 3.3 that a sequential search of n elements requires n comparisons in the worst case, while a binary search requires $1 + \log n$ comparisons in the worst case (assuming n is a power of 2). Rather than compute the exact functions for the amount of work

done, it is easier and often just as useful to settle for order-of-magnitude information. Sequential search is $\Theta(n)$ and binary search is $\Theta(\log n)$ in the worst case. Thus binary search is an order-of-magnitude improvement over sequential search. The order of magnitude of an algorithm is also known as its *computational complexity* because it reflects the amount of work inherent in the algorithm. Table 5.3 gives a summary of the worst-case order of magnitude for algorithms discussed in the text and exercises of Section 3.3.

Algorithm	Operation	Worst-Case Order of Magnitude
<i>SequentialSearch</i> on list size n	comparisons	$\Theta(n)$
<i>BinarySearch</i> on sorted list size n	comparisons	$\Theta(\log n)$
Pattern matching of pattern of length m in text of length n	comparisons	$\Theta(mn)$
Evaluation of a polynomial of degree n	multiplications and additions	$\Theta(n)$
<i>BubbleSort</i> of list size n	comparisons	$\Theta(n^2)$
<i>SelectionSort</i> of list size n	comparisons	$\Theta(n^2)$
<i>MergeSort</i> of list size n	comparisons	$\Theta(n \log n)$
<i>QuickSort</i> of list size n	comparisons	$\Theta(n^2)$

To appreciate the effect of order of magnitude in evaluating algorithms, suppose we have two algorithms A and A' to do the same job but they differ in order of magnitude—say, A is $\Theta(n)$ and A' is $\Theta(n^2)$. Even if each step in a computation takes only 0.0001 second, this difference will affect total computation time as n grows larger. The first two rows of Table 5.4 give total computation times for A and A' for various values of input length. Now suppose a third algorithm A'' exists whose order of magnitude is not even given by a polynomial function but by an exponential function, say 2^n . The total computation times for A'' are shown in the third row of Table 5.4.

Total Computation Time				
		Size of Input n		
Algorithm	Order	10	50	100
A	n	0.001 second	0.005 second	0.01 second
A'	n^2	0.01 second	0.25 second	1 second
A''	2^n	0.1024 second	3570 years	4×10^{16} centuries

Note that the exponential case grows at a fantastic rate! Even if we assume that each computation step takes much less time than 0.0001 second, the relative growth rates between polynomial and exponential functions still follow this same pattern. Because of this immense growth rate, algorithms not of polynomial order are generally not useful for large values of n . In fact, problems for which no polynomial time algorithms exist are called **intractable**.

Sometimes algorithms that are not polynomial in the worst case may still be efficient for “average”—and useful—input cases.² Nonetheless, in attempting to improve efficiency, we should ask whether a different algorithm of a lower order of magnitude exists before we worry about the details of fine-tuning a given algorithm.

If $f(n)$ represents the work done by an algorithm on an input of size n , it may be difficult to find a simple function g such that $f = \Theta(g)$. Remember that if we can find such a g , then f and g are functions that eventually (for large enough n) have roughly the same shape. But we may still be able to find a function g that serves as an upper bound for f . In other words, while f may not have the same shape as g , f will never grow significantly faster than g . Formally, this is expressed by saying that $f = O(g)$ (“ f is big O of g ”).

● **DEFINITION BIG OH**

Let f and g be functions mapping nonnegative reals into nonnegative reals. Then f is **big oh** of g , written $f = O(g)$, if there exist positive constants n_0 and c such that for $x \geq n_0$, $f(x) \leq cg(x)$.

If $f = O(g)$, then g is a ceiling for f and gives us a worst-case picture of the growth of f . In Section 3.3, we learned that if $E(n)$ is the number of divisions required by the Euclidean algorithm to find $\gcd(a, b)$, where $b < a = n$, then $E(n) = O(\log n)$.

The big oh notation $f = O(g)$ says that f grows at the same rate or at a slower rate than g . But if we know that f definitely grows at a slower rate than g , then we can say something stronger, namely that f is **little oh** of g , written $f = o(g)$. The relationship between big oh and little oh is this: If $f = O(g)$, then either $f = \Theta(g)$ or $f = o(g)$, much like $a \leq b$ says that either $a = b$ or $a < b$.

The Master Theorem

In Section 3.2, we learned that a solution for divide-and-conquer recurrence relations of the form

$$S(n) = cS\left(\frac{n}{2}\right) + g(n) \text{ for } n \geq 2, n = 2^m$$

is given by

$$S(n) = c^{\log n} S(1) + \sum_{i=1}^{\log n} c^{(\log n) - i} g(2^i)$$

²This is the case with the well-known *simplex method* for solving linear programming problems, which are a generalization of systems of linear equations to systems of inequalities—see Section 5.7.

This type of recurrence relation arises in the analysis of an algorithm that splits the input in half and operates recursively on one or more of the halves. A more general divide-and-conquer recurrence relation divides the input into subproblems that are each of size n/b and then operates recursively on a of the subproblems. Each subproblem requires $S(n/b)$ work with $g(n)$ representing the work required to divide into subproblems or recombine the results of solving the subproblems. Such a recurrence relation would have the form

$$S(n) = aS\left(\frac{n}{b}\right) + g(n) \text{ for } n \geq 2, n = b^m \quad (2)$$

We assume that n is an integral power of b so that dividing n by b over and over always results in an integer.

A result called the *master theorem* gives us order-of-magnitude results via a cookbook formula for the case when $g(n) = n^c$. Thus, assume the recurrence relations of interest look like this:³

$$S(n) = aS\left(\frac{n}{b}\right) + n^c \text{ for } n \geq 2, n = b^m \quad (3)$$

THEOREM MASTER THEOREM

Consider the recurrence relation

$$S(1) \geq 0$$

$$S(n) = aS\left(\frac{n}{b}\right) + n^c \text{ for } n \geq 2$$

where $n = b^m$, a , and b are integers, $a \geq 1$, $b > 1$, and c is a nonnegative real number. Then

- | | |
|-----------------|-------------------------------|
| 1. if $a < b^c$ | $S(n) = \Theta(n^c)$ |
| 2. if $a = b^c$ | $S(n) = \Theta(n^c \log n)$ |
| 3. if $a > b^c$ | $S(n) = \Theta(n^{\log_b a})$ |

EXAMPLE 46

A recurrence relation of the form

$$S(n) = 4S\left(\frac{n}{5}\right) + n^3 \text{ for } n \geq 2, n = 5^m$$

matches the form of the recurrence relation shown in the master theorem where $a = 4$, $b = 5$, and $c = 3$. Because $4 < 5^3$, Case 1 applies and the master theorem says that $S(n) = \Theta(n^3)$. ●

³A more complicated master theorem gives similar results for the case of Equation (2).

Note that the master theorem does not give exact solutions; it gives order-of-magnitude results. Also, the value of $S(1)$ (a constant) plays no part in determining the result.

EXAMPLE 47

The worst-case recurrence relation for comparisons done by the *BinarySearch* algorithm on a sorted list of size n (Example 30 in Chapter 3) is

$$C(1) = 1$$

$$C(n) = C\left(\frac{n}{2}\right) + 1 \text{ for } n \geq 2, n = 2^m$$

We found the exact solution to be $C(n) = 1 + \log n$, certainly $\Theta(\log n)$. We can find this directly from the master theorem with $a = 1$, $b = 2$, $c = 0$. Because $a = b^c$, $C(n) = \Theta(n^0 \log n) = \Theta(\log n)$. ●

Proof of the Master Theorem

To prove the master theorem, we'll go back to an expansion technique. We apply the "recipe" of Equation (3) over and over. The recipe is that S at some value is a times S at that value divided by b , plus that value to the c power. Therefore

$$\begin{aligned} S(n) &= aS\left(\frac{n}{b}\right) + n^c \\ &= a\left[aS\left(\frac{n}{b^2}\right) + \left(\frac{n}{b}\right)^c \right] + n^c = a^2S\left(\frac{n}{b^2}\right) + a\left(\frac{n}{b}\right)^c + n^c \\ &= a^2\left[aS\left(\frac{n}{b^3}\right) + \left(\frac{n}{b^2}\right)^c \right] + a\left(\frac{n}{b}\right)^c + n^c = a^3S\left(\frac{n}{b^3}\right) + a^2\left(\frac{n}{b^2}\right)^c + a\left(\frac{n}{b}\right)^c + n^c \\ &\vdots \\ &= a^kS\left(\frac{n}{b^k}\right) + a^{k-1}\left(\frac{n}{b^{k-1}}\right)^c + \cdots + a\left(\frac{n}{b}\right)^c + n^c \end{aligned}$$

This expansion must stop when $n/b^k = 1$, or $n = b^k$, which means $k = m$. At that point, using summation notation,

$$\begin{aligned} S(n) &= a^m S(1) + \sum_{i=0}^{m-1} a^i \left(\frac{n}{b^i}\right)^c \\ &= a^m S(1) + \sum_{i=0}^{m-1} n^c \left(\frac{a^i}{b^{ci}}\right) \\ &= a^m S(1) + n^c \sum_{i=0}^{m-1} \left(\frac{a}{b^c}\right)^i \end{aligned} \tag{4}$$

Because $n = b^m$, it follows that $m = \log_b n$, so Equation (4) can be written as

$$S(n) = a^{\log_b n} S(1) + n^c \sum_{i=0}^{m-1} \left(\frac{a}{b^c}\right)^i$$

and this equation in turn, using property 11 of the logarithm function (see Appendix C), can be written as

$$S(n) = n^{\log_b a} S(1) + n^c \sum_{i=0}^{m-1} \left(\frac{a}{b^c}\right)^i \quad (5)$$

We'll see the expression $\log_b a$ frequently, so we'll temporarily give it a simpler name: Let $w = \log_b a$. Then, expanding the summation, Equation (5) can be written as

$$S(n) = n^w S(1) + n^c \left[1 + \left(\frac{a}{b^c}\right) + \left(\frac{a}{b^c}\right)^2 + \cdots + \left(\frac{a}{b^c}\right)^{m-1} \right]$$

We can see that the expression in brackets represents the sum of the first m terms of a geometric sequence with first term equal to 1 and common ratio r equal to (a/b^c) . If $r \neq 1$, this sum (see Exercise 27 in Section 2.2) has the value

$$\frac{1 - \left(\frac{a}{b^c}\right)^m}{1 - \left(\frac{a}{b^c}\right)}$$

and

$$S(n) = n^w S(1) + n^c \left[\frac{1 - \left(\frac{a}{b^c}\right)^m}{1 - \left(\frac{a}{b^c}\right)} \right] \quad (6)$$

The condition $r \neq 1$ means $(a/b^c) \neq 1$ or $a \neq b^c$. This means that Equation (6) holds for both Case 1 and Case 3 of the master theorem, so to prove those cases we'll do some tiresome algebra on Equation (6).

First, note that

$$n^c \left(\frac{a}{b^c}\right)^m = n^c \left(\frac{a}{b^c}\right)^{\log_b n} = \frac{n^c a^{\log_b n}}{b^{c \log_b n}} = \frac{n^c n^{\log_b a}}{b^{\log_b n^c}} = \frac{n^c n^{\log_b a}}{n^c} = n^{\log_b a} = n^w$$

so Equation (6) becomes

$$S(n) = n^w S(1) + \frac{n^c - n^c \left(\frac{a}{b^c}\right)^m}{\frac{b^c - a}{b^c}} = n^w S(1) + \frac{b^c (n^c - n^w)}{b^c - a}$$

or

$$S(n) = n^w S(1) + \frac{b^c}{b^c - a} n^c + \frac{b^c}{a - b^c} n^w \quad (7)$$

Case 1: $a < b^c$. From $a < b^c$ we get (taking \log_b of both sides)

$$\log_b a < c \text{ or } w < c$$

In the right side of Equation (7), n^c is the highest-power term and has a positive coefficient (because $b^c - a > 0$). Therefore $S(n) = \Theta(n^c)$.

Case 3: $a > b^c$. From $a > b^c$ we get (taking \log_b of both sides)

$$\log_b a > c \text{ or } w > c$$

In the right side of Equation (7), n^w is the highest-power term and has a positive coefficient (because $a - b^c > 0$). Therefore $S(n) = \Theta(n^w) = \Theta(n^{\log_b a})$.

Case 2: $a = b^c$. This is an easy case to prove and is left for you to do (see Exercise 28).

SECTION 5.5 REVIEW

TECHNIQUES

- Determine whether two functions are the same order of magnitude.
- W Use the master theorem to find an order-of-magnitude expression for the solution to certain divide-and-conquer recurrence relations.

MAIN IDEAS

- Functions can be grouped into equivalence classes according to their order of magnitude, which is a measure of their growth rate.
- Big theta, big oh and little oh (Θ , O , o) are notations for relating the growth rates of two functions.

EXERCISES 5.5

1. Prove, by finding constants that satisfy the definition of order of magnitude, that $f = \Theta(g)$ if $f(x) = x$ and $g(x) = 17x + 1$.
2. Prove, by finding constants that satisfy the definition of order of magnitude, that $f = \Theta(g)$ if $f(x) = 3x^3 - 7x$ and $g(x) = x^3/2$.
3. Prove, by finding constants that satisfy the definition of order of magnitude, that $f = \Theta(g)$ if $f(x) = 29x^2 - 4x - 15$ and $g(x) = 15x^2 + x$.
4. Prove, by finding constants that satisfy the definition of order of magnitude, that $f = \Theta(g)$ if $f(x) = \sqrt{x + 100}$ and $g(x) = \sqrt{x}$.
5. Prove, by finding constants that satisfy the definition of order of magnitude, that $f = \Theta(g)$ if $f(x) = x^3 + \log x$ and $g(x) = x^3$.
6. Prove, by finding constants that satisfy the definition of order of magnitude, that $f = \Theta(g)$ if $f(x) = \log(3x^2)$ and $g(x) = \log x$.
7. In this section, we noted that $h_1 = \Theta(h)$ implies that from some point on, h_1 is within an “envelope” of h . Can this envelope ever be entirely above or entirely below h ? Explain.
8. Prove that if f_1 is a function that is $\Theta(g_1)$ and f_2 is a function that is $\Theta(g_2)$, then the function $f_1 + f_2$, defined by $(f_1 + f_2)(x) = f_1(x) + f_2(x)$, is $\Theta(\max(g_1, g_2))$, where $(\max(g_1, g_2))(x) = \max(g_1(x), g_2(x))$.
9. Find the smallest integer n for which $x \log x$ is $O(x^n)$.
10. Find the smallest integer n for which $(x^4 + 4x)/(x + 2)$ is $O(x^n)$.

Exercises 11–18 require familiarity with ideas from calculus. As an alternative to the definition of order of magnitude, a limit test can be used to prove that $f = \Theta(g)$:

$$f = \Theta(g) \text{ if } \lim_{x \rightarrow \infty} \frac{f(x)}{g(x)} = p \quad \text{where } p \text{ is a positive real number}$$

As an aid in finding the limit of a quotient, if $\lim_{x \rightarrow \infty} f(x) = \infty$ and $\lim_{x \rightarrow \infty} g(x) = \infty$ and f and g are differentiable functions, then L'Hôpital's rule says that

$$\lim_{x \rightarrow \infty} \frac{f(x)}{g(x)} = \lim_{x \rightarrow \infty} \frac{f'(x)}{g'(x)}$$

Thus $2x^2 + 7 = \Theta(x^2)$ because

$$\lim_{x \rightarrow \infty} \frac{2x^2 + 7}{x^2} = \lim_{x \rightarrow \infty} \frac{4x}{2x} = 2$$

11. Use the limit test to do Exercise 1 again.

12. Use the limit test to do Exercise 2 again.

As another limit test, if

$$\lim_{x \rightarrow \infty} \frac{f(x)}{g(x)} = 0$$

then $f = o(g)$.

13. Use the second limit test to prove that $x = o(x^2)$.

14. Use the second limit test to prove that $\sqrt{x} = o(x)$.

15. Use the second limit test to prove that $\log x = o(x)$.

16. Use the second limit test to prove that $(\ln x)^2 = o(x^{0.5})$ where $\ln x$ is the natural log of x , $\log_e x$.

17. Use both limit tests to group the following functions into classes by order of magnitude and to order those classes. Here $\ln x$ is the natural log of x , $\log_e x$.

$$17x \log x, 200 \log x, 2^x - x^2, \sqrt[4]{x}, 10x^2 - 3x + 5, 420x, 41 \ln x^2$$

18. Use both limit tests to group the following functions into classes by order of magnitude and to order those classes. Here $\ln x$ is the natural log of x , $\log_e x$.

$$x, \sqrt{x}, \log x, x^3, x \log x, 2x^3 + x, e^x, (\log x)^2, \ln x, x^3 + \log x$$

19. You ask three different people to give you a worst-case order-of-magnitude expression for the work done by a particular algorithm on an input of size n . You receive three answers:

i. $O(n^3)$

ii. $o(n^3)$

iii. $\Theta(n^2)$

Which is the most useful and why?

20. An algorithm to determine whether a propositional wff with n statement letters is a tautology works by assigning, one at a time, all possible sets of truth values to the statement letters. The unit of work for this algorithm is the examination of one set of truth values. Explain why this algorithm is $\Theta(2^n)$ in the worst case.

For Exercises 21–26, use the master theorem to determine an order-of-magnitude expression for the work done.

$$21. S(n) = 2S\left(\frac{n}{4}\right) + n^2$$

$$22. S(n) = 4S\left(\frac{n}{3}\right) + n$$

$$23. S(n) = 4S\left(\frac{n}{4}\right) + n$$

$$24. S(n) = 4S\left(\frac{n}{2}\right) + n^2$$

$$25. S(n) = 3S\left(\frac{n}{3}\right) + \sqrt{n}$$

$$26. S(n) = 2S\left(\frac{n}{2}\right) + n^3$$

27. The worst-case recurrence relation for comparisons done by the *MergeSort* sorting algorithm is

$$C(1) = 0$$

$$C(n) = 2C\left(\frac{n}{2}\right) + (n - 1) \text{ for } n \geq 2, n = 2^m$$

- a. Use the master theorem to find an order-of-magnitude expression for the solution to the related recurrence relation.

$$C'(n) = 2C'\left(\frac{n}{2}\right) + n \text{ for } n \geq 2, n = 2^m$$

- b. Compare the result of part (a) with the exact solution to $C(n)$ (Exercise 22 of Section 3.3).

28. Prove the master theorem for the case where $a = b^c$. (*Hint*: Start with Equation (5). Also remember that a change of base of logarithms only involves multiplication by a constant.)

SECTION 5.6 THE MIGHTY MOD FUNCTION

In Section 5.4 we defined the modulo n function as follows:

If $x = qn + r, 0 \leq r < n$, then $x \bmod n = r$.

In other words, $x \bmod n$ is the nonnegative remainder (also called the **residue of x modulo n**) when x is divided by the positive integer n . This seemingly innocuous function turns up in a surprising number of applications, some of which will be explored in this section.

EXAMPLE 48

Military time uses a 24-hour clock, where the hours run from 0 through 23. One half hour before midnight in military time is 23:30. Standard time uses AM and PM for the first half of the day and the second half of the day, respectively, with the hours in each half running from 0 through 11. One half hour before midnight in standard time is 11:30 PM. Conversion from military time to standard time uses the modulo 12 function:

8:00 military time: Compute $8 \bmod 12 = 8$ gives 8:00 AM standard time
 16:00 military time: Compute $16 \bmod 12 = 4$ gives 4:00 PM standard time

The AM or PM designation is determined by whether the quotient in the division is 0 (AM) or 1 (PM). Counting modulo 12 begins at 0 and wraps around to 0 again when 12 is reached:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 1, 2, 3, ...

This counting scheme is sometimes taught to young children as “clock arithmetic.” ●

Section 5.1 defined an equivalence relation called congruence modulo n :

$$x \equiv y \pmod{n} \text{ if } x - y \text{ is an integral multiple of } n.$$

Given the similar terminology, one would expect that congruence modulo n and the modulo n function would surely be related.

PRACTICE 43

- a. Prove that $x \equiv y \pmod{n}$ if and only if $(x - y) \bmod n = 0$.
- b. Prove that $x \equiv y \pmod{n}$ if and only if $x \bmod n = y \bmod n$. ■

Exercises at the end of this section show how some arithmetic operations using mod n can be broken down for simpler calculation. One such result (Exercise 7) is particularly useful:

$$(x \cdot y) \bmod n = (x \bmod n \cdot y \bmod n) \bmod n \quad (1)$$

By Equation (1),

$$220 \bmod 6 = (22 \bmod 6 \cdot 10 \bmod 6) \bmod 6 = (4 \cdot 4) \bmod 6 = 16 \bmod 6 = 4$$

In addition, spreadsheet software usually contains a mod function that speeds up computations.

Hashing

A **hash function** is a function $h: S \rightarrow T$ where the domain S is a set of text strings or integer values, and the codomain T is the set of integers $\{0, 1, \dots, t - 1\}$

where t is some relatively small positive integer. If the domain S consists of text strings, we can imagine them encoded in some way into integer values, perhaps by an algorithm as simple as converting each individual letter of a text string into its position in the alphabet ($a = 1$, $b = 2$, and so on) and adding up the resulting list of integers to get one integer value. Therefore we can assume that S consists of integer values to begin with.

The function h therefore maps a potentially large set of integer values S into a relatively small window of integer values T . Consequently h isn't likely to be a one-to-one function because there may be many values x_1, x_2, x_3, \dots from S such that $h(x_1) = h(x_2) = h(x_3)$, in which case we say that x_1, x_2 , and x_3 all “hashed” to the same value. The term “hash function” came about because the domain value x is often chopped up into pieces in the process of computing $h(x)$. Whatever the function $h(x)$ does to x , the last step is almost always to apply the mod t function so the final value of $h(x)$ falls in the codomain $\{0, 1, \dots, t - 1\}$.

EXAMPLE 49

S is a set of positive integers, T is the set of integers $\{0, \dots, t - 1\}$ and the hash function $h(x)$ is given by

$$h(x) = x \bmod t$$

If $t = 10$, for example, then values are computed modulo 10:

$$h(7) = 7 \bmod 10 = 7$$

$$h(23) = 23 \bmod 10 = 3$$

$$h(59) = 59 \bmod 10 = 9$$

$$h(158) = 158 \bmod 10 = 8$$

$$h(48) = 48 \bmod 10 = 8$$

Here 158 and 48 hash to the same value. 

A hash function is often used as part of a search algorithm. We have already discussed two search algorithms. In sequential search, n elements are stored in an unordered (random) list and a given target value is compared one-by-one with the list elements. In binary search, n elements are stored in a sorted list. The given target value is compared with the midpoint of the list, the midpoint of half the list, and so on. In a search using a hash function, n elements are stored in an array (a one-dimensional table) called a **hash table**, where the array is indexed from 0 through $t - 1$; the table is of size t . The element x is passed as the argument to the hash function and the resulting $h(x)$ value gives the array index at which the element is then stored. Later, when a search is carried out, the target value is run through the same hash function, giving an index location in the hash table, which is where to look for the matching stored element.

However, because the hash function is not one-to-one, things are not quite that simple. As the search list is loaded into the hash table, different values may hash to the same array index, producing a **collision**. There are several collision resolution algorithms available. One is called **linear probing**—just keep going in

the array (looping back to the top of the array when you get to the bottom) and store element x in the next available empty slot. Another method, called **chaining**, builds a linked list for each array index; the list is initially empty but eventually contains all elements that hashed to that index value.

EXAMPLE 50

Using the hash function of Example 49, the elements 7, 23, 59, 158 and 48 are loaded in order into the hash table. The size of the hash table is the same as the modulo value used (10). With linear probing, the resulting hash table looks like Figure 5.29(a); 48 hashes to index 8 (occupied), tries index 9 (occupied), and finally finds a slot at index 0. With chaining, assuming elements are added to the front of the list, the hash table looks like Figure 5.29(b).

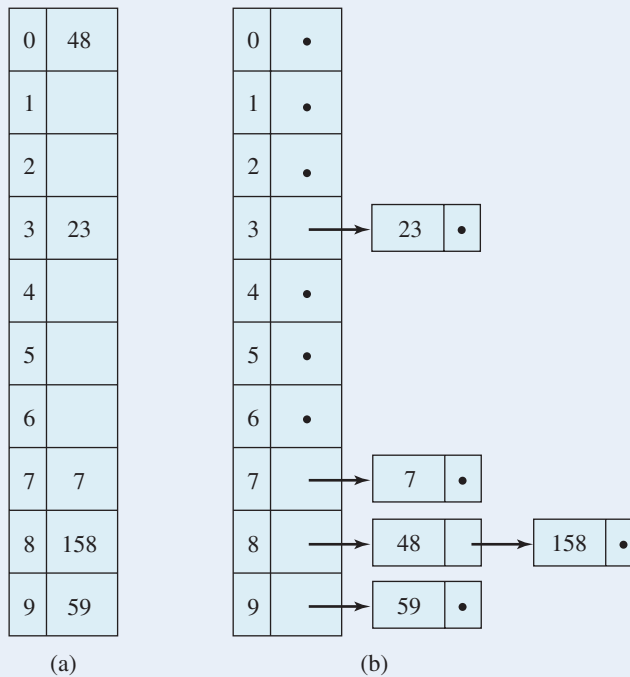


Figure 5.29

After the hash table is built, we can perform searches for target elements. To search for a target element of 48, the linear collision resolution scheme would repeat the steps used to store the 48—hash the target 48 to 8, then check index 9 and then index 0, where the element is found. The target would be compared with three different values (158, 59, 48) to reach a successful conclusion. To search for a target of 68, the target would hash to index 8 (no match); then the elements at index 9 and index 0 would be checked, also with no success. Advancing to index 1, the hash table contains an empty slot, so the search concludes (after four comparisons, counting comparing the target against an empty cell) that 68 is not stored in the hash table. The chaining scheme would hash a target value of 48 to index 8 and then search (sequentially) the resulting linked list, finding the target value at the beginning of the list after one comparison. To search for a target value of 68, the search algorithm would search the same linked list but would get to the end, concluding that 68 is not stored in the hash table. ●

PRACTICE 44 Show the resulting hash tables if 28 is the next value stored in the hash tables of Figure 5.29(a) and (b).

EXAMPLE 51

Voting records for the citizens of a certain precinct are stored in a hash table of size 197 using the voter's Social Security number as a key value (the value that is hashed). Using a hash function modulo the hash table size, the index to search first for data on a voter with Social Security number

$$328356770$$

is

$$328356770 \bmod 197 = 125$$

Two goals should be kept in mind in designing a hash function:

1. Given an argument value x , $h(x)$ can be computed quickly.
2. The number of collisions will be reduced because $h(x)$ does a good job of distributing values throughout the hash table.

Use of a modulo function as the hash function accomplishes the first goal. The second goal is harder to achieve because data can always be found that produce a large number of collisions, but distribution seems to work better on the average if the table size (the modulo value) is a prime number.

The average number of comparisons required to search for an element using hashing depends not on the total number n of elements in the hash table, but rather on the ratio of n to the total table size t . If this ratio is low, then (using linear probing) there are lots of empty slots, so you won't have to look very far to find a place to insert a new element into the table or, correspondingly, to search for a target element that is in the hash table. Similarly, if this ratio is low and chaining is used, the average length of any linked list you may have to (sequentially) search for a target element should be short. This ratio n/t is called the **load factor** of the hash table. In sequential search or binary search, more elements in the set to be searched increases the work (number of comparisons) required. But using hashing, more elements can be searched just as efficiently as fewer provided the hash table size grows accordingly so that the load factor stays low.

Computer Security

Computer security (or the more general term information assurance) is a topic of critical interest when our economy, defense, and indeed our entire way of life depend so much on computers and information. The mod function plays a part in many aspects of security.

Cryptography

Children often delight in sending "secret messages" using some encoding/decoding scheme that they know and that (they believe) their parents don't know!

In the adult world, military information, financial information, and company proprietary information that must be transmitted securely uses the same process. The original information (called the **plaintext**) is encrypted using an encryption key, resulting in coded text called the **ciphertext**. The ciphertext is transmitted and when it is received, it can be decoded using the corresponding decryption key. Encryption and decryption are inverse functions in the sense that

$$\text{decryption}(\text{encryption}(\text{plaintext})) = \text{plaintext}$$

If the message is intercepted by someone with no knowledge of the decryption key, it will not be useful unless the encryption scheme can be broken. **Cryptography** is the study of various encryption/decryption schemes. The broader term **cryptology** includes not only cryptography but also the techniques used to analyze and “break” coded messages. One of the most famous examples of breaking coded messages occurred during World War II when a British team that included mathematician Alan Turing, working at Bletchley Park, was able to break the supposedly invincible German “Enigma” code and decipher the plans of German submarine movement.

Military use of cryptographic techniques can be traced back to Julius Caesar who sent messages to his generals in the field using a scheme now known as the **Caesar cipher**. Let us assume that plaintext messages use only the 26 capital letters of the alphabet, that spaces between words are suppressed, and that each letter is first mapped to its corresponding position in the alphabet:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

We’ll denote this mapping as the bijection $g: \{A, \dots, Z\} \rightarrow \{0, \dots, 25\}$. Then a positive integer key value k is chosen that shifts each number k positions to the right with a “wrap-around” back to the beginning if needed (this is the mod function). Finally, the function g^{-1} is applied to translate the resulting number back into a letter. This produces the final ciphertext character c that corresponds to an original plaintext character p . Informally, we could compute c from p by just shifting over, in a circular fashion, k positions in the list of letters, but the conversion to numbers allows us to write a mathematical function to carry out the encoding and decoding algorithmically. The encoding function is given by

$$f(p) = g^{-1}([g(p) + k] \bmod 26)$$

The decoding process is to find the number for c , shift left k positions with a “wrap-around” back to the end if needed, then change the resulting digit back into a character. The decoding function is

$$f^{-1}(c) = g^{-1}([g(c) - k] \bmod 26)$$

EXAMPLE 52

In a Caesar cipher with $k = 3$,

$$E \text{ encodes to } g^{-1}([4 + 3] \bmod 26) = g^{-1}(7 \bmod 26) = g^{-1}(7) = H$$

$$Y \text{ encodes to } g^{-1}([24 + 3] \bmod 26) = g^{-1}(27 \bmod 26) = g^{-1}(1) = B$$

And for $k = 3$,

$$H \text{ decodes to } g^{-1}([7 - 3] \bmod 26) = g^{-1}(4 \bmod 26) = g^{-1}(4) = E$$

$$B \text{ decodes to } g^{-1}([1 - 3] \bmod 26) = g^{-1}(-2 \bmod 26) = g^{-1}(24) = Y \quad \bullet$$

Notice that if you intercept a message encoded using a Caesar cipher, you need to try only 26 possible key values to break the code (actually only 25 since $k = 26$ shifts each number back to itself). The Caesar cipher is not a very secure code; it may have served well, however, in an era when not many people could read at all. For additional security, the function g could be a less obvious bijection; the number of possible bijections g is $26!$ This would mean a total of $26! \cdot 25$ possibilities to try for g and k , too large a number even if you had your handy laptop with you in Gaul. As a cryptanalyst, you would narrow down the possibilities by making use of statistical characteristics of the language, such as individual letter frequency, frequency of certain letter pairs, and so on.

PRACTICE 45

Decode the following ciphertext that was encoded with a Caesar cipher using a key of 7: AOLJHAPUAOLOHA.

The Caesar cipher is a **simple substitution cipher**, meaning that each plaintext character is coded consistently into the same single ciphertext character. Encryption techniques where a single plaintext character contributes to several ciphertext characters (and one ciphertext character is the result of several plaintext characters) introduce **diffusion**. The advantage to diffusion is that it hides the frequency statistics of individual letters, making analysis of an intercepted ciphertext message much more difficult.

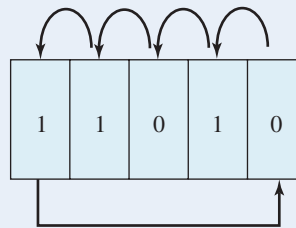
DES (Data Encryption Standard) is an internationally standard encryption algorithm developed in 1976. DES was developed to safeguard the security of digital information, so we may consider the plaintext to be a string of bits (a string of 0s and 1s). Unlike the Caesar cipher that encodes each single plaintext character individually, DES is a **block cipher**. A block of 64 plaintext bits is encoded as a unit using a 56-bit key. This results in a block of 64 ciphertext bits. However, changing one bit in the plaintext or one bit in the key changes about half of the resulting 64 ciphertext bits, so DES exhibits high diffusion. One might expect this effect to require some extremely complex mathematical encoding function, but DES actually uses many simple operations over and over. The DES algorithm calls for 16 “rounds” to be done; the original 56-bit key is modified from one round to the next, as is the original 64-bit plaintext block. The modifications involve the following, among other things:

- Running bit strings through permutation functions $f(i) = j$ such that the new bit value at position i in the string is the old bit value at position j
- Combining bit strings from the plaintext and the key using a bitwise exclusive-OR operation \oplus (two 0s or two 1s result in a 0 bit, while a single 0 and a single 1 result in a 1 bit)
- Changing the key by splitting the bit string in half and performing a **circular left shift** on each half of 1 bit or 2 bits, depending on the round number

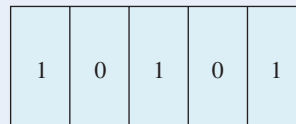
It's easy to look at a bit string and see informally what the result of a circular left shift will be. But, as in the case of the Caesar cipher, we'd like to put the shift on a mathematical (algorithmic) footing, and the mod function comes into play.

EXAMPLE 53

Consider a 5-bit binary string x , such as 11010. A 1-bit circular left shift of x would involve moving bits as follows—



—resulting in



Thinking of 11010 as a binary number, each column represents a power of 2, as opposed to the powers of 10 that our decimal numbers represent. The decimal equivalent of $x = 11010$ is therefore

$$1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 16 + 8 + 2 = 26$$

and the decimal equivalent of $y = 10101$ is $16 + 4 + 1 = 21$.

To mathematically compute y from x , we can carry out the following steps. While all steps are done on binary strings, we will also show the decimal equivalents. Multiplication of a binary string by 2 moves the digits one column to the left, padding the rightmost column with 0; multiplication by 2^{-4} moves the digits four columns to the right, padding the four leftmost columns with 0, just as happens when multiplying a decimal number by 10 or by 10^{-4} , respectively.

Binary	Decimal
$x = 11010$	26
let $p = x \bmod 2^4 = 01010$	10 (26 mod 16)
let $q = p \cdot 2 = 10100$	20
let $s = x \oplus p = 10000$	16
let $t = s \cdot 2^{-4} = 00001$	1
let $y = q + t = 10101$	21

Here y is the result of a 1-bit left circular shift of x . This algorithm generalizes to an n -bit string x ($p = x \bmod 2^{n-1}$, $q = p \cdot 2$, $s = x \oplus p$, $t = s \cdot 2^{-(n-1)}$, and $y = q + t$). A 2-bit left circular shift can be accomplished by two 1-bit shifts. ●

PRACTICE 46

Use the algorithm of Example 53 to compute the 1-bit circular left shift of 1011 (write the bit strings for x , p , q , and so on).

A ciphertext that is the result of a DES encryption can be decoded into plaintext by reversing all the steps of the encoding process, including applying the keys for each round in the reverse order in which they were used for encoding. Because the DES algorithm is well-known, the only “secret” part is the 56-bit key that is used. Hence a DES ciphertext can be decrypted by a brute force technique of trying all possible 56 bit keys, of which there are 2^{56} . Even assuming that on the average one would hit the right key after trying about half the possibilities, this is still 2^{55} binary strings to generate and test in the DES algorithm. This number used to be considered so impossibly large that DES encoding was deemed perfectly secure, and the U.S. government insisted for many years that it was “unbreakable.” But with faster computers and using parallelism, it’s possible to find the key in a matter of hours.

In recognition of this potential weakness, a new encoding scheme called **AES (Advanced Encryption Standard)** was adopted by the U.S. National Institute of Standards and Technology in 2001 after a five-year design competition. AES is also a block encryption scheme, but it uses a key length of 128 bits or more. (AES also uses a form of the Euclidean algorithm, discussed in Section 2.3, the idea of relatively prime numbers, discussed in Section 2.4, arithmetic modulo n , illustrated in the first few exercises at the end of this section, and ideas from Section 9.1.)

A disadvantage of both DES and AES is that they are **symmetric encryption** (also called **private key encryption**) schemes. The same key is used to both encode and decode the message; for example, in a Caesar cipher, the key is the amount of shift k , and decoding is accomplished by shifting left instead of right. In a private key encryption scheme, both the sender and receiver must know the key. The problem of securely transmitting a message turns into the problem of securely transmitting the key to be used for the encryption and decryption.

Asymmetric encryption (public key encryption) schemes use different keys for encoding and decoding. The decryption key cannot be derived in any practical way from the encryption key, so the encryption key can be made public. Anyone can send a message to the intended receiver in encrypted form using the receiver’s public key, but only the intended receiver, who has the decryption key, can decode it. The best-known asymmetric encryption scheme is the **RSA public key encryption algorithm**, named for its developers Ron Rivest, Adi Shamir, and Len Adleman. RSA uses the mod function, as well as the Euler phi function discussed in Section 2.4, to produce secure public keys. The RSA method works as follows:

1. Two large prime numbers p and q (on the order of 200 digits each) are chosen at random (such numbers can be found relatively easily), and $p \cdot q = n$ is computed.

2. RSA, like DES and AES, is a block cipher, so let us assume that a block B of text has been encoded by some invertible function g into an integer T (much as is done in the Caesar cipher) where T is an integer, $0 < T < n$.
3. The Euler phi function $\varphi(n)$ is computed. Because n is the product of two primes, we know from Equation (2) of Section 2.4 that $\varphi(n) = (p - 1)(q - 1)$.
4. A value e is chosen with $1 < e < \varphi(n)$ such that e and $\varphi(n)$ are relatively prime, that is, $\gcd(e, \varphi(n)) = 1$. (This will surely be true if e is itself a prime number.)
5. $\gcd(e, \varphi(n)) = 1$ means that 1 can be written as a linear combination of e and $\varphi(n)$

$$d \cdot e + f \cdot \varphi(n) = 1$$

or

$$d \cdot e \equiv 1 \pmod{\varphi(n)}$$

The Euclidean algorithm can be used to find unique values for d (and f) such that $1 < d < \varphi(n)$.

6. The public key is the pair of values (n, e) . T is encoded by computing $T^e \pmod n$.
7. The private key is the pair of values (n, d) , of which only the “ d ” part is secret. The received message $T^e \pmod n$ is decoded by computing $(T^e \pmod n)^d \pmod n$.

$$\begin{aligned} (T^e \pmod n)^d \pmod n &= (T^e)^d \pmod n \text{ by Equation (1)} \\ &= T^{ed} \pmod n \text{ with } d \cdot e \equiv 1 \pmod{\varphi(n)} \end{aligned}$$

and it happens that this expression equals T . (The proof depends on two classic results from number theory called *Fermat’s little theorem* and the *Chinese remainder theorem*. See Exercises 37–41.)

8. Then $g^{-1}(T) = B$, the original text block.

Because n and e are public, anyone who could factor n into its prime factors p and q could then reconstruct the decryption key d and decode the message. However, there is no efficient algorithm to find the prime factors of a (large) n . Hence, while the RSA cryptosystem is not technically secure, it is practically so, although a number of demonstration “factor attacks” have been made using many people working together via PCs and the Internet.

Although public key encryption solves the “key distribution” problem of shared keys between sender and receiver, it is a relatively slow encryption/decryption method. Hence in applications such as financial transactions over the Internet, public key encryption is often used for secure transmission of keys for private key communication, which is then used to transmit the actual message. The user’s browser may send a request to the Web server for the server’s public encryption key. Upon receipt, the browser uses this public key to send an encrypted message back to the server that contains a symmetric key for DES encryption. At this point both the browser and the server share a secret key that has been securely transmitted, and the rest of the transaction can be carried out using the faster DES encryption.

EXAMPLE 54

Using RSA encryption/decryption, let $p = 17$ and $q = 13$. Then $n = 221$ and $\varphi(n) = 16 \cdot 12 = 192$. Pick $e = 11$; e and $\varphi(n)$, 11 and 192, are relatively prime. Using the Euclidean algorithm, with 11 and 192,

$$\begin{aligned} 192 &= 17 \cdot 11 + 5 & \text{or} & & 5 &= 192 - 17 \cdot 11 \\ 11 &= 2 \cdot 5 + 1 & \text{or} & & 1 &= 11 - 2 \cdot 5 \\ 5 &= 5 \cdot 1 + 0 \end{aligned}$$

from which

$$1 = 11 - 2 \cdot 5 = 11 - 2[192 - 17 \cdot 11] = 35 \cdot 11 - 2 \cdot 192$$

which gives a value for d of 35. If the integer $T = 8$, then 8 is encoded as follows, where the tedious modulo arithmetic can be broken down using Equation (1). The encoding key is (n, e) and both values are used in the computation.

$$\begin{aligned} 8^{11} \bmod 221 &= 8^3 \cdot 8^3 \cdot 8^3 \cdot 8^2 \bmod 221 \\ &= 512 \cdot 512 \cdot 512 \cdot 64 \bmod 221 \\ &= 70 \cdot 70 \cdot 70 \cdot 64 \bmod 221 \\ &= 4900 \cdot 4480 \bmod 221 \\ &= 38 \cdot 60 \bmod 221 \\ &= 2280 \bmod 221 \\ &= 70 \end{aligned}$$

To decode the encrypted value 70, compute

$$\begin{aligned} 70^{35} \bmod 221 &= (70^2)^{17} \cdot 70 \bmod 221 = (4900)^{17} \cdot 70 \bmod 221 \\ &= 38^{17} \cdot 70 \bmod 221 = (38^2)^8 \cdot 38 \cdot 70 \bmod 221 \\ &= (1444)^8 \cdot 38 \cdot 70 \bmod 221 = (118)^8 \cdot 38 \cdot 70 \bmod 221 \\ &= [(118)^2]^4 \cdot 38 \cdot 70 \bmod 221 = (13924)^4 \cdot 38 \cdot 70 \bmod 221 \\ &= 1^4 \cdot 38 \cdot 70 \bmod 221 = 2660 \bmod 221 = 8 \end{aligned}$$

which was the original encoded integer

This is not a realistic example because p and q (and n) are relatively small. But even here the computations are quite tedious. Again, a spreadsheet and its mod function will be helpful. ●

PRACTICE 47

Assume you receive a ciphertext message of 166 that was encoded using your public key (n, e) from Example 54. Decode the message to obtain the original integer T . ■

Hashing for Password Encryption

The user of a computer system typically has to enter a user ID and password to authenticate him or her as a legitimate user, a person entitled to use these computing resources. The list of user IDs and corresponding passwords must be stored

somewhere in the computer system, and this information is obviously sensitive. Anyone with a copy of the password file has open access to the computer and could even use the ID/password that has the highest level of computer privileges (such as the system administrator ID/password). Therefore the password file, like the secret messages discussed earlier, must be protected. The operating system will, by default, store the password file with the highest level of protection so that only a system administrator account has access. But as an added protection, the file will be encrypted in some way. A password file might have entries something like this:

User ID	Encrypted PW
jgarcia	ax*79%
wbriggs	ee&46#

Unlike secret messages being transmitted, the password file is in constant use. The operating system must be able to quickly authenticate a given user by checking that the password the user enters matches the password stored for that user ID. When the user ID *jgarcia* is entered, one approach would be to find *jgarcia* in the password file, decrypt the corresponding encrypted password (*ax*79%* in this case) and see if the result matches what the user entered. But this means that, however fleetingly, a legitimate “clear” password would be stored within the computer for this user ID, and might be captured by someone hacking the system at that moment, perhaps even someone posing as the *jgarcia* user and guessing at the correct password.

A better approach is to apply the encryption key (also stored in the system) to the password entered and then check the resulting encrypted password against the table entry of *ax*79%*. A match occurs only if the correct password was entered. If a hacker’s attempt fails, he or she does gain the knowledge that what they entered is not the *jgarcia* password, but that is of no help in finding the correct password.

Better still is to use a form of encryption that does not require storing an encryption key. A hash function is often used to encrypt passwords. The ideal **cryptographic hash function** h has two characteristics:

1. Given x , it is easy to compute the hashed value $h(x)$.
2. Given a hashed value z , it is difficult to find a value x for which $h(x) = z$.

Because of these characteristics, a hash function is also called a **one-way encryption**. The password file would now have entries such as

User ID	Hashed PW
jgarcia	$h(\text{password 1})$
wbriggs	$h(\text{password 2})$

Property (2) means that if the password file falls into the wrong hands, it is of little use in trying to break into the system, even if the hash algorithm being used is known. Hence securing the password file is no longer a concern. Property (2) also means that a cryptographic hash function is likely to be more complex than a hash function used to build a hash table for searching, as discussed earlier. There are many well-known encryption/hashing functions, most of which involve use of the modulo n function where n is some power of 2.

There is the slight possibility that user A and user B could pick different passwords that hash to the same value, or even that A and B pick the same password. As far as ordinary usage is concerned, this doesn't matter; both A and B will be authenticated as legitimate users. However, if A steals the password file (no longer securely protected), A could notice that user B 's password hashes to the same value as A 's, and if indeed the passwords are the same, A could log in as B and do damage to B 's files. Most password encryption schemes append some sort of timestamp to a password when the password is first created, and (password + timestamp) is encrypted. If A and B choose the same password, their timestamps (and hashed values) will differ. The timestamp is stored in the password file along with the user ID and when the user enters his or her password, the timestamp is appended and the result hashed and compared to the table entry.

Miscellaneous Applications

Identification Codes

Sometimes codes are used for identification purposes, not the “secret code” purposes we discussed earlier.

EXAMPLE 55

An **International Standard Book Number**, or ISBN, is a numeric identification code associated with a published book. The 10-digit ISBN standard was adopted in 1970, but it was replaced by a 13-digit ISBN standard in 2007. Books published since 2007 generally carry both a 10-digit code and a 13-digit code.

The ISBN-10 is written as four blocks of digits separated by hyphens or blanks. Reading left to right, the first block of digits is a group identifier for a country, area, or language area participating in the ISBN system, the second block of digits identifies the publisher within that group, and the third block is the number assigned by the publisher to this particular work. The final block consists of a single digit from 0–9 or a single X to represent 10. If the first nine digits of the ISBN-10 are

$$a_1a_2a_3a_4a_5a_6a_7a_8a_9$$

then the tenth digit, or check digit, C , is computed by the formula

$$C = \left[\sum_{i=1}^9 i(a_i) \right] \bmod 11$$

—that is, sum up the product of each digit times its position in the list and to the result apply the modulo 11 function.

For example, in the ISBN-10 0-394-80001- X , the 0 indicates the English language group, the 394 identifies the publisher as Random House, the 80001 identifies the title *The Cat in the Hat* by Dr. Seuss, and the check digit X (10) is computed from

$$[1 \cdot 0 + 2 \cdot 3 + 3 \cdot 9 + 4 \cdot 4 + 5 \cdot 8 + 6 \cdot 0 + 7 \cdot 0 + 8 \cdot 0 + 9 \cdot 1] \bmod 11 = 98 \bmod 11 = 10$$

The purpose of the check digit is to detect certain types of errors in the ISBN code, such as a mistyped digit or a transposition of two digits.

In the ISBN-13, there is an additional block of three digits in the front that identifies the type of industry. For a published book, this block is always 978. The next three blocks agree with the ISBN-10, but the final block, again a check digit, is computed differently. If the first 12 digits of the ISBN-13 are

$$a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} a_{11} a_{12}$$

then the 13th digit, the check digit C , is computed from the 12 digits by the formula

$$\left[3 \sum_{i=1}^6 a_{2i} + \sum_{i=0}^5 a_{2i+1} + C \right] \bmod 10 = 0$$

—that is, three times the sum of all the even digits + the sum of all the odd digits plus the check digit should give a multiple of 10.

The ISBN-13 for *The Cat in the Hat* is 978-0-394-80001-1. The check digit C (digit 13) is computed from

$$\begin{aligned} 9 + 3 \cdot 7 + 8 + 3 \cdot 0 + 3 + 3 \cdot 9 + 4 + 3 \cdot 8 + 0 \\ + 3 \cdot 0 + 0 + 3 \cdot 1 + C = 99 + C \end{aligned}$$

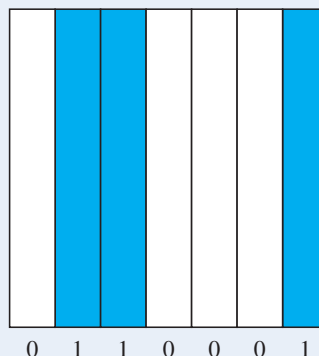
so $C = 1$ to make a total of 100, a multiple of 10. ●

PRACTICE 48

- If an ISBN-10 begins with 0-534-37488, what is the check digit?
- If digits in this number are transposed to 0-534-37848, what is the check digit? ■

EXAMPLE 56

The **UPC-A (Universal Product Code)** is the common bar code found on all goods in stores (food, magazines, and so on). It is an encoding of a 12-digit number, where the first digit is a sort of classification digit, the next five digits are a manufacturer code, the next five digits are the product code, and the last digit is a check sum digit. Each digit is assigned a 7-bit binary code, but this is not the usual binary number representation; for example, the binary code for 5 is 0110001. The 7-bit code for any single digit is represented by a pattern of vertical bars and spaces (space-bar-space-bar if the digit occurs in the left half of the 12-digit string, bar-space-bar-space if it occurs in the right half). The spaces and bars can have varying widths from 1 to 4, 1 being the thinnest and 4 the thickest. If a 5 occurs in the left half of the 12-digit string, its binary code would be represented by a space of width 1, a bar of width 2, a space of width 3, and a bar of width 1.



If a 5 occurs in the right half, the same binary code would be represented by a bar of width 1, a space of width 2, a bar of width 3, and a space of width 4. In addition, there are some extra bits inserted at each end and in the middle for a total of 95 bits.

The check digit C (digit 12) is computed from the previous 11 digits as follows:

$$\left[3 \sum_{i=0}^5 a_{2i+1} + \sum_{i=1}^5 a_{2i} + C \right] \bmod 10 = 0$$

—that is, three times the sum of all the odd digits plus the sum of all the even digits plus the check digit should give a multiple of 10. This formula is similar to the one for the ISBN-13 check digit except that for the ISBN-13 it is the even digits that are multiplied by 3 and for the bar code it is the odd digits. ●

EXAMPLE 57

The American Bankers Association (ABA) devised a code to represent a specific financial institution. This number was originally designed for paper checks, and it does appear on the bottom of paper checks, but today it is also used for such transactions as direct deposit or electronic funds transfer. This **routing number** consists of 8 digits plus a check digit. The first four digits represent Federal Reserve information connected with the financial institution and the second four digits identify the institution itself. The check digit C is computed by

$$\left[3 \sum_{i=0}^2 a_{3i+1} + 7 \sum_{i=0}^2 a_{3i+2} + 1 \sum_{i=0}^1 a_{3i+3} + C \right] \bmod 10 = 0$$

—that is, digits 1, 4, and 7 are multiplied by 3 and added, digits 2, 5, and 8 are multiplied by 7 and added, and digits 3 and 6 are multiplied by 1 and added. The total plus the check digit should be a multiple of 10. ●

Generating and Decomposing Integers

The modulo function provides an easy way to generate integer values within some range 0 through $n - 1$ for some positive integer n . Take any positive integer m and compute $m \bmod n$. If you have a function to generate a random (or pseudorandom) integer m , this process generates a random (or pseudorandom) integer within the desired range. You may also want to cycle through the integers in this range in a controlled fashion.

EXAMPLE 58

You want to show a sequence of five images on your Web page. The images are stored in an arraylike structure as $Image(1)$, ..., $Image(5)$, and the actual image to be displayed is stored in the variable $DisplayImage$, which is initialized to $Image(1)$:

$$i = 1$$

$$DisplayImage = Image(i)$$

$Image(1)$ is the first image displayed. After that, at every clock tick (or at pre-defined intervals), the statements

$$\begin{aligned}i &= i \bmod 5 + 1 \\ DisplayImage &= Image(i)\end{aligned}$$

are executed. The effect is to cycle the index i through the values 2, 3, 4, 5, 1, and so on, and to cycle the displayed images as $Image(2)$, $Image(3)$, $Image(4)$, $Image(5)$, $Image(1)$, and so on. ●

The modulo function can also be used to decompose a multidigit integer into its component digits.

EXAMPLE 59

To decompose a three-digit integer into the ones, tens, and hundreds digits, one can use the following algorithm:

$$\begin{aligned}temp &= number \\ ones &= temp \bmod 10 \\ temp &= (temp - ones)/10\end{aligned}$$

$$\begin{aligned}tens &= temp \bmod 10 \\ temp &= (temp - tens)/10 \\ hundreds &= temp\end{aligned}$$

PRACTICE 49

Given the integer 375, walk through the algorithm of Example 59 to separate the digits. ■

Modular Arithmetic Designs

The mod function can be used to make interesting quiltlike patterns by arranging “tile” images (small square images) in tables based on addition modulo n and then repeating these tables to form the completed quilt image.⁴ **Addition modulo n** is defined on the set of integers $\{0, 1, 2, \dots, n - 1\}$ by

$$x +_n y = (x + y) \bmod n$$

EXAMPLE 60

In this example we will use addition modulo 6. Addition modulo 6 applies to the integers $\{0, 1, 2, 3, 4, 5\}$ using the rule $x +_6 y = (x + y) \bmod 6$. The addition table follows. It shows, for example,

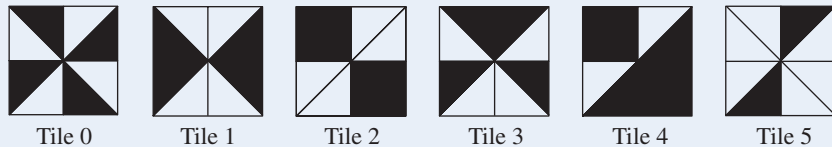
$$3 +_6 4 = (3 + 4) \bmod 6 = 7 \bmod 6 = 1$$

⁴This section on modular arithmetic designs was adapted with permission from <http://britton.disted.camosun.bc.ca/modart/jbmodart2.htm>; the software used to generate the design is Cayley Quilter, available to download at <http://www.wou.edu/~burtonl/cquilter.html>

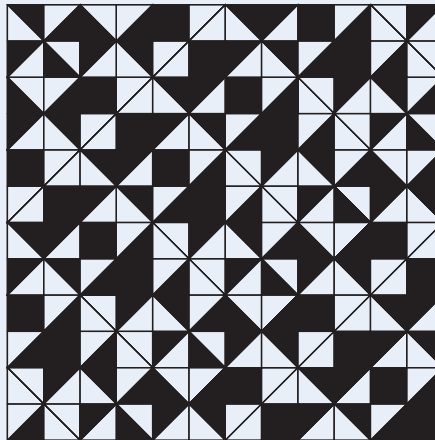
No values of 6 appear in the table, and each row is a circular left shift of the previous row.

$+_6$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

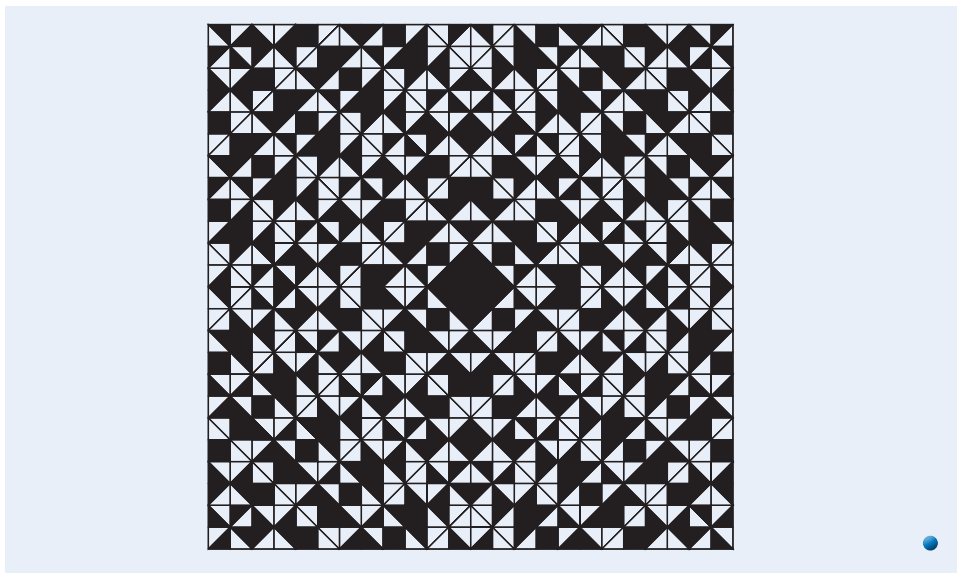
We will need six tile images of various geometric designs:



The tiles are then arranged in a 6×6 table according to the addition modulo 6 pattern, that is, the interior of the addition modulo 6 table. The top row shows images left to right of 0, 1, 2, 3, 4, 5, the next row shows images left to right of 1, 2, 3, 4, 5, 0, and so forth.



The final image is composed of four copies of the table arranged in a 2×2 grid where the original table goes in the upper-left quadrant, its horizontal reflection goes in the upper-right quadrant, and the bottom two quadrants are vertical reflections of the top half.



SECTION 5.6 REVIEW

TECHNIQUES

- Perform calculations using the mod function.
- Build or search a hash table that uses linear probing for collision resolution.
- Ⓜ Build or search a hash table that uses chaining for collision resolution.
- Encode and decode using a Caesar cipher.
- Apply the algorithm to compute a circular left shift of a binary string.

- Ⓜ Perform encryption and decryption using the RSA public key encryption algorithm.
- Compute the check digit for the ISBN-10, ISBN-13, UPC-A, and ABA codes.

MAIN IDEAS

- The humble modulo function is useful in many important applications.

EXERCISES 5.6

Exercises 1–8 concern modular arithmetic mod n .

1. $25 = 11 + 14$. Show by computing each expression that

$$25 \bmod 6 = (11 \bmod 6 + 14 \bmod 6) \bmod 6$$

2. $395 = 129 + 266$. Show by computing each expression that

$$395 \bmod 4 = (129 \bmod 4 + 266 \bmod 4) \bmod 4$$

3. $262 = 74 + 188$. Show by computing each expression that

$$262 \bmod 13 = (74 \bmod 13 + 188 \bmod 13) \bmod 13$$

4. Prove that for any integers x and y ,

$$(x + y) \bmod n = (x \bmod n + y \bmod n) \bmod n$$

5. $486 = 18 \cdot 27$. Show by computing each expression that

$$486 \bmod 5 = (18 \bmod 5 \cdot 27 \bmod 5) \bmod 5$$

6. $7067 = 191 \cdot 37$. Show by computing each expression that

$$7067 \bmod 8 = (191 \bmod 8 \cdot 37 \bmod 8) \bmod 8$$

7. Prove that for any integers x and y ,

$$(x \cdot y) \bmod n = (x \bmod n \cdot y \bmod n) \bmod n$$

8. Prove or disprove: For any positive integer x , $(-x) \bmod n = -(x \bmod n)$.

9. Using the hash function of Example 51, which of the following Social Security numbers would cause a collision with 328356770, the Social Security number of that example?

- a. 060357896
- b. 896137243
- c. 712478993
- d. 659027781

10. Find a set of five numbers in the range $[0, 200]$ that cause 100% collision using the hash function

$$h(x) = x \bmod 13$$

11. Using a hash table of size 11 and the hash function $x \bmod 11$, show the results of hashing the following values into a hash table using linear probing for collision resolution:

$$1, 13, 12, 34, 38, 33, 27, 22$$

12. Using the completed hash table from Exercise 11, compute the average number of comparisons needed to perform a successful search for a value in the table.

13. When a computer program is compiled, the compiler builds a symbol table for storing information about the identifiers used in the program. A scheme is needed to quickly decide whether a given identifier has already been stored in the table and, if not, to store the new identifier. A hash function is often used to locate a position in the table at which to store information about an item.

For simplicity, assume that the items to be stored are integers, that the hash table can hold 17 items in positions 0–16, and that the hash function $h(x)$ is given by $h(x) = x \bmod 17$.

Linear probing is used for collision resolution.

- a. Using the hash function and collision resolution scheme described, store the sequence of values 23, 14, 52, 40, 24, 18, 33, 58, 50. Give the location in the table at which each is stored.
- b. After the table of part (a) has been filled, describe the process to search for 58 in the table. Describe the process to search (unsuccessfully) for 41 in the table.

14. Explain what problem can arise if an item stored in a hash table is later deleted.
15. A disadvantage of hashing with linear probing for collision resolution is that elements begin to cluster together in groups of adjacent array locations. Assume that you have a very good hashing function (it distributes elements evenly throughout the hash table). Start with an empty hash table of size t that will store data using linear probing for collision resolution.
- What is the probability of hashing the first element to location p (and storing it there, since it is the first item and there will be no collisions)?
 - Once location p is occupied, what is the probability of storing the second item in location $p + 1$ (modulo the table size)?
 - Once locations p and $p + 1$ are occupied, what is the probability of storing the third item in location $p + 2$ (modulo the table size)?
16. Generalize the answers to Exercise 15 to explain why using linear probing for collision resolution causes clustering.
17. Decode the following ciphertext messages that were encoded using a Caesar cipher with the given key.
- JUUBFNUUCQJCNWMBFNUU, $k = 9$
 - XAEWFVMPMKERHXLWPMXLCXSZIWHMHKCVIERHKMQFPIMRXLIAEFI, $k = 4$
 - IURSAYZGXJCOZLNKQOTOLKOTZLNKROHXGXE, $k = 6$
18. Using a Caesar cipher, encode the following plaintext messages.
- ATTACK FROM BEYOND THE RIVER, $k = 5$
 - WE ARE SHORT OF SUPPLIES, $k = 10$
 - BADLY NEED AMMUNITION, $k = 8$
19. The following ciphertext message is intercepted; you suspect it is a Caesar cipher. Find a value of k that decodes the message, and give the corresponding plaintext.

BUNNYWXFFNVJALQXWAXVNCXVXAAXF

20. The following ciphertext message is intercepted; you suspect it is a Caesar cipher. Find a value of k that decodes the message, and give the corresponding plaintext.

EQQKAGAZRMOQNAAWPGPQ

21. Use the algorithm of Example 53 to compute the 1-bit circular left shift of the following binary strings (write the bit strings for x , p , q , and so on).
- 10011
 - 0011
22. Use the algorithm of Example 53 to compute the 1-bit circular left shift of the following binary strings (write the bit strings for x , p , q , etc.).
- 10110
 - 1110
23. Consider a “short form” of DES that uses 16-bit keys. Given the 16-bit key

1101000101110101

as input to a DES round that uses a circular left shift of 2 bits, what would be the key for the next round?

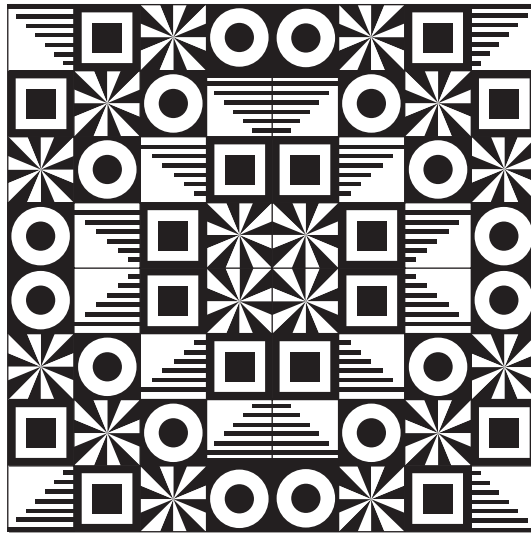
24. Describe how to perform a 1-bit right circular shift on a 4-bit binary string x using the left circular shift operation.
25. Using RSA encryption/decryption, let $p = 5$ and $q = 3$. Then $n = 15$ and $\varphi(n) = 4 \cdot 2 = 8$. Pick $e = 3$.
- Use the Euclidean algorithm to find the value of d .
 - Encode $T = 8$ using the public key (n, e) .
 - Decode your answer to part (b) to retrieve the 8.
26. Why is the RSA encryption of Exercise 25 a poor choice?
27. Using RSA encryption/decryption, let $p = 5$ and $q = 11$. Then $n = 55$ and $\varphi(n) = 4 \cdot 10 = 40$. Pick $e = 7$.
- Use the Euclidean algorithm to find the value of d . (*Hint: If the Euclidean algorithm produces an equation $1 = x \cdot e + f \cdot \varphi(n)$ where the value of x is negative, add and subtract the product $e \cdot \varphi(n)$ to the right side of the equation to get a positive value for d .)*
 - Encode $T = 12$ using the public key (n, e) .
 - Decode your answer to part (b) to retrieve the 12.
28. Using RSA encryption/decryption, let $p = 23$ and $q = 31$. Then $n = 713$ and $\varphi(n) = 22 \cdot 30 = 660$. Pick $e = 17$.
- Use the Euclidean algorithm to find the value of d .
 - Encode $T = 52$ using the public key (n, e) .
 - Decode your answer to part (b) to retrieve the 52.
29. a. All n people in a group wish to communicate with each other using messages encrypted with DES or AES. A different secret key must be shared between each pair of users. How many keys are required?
b. All n people in a group wish to communicate with each other using messages encrypted with a public key encryption system. How many keys are required?
30. Computer users are notoriously lax about choosing passwords; left to their own devices, they tend to pick short or really obvious passwords. At Simpleton University, passwords must contain only lowercase letters, and the campus computer system uses an (unrealistically simple) cryptographic hash function given by the following algorithm:
- Letters in the password are converted to an integer equivalent ($a \rightarrow 1, b \rightarrow 2$, and so on).
 - In the result of step 2, all individual digits are added (for example, 17 becomes $1 + 7$) to give an integer value x .
 - $h(x) = x \bmod 2^5$
- Joe Hack has managed to steal the password table, and he notices that there is a password entry of 20 for *bsmith*. Joe decides to try to hack into the system as *bsmith* by guessing *bsmith*'s password. Can you guess *bsmith*'s password?
31. a. The ISBN-10 of the sixth edition of this book is 0-7167-6864- C where C is the check digit. What is the check digit?
b. What is the ISBN-13 of the sixth edition?
32. A bookstore placed an order for 2000 copies of *Harry Potter and the Deathly Hallows*, the seventh and final volume in the hugely popular Harry Potter series by J. K. Rowling. When placing the order with the publisher, the ISBN-13 978-0-545-01022-5 was used. Is this correct?
33. Given the 11 digits 02724911637, compute the check digit for the UPC-A code.

34. A taxpayer wants his tax refund to be deposited directly to his bank. He enters the bank's routing number, including the check digit, as

025107036

Is this routing number correct?

35. a. Write an algorithm to decompose a four-digit integer into the ones, tens, hundreds, and thousands digits.
 b. Apply this algorithm to decompose the integer 7426.
36. The following quilt image is based on addition modulo n for what value of n ?



Exercises 37–41 involve a proof of step 7 of the RSA method.

37. Prove that if $x \equiv y \pmod{n}$ and c is a constant integer, then $xc \equiv yc \pmod{n}$.
38. This exercise explores the converse of Exercise 37, which is the issue of *cancellation under congruence modulo n* . In other words, if $xc \equiv yc \pmod{n}$ for some constant integer c , is it then true that $x \equiv y \pmod{n}$? Not always, as it turns out.
- a. Prove that if $\gcd(c, n) = 1$, then $xc \equiv yc \pmod{n}$ implies $x \equiv y \pmod{n}$.
- b. Prove that $xc \equiv yc \pmod{n}$ implies $x \equiv y \pmod{n}$ only if $\gcd(c, n) = 1$. To do this, it is easier to prove the contrapositive:

If $\gcd(c, n) \neq 1$, then there exist integer values x and y for which $xc \equiv yc \pmod{n}$ but $x \not\equiv y \pmod{n}$.

(Hint: Suppose $c = m_1k$ and $n = m_2k$ with $k > 1$. Consider $x = m_2k$ and $y = m_2$.)

- c. Find values for x, y, c , and n where $xc \equiv yc \pmod{n}$ but $x \not\equiv y \pmod{n}$.
39. If p is a prime number and a is a positive integer not divisible by p , then

$$a^{p-1} \equiv 1 \pmod{p}$$

This result is known as *Fermat's little theorem* (as opposed to the very famous Fermat's last theorem mentioned in Section 2.4).

Let $S = \{0, a, 2a, \dots, (p-1)a\}$, $T = \{0, 1, 2, \dots, (p-1)\}$. Let f be given by $f(ka) = (ka) \bmod p$; that is, f computes the residue modulo p .

- Prove that f is a one-to-one function from S to T .
 - Prove that f is an onto function.
 - Prove that $[a \cdot 2a \cdots (p-1)a] \bmod p = (p-1)! \bmod p$
 - Prove that $a^{p-1} \equiv 1 \pmod{p}$
 - Let $a = 4$ and $p = 7$. Compute the set of residues modulo p of $\{4, 8, 12, \dots, 24\}$.
 - Let $a = 4$ and $p = 7$. Show by direct computation that $4^6 \equiv 1 \pmod{7}$.
40. Let m_1, m_2, \dots, m_n be pairwise relatively prime positive integers (that is, $\gcd(m_i, m_k) = 1$ for $1 \leq i, k \leq n, i \neq k$), let $m = m_1 m_2 \cdots m_n$, and let a_1, a_2, \dots, a_n be any integers. Then there is an integer x such that

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_n \pmod{m_n} \end{aligned}$$

and any other integer y that satisfies these relations is congruent to x modulo m . This result is known as the *Chinese remainder theorem* (based on work done by the Chinese mathematician Sun-Tsu in the first century AD). The following steps will prove the Chinese remainder theorem.

- Let s and t be positive integers with $\gcd(s, t) = 1$. Prove that there exists an integer w such that $sw \equiv 1 \pmod{t}$.
 - For each i , $1 \leq i \leq n$, let $M_i = m/m_i$. Prove that $\gcd(M_i, m_i) = 1$.
 - Prove that there is an integer x_i such that $M_i x_i \equiv 1 \pmod{m_i}$ and $a_i M_i x_i \equiv a_i \pmod{m_i}$.
 - Prove that $a_k M_k x_k \equiv 0 \pmod{m_i}$ for all $k \neq i$.
 - Let $x = a_1 M_1 x_1 + a_2 M_2 x_2 + \cdots + a_n M_n x_n$. Prove that for $1 \leq i \leq n$, $x \equiv a_i \pmod{m_i}$.
 - Let y be such that $y \equiv a_i \pmod{m_i}$, $1 \leq i \leq n$. Prove that $x \equiv y \pmod{m}$. (*Hint*: Use the fundamental theorem of arithmetic and write m as a product of distinct primes, $m = p_1^{k_1} p_2^{k_2} \cdots p_t^{k_t}$).
41. The remaining step in the proof of the RSA algorithm is to show that if $d \cdot e \equiv 1 \pmod{\varphi(n)}$, then $T^{ed} \bmod n = T$.
- Prove that T^{ed} can be written as $T(T^{p-1})^{k(q-1)}$ or as $T(T^{q-1})^{k(p-1)}$ for some integer k .
 - Prove that if T is not divisible by p , then $T^{ed} \equiv T \pmod{p}$, and if T is not divisible by q , then $T^{ed} \equiv T \pmod{q}$.
 - Prove that if $p|T$, then $T^{ed} \equiv T \pmod{p}$ and T is not divisible by q , so $T^{ed} \equiv T \pmod{q}$ by part (b).
 - Prove that if $q|T$, then $T^{ed} \equiv T \pmod{q}$ and T is not divisible by p , so $T^{ed} \equiv T \pmod{p}$ by part (b).
 - From parts (b)–(d) $T^{ed} \equiv T \pmod{p}$ and $T^{ed} \equiv T \pmod{q}$ in all cases. Prove that $T^{ed} \bmod n = T$.

SECTION 5.7 MATRICES

Terminology

Data about many kinds of problems can often be represented using a rectangular arrangement of values; such an arrangement is called a **matrix**. Thus

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 4 \\ 3 & -6 & 8 \end{bmatrix}$$

is a matrix with two rows and three columns. The **dimensions** of the matrix are the number of rows and columns; here \mathbf{A} is a 2×3 matrix.

Elements of a matrix \mathbf{A} are denoted by a_{ij} , where i is the row number of the element in the matrix and j is the column number. In the example matrix \mathbf{A} , $a_{23} = 8$ because 8 is the element in row 2, column 3, of \mathbf{A} .

EXAMPLE 61

Average temperatures in three different cities for each month can be neatly summarized in a 3×12 matrix. Here we interpret the 3 rows as the 3 cities and the 12 columns as the 12 months January–December. The average temperature in the third city in April, a_{34} , is 67.

$$\mathbf{A} = \begin{bmatrix} 23 & 26 & 38 & 47 & 58 & 71 & 78 & 77 & 69 & 55 & 39 & 33 \\ 14 & 21 & 33 & 38 & 44 & 57 & 61 & 59 & 49 & 38 & 25 & 21 \\ 35 & 46 & 54 & 67 & 78 & 86 & 91 & 94 & 89 & 75 & 62 & 51 \end{bmatrix}$$

EXAMPLE 62

In Practice 2(c) of Section 5.1, the binary relation $\{(7, 9), (2, 5), (9, 9), (2, 7)\}$ was defined on the set $S = \{2, 5, 7, 9\}$. Although a set is unordered, we can impose an ordering on the elements of S so that 2 is element 1 in S , 5 is element 2, and so forth. The matrix \mathbf{R} below represents this binary relation by a 1 entry in position i, j if element i in set S is related to element j . Because $(7, 9)$ says that element 3 is related to element 4, element $r_{3,4}$ of \mathbf{R} equals 1.

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

EXAMPLE 63

Solutions to many problems can be obtained by solving systems of linear equations. Suppose, for example, that you are placing an order for coffee beans for your sidewalk café. You want to order 70 pounds of beans, a mixture of Kona coffee beans and Colombian coffee beans. You are willing to spend \$1180; Kona coffee costs \$24 per pound, and Colombian coffee costs \$14 per pound. How many pounds of each should you order?

The constraints in this problem are represented by the system of linear equations

$$\begin{aligned}x + y &= 70 \\24x + 14y &= 1180\end{aligned}$$

These are **linear equations** because each of the two unknowns, x and y , appears only to the first power. (Also an equation such as $ax + by = c$ represents a straight line when graphed on an x - y coordinate system.) The solution to this system of equations is $x = 20, y = 50$ (you can easily check that this is a solution). The matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 24 & 14 \end{bmatrix}$$

is the **matrix of coefficients** for this system of linear equations. As we will see, the matrix of coefficients can be used to solve a system of linear equations. ●

PRACTICE 50 In the matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & 4 & -6 & 8 \\ 3 & 0 & 1 & -7 \end{bmatrix}$$

What is a_{23} ? What is a_{24} ? What is a_{13} ? ■

In a matrix, the arrangement of the entries is significant. Therefore, for two matrices to be **equal**, they must have the same dimensions and the same entries in each location.

EXAMPLE 64 Let

$$\begin{aligned}\mathbf{X} &= \begin{bmatrix} x & 4 \\ 1 & y \\ z & 0 \end{bmatrix} \\ \mathbf{Y} &= \begin{bmatrix} 3 & 4 \\ 1 & 6 \\ 2 & w \end{bmatrix}\end{aligned}$$

If $\mathbf{X} = \mathbf{Y}$, then $x = 3, y = 6, z = 2$, and $w = 0$. ●

We will often be interested in square matrices, in which the number of rows equals the number of columns. If \mathbf{A} is an $n \times n$ square matrix, then the elements $a_{11}, a_{22}, \dots, a_{nn}$ form the **main diagonal** of the matrix. If the corresponding elements match when we think of folding the matrix along the main diagonal, then the matrix is symmetric about the main diagonal. In a **symmetric** matrix, $a_{ij} = a_{ji}$.

EXAMPLE 65

The square 3×3 matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 5 & 7 \\ 5 & 0 & 2 \\ 7 & 2 & 6 \end{bmatrix}$$

is symmetric. The upper triangular part (the portion above the main diagonal) is a reflection of the lower triangular part. Note that $a_{21} = a_{12} = 5$. ●

A more general way to represent arrangements of data is the **array**. Arrays are n -dimensional arrangements of data, where n can be any positive integer. If $n = 1$, then the data are arranged in a single line, which is therefore a list or finite sequence of data items. This one-dimensional version of an array is called a **vector**. If $n = 2$, the array is a matrix. If $n = 3$, we can picture layers of two-dimensional matrices. For $n > 3$, we can formally deal with the array elements, but we can't really visualize the arrangement. The array data structure is available in many high-level programming languages because it is such a useful way to represent data in list form or tabular form. Generally, the number of elements expected in each dimension of the array must be declared in the program. The array \mathbf{X} of Example 64, for instance, would be declared as a 3×2 array—a two-dimensional array (matrix) with three elements in one dimension and two in the other (that is, three rows and two columns).

Matrix Operations

Although matrices are particular arrangements of individual elements, we can treat the matrices themselves as objects, just as we can treat sets of elements as objects. In each case we are abstracting up one level and looking at the *collection* as an entity, rather than looking at the individual elements that make up the collection. We defined operations on sets (union, intersection, and so forth) that made sets useful for solving counting problems. We can define arithmetic operations on matrices whose entries are numerical. These operations make matrices interesting objects to study in their own right, but they also make matrices more useful for certain tasks such as solving systems of equations.

The first operation, called **scalar multiplication**, calls for multiplying each entry of a matrix by a fixed single number called a **scalar**. The result is a matrix with the same dimensions as the original matrix.

EXAMPLE 66

The result of multiplying matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 4 & 5 \\ 6 & -3 & 2 \end{bmatrix}$$

by the scalar $r = 3$ is

$$\mathbf{A} = \begin{bmatrix} 3 & 12 & 15 \\ 18 & -9 & 6 \end{bmatrix}$$



Addition of two matrices \mathbf{A} and \mathbf{B} is defined only when \mathbf{A} and \mathbf{B} have the same dimensions; then it is simply a matter of adding the corresponding elements. Formally, if \mathbf{A} and \mathbf{B} are both $n \times m$ matrices, then $\mathbf{C} = \mathbf{A} + \mathbf{B}$ is an $n \times m$ matrix with entries

$$c_{ij} = a_{ij} + b_{ij}$$

EXAMPLE 67 For

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 6 \\ 2 & 0 & 4 \\ -4 & 5 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & -2 & 8 \\ 1 & 5 & 2 \\ 2 & 3 & 3 \end{bmatrix}$$

the matrix $\mathbf{A} + \mathbf{B}$ is

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} 1 & 1 & 14 \\ 3 & 5 & 6 \\ -2 & 8 & 4 \end{bmatrix}$$

PRACTICE 51 For $r = 2$,

$$\mathbf{A} = \begin{bmatrix} 1 & 7 \\ -3 & 4 \\ 5 & 6 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 4 & 0 \\ 9 & 2 \\ -1 & 4 \end{bmatrix}$$

find $r\mathbf{A} + \mathbf{B}$.

Subtraction of matrices is defined by $\mathbf{A} - \mathbf{B} = \mathbf{A} + (-1)\mathbf{B}$.

In a **zero matrix**, all entries are 0. If we add an $n \times m$ zero matrix, denoted by $\mathbf{0}$, to any $n \times m$ matrix \mathbf{A} , the result is matrix \mathbf{A} . We can symbolize this by the matrix equation

$$\mathbf{0} + \mathbf{A} = \mathbf{A}$$

This equation is true because of a similar equation that holds for all the individual numerical entries, $0 + a_{ij} = a_{ij}$. Other matrix equations are also true because of similar equations that hold for the individual entries.

EXAMPLE 68

If \mathbf{A} and \mathbf{B} are $n \times m$ matrices and r and s are scalars, the following matrix equations are true:

$$\begin{aligned}\mathbf{0} + \mathbf{A} &= \mathbf{A} \\ \mathbf{A} + \mathbf{B} &= \mathbf{B} + \mathbf{A} \\ (\mathbf{A} + \mathbf{B}) + \mathbf{C} &= \mathbf{A} + (\mathbf{B} + \mathbf{C}) \\ r(\mathbf{A} + \mathbf{B}) &= r\mathbf{A} + r\mathbf{B} \\ (r + s)\mathbf{A} &= r\mathbf{A} + s\mathbf{A} \\ r(s\mathbf{A}) &= (rs)\mathbf{A}\end{aligned}$$

To prove that $\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A}$, for instance, it is sufficient to note that $a_{ij} + b_{ij} = b_{ij} + a_{ij}$ for each entry in matrices \mathbf{A} and \mathbf{B} . ●

One might expect that **multiplication of matrices** would simply involve multiplying corresponding elements in the two matrices, but the definition is more complicated than that. The definition of matrix multiplication is based on the use of matrices in mathematics to represent functions called linear transformations, which map points in the real-number plane to points in the real-number plane. Although we won't use matrices in this way, we will use the standard definition for matrix multiplication.

To compute \mathbf{A} times \mathbf{B} , $\mathbf{A} \cdot \mathbf{B}$, the number of columns in \mathbf{A} must equal the number of rows in \mathbf{B} . (This requirement means that the number of elements in a single row of \mathbf{A} equals the number of elements in a single column of \mathbf{B} .) Thus we can compute $\mathbf{A} \cdot \mathbf{B}$ if \mathbf{A} is an $n \times m$ matrix and \mathbf{B} is an $m \times p$ matrix. The result is an $n \times p$ matrix. An entry in row i , column j of $\mathbf{A} \cdot \mathbf{B}$ is obtained by multiplying all the elements in row i of \mathbf{A} by the corresponding elements in column j of \mathbf{B} and adding the results. Formally, $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$, where

$$c_{ij} = \sum_{k=1}^m a_{ik}b_{kj}$$

EXAMPLE 69

Let

$$\mathbf{A} = \begin{bmatrix} 2 & 4 & 3 \\ 4 & -1 & 2 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 5 & 3 \\ 2 & 2 \\ 6 & 5 \end{bmatrix}$$

\mathbf{A} is a 2×3 matrix and \mathbf{B} is a 3×2 matrix, so the product $\mathbf{A} \cdot \mathbf{B}$ exists and is a 2×2 matrix \mathbf{C} . To find element c_{11} , we multiply corresponding elements of row 1 of \mathbf{A} and column 1 of \mathbf{B} and add the results.

$$2(5) + 4(2) + 3(6) = 10 + 8 + 18 = 36$$

$$\begin{bmatrix} \boxed{2} & 4 & 3 \\ 4 & -1 & 2 \end{bmatrix} \begin{bmatrix} \boxed{5} & 3 \\ 2 & 2 \\ 6 & 5 \end{bmatrix} = \begin{bmatrix} 36 & \text{---} \\ \text{---} & \text{---} \end{bmatrix}$$

Element c_{12} is obtained by multiplying corresponding elements of row 1 of \mathbf{A} and column 2 of \mathbf{B} and adding the results.

$$\begin{bmatrix} 2 & 4 & 3 \\ 4 & -1 & 2 \end{bmatrix} \begin{bmatrix} 5 & 3 \\ 2 & 2 \\ 6 & 5 \end{bmatrix} = \begin{bmatrix} 36 & 29 \\ - & - \end{bmatrix}$$

The complete product is

$$\begin{bmatrix} 2 & 4 & 3 \\ 4 & -1 & 2 \end{bmatrix} \begin{bmatrix} 5 & 3 \\ 2 & 2 \\ 6 & 5 \end{bmatrix} = \begin{bmatrix} 36 & 29 \\ 30 & 20 \end{bmatrix}$$

PRACTICE 52 Compute $\mathbf{A} \cdot \mathbf{B}$ and $\mathbf{B} \cdot \mathbf{A}$ for

$$\mathbf{A} = \begin{bmatrix} 1 & 4 \\ 6 & -2 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 3 & 6 \\ 3 & 4 \end{bmatrix}$$

From Practice 52 we see that even if \mathbf{A} and \mathbf{B} have dimensions so that both $\mathbf{A} \cdot \mathbf{B}$ and $\mathbf{B} \cdot \mathbf{A}$ are defined, $\mathbf{A} \cdot \mathbf{B}$ need not equal $\mathbf{B} \cdot \mathbf{A}$. There are, however, several matrix equations involving multiplication that are true.

EXAMPLE 70

Where \mathbf{A} , \mathbf{B} , and \mathbf{C} are matrices of appropriate dimensions and r and s are scalars, the following matrix equations are true:

$$\begin{aligned} \mathbf{A} \cdot (\mathbf{B} \cdot \mathbf{C}) &= (\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{C} \\ \mathbf{A}(\mathbf{B} + \mathbf{C}) &= \mathbf{A} \cdot \mathbf{B} + \mathbf{A} \cdot \mathbf{C} \\ (\mathbf{A} + \mathbf{B})\mathbf{C} &= \mathbf{A} \cdot \mathbf{C} + \mathbf{B} \cdot \mathbf{C} \\ r\mathbf{A} \cdot s\mathbf{B} &= (rs)(\mathbf{A} \cdot \mathbf{B}) \end{aligned}$$

Verifying these equations for matrices of particular dimensions is simple, if tedious.

The $n \times n$ matrix with 1s along the main diagonal and 0s elsewhere is called the **identity matrix**, denoted by \mathbf{I} . If we multiply \mathbf{I} times any $n \times n$ matrix \mathbf{A} , we get \mathbf{A} as the result. The equation

$$\mathbf{I} \cdot \mathbf{A} = \mathbf{A} \cdot \mathbf{I} = \mathbf{A}$$

holds.

PRACTICE 53 Let

$$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix}$$

Verify that $\mathbf{I} \cdot \mathbf{A} = \mathbf{A} \cdot \mathbf{I} = \mathbf{A}$.

An $n \times n$ matrix \mathbf{A} is **invertible** if there exists an $n \times n$ matrix \mathbf{B} such that

$$\mathbf{A} \cdot \mathbf{B} = \mathbf{B} \cdot \mathbf{A} = \mathbf{I}$$

In this case \mathbf{B} is called the **inverse** of \mathbf{A} , denoted by \mathbf{A}^{-1} .

EXAMPLE 71 Let

$$\mathbf{A} = \begin{bmatrix} -1 & 2 & -3 \\ 2 & 1 & 0 \\ 4 & -2 & 5 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -5 & 4 & -3 \\ 10 & -7 & 6 \\ 8 & -6 & 5 \end{bmatrix}$$

Then, following the rules of matrix multiplication, it can be shown (Practice 54) that $\mathbf{A} \cdot \mathbf{B} = \mathbf{B} \cdot \mathbf{A} = \mathbf{I}$, so $\mathbf{B} = \mathbf{A}^{-1}$.

PRACTICE 54 For the matrices \mathbf{A} and \mathbf{B} of Example 70,

- Compute $\mathbf{A} \cdot \mathbf{B}$.
- Compute $\mathbf{B} \cdot \mathbf{A}$.

It is easy to write an algorithm for matrix multiplication by simply following the definition. A pseudocode version of the algorithm follows, where bracket notation $\mathbf{A}[i, j]$ replaces the subscript notation a_{ij} .

ALGORITHM *MATRIXMULTIPLICATION*

```
//computes  $n \times p$  matrix  $\mathbf{A} \cdot \mathbf{B}$  for  $n \times m$  matrix  $\mathbf{A}$ ,  $m \times p$  matrix  $\mathbf{B}$ 
//stores result in  $\mathbf{C}$ 
for  $i = 1$  to  $n$  do
  for  $j = 1$  to  $p$  do
     $\mathbf{C}[i, j] = 0$ 
    for  $k = 1$  to  $m$  do
       $\mathbf{C}[i, j] = \mathbf{C}[i, j] + \mathbf{A}[i, k] * \mathbf{B}[k, j]$ 
    end for
  end for
end for
write out product matrix  $\mathbf{C}$ 
```

The computational steps done in this algorithm are multiplications and additions, one multiplication and one addition each time the statement $\mathbf{C}[i, j] = \mathbf{C}[i, j] + \mathbf{A}[i, k] * \mathbf{B}[k, j]$ is executed. This statement occurs within a triply nested loop and will be executed n^3 times. (Although this is quite obvious, it can also be justified by the multiplication principle as the number of possible outcomes of choosing indices i, j , and k .) If \mathbf{A} and \mathbf{B} are both $n \times n$ matrices, then there are $\Theta(n^3)$ multiplications and $\Theta(n^3)$ additions required. The total amount of work is therefore $\Theta(n^3) + \Theta(n^3) = \Theta(n^3)$.

Given the definition of matrix multiplication, it is hard to see how one could avoid $\Theta(n^3)$ steps in computing the product of two $n \times n$ matrices, but a sufficiently clever approach does yield an improvement under certain conditions (see Exercise 54).

Gaussian Elimination

In Example 63 we encountered the following system of two linear equations in two unknowns:

$$\begin{aligned}x + y &= 70 \\24x + 14y &= 1180\end{aligned}$$

The general form for a system of n linear equations in n unknowns is

$$\begin{aligned}a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\&\vdots \\a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n\end{aligned}$$

with a matrix of coefficients

$$\begin{bmatrix}a_{11} & a_{12} & \cdots & a_{1n} \\a_{21} & a_{22} & \cdots & a_{2n} \\& \vdots & & \\a_{n1} & a_{n2} & \cdots & a_{nn}\end{bmatrix}$$

To solve this system of equations, we first form the **augmented** $n \times (n + 1)$ matrix by adding the column of b 's to the matrix of coefficients:

$$\begin{bmatrix}a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\& \vdots & & & \\a_{n1} & a_{n2} & \cdots & a_{nn} & b_n\end{bmatrix}$$

(The augmented matrix is simply a convenience to avoid having to write all the unknowns.) The next step is to “transform” the augmented matrix into one where the matrix-of-coefficients part is an **upper triangular matrix**, that is, all values of this $n \times n$ matrix below the main diagonal are 0's. The result will be a matrix of the form

$$\begin{bmatrix} c_{11} & c_{12} & \dots & & c_{1n} & d_1 \\ 0 & c_{22} & \dots & & c_{2n} & d_1 \\ & \vdots & & & & \\ 0 & 0 & \dots & c_{(n-1)(n-1)} & c_{(n-1)n} & d_{n-1} \\ 0 & 0 & \dots & 0 & c_{nn} & d_n \end{bmatrix}$$

Now we can turn this back into a system of equations of the form

$$\begin{bmatrix} c_{11}x_1 & c_{12}x_2 & \dots & & c_{1n}x_n & d_1 \\ 0 & c_{22}x_2 & \dots & & c_{2n}x_n & d_1 \\ & \vdots & & & & \\ 0 & 0 & \dots & c_{(n-1)(n-1)}x_{n-1} & c_{(n-1)n}x_n & d_{n-1} \\ 0 & 0 & \dots & 0 & c_{nn}x_n & d_n \end{bmatrix}$$

and solve the equations from the bottom up. We solve

$$c_{nr}x_n = d_n$$

for x_n . Knowing the value of x_n , we can then solve the next-to-last equation

$$c_{(n-1)(n-1)}x_{n-1} + c_{(n-1)n}x_n = d_{n-1}$$

for x_{n-1} and so forth, back up to the top row.

But how do we do the transformation? The allowable operations to carry out this transformation are called **elementary row operations**, none of which change the solution set of the underlying equations. These operations (performed on the augmented matrix) are

- i. Switch any two rows of the matrix.
- ii. Multiply all the elements in any one row of the matrix by a non-zero scalar.
- iii. Add a scalar multiple of any one row to another row.

This process for solving systems of linear equations is known as **Gaussian elimination**, named for the famous German mathematician Karl Friedrich Gauss. Gauss did not actually invent this process, however; it was demonstrated in a Chinese mathematics treatise of the second century AD, and it was probably known in China even earlier.

EXAMPLE 72

Solving the system of equations

$$\begin{aligned} x + y &= 70 \\ 24x + 14y &= 1180 \end{aligned}$$

using Gaussian elimination, we first form the augmented matrix

$$\begin{bmatrix} 1 & 1 & 70 \\ 24 & 14 & 1180 \end{bmatrix}$$

We multiply row 1 by the scalar -24 and add the result to row 2 (the third elementary row operation), giving

$$\begin{bmatrix} 1 & 1 & 70 \\ 0 & -10 & -500 \end{bmatrix}$$

The last row represents the equation

$$-10y = -500$$

from which $y = 50$. The first row represents the equation

$$x + y = 70$$

and, because $y = 50$, this is

$$x + 50 = 70$$

so $x = 20$. The solution, as noted in Example 63, is $x = 20, y = 50$. ●

EXAMPLE 73

Let's apply Gaussian elimination to the system of 3 equations in 3 unknowns shown here:

$$\begin{aligned} 2x - 3y + z &= -22 \\ 7x + 9y - 3z &= 14 \\ 6x + 7y + 2z &= 91 \end{aligned}$$

The augmented matrix is

$$\begin{bmatrix} 2 & -3 & 1 & -22 \\ 7 & 9 & -3 & 14 \\ 6 & 7 & 2 & 91 \end{bmatrix}$$

A series of elementary row operations, as shown here, will convert the 3×3 matrix of coefficients to upper triangular form. First, multiply row 1 by $1/2$ to produce a 1 in the 1,1 position.

$$1/2 \begin{bmatrix} 2 & -3 & 1 & -22 \\ 7 & 9 & -3 & 14 \\ 6 & 7 & 2 & 91 \end{bmatrix}$$

Then multiply row 1 by -7 and add it to row 2; also multiply row 1 by -6 and add it to row 3.

$$\begin{array}{c} \begin{array}{l} \curvearrowright -7 \\ \curvearrowright -6 \end{array} \begin{bmatrix} 1 & -3/2 & 1/2 & -11 \\ 7 & 9 & -3 & 14 \\ 6 & 7 & 2 & 91 \end{bmatrix} \quad \text{giving} \quad \begin{bmatrix} 1 & -3/2 & 1/2 & -11 \\ & 39/2 & -13/2 & 91 \\ & 16 & -1 & 157 \end{bmatrix} \end{array}$$

Now multiply row 2 by $2/39$ to produce a 1 in the 2,2 position.

$$2/39 \begin{bmatrix} 1 & -3/2 & 1/2 & -11 \\ & 39/2 & -13/2 & 91 \\ & 16 & -1 & 157 \end{bmatrix}$$

Multiply row 2 by -16 and add it to row 3.

$$-16 \begin{bmatrix} 1 & -3/2 & 1/2 & -11 \\ & 1 & -1/3 & 14/3 \\ & 16 & -1 & 157 \end{bmatrix}$$

resulting in

$$\begin{bmatrix} 1 & -3/2 & 1/2 & -11 \\ & 1 & -1/3 & 14/3 \\ & & 13/3 & 247/3 \end{bmatrix}$$

We are almost done. The bottom row represents the equation

$$(13/3)z = 247/3$$

from which $z = 19$. The second row represents the equation

$$y - (1/3)z = 14/3 \quad \text{or} \quad y - (1/3)(19) = 14/3$$

from which $y = 11$. Finally, from the top row,

$$x - (3/2)y + (1/2)z = -11 \quad \text{or} \quad x - (3/2)(11) + (1/2)19 = -11$$

and $x = -4$. So the solution to this system of equations is $x = -4, y = 11, z = 19$. ●

PRACTICE 55 Solve the following system of equations using Gaussian elimination.

$$\begin{aligned} 3x - 5y &= 5 \\ 7x + y &= 37 \end{aligned}$$

Not every system of n linear equations in n unknowns has a solution. As a simple case, consider

$$\begin{aligned} 2x + 4y &= 10 \\ 4x + 8y &= 12 \end{aligned}$$

If we assume that there is a solution and proceed with Gaussian elimination, we perform the following elementary row operations:

$$\frac{1}{2} \begin{bmatrix} 2 & 4 & 10 \\ 4 & 8 & 12 \end{bmatrix} \quad \text{giving} \quad \begin{bmatrix} 1 & 2 & 5 \\ 4 & 8 & 12 \end{bmatrix}$$

Then

$$-4 \begin{array}{c} \curvearrowright \\ \curvearrowleft \end{array} \begin{bmatrix} 1 & 2 & 5 \\ 4 & 8 & 12 \end{bmatrix} \quad \text{results in} \quad \begin{bmatrix} 1 & 2 & 5 \\ 0 & 0 & -8 \end{bmatrix}$$

in which row 2 says that $0 = -8$. This contradiction says that our assumption about the existence of a solution is incorrect. Rewriting the two equations as

$$\begin{aligned} y &= (-1/2)x + 5/2 \\ y &= (-1/2)x + 3/2 \end{aligned}$$

shows that these are two parallel lines that never intersect, so there is no (x, y) pair that satisfies both equations.

More generally, systems of linear equations can involve m equations in n unknowns where n and m are not necessarily equal. Usually, if there are more equations than unknowns, the system is overconstrained and there will be no solution. And usually, if there are more unknowns than equations, there will be an infinite number of solutions.

EXAMPLE 74

Consider the system of equations

$$\begin{aligned} 3x - y - 5z &= 9 \\ x + 2y - 4z &= 10 \end{aligned}$$

In the augmented matrix, switch rows 1 and 2, then multiply row 1 by -3 and add it to row 2:

$$-3 \begin{array}{c} \curvearrowright \\ \curvearrowleft \end{array} \begin{bmatrix} 1 & 2 & -4 & 10 \\ 3 & -1 & -5 & 9 \end{bmatrix} \quad \text{giving} \quad \begin{bmatrix} 1 & 2 & -4 & 10 \\ -7 & 7 & 7 & -21 \end{bmatrix}$$

The last row represents the equation $-7y + 7z = -21$ or $y = z + 3$. Substituting into equation 1, we get

$$x + 2(z + 3) - 4z = 10 \quad \text{or} \quad x = 2z + 4$$

Both x and y have values in terms of a parameter z , which can have any value. The number of solutions is therefore infinite. ●

Despite the simple examples we have seen, neither the solutions to a system of linear equations nor the coefficients are always integers.

Boolean Matrices

In Chapter 6 we will be interested in matrices with only 0s and 1s as entries, called **Boolean matrices** (after George Boole, a nineteenth-century English mathematician; Boole also lent his name to *Boolean algebra*, which we will consider later in this book). Matrix **R** of Example 62 is a Boolean matrix. We can define an operation of Boolean matrix multiplication $\mathbf{A} \times \mathbf{B}$ on Boolean matrices using Boolean multiplication and Boolean addition instead of regular multiplication and addition. These are defined as follows:

$$\text{Boolean multiplication: } x \wedge y = \min(x, y)$$

$$\text{Boolean addition: } x \vee y = \max(x, y)$$

PRACTICE 56 Fill-in the following operation tables for Boolean multiplication and Boolean addition.

x	y	$x \wedge y$
1	1	
1	0	
0	1	
0	0	

x	y	$x \vee y$
1	1	
1	0	
0	1	
0	0	

Now take the tables from Practice 56 and substitute T for 1 and F for 0. They become the truth tables for conjunction and disjunction, respectively; for this reason, these operations are often called **Boolean and** (or **logical and**) and **Boolean or** (or **logical or**), which also explains the notation used for these operations. The operation of **Boolean matrix multiplication** $\mathbf{A} \times \mathbf{B}$ (on Boolean matrices of appropriate dimensions) is then defined by

$$c_{ij} = \bigvee_{k=1}^m (a_{ik} \wedge b_{kj})$$

We can also define two analogues of ordinary matrix addition (on Boolean matrices of the same dimensions): $\mathbf{A} \wedge \mathbf{B}$, where corresponding elements are combined using Boolean multiplication, and $\mathbf{A} \vee \mathbf{B}$, where corresponding elements are combined using Boolean addition.

EXAMPLE 75 Let **A** and **B** be Boolean matrices,

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Then

$$\mathbf{A} \wedge \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{A} \vee \mathbf{B} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

and the Boolean product $\mathbf{A} \times \mathbf{B}$ is

$$\mathbf{A} \times \mathbf{B} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

PRACTICE 57 In Example 75, does $\mathbf{A} \times \mathbf{B} = \mathbf{A} \cdot \mathbf{B}$? ■

PRACTICE 58 In Example 75, compute $\mathbf{B} \times \mathbf{A}$. ■

Solve Millions of Equations, Faster than Gauss

Systems of linear equations are used in many areas of application, including telecommunications, materials analysis, transportation, economics, and medical imaging. Instead of systems of 3 or 4 linear equations with 3 or 4 unknowns, such as we saw in our Example problems, the value of n in such applications can go into the millions or billions. In Exercise 55 of Section 5.7, we see that a worst-case analysis of Gaussian elimination results in $\Theta(n^3)$ computations (multiplications and additions). Such computations on real numbers are called *floating point operations*. The overall implication of an algorithm of $\Theta(n^3)$ is that increasing the size of n by a factor of 10 increases the work by a factor of $10^3 = 1000$.

If n has a value of 1 billion, $n = 10^9$, then $n^3 = 10^{27}$. Suppose we find a way to parallelize the Gaussian elimination algorithm and run it on a fast parallel-processing supercomputer. The Chinese Tianhe-2, the world's fastest supercomputer as of June 2013 can crank out about 33.86 petaflops (33.86×10^{15} floating-point operations per second). The worst-case solution would be bounded above by

$$\frac{10^{27} \text{ operations}}{(33.86) * 10^{15} \text{ operations/second}} = \text{over 936 years!}$$

Now $\Theta(n^3)$ was an upper bound, so maybe the closer value is something like $(2/3)\Theta(n^3)$, but at any rate it's clear that these are time-consuming problems, and faster solution techniques are of interest.

Gaussian elimination produces exact solutions to systems of linear equations. Other solution methods, called iterative solvers, produce a series of approximate solutions that approach the exact solution. This method sounds as if it would take even longer, but most iterative solvers start out with a sparse matrix (a matrix with many 0 entries) for which computations will be

faster but is nonetheless a good representative of the information in the original augmented matrix. Deciding which values can be zeroed out while modifying the values of the remaining coefficients so as to provide a good “preconditioner” for the eventual solution is a difficult problem in itself.

A team of researchers at Carnegie Mellon University announced in 2010 that they had found a new algorithm for creating a good preconditioner for a large system of linear equations of a certain type, called SDD (symmetric and diagonally dominant) systems. In a diagonally dominant matrix, the absolute value of the diagonal element (a_{ii}) of each row of a matrix is bigger than the sum of the absolute values of all other elements in that row. SDD systems turn out to have many important applications, such as maximizing flow through a network (a computer network, a water pipeline system, a transportation system) and recommendation systems such as Netflix that suggest movies you might like based on your past preferences and other user data. The new algorithm, based on sophisticated mathematical techniques, is approximately $\Theta(n(\log n)^2)$, nearly linear, promising much faster solutions to these massive problems.

“A Breakthrough in Algorithm Design,” Kroecker, K., *Communications of the ACM*, September 2011.

“A Fast Solver for a Class of Linear Systems,” Koutis, I., Miller, G. L., and Peng, R., *Communications of the ACM*, October 2012.

“Approaching Optimality for Solving SDD Linear Systems,” Koutis, I., Miller, G. L., and Peng, R., *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, Las Vegas, NV, October 23–26, 2010*.

Linear Equation Breakthrough, Dr. Dobbs, <http://drdobbs.com/architecture-and-design/227900457>

SECTION 5.7 REVIEW

TECHNIQUES

- W Add, subtract, multiply, and perform scalar multiplication on matrices.
 - Solve systems of linear equations using Gaussian elimination.
- W Perform Boolean and, or, and matrix multiplication on Boolean matrices.

MAIN IDEAS

- Matrices are rectangular arrangements of data that are used to represent information in tabular form.
- Matrices have their own arithmetic, with operations of addition, subtraction, multiplication, and scalar multiplication.
- Systems of linear equations can be solved by performing elementary row operations on an augmented matrix (Gaussian elimination).
- Boolean matrices can be manipulated using Boolean operations of and, or, and Boolean multiplication.

EXERCISES 5.7

1. For the matrix:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 0 \\ -4 & 1 \end{bmatrix}$$

What is a_{12} ? What is a_{31} ?

2. Find x and y if

$$\begin{bmatrix} 1 & 3 \\ x & x + y \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 2 & 6 \end{bmatrix}$$

3. Find x , y , z , and w if

$$\begin{bmatrix} x + y & 2x - 3y \\ z - w & z + 2w \end{bmatrix} = \begin{bmatrix} 4 & -7 \\ -6 & 6 \end{bmatrix}$$

4. If \mathbf{A} is a symmetric matrix, find u , v , and w :

$$\mathbf{A} = \begin{bmatrix} 2 & w & u \\ 7 & 0 & v \\ 1 & -3 & 4 \end{bmatrix}$$

For Exercises 5–7, assume the following:

$$r = 3, s = -2,$$

$$\mathbf{A} = \begin{bmatrix} 2 & 1 \\ -1 & 0 \\ 3 & 4 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 4 & 1 & 2 \\ 6 & -1 & 5 \\ 1 & 3 & 2 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 2 & 4 \\ 6 & -1 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 4 & -6 \\ 1 & 3 \\ 2 & -1 \end{bmatrix}$$

5. Compute (if possible)

a. $\mathbf{A} + \mathbf{D}$ b. $\mathbf{A} - \mathbf{D}$ c. $r\mathbf{B}$ d. $s\mathbf{C}$ e. $\mathbf{A} + r\mathbf{D}$

6. Compute (if possible)
- a. $\mathbf{B} - r\mathbf{C}$ c. $r(s\mathbf{C})$ e. $\mathbf{D} \cdot \mathbf{C}$
 b. $r(\mathbf{A} + \mathbf{D})$ d. $\mathbf{B} \cdot \mathbf{A} + \mathbf{D}$
7. Compute (if possible)
- a. $\mathbf{A} \cdot \mathbf{C}$ c. $\mathbf{B} \cdot \mathbf{D}$
 b. $\mathbf{C} \cdot \mathbf{A}$ d. $\mathbf{C}^2 = \mathbf{C} \cdot \mathbf{C}$
8. For

$$\mathbf{A} = \begin{bmatrix} 2 & 4 & 0 \\ 1 & 3 & -1 \\ 3 & -2 & 1 \end{bmatrix}$$

compute $\mathbf{A}^3 = \mathbf{A} \cdot \mathbf{A} \cdot \mathbf{A}$.

9. For

$$\mathbf{A} = \begin{bmatrix} 3 & -1 \\ 2 & 5 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 4 & 1 \\ 2 & -1 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 6 & -5 \\ 2 & -2 \end{bmatrix}$$

compute (if possible)

- a. $\mathbf{A} \cdot \mathbf{B}$ and $\mathbf{B} \cdot \mathbf{A}$
 b. $\mathbf{A} \cdot (\mathbf{B} \cdot \mathbf{C})$ and $(\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{C}$
 c. $\mathbf{A} \cdot (\mathbf{B} + \mathbf{C})$ and $\mathbf{A} \cdot \mathbf{B} + \mathbf{A} \cdot \mathbf{C}$
 d. $(\mathbf{A} + \mathbf{B}) \cdot \mathbf{C}$ and $\mathbf{A} \cdot \mathbf{C} + \mathbf{B} \cdot \mathbf{C}$
10. If

$$\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 4 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} x & 3 \\ y & 2 \end{bmatrix}$$

find x and y if $\mathbf{A} \cdot \mathbf{B} = \mathbf{B} \cdot \mathbf{A}$.

11. Prove that matrix multiplication is associative; that is, prove that if \mathbf{A} is an $n \times p$ matrix, \mathbf{B} is a $p \times r$ matrix and \mathbf{C} is an $r \times m$ matrix, then $\mathbf{A} \cdot (\mathbf{B} \cdot \mathbf{C}) = (\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{C}$.
12. a. Prove that $\mathbf{I}^2 = \mathbf{I}$ for any identity matrix \mathbf{I} .
 b. Prove that $\mathbf{I}^n = \mathbf{I}$ for any identity matrix \mathbf{I} and any positive integer n .
13. Let \mathbf{A} and \mathbf{B} be $n \times n$ matrices.
- a. Prove that if \mathbf{A} has one row consisting of all 0's, then so does $\mathbf{A} \cdot \mathbf{B}$.
 b. Prove that if \mathbf{B} has one column consisting of all 0's, then so does $\mathbf{A} \cdot \mathbf{B}$.

14. An $n \times n$ matrix \mathbf{A} is *diagonal* if all elements a_{ij} with $i \neq j$ are 0. For example, \mathbf{A} below is a 3×3 diagonal matrix.

$$\mathbf{A} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & -7 \end{bmatrix}$$

- Prove that if \mathbf{A} and \mathbf{B} are $n \times n$ diagonal matrices, then $\mathbf{A} + \mathbf{B}$ is diagonal.
 - Prove that if \mathbf{A} is an $n \times n$ diagonal matrix and r is a scalar, then $r\mathbf{A}$ is diagonal.
 - Prove that if \mathbf{A} and \mathbf{B} are $n \times n$ diagonal matrices, then $\mathbf{A} \cdot \mathbf{B}$ is diagonal.
15. The *transpose* of a matrix \mathbf{A} , \mathbf{A}^T , is obtained by interchanging its rows and columns. Thus, if we denote the element in row i , column j of \mathbf{A} by $\mathbf{A}(i, j)$, then $\mathbf{A}^T(i, j) = \mathbf{A}(j, i)$.
- Find \mathbf{A}^T for

$$\mathbf{A} = \begin{bmatrix} 1 & 3 & 4 \\ 6 & -2 & 1 \end{bmatrix}$$

- Prove that if \mathbf{A} is a square matrix, then \mathbf{A} is symmetric if and only if $\mathbf{A}^T = \mathbf{A}$.
 - Prove that $(\mathbf{A}^T)^T = \mathbf{A}$.
 - Prove that $(\mathbf{A} + \mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$.
 - Prove that $(\mathbf{A} \cdot \mathbf{B})^T = \mathbf{B}^T \cdot \mathbf{A}^T$.
16. Prove that $\mathbf{A} \cdot \mathbf{A}^T$ is symmetric for any matrix \mathbf{A} (see Exercise 15).
17. Find two 2×2 matrices \mathbf{A} and \mathbf{B} such that $\mathbf{A} \cdot \mathbf{B} = \mathbf{0}$ but $\mathbf{A} \neq \mathbf{0}$ and $\mathbf{B} \neq \mathbf{0}$.
18. Find three 2×2 matrices \mathbf{A} , \mathbf{B} , and \mathbf{C} such that $\mathbf{A} \cdot \mathbf{C} = \mathbf{B} \cdot \mathbf{C}$, $\mathbf{C} \neq \mathbf{0}$, but $\mathbf{A} \neq \mathbf{B}$.
19. If \mathbf{A} and \mathbf{B} are $n \times n$ matrices, is it always true that $(\mathbf{A} + \mathbf{B})^2 = \mathbf{A}^2 + 2(\mathbf{A} \cdot \mathbf{B}) + \mathbf{B}^2$? Will it ever be true?
20. The vector of real numbers $\mathbf{U} = [u_1 \ u_2]$ can be visualized on the real-number plane as an arrow from the origin to the point (u_1, u_2) . The length of the arrow, also called the *magnitude* of the vector, is given by $\|\mathbf{U}\| = \sqrt{u_1^2 + u_2^2}$. The *dot product* of two such vectors, $\mathbf{U} \cdot \mathbf{V}$, is defined to be the real number $u_1v_1 + u_2v_2$. Show that if θ is the angle between \mathbf{U} and \mathbf{V} , $0 \leq \theta \leq \pi$, then

$$\cos \theta = \frac{\mathbf{U} \cdot \mathbf{V}}{\|\mathbf{U}\| \cdot \|\mathbf{V}\|}$$

(Hint: Use the law of cosines.)

- Prove that if a square matrix \mathbf{A} is symmetric, then so is \mathbf{A}^2 , where $\mathbf{A}^2 = \mathbf{A} \cdot \mathbf{A}$.
- Prove that if a square matrix \mathbf{A} is symmetric, then so is \mathbf{A}^{2^n} for any integer $n \geq 1$.

23. Let

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

For $n \geq 1$, let $F(n)$ equal the n th value in the Fibonacci sequence (see Example 2 in Chapter 3); let $F(0) = 0$. Prove that for any $n \geq 1$, \mathbf{A}^n is given by

$$\begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix}$$

24. a. Show that for

$$\mathbf{A} = \begin{bmatrix} 1 & 3 \\ 2 & 2 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} -1/2 & 3/4 \\ 1/2 & -1/4 \end{bmatrix}$$

$$\mathbf{A} \cdot \mathbf{B} = \mathbf{B} \cdot \mathbf{A} = \mathbf{I}, \text{ so } \mathbf{B} = \mathbf{A}^{-1}.$$

b. Show that

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$$

is not invertible.

c. Show that

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

is invertible with inverse

$$\mathbf{B} = \frac{1}{a_{11}a_{22} - a_{12}a_{21}} \begin{bmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{bmatrix}$$

if and only if $a_{11}a_{22} - a_{12}a_{21} \neq 0$.

25. Prove that if \mathbf{A} is invertible and r is a non-zero scalar, then $r\mathbf{A}$ is invertible with

$$(r\mathbf{A})^{-1} = (1/r)\mathbf{A}^{-1}.$$

26. Prove that if \mathbf{A} is invertible and $\mathbf{A} \cdot \mathbf{B} = \mathbf{A} \cdot \mathbf{C}$, then $\mathbf{B} = \mathbf{C}$.

For Exercises 27–34, use Gaussian elimination to solve the systems of equations, if possible.

27. $x + 5y = 1$

$2x - 3y = 15$

28. $x + 5y = 38.7$

$4x - 2y = -1.4$

29. $-x + 2y + z = -1$
 $3x - 5y - z = 5$
 $2x - y + 3z = 22$
30. $x - y + z = 6$
 $x + 2y - 3z = 10$
 $2x + 3y + 5z = 12$
31. $x + 2y - z = -1$
 $x - 3y + z = 2$
 $2x + y + 2z = 6$
32. $2x - 7y + z + 2w = 5$
 $x + y - 2z + 3w = 8$
 $4x + 2y + z - 4w = 12$
 $5x + 3y - z - w = 10$
33. $x + 2y - z + w = -3$
 $2x - y + 4z + 2w = 33$
 $x - y + 3z - 7w = 6$
 $-3x + 3y + z + 4w = -12$
34. $x - 2y + 3z - w = 7$
 $2x + 5y - 7z + 2w = 12$
 $4x - 3y + 12z + w = 8$
35. Find an example of a system of 3 linear equations with 2 unknowns that has a solution. Explain what happens when you use Gaussian elimination on this system.
36. Find an example of a system of 4 linear equations with 3 unknowns that has a solution. Explain what happens when you use Gaussian elimination on this system.
37. You purchase an ancient Egyptian medallion at the State Fair from a vendor who swears it is pure gold. The medallion weighs 859.4 grams and its volume in cubic centimeters is 52. You suspect that the medallion is actually a mixture of copper and gold. You know that copper weighs 9 grams per cubic centimeter and that gold weighs 19.3 grams per cubic centimeter. Set up and solve a system of equations to find the percentage of copper by volume in the medallion.
38. Cell phone Plan A charges a flat monthly fee of \$30.00 for the first 400 minutes, plus \$0.07 for each minute > 400 . Plan B charges a flat monthly fee of \$45.00 for the first 600 minutes, plus \$0.19 for each minute > 600 . You know you will use more than 600 minutes per month.
- At what number of minutes do the plans cost the same amount per month, and what is that amount?
 - Above this number of minutes, which plan is more expensive?
39. If \mathbf{A} is an $n \times n$ invertible matrix, the following method can be used to find \mathbf{A}^{-1} .
- Operate on \mathbf{A} using any combination of the two following elementary row operations until the resulting matrix is the $n \times n$ identity matrix \mathbf{I} .
 - Multiply all the elements in any one row of \mathbf{A} by a non-zero scalar.
 - Add a scalar multiple of any row to any other row.
 - At the same time, perform exactly the same sequence of operations on the $n \times n$ identity matrix \mathbf{I} .
 - The matrix that results from \mathbf{I} after step 2 is \mathbf{A}^{-1} .
- Use this method to find the inverse of matrix \mathbf{A} in Exercise 24(a).

40. Use the method of Exercise 39 to find the inverse of matrix \mathbf{A} in Example 71.
 41. Consider a system of n linear equations in n unknowns, such as the one from Example 63:

$$\begin{aligned}x + y &= 70 \\24x + 14y &= 1180\end{aligned}$$

If

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 24 & 14 \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} x \\ y \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 70 \\ 1180 \end{bmatrix}$$

then the system of equations can be represented in matrix form by

$$\mathbf{A} \cdot \mathbf{X} = \mathbf{B}$$

If \mathbf{A} , the matrix of coefficients, is invertible, then we can multiply both sides of the above equation by \mathbf{A}^{-1} , giving

$$\begin{aligned}\mathbf{A}^{-1} \cdot (\mathbf{A} \cdot \mathbf{X}) &= \mathbf{A}^{-1} \cdot \mathbf{B} \\(\mathbf{A}^{-1} \cdot \mathbf{A}) \cdot \mathbf{X} &= \mathbf{A}^{-1} \cdot \mathbf{B} && \text{(matrix multiplication is associative)} \\ \mathbf{I} \cdot \mathbf{X} &= \mathbf{A}^{-1} \cdot \mathbf{B} && \text{(definition of } \mathbf{A}^{-1} \text{)} \\ \mathbf{X} &= \mathbf{A}^{-1} \cdot \mathbf{B} && \text{(definition of } \mathbf{I} \text{)}\end{aligned}$$

Therefore the solution to the system of equations is given by

$$\mathbf{X} = \mathbf{A}^{-1} \cdot \mathbf{B}$$

Make use of Exercise 39 to find \mathbf{A}^{-1} , and use this approach to solve the system of equations.

In Exercises 42–46, solve the systems of equations using the method of Exercise 41.

42. $x + 2y = -4$
 $x + y = 5$

43. The system of Exercise 27

44. The system of Exercise 28

45. The system of Exercise 29

46. The system of Exercise 30

47. For Boolean matrices

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

find $\mathbf{A} \wedge \mathbf{B}$, $\mathbf{A} \vee \mathbf{B}$, $\mathbf{A} \times \mathbf{B}$, and $\mathbf{B} \times \mathbf{A}$.

48. For Boolean matrices

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

find $\mathbf{A} \wedge \mathbf{B}$, $\mathbf{A} \vee \mathbf{B}$, $\mathbf{A} \times \mathbf{B}$, and $\mathbf{B} \times \mathbf{A}$.

49. For Boolean matrices

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

find $\mathbf{A} \wedge \mathbf{B}$, $\mathbf{A} \vee \mathbf{B}$, $\mathbf{A} \times \mathbf{B}$, and $\mathbf{B} \times \mathbf{A}$.

50. For Boolean matrices

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

find $\mathbf{A} \wedge \mathbf{B}$, $\mathbf{A} \vee \mathbf{B}$, $\mathbf{A} \times \mathbf{B}$, and $\mathbf{B} \times \mathbf{A}$.

51. For Boolean matrices \mathbf{A} and \mathbf{B} , can it ever be the case that $\mathbf{A} \vee \mathbf{B} = \mathbf{A} \wedge \mathbf{B}$? If so, when?

52. For Boolean matrices \mathbf{A} and \mathbf{B} , prove that $\mathbf{A} \vee \mathbf{B} = \mathbf{B} \vee \mathbf{A}$ and that $\mathbf{A} \wedge \mathbf{B} = \mathbf{B} \wedge \mathbf{A}$.

53. How many distinct symmetric $n \times n$ Boolean matrices are there?

54. *Strassen's algorithm* reduces the amount of work to compute the product of two $n \times n$ matrices of sufficient size. For simplicity, assume that $n = 2^m$ for some $m \geq 0$. First consider a simple case of multiplying two 2×2 matrices.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

The product

$$\mathbf{C} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix} \quad (1)$$

can also be written (check the calculations) as

$$\mathbf{C} = \begin{bmatrix} p_1 + p_4 - p_5 + p_7 & p_3 + p_5 \\ p_2 + p_4 & p_1 + p_3 - p_2 + p_6 \end{bmatrix} \quad (2)$$

where

$$\begin{aligned} p_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) & p_2 &= (a_{21} + a_{22})b_{11} \\ p_3 &= a_{11}(b_{12} - b_{22}) & p_4 &= a_{22}(b_{21} - b_{11}) \\ p_5 &= (a_{11} + a_{12})b_{22} & p_6 &= (a_{21} - a_{11})(b_{11} + b_{12}) \\ p_7 &= (a_{12} - a_{22})(b_{21} + b_{22}) \end{aligned}$$

Computing the various p_i quantities requires 7 multiplications and 10 additions (counting subtractions as additions). Computing the product \mathbf{C} once the p_i quantities exist takes 0 multiplications and an additional 8 additions. In total, computing \mathbf{C} by this method requires 7 multiplications and 18 additions. Now take two $n \times n$ matrices \mathbf{A} and \mathbf{B} and partition each of them into four $(n/2 \times n/2)$ matrices:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix}$$

The product $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$ is still given by Equation (1) using \mathbf{A}_{ij} and \mathbf{B}_{ij} instead of a_{ij} and b_{ij} , respectively. Therefore Equation (2) still holds, and the product \mathbf{C} requires 7 $(n/2 \times n/2)$ matrix multiplications. This is an example of a divide and conquer algorithm, where the work has been reduced to several instances of the same problem on a significantly reduced input size (although there is an additional overhead of 18 $(n/2 \times n/2)$ matrix additions).

Let $M(n)$ represent the number of multiplications required for a product of two $n \times n$ matrices. Using Strassen's algorithm, we can write

$$\begin{aligned} M(1) &= 1 && \text{(one multiplication in the product of two } 1 \times 1 \text{ matrices)} \\ M(n) &= 7M\left(\frac{n}{2}\right) \end{aligned}$$

- Solve this recurrence relation for $M(n)$.
- If $A(n)$ represents the number of additions required for a product of two $n \times n$ matrices, justify the following recurrence relation:

$$\begin{aligned} A(1) &= 0 \\ A(n) &= 7A\left(\frac{n}{2}\right) + 18\left(\frac{n}{2}\right)^2 \end{aligned}$$

- Solve this recurrence relation to obtain an order-of-magnitude expression for $A(n)$.
 - Find an approximate order-of-magnitude value for the total work (additions and multiplications) for matrix multiplication using Strassen's algorithm, and compare it with the $\Theta(n^3)$ work (additions and multiplications) for traditional matrix multiplication. (Although Strassen's algorithm is theoretically an improvement for all values of n , the constants that are ignored in an order-of-magnitude argument mean that in actual implementation the traditional algorithm may run faster for values of n less than about $2^6 = 64$).
55. The Gaussian elimination algorithm is labor-intensive. A worst-case analysis can be done simply by counting, where multiplications and additions are the units of work (divisions are counted as multiplications and subtractions are counted as additions). Consider a system of n linear equations in n unknowns.
- In the worst-case, the first non-zero element of each row of the augmented matrix as it is being transformed into upper triangular form is not 1 and that row must be multiplied by a non-zero scalar to create a 1. Show that this requires a total of

$$\frac{(n+1)(n+2)}{2} - 3$$

multiplications. (*Hint:* Consider Practice 7 in Chapter 2.)

- Aside from the multiplications required in part (a), show that $\frac{2n^3 + 3n^2 - 5n}{6}$ multiplications and $\frac{2n^3 + 3n^2 - 5n}{6}$ additions are required to transform the augmented matrix into upper triangular form.

(*Hint:* Consider Exercise 11 in Section 2.2.)

- c. After the matrix has been reduced to upper triangular form, show that there are $\frac{n(n+1)}{2}$ multiplications and $\frac{(n-1)n}{2}$ additions to solve for the n unknowns.
- d. Explain why Gaussian elimination is $\Theta(n^3)$ in the worst case.
56. DES, discussed in Section 5.6, is an encryption algorithm that is an example of a block cipher, where a block of bits is encoded into a block of bits. Matrices can be used to create a simple block cipher. Consider a 2×2 matrix with integer entries, for example,

$$\mathbf{A} = \begin{bmatrix} 2 & 7 \\ 1 & 4 \end{bmatrix}$$

\mathbf{A} is an invertible matrix with

$$\mathbf{A}^{-1} = \begin{bmatrix} 4 & -7 \\ -1 & 2 \end{bmatrix}$$

because

$$\begin{bmatrix} 2 & 7 \\ 1 & 4 \end{bmatrix} \cdot \begin{bmatrix} 4 & -7 \\ -1 & 2 \end{bmatrix} = \begin{bmatrix} 4 & -7 \\ -1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 2 & 7 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Break up the message to be encrypted into blocks of two characters, and apply a function mapping the letters of the alphabet into the integers 0–25 as follows:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Thus $[B \ R] \rightarrow [1 \ 17]$. The heart of the encryption algorithm consists of multiplying the resulting 1×2 matrix by \mathbf{A} using arithmetic modulo 26. Thus

$$[1 \ 17] \cdot \begin{bmatrix} 2 & 7 \\ 1 & 4 \end{bmatrix} = [19 \ 75] \rightarrow [19 \ 23]$$

and $[19 \ 23] \rightarrow [T \ X]$. Therefore $[B \ R]$ is encrypted as $[T \ X]$. To decrypt, convert $[T \ X]$ back to $[19 \ 23]$ and multiply the resulting 1×2 matrix by \mathbf{A}^{-1} , again using modulo 26 arithmetic.

$$[19 \ 23] \cdot \begin{bmatrix} 4 & -7 \\ -1 & 2 \end{bmatrix} = [53 \ -87] \rightarrow [1 \ 17]$$

to be converted back to the original message $[B \ R]$.

- Using the encryption matrix \mathbf{A} above, encrypt the block $[V \ I]$.
- Decrypt the result from part (a) to recover $[V \ I]$.
- Explain why the decoding process recovers the original (numerical) block.

CHAPTER 5 REVIEW

TERMINOLOGY

- ABA routing number (p. 437)
 add to a database (p. 375)
 addition modulo n (p. 343, p. 438)
 addition of matrices (p. 449)
 AES (Advanced Encryption Standard) (p. 431)
 antisymmetric relation (p. 332)
 array (p. 448)
 asymmetric encryption (p. 431)
 augmented matrix (p. 453)
 big oh (p. 417)
 big theta (p. 413)
 bijection (p. 390)
 binary relation from S to T (p. 330)
 binary relation on a set S (p. 329)
 binary relation on $S \times T$ (p. 330)
 blind key (p. 368)
 block (p. 339)
 block cipher (p. 429)
 Boolean and (logical and) (p. 458)
 Boolean matrix (p. 458)
 Boolean matrix multiplication ($A \times B$) (p. 458)
 Boolean or (logical or) (p. 458)
 Caesar cipher (p. 428)
 Cantor's theorem (p. 401)
 cardinality of a relation (p. 366)
 cardinality of a set (p. 401)
 ceiling function (p. 386)
 chain (p. 338)
 chaining (p. 426)
 ciphertext (p. 428)
 circular left shift (p. 430)
 closure of a relation (p. 334)
 codomain (p. 383)
 collision (p. 425)
 commutative diagram (p. 391)
 composition function (p. 391)
 composite primary key (p. 367)
 conceptual model (p. 365)
 congruence modulo 4 (p. 343)
 congruence modulo n (p. 343)
 critical path (p. 359)
 cryptographic hash function (p. 434)
 cryptography (p. 428)
 cryptology (p. 428)
 data integrity (p. 375)
 database (p. 365)
 degree of relation (p. 366)
 delete from a database (p. 375)
 derangement (p. 396)
 DES (Data Encryption Standard) (p. 429)
 diffusion (p. 429)
 dimensions of a matrix (p. 446)
 disjoint cycles (p. 395)
 domain (p. 383)
 elementary row operations (p. 454)
 entities (p. 365)
 entity integrity (pp. 367, 375)
 entity-relationship diagram (E-R diagram) (p. 365)
 entity-relationship model (p. 365)
 equal functions (p. 387)
 equal matrices (p. 447)
 equivalence class (p. 339)
 equivalence relation (p. 339)
 equivalent sets (p. 401)
 floor function (p. 386)
 foreign key (p. 368)
 function (p. 383)
 Gaussian elimination (p. 454)
 greatest element (p. 338)
 hash function (p. 424)
 hash table (p. 425)
 Hasse diagram (p. 336)
 identity function (p. 392)
 identity matrix (p. 451)
 identity permutation (p. 396)
 image (p. 383)
 immediate predecessor in a partial ordering (p. 336)
 intractable problem (p. 417)
 inverse function (p. 393)
 inverse of a matrix (p. 452)
 invertible matrix (p. 452)
 International Standard Book Number (ISBN) (p. 435)
 join (p. 370)
 least element (p. 338)
 linear equation (p. 447)
 linear probing (p. 425)
 little oh (p. 417)
 load factor (p. 427)
 main diagonal (p. 447)
 many-to-many relation (p. 330)
 many-to-one relation (p. 330)
 mapping (p. 383)
 master theorem (p. 418)
 matrix (p. 446)
 matrix of coefficients (p. 447)
 maximal element (p. 338)
 metadata (p. 366)
 minimal element (p. 338)
 modify a database (p. 375)
 modulo function (p. 386)
 multiplication of matrices (p. 450)
 n -ary relation on $S_1 \times S_2 \times \cdots \times S_n$ (p. 330)
 node (p. 337)
 one-to-many relation (p. 330)
 one-to-one (injective) function (p. 389)
 one-to-one relation (p. 330)
 one-way encryption (p. 434)
 onto (surjective) function (p. 388)
 order of magnitude (p. 413)
 partial ordering (p. 336)
 partially ordered set (poset) (p. 336)
 partition (p. 339)
 permutation function (p. 394)
 PERT chart (p. 357)
 plaintext (p. 428)
 poset (p. 336)
 predecessor in a partial ordering (p. 336)
 preimage (p. 383)
 primary key (p. 367)
 private key encryption (p. 431)
 project (p. 369)
 public key encryption (p. 431)
 range (p. 388)
 referential integrity (p. 375)
 reflexive closure (p. 334)
 reflexive relation (p. 332)
 relational algebra (p. 372)

relational calculus (p. 372)	SELECT (p. 372)	topological sorting (p. 359)
relational database (p. 366)	simple substitution cipher (p. 429)	total ordering (p. 338)
relational model (p. 366)	SQL (p. 372)	transitive closure (p. 334)
residue of x modulo n (p. 423)	subtraction of matrices (p. 449)	transitive relation (p. 332)
restrict (p. 369)	successor in a partial ordering (p. 336)	tuple (p. 366)
restriction of a partial ordering (p. 336)	surrogate key (p. 368)	UPC-A (Universal Product Code) (p. 436)
RSA public key encryption algorithm (p. 431)	symmetric closure (p. 334)	upper triangular matrix (p. 453)
scalar (p. 448)	symmetric encryption (p. 431)	vector (p. 448)
scalar multiplication (p. 448)	symmetric matrix (p. 447)	vertex (p. 337)
	symmetric relation (p. 332)	zero matrix (p. 449)

SELF-TEST

Answer the following true-false questions.

Section 5.1

1. In a one-to-many binary relation, at least one first component must appear in two different ordered pairs.
2. If an antisymmetric binary relation contains (x, y) , then (y, x) will not belong to the relation.
3. A least element of a partially ordered set precedes all elements except itself.
4. An equivalence relation cannot also be a partial ordering.
5. A partial ordering on a set determines a partition of that set.

Section 5.2

1. If a task is not on the critical path in a PERT chart, then that task is optional.
2. A topological sort turns a partially ordered set into a totally ordered set.
3. If x precedes y after a topological sort on a finite partially ordered set, then x preceded y in the original partial ordering.
4. The times to complete parallel tasks are added together in determining a critical path in a PERT chart.
5. A given set of data results in a unique topological sort.

Section 5.3

1. A relation in a relational database is a set of n tuples of attribute values.
2. A primary key in a relation is a minimum subset of attribute values that will uniquely identify each tuple.
3. The restrict operation can be achieved by doing a union followed by an intersection.
4. The join operation can be achieved by doing a Cartesian product followed by a restrict.

5. Deleting a tuple from a relation may result in additional deletions being done in order to satisfy data integrity.

Section 5.4

1. A binary relation on $S \times T$ that is not one-to-many or many-to-many is a function from S to T .
2. To prove that a function is onto, begin with an arbitrary element of the range and show that it has a preimage.
3. To prove that a function is one-to-one, assume $f(s_1) = f(s_2)$ for some s_1 and s_2 in the domain and show that $s_1 = s_2$.
4. The composition of two permutation functions on a set is a permutation function on the set.
5. Any one-to-one function has an inverse function.

Section 5.5

1. If f is $\Theta(g)$, then beyond some point the values for $f(x)$ must fall between $\frac{1}{2}g(x)$ and $2g(x)$.
2. If $f = O(g)$, then either $f = \Theta(g)$ or $g = o(f)$.
3. If $f(x) = 3x^2 + 15x - 2$ and $g(x) = 5000x^3/(x - 1)$ then $f = \Theta(g)$.
4. An intractable problem is one that only has solution algorithms of the form $\Theta(n^c)$ where $c \geq 5$.
5. In a recurrence relation of the form

$$S(n) = aS\left(\frac{n}{b}\right) + n^c \quad \text{for } n \geq 2$$

the order of magnitude of the solution, as determined by the master theorem, depends on the ratio of a to b^c .

Section 5.6

1. $37 \equiv 15 \pmod{11}$
2. When searching a hash table for a particular target value, first apply the hash function to the value to obtain the hash table index, then examine the table entry at that index. If the table entry stored there matches the target value, the search was successful; if not, the search fails.
3. A Caesar cipher with a shift $k = 5$ will encode “W” as “B.”
4. In the RSA algorithm, the public key is (n, e) . Security derives from the difficulty of computing $\varphi(n)$.
5. The check digit in the ISBN-10 0-321-18059-3 is correct.

ON THE COMPUTER

For Exercises 1–17, write a computer program that produces the desired output from the given input.

1. *Input:* The elements in a finite set S and a list of ordered pairs representing a binary relation on S
Output: Statement indicating whether the relation is one-to-one, one-to-many, many-to-one, or many-to-many
2. *Input:* The elements in a finite set S and two lists of ordered pairs representing two binary relations on S
Output: The ordered pairs in the union and in the intersection of the two relations, and the ordered pairs in the complements of each relation
3. *Input:* The elements in a finite set S and a list of ordered pairs representing a binary relation on S
Output: Statement of which properties—reflexive, symmetric, transitive, and/or antisymmetric—the relation has
4. *Input:* The elements in a finite set S and a list of ordered pairs representing a binary relation on S
Output: Reflexive, symmetric, and transitive closures of the relation
5. *Input:* The elements in a finite set S and a list of ordered pairs representing a partial ordering on S
Output: A list of all minimal and maximal elements

Section 5.7

1. Two matrices that do not have the same dimensions cannot be added.
2. If \mathbf{A} and \mathbf{B} are square matrices, then $\mathbf{A} \cdot \mathbf{B} = \mathbf{B} \cdot \mathbf{A}$.
3. The usual algorithm for matrix multiplication is $\Theta(n^3)$.
4. The process of Gaussian elimination reduces the augmented matrix of a system of n linear equations in n unknowns to the $n \times n$ identity matrix.
5. If \mathbf{A} and \mathbf{B} are square Boolean matrices, then $\mathbf{A} \times \mathbf{B} = \mathbf{B} \times \mathbf{A}$, where $\mathbf{A} \times \mathbf{B}$ denotes the Boolean product.

6. *Input:* The elements in a finite set S and a list of ordered pairs representing a partial ordering on S
Output: A list of any least or greatest elements. Note that this task is more difficult than that in Exercise 5.
7. *Input:* The elements in a finite set S , a list of ordered pairs representing an equivalence relation on S , and an element x of S
Output: The members of $[x]$
8. *Input:* Array representations of relation tables and appropriate input data for restrict, project, and join operations
Output: Array representations of the resulting relation tables
9. *Input:* The elements in a finite set S and a list of ordered pairs representing a partial ordering on S
Output: Sequence representing the total ordering that results from doing a topological sort (Hint: Reuse some of your code from Exercise 5.)
10. *Input:* The elements in a finite set S and in a finite set T , and a list of ordered pairs representing a binary relation on $S \times T$
Output: An indication of whether the relation is a function from S to T and if so, whether it is onto or one-to-one or both

- 11. Input:** The number of elements in two finite sets S and T
Output: The number of functions from S to T , the number of one-to-one functions from S to T (or an indication that none exist), and the number of onto functions from S to T (or an indication that none exist)
- 12. Input:** Two lists of ordered pairs representing functions f and g from S to S
Output: List of ordered pairs representing the composition function $g \circ f$
- 13. Input:** The elements in a finite set S and two lists that represent (in cycle form) permutations f and g on S
Output: One or more lists that represent the composition function $g \circ f$ in cycle or product-of-cycle form
- 14. Input:** The number of elements in a finite set S
Output: The number of derangements on S
- 15. Input:** n and the entries in two $n \times n$ matrices \mathbf{A} and \mathbf{B}
Output: Sum $\mathbf{A} + \mathbf{B}$ and products $\mathbf{A} \cdot \mathbf{B}$ and $\mathbf{B} \cdot \mathbf{A}$
- 16. Input:** Dimensions of a matrix \mathbf{A} and the entries in \mathbf{A}
Output: \mathbf{A}^T (see Exercise 15 in Section 5.7)
- 17. Input:** The augmented matrix of a system of n linear equations in n unknowns with a unique set of n solutions
Output: The n solutions, determined by using Gaussian elimination
- 18.** The **determinant** of an $n \times n$ matrix can be used in solving systems of linear equations, as well as for other purposes. The determinant of \mathbf{A} can be defined in terms of minors and cofactors. The **minor** of element a_{ij} is the determinant of the $(n-1) \times (n-1)$ matrix obtained from \mathbf{A} by crossing out the elements in row i and column j ; denote this minor by M_{ij} . The **cofactor** of element a_{ij} , denoted by C_{ij} , is defined by

$$C_{ij} = (-1)^{i+j} M_{ij}$$

The determinant of \mathbf{A} is computed by multiplying all the elements in some fixed row of \mathbf{A} by their respective cofactors and summing the results. For example, if the first row is used, then the determinant of \mathbf{A} is given by

$$\sum_{k=1}^n (a_{1k})(C_{1k})$$

Write a program that, when given n and the entries in an $n \times n$ array \mathbf{A} as input, computes the determinant of \mathbf{A} . Use a recursive algorithm.

This page intentionally left blank

Graphs and Trees

CHAPTER OBJECTIVES

After studying this chapter, you will be able to:

- Understand and use the many terms associated with graphs, directed graphs, and trees.
- Appreciate the use of graphs, directed graphs, and trees as representation tools in a wide variety of contexts.
- Prove that two given graphs are isomorphic or give a reason why they are not.
- Use Euler's formula for a simple, connected, planar graph.
- Understand the role of the two specific graphs K_5 and $K_{3,3}$ in graph planarity.
- Prove elementary properties about graphs and trees.
- Use adjacency matrix and adjacency list representations for graphs and directed graphs.
- Do preorder, inorder, and postorder tree traversal.
- Use array and pointer representations for binary trees.
- Use decision trees to represent the steps a searching or sorting algorithm carries out.
- Build a binary search tree and conduct a binary tree search.
- Express lower bounds on the worst-case number of comparisons for searching or sorting on a list with n elements.
- Find Huffman codes for characters whose frequency of occurrence is given.

You work in the Information Systems Department at World Wide Widgets (WWW), the leading widget manufacturer. Widgets are extremely complex devices made up of an enormous number of very simple parts. Each part is one of the following types: Bolt (B), Component (C), Gear (G), Rod (R), or Screw (S). There are many different variations of each basic type. Part numbers consist of a leading character B, C, G, R, or S to identify the part type, followed by an 8-digit number. Thus

C00347289

B11872432

S45003781

are all legitimate part numbers. Using the multiplication principle, there are 5×10^8 different potential part numbers! WWW maintains a data file of the part numbers it uses, which, as it turns out, is most of the potential numbers. Most computers,

including those at WWW, use the ASCII encoding scheme for converting characters into binary form, under which each character requires 1 byte (8 bits) of storage. Because each different part number consists of 9 characters, the WWW parts data file is approximately $9 \times 5 \times 10^8$ bytes, or 4.5 Gb.

Question: How can you compress this data file so that it takes less storage space?

One answer to this question involves working with binary tree structures. A tree is a visual representation of data items and the connections between some of these items. It is a special case of a more general structure called a graph. Graphs or trees can be used to represent a surprising number of real-world situations—organization charts, road maps, transportation and communications networks, and so forth. Later we will see other uses of graphs and trees to represent logic networks, finite-state machines, and formal-language derivations.

Graph theory is an extensive topic. Sections 6.1 and 6.2 present some of the considerable terminology connected with graphs and trees and some elementary results about these structures. To represent a graph or a tree in computer memory, data must be arranged in a way that preserves all the information contained in the visual representation. Several approaches to representing graphs and trees within a computer are discussed.

Decision trees are graphical representations of the activities of certain types of algorithms. In Section 6.3, decision trees are presented and used to find lower bounds on the worst-case behavior of searching and sorting algorithms. In Section 6.4, an algorithm is given for constructing binary trees that allow for data compression of large files.

SECTION 6.1 GRAPHS AND THEIR REPRESENTATIONS

Definitions of a Graph

One way to while away the hours on an airplane trip is to look at the literature in the seat pockets. This material almost always includes a map showing the routes of the airline you are flying, such as the one in Figure 6.1. All this route information could be expressed in paragraph form; for example, there is a direct route between Chicago and Nashville but not between St. Louis and Nashville. However, the paragraph would be rather long and involved, and we would not be able to assimilate the information as quickly and clearly as we can from the map. There are many cases where “a picture is worth a thousand words.”

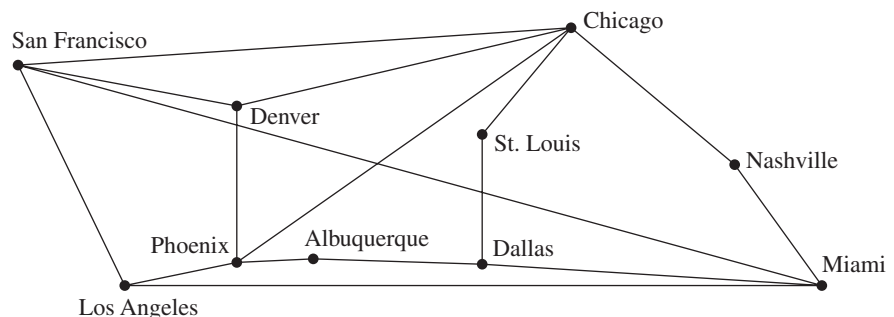


Figure 6.1

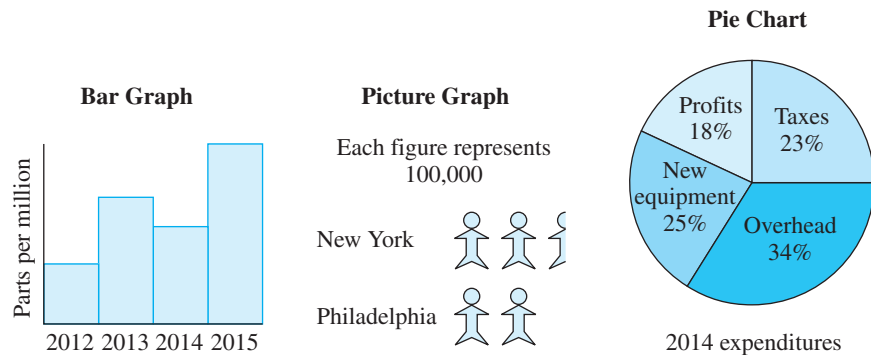


Figure 6.2

The term “graph” is often used informally for any visual representation of data, such as that in Figure 6.1; other forms include the bar graph, picture graph, and pie chart, which are shown in Figure 6.2. We have also talked about graphs of functions on rectangular coordinate systems. We will use two definitions of a graph; one relies on a visual representation like that of Figure 6.1, and the other is a more formal definition that actually says nothing about a visual representation.

DEFINITION (Informal) GRAPH A **graph** is a nonempty set of **nodes (vertices)** and a set of **arcs (edges)** such that each arc connects two nodes.

Our graphs will always have a finite number of nodes and arcs.

EXAMPLE 1

The set of nodes in the airline map of Figure 6.1 is {Chicago, Nashville, Miami, Dallas, St. Louis, Albuquerque, Phoenix, Denver, San Francisco, Los Angeles}. There are 16 arcs; Phoenix-Albuquerque is an arc (here we are naming an arc by the nodes it connects), Albuquerque-Dallas is an arc, and so on.

EXAMPLE 2

In the graph of Figure 6.3, there are five nodes and six arcs. Arc a_1 connects nodes 1 and 2, arc a_3 connects node 2 and 2, and so forth.

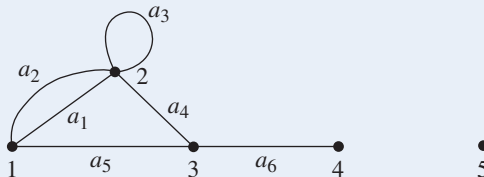


Figure 6.3

The informal definition of a graph works quite well if we have the visual representation of the graph before us to show which arcs connect which nodes. Without the picture, however, we need a concise way to convey this information. Hence our second definition of a graph.

• **DEFINITION GRAPH (Formal)**

A **graph** is an ordered triple (N, A, g) where

N = a nonempty set of **nodes (vertices)**

A = a set of **arcs (edges)**

g = a function associating with each arc a an *unordered* pair x - y of nodes called the **endpoints** of a

EXAMPLE 3

For the graph of Figure 6.3, the function g associating arcs with endpoints performs the following mapping: $g(a_1) = 1$ - 2 , $g(a_2) = 1$ - 2 , $g(a_3) = 2$ - 2 , $g(a_4) = 2$ - 3 , $g(a_5) = 1$ - 3 , and $g(a_6) = 3$ - 4 .

PRACTICE 1

Sketch a graph having nodes $\{1, 2, 3, 4, 5\}$, arcs $\{a_1, a_2, a_3, a_4, a_5, a_6\}$, and function $g(a_1) = 1$ - 2 , $g(a_2) = 1$ - 3 , $g(a_3) = 3$ - 4 , $g(a_4) = 3$ - 4 , $g(a_5) = 4$ - 5 , and $g(a_6) = 5$ - 5 .

We might want the arcs of a graph to begin at one node and end at another, in which case we would use a *directed graph*.

• **DEFINITION DIRECTED GRAPH**

A **directed graph (digraph)** is an ordered triple (N, A, g) where

N = a nonempty set of nodes

A = a set of arcs

g = a function associating with each arc a an *ordered* pair (x, y) of nodes where x is the **initial point** and y is the **terminal point** of a .

In a directed graph, then, there is a direction associated with each arc.

EXAMPLE 4

Figure 6.4 shows a directed graph. There are 4 nodes and 5 arcs. The function g associating arcs with endpoints performs the mapping $g(a_1) = (1, 2)$, meaning that arc a_1 begins at node 1 and ends at node 2. Also, $g(a_3) = (1, 3)$, but $g(a_4) = (3, 1)$.

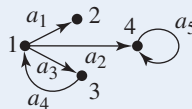


Figure 6.4

Besides imposing direction on the arcs of a graph, we may want to modify the basic definition of a graph in other ways. We often want the nodes of a graph to carry identifying information, like the names of the cities in the map of airline routes.

This map would be a **labeled graph**. We may want to use a **weighted graph**, where each arc has some numerical value, or weight, associated with it. For example, we might want to indicate the distances of the various routes in the airline map.

In this book, the term “graph” will mean an undirected graph. To refer to a directed graph, we will always say “directed graph.”

Applications of Graphs

Although the idea of a graph is very simple, an amazing number of situations have relationships between items that lend themselves to graphical representation. Not surprisingly, there are many graphs in this book. Graphical representations of partially ordered sets (Hasse diagrams) were introduced in Chapter 5. A PERT chart (for example, Figure 5.7) is a directed graph. The *E-R* diagram (for example, Figure 5.10) is a graph. The commutative diagram illustrating composition of functions (Figure 5.23) is a directed graph. Chapter 8 will introduce logic networks and represent them as directed graphs. Directed graphs will also be used to describe finite-state machines in Chapter 9.

We saw that the airline route map was a graph. A representation of any network of transportation routes (a road map, for example), communications lines (as in a computer network), or product or service distribution routes such as natural gas pipelines or water mains is a graph. The chemical structure of a molecule is represented graphically.

PRACTICE 2 Draw the underlying graph in each of the following cases.

- Figure 6.5 is a road map for part of Arizona.
- Figure 6.6 is a representation of an ozone molecule with three oxygen atoms.

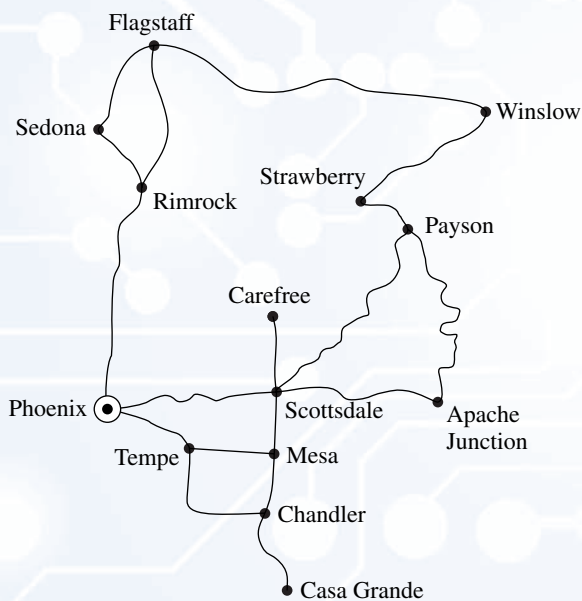


Figure 6.5

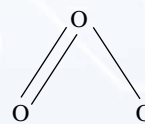


Figure 6.6

EXAMPLE 5

A high-level view of the information flow in a state automobile licensing office is prepared as the first step in developing a new computerized licensing system. Figure 6.7 shows the resulting directed graph, often called a **data flow diagram**.

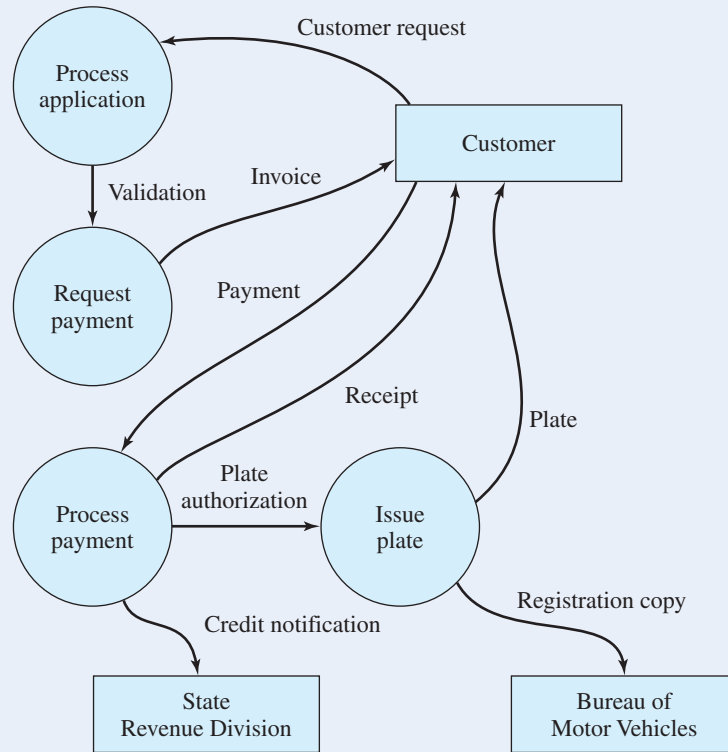


Figure 6.7

EXAMPLE 6

Figure 6.8 shows a graph representation of a local area network of computers in an office complex. In this “star topology,” all machines communicate through a central server. The graph representation highlights one of the weaknesses of such a network design, namely its reliance on continued, dependable operation of the central server.

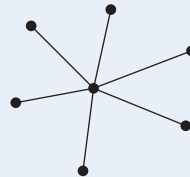


Figure 6.8

EXAMPLE 7

Neural networks, tools used in artificial intelligence for such tasks as pattern recognition, are represented by weighted directed graphs. Figure 6.9 shows a multi-layer network consisting of input units, output units, and a “hidden layer” of units. Weights on the arcs of the graph are adjusted as the neural network “learns” how to recognize certain trial patterns.

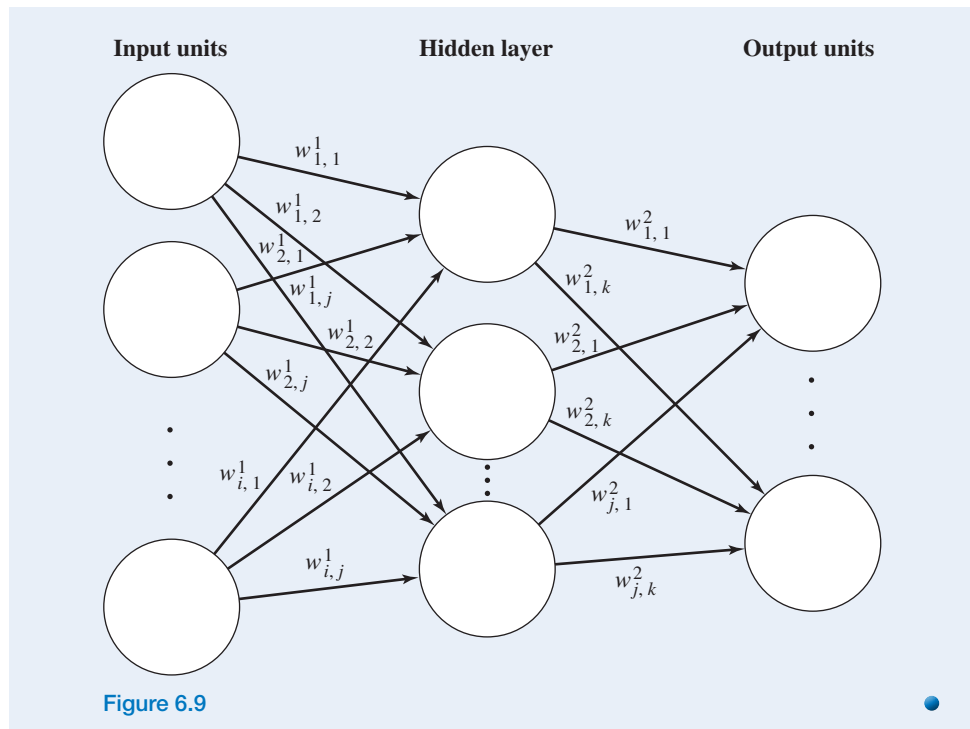
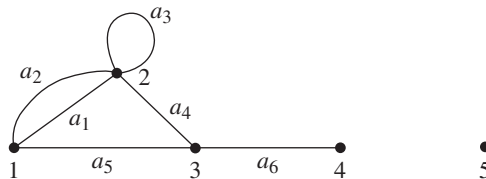


Figure 6.9

Graph Terminology

Before proceeding, we need some terminology about graphs. Surprisingly, although there is a large body of literature in graph theory, the terminology is not completely standard. Therefore other books may give slightly different variations of some of these terms.

Two nodes in a graph are **adjacent** if they are the endpoints associated with an arc. For example in the graph of Figure 6.3 (reproduced here), 1 and 3 are adjacent nodes but 1 and 4 are not. Node 2 is adjacent to itself. A **loop** in a graph is an arc with endpoints $n-n$ for some node n ; in Figure 6.3, arc a_3 is a loop with endpoints 2–2. A graph with no loops is **loop-free**. Two arcs with the same endpoints are **parallel arcs**; arcs a_1 and a_2 in Figure 6.3 are parallel. A **simple graph** is one with no loops or parallel arcs. An **isolated node** is adjacent to no other node; in Figure 6.3, 5 is an isolated node. The **degree** of a node is the number of arc ends at that node. In Figure 6.3, nodes 1 and 3 have degree 3, node 2 has degree 5, node 4 has degree 1, and node 5 has degree 0.



Because the function g that relates arcs to endpoints in the formal definition of a graph is indeed a function, each arc has a unique pair of endpoints. If g is a one-to-one function, then there is at most one arc associated with a pair of endpoints; such

graphs have no parallel arcs. A **complete graph** is one in which any two distinct nodes are adjacent. In this case, g is almost an onto function—every pair x - y of distinct nodes is the image under g of an arc—but there does not have to be a loop at every node. Consequently, pairs of the form x - x need not have a preimage.

A **subgraph** of a graph consists of a set of nodes and a set of arcs that are subsets of the original node set and arc set, respectively, in which the endpoints of an arc must be the same nodes as in the original graph. In other words, it is a graph obtained by erasing part of the original graph and leaving the rest unchanged. Figure 6.10 shows two subgraphs of the graph in Figure 6.3. Note that the graph in Figure 6.10a is simple and also complete.

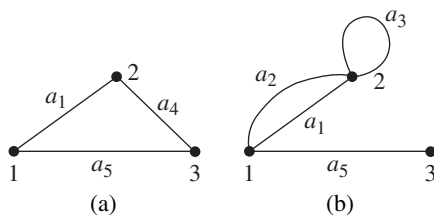


Figure 6.10

A **path** from node n_0 to node n_k is a sequence

$$n_0, a_0, n_1, a_1, \dots, n_{k-1}, a_{k-1}, n_k$$

of nodes and arcs where for each i , the endpoints of arc a_i are n_i - n_{i+1} . In the graph of Figure 6.3, one path from node 2 to node 4 consists of the sequence 2, a_1 , 1, a_2 , 2, a_4 , 3, a_6 , 4. The **length** of a path is the number of arcs it contains; if an arc is used more than once, it is counted each time it is used. The length of the path just described from node 2 to node 4 is 4.

A graph is **connected** if there is a path from any node to any other node. The graphs in Figure 6.10 are each connected, but the graph of Figure 6.3 is not connected. A **cycle** in a graph is a path from some node n_0 back to n_0 where no arc appears more than once in the path sequence, n_0 is the only node appearing more than once, and n_0 occurs only at the ends. (Nodes and arcs may be repeated in a path but not, except for node n_0 , in a cycle.) In the graph of Figure 6.3,

$$1, a_1, 2, a_4, 3, a_5, 1$$

is a cycle. A graph with no cycles is **acyclic**.

PRACTICE 3 Refer to the graph created in Practice 1.

- Find two nodes that are not adjacent.
- Find a node adjacent to itself.
- Find a loop.
- Find two parallel arcs.
- Find the degree of node 3.
- Find a path of length 5.
- Find a cycle.
- Is this graph complete?
- Is this graph connected?

EXAMPLE 8

Figure 6.11 illustrates the simple, complete graphs with 1, 2, 3, and 4 vertices. The simple, complete graph with n vertices is denoted by K_n .

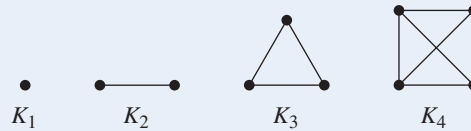


Figure 6.11

PRACTICE 4 Draw K_5 .

Now consider the simple graph in Figure 6.12. It is not a complete graph because it is not true that every node is adjacent to every other node. However, the nodes can be divided into two disjoint sets, $\{1, 2\}$ and $\{3, 4, 5\}$, such that any two nodes chosen from the same set are not adjacent but any two nodes chosen one from each set are adjacent. Such a graph is a *bipartite complete graph*.

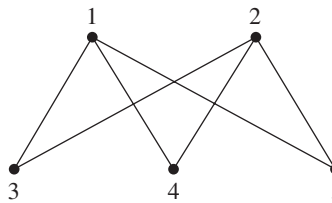


Figure 6.12

DEFINITION BIPARTITE COMPLETE GRAPH

A graph is a **bipartite complete graph** if its nodes can be partitioned into two disjoint nonempty sets N_1 and N_2 such that two nodes x and y are adjacent if and only if $x \in N_1$ and $y \in N_2$. If $|N_1| = m$ and $|N_2| = n$, such a graph is denoted by $K_{m,n}$.

Figure 6.12 therefore illustrates $K_{2,3}$.

PRACTICE 5 Draw $K_{3,3}$.

The concept of a path extends to a directed graph, as we might expect: A **path** from node n_0 to node n_k in a directed graph is a sequence

$$n_0, a_0, n_1, a_1, \dots, n_{k-1}, a_{k-1}, n_k$$

where for each i , n_i is the initial point and n_{i+1} is the terminal point of a_i . If a path exists from node n_0 to node n_k , then n_k is **reachable** from n_0 . The definition of a cycle also carries over to directed graphs.

EXAMPLE 9

In the directed graph of Figure 6.13, there are many paths from node 1 to node 3: $1, a_4, 3$ and $1, a_1, 2, a_2, 2, a_2, 2, a_3, 3$ are two possibilities. Node 3 is certainly reachable from 1. Node 1, however, is not reachable from any other node. The cycles in this graph are the loop a_2 and the path $3, a_5, 4, a_6, 3$.

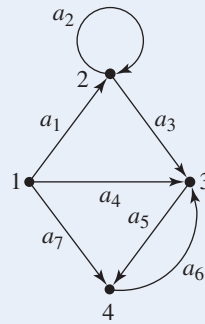


Figure 6.13

We can prove some (fairly trivial) statements about graphs that follow directly from the definitions.

EXAMPLE 10

Prove that an acyclic graph is simple.

We'll use a proof by contraposition. If a graph is not simple, it has either parallel arcs or a loop. The parallel arcs and their endpoints, or the loop and its endpoints, then constitute a cycle, and the graph is not acyclic.

Note that the converse to the statement in Example 10 is not true: Figure 6.10a is a simple graph, but it contains a cycle.

PRACTICE 6

- Prove that every complete graph is connected.
- Find a connected graph that is not complete.

Isomorphic Graphs

Two graphs may appear quite different in their visual representation but still be the same graph according to our formal definition. We want to distinguish between two graphs that have cosmetic visual differences and those that have fundamentally different structures. The graphs in Figures 6.14 and 6.15 are the same—they have the same nodes, the same arcs, and the same arc-to-endpoint function. (In a representation of a graph, arcs can intersect at points that are not nodes of the graph.) The graph in Figure 6.16 is essentially the same graph as well. If we relabeled the nodes and arcs of the graph of Figure 6.14 by the following mappings, the graphs would be the same:

$$\begin{array}{ll} f_1: 1 \rightarrow a & f_2: a_1 \rightarrow e_2 \\ & 2 \rightarrow c \\ & 3 \rightarrow b \\ & 4 \rightarrow d \end{array} \qquad \begin{array}{l} a_2 \rightarrow e_1 \end{array}$$

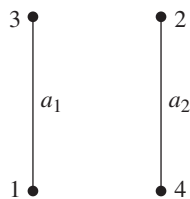


Figure 6.14

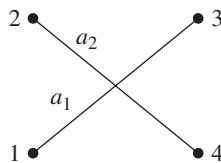


Figure 6.15

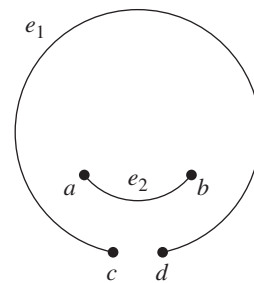


Figure 6.16

Structures that are the same except for relabeling are called *isomorphic* structures. To show that two structures are isomorphic, we must produce a relabeling (one-to-one, onto mappings between the elements of the two structures) and then show that the important properties of the structures are “preserved” (maintained) under this relabeling. In the case of graphs, the elements are nodes and arcs. The “important property” in a graph is which arcs connect which nodes.

The given mappings f_1 and f_2 are one-to-one, onto functions from the nodes and arcs, respectively, of the graph in Figure 6.14 to the nodes and arcs of the graph in Figure 6.16. Furthermore, if an arc a in the graph of Figure 6.14 has endpoints x – y , then the arc $f_2(a)$ in the graph of Figure 6.16 has endpoints $f_1(x)$ – $f_1(y)$, and vice versa. For example, arc a_1 in Figure 6.14 has endpoints 1–3, while its corresponding arc e_2 in Figure 6.16 has endpoints a – b , which are the nodes in Figure 6.16 that correspond to nodes 1 and 3 in Figure 6.14. We can formalize this idea.

DEFINITION ISOMORPHIC GRAPHS

Two graphs (N_1, A_1, g_1) and (N_2, A_2, g_2) are **isomorphic** if there are bijections $f_1: N_1 \rightarrow N_2$ and $f_2: A_1 \rightarrow A_2$ such that for each arc $a \in A_1$, $g_1(a) = x$ – y if and only if $g_2[f_2(a)] = f_1(x)$ – $f_1(y)$.

EXAMPLE 11

The graphs shown in Figure 6.17 are isomorphic. The bijections that establish the isomorphism are partially given here:

$$\begin{array}{ll}
 f_1: 1 \rightarrow c & f_2: a_1 \rightarrow e_1 \\
 2 \rightarrow e & a_2 \rightarrow e_4 \\
 3 \rightarrow d & a_3 \rightarrow e_2 \\
 4 \rightarrow b & \dots \\
 5 \rightarrow a &
 \end{array}$$

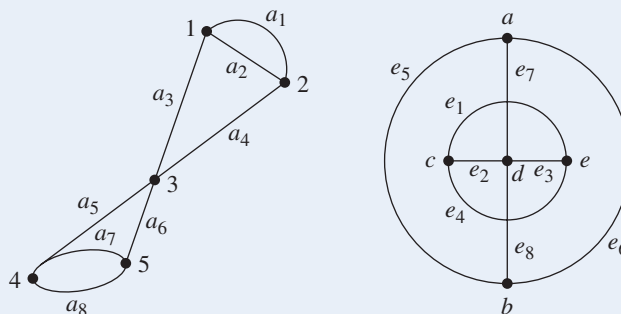


Figure 6.17

Using these bijections, $g_1(a_3) = 1-3$ and $g_2[f_2(a_3)] = g_2(e_2) = c-d = f_1(1)-f_1(3)$. We can see that the arc-to-endpoint relationship is preserved under the relabeling for the case of arc a_3 . To prove that the graphs are isomorphic, we would have to complete the definition of the f_2 function and then demonstrate that the arc-to-endpoint relationship is preserved under these mappings by examining all possible cases.

PRACTICE 7 Complete the definition of the function f_2 in Example 11.

Graph isomorphism is easier to establish if we restrict our attention to simple graphs. If we can find an appropriate function f_1 mapping nodes to nodes, then a function f_2 mapping arcs to arcs is trivial because there is at most one arc between any pair of endpoints. Hence the following theorem is true.

THEOREM ON SIMPLE GRAPH ISOMORPHISM

Two simple graphs (N_1, A_1, g_1) and (N_2, A_2, g_2) are isomorphic if there is a bijection $f: N_1 \rightarrow N_2$ such that for any nodes n_i and n_j of N_1 , n_i and n_j are adjacent if and only if $f(n_i)$ and $f(n_j)$ are adjacent. (The function f is called an **isomorphism** from graph 1 to graph 2.)

PRACTICE 8 Find an isomorphism from the graph of Figure 6.18a to that of Figure 6.18b.

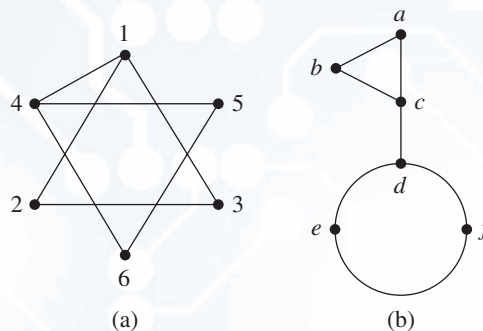


Figure 6.18

Proving that two graphs are isomorphic requires finding the bijection (or, for non-simple graphs, bijections) and then showing that the adjacency property (or arc-to-endpoint relationship) is preserved. To prove that two graphs are not isomorphic, we must prove that the necessary bijection(s) do not exist. We could try all possible bijections (because there is a finite number of nodes and arcs, there is a finite number of bijections). However, this method would quickly get out of hand in graphs of any size at all. Instead, we can try to find some other reason that such bijections could not exist. Although this task is not always easy, there are certain conditions under which it is clear that two graphs are not isomorphic (see Exercise 21). These include the following:

1. One graph has more nodes than the other.
2. One graph has more arcs than the other.

3. One graph has parallel arcs and the other does not.
4. One graph has a loop and the other does not.
5. One graph has a node of degree k and the other does not.
6. One graph is connected and the other is not.
7. One graph has a cycle and the other does not.

PRACTICE 9 Prove that the two graphs in Figure 6.19 are not isomorphic.

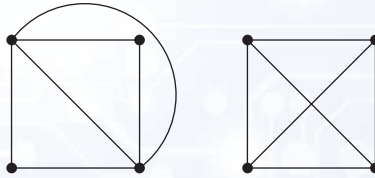


Figure 6.19

EXAMPLE 12

The two graphs of Figure 6.20 are not isomorphic. Note that each graph has six nodes and seven arcs. Neither has parallel arcs or loops. Both are connected. Both have three cycles, four nodes of degree 2, and two nodes of degree 3. Therefore none of the obvious nonisomorphism tests apply. However, the graph in Figure 6.20b has a node of degree 2 that is adjacent to two nodes of degree 3; this is not the case in Figure 6.20a, so the graphs are not isomorphic.

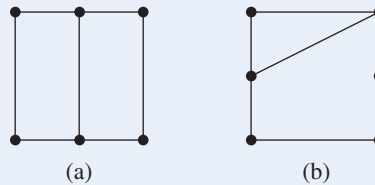


Figure 6.20

Again, graphs that are isomorphic are considered to be “the same” regardless of cosmetic differences in how they are drawn or labeled, whereas nonisomorphic graphs have fundamental structural differences.

Planar Graphs

A **planar graph** is one that can be represented (on a sheet of paper, that is, in the plane) so that its arcs intersect only at nodes. Designers of integrated circuits want all components in one layer of a chip to form a planar graph so that no connections cross. The graph of Figure 6.14 is clearly planar. However, we know that it is isomorphic to the graph of Figure 6.15, so the graph of Figure 6.15 is also planar. The key word in the definition of a planar graph is that it *can* be drawn in a certain way.

PRACTICE 10 Prove that K_4 is a planar graph.

EXAMPLE 13

Consider K_5 , the simple, complete graph with five vertices. We will try to construct K_5 with no intersecting arcs by starting with some of the arcs and then adding as many new arcs as possible without crossing existing arcs. We'll first lay out five vertices and connect them as shown in Figure 6.21a. (Because all the vertices in K_n are symmetric, it doesn't matter how we label them.)

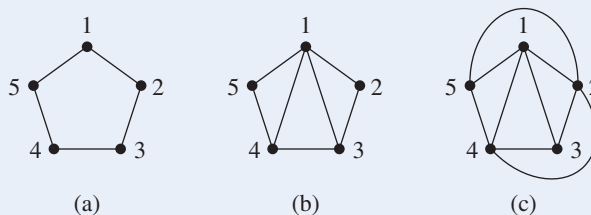


Figure 6.21

Next we connect 1 to 3 and 1 to 4, as shown in Figure 6.21b. Now 2 must be connected to both 4 and 5. This can be accomplished while still preserving the planarity of the graph by putting these new arcs on the outside, as in Figure 6.21c. The final connection is between nodes 3 and 5. But there is no way to draw an arc from node 3 to node 5 without crossing either the 2-4 arc or one or more of the interior arcs, such as 1-4.

We did have a choice of how to place arcs 1-3 and 1-4; we made them interior arcs. We could explore whether making these arcs exterior would change anything, but it turns out that it does not (see Practice 11). Thus it appears that K_5 is not a planar graph. However, we'd still like a proof of this with a firmer foundation—this sounds too much like an “I can't do it so it can't be done” argument. Such a proof will be given shortly. ●

PRACTICE 11

Show that adding arcs 1-3 and 1-4 as exterior arcs when constructing K_5 still leads to a situation where arcs must intersect. ■

PRACTICE 12

Present a construction-type argument that $K_{3,3}$ is not a planar graph. ■

One fact about planar graphs was discovered by the eighteenth-century Swiss mathematician Leonhard Euler (pronounced “oiler”). A simple, connected, planar graph (when drawn in its planar representation, with no arcs crossing) divides the plane into a number of regions, including totally enclosed regions and one infinite exterior region. Euler observed a relationship between the number n of nodes, the number a of arcs, and the number r of regions in such a graph. This relationship is known as **Euler's formula**:

$$n - a + r = 2 \quad (1)$$

PRACTICE 13

Verify Euler's formula for the simple, connected, planar graph in Figure 6.18b. ■

To prove Euler's formula, we will do a proof by induction on a , the number of arcs. In the base case, $a = 0$ and the graph consists of a single node; the only region is the exterior region (Figure 6.22a). Here $n = 1$, $a = 0$, and $r = 1$, and Equation (1) holds.

Now assume that the formula holds for the planar representation of any simple, connected, planar graph with k arcs, and consider such a graph with $k + 1$ arcs. As usual, we must somehow relate the “ $k + 1$ instance” to a “ k instance” so that we can make use of the inductive hypothesis. Here we consider two cases for the graph with $k + 1$ arcs.

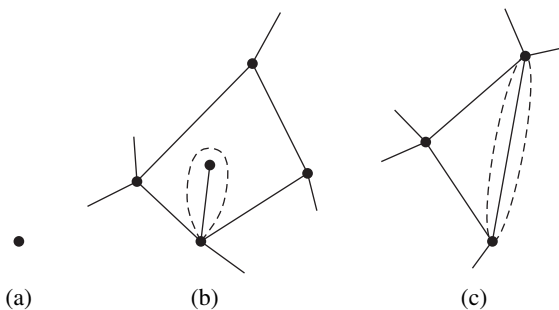


Figure 6.22

Case 1. The graph has a node of degree 1. Temporarily erasing this node and its connecting arc (Figure 6.22b) leaves a simple, connected, planar graph with k arcs, some number n of nodes, and some number r of regions for which (by the inductive hypothesis)

$$n - k + r = 2$$

In the original graph, there was one more arc and one more node but the same number of regions, so the appropriate formula is

$$(n + 1) - (k + 1) + r = 2$$

which, by the inductive hypothesis, is true.

Case 2. The graph has no nodes of degree 1. Then temporarily erase one arc that helps define an enclosed region (Figure 6.22c). (If no arcs help define an enclosed region, the graph is a chain and there is a node of degree 1.) This leaves a simple, connected, planar graph with k arcs, some number n of nodes, and some number r of regions for which (by the inductive hypothesis)

$$n - k + r = 2$$

n the original graph, there was one more arc and one more region, but the same number of nodes, so the appropriate formula is

$$n - (k + 1) + (r + 1) = 2$$

which, by the inductive hypothesis, is true.

PRACTICE 14

In the proof of Euler’s formula, explain why in case 2 the arc to be erased must help define an enclosed region. Give two reasons.

There are two consequences of Euler’s formula if we place further restrictions on the graph. Suppose we require that the graph not only be simple, connected, and planar but also have at least three nodes. In a planar representation of such a graph, we

can count the number of edges that are adjacent to (form the boundaries of) each region, including the exterior region. Arcs that are wholly interior to a region contribute two edges to that region; for example, if we trace the boundary of the interior region shown in Figure 6.22b, we travel six edges, including the arc out to the node of degree 1 and then back again. Arcs that separate two regions contribute one edge to each region. Therefore, if there are a arcs in the graph, the number of region edges is $2a$.

There are no regions with exactly one adjacent edge, because there are no loops in the graph. There are no regions with exactly two adjacent edges, because there are no parallel arcs and the graph consisting entirely of one arc joining two nodes (which would have two edges adjacent to the exterior region) is excluded. Therefore each region has at least three adjacent edges, so $3r$ is the minimum number of region edges. Thus

$$2a \geq 3r$$

or, from Equation (1),

$$2a \geq 3(2 - n + a) = 6 - 3n + 3a$$

and finally

$$a \leq 3n - 6 \tag{2}$$

If a final restriction that there are no cycles of length 3 is placed on the graph, then each region has at least four adjacent edges, so $4r$ is the minimum number of region edges. This leads to the inequality

$$2a \geq 4r$$

which becomes

$$a \leq 2n - 4 \tag{3}$$

These results are summarized in the following theorem.

● **THEOREM ON THE NUMBER OF NODES AND ARCS**

For a simple, connected, planar graph with n nodes and a arcs:

1. If the planar representation divides the plane into r regions, then

$$n - a + r = 2 \tag{1}$$

2. If $n \geq 3$, then

$$a \leq 3n - 6 \tag{2}$$

3. If $n \geq 3$ and there are no cycles of length 3, then

$$a \leq 2n - 4 \tag{3}$$

Note that inequality (3) places a tighter bound on the number of arcs than inequality (2), but an additional condition has been imposed on the graph.

We can use this theorem to prove that certain graphs are not planar.

EXAMPLE 14

K_5 is a simple, connected graph with 5 nodes (and 10 arcs). If it were a planar graph, inequality (2) of our theorem would hold, but $10 > 3(5) - 6$. Therefore, just as our construction argument showed, K_5 is not planar. $K_{3,3}$ is a simple, connected graph with 6 nodes (and 9 arcs). It has no cycles of length 3, because this would require two nodes in one of the two subsets to be adjacent. If it were a planar graph, inequality (3) would hold, but $9 > 2(6) - 4$. Therefore $K_{3,3}$ is not planar. ●

PRACTICE 15

Show that inequality (2) does hold for $K_{3,3}$, which shows that this inequality is a necessary but not sufficient condition for planarity in graphs with $n \geq 3$. ■

The nonplanar graphs K_5 and $K_{3,3}$ play a central role in all nonplanar graphs. To state what this role is, we need one more definition.

● **DEFINITION HOMEOMORPHIC GRAPHS**

Two graphs are **homeomorphic** if both can be obtained from the same graph by a sequence of elementary subdivisions, in which a single arc $x-y$ is replaced by two new arcs $x-v$ and $v-y$ connecting to a new node v .

EXAMPLE 15

The graphs in parts (b) and (c) of Figure 6.23 are homeomorphic because each can be obtained from the graph of Figure 6.23a by a sequence of elementary subdivisions. (However, neither can be obtained from the other by a sequence of elementary subdivisions.)

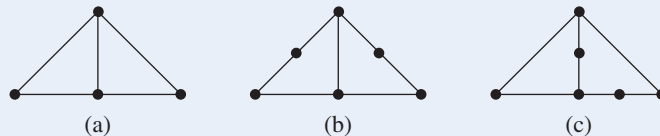


Figure 6.23

A graph that is planar cannot be turned into a nonplanar graph by elementary subdivisions, and a graph that is nonplanar cannot be turned into a planar graph by elementary subdivisions (see Exercise 32). As a result, homeomorphic graphs are either both planar or both nonplanar. The following theorem, due to the Polish mathematician Kazimierz Kuratowski, characterizes nonplanar graphs.

● **THEOREM KURATOWSKI THEOREM**

A graph is nonplanar if and only if it contains a subgraph that is homeomorphic to K_5 or $K_{3,3}$.

We won't prove this theorem, although one direction is easy to see. If a graph has a subgraph homeomorphic to the nonplanar graphs K_5 or $K_{3,3}$, then the subgraph—and hence the entire graph—is nonplanar.

EXAMPLE 16

Figure 6.24a shows the “Petersen graph.” We will prove that this graph is not planar by finding a subgraph homeomorphic to $K_{3,3}$. By looking at the top of the graph, we can see that node a is adjacent to nodes $e, f,$ and b , none of which are adjacent to each other. Also, node e is adjacent to nodes d and j as well as a , and nodes $a, d,$ and j are not adjacent to each other. This information is incorporated in the graph of Figure 6.24b, which is also a subgraph of $K_{3,3}$. The arcs needed to complete $K_{3,3}$ are shown as dotted lines in Figure 6.24c. These arcs are not in the Petersen graph; for example, no $j-f$ arc is present. However, there is a path in the Petersen graph from j to f using the intermediate node h , that is, $j-h$ and $h-f$. Similarly, there are paths $j-g$ and $g-b$, $d-i$ and $i-f$, and $d-c$ and $c-b$. Adding these paths to Figure 6.24b results in Figure 6.24d, which is a subgraph of the Petersen graph and is also obtainable from Figure 6.24c by a sequence of elementary subdivisions.

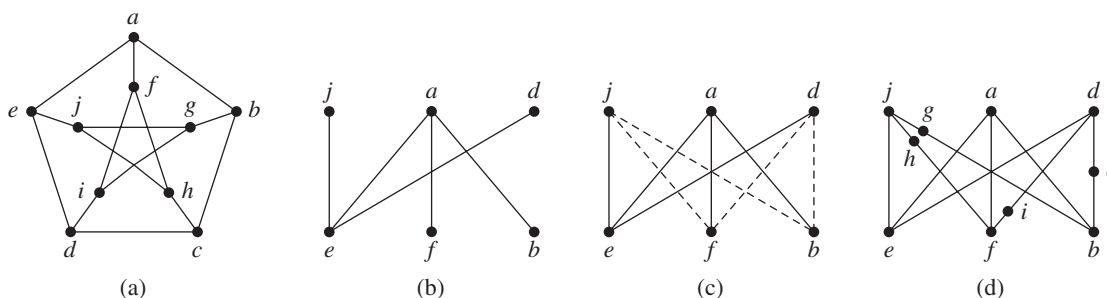


Figure 6.24

Computer Representation of Graphs

We have said that the major advantage of a graph is its visual representation of information. What if we want to store a graph in digital form? Although it is possible to store a digital image of a graph, it takes a lot of space. Furthermore, such an image remains but a picture—the data it represents can’t be manipulated in any way. What we need to store are these essential data that are part of the definition of a graph—what the nodes are and which nodes have connecting arcs. From this information a visual representation could be reconstructed if desired. The usual computer representations of a graph involve one of two data structures, either an adjacency matrix or an adjacency list.

Adjacency Matrix

Suppose a graph has n nodes, numbered n_1, n_2, \dots, n_n . This numbering imposes an arbitrary ordering on the set of nodes; recall that a set is an unordered collection. However, this is done merely as a means to identify the nodes—no significance is attached to one node appearing before another in this ordering. Having ordered the nodes, we can form an $n \times n$ matrix where entry i, j is the number of arcs between nodes n_i and n_j . This matrix is called the **adjacency matrix A** of the graph with respect to this ordering. Thus,

$$a_{ij} = p \text{ where there are } p \text{ arcs between } n_i \text{ and } n_j$$

EXAMPLE 17

The adjacency matrix for the graph in Figure 6.25 with respect to the ordering 1, 2, 3, 4 is a 4×4 matrix. Entry 1,1 is a 1 due to the loop at node 1. All other elements on the main diagonal are 0. Entry 2,1 (second row, first column) is a 1 because there is one arc between node 2 and node 1, which also means that entry 1, 2 is a 1.

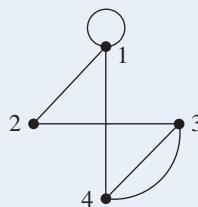


Figure 6.25

So far we have

$$A = \begin{bmatrix} 1 & 1 & - & - \\ 1 & 0 & - & - \\ - & - & 0 & - \\ - & - & - & 0 \end{bmatrix}$$

PRACTICE 16 Complete the adjacency matrix for Figure 6.25.

The adjacency matrix in Practice 16 is symmetric, which will be true for the adjacency matrix of any undirected graph—if there are p arcs between n_i and n_j , there are certainly p arcs between n_j and n_i . The symmetry of the matrix means that only elements on or below the main diagonal need to be stored. Therefore, all the information contained in the graph in Figure 6.25 is contained in the “lower triangular” array shown, and the graph could be reconstructed from this array. (The “upper triangular” version could also be used.)

$$\begin{bmatrix} 1 & & & \\ 1 & 0 & & \\ 0 & 1 & 0 & \\ 1 & 0 & 2 & 0 \end{bmatrix}$$

In a directed graph, the adjacency matrix A reflects the direction of the arcs. For a directed matrix,

$$a_{ij} = p \text{ where there are } p \text{ arcs from } n_i \text{ to } n_j$$

An adjacency matrix for a directed graph will not necessarily be symmetric, because an arc from n_i to n_j does not imply an arc from n_j to n_i .

EXAMPLE 18

Consider the directed graph of Figure 6.26.

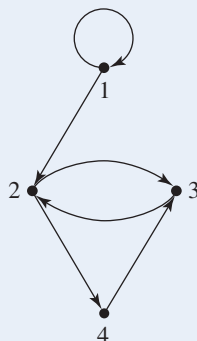


Figure 6.26

The adjacency matrix is

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

In a simple weighted graph, the entries in the adjacency matrix can indicate the weight of an arc by the appropriate number rather than just indicating the presence of an arc by the number 1.

Adjacency List

Many graphs, far from being complete graphs, have relatively few arcs. Such graphs have **sparse** adjacency matrices; that is, the adjacency matrices contain many zeros. Yet if the graph has n nodes, it still requires n^2 data items to represent the adjacency matrix (or more than $n^2/2$ if a triangular matrix is used), even if many of these items are zero. Any algorithm or procedure in which every arc in the graph must be examined requires looking at all n^2 items in the matrix, since there is no way of knowing which entries are nonzero without examining them. To find all the nodes adjacent to a given node n_i requires scanning the entire i th row of the adjacency matrix, a total of n items.

A graph with relatively few arcs can be represented more efficiently by storing only the nonzero entries of the adjacency matrix. This representation consists of a list for each node of all the nodes adjacent to it. Pointers are used to get us from one item in the list to the next. Such an arrangement is called a **linked list**. There is an array of n pointers, one for each node, to get each list started. This **adjacency list** representation, although it requires extra storage for the pointers, may still be more efficient than an adjacency matrix. To find all the nodes adjacent to n_i requires traversing the linked list for n_i , which may have far fewer than the n elements we had to examine in the adjacency matrix. However, there are trade-offs; if we want to determine whether one particular node n_j is adjacent to n_i , we may have to traverse all of n_i 's linked list, whereas in the adjacency matrix we could access element i, j directly.

EXAMPLE 19

The adjacency list for the graph of Figure 6.25 contains a four-element array of pointers, one for each node. The pointer for each node points to an adjacent node, which points to another adjacent node, and so forth. The adjacency list structure is shown in Figure 6.27.

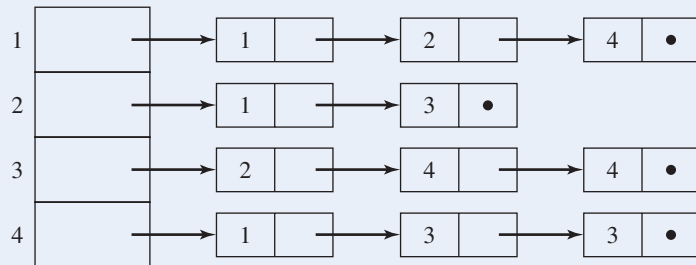


Figure 6.27

In the figure the dot indicates a **null pointer**, meaning that there is nothing more to be pointed to or that the end of the list has been reached. We have dealt with parallel arcs by listing a given node more than once on the adjacency list for n_i if there is more than one arc between n_i and that node. Note that the arrow in list 1 from the 2 node to the 4 node does not mean that there is an arc from node 2 to node 4; all the elements in the node 1 list are adjacent to node 1, not necessarily to each other. ●

PRACTICE 17

Draw the adjacency list representation for the graph shown in Figure 6.28.

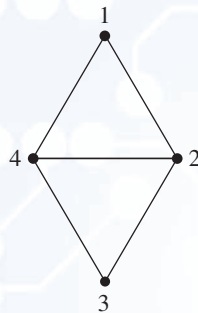


Figure 6.28

In an undirected graph, each arc is represented twice. If n_j is on the adjacency list of n_i , then n_i is also on the adjacency list of n_j . The adjacency list representation for a directed graph puts n_j on the list for n_i if there is an arc from n_i to n_j ; n_i would not necessarily be on the adjacency list for n_j . For a labeled graph or a weighted graph, additional data items can be stored with the node name in the adjacency list.

EXAMPLE 20

Figure 6.29a shows a weighted directed graph. The adjacency list representation for this graph is shown in Figure 6.29b. For each record in the list, the first data item is the node, the second is the weight of the arc to that node, and the third is the pointer. Note that entry 4 in the array of startup pointers is null because there are no arcs that begin at node 4.

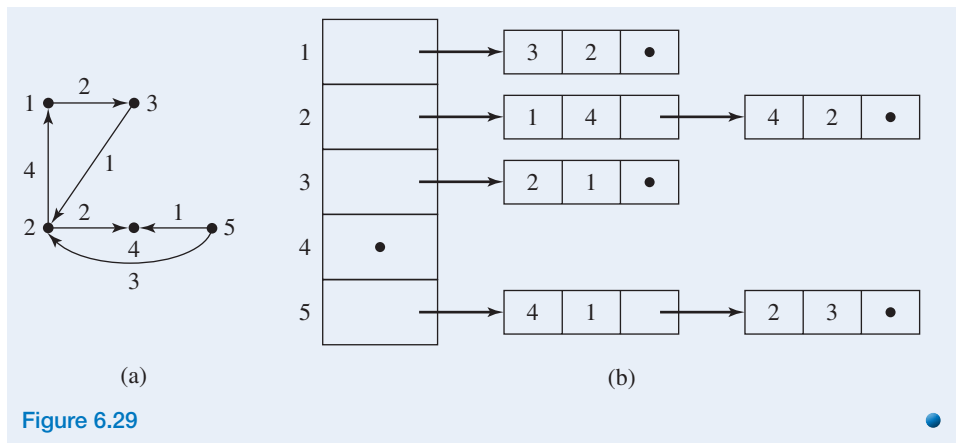


Figure 6.29

In a programming language that does not support pointers, we can still achieve the effect of an adjacency list by using a multicolumn array (or an array of records), where one column contains the nodes and another column contains the array index of the next node on the adjacency list—a “pseudopointer.” The disadvantage of this approach is that the maximum amount of storage space that might be needed for an n -node graph must be set aside for the array; once we start to fill the array, new space cannot be dynamically created if we learn that there are still more adjacent nodes.

EXAMPLE 21

The array-pointer representation of the graph of Figure 6.29a is shown in Figure 6.30. A null pointer is indicated by an array index of 0.

	Node	Weight	Pointer
1			6
2			7
3			9
4			0
5			10
6	3	2	0
7	1	4	8
8	4	2	0
9	2	1	0
10	4	1	11
11	2	3	0

Figure 6.30

In this array, row 2, representing node 2, has a pointer to index 7. At index 7 of the array, we find node 1 with weight 4, representing the arc of weight 4 from node 2 to node 1. The pointer to index 8 says that the adjacency list for node 2 has more entries. At index 8, we learn that there is an arc from 2 to 4 of weight 2, and that this completes the adjacency list for node 2.

Isomorphic Protein Graphs

The question of whether two graphs are isomorphic may seem to be of only academic interest, but in fact it is an important question in modern biological research. We are all familiar with the concept that the DNA in our cells carries our genetic information. But each cell also contains thousands of proteins, long chains of various kinds of amino acids. There are up to 20 different kinds of amino acids, and the sequence of amino acids determines the three-dimensional shape of a given protein as well as its function within the cell. Common categories of proteins according to their functions include the following:

- Enzymes: Initiate chemical processes within the cell and form new molecules
- Antibodies: Recognize and defend against foreign particles such as bacteria
- Transport: Carry molecules such as oxygen throughout the cell and the body
- Structural: Give the cell its shape and structure
- Hormones: Transmit signals throughout the body to coordinate biological processes

Proteins within a cell do not act in isolation; rather, they interact with other proteins in the cell to carry out complex tasks such as DNA replication or to transport oxygen throughout the cell. Protein-to-protein interactions (PPI) are therefore what make a cell function, and any change, however minor, in a given protein could affect the PPIs for that protein and thus affect the entire cell. Knowledge of cell biology at the PPI level is of great interest for identifying the underlying cause of disease and development of therapeutics.

Now consider the various proteins within a given cell as nodes, and two proteins that interact as

connected by an arc. Voilà—we have a graph! And given the number of proteins within a cell and the number and complexity of their interactions, it is a huge graph. A number of experiments have been done to determine what this graph looks like in various organisms, but the data gained seem to have low reliability. This problem has promoted interest in comparing PPIs across species to find commonalities that would reinforce information about the functionality of those interactions. Basically, researchers are looking for isomorphic graphs (or subgraphs) between cells from two different species.

The general problem of determining whether two graphs are isomorphic has no known polynomial-time solution—there is no known efficient algorithm. The performance of an inefficient algorithm might improve if certain conditions on the graphs are met: Are they both planar? Are they both trees? Such simplifications are unlikely to occur in the biological world of cell proteins. Consequently, various “heuristic” approaches (read “educated guesses”) are used. One of these approaches ranks the similarities between the sequences of amino acids forming the proteins, one from each species, that might be considered as matching pairs for a graph isomorphism. This ranking algorithm is based on the iterative PageRank algorithm (invented by Google cofounder Larry Page) that ranks Web page x based on the number of pages that link to x and the PageRank value of those pages.

“Comparative Analysis of Protein Networks: Hard Problems, Practical Solutions,” Atias, N., and Sharan, R., *Communications of the ACM*, May 2012.

http://www.ncbi.nlm.nih.gov/About/primer/genetics_genome.html

<http://ghr.nlm.nih.gov/handbook/howgeneswork/protein>

SECTION 6.1 REVIEW

TECHNIQUES

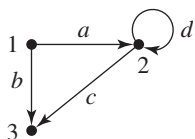
- Use graph terminology.
- W Prove or disprove that two graphs are isomorphic.
- Find a planar representation of a simple graph or prove that none exists.
- W Construct adjacency matrices and adjacency lists for graphs and directed graphs.

MAIN IDEAS

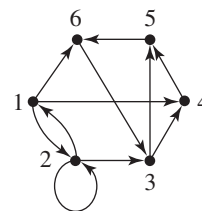
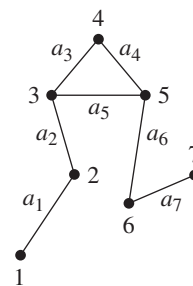
- Diverse situations can be modeled by graphs.
- Graphs can be represented in a computer by matrices or by linked lists.

EXERCISES 6.1

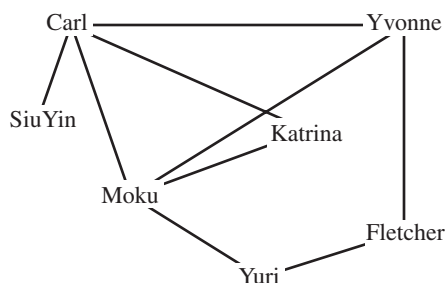
1. Give the function g that is part of the formal definition of the directed graph shown.



2. Use the graph in the figure to answer the questions that follow.
 - a. Is the graph simple?
 - b. Is the graph complete?
 - c. Is the graph connected?
 - d. Can you find two paths from 3 to 6?
 - e. Can you find a cycle?
 - f. Can you find an arc whose removal will make the graph acyclic?
 - g. Can you find an arc whose removal will make the graph not connected?
3. Sketch a picture of each of the following graphs.
 - a. Simple graph with three nodes, each of degree 2
 - b. Graph with four nodes, with cycles of length 1, 2, 3, and 4
 - c. Noncomplete graph with four nodes, each of degree 4
4. Use the directed graph in the figure to answer the questions that follow.
 - a. Which nodes are reachable from node 3?
 - b. What is the length of the shortest path from node 3 to node 6?
 - c. What is a path from node 1 to node 6 of length 8?
5. Draw K_6 .
6. Draw $K_{3,4}$.
7. For each of the following characteristics, draw a graph or explain why such a graph does not exist.
 - a. Four nodes of degree 1, 2, 3, and 4, respectively
 - b. Simple, four nodes of degree 1, 2, 3, and 4, respectively
 - c. Four nodes of degree 2, 3, 3, and 4, respectively
 - d. Four nodes of degree 2, 3, 3, and 3, respectively



8. For each of the following characteristics, draw a graph or explain why such a graph does not exist.
- Simple graph with seven nodes, each of degree 3
 - Four nodes, two of degree 2 and two of degree 3
 - Three nodes of degree 0, 1, and 3, respectively
 - Complete graph with 4 nodes each of degree 2
9. An *acquaintanceship graph* is an undirected graph in which the nodes represent people and nodes a and b are adjacent if a and b are acquainted.
- The acquaintanceship graph for the IT department and the marketing department of a major corporation is an unconnected graph. What does this imply?
 - The following figure represents an acquaintanceship graph for residents of an apartment building. Are Carl and Fletcher acquainted? How many people is SiuYin acquainted with?



- The length of the shortest path between node a and node b in an acquaintanceship graph is sometimes called the *degree of separation* between a and b . What is the degree of separation between Carl and Yuri?
10. The “small world effect” states that the average degree of separation (see Exercise 9) in an acquaintanceship graph of the whole world is 6. In other words, a path of acquaintance relationships from you to any other person on earth exists with, on the average, a path length of 6 (5 intermediate persons). Experiments in delivering hard-copy letters and e-mail messages have empirically confirmed this theory.¹
- What are the potential implications for e-mail traffic if the small world effect holds for computer networks?
 - What are the potential implications for epidemiology if the small world effect holds for physical contact between humans?
11. The small world effect (see Exercise 10) has been found to be true between root words (that is, basic words found in a thesaurus) in the English language, with an average degree of separation equal to 3. Here “adjacent words” are those that are listed as synonyms in an English thesaurus. For example, “gate” and “commotion” are related by 3 degrees of separation, as follows:

gate → door → flap → commotion

Can you think of 3 degrees of separation between the following pairs of words?

- “star” and “sculpture”
- “burden” and “influence”
- “piano” and “significance”

¹But more recent analyses of 721 million Facebook users, a much larger community than was available to earlier studies, suggests that the average number of intermediaries between persons A and B is 3.74. It’s a small world indeed, at least for Facebook users.

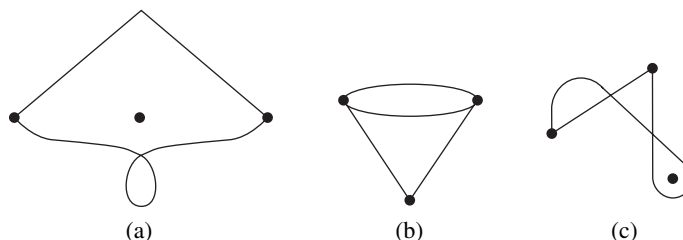
12. An idea closely related to the average degree of separation in a graph is that of clustering. The *global clustering coefficient* for a given graph is given by

$$C = \frac{3 * T}{t}$$

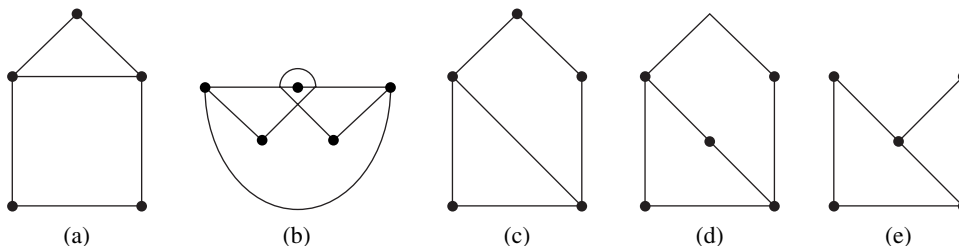
where T = the number of triangles in the graph and t = the number of connected node triples.

A connected node triple is a “center” node adjacent to an unordered pair of other nodes. For example, in the graph of Exercise 2, 3–4–5 (or 5–4–3) and 4–5–6 (6–5–4) are two such triples. Nodes that make up a triangle demonstrate transitivity; if a is adjacent to b and b is adjacent to c , then a is adjacent to c . Therefore C is a ratio of nodes in a transitive threesome to all nodes in a threesome. (One might think of this in terms of a social network as the probability that if you are a “friend” of mine and x is a “friend” of yours, then x is also a “friend” of mine.)

- Consider the graph in Figure 6.28 and the graph for Exercise 2. Which do you think has the higher clustering coefficient?
 - Compute the clustering coefficient for the graph in Figure 6.28
 - Compute the clustering coefficient for the graph for Exercise 2.
13. Which of the following graphs is not isomorphic to the others, and why?

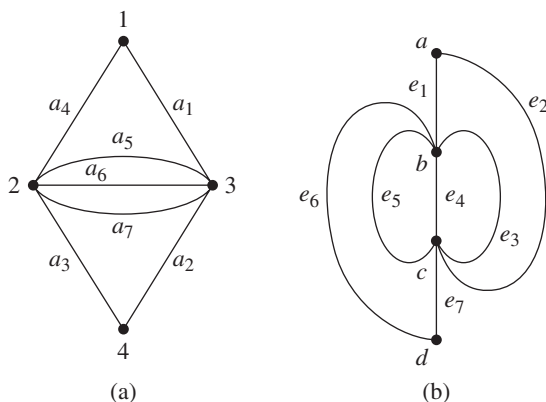


14. Which of the following graphs is not isomorphic to the others, and why?

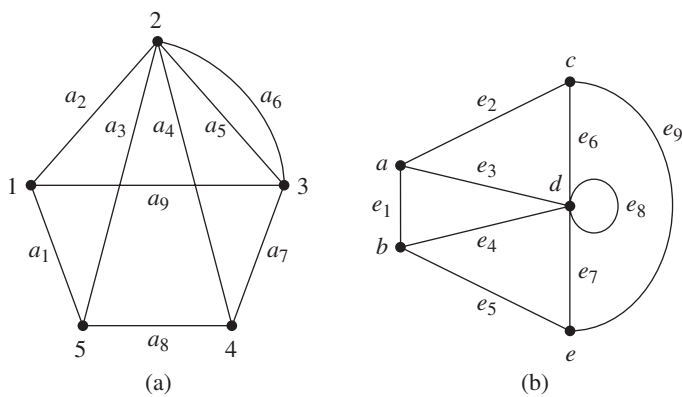


For Exercises 15–20, decide if the two graphs are isomorphic. If so, give the function or functions that establish the isomorphism; if not, explain why.

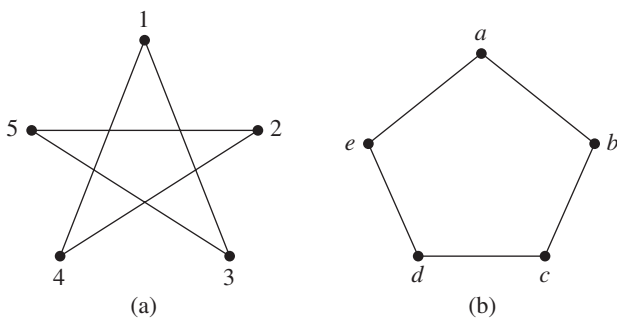
- 15.



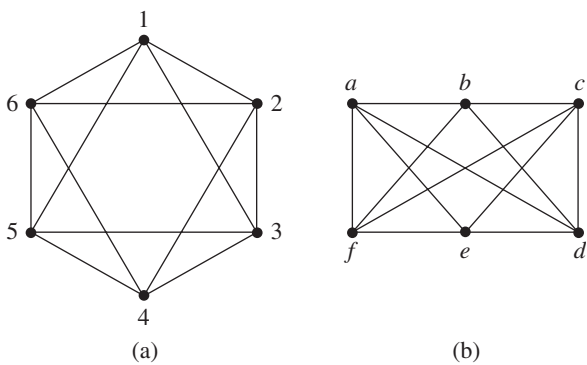
16.



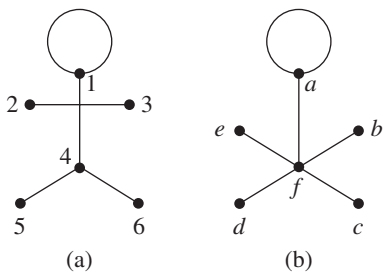
17.



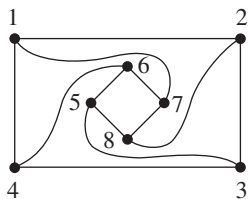
18.



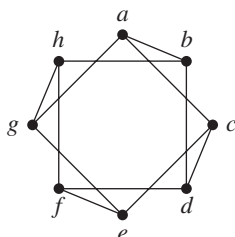
19.



20.



(a)



(b)

21. Prove that two graphs are not isomorphic if one of them

- a. has more nodes than the other.
- b. has more arcs than the other.
- c. has parallel arcs and the other does not.
- d. has a loop and the other does not.
- e. has a node of degree k and the other does not.
- f. is connected and the other is not.
- g. has a cycle and the other does not.

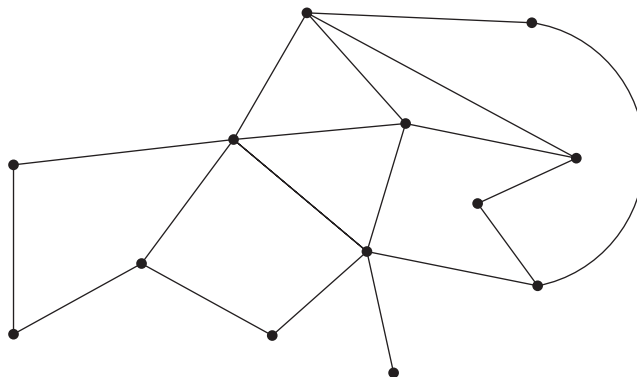
22. Draw all the nonisomorphic, simple graphs with two nodes.

23. Draw all the nonisomorphic, simple graphs with three nodes.

24. Draw all the nonisomorphic, simple graphs with four nodes.

25. Find an expression for the number of arcs in K_n and prove that your expression is correct.

26. Verify Euler's formula for the following simple, connected, planar graph.



27. Prove that $K_{2,3}$ is a planar graph.

28. Prove that the following graph is a planar graph.

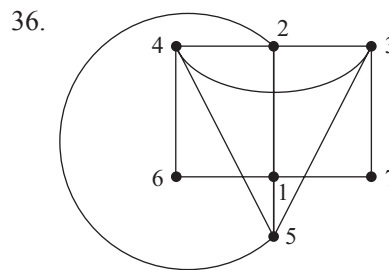
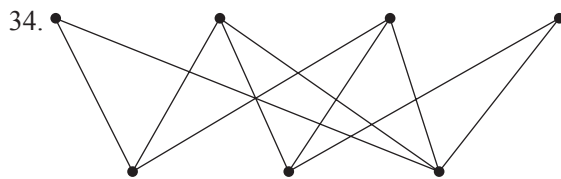
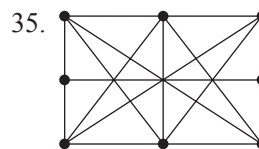
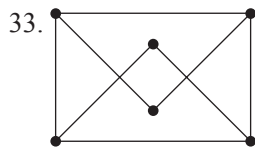


29. If a simple, connected, planar graph has six nodes, all of degree 3, into how many regions does it divide the plane?

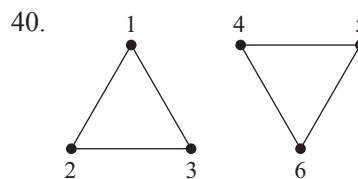
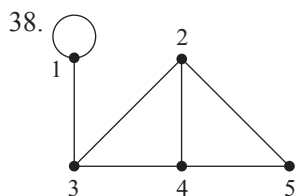
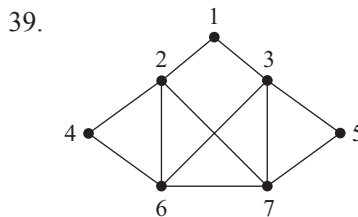
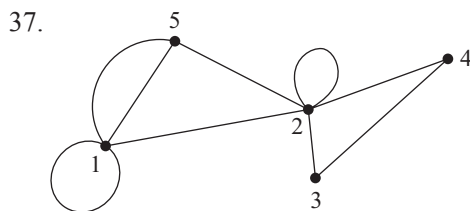
30. If all the nodes of a simple, connected, planar graph have degree 4 and the number of arcs is 12, into how many regions does it divide the plane?
31. Does Euler's formula (Equation (1) of the theorem on the number of nodes and arcs) hold for nonsimple graphs? What about inequalities (2) and (3) of the theorem?
32. What is wrong with the following argument that claims to use elementary subdivisions to turn a nonplanar graph into a planar graph?

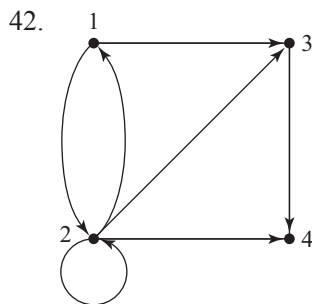
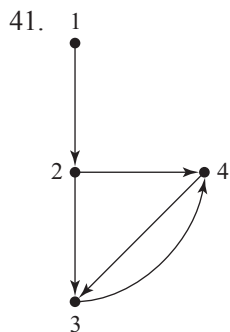
In a nonplanar graph there must be two arcs a_i and a_j that intersect at a point v that is not a node. Do an elementary subdivision on a_i with an inserted node at v and an elementary subdivision on a_j with an inserted node at v . In the resulting graph, the point of intersection is a node. Repeat this process with any non-node intersections; the result is a planar graph.

For Exercises 33–36, determine whether the graph is planar (by finding a representation where arcs intersect only at nodes) or nonplanar (by finding a subgraph homeomorphic to K_5 or $K_{3,3}$).



For Exercises 37–42, write the adjacency matrix for the given graph.





For Exercises 43–46, draw the graph represented by the adjacency matrix.

43.
$$\begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & 2 \\ 0 & 2 & 0 \end{bmatrix}$$

45.
$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

44.
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 0 \end{bmatrix}$$

46.
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

47. The adjacency matrix for an undirected graph is given in lower triangular form by

$$\begin{bmatrix} 2 & & & & \\ 1 & 0 & & & \\ 0 & 1 & 1 & & \\ 0 & 1 & 2 & 0 & \end{bmatrix}$$

Draw the graph.

48. The adjacency matrix for a directed graph is given by

$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Draw the graph.

49. Describe the graph whose adjacency matrix is I_n , the $n \times n$ identity matrix.

50. Describe the graph whose adjacency matrix is 0_n , the $n \times n$ matrix of all 0's.

51. Describe the adjacency matrix for K_n , the simple, complete graph with n nodes.

52. Given the adjacency matrix A for a directed graph G , describe the graph represented by the adjacency matrix A^T (see Exercise 15 in Section 5.7).

For Exercises 53–58, draw the adjacency list representation for the indicated graph.

53. Exercise 37

54. Exercise 38

55. Exercise 39

56. Exercise 40

57. Exercise 41

58. Exercise 42

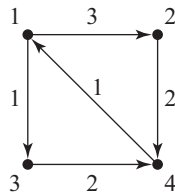
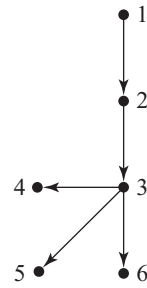
59. Refer to the accompanying graph.

a. Draw the adjacency list representation.

b. How many storage locations are required for the adjacency list? (A pointer takes one storage location.)

c. How many storage locations would be required in an adjacency matrix for this graph?

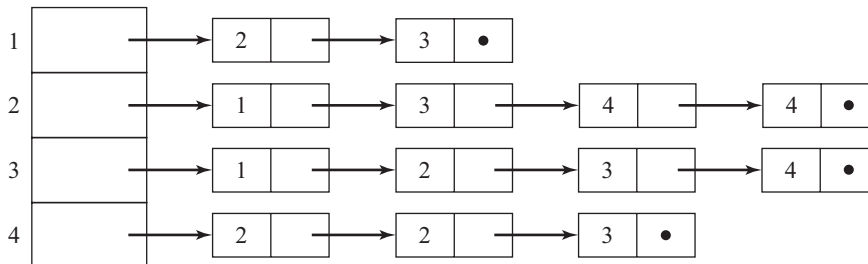
60. Draw the adjacency list representation for the following weighted directed graph.



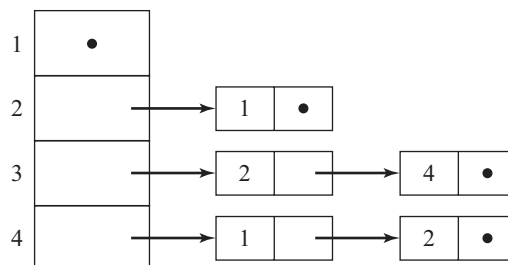
61. For the directed graph of Exercise 42, construct the array-pointer representation.

62. For the weighted directed graph of Exercise 60, construct the array-pointer representation.

63. Draw the undirected graph represented by the following adjacency list.

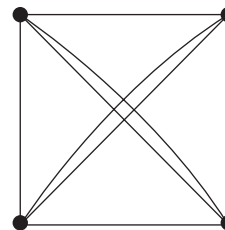


64. Draw the directed graph represented by the following adjacency list.



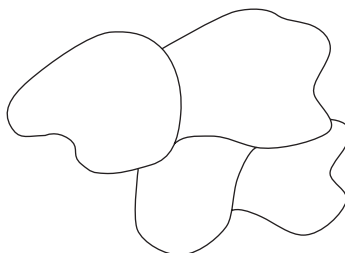
Exercises 65–72 refer to the complement of a graph. If G is a simple graph, the *complement* of G , denoted G' , is the simple graph with the same set of nodes as G , where nodes x – y are adjacent in G' if and only if they are not adjacent in G .

65. Draw G' for the graph of Figure 6.18a.
66. Draw K'_4 .
67. Show that if two simple graphs G_1 and G_2 are isomorphic, so are their complements G'_1 and G'_2 .
68. A simple graph is *self-complementary* if it is isomorphic to its complement. Prove that in a self-complementary graph with n nodes ($n > 1$), $n = 4k$ or $n = 4k + 1$ for some integer k . (*Hint*: Use the result of Exercise 25.)
69. Prove that in any simple graph G with at least two nodes, if G is not connected, then G' is connected. (*Hint*: If G is not connected, then G consists of a collection of “disjoint” connected subgraphs.)
70. Find a simple graph G with at least two nodes where both G and G' are connected, thus showing that the converse of Exercise 69 is false.
71. Given an adjacency matrix A for a simple graph G , describe the adjacency matrix for G' .
72. Prove that if $|N| \geq 11$ in a simple, connected graph G , then not both G and G' can be planar.
73. Prove that in any simple graph G with n nodes and a arcs, $2a \leq n^2 - n$.
74. Prove that a simple, connected graph with n nodes has at least $n - 1$ arcs. (*Hint*: Show that this can be restated as “A simple, connected graph with m arcs has at most $m + 1$ nodes.” Then use the second principle of induction on m .)
75. Prove that a simple graph with n nodes ($n \geq 2$) and more than $C(n - 1, 2)$ arcs is connected. (*Hint*: Use Exercises 69 and 74.)
76. Euler’s formula is stated for simple, connected planar graphs, but in fact the word “simple” could be omitted.
 - a. A non-simple graph is a simple graph with loops or parallel arcs added. Prove that any simple connected planar graph remains a connected planar graph if parallel arcs or loops are added. (*Hint*: Temporarily erase the parallel arcs and loops.)
 - b. Prove that the graph in the figure is a planar graph.
 - c. Prove that Euler’s formula holds for the graph of part (b) when drawn in its planar form.
 - d. Prove that a connected planar graph with parallel arcs or loops obeys Euler’s formula when drawn in its planar form.



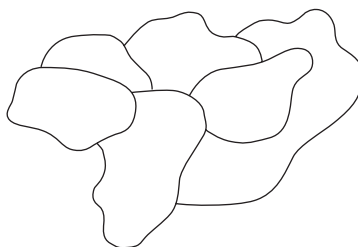
Exercises 77–86 refer to the problem of *graph colorability*. The origin of graph-coloring problems is a *map-coloring problem*: Suppose that a map of various countries, drawn on a sheet of paper, is to be colored so that no two countries with a common border have the same color. (We need not worry about countries that meet only at a point, and we will assume that each country is “connected.”) What is the minimum number of colors required to carry out this task for any map?

77. Show that a coloring of the accompanying map requires three colors and no more than three colors.

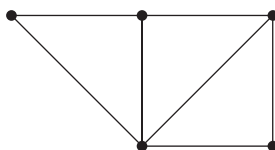


78. Draw a map that requires four colors.

79. Associated with any map is a graph, called the *dual graph* for the map, formed as follows: Put one node in each region of the map and an arc between two nodes representing adjacent countries.
- Draw the dual graph for the map of Exercise 77.
 - Draw the dual graph for the following map.

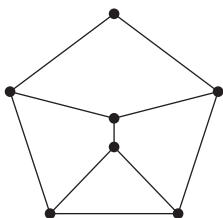


- Draw a map for which the following graph would serve as the dual.

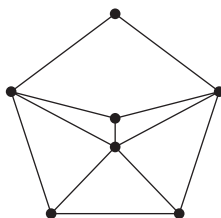


80. A *coloring* of a graph is an assignment of a color to each node of the graph in such a way that no two adjacent nodes have the same color. The *chromatic number* of a graph is the smallest number of colors needed to achieve a coloring. Find the chromatic number of the following graphs.

a.



b.

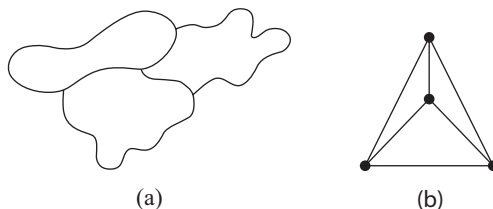


81. At least four colors are required to solve the general map-coloring problem (see Exercise 78). Because no one could produce a map requiring more than four colors, the conjecture was formulated that four colors are indeed sufficient. This conjecture became known as the *four-color problem*. It was first proposed to the mathematician Augustus De Morgan by one of his students in 1852, and it subsequently received much attention. It remained unproved, however, for over a hundred years. In 1976 two mathematicians at the University of Illinois, Wolfgang Haken and Kenneth Appel, used a computer to work through a large number of cases in a proof by contradiction, thus verifying the four-color conjecture.

The dual graph for a map (see Exercise 79), by the way it is constructed, will always be simple, connected, and planar. In addition, any simple, connected, planar graph can be viewed as the dual graph of a map. Restate the four-color conjecture in terms of the chromatic number (see Exercise 80) of a graph.

82. Prove that in a simple, connected, planar graph with three or more nodes, there is at least one node with degree less than or equal to 5. (*Hint*: Use a proof by contradiction.)
83. (Challenging problem) The *five-color theorem* states that the chromatic number for any simple, connected, planar graph is at most 5. While the four-color theorem (Exercise 81) is very difficult to prove, the five-color theorem can be proved by induction on the number of nodes in the graph. Prove the five-color theorem, making use of the result in Exercise 82.

84. The *six-color theorem* can be proved as a map-coloring problem without using the dual graph. Instead of creating the dual graph, put nodes at the intersections of boundaries and straighten the boundaries of regions so that the problem of coloring the map shown in figure (a) is represented by the problem of coloring the enclosed regions of the graph in figure (b). First assume that no country has a hole in it. Then the graph will be loop-free, planar, and connected. Also, every node will have degree at least 3.



- Show that the graph can be assumed to be simple by proving that if six colors are sufficient for coloring a simple graph, they are sufficient for a graph with parallel arcs as well. (*Hint*: Use temporary small countries at nodes.)
 - Prove that in a simple, connected, planar graph with R enclosed regions, $n - a + R = 1$.
 - Consider a simple, connected, planar graph and assume that every enclosed region has at least six edges adjacent to it. Show that $2a \leq 3n - 3$.
 - Now consider a simple, connected, planar graph where every node has degree at least 3. Show that such a graph has at least one enclosed region with no more than five adjacent edges.
 - Prove that six colors are sufficient to color any planar map where no country has a hole in it.
 - Prove that six colors are sufficient to color any planar map. (*Hint*: Cut some temporary slits in the map.)
85. Five political lobbyists are visiting seven members of Congress (labeled A through G) on the same day. The members of Congress the five lobbyists must see are
- A, B, D
 - B, C, F
 - A, B, D, G
 - E, G
 - D, E, F

Each member of Congress will be available to meet with lobbyists for one hour. What is the minimum number of time slots that must be used to set up the one-hour meetings so that no lobbyist has a conflict? (*Hint*: Treat this as a graph-coloring problem.) What if lobbyist 3 discovers that she does not need to see B and lobbyist 5 discovers that he does not need to see D ?

86. In a multiprocessor machine, six processors labeled A through F share blocks in a common data store. Two processors cannot simultaneously write to the same block. The following table shows which processors will write to the data store at the same time. How many distinct blocks are needed? (*Hint*: Treat this as a graph-coloring problem.)

A, F, C
 B, D
 F, D, A
 B, E
 F, C, E

SECTION 6.2 TREES AND THEIR REPRESENTATIONS

Tree Terminology

A special type of graph called a *tree* turns out to be a very useful representation of data.

DEFINITION TREE

A **tree** is an acyclic, connected graph with one node designated as the **root** of the tree.

Figure 6.31 pictures two trees. Perversely, computer scientists like to draw trees with the root at the top. An acyclic connected graph with no designated root node is called a **nonrooted** tree or a **free tree**. (Again, terminology is nonstandard. Some books define trees as acyclic, connected graphs and then call them “rooted trees” when there is a designated root node.)

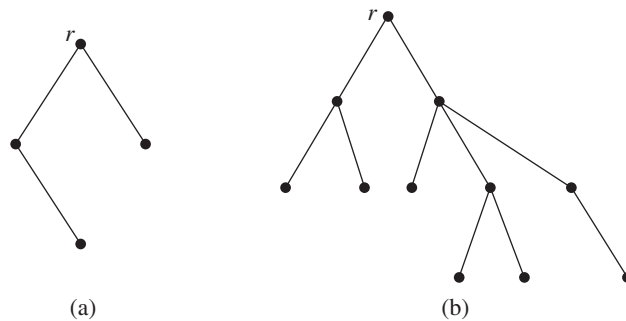


Figure 6.31

A tree can also be defined recursively. A single node is a tree (with that node as its root). If T_1, T_2, \dots, T_t are disjoint trees with roots r_1, r_2, \dots, r_t , the graph formed by attaching a new node r by a single arc to each of r_1, r_2, \dots, r_t , is a tree with root r . The nodes r_1, r_2, \dots, r_t are **children** of r , and r is a **parent** of r_1, r_2, \dots, r_t . Figure 6.32 shows the final step in the recursive construction of the tree in Figure 6.31b. It is often helpful to process a tree structure by working with it recursively, treating the subtrees as smaller tree objects.

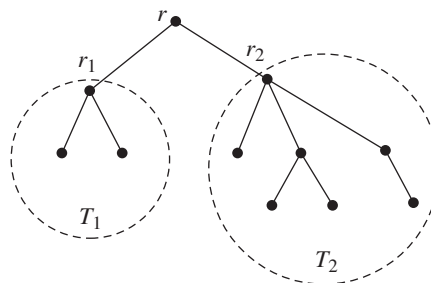


Figure 6.32

Because a tree is a connected graph, there is a path from the root to any other node in the tree; because the tree is acyclic, that path is unique. The **depth of a node** in a tree is the length of the path from the root to the node; the root itself has depth 0. The **depth (height) of the tree** is the maximum depth of any node in the tree; in other words, it is the length of the longest path from the root to any node. A node with no children is called a **leaf** of the tree; all non-leaves are **internal nodes**. A **forest** is an acyclic graph (not necessarily connected); thus a forest is a disjoint collection of trees. Figures 6.31a and 6.31b together form a forest.

Binary trees, where each node has at most two children, are of particular interest. In a binary tree, each child of a node is designated as either the **left child** or the **right child**. A **full binary tree** occurs when all internal nodes have two children and all leaves are at the same depth. Figure 6.33 shows a binary tree of height 4, and Figure 6.34 shows a full binary tree of height 3. A **complete binary tree** is an almost full binary tree; the bottom level of the tree is filling from left to right but may not have its full complement of leaves. Figure 6.35 shows a complete binary tree of height 3. (Note that while a tree is a graph, a complete tree is not a complete graph!)

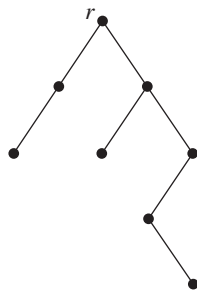


Figure 6.33

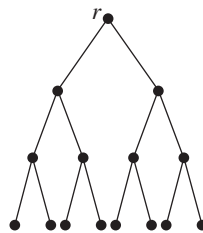


Figure 6.34

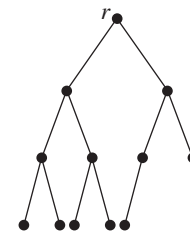


Figure 6.35

PRACTICE 18

Answer the following questions about the binary tree shown in Figure 6.36. (Assume that node 1 is the root.)

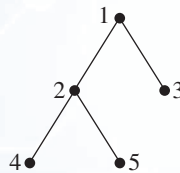


Figure 6.36

- What is the height?
- What is the left child of node 2?
- What is the depth of node 5?



Applications of Trees

Decision trees were used to solve counting problems in Chapter 4 and will be used in Section 6.3 to help establish lower bounds on the work for certain algorithms. Exercise 43 of Section 5.1 describes the organization of data into a binary tree structure. By using these trees, a collection of records can be efficiently searched to locate a particular record or to determine that a record is not in the collection. Examples of such a search would be checking for a volume in a library, for a patient's medical record in a hospital, or for an individual's credit record at the bank. We will also look at binary tree search in Section 6.3. The derivations of words in certain formal languages will be shown as trees in Chapter 9 (these are the parse trees generated by a compiler while analyzing a computer program).

A family tree is usually, indeed, a tree, although if there were intermarriages, it would be a graph but not a tree in the technical sense. (Information obtained from a family tree is not only interesting but also useful for research in medical genetics.) The organization chart indicating who reports to whom in a large company or other enterprise is usually a tree (Figure 6.37).

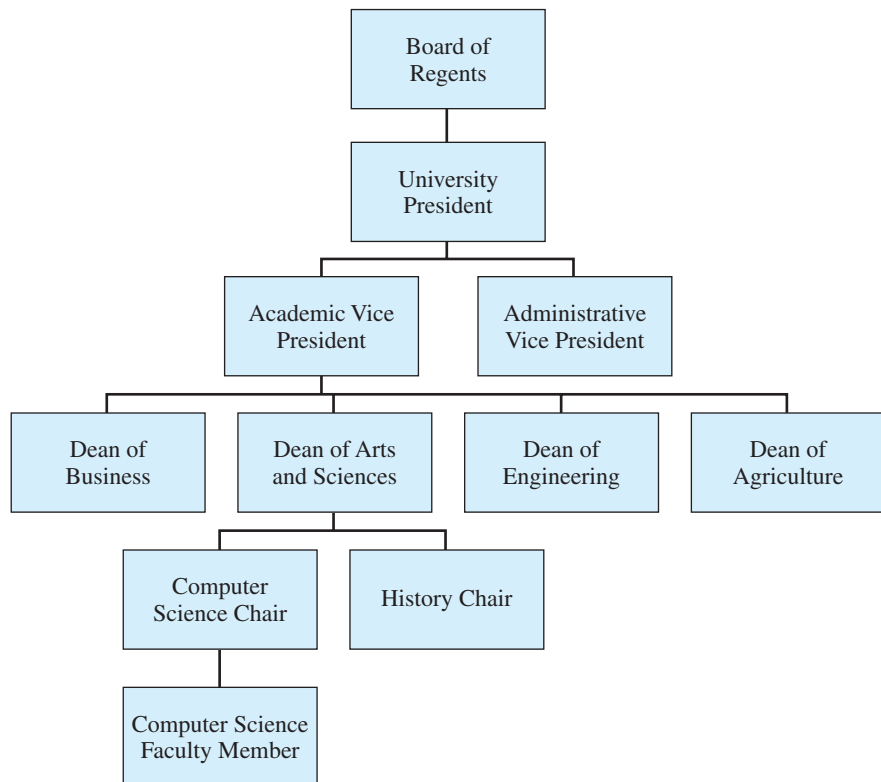


Figure 6.37

Folders and files on your computer are organized in a hierarchical (treelike) structure. In Figure 6.38, the CSCI 34000 tree (folder) has been expanded to show two subtrees; the Web Materials subtree has been expanded to show additional folders, the Activities subtree has not been expanded.

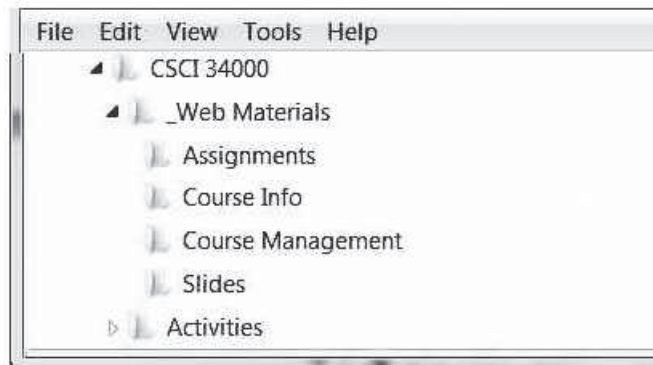


Figure 6.38

EXAMPLE 22

A computer virus is spread via e-mail. Each second, 4 new machines are infected. A 4-ary tree structure (Figure 6.39) represents the spread of the virus. By the multiplication principle, 4^n machines have been infected after n seconds.

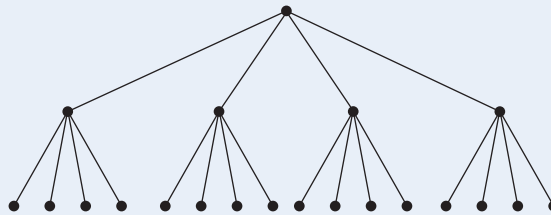


Figure 6.39

EXAMPLE 23

Algebraic expressions involving binary operations can be represented by labeled binary trees. The leaves are labeled as operands, and the internal nodes are labeled as binary operations. For any internal node, the binary operation of its label is performed on the expressions associated with its left and right subtrees. Thus the binary tree in Figure 6.40 represents the algebraic expression $(2 + x) - (y * 3)$.

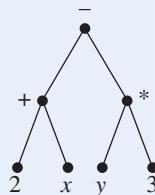


Figure 6.40

PRACTICE 19 What is the expression tree for $(2 + 3) * 5$?

Binary Tree Representation

Because a tree is also a graph, the representations discussed in Section 6.1 for graphs in general can also be used for trees. Binary trees, however, have special characteristics that we want to capture in the representation, namely, the identity of the left and right child. The equivalent of an adjacency matrix is a two-column array (or an array of records) where the data values for each node are the left and right child of that node. The equivalent of the adjacency list representation is a collection of records with three fields containing, respectively, the current node, a pointer to the record for the left-child node, and a pointer to the record for the right-child node.

EXAMPLE 24

For the binary tree shown in Figure 6.41, the left child-right child array representation is given in Figure 6.42a. Zeros again indicate null pointers. The pointer representation is given in Figure 6.42b.

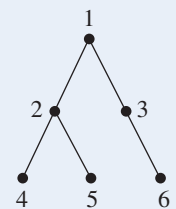
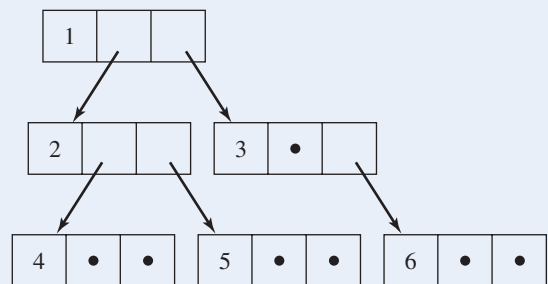


Figure 6.41

	Left child	Right child
1	2	3
2	4	5
3	0	6
4	0	0
5	0	0
6	0	0

(a)



(b)

Figure 6.42

PRACTICE 20 For the binary tree in Figure 6.43

- Give the left child-right child array representation.
- Give the pointer representation.

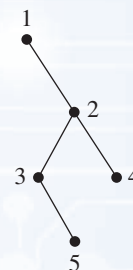


Figure 6.43

Tree Traversal Algorithms

If a tree structure is being used to store data, it is often helpful to have a systematic mechanism for writing out the data values stored at all the nodes, which can be accomplished by *traversing* the tree, that is, visiting each of the nodes in the tree structure. The three common **tree traversal** algorithms are *preorder*, *inorder*, and *postorder* traversal.

In the three traversal methods, it is helpful to use the recursive view of a tree, where the root of a tree has branches down to roots of subtrees. We will therefore assume that a tree T has a root r ; any subtrees are labeled left to right as T_1, T_2, \dots, T_t (Figure 6.44). Because we are using a recursive definition of a tree, it will be easy to state the tree traversal algorithms in recursive form.

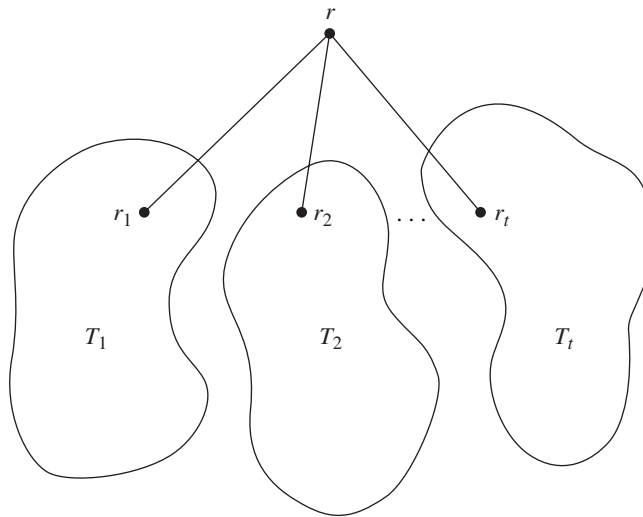


Figure 6.44

The terms *preorder*, *inorder*, and *postorder* refer to the order in which the root of a tree is visited compared to the subtree nodes. In **preorder traversal**, the root of the tree is visited first and then the subtrees are processed left to right, each in preorder.

ALGORITHM **PREORDER**

```

Preorder(tree  $T$ )
//Writes the nodes of a tree with root  $r$  in preorder

    write( $r$ )
    for  $i = 1$  to  $t$  do
        Preorder( $T_i$ )
    end for
end Preorder

```

In **inorder traversal**, the left subtree is processed by an inorder traversal, then the root is visited, and then the remaining subtrees are processed from left to right, each in inorder. If the tree is a binary tree, the result is that the root is visited between processing of the two subtrees. In a nonbinary tree, if there is a single subtree joined to its parent by a vertical arc, that is considered the left subtree and there are no additional subtrees.

ALGORITHM *INORDER*

```

Inorder(tree  $T$ )
//Writes the nodes of a tree with root  $r$  in inorder

    Inorder( $T_1$ )
    write( $r$ )
    for  $i = 2$  to  $t$  do
        Inorder( $T_i$ )
    end for
end Inorder

```

Finally, in **postorder traversal**, the root is visited last, after all subtrees have been processed from left to right in postorder.

ALGORITHM *POSTORDER*

```

Postorder(tree  $T$ )
//Writes the nodes of a tree with root  $r$  in postorder

    for  $i = 1$  to  $t$  do
        Postorder( $T_i$ )
    end for
    write( $r$ )
end Postorder

```

EXAMPLE 25

For the binary tree of Figure 6.45, the preorder traversal algorithm (root, left, right) says to write the root first, a , and then process the left subtree. At the left subtree, rooted at b , a preorder traversal writes the root, b , and moves again to the left subtree, which is the single node d . This single node is the root of a tree, so it is written out. Then d 's left subtree (empty) and d 's right subtree (empty) are traversed. Backing up to the tree rooted at b , its left subtree has been traversed, so now the right subtree is traversed, producing node e . The subtree rooted at

b has now been completely traversed. Backing up to a , it is time to traverse a 's right subtree. A preorder traversal of the tree rooted at c causes c to be written, then traversal goes to c 's left subtree, which results in f , h , and i being written. Backing up to c , traversing c 's right subtree produces g . The subtree rooted at c has now been completely traversed, and the algorithm terminates. The preorder traversal produced

$a, b, d, e, c, f, h, i, g$

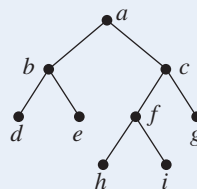


Figure 6.45

REMINDER

For a binary tree:

Preorder traversal is root, left, right.

Inorder traversal is left, root, right.

Postorder traversal is left, right, root.

EXAMPLE 26

Using the tree of Figure 6.45 again, an inorder traversal (left, root, right), travels down to the farthest left subtree, rooted at d . An inorder traversal here traverses the left subtree (empty), writes out the root, d , then traverses the right subtree (empty). Backing up the tree to b , b 's left subtree has been traversed, so it is time to write out the root, b . Proceeding then to b 's right subtree, e is written. Backing up to a , a 's left subtree has been traversed, so the root, a , is written out. Proceeding to a 's right subtree, an inorder traversal says to go to the farthest left subtree first, which would cause the h to be written. After that, f and i are written, then the root c , then the right subtree of c , which is g . The nodes are therefore written as

$d, b, e, a, h, f, i, c, g$

A postorder traversal (left, right, root) would produce

$d, e, b, h, i, f, g, c, a$

EXAMPLE 27

Consider the tree shown in Figure 6.46, which is not a binary tree. A preorder traversal first writes the root a and then does a preorder traversal on the left subtree, rooted at b . The preorder traversal of this subtree writes out b and then proceeds to a preorder traversal of the left subtree of b , which is rooted at d . Node d is written and then a preorder traversal of the left (the only) subtree of d , which is rooted at i , is done. After writing out i , the traversal backs up to consider any other subtrees of d ; there are none.

Backing up to b , there are other subtrees of b . Processing these left to right, nodes e and then f are written. All the subtrees of b have now been traversed; backing up to node a to look for subtrees farther to the right reveals a subtree rooted at c . The algorithm writes out the root c , then moves to its leftmost subtree rooted at

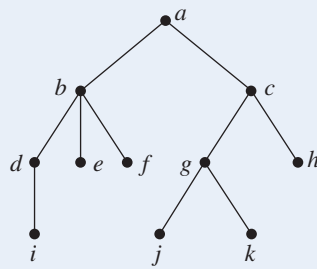


Figure 6.46

g and writes out g . Processing the subtrees of g , the nodes j and k are written; then backing up to c , its remaining subtree is processed, producing h . Node c has no other subtrees; backing up to a , a has no other subtrees, and the algorithm terminates. The list of nodes in preorder traversal is

$$a, b, d, i, e, f, c, g, j, k, h$$

To do an inorder traversal of the tree in Figure 6.46, process left subtrees first, leading down to node i , which has no subtrees. Therefore i is written out. Backing up to d , the left (only) subtree of d has been traversed, so d is written out. Since node d has no further subtrees, the algorithm backs up to b . The left subtree of b has been processed, so b is written out and then its remaining subtrees are traversed, writing out e and f . Backing up to a , a is written out, and then the right subtree of a is processed, leading to nodes j , g , k , c , and h , in that order, and we are done. Thus the inorder list of nodes is

$$i, d, b, e, f, a, j, g, k, c, h$$

The following list of nodes results from a postorder traversal:

$$i, d, e, f, b, j, k, g, h, c, a$$

PRACTICE 21 Do a preorder, inorder, and postorder traversal of the tree in Figure 6.47.

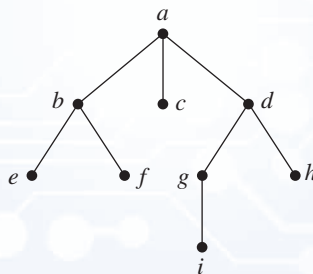


Figure 6.47

EXAMPLE 28

Example 23 showed how algebraic expressions can be represented as binary trees. If we do an inorder traversal of the expression tree, we retrieve the original algebraic expression. For the expression tree of Figure 6.48, for example, an inorder traversal gives the expression

$$(2 + x) * 4$$

where the parentheses are added as we complete the processing of a subtree. This form of an algebraic expression, where the operation symbol appears between the two operands, is called **infix notation**. Parentheses are necessary here to indicate the order of operations. Without parentheses, the expression becomes $2 + x * 4$, which is also an infix expression but, because of the order of precedence of multiplication over addition, is not what is intended.

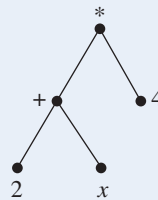


Figure 6.48

A preorder traversal of Figure 6.48 gives the expression

$$* + 2 x 4$$

Here the operation symbol precedes its operands. This form of an expression is called **prefix notation**, or **Polish notation**.² The expression can be translated into infix form as follows:

$$* + 2 x 4 \rightarrow * (2 + x) 4 \rightarrow (2 + x) * 4$$

A postorder traversal gives the expression

$$2 x + 4 *$$

where the operation symbol follows its operands. This form of an expression is called **postfix notation**, or **reverse Polish notation** (or just **RPN**). The expression can be translated into infix form as follows:

$$2 x + 4 * \rightarrow (2 + x) 4 * \rightarrow (2 + x) * 4$$

Neither prefix nor postfix form requires parentheses to avoid ambiguity. These notations therefore provide more efficient, if less familiar, representations of algebraic expressions than infix notation. Such forms can be evaluated sequentially, with no need for “look-ahead” to locate parenthesized expressions. Compilers often change algebraic expressions in computer programs from infix to postfix notation for more efficient processing. ●

²Named for the Polish logician J. Lukasiewicz, who first used it.

PRACTICE 22 Write the expression tree for

$$a + (b * c - d)$$

and write the expression in prefix and postfix notation. ■

Results about Trees

Trees are fertile ground (no pun intended) for proofs by induction, either on the number of nodes or arcs or on the height.

EXAMPLE 29

After drawing a few trees and doing some counting, it appears that the number of arcs in a tree is always one less than the number of nodes. More formally, it appears that

A tree with n nodes has $n - 1$ arcs.

We will prove this statement using induction on n , $n \geq 1$. For the base case, $n = 1$, the tree consists of a single node and no arcs (Figure 6.49), so the number of arcs is 1 less than the number of nodes.

● $n = 1, a = 0$

Figure 6.49

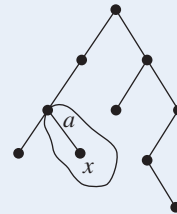


Figure 6.50

Assume that any tree with k nodes has $k - 1$ arcs, and consider a tree with $k + 1$ nodes. We want to show that this tree has k arcs. Let x be a leaf of the tree (a leaf must exist since the tree is finite). Then x has a unique parent. Remove from the tree the node x and the single arc a connecting x and its parent (Figure 6.50).

The remaining graph is still a tree and has k nodes. Therefore, by the inductive hypothesis, it has $k - 1$ arcs, and the original graph, containing arc a , had $(k - 1) + 1 = k$ arcs. The proof is complete. ●

Notice that in the inductive proof of Example 29 we had to support the proof with many more words than we used in some of our early inductive proofs. In Example 15 in Chapter 2, for instance, the inductive proof that

$$1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$$

consisted mainly of manipulating the mathematical expressions in this equation, but now we have to do more verbal reasoning. Words are not only acceptable in a proof, they may form the major part of the proof.

The inductive proof of Example 29 differs in another way from proofs like that of Example 15 in Chapter 2. In those proofs, there was always a single term in the series (the last term) whose removal would lead to the “ $P(k)$ case,” the inductive hypothesis. In proofs involving trees with $k + 1$ nodes, which node should be removed to generate the $P(k)$ case? Usually the node to remove is not unique, but it is not completely arbitrary either. In the proof of Example 29, for instance, removing a non-leaf node (and the arcs attached to it) from a tree with $k + 1$ nodes would result in a graph with k nodes, but not a tree with k nodes, so the inductive hypothesis would not apply.

PRACTICE 23

Prove that in any tree with n nodes, the total number of arc ends is $2n - 2$. Of course this result follows directly from Example 29 because $n - 1$ arcs means $2(n - 1) = 2n - 2$ arc ends, but do a proof by induction on the number of nodes. ■

EXAMPLE 30

Recall that a tree T can also be constructed recursively by hooking a root node to a collection of subtrees T_1, \dots, T_t (see Figure 6.44). This allows us to use structural induction, discussed in Chapter 3, to prove certain results about trees. We'll use structural induction to prove the result of Example 29, namely

A tree with n nodes has $n - 1$ arcs.

The base case where $n = 1$ is the same as before. Assume that any subtree T_i with n_i nodes has $n_i - 1$ arcs. The tree T constructed from subtrees T_1, \dots, T_t has n nodes where

$$n = 1 + \sum_{i=1}^t n_i$$

(The extra 1 is the root node.) Also, T has a arcs where

$$a = t + \sum_{i=1}^t a_i$$

(The extra t counts the arcs from the root of T to the t subtrees.)

By the inductive hypothesis, $a_i = n_i - 1$ for each subtree, so

$$\begin{aligned} a &= t + \sum_{i=1}^t a_i = t + \sum_{i=1}^t (n_i - 1) = t + \sum_{i=1}^t n_i - \sum_{i=1}^t 1 = t + \sum_{i=1}^t n_i - t \\ &= \sum_{i=1}^t n_i = n - 1 \end{aligned}$$

PRACTICE 24

Prove that in any tree with n nodes, the total number of arc ends is $2n - 2$. Use structural induction. ■

EXAMPLE 31

Sometimes a clever observation can take the place of an inductive proof. Every arc of a tree connects one node to its parent. Each node of the tree except the root has a parent, and there are $n - 1$ such nodes, therefore $n - 1$ arcs. Each arc has two arc ends, so there are $2(n - 1)$ arc ends. ●

SECTION 6.2 REVIEW**TECHNIQUES**

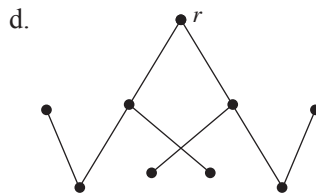
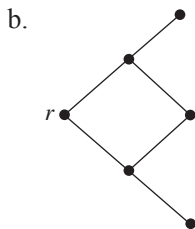
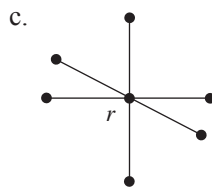
- Construct expression trees.
- Construct array and pointer representations for binary trees.
- W Conduct preorder, inorder, and postorder traversals of a tree.

MAIN IDEAS

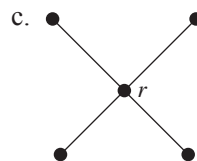
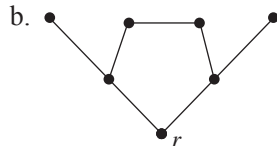
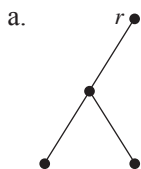
- Binary trees can be represented by arrays and by linked structures.
- Recursive procedures exist to systematically visit every node of a tree.

EXERCISES 6.2

1. Which of the following graphs are trees with root r ? If a graph is a tree, draw it in a more conventional way. If not, say what property fails.



2. Which of the following graphs are binary trees with root r ? If the graph is not a binary tree, say what property fails.



3. Sketch a picture of each of the following trees.

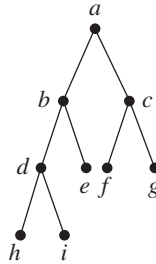
a. Tree with five nodes and depth 1

b. Full binary tree of depth 2

c. Tree of depth 3 where each node at depth i has $i + 1$ children

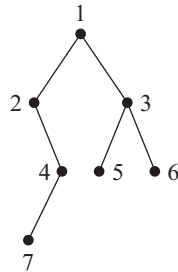
4. Answer the following questions about the accompanying graph with node *a* as the root.

- Is it a binary tree?
- Is it a full binary tree?
- Is it a complete binary tree?
- What is the parent of *e*?
- What is the right child of *e*?
- What is the depth of *g*?
- What is the height of the tree?

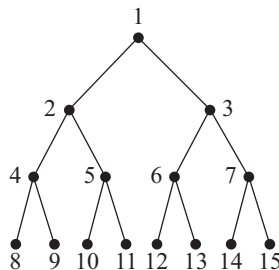


In Exercises 5–8, draw the expression tree.

- $[(2 * x - 3 * y) + 4 * z] + 1$
- $[(x - 2) * 3] + (5 + 4)$
- $1 - (2 - [3 - (4 - 5)])$
- $[(6/2) * 4] + [(1 + x) * (5 + 3)]$
- Write the left child–right child array representation for the binary tree in the figure.



10. Write the left child–right child array representation for the binary tree in the figure.



11. Draw the binary tree corresponding to the left child–right child representation that follows. (1 is the root.)

	Left child	Right child
1	2	3
2	4	0
3	5	0
4	6	7
5	0	0
6	0	0
7	0	0

12. Draw the binary tree corresponding to the left child–right child representation that follows. (1 is the root.)

	Left child	Right child
1	2	0
2	3	4
3	0	0
4	5	6
5	0	0
6	0	0

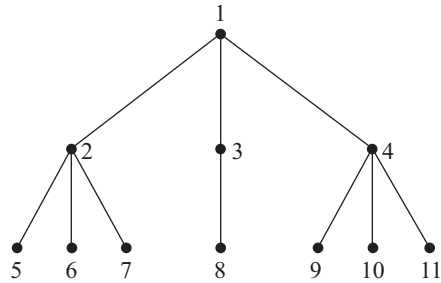
13. Write the left child–right child array representation for the binary search tree that is created by processing the following list of words: “All Gaul is divided into three parts” (see Exercise 43 of Section 5.1). Also store the name of each node.
14. Write the left child–right child array representation for the binary search tree that is created by processing the following list of words: “We hold these truths to be self-evident, that all men are created equal” (see Exercise 43 of Section 5.1). Also store the name of each node.
15. In the following binary tree representation, the left child and parent of each node are given. Draw the binary tree. (1 is the root.)

	Left child	Parent
1	2	0
2	4	1
3	0	1
4	0	2
5	0	2
6	0	3

16. The following represents a tree (not necessarily binary) where, for each node, the leftmost child and the closest right sibling of that node are given. Draw the tree. (1 is the root.)

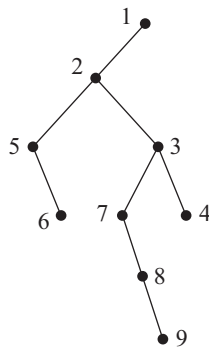
	Left child	Right sibling
1	2	0
2	5	3
3	0	4
4	8	0
5	0	6
6	0	7
7	0	0
8	0	0

17. a. For the following tree, write the leftmost child–right sibling array representation described in Exercise 16.



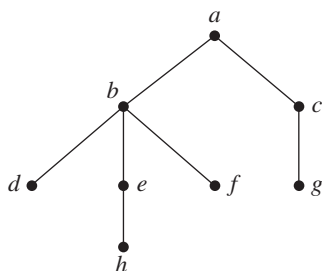
- b. Now draw the binary tree that results from treating the answer to part (a) as a left child–right child binary tree representation. An arbitrary tree can thus be thought of as having a binary tree representation.

18. The following binary tree is the representation of a general tree (as in part (b) of Exercise 17). Draw the tree.

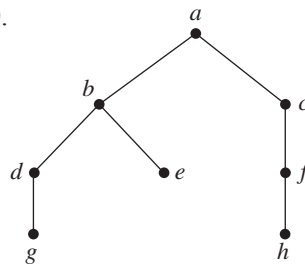


For Exercises 19–24, write the list of nodes resulting from a preorder traversal, an inorder traversal, and a postorder traversal of the tree.

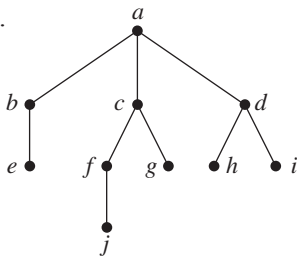
19.



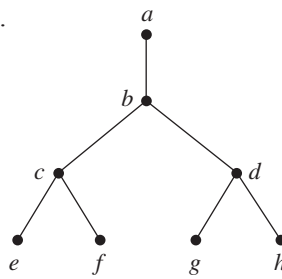
20.



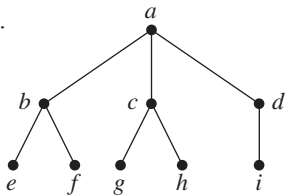
21.



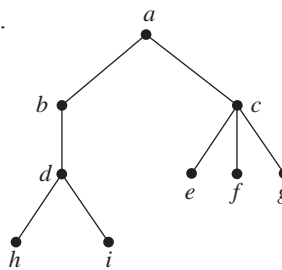
23.



22.



24.

25. Write in prefix and postfix notation: $3/4 + (2 - y)$.26. Write in prefix and postfix notation: $(x * y + 3/z) * 4$.27. Write in infix and postfix notation: $- * + 2 3 * 6 x 7$.28. Write in infix and postfix notation: $- + - x y z w$.29. Write in prefix and infix notation: $4 7 x - * z +$.30. Write in prefix and infix notation: $x 2 w + y z * - /$.31. Evaluate the postfix expression $8 2 / 2 3 * +$.32. Evaluate the postfix expression $5 3 + 1 3 + / 7 *$.

33. Draw a single tree whose preorder traversal is

 a, b, c, d, e

and whose inorder traversal is

 b, a, d, c, e

34. Draw a single tree whose inorder traversal is

 $f, a, g, b, h, d, i, c, j, e$

and whose postorder traversal is

 $f, g, a, h, i, d, j, e, c, b$

35. Find an example of a tree whose inorder and postorder traversals yield the same list of nodes.

36. Find two different trees that have the same list of nodes under a preorder traversal.

37. Informally describe a recursive algorithm to compute the height of a binary tree, given the root node.

38. Informally describe a recursive algorithm to compute the number of nodes in a binary tree, given the root node.

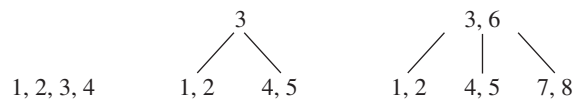
39. Prove that a simple graph is a nonrooted tree if and only if there is a unique path between any two nodes.

40. What is the minimum number of nodes and arcs that need to be deleted to reduce a full binary tree of height ≥ 2 to a forest of 4 binary trees?
41. Let G be a simple graph. Prove that G is a nonrooted tree if and only if G is connected and if the removal of any single arc from G makes G unconnected.
42. Let G be a simple graph. Prove that G is a nonrooted tree if and only if G is connected and the addition of one arc to G results in a graph with exactly one cycle.
43. Prove that a binary tree has at most 2^d nodes at depth d .
44. Prove that a tree with n nodes, $n \geq 2$, has at least two nodes of degree 1.
45. a. Draw a full binary tree of height 2. How many nodes does it have?
 b. Draw a full binary tree of height 3. How many nodes does it have?
 c. Conjecture how many nodes there are in a full binary tree of height h .
46. Prove your conjecture from Exercise 45(c) three different ways.
 a. Use induction on the height h of the full binary tree. (*Hint*: Use Exercise 43.)
 b. Add up the nodes at each level of the tree (*Hint*: Use Exercise 43).
 c. Use structural induction.
47. a. Prove that a full binary tree with x internal nodes has $2x + 1$ total nodes.
 b. Prove that a full binary tree with x internal nodes has $x + 1$ leaves.
 c. Prove that a full binary tree with n nodes has $(n - 1)/2$ internal nodes and $(n + 1)/2$ leaves.
48. Prove that the number of leaves in any binary tree is 1 more than the number of nodes with two children.
49. Find an expression for the height of a complete binary tree with n nodes. (*Hint*: Use Exercise 45.)
50. Prove that in the pointer representation of a binary tree with n nodes there are $n + 1$ null pointers. (*Hint*: Use Exercise 48).
51. Find the chromatic number of a tree (see Section 6.1, Exercise 80).
52. Let E be the external path length of a tree, that is, the sum of the path lengths to all the leaves. Let I be the internal path length, that is, the sum of the path lengths to all the internal nodes. Let i be the number of internal nodes. Prove that in a binary tree where all internal nodes have two children, $E = I + 2i$.
53. Let $B(n)$ represent the number of different binary trees with n nodes.
 a. Define $B(0)$ to have the value 1 (there is one binary tree with 0 nodes). Prove that $B(n)$ is given by the recurrence relation

$$B(1) = 1$$

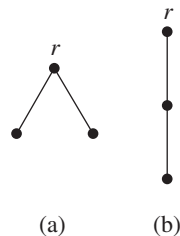
$$B(n) = \sum_{k=0}^{n-1} B(k)B(n-1-k)$$

- b. Compare the sequence $B(n)$ to the sequence of Catalan numbers (Exercise 97, Section 4.4). Write the closed-form expression for $B(n)$.
- c. Compute the number of different binary trees with 3 nodes. Draw all these trees.
- d. Compute the number of different binary trees with 6 nodes.
54. In the data structure known as a B -tree of order 5, each node of the tree can contain multiple data values, maintained in sorted order. Between and around the data values at an internal node are arcs that lead to children of the node. New data values are inserted into the leaf nodes of the tree, but when a leaf (or internal node) gets up to five values, it splits in two and the median value pops up to the next level of the tree. The figure shows the tree at various points as the data values 1 through 8 are inserted into an initially empty tree.

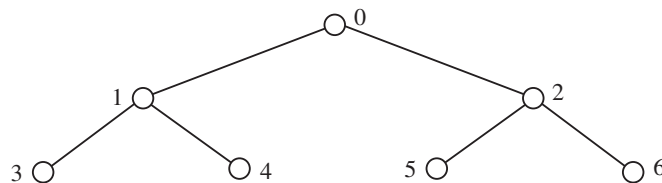


- The minimum number of data values to insert into a B-tree of order 5 to force it to have two levels is 5. Find the minimum number of data values required to force the tree to have three levels.
- Prove that when a B-tree of order 5 has the minimum number of data values to force it to have n levels, $n \geq 2$, the bottom level contains $2(3^{n-2})$ nodes.
- Find (and justify) a general expression for the minimum number of data values required to force a B-tree of order 5 to have n levels. (Hint: $3^0 + 3^1 + \dots + 3^{n-2} = \left(\frac{3^n - 3}{6}\right)$.)

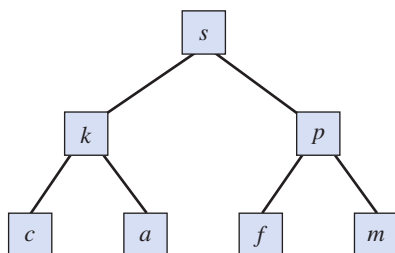
In Exercises 55 and 56, two trees are *isomorphic* if there is a bijection $f: N_1 \rightarrow N_2$, where f maps the root of one tree to the root of the other and where $f(y)$ is a child of $f(x)$ in the second tree when y is a child of x in the first tree. Thus the two trees shown are isomorphic graphs but not isomorphic trees (in part (a) the root has two children and in part (b) it does not). These are the only two nonisomorphic trees with three nodes.



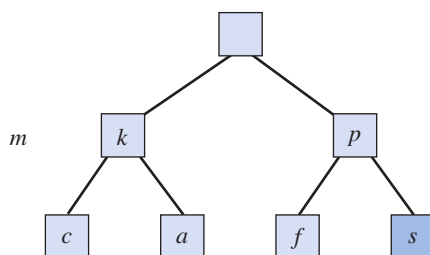
- Draw all the nonisomorphic trees with four nodes.
- Draw all the nonisomorphic trees with five nodes.
- One of the most efficient sorting algorithms is *HeapSort*, which sorts an array of values into increasing order. To understand how the *HeapSort* algorithm works, it is best to imagine that the array elements are stored in level order as the nodes of a binary tree. Thus the values in a 7-element array that is indexed from 0 through 6 would be stored in a binary tree with element 0 at the root, elements 1 and 2 at depth 1, and so on.



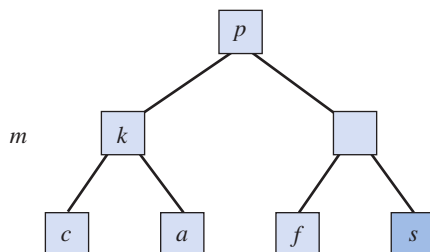
A *heap* is a binary tree in which the value at every node is greater than the value at its two child nodes. *HeapSort* is a two-phase process. The first phase is to reorganize the tree elements into a heap (more on this later), and the second is to sort the heap. The key idea is that in a heap, the largest element is the root of the tree; its proper place in the sorted array is at the end of the unsorted section of the array (at the lowest, rightmost tree element not yet in its sorted position). The tree root gets thrown to the last unsorted position, and the element that formerly occupied that position must be inserted back into the unsorted section in such a way as to preserve the heap property. Consider the following binary tree, which is a heap—each node value is larger than the values at the two child nodes.



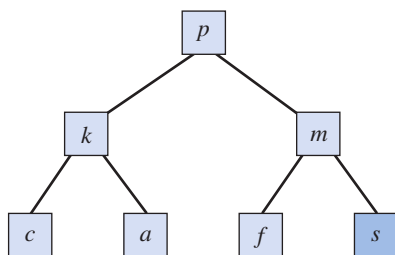
The root value, s , belongs at the bottom right of the tree (the position of the last array element), and once positioned there, s will never be looked at again. The previous value, m , has been temporarily displaced.



To insert m back into the heap, there is a series of “competitions” between the displaced element and the child nodes of the empty node so that the largest value is promoted to fill the empty slot. Here m , k , and p compete, and p is the winner.



Now m and f compete (remember that s is no longer considered), the largest value is m , and once m is inserted, the heap has been reestablished.



- Complete phase 2 of the process so that the array is sorted. (*Hint:* The next step is to throw the root value, p , into the position currently occupied by f . The displaced value f must then be reinserted into the tree.)
- For phase 1, to construct the heap, put the original random elements into the binary tree. Consider the leaf nodes to be temporarily in place, since their relative order does not matter. Beginning with the rightmost node in the level above the leaves and working up level-by-level to the root, temporarily displace that node element and then insert it using the “competition” rules described. Beginning with the array r, w, f, g, k, y, d , construct a heap.

SECTION 6.3 DECISION TREES

We used decision trees in Chapter 4 to solve counting problems. Figure 6.51 shows the tree used in Example 39 of Chapter 4 to represent the various possibilities for five coin tosses under the constraint that two heads in a row do not occur. Each internal node of the tree represents an action (a coin toss), and the arcs to the children of internal nodes represent the outcomes of that action (heads or tails). The leaves of the tree represent the final outcomes, that is, the different ways that five tosses could occur.

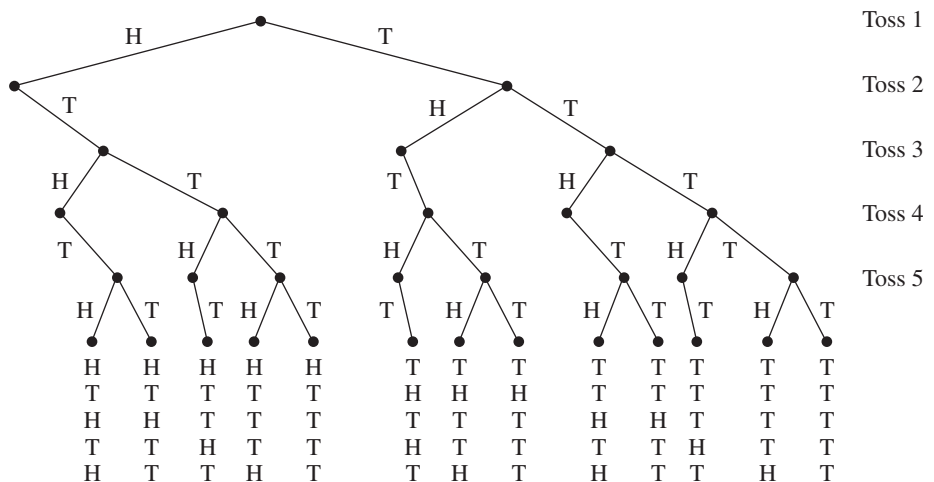


Figure 6.51

Although we have used decision trees, we haven't given a formal definition of what a decision tree is.

● **DEFINITION** **DECISION TREE**

A **decision tree** is a tree in which the internal nodes represent actions, the arcs represent outcomes of an action, and the leaves represent final outcomes.

Sometimes useful information can be obtained by using a decision tree to represent the activities of a real algorithm; actions that the algorithm performs take place at internal nodes, the children of an internal node represent the next action taken, based on the outcome of the previous action, and the leaves represent some sort of circumstance that can be inferred upon algorithm termination. Note that, unlike the trees we talked about in Section 6.2, a decision tree is not a data structure; that is, the nodes of the tree have no data values associated with them. Nor are the algorithms we are representing necessarily acting on a tree structure. In fact, we will use decision trees in this section to learn more about algorithms for searching and sorting, and these algorithms act on lists of data items.

Searching

A search algorithm either finds a target element x within a list of elements or determines that x is not in the list. Such an algorithm generally works by making

successive comparisons of x to the list items. We have already seen two such algorithms, sequential search and binary search. We can model the activities of these algorithms by using decision trees. The nodes represent the actions of comparing x to the list items, where the comparison of x to the i th element in the list is denoted by $x:L[i]$.

Sequential search only distinguishes between two possible outcomes of a comparison of x to $L[i]$. If $x = L[i]$, the algorithm terminates because x has been found in the list. If $x \neq L[i]$, the next comparison performed is $x:L[i + 1]$, regardless of whether x was less than or greater than $L[i]$. The leaves of this decision tree correspond to the final outcomes, where either x is one of the list items or x is not in the list.

EXAMPLE 32

Figure 6.52 shows the decision tree for the sequential search algorithm acting on a list of five elements.

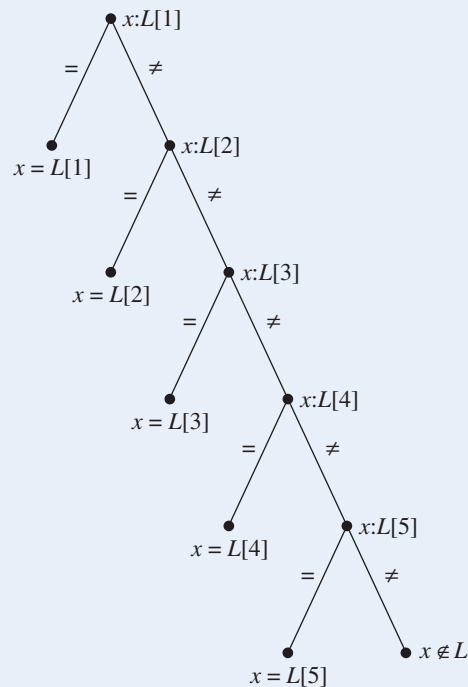


Figure 6.52

From the decision tree for a given search algorithm, we can see that the number of comparisons required to reach any particular outcome (leaf of the tree) is the number of internal nodes from the root to that leaf. This number equals the length of the path from the root to that leaf. The worst case, that is, the maximum number of comparisons, is the maximum length of any such path, which is the depth of the tree.

Because every decision tree for sequential search looks like Figure 6.52, it is clear that the depth of such a tree, for an n -element list, is n . This agrees with what we already know, namely, that the worst case for sequential search on a list of n elements is n .

The decision tree for the binary search algorithm is more interesting. Binary search acts on a sorted list and distinguishes between three possible outcomes of the comparison:

- $x = L[i]$: algorithm terminates, x has been found
- $x < L[i]$: algorithm proceeds to the left half of the list
- $x > L[i]$: algorithm proceeds to the right half of the list

We will follow the usual custom and not write the leaf that corresponds to the “middle branch,” $x = L[i]$ (see Exercise 21 for a discussion of the consequences of this convention). If $x < L[i]$, the next comparison the algorithm performs is found at the left child of this node; if $x > L[i]$, the algorithm’s next comparison is found at the right child. If no child exists, the algorithm terminates because x is not in the list. The tree we’ve described is a binary tree whose leaves represent all the possible outcomes where x is not in the list. There are many more failure leaves in binary search than in sequential search, because binary search indicates *how* x fails to be in the list (e.g., $x < L[1]$ or $L[1] < x < L[2]$).

EXAMPLE 33

Figure 6.53 shows the decision tree for the binary search algorithm acting on a sorted list of eight elements.

The worst case, that is, the maximum number of comparisons, will again be the depth of the tree, which is 4 in Figure 6.53. In Chapter 3 we solved a recurrence relation to get the worst-case behavior for binary search where n is a power of 2 and found this to be $1 + \log n$ (remember that we are using base 2 logarithms). Note that $1 + \log 8 = 4$, so the decision tree agrees with our previous result. The restriction of n to a power of 2 made the arithmetic of solving the recurrence relation simpler. If n is not a power of 2, then the depth of the tree is given by the expression $1 + \lceil \log n \rceil$.

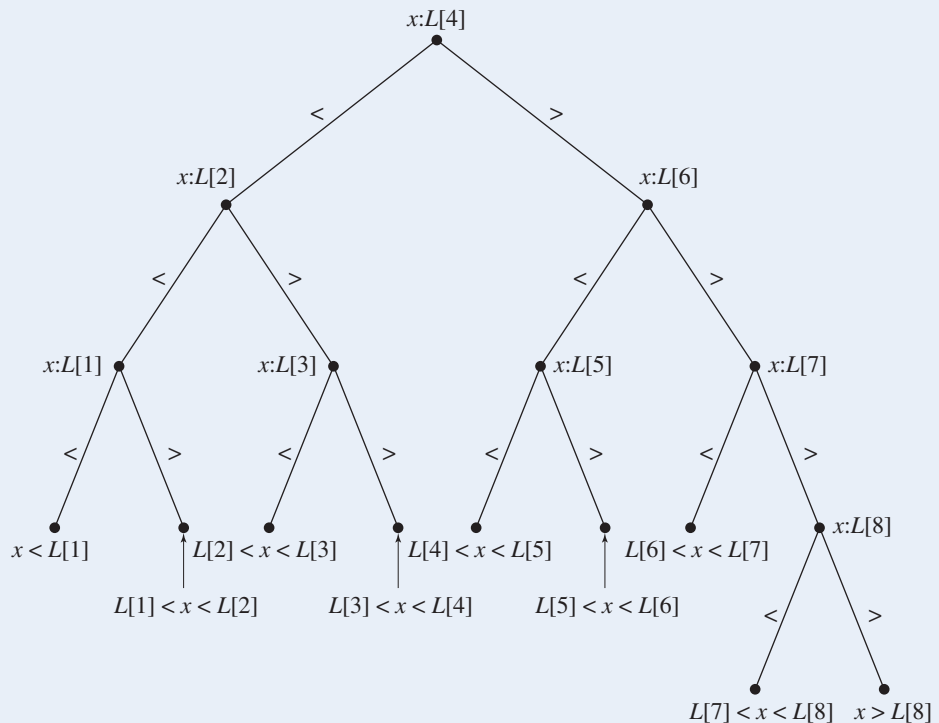


Figure 6.53

PRACTICE 25

- Draw the decision tree for the binary search algorithm on a sorted list of five elements.
- Find the depth of the tree and compare to $1 + \lfloor \log 5 \rfloor$. ■

Lower Bounds on Searching

We have used decision trees to represent the actions of two particular search algorithms. Such a tree could be used to represent the actions of any algorithm that solves the search problem by comparing the target element to the list elements. The internal nodes of such a tree would represent the comparisons done, and the depth of the tree would be the worst-case number of comparisons over all possible cases. What can be said about such a tree when we don't know the particulars of the algorithm involved? We can say that x must be compared to every element in the list at least once (perhaps more than once if the algorithm is quite stupid). For if there is some list element that escapes being compared to x , the algorithm cannot say whether that element equals x and thus cannot decide with certainty whether x belongs to the list. Comparisons are internal nodes in the decision tree. Therefore, if m is the number of internal nodes in the decision tree T_1 for any search algorithm acting on an n -element list, then $m \geq n$.

Before proceeding further with decision trees, we need some additional facts about binary trees in general. The number of nodes at each level in a *full* binary tree follows a geometric progression: 1 node at level 0, 2^1 nodes at level 1, 2^2 nodes at level 2, and so on. In a full binary tree of depth d , the total number of nodes is therefore

$$1 + 2 + 2^2 + 2^3 + \cdots + 2^d = 2^{d+1} - 1$$

(see Example 15 of Chapter 2). A full binary tree has the maximum number of nodes for a given depth of any binary tree. This gives us fact 1:

- Any binary tree of depth d has at most $2^{d+1} - 1$ nodes.

Fact 2, which we'll prove momentarily, is

- Any binary tree with m nodes has depth $\geq \lfloor \log m \rfloor$.

To prove fact 2, we'll use a proof by contradiction. Suppose a binary tree has m nodes and depth $d < \lfloor \log m \rfloor$. Then $d \leq \lfloor \log m \rfloor - 1$. From fact 1,

$$\begin{aligned} m &\leq 2^{d+1} - 1 \leq 2^{(\lfloor \log m \rfloor - 1) + 1} - 1 \\ &= 2^{\lfloor \log m \rfloor} - 1 \leq 2^{\log m} - 1 = m - 1 \end{aligned}$$

or

$$m \leq m - 1$$

—a contradiction. Therefore $d \geq \lfloor \log m \rfloor$.

Now back to decision trees representing search algorithms on n -element lists. Temporarily strip the leaves from tree T_1 (with m internal nodes) to create a new tree T_2 with m nodes, $m \geq n$. By fact 2, T_2 has depth $d \geq \lfloor \log m \rfloor \geq \lfloor \log n \rfloor$.

Therefore tree T_1 has depth $\geq \lceil \log n \rceil + 1$. Because the depth of the decision tree gives the worst-case number of comparisons, we can state the following theorem.

● **THEOREM ON THE LOWER BOUND FOR SEARCHING**

Any algorithm that solves the search problem for an n -element list by comparing the target element x to the list items must do at least $\lceil \log n \rceil + 1$ comparisons in the worst case.

This theorem gives us a lower bound on the number of comparisons required in the worst case for any algorithm that uses comparisons to solve the search problem. Since binary search does no more work than this required minimum amount, binary search is an **optimal algorithm** in its worst-case behavior.

Binary Tree Search

The binary search algorithm requires that data already be sorted. Arbitrary data can be organized into a structure called a *binary search tree*, which can then be searched using a different algorithm called *binary tree search*. To build a binary search tree, the first item of data is made the root of the tree. Successive items are inserted by comparing them to existing nodes, beginning with the root. If the item is less than a node, the next node tested is the left child; otherwise it is the right child. When no child node exists, the new item becomes the child.

EXAMPLE 34

The data items

5, 8, 2, 12, 10, 14, 9

are to be organized into a binary search tree. Figure 6.54 shows the successive stages of constructing the tree.

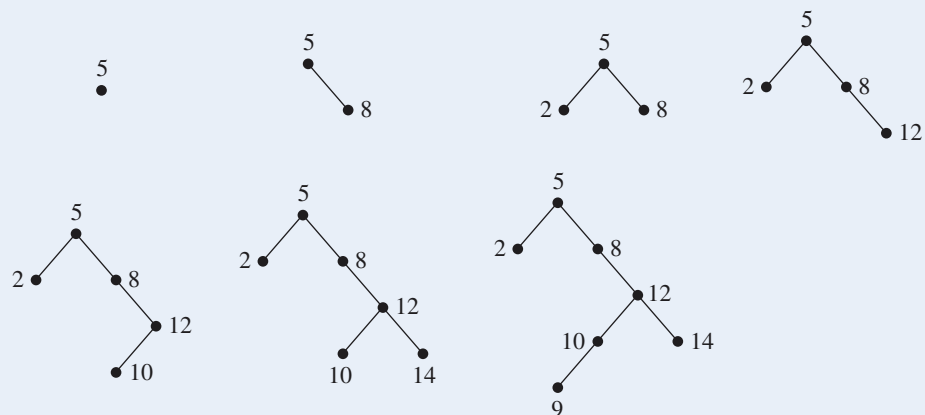


Figure 6.54

A **binary search tree**, by the way it is constructed, has the property that the value at each node is greater than all values in its left subtree (the subtree rooted at its left child) and less than all values in its right subtree. A **binary tree search** compares item x with a succession of nodes beginning with the root. If x equals the node item, the algorithm terminates; if x is less than the item, the left child is checked next; if x is greater than the item, the right child is checked next. If no child exists, the algorithm terminates because x is not in the list. Thus the binary search tree, except for the leaves, becomes the decision tree for the binary tree search algorithm. (Here is a case where the algorithm itself is described in terms of a tree.) The worst-case number of comparisons equals the depth of the tree plus 1 (for the missing leaves). However, a binary search tree for a given set of data is not unique; the tree (and hence the depth of the tree) depends on the order in which the data items are inserted into the tree.

EXAMPLE 35

The data in Example 34 entered in the order

9, 12, 10, 5, 8, 2, 14

produce the binary search tree of Figure 6.55.

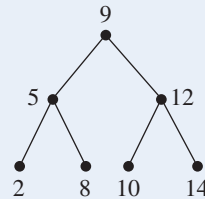


Figure 6.55

The actions performed in a binary tree search certainly resemble those in the “regular” binary search algorithm; in both cases the procedure is to make a comparison and, if unsuccessful, to look left or right (in the tree if it is a binary tree search or in the list if it is a binary search). It is possible to order the data for a binary tree search such that the search tree built from these data matches the decision tree (minus the leaves) for a binary search of the same data in sorted order. This match is illustrated in Example 35 (note that the tree was not built from data items in sorted order). Here the binary search tree has the minimum depth and requires the least amount of work in the worst case.

The depth of a binary search tree for a given set of data items can vary. The depth of the tree in Figure 6.54 is 4, while that of Figure 6.55 is 2. Thus the worst-case number of comparisons to search for an item can also vary. The tree-building process can be modified to keep the tree more “balanced,” that is, short and wide rather than tall and skinny; such a modification reduces the depth of the tree and therefore the search time. Of course, we know from the theorem on the lower bound for searching that a certain minimum amount of work is required no matter how clever we are in building the tree.

PRACTICE 26

a. Construct the binary search tree for the data of Example 34 entered in the order

12, 9, 14, 5, 10, 8, 2

b. What is the depth of the tree? ■

Sorting

Decision trees can also model algorithms that sort a list of items by a sequence of comparisons between two items from the list. The internal nodes of such a decision tree are labeled $L[i]:L[j]$ to indicate a comparison of list item i to list item j . To simplify our discussion, let's assume that the list does not contain duplicate items. Then the outcome of such a comparison is either $L[i] < L[j]$ or $L[i] > L[j]$. If $L[i] < L[j]$, the algorithm proceeds to the comparison indicated at the left child of this node; if $L[i] > L[j]$, the algorithm proceeds to the right child. If no child exists, the algorithm terminates because the sorted order has been determined. The tree is a binary tree, and the leaves represent the final outcomes, that is, the various sorted orders.

EXAMPLE 36

Figure 6.56 shows the decision tree for a sorting algorithm acting on a list of three elements. This algorithm is not particularly astute because it ignores the transitive property of $<$ and therefore performs some unnecessary comparisons. The leaves of the tree indicate the various final outcomes, including two cases (marked with an X) that result from contradictory information. For example, one X results from the following inconsistent sequence of outcomes: $L[1] < L[2]$, $L[2] < L[3]$, $L[1] > L[3]$.

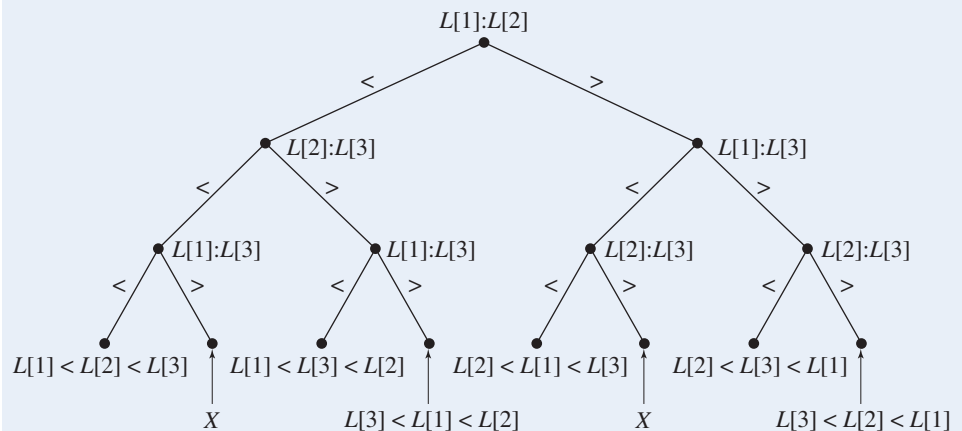


Figure 6.56

PRACTICE 27

Draw the decision tree that would result if the algorithm of Example 36 were modified to eliminate unnecessary comparisons. ■

A decision tree argument can also be used to establish a lower bound on the worst-case number of comparisons required to sort a list of n elements. As we did for the search problem, let us see what we can say about a decision tree for sorting based on comparisons, regardless of the algorithm it represents. The leaves of such a tree represent the final outcomes, that is, the various ordered arrangements of the n items. There are $n!$ such arrangements, so if p is the number of leaves in the decision tree, then $p \geq n!$. The worst case will equal the depth of the tree. But it is also true that if the tree has depth d , then $p \leq 2^d$ (Exercise 43 of Section 6.2). Taking the base 2 logarithm of both sides of this inequality, we get $\log p \leq d$ or, because d is an integer, $d = \lceil \log p \rceil$. Finally, we obtain

$$d = \lceil \log p \rceil \geq \lceil \log n! \rceil$$



This result proves the following theorem.

THEOREM ON THE LOWER BOUND FOR SORTING
 Any algorithm that sorts an n -element list by comparing pairs of items from the list must do at least $\lceil \log n! \rceil$ comparisons in the worst case.

It can be shown (Exercise 23) that $\log n! = \Theta(n \log n)$. Therefore we have proved that sorting n elements by comparing pairs of list items is bounded below by $\Theta(n \log n)$, whereas searching by comparing the target element to the list items is bounded below by $\Theta(\log n)$. As expected, it takes more work to sort than to search.

SECTION 6.3 REVIEW

TECHNIQUES

-  Draw decision trees for sequential search and binary search on n -element lists.
-  Create a binary search tree.

MAIN IDEAS

- Decision trees represent the sequences of possible actions for certain algorithms.
- Analysis of a generic decision tree for algorithms that solve a certain problem may lead to lower

bounds on the minimum amount of work needed to solve the problem in the worst case.

- The task of searching an n -element list for a target value x , if done by comparing x to elements in the list, requires at least $\lceil \log n \rceil + 1$ comparisons in the worst case.
- The task of sorting an n -element list, if done by comparing pairs of list elements, requires at least $\lceil \log n! \rceil$ comparisons in the worst case.

EXERCISES 6.3

1. Draw the decision tree for sequential search on a list of three elements.
2. Draw the decision tree for sequential search on a list of six elements.
3. Draw the decision tree for binary search on a sorted list of seven elements. What is the depth of the tree?
4. Draw the decision tree for binary search on a sorted list of four elements. What is the depth of the tree?

5. Consider a search algorithm that compares an item with the last element in a list, then the first element, then the next-to-last element, then the second element, and so on. Draw the decision tree for searching a six-element sorted list. What is the depth of the tree? Does it appear that this is an optimal algorithm in the worst case?
6. Consider a search algorithm that compares an item with an element one-third of the way through the list; based on that comparison, it then searches either the first one-third or the second two-thirds of the list. Draw the decision tree for searching a nine-element sorted list. What is the depth of the tree? Does it appear that this is an optimal algorithm in the worst case?
7. a. Given the data

9, 5, 6, 2, 4, 7

construct the binary search tree. What is the depth of the tree?

- b. Find the average number of comparisons done to search for an item that is known to be in the list using binary tree search on the tree of part (a). (*Hint*: Find the number of comparisons for each of the items.)
8. a. Given the data

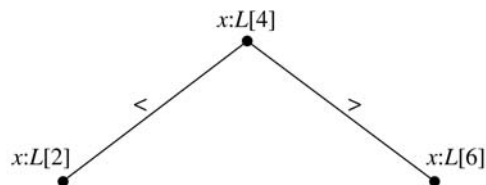
g, d, r, s, b, q, c, m

construct the binary search tree. What is the depth of the tree?

- b. Find the average number of comparisons done to search for an item that is known to be in the list using binary tree search on the tree of part (a). (*Hint*: Find the number of comparisons for each of the items.)
9. a. For a set of six data items, what is the minimum worst-case number of comparisons a search algorithm must perform?
- b. Given the set of data items $\{a, d, g, i, k, s\}$, find an order in which to enter the data so that the corresponding binary search tree has the minimum depth.
10. a. For a set of nine data items, what is the minimum worst-case number of comparisons a search algorithm must perform?
- b. Given the set of data items $\{4, 7, 8, 10, 12, 15, 18, 19, 21\}$, find an order in which to enter the data so that the corresponding binary search tree has the minimum depth.
11. An inorder tree traversal of a binary search tree produces a listing of the tree nodes in alphabetical or numerical order. Construct a binary search tree for “To be or not to be, that is the question,” and then do an inorder traversal.
12. Construct a binary search tree for “In the high and far-off times the Elephant, *O* Best Beloved, had no trunk,” and then do an inorder traversal. (See Exercise 11.)
13. Use the theorem on the lower bound for sorting to find lower bounds on the number of comparisons required in the worst case to sort lists of the following sizes:
 - a. 4
 - b. 8
 - c. 16
14. Contrast the number of comparisons required for selection sort and merge sort in the worst case with the lower bounds found in Exercise 13 (see Exercise 23 in Section 3.3). What are your conclusions?

Exercises 15–20 concern the problem of identifying a counterfeit coin (one that is too heavy or too light) from a set of n coins. A balance scale is used to weigh a group of any number of coins from the set against a like number of coins from the set. The outcome of such a comparison is that group A weighs less than, the same as, or more than group B . A decision tree representing the sequence of comparisons done will thus be a ternary tree, where an internal node can have three children.

15. One of five coins is counterfeit and is lighter than the other four. The problem is to identify the counterfeit coin.
 - a. What is the number of final outcomes (the number of leaves in the decision tree)?
 - b. Find a lower bound on the number of comparisons required to solve this problem in the worst case.
 - c. Devise an algorithm that meets this lower bound (draw its decision tree).
16. One of five coins is counterfeit and is either too heavy or too light. The problem is to identify the counterfeit coin and determine whether it is heavy or light.
 - a. What is the number of final outcomes (the number of leaves in the decision tree)?
 - b. Find a lower bound on the number of comparisons required to solve this problem in the worst case.
 - c. Devise an algorithm that meets this lower bound (draw its decision tree).
17. One of four coins is counterfeit and is either too heavy or too light. The problem is to identify the counterfeit coin but not to determine whether it is heavy or light.
 - a. What is the number of final outcomes (the number of leaves in the decision tree)?
 - b. Find a lower bound on the number of comparisons required to solve this problem in the worst case.
 - c. Devise an algorithm that meets this lower bound (draw its decision tree).
18. One of four coins is counterfeit and is either too heavy or too light. The problem is to identify the counterfeit coin and determine whether it is heavy or light.
 - a. What is the number of final outcomes (the number of leaves in the decision tree)?
 - b. Find a lower bound on the number of comparisons required to solve this problem in the worst case.
 - c. Prove that no algorithm exists that can meet this lower bound. (*Hint*: The first comparison can be made with either two coins or four coins. Consider each case.)
19. Devise an algorithm to solve the problem of Exercise 18 using three comparisons in the worst case.
20. One of eight coins is counterfeit and is either too heavy or too light. The problem is to identify the counterfeit coin and determine whether it is heavy or light.
 - a. What is the number of final outcomes (the number of leaves in the decision tree)?
 - b. Find a lower bound on the number of comparisons required to solve this problem in the worst case.
 - c. Devise an algorithm that meets this lower bound (draw its decision tree).
21. In the decision tree for the binary search algorithm (and the binary tree search algorithm), we have counted each internal node as one comparison. For example, the top of Figure 6.53 looks like this:



To get to either of the child nodes of the root, we have assumed that one comparison has been done. However, the outcome of the comparison at each internal node is really a three-way branch:

$x =$ node element
 $x <$ node element
 $x >$ node element

Think about how this three-way branch would be implemented in most programming languages, and write a more accurate expression than $1 + \lceil \log n \rceil$ for the number of comparisons in the worst case.

22. Our existing binary search algorithm (Chapter 3, Example 13) contains the pseudocode instruction
 find the index k midway between i and j

after which the target x is compared to the list item at index k , the “midpoint item.” Suppose that this instruction is replaced by

```

if  $i = j$  then
     $k = j$ 
else
     $k = i + 1$ 
end if

```

- Draw the decision tree that results from using the modified algorithm on a sorted list with $n = 8$.
 - Give the exact number of comparisons required (see Exercise 21) in the worst case for $n = 8$.
 - Give a worst-case order-of-magnitude expression for the number of comparisons required as a function of n , and justify your expression. Comment on the use of this algorithm as opposed to the original binary search algorithm, which is $\Theta(\log n)$.
23. To prove that $\log n! = \Theta(n \log n)$, we can use the definition of order of magnitude (see Section 5 of Chapter 5) and show that there exist positive constants n_0 , c_1 , and c_2 such that for $n \geq n_0$, $c_1(n \log n) \leq \log n! \leq c_2(n \log n)$.
- Show that for $n \geq 1$, $\log n! \leq n \log n$. (*Hint:* Use the definition of $n!$ and properties of logarithms.)
 - Show that for $n \geq 4$, $\log n! \geq (1/4)(n \log n)$. (*Hint:* Use the definition of $n!$ and properties of logarithms, but stop at $\log \lceil n/2 \rceil$.)

SECTION 6.4 HUFFMAN CODES

Problem and Trial Solution

Character data consist of letters of the alphabet (both uppercase and lowercase), punctuation symbols, and other keyboard symbols such as @ and %. Computers store character data in binary form, as a sequence of 0s and 1s, usually by fixing some length n so that 2^n is at least as large as the number of distinct characters and then encoding each distinct character as a particular sequence of n bits. Each character must be encoded into its fixed binary sequence for electronic storage, and then the binary sequence must be decoded when the character is to be displayed. The most common encoding scheme for many years was ASCII (American Standard Code for Information Interchange), which uses $n = 8$, so that each character requires 8 bits to store. However, $2^8 = 256$, so a maximum of 256 characters could be encoded. This was enough for the English alphabet, punctuation, and special characters, but as electronic data storage spread around the world, it was not enough to include characters found in other languages such as Russian, Japanese, Arabic, Greek, and many others. Unicode (in general) uses 16 bits to encode a single character, so that $2^{16} = 65536$ character encodings are now available. But whatever value is chosen for n , each character requires the same amount of storage space.

Suppose a collection of character data to be stored in a file in binary form is large enough that the amount of storage required is a consideration. Suppose also that the file is archival in nature, and its contents will not often be changed. Then

it may be worthwhile to invest some extra effort in the encoding process if the amount of storage space required for the file could be reduced.

Rather than using a fixed number of bits per character, an encoding scheme could use a variable number of bits and store frequently occurring characters as sequences with fewer bits. To store all the distinct characters, some sequences will still have to be long, but if the longer sequences are used for characters that occur less frequently, the overall storage required should be reduced. This approach requires knowledge of the particular file contents, which is why it is best suited for a file whose contents will not be frequently changed. We will study such a **data compression** or **data compaction** scheme here, because it is best described as a series of actions taken on binary trees.

EXAMPLE 37

As a trivial example, suppose that a collection of data contains 50,000 instances of the six characters a , c , g , k , p , and $?$, which occur with the following percent frequencies:

Character	a	c	g	k	p	$?$
Frequency	48	9	12	4	17	10

Because six distinct characters must be stored, the fixed-length scheme would require at a minimum three bits for each character ($2^3 = 8 \geq 6$). The total storage required would then be $50,000 * 3 = 150,000$ bits. Suppose instead that the following encoding scheme is used:

Character	a	c	g	k	p	$?$
Encoding scheme	0	1101	101	1100	111	100

Then the storage requirement (number of bits) is

$$50,000(0.48 * 1 + 0.09 * 4 + 0.12 * 3 + 0.04 * 4 + 0.17 * 3 + 0.10 * 3) = 108,500$$

which is roughly two-thirds of the previous requirement. ●

In the fixed-length storage scheme with n bits for each character, the long string of bits within the encoded file can be broken up into the code for successive characters by simply looking at n bits at a time. This makes it easy to decode the file. In the variable-length code, there must be a way to tell when the sequence for one character ends and the sequence for another character begins.

PRACTICE 28 Using the variable-length code of Example 37, decode each of the following strings:

- 11111111010100
- 1101010101100
- 100110001101100

In Practice 28 the strings can be broken into the representation of characters in only one way. As each new digit is considered, the possibilities are narrowed as to which character is being represented until the character is uniquely identified by the end of that character's representation. There is never any need to guess

at what the character might be and then backtrack if our guess proves wrong. This ability to decode uniquely without false starts and backtracking comes about because the code is an example of a **prefix code**. In a prefix code, the code for any character is never the prefix of the code for any other character. (A prefix code is therefore an “antiprefix” code!)

EXAMPLE 38

Consider the code

Character	<i>a</i>	<i>b</i>	<i>c</i>
Encoding scheme	01	101	011

which is not a prefix code. Given the string 01101, it could represent either *ab* (01–101) or *ca* (011–01). Furthermore, in processing the string 011011 digit by digit as a computer would do, the decoding could begin with *ab* (01–101) and only encounter a mismatch at the last digit. Then the process would have to go all the way back to the first digit in order to recognize *cc* (011–011).

As an aside, Morse code is a variable-length code. Morse’s encoding scheme for telegraphic communication, invented in 1838, uses strings of dots and dashes to represent letters of the alphabet. The most frequently occurring letter of the alphabet in English text is the letter “*e*,” which is assigned the shortest code, a single dot. The Morse code for

“hello world”

is

“.....-.-. -.-. --- -.- --- -.-.-.-.”

Here you can see that “*e*” is a single dot, that “*l*” is dot-dash-dot-dot, and that “*r*” is dot-dash-dot. Morse code is not a prefix code, however; note that the “*r*” code is the first part of the “*l*” code. To avoid this kind of ambiguity, Morse code inserts a pause between the code for each letter. To decode, you wait for the pause and at that point you have the code for exactly one letter.

In our approach to prefix codes, we will build binary trees with the characters as leaves. Once the tree is built, a binary code can be assigned to each character by simply tracing the path from the root to that leaf, using 0 for a left branch and 1 for a right branch. Because no leaf precedes any other leaf on some path from the root, the code will be a prefix code. The binary tree for the code of Example 37 is shown in Figure 6.57.

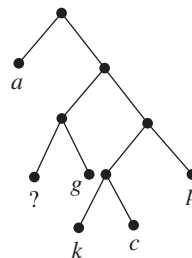


Figure 6.57

Suppose a code tree T exists, with leaves representing characters. For any leaf i , its depth $d(i)$ in T equals the number of bits in the code for the corresponding character. Let $f(i)$ denote the percentage frequency of that character in the data to be stored, and let S be the total number of characters to be stored. Then, just as in Example 37, the total number of bits required is given by the expression

$$S * \left[\sum_{\text{all leaves } i} (d(i)f(i)) \right]$$

We seek to build an optimal tree T , one for which the expression

$$E(T) = \left[\sum_{\text{all leaves } i} (d(i)f(i)) \right] \quad (1)$$

is a minimum and hence the file size is a minimum.

This process could be done by trial and error, because there is only a finite number of characters and thus only a finite number of ways to construct a tree and assign characters to its leaves. However, the finite number quickly becomes very large! Instead we will use the algorithm known as **Huffman encoding**.

Huffman Encoding Algorithm

Suppose, then, that we have m characters in a file and we know the percentage frequency of each character. The algorithm to build the tree works by maintaining a list L of nodes that are roots of binary trees. Initially L will contain m roots, each labeled with the frequency of one of the characters; the roots will be ordered according to increasing frequency, and each will have no children. A pseudocode description of the algorithm follows.

ALGORITHM *HUFFMANTREE*

```

HuffmanTree (node list  $L$ ; integer  $m$ )
//Each of the  $m$  nodes in  $L$  has an associated frequency  $f$ , and  $L$  is
//ordered by increasing frequency; algorithm builds the Huffman tree

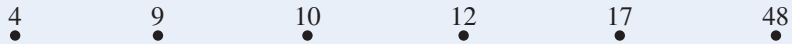
  for ( $i = 1$  to  $m - 1$ ) do
    create new node  $z$ 
    let  $x, y$  be the first two nodes in  $L$       //minimum frequency nodes
     $f(z) = f(x) + f(y)$ 
    insert  $z$  in order into  $L$ 
    left child of  $z =$  node  $x$ 
    right child of  $z =$  node  $y$                 // $x$  and  $y$  are no longer in  $L$ 
  end for
end HuffmanTree

```

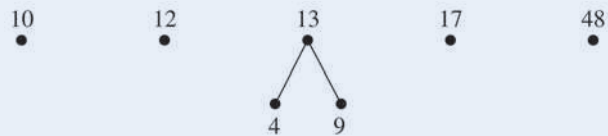
When this algorithm terminates, L consists of just one node, which is the root of the final binary tree. Codes can then be assigned to each leaf of the tree by tracing the path from the root to the leaf and accumulating 0s for left branches and 1s for right branches. By the way the tree is constructed, every internal node will have exactly two children.

EXAMPLE 39

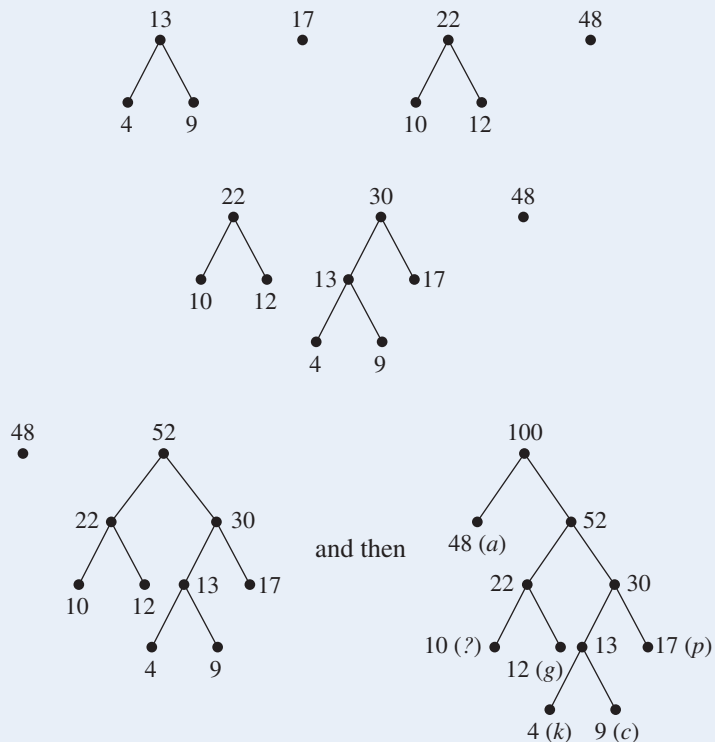
We'll use algorithm *HuffmanTree* to build the tree of Figure 6.57, which is based on the data of Example 37. L initially contains the six nodes, ordered by frequency:



Following the algorithm, we enter the **for** loop for the first time. The x and y nodes are those with frequencies 4 and 9, respectively. A new node z with frequency $4 + 9 = 13$ is created and inserted in order into L , with the x node as its left child and the y node as its right child. The new L looks like the following:



This process is repeated four more times. The resulting L at each stage follows:



At this point the tree is complete and the codes can be assigned. The code for c , for example, is 1101 (right branch, right branch, left branch, right branch). ●

PRACTICE 29 Construct the Huffman tree for the following characters and frequencies:

Character	<i>w</i>	<i>q</i>	<i>h</i>	<i>e</i>
Frequency	10	12	20	58

PRACTICE 30 Find the Huffman codes for the characters of Practice 29.

Table 6.1 shows the steps in Huffman encoding/decoding for data compression.

Encoding Step 1	On the original CLEARTEXT file, perform a frequency analysis; that is, create a file FREQUENCY that contains data of the form <i>a</i> —18 <i>b</i> —7 and so forth.
Encoding Step 2	Using FREQUENCY, create a file CODETABLE that contains the Huffman code for each character, i.e., <i>a</i> —001 <i>b</i> —1110 and so forth.
Encoding Step 3	Using CLEARTEXT and CODETABLE, create a file called CODED that contains the compressed data.
Decoding	Using CODED and CODETABLE, decode the data to recover CLEARTEXT.

The CODED file is the data-compressed version of CLEARTEXT, and presumably it requires less storage space. However, the CODETABLE file must also be stored in order to be able to decode the file.

Justification

Although the algorithm to construct the Huffman tree T is easy enough to describe, we must justify that it gives us the minimum possible value for $E(T)$.

First, if we have an optimal tree T for m characters, the nodes with the lowest frequencies can always be assumed to be the left and right children of some node. To prove this assumption, label the two nodes with the lowest frequencies x and y . If x and y are not siblings in the tree, then find two siblings p and q at the lowest level of the tree, and consider the case where x and y are not at that level (Figure 6.58a). Because $f(x)$ is one of the two smallest values, we know that $f(x) \leq f(p)$. If $f(x) < f(p)$, then interchanging x and p in the tree would result in a new tree T' with $E(T') < E(T)$ (Figure 6.58b: the larger frequency is now at a lesser depth—see Exercise 20a), but this would contradict the fact that T was optimal. Therefore $f(x) = f(p)$, and x and p can be interchanged in the tree with no effect on $E(T)$. Similarly, y and q can be interchanged, resulting in Figure 6.58c, in which x and y are siblings. If x or y are at the same level as p and q to begin with, they can certainly be interchanged with p or q without affecting $E(T)$ (Figure 6.58d).

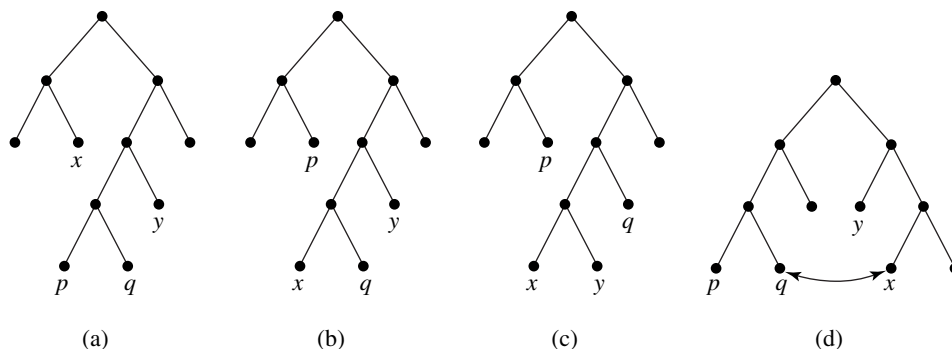


Figure 6.58

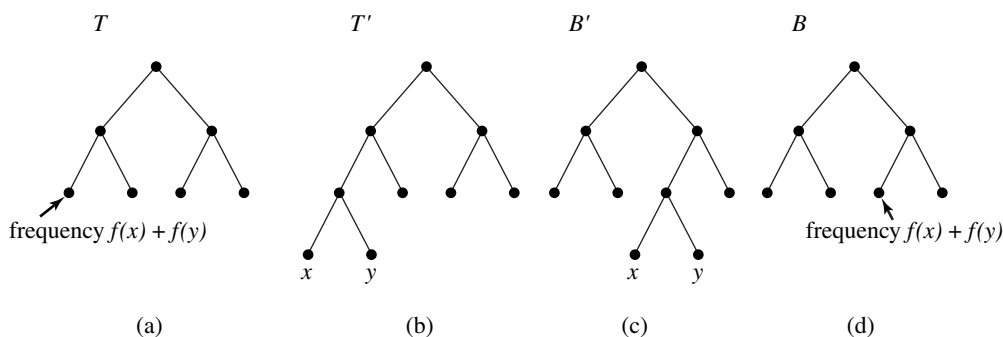


Figure 6.59

Now again let $f(x)$ and $f(y)$ be the minimum frequencies, and suppose we have a tree T that is optimal for the other frequencies together with the sum $f(x) + f(y)$ (Figure 6.59a). This sum will be the frequency of a leaf node; create a tree T' that has this node as an interior node with children x and y having frequencies $f(x)$ and $f(y)$ (Figure 6.59b). T' will be optimal for frequencies $f(x), f(y)$, and the rest. The proof of this fact begins with some optimal tree B' for frequencies $f(x), f(y)$, and the rest. We know such an optimal tree exists (since it could be found by trial and error), and from the preceding paragraph, we can assume that x and y are siblings in B' (Figure 6.59c). Now create a tree B by stripping nodes x and y from B' and giving frequency $f(x) + f(y)$ to their parent node, now a leaf (Figure 6.59d). Because T is optimal for the other frequencies together with $f(x) + f(y)$, we have

$$E(T) \leq E(B) \tag{1}$$

But the difference between $E(B)$ and $E(B')$ is one arc each for x and y ; that is, $E(B') = E(B) + f(x) + f(y)$ (see Exercise 20b). Similarly, we have $E(T') = E(T) + f(x) + f(y)$. Thus, if we add $f(x) + f(y)$ to both sides of (1), we get

$$E(T') \leq E(B') \tag{2}$$

Because B' was optimal, it cannot be the case that $E(T') < E(B')$, so $E(T') = E(B')$, and T' is optimal.

Finally, a tree with a single node whose frequency is the sum of all the frequencies is trivially optimal for that sum. We can repeatedly split up this sum and drop down children in such a way that we end up with the Huffman tree. By the preceding paragraph, each such tree, including the final Huffman tree, is optimal.

EXAMPLE 40

If we applied the process that preserves optimality to the tree of Figure 6.57, we would begin with a single node with frequency 100 and “grow” that tree downward, as shown in Figure 6.60.

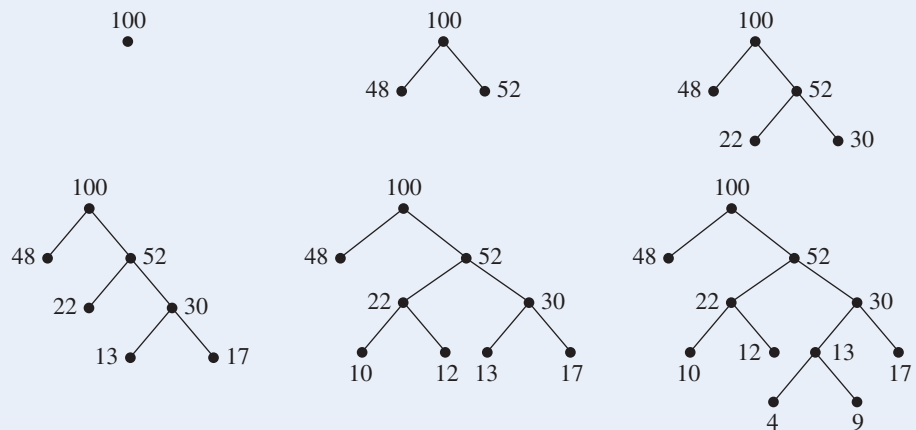


Figure 6.60

Application of Huffman Codes

The cost of data storage has certainly declined in recent years, and the relatively low cost makes it seem as if practically unlimited storage should be available to all. But “relatively inexpensive” doesn’t mean “free,” and during the same period in which data storage costs have decreased, the legal requirements on many businesses and other organizations to keep electronic records have increased. Much of this information is archival in nature and suitable for data compression. As just one example, the Sarbanes–Oxley Act became United States federal law in 2002. This law was enacted as a reaction to a series of high-profile scandals about accounting irregularities and irresponsible corporate governance that shook financial markets and public trust. Congress authorized the Securities and Exchange Commission (SEC) to set up rules regarding retention of documents concerned with financial audits or reviews. The final SEC rule requires that all publicly traded companies must retain electronic records of all financial transactions for seven years. These records are to include not just formal documents but any e-mail, memos, working papers, and so forth, sent or received, that contain opinions, conclusions, analyses, or financial data.

As a more specific application of Huffman codes, let's look at JPEG, a standardized image compression mechanism for photographic-quality images. JPEG stands for Joint Photographic Experts Group, the name of the group that developed this international standard. The need for improved image compression was largely fueled by the desire to transmit images over the Internet. There are actually two versions of JPEG encoding, lossy and lossless, but the lossy version is by far the more common. A *lossy compression scheme* means that once the compressed data has been unencoded, it does not precisely match the original—some information has been “lost.” In the case of lossy JPEG compression, data loss comes from the preprocessing of the image before Huffman encoding is applied; the Huffman encoding/decoding faithfully restores the data it starts with.

JPEG compression is intended for images to be viewed by humans, and it takes advantage of the fact that the human eye is much more sensitive to gradients of light and dark than it is to small changes in color. The first step in the JPEG process is therefore to take the color image information, which is usually given as 24 bits per pixel, 8 bits for each of the red, green, and blue components, and transform each pixel into components that capture the luminance (lightness/darkness) with reduced information about the color components. Next, pixels with similar color information are grouped together and an “average” color value is used, while more accurate luminance data are maintained. The data are then transformed into frequency data (that is, the data are represented as a combination of cosine waves of varying frequencies), which in turn go through a “quantization” process (basically rounding the results of a computation) to end up in integer form. Higher-frequency variations, to which the human eye is less sensitive, are lost in this process, but again the luminance data are treated at a finer grain than the color data. Huffman encoding is applied to the result. Areas of the image whose representations occur often will encode to smaller bit strings.

A JPEG image file contains not only the compressed data but also the information needed to reverse the compression process (including the information to reverse the Huffman encoding). The resulting image has lost the high-frequency changes and color variations that were eliminated in the stages before the Huffman coding was applied. Parameters in the JPEG encoding process allow tradeoffs to be made between the amount of compression to be obtained and the faithfulness of the restored image to the original. Because of the nature of the algorithms used, JPEG encoding has little or no effect on black-and-white line drawings where there are no data to throw away.

SECTION 6.4 REVIEW

TECHNIQUE

- W Find Huffman codes, given a set of characters and their frequencies.

MAIN IDEA

- Given the frequency of characters in a collection of data, a binary encoding scheme can be found that minimizes the number of bits required to store the data but still allows for easy decoding.

EXERCISES 6.4

1. Is the following code a prefix code? Why or why not?

Character	<i>m</i>	<i>b</i>	<i>d</i>	<i>w</i>
Encoding scheme	01	100	011	101

2. Using the code of Exercise 1, decode the string 01101.
 3. Given the codes

Character	<i>a</i>	<i>e</i>	<i>i</i>	<i>o</i>	<i>u</i>
Encoding scheme	00	01	10	110	111

decode the sequences

- a. 11011011101
 b. 1000110111
 c. 010101
4. Given the codes

Character	<i>b</i>	<i>h</i>	<i>q</i>	<i>w</i>	%
Encoding scheme	1000	1001	0	11	101

decode the sequences

- a. 10001001101101
 b. 11110
 c. 01001111000
5. Given the codes

Character	<i>a</i>	<i>p</i>	<i>w</i>	()
Encoding scheme	001	1010	110	1111	1110

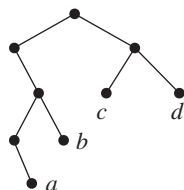
decode the sequences

- a. 111110101101110001
 b. 1010001110
 c. 111111100111101110
6. Given the nonprefix codes

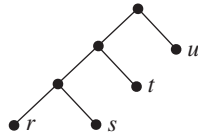
Character	1	3	5	7	9
Encoding scheme	1	111	101	10	10101

give all possible decodings of the sequence 111110101.

7. Write the Huffman codes for *a*, *b*, *c*, and *d* in the binary tree shown.



8. Write the Huffman codes for r, s, t, u in the binary tree shown.



9. a. Construct the Huffman tree for the following characters and frequencies.

Character	c	d	g	m	r	z
Frequency	28	25	6	20	3	18

- b. Find the Huffman codes for these characters.

10. a. Construct the Huffman tree for the following characters and frequencies.

Character	b	n	p	s	w
Frequency	6	32	21	14	27

- b. Find the Huffman codes for these characters.

11. a. Construct the Huffman tree for the following characters and frequencies.

Character	a	z	t	e	c
Frequency	27	12	15	31	15

- b. Find the Huffman codes for these characters.

12. a. Construct the Huffman tree for the following characters and frequencies.

Character	?	x	w	e	t	s	a
Frequency	14	3	11	27	18	22	5

- b. Find the Huffman codes for these characters.

- c. A file consisting of 100,000 instances of these seven characters is stored using a fixed-length binary encoding scheme. How many bits are required for each code and what is the total number of bits needed?

- d. Storing the same file using the Huffman code of part (b), how many bits are needed?

In Exercises 13–14, the integers could represent the results of the “quantization” step in a JPEG image compression, and the number of occurrences of each in the image. (Note that these are instances of occurrence rather than percent frequencies; this simply means that the Huffman tree will not end up with the root value equal to 100.)

13. Construct the Huffman tree and find the Huffman codes for the following integers and occurrences.

Integer	82	664	327	349	423	389
Occurrences	416	97	212	509	446	74

14. Construct the Huffman tree and find the Huffman codes for the following integers and occurrences.

Integer	190	205	514	333	127	901	277
Occurrences	52	723	129	233	451	820	85

15. JPEG can achieve various compression levels; the higher the compression, the lower the quality of the reconstructed image.
- A compression ratio of 10:1 results in virtually imperceptible loss of image quality. A file of 850,000 bytes that is compressed with a 10:1 compression ratio results in a compressed file of what size?
 - At a compression ratio of 25:1, some degradation is visible in the reconstructed image. A file of 850,000 bytes that is compressed with a 25:1 compression ratio results in a compressed file of what size?
16. Explain why JPEG encoding results in less compression for gray-scale images than for full-color images.
17. Someone does a global substitution on the text file of Exercise 11, replacing all instances of “z” with “sh.” Find the new Huffman codes.
18. Consider the following paragraph.

However, in my thoughts I could not sufficiently wonder at the intrepidity of these diminutive mortals who durst venture to mount and walk upon my body, while one of my hands was at liberty, without trembling at the very sight of so prodigious a creature as I must appear to them.³

If this paragraph were to be compressed using a Huffman code, what single character, aside from punctuation or uppercase characters, would be apt to have one of the longest codes? Which would have one of the shortest?

19. Recall the problem posed at the beginning of this chapter.

You work in the Information Systems Department at World Wide Widgets (WWW), the leading widget manufacturer. Part numbers consist of a leading character *B*, *C*, *G*, *R*, or *S* to identify the part type, followed by an 8-digit number. Thus

C00347289
B11872432
S45003781

are all legitimate part numbers. WWW maintains a data file of the part numbers it uses, which, as it turns out, is most of the potential numbers.

How can you compress this data file so it takes less storage space than the approximately 4.5 Gb required using the ASCII encoding scheme of eight bits per character?

- a. Running a frequency count on the WWW data file reveals the following information:

Character	<i>B</i>	<i>C</i>	<i>G</i>	<i>R</i>	<i>S</i>	0	1	2	3	4	5	6	7	8	9
Frequency	2	5	1	2	1	18	13	7	12	9	6	11	7	2	4

Construct a Huffman code for these characters.

- Compute the space requirements of the compressed file as a percent of the uncompressed file.
20. In the justification that the Huffman algorithm produces an optimal tree, the following two assertions were made. Prove that each is true.
- $E(T') < E(T)$
 - $E(B') = E(B) + f(x) + f(y)$

³From *Gulliver's Travels* by Jonathan Swift, London, 1726.

CHAPTER 6 REVIEW

TERMINOLOGY

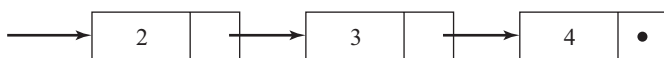
- acyclic graph (p. 482)
 adjacency list (p. 494)
 adjacency matrix (p. 492)
 adjacent nodes (p. 481)
 arc (edge) (p. 477, 478)
 binary search tree (p. 534)
 binary tree (p. 510)
 binary tree search (p. 534)
 bipartite complete graph (p. 483)
 children nodes (p. 509)
 complete binary tree (p. 510)
 complete graph (p. 482)
 connected graph (p. 482)
 cycle (p. 482)
 data compression (data compaction) (p. 540)
 data flow diagram (p. 480)
 decision tree (p. 529)
 degree of a node (p. 481)
 depth of a node (p. 510)
 depth (height) of a tree (p. 510)
 directed graph (digraph) (p. 478)
 endpoints (p. 478)
 Euler's formula (p. 488)
 five-color theorem (p. 507)
 forest (p. 510)
 full binary tree (p. 510)
 graph (p. 477, 478)
 graph colorability (p. 506)
 height of a tree (p. 510)
 homeomorphic graphs (p. 491)
 Huffman encoding (p. 542)
 infix notation (p. 518)
 initial point (p. 478)
 inorder traversal (p. 515)
 internal node (p. 510)
 isolated node (p. 481)
 isomorphic graphs (p. 485)
 isomorphism (p. 486)
 labeled graph (p. 479)
 leaf (p. 510)
 left child (p. 510)
 length of a path (p. 482)
 linked list (p. 494)
 loop (p. 481)
 loop-free graph (p. 481)
 map-coloring problem (p. 506)
 node (vertex) (p. 477, 478)
 nonrooted (free) tree (p. 509)
 null pointer (p. 495)
 optimal algorithm (p. 533)
 parallel arcs (p. 481)
 parent node (p. 509)
 path (pp. 482, 483)
 planar graph (p. 487)
 Polish notation (p. 518)
 postfix notation (p. 518)
 postorder traversal (p. 515)
 prefix code (p. 541)
 prefix notation (p. 518)
 preorder traversal (p. 514)
 reachable node (p. 483)
 reverse Polish notation (RPN) (p. 518)
 right child (p. 510)
 root of a tree (p. 509)
 simple graph (p. 481)
 sparse matrix (p. 494)
 subgraph (p. 482)
 terminal point (p. 478)
 tree (p. 509)
 tree traversal (p. 514)
 weighted graph (p. 479)

SELF-TEST

Answer the following true–false questions.

Section 6.1

1. A connected graph has an arc between any two nodes.
2. If graph G_1 is isomorphic to graph G_2 , then a node of degree 5 in G_1 will be mapped to a node of degree 5 in G_2 .
3. No matter how a planar graph is drawn, its arcs will intersect only at nodes.
4. If part of the adjacency list representation of a graph contains



then node 2 is adjacent to node 3 and node 3 is adjacent to node 4.

5. The adjacency matrix of a directed graph is not symmetric.

Section 6.2

1. The depth of any node in a tree is less than or equal to the height of the tree.
2. Because a tree is a graph, a complete tree is also a complete graph.
3. In the left child–right child array representation of a binary tree, any row of the array that corresponds to a leaf will have all zero entries.
4. Postorder traversal of an expression tree results in an algebraic expression in reverse Polish notation.
5. In preorder tree traversal, the root is always the first node visited.

Section 6.3

1. The root of a decision tree for the binary search algorithm acting on a sorted list of 11 items would represent the comparison of the target element with the sixth list item.
2. Searching for any target element x in a list of n elements requires at least $1 + \lceil \log n \rceil$ comparisons.
3. A binary tree search is done with a target element of 14 on a binary search tree whose root has the value 10; the right subtree will be searched next.
4. A binary search tree is unique for any given set of data.
5. A decision tree for sorting n elements must have a depth of at least $n!$

Section 6.4

1. The ASCII encoding scheme requires 8 bits to store each character.
2. In a prefix code, each code word is the prefix of another code word.
3. In a Huffman code, characters that occur most frequently have the most 0s in their binary string representation.
4. The maximum number of bits for any encoded character using a Huffman code will be the depth of the Huffman tree.
5. To be able to decode an encoded file, a frequency count from the original file must be stored along with the encoded file.

ON THE COMPUTER

For Exercises 1–4, write a computer program that produces the desired output from the given input.

1. *Input:* Adjacency list for a graph
Output: Adjacency matrix for the graph
2. *Input:* Adjacency matrix for a graph
Output: Adjacency list for the graph
3. *Input:* Adjacency list for a graph and the name of a node n in the graph
Output: Adjacency list for the graph with node n and its associated arcs removed
4. *Input:* List of n characters and their (integer) frequencies
Output: Huffman code for the characters
(*Hint:* Maintain a sorted linked list of records that represent the roots of binary trees. Initially there will be n such records, each with no children; at the end, there will be one such record, the root of the Huffman tree.)
5. Write a program that allows the user to enter a list of integers and constructs a binary search tree with those integers as nodes. The user can then enter one integer at a time, and the program will do a binary tree search and indicate whether the given integer is in the list.
6. Write a program that allows the user to enter a list of integers and then constructs a binary search tree with those integers as nodes. The user can then enter the type of traversal desired (inorder, preorder, or postorder), and the program will write out the nodes in the appropriate order.
7. Write a program that carries out the first three steps in Table 6.1. That is, beginning with a text file, the program should produce a frequency count file, then a code table file, then an encoded version of the original file. Write a second program that uses the encoded file and the code table, and recreates the original file.

Graph Algorithms

CHAPTER OBJECTIVES

After studying this chapter, you will be able to:

- Convert between adjacency matrix, adjacency relation, and directed graph representations.
- Use the reachability matrix of a directed graph to determine whether one node is reachable from another.
- Compute the reachability matrix of a directed graph either directly or by using Warshall's algorithm.
- Test a graph for the existence of an Euler path (solve the highway inspector problem).
- Understand the Hamiltonian circuit problem (and the traveling salesman problem) and how they are fundamentally different from the Euler path problem.
- Use Dijkstra's algorithm to find the shortest path between two nodes in a simple, weighted, connected graph.
- Use Prim's algorithm to find the minimal spanning tree in a simple, weighted, connected graph.
- Carry out depth-first search and breadth-first search in a simple, connected graph.
- Understand how depth-first search can be used to test for reachability in a directed graph, perform a topological sort on a partially ordered set represented by a directed graph, and find the connected components of an unconnected graph.
- Identify articulation points in a simple connected graph.

You are the network administrator for a wide-area backbone network that serves your company's many offices across the country. Messages travel through the network by being routed from point to point until they reach their destination. Each node in the network therefore acts as a switching station to forward messages to other nodes according to a routing table maintained at each node. Some connections in the network carry heavy traffic, while others are less used. Traffic may vary with the time of day; in addition, new nodes occasionally come on line and existing nodes may go off line. Therefore you must periodically provide each node with updated information so that it can forward messages along the most efficient (that is, the least heavily traveled) route.

Question: How can you compute the routing table for each node?

If the network described is viewed as a graph, your task as network administrator is to find the “shortest” path from one node to another in the graph. Because graphs have so many applications, there is a great deal of interest in finding efficient algorithms to answer certain questions about graphs, directed graphs, or trees, and to perform certain tasks on them, such as finding shortest paths. All graph algorithms use one of the convenient representations (adjacency matrix or adjacency list) presented in Chapter 6.

This chapter covers many of the “classical” graph algorithms. Section 7.1 first relates directed graphs to binary relations, and reachability in a graph to the transitive closure of a binary relation. Then two different algorithms pertaining to reachability are given.

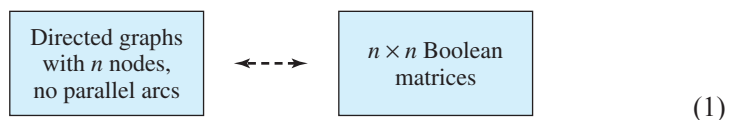
In Section 7.2 we will look at algorithms that answer two historically interesting questions about graphs. These questions are known as the *highway inspector problem* and the *traveling salesman problem*. The highway inspector problem asks whether there is a path through a given graph that uses each arc exactly once, thereby providing an efficient way for a highway inspector to check all roads without going over the same road twice. The traveling salesman problem asks whether there is a cycle in a given graph that visits each node of the graph and, if so, which such cycle requires the minimum distance to travel. Recall that a cycle is a path that ends where it started and does not use any other node more than once; thus such a cycle would provide an efficient way for a salesperson to visit all cities in the sales territory only once and end up at home.

Section 7.3 provides algorithmic solutions to the two problems of finding the minimum path between two nodes in a simple, connected graph and of minimizing the number of arcs used to connect all nodes in a simple, connected graph. Section 7.4 discusses algorithms for traversing simple graphs—“visiting” all the nodes in some systematic way. Section 7.5 uses one of these traversal algorithms to detect articulation points in a simple connected graph, points whose removal would disconnect the graph.

SECTION 7.1

DIRECTED GRAPHS AND BINARY RELATIONS; WARSHALL’S ALGORITHM

In this section we confine our attention to (unweighted) directed graphs with no parallel arcs. (In a directed graph, two arcs from node a to node b would be parallel, but one arc from a to b and another from b to a are not parallel arcs.) Consider the adjacency matrix of the graph (assuming some arbitrary ordering of the n nodes, which we always assume when discussing the adjacency matrix of a graph). This will be an $n \times n$ matrix, not necessarily symmetric. Furthermore, because there are no parallel arcs in the graph, the adjacency matrix will be a Boolean matrix, that is, a matrix whose only elements are 0s and 1s. Conversely, given an $n \times n$ Boolean matrix, we can reconstruct the directed graph that the matrix represents, and it will have no parallel arcs. Thus there is a one-to-one correspondence, which we can picture as



Now we will see how binary relations tie in to this correspondence.

Directed Graphs and Binary Relations

Suppose G is a directed graph with n nodes and no parallel arcs. Let N be the set of nodes. If (n_i, n_j) is an ordered pair of nodes, then there either is or is not an arc in G from n_i to n_j . We can use this property to define a binary relation on the set N :

$$n_i \rho n_j \leftrightarrow \text{there is an arc in } G \text{ from } n_i \text{ to } n_j$$

This relation is the **adjacency relation** of the graph.

EXAMPLE 1

For the directed graph of Figure 7.1, the adjacency relation is $\{(1, 2), (1, 3), (3, 3), (4, 1), (4, 2), (4, 3)\}$.

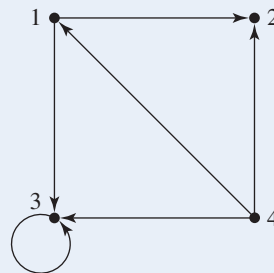


Figure 7.1

Conversely, if ρ is a binary relation on a set N , we can define a directed graph G with N as the set of nodes, and an arc from n_i to n_j if and only if $n_i \rho n_j$. G will have no parallel arcs.

EXAMPLE 2

For the set $N = \{1, 2, 3, 4\}$ and the binary relation $\{(1, 4), (2, 3), (2, 4), (4, 1)\}$ on N , we obtain the associated directed graph shown in Figure 7.2.

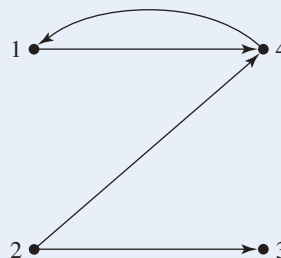


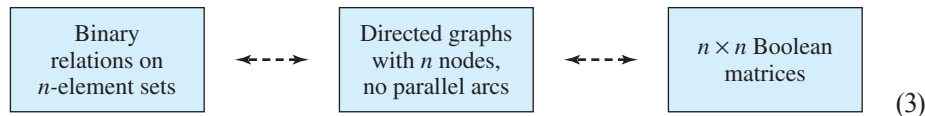
Figure 7.2

We now have another one-to-one correspondence:



(2)

Of course, a one-to-one correspondence means the existence of a bijection. If function composition is carried out on the bijections in (1) and (2), the result is a bijection that gives us a one-to-one correspondence between binary relations and matrices. Thus we have three equivalent sets:



An item from any of the three sets has corresponding representations in the other two sets.

PRACTICE 1

Give the collection of ordered pairs in the adjacency relation for the following Boolean matrix; also draw the directed graph.

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

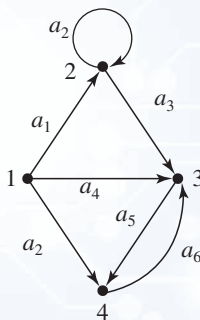
Recall the reflexive, symmetric, antisymmetric, and transitive properties of a binary relation on a set that we studied in Chapter 5. If a binary relation on a set N has a certain property, it will be reflected in the corresponding graph and the corresponding Boolean matrix. Conversely, certain characteristics of a directed graph or of a Boolean matrix imply certain properties of the corresponding adjacency relation.

EXAMPLE 3

If ρ is a reflexive relation on a set N , then for each $n_i \in N$, $n_i \rho n_i$. In the corresponding directed graph there will be a loop at each node, and in the corresponding Boolean matrix there will be 1s on the main diagonal. ■

PRACTICE 2

Explain why the corresponding binary relation is not antisymmetric for the directed graph in Figure 6.13 reproduced here.



In Chapter 5 we represented partial orderings on a set by using a Hasse diagram. How does this representation differ from the directed graph representation? The Hasse diagram is a simplification of the directed graph representation. Suppose that G is the directed graph representation of a partial ordering. Because a partial ordering is reflexive, G will have a loop at each node. We can eliminate these loops in the Hasse diagram without losing any information because we know that each node has a loop; that is, each node is related to itself. Because a partial ordering is transitive, if $a\rho b$ and $b\rho c$, then $a\rho c$. In the directed graph there would be an arc from a to b , an arc from b to c , and an arc from a to c . In the Hasse diagram we can eliminate the arc from a to c without losing any information if we keep the transitive property in mind. Finally, the Hasse diagram is not a directed graph at all, but we did impose the convention that if a is an immediate predecessor of b , then node a will appear below node b in the Hasse diagram. Thus we could achieve a directed graph from the Hasse diagram by making all the arc directions point upward. The antisymmetry property prevents any potential conflict where node a should be below node b and node b should be below node a .

In Chapter 5 we also noted set operations that could be performed on two binary relations ρ and σ on a set N , $\rho \cup \sigma$ and $\rho \cap \sigma$. The relation $\rho \cup \sigma$ is the union of the ordered pairs in ρ or σ , while $\rho \cap \sigma$ is the intersection of the ordered pairs in ρ and σ . Let \mathbf{R} and \mathbf{S} be the Boolean matrices for ρ and σ , respectively. The Boolean matrix for $\rho \cup \sigma$ will have a 1 in position i, j if and only if there is a 1 in position i, j of \mathbf{R} or a 1 in position i, j of \mathbf{S} . Each entry in the Boolean matrix for $\rho \cup \sigma$ is thus the maximum of the two corresponding entries in \mathbf{R} and \mathbf{S} , so the Boolean matrix for $\rho \cup \sigma$ is $\mathbf{R} \vee \mathbf{S}$ (see the discussion of Boolean matrix operations in Section 5.7). Similarly, the Boolean matrix for $\rho \cap \sigma$ will have a 1 in position i, j if and only if there is a 1 in position i, j of both \mathbf{R} and \mathbf{S} . Therefore the Boolean matrix for $\rho \cap \sigma$ is $\mathbf{R} \wedge \mathbf{S}$.

Reachability

The “reachability” property has an interesting interpretation in each of the three equivalent forms in (3)—directed graph, adjacency relation, and adjacency matrix. We already have a definition for this term for directed graphs from Section 6.1, which we’ll restate now.

DEFINITION REACHABLE NODE

In a directed graph, node n_j is **reachable** from node n_i if there is a path from n_i to n_j .

EXAMPLE 4

In the directed graph of Figure 7.2, node 3 is not reachable from node 4 or node 1. Node 1 is reachable from node 2 by the path 2–4–1.

In a system modeled by a directed graph (a data flow diagram, for example) with a “start node,” any node that is unreachable from the start node can never affect the system and thus can be eliminated. If the directed graph represents something like airline routes or communication paths in a computer network, it would

be undesirable to have some node be unreachable from some other node. Thus the ability to test reachability has very practical applications.

The adjacency matrix \mathbf{A} of a directed graph G with n nodes and no parallel arcs will have a 1 in position i, j if there is an arc from n_i to n_j . This would be a path of length 1 from n_i to n_j . The adjacency matrix by itself therefore tells us about a limited form of reachability, via length-1 paths. However, let us perform the Boolean matrix multiplication $\mathbf{A} \times \mathbf{A}$. We'll denote this product by $\mathbf{A}^{(2)}$ to distinguish it from \mathbf{A}^2 , the result of $\mathbf{A} \cdot \mathbf{A}$ using ordinary matrix multiplication. Recalling from Section 5.7 the definition of Boolean matrix multiplication, the i, j entry of $\mathbf{A}^{(2)}$ is given by

$$\mathbf{A}^{(2)}[i, j] = \bigvee_{k=1}^n (a_{ik} \wedge a_{kj}) \quad (4)$$

REMINDER

To compute $\mathbf{A}^{(2)}[i, j]$ write two copies of \mathbf{A} side by side. Run a left-hand finger along row i of the left copy and a right-hand finger down column j of the right copy. The value is 1 if and only if both fingers hit a 1 at the same time.

If a term such as $a_{i2} \wedge a_{2j}$ in this sum is 0, then either $a_{i2} = 0$ or $a_{2j} = 0$ (or both), and there is either no path of length 1 from n_i to n_2 or no path of length 1 from n_2 to n_j (or both). Thus there are no paths of length 2 from n_i to n_j passing through n_2 . If $a_{i2} \wedge a_{2j}$ is not 0, then both $a_{i2} = 1$ and $a_{2j} = 1$. Then there is a path of length 1 from n_i to n_2 and a path of length 1 from n_2 to n_j , so there is a path of length 2 from n_i to n_j passing through n_2 . A path of length 2 from n_i to n_j will exist if and only if there is a path of length 2 passing through at least one of the nodes from 1 to n , that is, if and only if at least one of the terms in the sum (4) is 1 and therefore $\mathbf{A}^{(2)}[i, j] = 1$. Therefore the entries in $\mathbf{A}^{(2)}$ tell us about reachability via length-2 paths.

PRACTICE 3 Find \mathbf{A} for the graph of Figure 7.2 and compute $\mathbf{A}^{(2)}$. What does the 2,1 entry indicate? ■

The matrix $\mathbf{A}^{(2)}$ indicates the presence or absence of length-2 paths. We might surmise that this result holds for arbitrary powers and path lengths.

THEOREM ON BOOLEAN ADJACENCY MATRICES AND REACHABILITY

If \mathbf{A} is the Boolean adjacency matrix for a directed graph G with n nodes and no parallel arcs, then $\mathbf{A}^{(m)}[i, j] = 1$ if and only if there is a path of length m from node n_i to node n_j .

Proof: A proof by induction on m is called for. We have already shown the result true for $m = 1$ (and $m = 2$). Suppose that $\mathbf{A}^{(p)}[i, j] = 1$ if and only if there is a path of length p from n_i to n_j . We know that

$$\mathbf{A}^{(p+1)}[i, j] = \bigvee_{k=1}^n (\mathbf{A}^{(p)}[i, k] \wedge a_{kj})$$

which will equal 1 if and only if at least one term, say $\mathbf{A}^{(p)}[i, q] \wedge a_{qj} = 1$, or $\mathbf{A}^{(p)}[i, q] = 1$ and $a_{qj} = 1$. This condition will be true if and only if there is a path

of length p from n_i to n_q (by the inductive hypothesis) and there is a path of length 1 from n_q to n_j , which means there is a path of length $p + 1$ from n_i to n_j . End of proof.

PRACTICE 4 From the graph of Figure 7.2, what would you expect for the value of entry 2,1 in $\mathbf{A}^{(4)}$? Compute $\mathbf{A}^{(4)}$ and check this value. ■

If node n_j is reachable from node n_i , it is by a path of some length. Such a path will be shown by a 1 as the i, j entry in \mathbf{A} or $\mathbf{A}^{(2)}$ or $\mathbf{A}^{(3)}$ and so on, but we cannot compute an infinite number of matrix products. Fortunately, there is a limit to how far in this list we have to look. If there are n nodes in the graph, then any path with n or more arcs, and therefore $n + 1$ or more nodes, must have a repeated node. This is a consequence of the pigeonhole principle—there are n “bins” (distinct nodes) into which we are putting more than n objects (the nodes in a path with n or more arcs). The section of a path lying between the repeated nodes is a cycle. If $n_i \neq n_j$, the cycle can be eliminated to make a shorter path; then if a path exists from n_i to n_j , there will be such a path of length at most $n - 1$. If $n_i = n_j$, then the cycle could be the entire path from n_i to n_i with maximum length n ; although we could eliminate this cycle (noting that any node may be considered reachable from itself), we will retain it to show that a nontrivial path does exist from n_i to n_i .

Consequently, whether $n_i = n_j$ or $n_i \neq n_j$, we need never look for a path from n_i to n_j of length greater than n . Therefore to determine reachability, we need only consult element i, j in $\mathbf{A}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(n)}$. Alternatively, we can define a **reachability matrix \mathbf{R}** by

$$\mathbf{R} = \mathbf{A} \vee \mathbf{A}^{(2)} \vee \dots \vee \mathbf{A}^{(n)}$$

Then n_j is reachable from n_i if and only if entry i, j in \mathbf{R} is positive.

We now see how reachability in a graph can be expressed in terms of the adjacency matrix. How is reachability represented in terms of the adjacency relation that corresponds to the graph?

If ρ is the adjacency relation for a graph G , we let $\rho^{\mathbf{R}}$ denote the binary relation of reachability; that is, $(n_i, n_j) \in \rho^{\mathbf{R}}$ exactly when there is a path in G from n_i to n_j . Then we can show that $\rho^{\mathbf{R}}$ is the transitive closure of ρ . Recall from the definition of closure of a relation that the transitive closure of ρ is a relation that is transitive, contains ρ , and is a subset of any transitive relation containing ρ .

To see that $\rho^{\mathbf{R}}$ is transitive, let (n_i, n_j) and (n_j, n_k) belong to $\rho^{\mathbf{R}}$. Then there is a path in G from n_i to n_j and a path in G from n_j to n_k . Therefore there is a path in G from n_i to n_k , and (n_i, n_k) belongs to $\rho^{\mathbf{R}}$. To see that $\rho^{\mathbf{R}}$ contains ρ , let (n_i, n_j) belong to ρ . Then there is an arc from n_i to n_j in G , which means there is a path of length 1 from n_i to n_j , and (n_i, n_j) belongs to $\rho^{\mathbf{R}}$. Finally, suppose σ is any transitive relation on the nodes of G that includes ρ , and let (n_i, n_j) belong to $\rho^{\mathbf{R}}$. This means that there is a path from n_i to n_j using, say, nodes $n_i, n_x, n_y, \dots, n_w, n_j$. Then there is an arc from each node in this path to the next, and the ordered pairs $(n_i, n_x), (n_x, n_y), \dots, (n_w, n_j)$, all belong to ρ , and therefore all belong to σ . Because σ is

transitive, (n_i, n_j) belongs to σ , and $\rho^{\mathbf{R}}$ is a subset of σ . Therefore $\rho^{\mathbf{R}}$ is the transitive closure of ρ .

To summarize, corresponding to the three equivalent representations of adjacency relation ρ , directed graph G , and adjacency matrix \mathbf{A} , we have

$$(n_i, n_j) \text{ belongs to the transitive closure of } \rho \iff n_j \text{ is reachable from } n_i \text{ in } G \iff \mathbf{R}[i, j] = 1 \text{ where } \mathbf{R} = \mathbf{A} \vee \mathbf{A}^{(2)} \vee \dots \vee \mathbf{A}^{(n)}$$

EXAMPLE 5

Let G be the directed graph in Figure 7.3; G has 5 nodes.

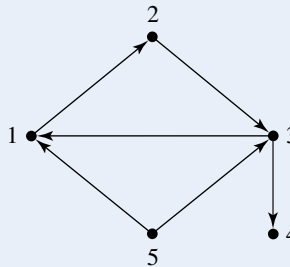


Figure 7.3

The adjacency matrix \mathbf{A} for G is

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

The adjacency relation ρ is $\rho = \{(1, 2), (2, 3), (3, 1), (3, 4), (5, 1), (5, 3)\}$.

The successive powers of \mathbf{A} are

$$\mathbf{A}^{(2)} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{A}^{(3)} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{A}^{(4)} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix} \qquad \mathbf{A}^{(5)} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

REMINDER

Don't try to compute matrix products without writing the two matrices side by side; you'll surely make a mistake.

These matrices indicate, for example, that there is a path of length 2 from 2 to 1 because $\mathbf{A}^{(2)}[2, 1] = 1$ (the path is 2–3–1), and there is a path of length 4 from 5 to 3 because, $\mathbf{A}^{(4)}[5, 3] = 1$ (the path is 5–3–1–2–3), but there is no path of length 3 from 1 to 3 because $\mathbf{A}^{(3)}[1, 3] = 0$.

The reachability matrix \mathbf{R} is the Boolean sum of \mathbf{A} , $\mathbf{A}^{(2)}$, $\mathbf{A}^{(3)}$, $\mathbf{A}^{(4)}$, and $\mathbf{A}^{(5)}$:

$$\mathbf{R} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

The 1 values in \mathbf{R} indicate that there are paths in G from nodes 1, 2, 3, and 5 to every node except 5, but no path from node 4 to anywhere, which can be confirmed by looking at Figure 7.3.

We have proved that the 1 entries in \mathbf{R} mark the ordered pairs of nodes that belong to the transitive closure of ρ . The transitive closure will therefore be the following set of ordered pairs:

$$\{(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (2, 4), \\ (3, 1), (3, 2), (3, 3), (3, 4), (5, 1), (5, 2), (5, 3), (5, 4)\}$$

Beginning with ρ and following the ad hoc procedure described in Chapter 5 for finding the transitive closure of a relation, we see that to obtain transitivity, we must first add the pairs (1, 3), (2, 1), (2, 4), (3, 2), (5, 2), and (5, 4). Reviewing the new set, we see that we must also add (1, 1), (1, 4), (2, 2), and (3, 3). The resulting collection of ordered pairs is transitive (and agrees with what we obtained earlier). ●

PRACTICE 5 Compute \mathbf{R} for the directed graph of Figure 7.2. What information does column 2 convey? ■

In Chapter 5, we promised a better algorithm to find the transitive closure of a relation. Here it is. Write the binary relation in adjacency matrix form and compute

$$\mathbf{R} = \mathbf{A} \vee \mathbf{A}^{(2)} \vee \cdots \vee \mathbf{A}^{(n)}$$

How much work is required to carry out this algorithm? The expression for \mathbf{R} indicates that Boolean matrix operations are to be done, but matrix operations in turn require Boolean **and** and Boolean **or** operations on matrix elements. We will therefore use Boolean **and** and Boolean **or** as the measure of work. In Section 5.7, we noted that ordinary matrix multiplication of two $n \times n$ matrices requires $\Theta(n^3)$ multiplications and additions; by a similar argument, Boolean matrix

multiplication of two $n \times n$ Boolean matrices requires $\Theta(n^3)$ Boolean **and/or** operations. The algorithm to compute \mathbf{R} requires $n - 1$ Boolean matrix multiplications (to find the products $\mathbf{A}^{(2)}, \mathbf{A}^{(3)}, \dots, \mathbf{A}^{(n)}$). To compute $n - 1$ such products require $(n - 1)\Theta(n^3) = \Theta(n^4)$ Boolean operations. To compute $\mathbf{C} \vee \mathbf{D}$ where \mathbf{C} and \mathbf{D} are two $n \times n$ Boolean matrices requires n^2 Boolean **or** operations. To compute \mathbf{R} , $n - 1$ such matrix operations are required, so $(n - 1)n^2 = \Theta(n^3)$ Boolean **or** operations are performed. The total amount of work is $\Theta(n^4) + \Theta(n^3) = \Theta(n^4)$.

Next we discuss a more efficient algorithm for computing the transitive closure of a relation (or the reachability matrix of a graph).

Warshall's Algorithm

For a graph G with n nodes, Warshall's algorithm computes a sequence of $n + 1$ matrices $\mathbf{M}_0, \mathbf{M}_1, \mathbf{M}_2, \dots, \mathbf{M}_n$. For each $k, 0 \leq k \leq n, \mathbf{M}_k[i, j] = 1$ if and only if there is a path in G from n_i to n_j whose interior nodes (i.e., nodes that are not the endpoints of the path) come only from the set of nodes $\{n_1, n_2, \dots, n_k\}$.

Let us examine the "end conditions." When $k = 0$, the set $\{n_1, n_2, \dots, n_0\}$ is the empty set, so $\mathbf{M}_0[i, j] = 1$ if and only if there is a path in G from n_i to n_j whose interior nodes come from the empty set; that is, there are no interior nodes. The path from n_i to n_j must then consist only of the endpoints and one connecting arc, so n_i and n_j are adjacent nodes. Thus $\mathbf{M}_0 = \mathbf{A}$. The other end condition occurs when $k = n$. Then the set $\{n_1, n_2, \dots, n_n\}$ consists of all the nodes in G , so there is really no restriction at all on the interior nodes in the path and $\mathbf{M}_n[i, j] = 1$ if and only if there is a path from n_i to n_j , which means that $\mathbf{M}_n = \mathbf{R}$.

Therefore Warshall's algorithm begins with $\mathbf{A} = \mathbf{M}_0$ and successively computes $\mathbf{M}_1, \mathbf{M}_2, \dots, \mathbf{M}_n = \mathbf{R}$. This computation can be defined inductively. The base case is to let $\mathbf{M}_0 = \mathbf{A}$. Now assume that \mathbf{M}_k has been computed, and consider how to compute \mathbf{M}_{k+1} or, more specifically, $\mathbf{M}_{k+1}[i, j]$. We have $\mathbf{M}_{k+1}[i, j] = 1$ if and only if there is a path from n_i to n_j whose interior nodes come only from the set $\{n_1, n_2, \dots, n_{k+1}\}$. This can happen in two ways:

1. All the interior nodes come from $\{n_1, n_2, \dots, n_k\}$, in which case $\mathbf{M}_k[i, j] = 1$. We should therefore carry forward any 1 entries in \mathbf{M}_k into \mathbf{M}_{k+1} .
2. Node n_{k+1} is an interior node. We can assume that n_{k+1} is an interior node only once, because cycles can be eliminated from a path. Then there must be a path from n_i to n_{k+1} whose interior nodes come from $\{n_1, n_2, \dots, n_k\}$ and a path from n_{k+1} to n_j whose interior nodes come from $\{n_1, n_2, \dots, n_k\}$. This means that $\mathbf{M}_k[i, k+1] = 1$ and $\mathbf{M}_k[k+1, j] = 1$, which is to say that $\mathbf{M}_k[i, k+1] \wedge \mathbf{M}_k[k+1, j] = 1$; this condition can be tested because our assumption is that \mathbf{M}_k has already been computed.

In the following pseudocode version of Warshall's algorithm, the initial value of matrix \mathbf{M} is \mathbf{A} . Each pass through the outer loop computes the next matrix in the sequence $\mathbf{M}_1, \mathbf{M}_2, \dots, \mathbf{M}_n = \mathbf{R}$.

ALGORITHM *WARSHALL'S ALGORITHM*

```

Warshall( $n \times n$  Boolean matrix  $\mathbf{M}$ )
//Initially,  $\mathbf{M}$  = adjacency matrix of a directed graph  $G$  with no parallel arcs

  for  $k = 0$  to  $n - 1$  do
    for  $i = 1$  to  $n$  do
      for  $j = 1$  to  $n$  do
         $\mathbf{M}[i, j] = \mathbf{M}[i, j] \vee (\mathbf{M}[i, k + 1] \wedge \mathbf{M}[k + 1, j])$ 
      end for
    end for
  end for
//at termination,  $\mathbf{M}$  = reachability matrix of  $G$ 
end Warshall

```

This pseudocode gives a nice neat description of Warshall's algorithm, which can be implemented as computer code rather easily. These steps are confusing to do by hand, however, requiring some bookkeeping to keep track of all the indices. We can write the algorithm more informally, making it easier to do manually. Suppose again that matrix \mathbf{M}_k in the sequence exists and we are trying to write row i of the next matrix in the sequence. For the various values of j , we must evaluate the expression

$$\mathbf{M}[i, j] \vee (\mathbf{M}[i, k + 1] \wedge \mathbf{M}[k + 1, j]) \quad (5)$$

If entry $\mathbf{M}[i, k + 1]$ is 0, then $\mathbf{M}[i, k + 1] \wedge \mathbf{M}[k + 1, j] = 0$ for all j . Expression (5) then reduces to

$$\mathbf{M}[i, j] \vee 0 = \mathbf{M}[i, j]$$

In other words, row i of the matrix remains unchanged. If, on the other hand, entry $\mathbf{M}[i, k + 1]$ is 1, then $\mathbf{M}[i, k + 1] \wedge \mathbf{M}[k + 1, j] = \mathbf{M}[k + 1, j]$ for all j . Expression (5) then becomes

$$\mathbf{M}[i, j] \vee \mathbf{M}[k + 1, j]$$

In other words, row i of the matrix becomes the Boolean **or** of the current row i and the current row $k + 1$.

Table 7.1 describes the (informal) steps to compute entries in \mathbf{M}_{k+1} from matrix \mathbf{M}_k .

TABLE 7.1

1. Consider column $k + 1$ in \mathbf{M}_k .
2. For each row with a 0 entry in this column, copy that row to \mathbf{M}_{k+1} .
3. For each row with a 1 entry in this column, **or** that row with row $k + 1$ and write the resulting row in \mathbf{M}_{k+1} .

EXAMPLE 6 For the graph of Example 5, the initial matrix \mathbf{M}_0 is the adjacency matrix.

$$\mathbf{M}_0 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

We know \mathbf{M}_0 (so $k = 0$) and we want to compute \mathbf{M}_1 ($k + 1 = 1$). Using step 1 of Table 7.1, we consider column 1 of \mathbf{M}_0 . Using step 2 of Table 7.1, rows 1, 2, and 4 of \mathbf{M}_0 contain 0s in column 1, so these rows get copied directly to \mathbf{M}_1 :

$$\mathbf{M}_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Now we finish up by using step 3 of Table 7.1 Row 3 of column 1 of \mathbf{M}_0 contains a 1, so row 3 of \mathbf{M}_0 is **or**-ed with row 1 of \mathbf{M}_0 and the result becomes the new row 3:

$$\mathbf{M}_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Row 5 of column 1 of \mathbf{M}_0 contains a 1, so row 5 is **or**-ed with row 1 and the result becomes the new row 5:

$$\mathbf{M}_1 = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

To compute the entries in \mathbf{M}_2 , consider column 2. Rows 2 and 4 (the 0 positions in column 2) will be copied unchanged. Row 1 will be **or**-ed with row 2 to give the new row 1, row 3 will be **or**-ed with row 2 to give the new row 3, and row 5 will be **or**-ed with row 2 to give the new row 5:

$$\mathbf{M}_2 = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

\mathbf{M}_3 is computed in a similar fashion:

$$\mathbf{M}_3 = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

\mathbf{M}_4 and \mathbf{M}_5 will be the same as \mathbf{M}_3 ; row 4 is all 0s, so any row that gets **or**-ed with it will be unchanged, and column 5 is all 0s so all rows are copied directly. In terms of the graph, no new 1 entries are produced because there are no paths from 4 to any node or from any node to 5. Thus $\mathbf{M}_3 = \mathbf{M}_4 = \mathbf{M}_5 = \mathbf{R}$, as computed in Example 5. Note, however, that the matrices computed by Warshall's algorithm, except for $\mathbf{M}_0 = \mathbf{A}$ and $\mathbf{M}_n = \mathbf{R}$, do not agree with the matrices that are powers of \mathbf{A} used in our previous algorithm for \mathbf{R} . •

Each pass through the outer loop of Warshall's algorithm modifies in place the matrix that existed at the end of the previous pass. Warshall's algorithm requires no additional storage for other matrices, even though we wrote down new matrices in our example. There is one more point we need to check. Because we are modifying the (only) matrix as we go along, during any one pass through the outer loop, some of the entries will belong to \mathbf{M}_{k+1} while others still belong to \mathbf{M}_k . Specifically, on pass $k + 1$, we may consider $\mathbf{M}[i, k + 1] \wedge \mathbf{M}[k + 1, j]$ in Expression (5), where these values have already been computed on this pass and therefore represent $\mathbf{M}_{k+1}[i, k + 1]$ and $\mathbf{M}_{k+1}[k + 1, j]$ rather than the values $\mathbf{M}_k[i, k + 1]$ and $\mathbf{M}_k[k + 1, j]$ we used in our justification for this algorithm. Can there be a case where the values $\mathbf{M}_{k+1}[i, k + 1]$ and $\mathbf{M}_{k+1}[k + 1, j]$ are 1, so that a 1 value goes into $\mathbf{M}_{k+1}[i, j]$, whereas the values $\mathbf{M}_k[i, k + 1]$ and $\mathbf{M}_k[k + 1, j]$ are 0? No—if $\mathbf{M}_{k+1}[i, k + 1] = 1$, there is a path from n_i to n_{k+1} with interior nodes drawn from the set $\{n_1, n_2, \dots, n_{k+1}\}$. However, because n_{k+1} is an endpoint and cycles can be eliminated, there must also be a path with interior nodes drawn from the set $\{n_1, n_2, \dots, n_k\}$ so that $\mathbf{M}_k[i, k + 1] = 1$. A similar argument holds for $\mathbf{M}_{k+1}[k + 1, j]$.

PRACTICE 6 Use Warshall's algorithm (formally or informally) to compute \mathbf{R} for the graph of Figure 7.2. Compare your answer with that for Practice 5. ■

How much work does Warshall's algorithm require as measured by the number of Boolean **and/or** operations? Consider the formal algorithm. The single assignment statement in the algorithm lies within a triply nested loop; it will be executed n^3 times. Each execution of the assignment statement requires one **and** and one **or**; therefore, the total amount of work is $2n^3 = \Theta(n^3)$. Recall that our previous algorithm for computing \mathbf{R} was an $\Theta(n^4)$ algorithm.

SECTION 7.1 REVIEW

TECHNIQUES

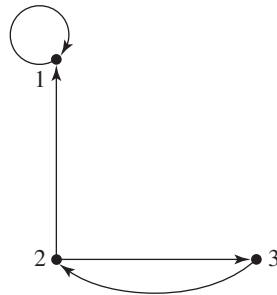
- Find any two of the adjacency relation, directed graph, or adjacency matrix representation, given the third.
- **W** Compute the reachability matrix \mathbf{R} for a graph G (or, equivalently, find the transitive closure of the adjacency relation on G) by using the formula $\mathbf{R} = \mathbf{A} \vee \mathbf{A}^{(2)} \vee \cdots \vee \mathbf{A}^{(n)}$ and by using Warshall's algorithm.

MAIN IDEAS

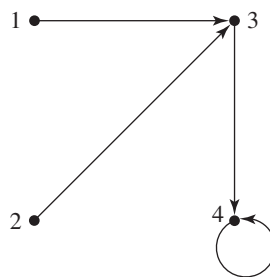
- There is a one-to-one correspondence between a directed graph G with no parallel arcs, the adjacency relation on G , and the adjacency matrix for G (with respect to some arbitrary ordering of the nodes).
- The reachability matrix of a graph G also represents the transitive closure of the adjacency relation on G .
- The reachability matrix for a graph can be computed with $\Theta(n^4)$ Boolean **and/or** operations by summing powers of the adjacency matrix \mathbf{A} or with $\Theta(n^3)$ Boolean **and/or** operations by using Warshall's algorithm.

EXERCISES 7.1

1. Find the adjacency matrix and adjacency relation for the following graph.



2. Find the adjacency matrix and adjacency relation for the following graph.



3. Find the corresponding directed graph and adjacency relation for the following adjacency matrix.

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

4. Find the corresponding directed graph and adjacency relation for the following adjacency matrix.

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

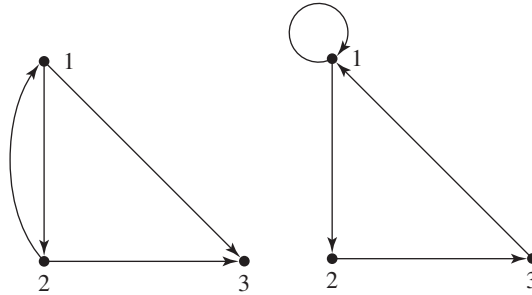
5. Given the adjacency relation $\rho = \{(1, 4), (1, 5), (1, 6), (6, 2), (6, 3), (6, 5)\}$ on the set $N = \{1, 2, 3, 4, 5, 6\}$, find the corresponding directed graph and adjacency matrix.
6. Given the adjacency relation $\rho = \{(2, 1), (3, 2), (3, 3), (3, 4), (4, 5), (6, 3), (6, 6)\}$ on the set $N = \{1, 2, 3, 4, 5, 6\}$, find the corresponding directed graph and adjacency matrix.
7. Let ρ be a binary relation defined on the set $\{0, 1, 2, 3, 4, 5, 6\}$ by $x\rho y \leftrightarrow y = x + 2$. Draw the associated directed graph.
8. Let ρ be a binary relation defined on the set $\{0, \pm 1, \pm 2, \pm 4, \pm 16\}$ by $x\rho y \leftrightarrow y = x^2$. Draw the associated directed graph.
9. Describe a property of a directed graph whose adjacency matrix is symmetric.
10. Describe the directed graph whose adjacency matrix has all 0s on the main diagonal and 1s everywhere else.
11. Describe the directed graph whose adjacency matrix has all 1s in row 1 and column 1, and 0s elsewhere.
12. Describe the directed graph whose adjacency matrix has 1s in positions $(i, i + 1)$ for $1 \leq i \leq n - 1$, a 1 in position $(n, 1)$, and 0s elsewhere.
13. Describe a property of a directed graph whose adjacency relation is irreflexive (see Exercise 26, Section 5.1).
14. Describe a property of the adjacency matrix of a graph whose adjacency relation is antisymmetric.
15. Adjacency relations ρ and σ have the following associated adjacency matrices \mathbf{R} and \mathbf{S} . Find the adjacency matrices associated with the relations $\rho \cup \sigma$ and $\rho \cap \sigma$.

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{S} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

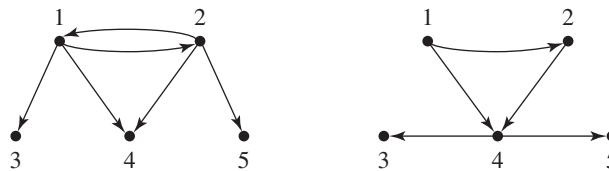
16. Adjacency relations ρ and σ have the following associated adjacency matrices \mathbf{R} and \mathbf{S} . Find the adjacency matrices associated with the relations $\rho \cup \sigma$ and $\rho \cap \sigma$.

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{S} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

17. The two directed graphs that follow have adjacency relations ρ and σ . Draw the graphs associated with the relations $\rho \cup \sigma$ and $\rho \cap \sigma$.



18. The two directed graphs that follow have adjacency relations ρ and σ . Draw the graphs associated with the relations $\rho \cup \sigma$ and $\rho \cap \sigma$.



19. Let \mathbf{A} be the matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Find the products \mathbf{A}^2 and $\mathbf{A}^{(2)}$.

20. Let \mathbf{A} be the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

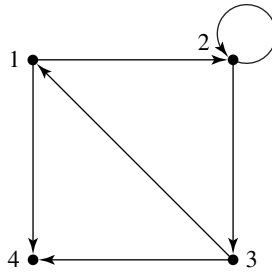
Find the products \mathbf{A}^2 and $\mathbf{A}^{(2)}$.

21. The definition of a *connected graph* can be extended to directed graphs. Describe the reachability matrix \mathbf{R} for a connected, directed graph.

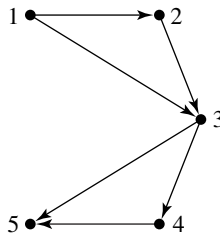
22. Describe the directed graph with the following reachability matrix \mathbf{R} .

$$\mathbf{R} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

23. For the following graph, write the reachability matrix \mathbf{R} by simply inspecting the graph.



24. For the following graph, write the reachability matrix \mathbf{R} by simply inspecting the graph.



For Exercises 25–30, compute the reachability matrix \mathbf{R} by using the formula $\mathbf{R} = \mathbf{A} \vee \mathbf{A}^{(2)} \vee \cdots \vee \mathbf{A}^{(n)}$

25. Exercise 1

27. Exercise 3

29. Exercise 5

26. Exercise 2

28. Exercise 4

30. Exercise 6

For Exercises 31–36, compute the reachability matrix \mathbf{R} by using Warshall's algorithm.

31. Exercise 1

33. Exercise 3

35. Exercise 5

32. Exercise 2

34. Exercise 4

36. Exercise 6

37. Given the binary relation $\rho = \{(1, 3), (3, 2), (2, 3)\}$ on the set $\{1, 2, 3\}$, use Warshall's algorithm to find the transitive closure of ρ .

38. Given the binary relation $\rho = \{(1, 2), (2, 3), (4, 1)\}$ on the set $\{1, 2, 3, 4\}$, use Warshall's algorithm to find the transitive closure of ρ .

39. Use Warshall's algorithm to find the transitive closure of the following binary relations on the set $\{1, 2, 3\}$ (see Exercise 23 in Section 5.1).

a. $\rho = \{(1, 3), (3, 3), (3, 1), (2, 2), (2, 3), (1, 1), (1, 2)\}$

b. $\rho = \{(1, 1), (3, 3), (2, 2)\}$

c. $\rho = \{(1, 1), (1, 2), (2, 3), (3, 1), (1, 3)\}$

d. $\rho = \{(1, 1), (1, 2), (2, 3), (1, 3)\}$

40. Use Warshall's algorithm to find the transitive closure of the following binary relations on the set $\{0, 1, 2, 4, 6\}$ (see Exercise 24 in Section 5.1).

a. $\rho = \{(0, 0), (1, 1), (2, 2), (4, 4), (6, 6), (0, 1), (1, 2), (2, 4), (4, 6)\}$

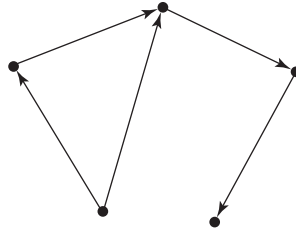
b. $\rho = \{(0, 1), (1, 0), (2, 4), (4, 2), (4, 6), (6, 4)\}$

c. $\rho = \{(0, 1), (1, 2), (0, 2), (2, 0), (2, 1), (1, 0), (0, 0), (1, 1), (2, 2)\}$

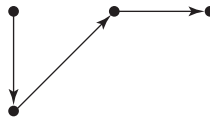
d. $\rho = \{(0, 0), (1, 1), (2, 2), (4, 4), (6, 6), (4, 6), (6, 4)\}$

e. $\rho = \emptyset$

41. The following directed graph represents a binary relation ρ on the nodes. Draw the directed graph that would represent the transitive closure of ρ .

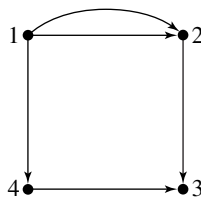


42. The following directed graph represents a binary relation ρ on the nodes. Draw the directed graph that would represent the transitive closure of ρ .

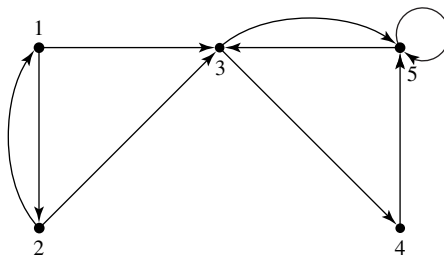


Exercises 43–46 use regular matrix multiplication to obtain information about a graph.

43. Let G be a directed graph, possibly with parallel arcs, and let \mathbf{A} be its adjacency matrix. Then \mathbf{A} may not be a Boolean matrix. Prove that the i, j entry of matrix \mathbf{A}^2 is the number of paths of length 2 from node i to node j .
44. Let \mathbf{A} be the adjacency matrix of a directed graph G , possibly with parallel arcs. Prove that the i, j entry of matrix \mathbf{A}^n gives the number of paths of length n from node i to node j .
45. For the following graph, count the number of paths of length 2 from node 1 to node 3. Check by computing \mathbf{A}^2 .



46. For the following graph, count the number of paths of length 4 from node 1 to node 5. Check by computing \mathbf{A}^4 .



SECTION 7.2 EULER PATH AND HAMILTONIAN CIRCUIT

Euler Path Problem

The Euler path problem (the highway inspector problem) originated many years ago. Swiss mathematician Leonhard Euler (pronounced “oiler”) (1707–1783) was intrigued by a puzzle popular among the townsfolk of Königsberg (an East Prussian city later called Kaliningrad, which is in Russia). The river flowing through the city branched around an island. Various bridges crossed the river as shown in Figure 7.4.

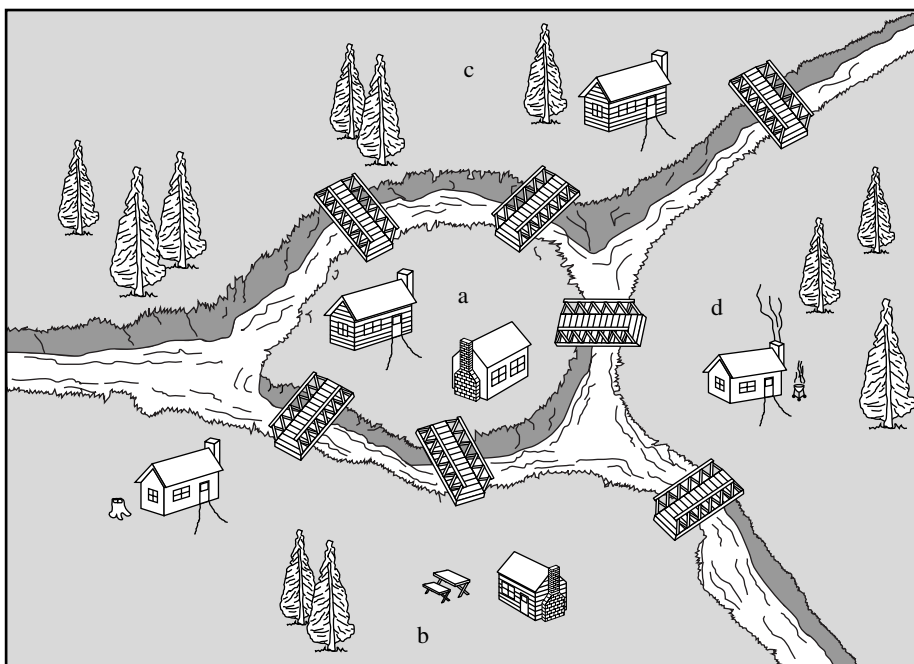


Figure 7.4

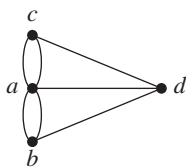


Figure 7.5

The puzzle was to decide whether a person could walk through the city crossing each bridge only once. It is possible to answer the question by trial and error, listing (or walking) all possible routes, so some dedicated Königsberger could have solved this particular puzzle. Euler solved this problem, and indeed a more general version of the problem, by a better mechanism than trial and error. The problem is usually represented as a graph (Figure 7.5) where the bridges are arcs and the land masses (labeled a through d) are nodes. The more general question is to determine when an Euler path exists in any graph.¹

¹Euler did not actually represent the problem as a graph, and such a representation was not connected to Euler’s work until over 150 years later. See “The Truth about Königsberg” by Brian Hopkins and Robin Wilson, *The College Mathematics Journal*, May 2004.

DEFINITION EULER PATH

An **Euler path** in a graph G is a path that uses each arc of G exactly once.

PRACTICE 7

Do Euler paths exist for either graph in Figure 7.6? (Use trial and error to answer. This is the old children's game of whether you can trace the whole graph without lifting your pencil and without retracing any arcs.)

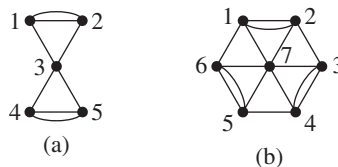


Figure 7.6

For this discussion we will assume that all graphs are connected, since an Euler path generally cannot exist otherwise. Whether an Euler path exists in a given graph hinges on the degrees of its nodes. A node is **even** if its degree is even and **odd** if its degree is odd. It turns out that every graph has an even number of odd nodes. To see this, choose any graph and let N be the number of odd nodes in it, $N(1)$ the number of nodes of degree 1, $N(2)$ the number of nodes of degree 2, and so on. Then the sum S of the degrees of all the nodes of the graph is

$$S = 1 \cdot N(1) + 2 \cdot N(2) + 3 \cdot N(3) + \cdots + k \cdot N(k) \quad (1)$$

for some k . This sum is, in fact, a count of the total number of arc ends in the graph. Because the number of arc ends is twice the number of arcs, S is an even number. We will reorganize equation (1) to group together terms for odd nodes and terms for even nodes:

$$S = \underbrace{2 \cdot N(2) + 4 \cdot N(4) + \cdots + 2m \cdot N(2m)}_{\text{even nodes}} + \underbrace{1 \cdot N(1) + 3 \cdot N(3) + \cdots + (2n + 1) \cdot N(2n + 1)}_{\text{odd nodes}}$$

The sum of the terms representing even nodes is an even number. If we subtract it from both sides of the equation, we get a new equation—

$$S' = 1 \cdot N(1) + 3 \cdot N(3) + \cdots + (2n + 1) \cdot N(2n + 1) \quad (2)$$

—where S' (the difference of two even numbers) is an even number. Now if we rewrite equation (2) as

$$S' = \underbrace{1 + 1 + \cdots + 1}_{N(1) \text{ terms}} + \underbrace{3 + 3 + \cdots + 3}_{N(3) \text{ terms}} + \cdots + \underbrace{(2n + 1) + (2n + 1) + \cdots + (2n + 1)}_{N(2n + 1) \text{ terms}}$$

we see that there are N terms altogether in the sum (the number of odd nodes) and that each term is an odd number. For the sum of N odd numbers to be even, N must be even. (Can you prove this statement?) We have thus proved the following theorem.

● **THEOREM ON ODD NODES IN A GRAPH**
The number of odd nodes in any graph is even.

Now suppose a graph has an odd node n of degree $2k + 1$ and that an Euler path exists in the graph but does not start at n . Then for each arc we use to enter n , there is another unused arc for leaving n until we have used k pairs of arcs. The next time we enter n , there is no new arc on which to leave. Thus, if our path does not begin at n , it must end at n . The path either begins at n or it does not, and in the latter case it ends at n , so the path either begins or ends at this arbitrary odd node. Therefore, if there are more than two odd nodes in the graph, there can be no path. Thus, there are two possible cases where an Euler path may exist—on a graph with no odd nodes or on one with two odd nodes.

Consider the graph with no odd nodes. Pick any node m and begin an Euler path. Whenever you enter a different node, you will always have another arc on which to exit until you get back to m . If you have used up every arc of the graph, you are done. If not, there is some node m' of your path with unused arcs. Then construct an Euler path beginning and ending at m' , much as you did the previous section of path, using all new arcs. Attach this cycle as a side trip on the original path. If you have now used up every arc of the graph, you are done. If not, continue this process until every arc has been covered.

If there are exactly two odd nodes, an Euler path can be started beginning at one odd node and ending at the other. If the path has not covered all of the arcs, extra cycles can be patched in as in the previous case.

We now have a complete solution to the Euler path problem.

● **THEOREM ON EULER PATHS**
An Euler path exists in a connected graph if and only if there are either no odd nodes or two odd nodes. For the case of no odd nodes, the path can begin at any node and will end there; for the case of two odd nodes, the path must begin at one odd node and end at the other.

PRACTICE 8 Using the preceding theorem, work Practice 7 again. ■

PRACTICE 9 Is the Königsberg walk possible? ■

The theorem on Euler paths is actually an algorithm to determine whether an Euler path exists on an arbitrary connected graph. To make it look more like an algorithm, we'll rewrite it in pseudocode, but first we'll make a simplifying

assumption that the graph has no loops. If the graph G has loops, we can strip them off and consider the modified graph H . If H has an Euler path, then so does G —whenever we come to a node with a loop, we traverse the loop. If H has no Euler path, then neither does G .

In the accompanying algorithm (algorithm *EulerPath*), the input is a connected graph with no loops represented by an $n \times n$ adjacency matrix \mathbf{A} . The essence of the algorithm is to count the number of nodes adjacent to each node and to determine whether this is an odd or an even number. If there are too many odd numbers, an Euler path does not exist. The variable *total* keeps track of the number of odd nodes found in the graph. The degree of any particular node, *degree*, is found by adding the numbers in that node's row of the adjacency matrix. (This is why we exclude loops; a loop at node i adds only 1 to the adjacency matrix at position $[i, i]$, yet such a loop contributes 2 arc ends.) The function *odd* results in a value “true” if and only if the argument is an odd integer.

ALGORITHM *EULERPATH*

```

EulerPath ( $n \times n$  matrix  $\mathbf{A}$ )
//Determines whether an Euler path exists in a connected graph with
//no loops and adjacency matrix  $\mathbf{A}$ 
Local variables:
integer total //number of odd nodes so far found
integer degree //the degree of a node
integer i, j //array indices

    total = 0
    i = 1
    while total <= 2 and i <= n do
        degree = 0
        for j = 1 to n do
            degree = degree +  $\mathbf{A}[i, j]$  //find degree of node i (*)
        end for
        if odd(degree) then
            total = total + 1 //another odd degree node found
        end if
        i = i + 1
    end while

    if total > 2 then
        write (“No Euler path exists”)
    else
        write (“Euler path exists”)
    end if
end EulerPath

```

EXAMPLE 7

The adjacency matrix for the graph of Figure 7.6a follows.

$$\begin{bmatrix} 0 & 2 & 1 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 & 0 \end{bmatrix}$$

When the algorithm first enters the **while** loop, *total* is 0 and *i* is 1. Then *degree* is initialized to 0. Within the **for** loop, the values of row 1 of the adjacency matrix are added in turn to *degree*, resulting in a value for *degree* of 3. The *odd* function applied to *degree* returns the value “true,” so the value of *total* is increased from 0 to 1; one node of odd degree has been found. Then *i* is incremented to 2. Neither the bounds on *total* nor the bounds on the array size have been exceeded, so the **while** loop executes again, this time for row 2 of the array. Once again, *degree* is found to be odd, so the value of *total* is changed to 2. When the **while** loop is executed for row 3 of the array, the value of *degree* is even (4), so *total* does not change, and the **while** loop is executed again with *i* = 4. Row 4 again produces an odd value for *degree*, so *total* is raised to 3. This terminates the **while** loop. The bad news is written that there is no Euler path because the number of odd nodes exceeds 2. ●

PRACTICE 10

Write the adjacency matrix for the Königsberg walk problem and trace the execution of algorithm *EulerPath*. ■

Let us analyze algorithm *EulerPath*. The important operation done by the algorithm is an examination of the elements of the adjacency matrix, which occurs in the line marked at the end with an asterisk (*). In the worst case, the **while** loop in the algorithm is executed *n* times, once for each row. Within the **while** loop, the **for** loop, containing line (*), is executed *n* times, once for each column. *EulerPath* is therefore an $\Theta(n^2)$ algorithm in the worst case.

At the cost of some extra decision logic, we could modify the algorithm because we never have to examine the last row of the matrix. We know from the theorem on Euler paths that the total number of odd nodes is even. If the number of odd nodes after processing the next-to-last row is odd, then the last row must represent an odd node; if that number is even, then the last row must represent an even node. This modification results in $(n - 1)n$ elements to examine in the worst case, which is still $\Theta(n^2)$.

If we represented the graph *G* by an adjacency list rather than an adjacency matrix, then the corresponding version of the algorithm would have to count the length of the adjacency list for each node and keep track of how many are of odd length. There would be *n* adjacency lists to examine, just as there were *n* rows of the adjacency matrix to examine, but the length of each adjacency list might be shorter than *n*, the length of a row of the matrix. It is possible to reduce the order of magnitude below n^2 if the number of arcs in the graph is small, but the worst case is still $\Theta(n^2)$.

Hamiltonian Circuit Problem

Another famous mathematician, William Rowan Hamilton (1805–1865), posed a problem in graph theory that sounds very much like Euler’s. He asked how to tell whether a graph has a **Hamiltonian circuit**.

● **DEFINITION HAMILTONIAN CIRCUIT**
A Hamiltonian circuit in a graph is a cycle using every node of the graph.

(Recall that in a cycle, only the node that forms the beginning and the ending of the cycle is repeated.)

An Euler path in a graph requires that each and every arc of the graph be used once and only once, but nodes can be repeated. A Hamiltonian circuit requires that each and every node of the graph be visited once and only once (except for the start node, which is also the end node) but there can be unused arcs; no arc can be used more than once because that would involve revisiting a node.

PRACTICE 11 Do Hamiltonian circuits exist for the graphs of Figure 7.6? (Use trial and error to answer.) ■

Like the Euler path problem, the Hamiltonian circuit problem can be solved for a given graph by trial and error. The algorithm is as follows: Start from one node of the graph and try some path by choosing various arcs. If the path results in a repeated node, it is not a cycle, so throw it away and try a different path. If the path can be completed as a cycle, then see whether it visited every node; if not, throw it away and try a different path. Continue in this fashion until all possible paths have been tried or a Hamiltonian circuit has been found. This will involve some careful record keeping so that no path is tried more than once. The trial-and-error approach is theoretically possible—but it is practically impossible! In all but the smallest of graphs, there will simply be too many paths to try.

Euler found a simple, efficient algorithm to determine, for an arbitrary graph, if an Euler path exists. Although the Hamiltonian circuit problem sounds very similar to the Euler path problem, there is a basic difference. No efficient algorithm has ever been found to determine if a Hamiltonian circuit exists. In fact, there is some evidence (see Section 9.3) to suggest that no such algorithm will ever be found.

In certain types of graphs we can easily determine whether a Hamiltonian circuit exists. For example, an unconnected graph cannot have a Hamiltonian circuit because there would be no way to construct a path reaching all nodes. A complete graph with $n > 2$ has a Hamiltonian circuit because for any node on the path, there is always an arc to travel to any unused node and finally an arc to return to the starting point. Exercise 37 describes an additional condition that guarantees the existence of a Hamiltonian circuit. But in general—that is, for an arbitrary graph—we cannot readily make a determination about the existence of a Hamiltonian circuit.

Suppose we are dealing with a weighted graph. If a Hamiltonian circuit exists for the graph, can we find one with minimum weight? This is the traveling

salesman problem. Once again it can be solved using trial and error by tracing all possible paths and keeping track of the weights of those paths that are Hamiltonian circuits, but, again, this is not an efficient algorithm. (Incidentally, the traveling salesman problem for visiting all 48 capitals of the contiguous United States has been solved—a total of 10,628 miles is required!)

SECTION 7.2 REVIEW

TECHNIQUE

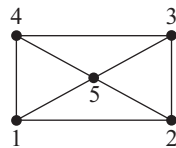
W Using algorithm *EulerPath*, determine whether an Euler path exists in a graph.

MAIN IDEAS

- There is a simple criterion for determining whether Euler paths exist in a graph but no such criterion for whether Hamiltonian circuits exist.
- An algorithm that is $\Theta(n^2)$ in the worst case can determine the existence of an Euler path in a connected graph with n nodes.

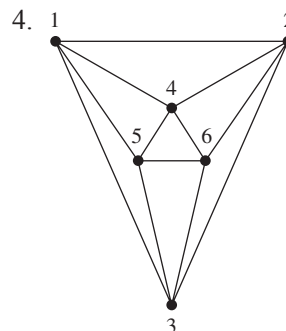
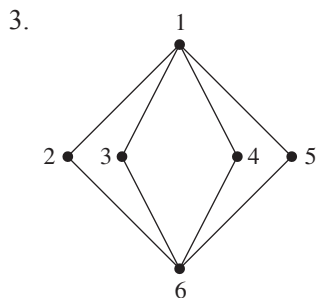
EXERCISES 7.2

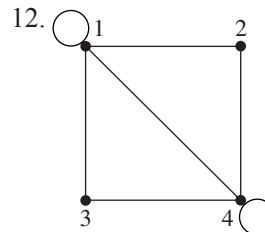
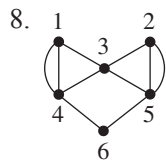
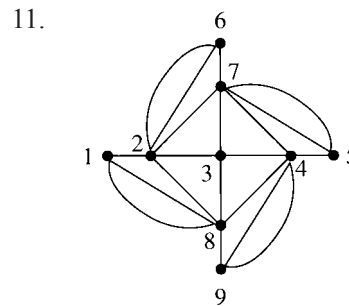
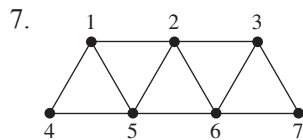
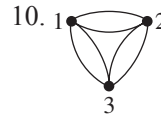
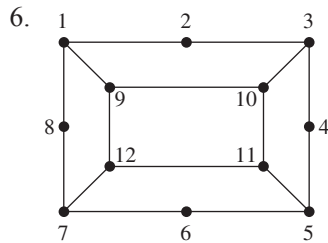
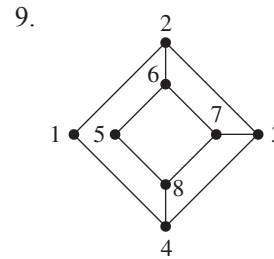
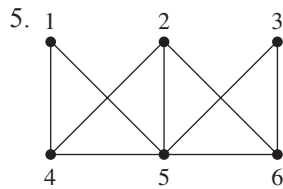
1. Rework Example 3 of Chapter 2 using the theorem on Euler paths. Here is the graph, where the nodes have been numbered.



2. a. Add a single arc to the graph of Exercise 1 so that there is an Euler path.
- b. List the nodes in such a path.

For Exercises 3–12, determine whether the given graph has an Euler path by using the theorem on Euler paths. If so, list the nodes in such a path.

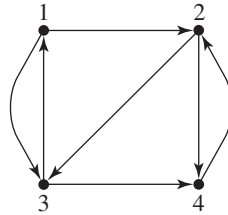




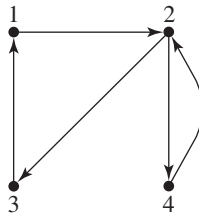
13. Draw the adjacency matrix for the graph of Exercise 3. In applying algorithm *EulerPath*, what is the value of *total* after the second pass through the **while** loop?
14. Draw the adjacency matrix for the graph of Exercise 5. In applying algorithm *EulerPath*, what is the value of *total* after the fourth pass through the **while** loop?
15. Draw the adjacency matrix for the graph of Exercise 7. In applying algorithm *EulerPath*, what is the value of *i* after the **while** loop is exited?
16. Draw the adjacency matrix for the graph of Exercise 9. In applying algorithm *EulerPath*, what is the value of *i* after the **while** loop is exited?

The definition of an Euler path extends to directed graphs. Instead of just the degree of a node as the total number of arc ends, we must now keep track of arcs coming into a node and arcs leaving a node. The total number of arc ends coming into a node is its *in-degree*; the total number of arc ends leaving a node is its *out-degree*. Exercises 17–20 talk about Euler paths in directed graphs.

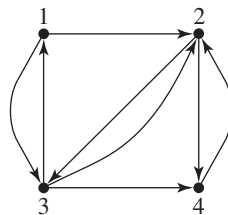
17. Describe two conditions on a connected directed graph, either of which would guarantee the existence of an Euler path.
18. Determine whether this graph has an Euler path. If so, list the nodes in such a path.



19. Determine whether this graph has an Euler path. If so, list the nodes in such a path.



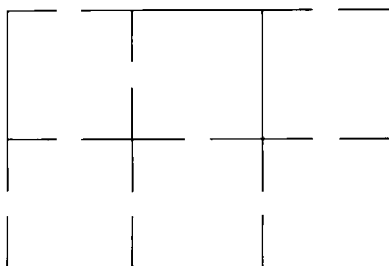
20. Determine whether this graph has an Euler path. If so, list the nodes in such a path.



For Exercises 21–28, decide by trial and error whether Hamiltonian circuits exist for the graphs of the given exercise. If so, list the nodes in such a cycle.

- | | | | |
|----------------|----------------|----------------|-----------------|
| 21. Exercise 3 | 23. Exercise 5 | 25. Exercise 7 | 27. Exercise 9 |
| 22. Exercise 4 | 24. Exercise 6 | 26. Exercise 8 | 28. Exercise 11 |
29. Prove that any graph with a Hamiltonian circuit is connected.
30. Find an example of an unconnected graph that has an Euler path. (*Hint*: Because this seems intuitively contradictory, you should look for a trivial case.)

31. Consider a simple, complete graph with n nodes. Testing for a Hamiltonian circuit by trial and error could be done by selecting a fixed starting node and then generating all possible paths from that node of length n .
- How many paths of length n are there if repetition of arcs and nodes is allowed?
 - How many paths of length n are there if repetition of arcs and nodes is allowed but an arc may not be used twice in succession?
 - How many paths of length n are there if nodes and arcs cannot be repeated except for the starting node? (These are the Hamiltonian circuits.)
 - To solve the traveling salesman problem in a weighted graph, assume a fixed starting point at node 1 and generate all possible Hamiltonian circuits of length n to find one with minimum weight. If it takes 0.000001 seconds to generate a single Hamiltonian circuit, how long will this process take in a simple, complete graph with 15 nodes?
32. Is it possible to walk in and out of each room in the house shown in the following figure so that each door of the house is used exactly once? Why or why not?



33. Recall that K_n denotes the simple, complete graph of order n .
- For what values of n does an Euler path exist in K_n ?
 - For what values of n does a Hamiltonian circuit exist in K_n ?
34. Recall that $K_{m,n}$ denotes a bipartite, complete graph with $m + n$ nodes.
- For what values of m and n does an Euler path exist in $K_{m,n}$?
 - For what values of m and n does a Hamiltonian circuit exist in $K_{m,n}$?
35. Prove that a Hamiltonian circuit always exists in a connected graph where every node has degree 2.
36. Consider a connected graph with $2n$ odd vertices, $n \geq 2$. By the theorem on Euler paths, an Euler path does not exist for this graph.
- What is the minimum number of disjoint Euler paths, each traveling some of the arcs of the graph, necessary to travel each arc exactly once?
 - Show that the minimum number is sufficient.
37. Ore's theorem (Oystein Ore, 1960) states that a Hamiltonian circuit exists in any graph G with the following properties:
- G is a simple graph with n nodes, $n \geq 3$.
 - For any two nonadjacent nodes x and y , $\text{degree}(x) + \text{degree}(y) \geq n$.
- Ore's Theorem is proved by contradiction in the following steps.
- Assume that a graph G with properties 1 and 2 above does not have a Hamiltonian circuit. Beginning with G , add new edges to produce a simple graph H that does not have a Hamiltonian circuit but would have such a circuit with the addition of any single new arc. Describe a process for creating H .
 - Prove that H has a Hamiltonian path, that is, a path that visits each node exactly once.
 - Denote the nodes on the Hamiltonian path by $p = x_1, x_2, x_3, \dots, x_{n-1}, x_n = q$. Prove that for any node $x_i, 2 \leq i \leq n - 1$, if an arc exists in H between x_i and p , then no arc exists in H between x_{i-1} and q .

- d. Using the result from part (c), prove that in graph H , $\text{degree}(p) + \text{degree}(q) < n$.
- e. Prove that in graph G , $\text{degree}(p) + \text{degree}(q) < n$.
- f. Conclude from part (e) that G has a Hamiltonian circuit.
38. Ore's theorem (Exercise 37) gives a sufficient condition for a Hamiltonian circuit to exist, but it is not a necessary condition. Find a simple graph G with n nodes, $n \geq 3$, that has a Hamiltonian circuit but for which condition (2) does not hold.

SECTION 7.3 SHORTEST PATH AND MINIMAL SPANNING TREE

Shortest-Path Problem

Assume that we have a simple, weighted, connected graph, where the weights are positive. Then a path exists between any two nodes x and y . Indeed, there may be many such paths. The question is, How do we find a path with minimum weight? Because weight often represents distance, this problem has come to be known as the “shortest-path” problem. It is an important problem to solve for a computer or communications network, where information at one node must be routed to another node in the most efficient way possible, or for a transportation network, where products in one city must be shipped to another.

The traveling salesman problem is a minimum-weight path problem with such severe restrictions on the nature of the path that such a path may not exist at all. In the shortest-path problem, we put no restrictions (other than minimum weight) on the nature of the path, and because the graph is connected, we know that such a path exists. For this reason we may hope for an efficient algorithm to solve the problem, even though no such algorithm is known for the traveling salesman problem. Indeed such an algorithm does exist; it was published in 1959 by Edsger W. Dijkstra, a prominent computer scientist of the twentieth century.

The shortest-path algorithm known as *Dijkstra's algorithm* works as follows. We want to find the minimum-distance path from a given node x to a given node y . We build a set (we'll call it IN) that initially contains only x but grows as the algorithm proceeds. At any given time IN contains every node whose shortest path from x , using only nodes in IN, has so far been determined. For every node z outside IN, we keep track of the shortest distance $d[z]$ from x to that node, using a path whose only non-IN node is z . We also keep track of the node adjacent to z on this path, $s[z]$.

How do we let IN grow; that is, which node should be moved into IN next? We pick the non-IN node with the smallest distance d . Once we add that node, call it p , to IN, then we have to recompute d for all the remaining non-IN nodes, because there may be a shorter path from x going through p than there was before p belonged to IN. So we compare the current distance of z from x , $d[z]$, with the distance of p from x , $d[p]$, plus the distance from p to z , $A[p, z]$ where A is the adjacency matrix. If there is a shorter path, we must also update $s[z]$ so that p is now shown to be the node adjacent to z on the current shortest path, that is, $s[z]$ is the node just before z on this path from x . As soon as y is moved into IN, IN stops growing. The current value of $d[y]$ is the distance for the shortest path, and its nodes are found by looking at y , $s[y]$, $s[s[y]]$, and so forth, until we have traced the path back to x .

A pseudocode form of the algorithm is given in the accompanying box. The input is the adjacency matrix for a simple, connected graph G with positive weights

and nodes x and y ; the algorithm writes out the shortest path between x and y and the distance for that path. Here shortest path means minimum-weight path. We actually assume a modified adjacency matrix \mathbf{A} , where $\mathbf{A}[i, j]$ is the weight of the arc between i and j if one exists and $\mathbf{A}[i, j]$ has the value ∞ if no arc exists (here the symbol ∞ denotes a number larger than any weight in the graph).

ALGORITHM *DIJKSTRA'S ALGORITHM*

```

Dijkstra ( $n \times n$  matrix  $\mathbf{A}$ ; nodes  $x, y$ )
//Computes the shortest path between a source node  $x$  and a destination node  $y$ 
//in a simple, connected graph with positive weights.  $\mathbf{A}$  is a modified adjacency
//matrix. Writes out nodes in the shortest path from  $x$  to  $y$ , and the
//distance for that path.
Local variables:
set of nodes  $IN$            //set of nodes whose shortest path from  $x$  is known
nodes  $z, p$                 //temporary nodes
array of integers  $d$         //for each node, the distance from  $x$  using nodes in  $IN$ 
array of nodes  $s$           //for each node, the previous node in the shortest path
integer  $OldDistance$       // distance to compare against

//initialize set  $IN$  and arrays  $d$  and  $s$ 
 $IN = \{x\}$ 
 $d[x] = 0$ 
for all nodes  $z$  not in  $IN$  do
     $d[z] = \mathbf{A}[x, z]$ 
     $s[z] = x$ 
end for

//process nodes into  $IN$ 
while  $y$  not in  $IN$  do
    //add minimum-distance node not in  $IN$ 
     $p =$  node  $z$  not in  $IN$  with minimum  $d[z]$ 
     $IN = IN \cup \{p\}$ 

    //recompute  $d$  for non  $IN$  nodes, adjust  $s$  if necessary
    for all nodes  $z$  not in  $IN$  do
         $OldDistance = d[z]$ 
         $d[z] = \min(d[z], d[p] + \mathbf{A}[p, z])$ 
        if  $d[z] \neq OldDistance$  then
             $s[z] = p$ 
        end if
    end for
end while

```

```

//write out path nodes
write ("In reverse order, the path is")
write (y)
z = y
repeat
  write (s[z])
  z = s[z]
until z = x

// write out path distance
write ("The path distance is", d[y])
end Dijkstra

```

EXAMPLE 8

Consider the graph in Figure 7.7 and the corresponding modified adjacency matrix shown in Figure 7.8.

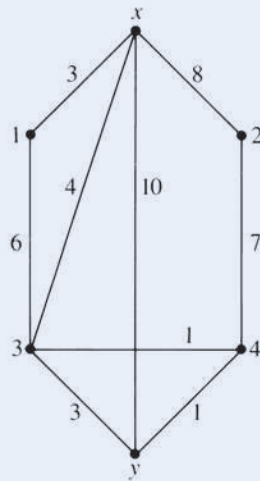


Figure 7.7

	x	1	2	3	4	y
x	∞	3	8	4	∞	10
1	3	∞	∞	6	∞	∞
2	8	∞	∞	∞	7	∞
3	4	6	∞	∞	1	3
4	∞	∞	7	1	∞	1
y	10	∞	∞	3	1	∞

Figure 7.8

Let's trace Dijkstra's algorithm on this graph. At the end of the initialization phase, IN contains only x , d contains all the direct distances (arc weights) from x to other nodes, and x is the immediate predecessor of all nodes except x . (Because of the ∞ in position $A[x, 4]$, there is no arc from x to 4, so $s[4]$ is meaningless here but it simplifies the initialization.)

$$IN = \{x\}$$

	x	1	2	3	4	y
d	0	3	8	4	∞	10
s	-	x	x	x	x	x

In Figure 7.9, circled nodes are those in set IN , heavy lines show the current shortest paths, and the d value for each node is written along with the node label. Figure 7.9a is the picture after initialization.

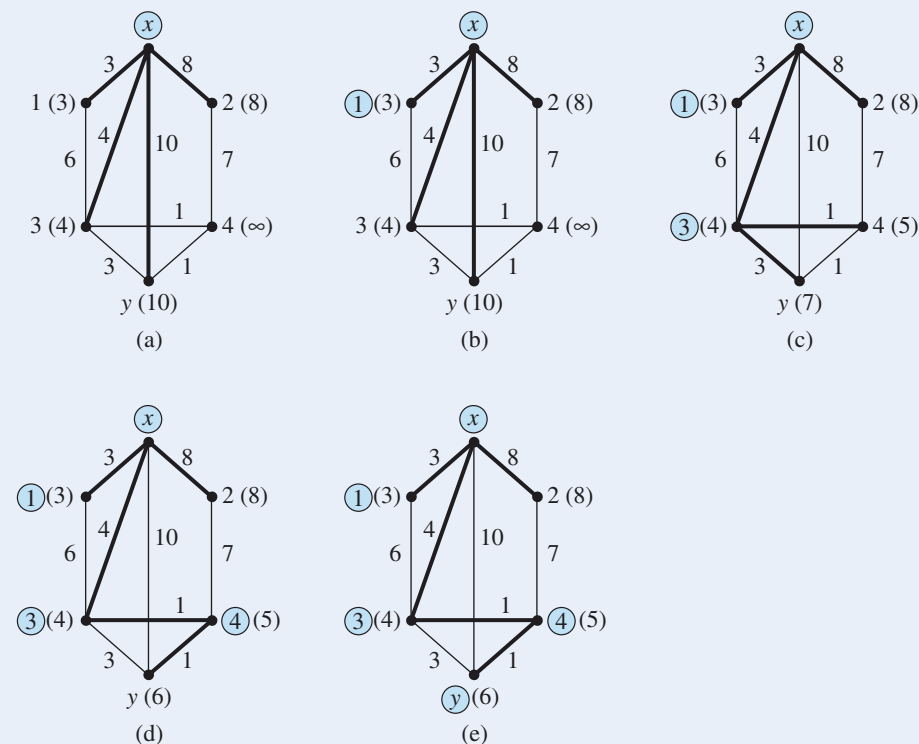


Figure 7.9

We now enter the **while** loop and search through the d values for the node of minimum distance that is not in IN ; this turns out to be node 1, with $d[1] = 3$. We throw node 1 into IN , and in the **for** loop we recompute all the d values for the remaining nodes, 2, 3, 4, and y .

$$p = 1$$

$$IN = \{x, 1\}$$

$$d[2] = \min(8, 3 + \mathbf{A}[1, 2]) = \min(8, \infty) = 8$$

$$d[3] = \min(4, 3 + \mathbf{A}[1, 3]) = \min(4, 9) = 4$$

$$d[4] = \min(\infty, 3 + \mathbf{A}[1, 4]) = \min(\infty, \infty) = \infty$$

$$d[y] = \min(10, 3 + \mathbf{A}[1, y]) = \min(10, \infty) = 10$$

There were no changes in the d values, so there were no changes in the s values (there were no shorter paths from x by going through node 1 than by going directly from x). Figure 7.9b shows that 1 is now in IN .

The second pass through the **while** loop produces the following:

$$p = 3 \text{ (3 has the smallest } d \text{ value, namely 4, of 2, 3, 4, or } y)$$

$$IN = \{x, 1, 3\}$$

REMINDER

Distances in Dijkstra's algorithm are always recomputed relative to the node most recently added to IN .

$$d[2] = \min(8, 4 + \mathbf{A}[3, 2]) = \min(8, 4 + \infty) = 8$$

$$d[4] = \min(\infty, 4 + \mathbf{A}[3, 4]) = \min(\infty, 4 + 1) = 5 \text{ (a change, so update } s[4] \text{ to 3)}$$

$$d[y] = \min(10, 4 + \mathbf{A}[3, y]) = \min(10, 4 + 3) = 7 \text{ (a change, so update } s[y] \text{ to 3)}$$

	x	1	2	3	4	y
d	0	3	8	4	5	7
s	-	x	x	x	3	3

Shorter paths from x to the two nodes 4 and y were found by going through 3, as reflected in Figure 7.9c.

On the next pass,

$$p = 4 \text{ (} d \text{ value} = 5)$$

$$IN = \{x, 1, 3, 4\}$$

$$d[2] = \min(8, 5 + 7) = 8$$

$$d[y] = \min(7, 5 + 1) = 6 \text{ (a change, update } s[y])$$

	x	1	2	3	4	y
d	0	3	8	4	5	6
s	-	x	x	x	3	4

See Figure 7.9d.

Processing the **while** loop again, we get

$$p = y$$

$$IN = \{x, 1, 3, 4, y\}$$

$$d[2] = \min(8, 6 + \infty) = 8$$

	x	1	2	3	4	y
d	0	3	8	4	5	6
s	-	x	x	x	3	4

See Figure 7.9e.

Now that y is part of IN , the **while** loop terminates. The path goes through y , $s[y] = 4$, $s[4] = 3$, and $s[3] = x$. Thus the path uses nodes x , 3, 4, and y . (The algorithm gives us these nodes in reverse order.) The distance for the path is $d[y] = 6$. By looking at the graph in Figure 7.7 and checking all the possibilities, we can see that this is the shortest path from x to y . ●

Dijkstra's algorithm terminates when y is put into IN , even though there may be other nodes in the graph not yet in IN (such as node 2 in Example 8). How do we know that a still shorter path cannot be found through one of these excluded nodes? If we continue processing until all nodes have been included in IN , the d values then represent the shortest path from x to any node, using all the values in IN , that is, the shortest path using any nodes of the graph. But new nodes are brought into IN in order of increasing d values. A node z that is brought into IN later than y must have as its shortest path from x one whose distance is at least as

great as the d value of y when y was brought into IN . Therefore there cannot be a shorter path from x to y via z because there is not even a shorter path just between x and z .

PRACTICE 12

Trace Dijkstra's algorithm on the graph shown in Figure 7.10. Show the values for p and IN and the d values and s values for each pass through the **while** loop. Write out the nodes of the shortest path and the distance of the path.

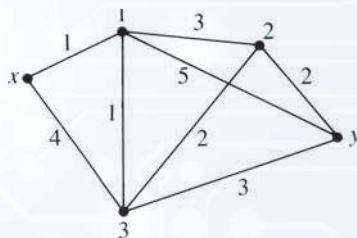


Figure 7.10

When looking for the next node to bring into IN in Dijkstra's algorithm, more than one node p may have a minimum d value, in which case p can be selected arbitrarily. There may also be more than one shortest path between x and y in a graph.

Dijkstra's algorithm also works for directed graphs if the adjacency matrix is in the appropriate form. And it works for unconnected graphs; if x and y are not in the same component, then $d[y]$ will remain ∞ throughout. After y has been brought into IN , the algorithm will terminate, and this value of ∞ for $d[y]$ will indicate that no path exists between x and y .

We may think of Dijkstra's algorithm as being a “nearsighted” algorithm. It cannot see the entire graph at once to pick out overall shortest paths; it only picks out shortest paths relative to the set IN at each step. Such an algorithm is called a **greedy algorithm**—it does what seems best based on its limited immediate knowledge. In this case, what seems best at the time turns out to be best overall.

How efficient is Dijkstra's algorithm? Most of the work seems to take place within the **for** loop that modifies the d and s arrays. Here the algorithm checks all n nodes to determine which nodes z are not in IN and recomputes $d[z]$ for those nodes, possibly also changing $s[z]$. The necessary quantities $d[z]$, $d[p]$, and $A[p, z]$ for a given z are directly available. Therefore the **for** loop requires $\Theta(n)$ operations. In addition, determining the node p to add to IN can also be done in $\Theta(n)$ operations by checking all n nodes. With the additional small amount of work to add p to IN , each execution of the **while** loop takes $\Theta(n)$ operations. In the worst case, y is the last node brought into IN , and the **while** loop will be executed $n - 1$ times. Therefore the total number of operations involved in the **while** loop is $\Theta(n(n - 1)) = \Theta(n^2)$. Initialization and writing the output together take $\Theta(n)$ operations, so the algorithm requires $\Theta(n + n^2) = \Theta(n^2)$ operations in the worst case.

What if we keep IN (or rather the complement of IN) as some sort of linked list, so that all the nodes of the graph do not have to be examined to see which are not in IN ? Surely this would make the algorithm more efficient. Note that the number of nodes not in IN is initially $n - 1$, and that number decreases by 1 for each pass through the **while** loop. Within the **while** loop the algorithm thus has to

perform on the order of $n - 1$ operations on the first pass, then $n - 2$, then $n - 3$, and so on. But, as proof by induction will show,

$$(n - 1) + (n - 2) + \cdots + 1 = (n - 1)n/2 = \Theta(n^2)$$

Thus the worst-case situation still requires $\Theta(n^2)$ operations.

Minimal Spanning Tree Problem

A problem encountered in designing networks is how to connect all the nodes efficiently, where nodes can be computers, telephones, warehouses, pumping stations, and so on. A minimal spanning tree may provide an economical solution, one that requires the least cable, pipeline, or whatever the connecting medium is. For reliability, however, the minimal spanning tree usually would be supplemented with additional arcs so that if one connection were broken for some reason, an alternative route could be found.

DEFINITION SPANNING TREE

A **spanning tree** for a connected graph is a nonrooted tree whose set of nodes coincides with the set of nodes for the graph and whose arcs are (some of) the arcs of the graph.

A spanning tree thus connects all the nodes of a graph with no excess arcs (no cycles). There are algorithms for constructing a **minimal spanning tree**, a spanning tree with minimal weight, for a given simple, weighted, connected graph. One of these algorithms, called *Prim's algorithm*, proceeds very much like Dijkstra's algorithm. There is a set IN , which initially contains one arbitrary node. For every node z not in IN , we keep track of the shortest distance $d[z]$ between z and any node in IN . We successively add nodes to IN , where the next node added is one that is not in IN and whose distance $d[z]$ is minimal. The arc having this minimal distance is then made part of the spanning tree. Because there may be ties between minimal distances, the minimal spanning tree of a graph may not be unique. The algorithm terminates when all nodes of the graph are in IN .

The key difference in the implementation of the two algorithms comes in the computations of new distances for the nodes not yet in IN . In Dijkstra's algorithm, if p is the node that has just been added to IN , distances for non- IN nodes are recalculated by

$$d[z] = \min(d[z], d[p] + \mathbf{A}[p, z])$$

that is, by comparing the current distance of z from x with the distance of p from x plus the distance of z from p . In Prim's algorithm, if p is the node that has just been added to IN , distances for non- IN nodes are recalculated by

$$d[z] = \min(d[z], \mathbf{A}[p, z])$$

that is, by comparing the current distance of z from IN with the distance of z from p .

We won't write out the algorithm (which, like Dijkstra's algorithm, requires $\Theta(n^2)$ operations in the worst case and is a greedy algorithm); we will simply illustrate it with an example.

EXAMPLE 9

We will find a minimal spanning tree for the graph of Figure 7.7. We let node 1 be the arbitrary initial node in IN . Next we consider all the nodes adjacent to any node in IN , that is, all nodes adjacent to 1, and select the closest one, which is node x . Now $IN = \{1, x\}$, and the arc between 1 and x is part of the minimal spanning tree. Next we consider all nodes not in IN that are adjacent to either 1 or x . The closest such node is 3, which is 4 units away from x . The arc between 3 and x is part of the minimal spanning tree. For $IN = \{1, x, 3\}$, the next closest node is node 4, 1 unit away from 3. The remaining nodes are added in the order y and then 2. Figure 7.11 shows the minimal spanning tree.

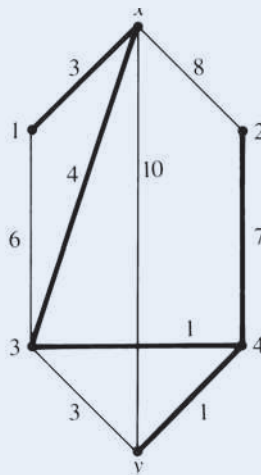


Figure 7.11

PRACTICE 13

Find a minimal spanning tree for the graph of Figure 7.10.

EXAMPLE 10

Seismic sensing instruments are to be distributed at sites along a volcanic rift zone, as shown in Figure 7.12a, where the distances in meters between sites are given. (Distances between some sites are not shown because of natural hazards that would prevent a direct connection.) The most economical way to wire the devices so that they are all connected is to create a minimal spanning tree for the graph, as shown in Figure 7.12b. The total length of wire involved is 975 meters.

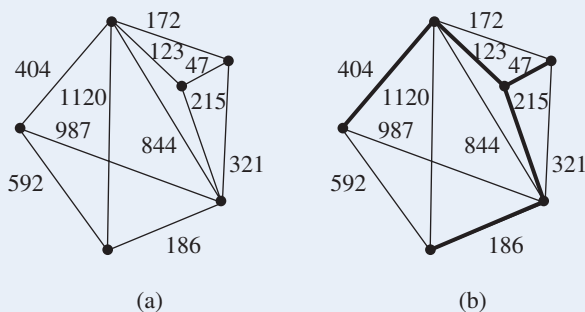


Figure 7.12

Pathfinding

Pathfinding—as the name suggests—seeks to find a path from point x to point y , more specifically, the shortest path. This is a problem encountered in video games, when entities must move to a target point while avoiding obstacles. On a much larger scale, Google Maps and other Web mapping sites allow us to enter a source x and a destination y and, in a very short time, receive driving directions to get from x to y .

Well, don't we already know how to do that? Dijkstra's algorithm is the classic shortest-path algorithm, but it's an $\Theta(n^2)$ algorithm. In the context of a real-time video game, quick response is essential. For Google maps, an $\Theta(n^2)$ algorithm might not be a problem for a graph with 50, 500, or 2000 nodes, but scale up to the hundreds of thousands of cities and towns, and you'll want to find a way to speed up the search. This is an important problem, and much research has been directed toward improving shortest-path performance.

One generalization of Dijkstra's algorithm, called the A^* algorithm, is used by many video game designers. The exact techniques used by Google and other sites aren't publicly revealed, but the A^* algorithm may well be part of their arsenal. The A^* algorithm generalizes Dijkstra's algorithm by adding ideas borrowed from artificial intelligence. In fact, A^* , strictly speaking, isn't even an algorithm because there's a big hole where one must plug in a "heuristic" (read an "educated guess"). The heuristic function used varies with the specifics of the application, and it's the hard part of using A^* .

Here's an outline of how A^* works. The general problem is that there is a graph with a source node x and a destination node y (for simplicity, assume that it's a connected graph so that a path from x to y exists), and the arcs of the graph are weighted with known distances between nodes. At any point in time, a "closed set" represents nodes that have been examined and need not be considered further. There is also an "open set" that represents nodes available for evaluation as the next node along the path. Initially, only node x is in the open set. Evaluation of a node n consists of three values that are maintained for each node in the open set:

G = the distance to get from x to n along the path being constructed

H = the result of the heuristic function that "guesses" the distance to get from n to y . This is a guess because the exact path from n to y is not yet known.

$$F = G + H$$

Pseudocode:

Put node x into the open set

Repeat the following process until y gets moved to the closed set:

Select the node p from the open set with the lowest F value (with some tie-breaking rule)

Move p to the closed set

For each node z that is adjacent to p and not in the closed set (you don't want to go back to a previous node on the path), do the following:

If z is not in the open set, move z to the open set, compute its G , H , and F values (the G value is just the G value for p + the weight of the p - z arc), and set p as z 's parent node.

If z is in the open set, compute a new G value for z by going through node p ; if this value is lower than z 's current G value, then recompute z 's F value (which will also be lower than before) and set p as z 's parent node.

Once the target node y is in the closed set, walk back through the parents to the start node x . Reversing this walk gives the shortest path from x to y .

Note the similarities to Dijkstra's algorithm (DA). There is a closed set (like the IN set of DA) that begins with the start node and eventually includes the end node. There is a next node p that gets moved into the closed set (moved into IN in DA). Once p is moved into the closed set, G values (distances from x in DA) are recomputed to see whether going through node p is an improvement. The algorithm terminates when the target node y is moved into the closed set (moved into IN in DA). The path is found by walking backward from y to x through a parent list (the s array in DA).

The difference between A^* and Dijkstra's algorithm is that in A^* the next node p is chosen based on the lowest F value, whereas in Dijkstra's algorithm p is chosen based on the shortest distance from the source node (which is the same as the G value). So A^* uses

F to make its choice and Dijkstra's algorithm uses G , but $F = G + H$. Dijkstra's algorithm is therefore A^* with a heuristic function that always returns 0. By using a non-zero heuristic H value, A^* makes smarter choices (and therefore closes in on the shortest path faster) provided, of course, that the heuristic function is a good guess. If the heuristic function consistently overestimates the distance to the target, then the path computed will not be the shortest path. Another way to look at the difference between A^* and Dijkstra's

algorithm is to remember that Dijkstra's algorithm is a greedy algorithm—it makes its decisions based on the local knowledge of how close adjacent nodes are to p . A^* uses global knowledge (or assumed global knowledge) about not only how close adjacent nodes are to p but also how far they are from the target node.

<http://www.policyalmanac.org/games/aStarTutorial.htm>

<http://www.heyes-jones.com/astar.html>

SECTION 7.3 REVIEW

TECHNIQUES

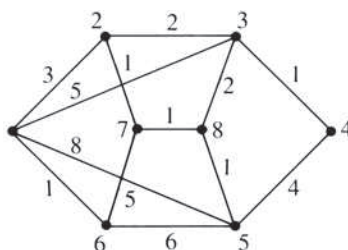
- W Find a shortest path from x to y in a graph (using Dijkstra's algorithm).
- W Find a minimal spanning tree for a graph (using Prim's algorithm).

MAIN IDEA

- Algorithms that are $\Theta(n^2)$ in the worst case can find a shortest path between two nodes or a minimal spanning tree in a simple, positively weighted, connected graph with n nodes.

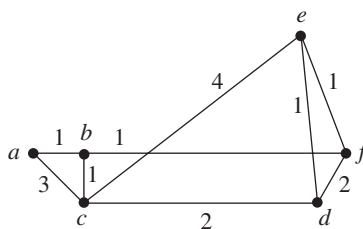
EXERCISES 7.3

For Exercises 1–4, use the graph that follows. Apply Dijkstra's algorithm for the pairs of nodes given; show the values for p and IN and the d values and s values for each pass through the **while** loop. Write out the nodes in the shortest path and the distance of the path.



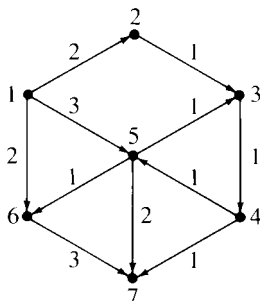
1. From 2 to 5
2. From 3 to 6
3. From 1 to 5
4. From 4 to 7

For Exercises 5 and 6, use the graph that follows. Apply Dijkstra's algorithm for the pairs of nodes given; show the values for p and IN and the d values and s values for each pass through the **while** loop. Write out the nodes in the shortest path and the distance of the path.



5. From a to e
6. From d to a

For Exercises 7 and 8, use the directed graph that follows. Apply Dijkstra's algorithm to the nodes given; show the values for p and IN and the d values and s values for each pass through the **while** loop. Write out the nodes in the shortest path and the distance of the path.



7. From 1 to 7
8. From 3 to 1
9. a. Modify Dijkstra's algorithm so that it finds the shortest paths from x to all other nodes in the graph.
b. Does this change the worst-case order of magnitude of the algorithm?
10. Give an example to show that Dijkstra's algorithm does not work when negative weights are allowed.

Another algorithm for finding shortest paths from a single source node to all other nodes in the graph is the *Bellman–Ford algorithm*. In contrast to Dijkstra's algorithm, which keeps a set of nodes whose shortest path (minimum-weight path) of whatever length (that is, number of hops) has been determined, the Bellman–Ford algorithm performs a series of computations that seeks to find successively smaller-weight paths of length 1, then of length 2, then of length 3, and so on, up to a maximum of length $n - 1$ (if a path exists at all, then there is a path of length no greater than $n - 1$). A pseudocode description of the Bellman–Ford algorithm is given in the accompanying box; when using this algorithm, the adjacency matrix \mathbf{A} must have $\mathbf{A}[i, i] = 0$ for all i .

ALGORITHM *BELLMAN–FORD ALGORITHM*

```

Bellman–Ford( $n \times n$  matrix  $\mathbf{A}$ ; node  $x$ ; array of integers  $d$ ; array of nodes  $s$ )
//Computes the shortest path between a source node  $x$  and all other nodes in a simple,
//weighted, connected graph.  $\mathbf{A}$  is a modified adjacency matrix with  $\mathbf{A}[i, i] = 0$ .
//When procedure terminates, the nodes in the shortest path from  $x$  to a node  $y$ 
//are  $y, s[y], s[s[y]], \dots, x$ ; the distance for that path is  $d[y]$ .
Local variables:
nodes  $z, p$  //temporary nodes
array of integers  $t$  //temporary distance array created at each iteration

//initialize arrays  $d$  and  $s$ ; this establishes the shortest 1-length paths from  $x$ 
 $d[x] = 0$ 
for all nodes  $z$  not equal to  $x$  do
     $d[z] = \mathbf{A}[x, z]$ 
     $s[z] = x$ 
end for

//find shortest paths of length 2, 3, etc.
for  $i = 2$  to  $n - 1$  do
     $t = d$  //copy current array  $d$  into array  $t$ 

```

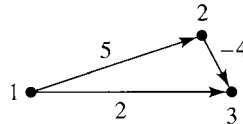
```

//modify  $t$  to hold shortest paths of length  $i$ 
for all nodes  $z$  not equal to  $x$  do
  //find the shortest path with one more link
   $p =$  node in  $G$  for which  $(d[p] + \mathbf{A}[p, z])$  is minimum
   $t[z] = d[p] + \mathbf{A}[p, z]$ 
  if  $p \neq z$  then
     $s[z] = p$ 
  end if
end for
 $d = t$  ; //copy array  $t$  back into  $d$ 
end for
end Bellman–Ford

```

For Exercises 11–14 use the Bellman–Ford algorithm to find the shortest path from the source node to any other node. Show the successive d values and s values.

11. Graph for Exercises 1–4, source node = 2 (compare your answer to Exercise 1)
12. Graph for Exercises 1–4, source node = 1 (compare your answer to Exercise 3)
13. Graph for Exercises 7–8, source node = 1 (compare your answer to Exercise 7)
14. a. Accompanying graph, source node = 1 (compare your answer to Exercise 10)
 - b. What does this say about the Bellman–Ford algorithm as opposed to Dijkstra’s algorithm?



To compute the distance for the shortest path between any two nodes in a graph, Dijkstra’s algorithm could be used repeatedly, with each node in turn as the source node. A different algorithm, *Floyd’s algorithm*, can also be used to solve this “all pairs” shortest-path problem, but while Floyd’s algorithm produces the weight of all shortest paths, it does not calculate what the shortest paths actually are, that is, what nodes are on a given shortest path. Floyd’s algorithm is very similar to Warshall’s algorithm. A description follows, where \mathbf{A} is the adjacency matrix of the graph with $\mathbf{A}[i, i] = 0$ for all i .

ALGORITHM FLOYD’S ALGORITHM

```

Floyd ( $n \times n$  matrix  $\mathbf{A}$ )
//Computes the shortest path between any two nodes in a simple, weighted,
//connected graph;  $\mathbf{A}$  is a modified adjacency matrix with  $\mathbf{A}[i, i] = 0$ .
//Upon termination,  $\mathbf{A}$  will contain all the shortest-path distances
for  $k = 1$  to  $n$  do
  for  $i = 1$  to  $n$  do
    for  $j = 1$  to  $n$  do
      if  $\mathbf{A}[i, k] + \mathbf{A}[k, j] < \mathbf{A}[i, j]$  then
         $\mathbf{A}[i, j] = \mathbf{A}[i, k] + \mathbf{A}[k, j]$ 
      end if
    end for
  end for
end for
end Floyd

```

For Exercises 15 and 16, use Floyd's algorithm to find the distances for all the shortest paths. Show the successive values of the **A** matrix for each pass through the outer loop.

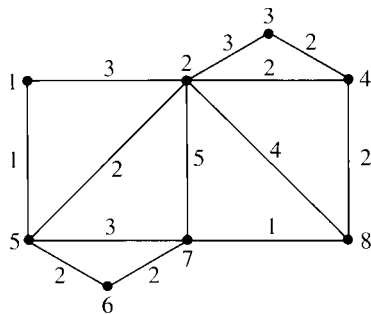
15. Figure 7.10

16. Graph for Exercises 1–4

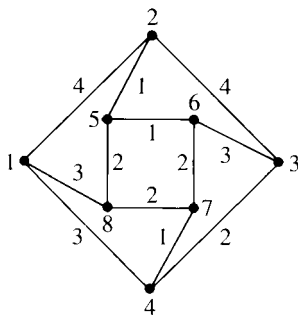
For Exercises 17–20, use Prim's algorithm to find a minimal spanning tree for the graph in the specified figure.

17. Graph for Exercises 1–4

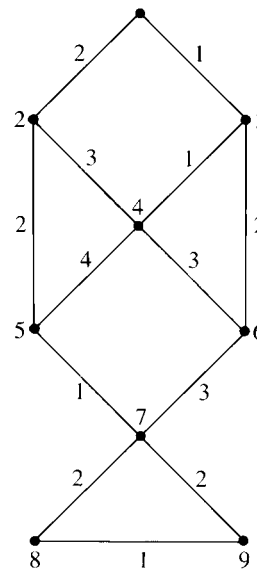
18.



19.



20.



Kruskal's algorithm is another algorithm for finding a minimal spanning tree in a connected graph. Whereas Prim's algorithm "grows" the tree from an arbitrary starting point by attaching adjacent short arcs, Kruskal's algorithm adds arcs in order by increasing distance wherever they may be in the graph. Ties are resolved arbitrarily. The only restriction is that an arc is not added if adding it would create a cycle. The algorithm terminates when all nodes have been incorporated into a connected structure. A (very informal) pseudocode description follows:

ALGORITHM *KRUSKAL'S ALGORITHM*

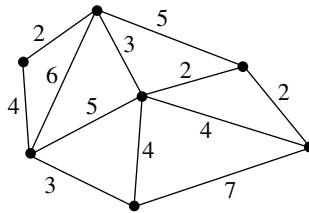
```

Kruskal ( $n \times n$  matrix A; collection of arcs  $T$ )
//Finds a minimal spanning tree;  $T$  is initially empty;
//at termination,  $T$  = minimal spanning tree
order arcs in  $G$  by increasing distance
repeat
  if next arc in order does not complete a cycle then
    add that arc into  $T$ 
  end if
until  $T$  is connected and contains all nodes of  $G$ 
end Kruskal

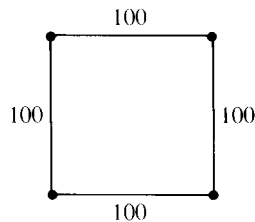
```

For Exercises 21–24 use Kruskal’s algorithm to find the minimal spanning tree.

21. Graph for Exercises 1–4
 22. Graph for Exercise 18
 23. Graph for Exercise 19
 24. Graph for Exercise 20
 25. Give an example to show that adding the node closest to IN at each step, as is done in Prim’s minimal spanning tree algorithm, will not guarantee a shortest path.
 26. Let a be the arc of lowest weight in a weighted graph. Show that a must be an arc in any minimal spanning tree.
 27. A city plans to lay out bike paths connecting various city parks. A map with the distances between the parks is shown in the figure. (Some direct connections would have to cross major highways, so these distances are not shown in the map.) Find which paths to pave so that all parks are connected but the cost is minimal.



28. Assume that arc weights represent distance. Then adding new nodes and arcs to a graph may result in a spanning tree for the new graph that has less weight than a spanning tree for the original graph. (The new spanning tree could represent a minimal-cost network for communications between a group of cities obtained by adding a switch in a location outside any of the cities.)
 a. Find a spanning tree of minimum weight for the following labeled graph. What is its weight?



- b. Put a node in the center of the square. Add new arcs from the center to the corners. Find a spanning tree for the new graph, and compute its (approximate) weight.
 29. At the beginning of this chapter, you received the following assignment:

You are the network administrator for a wide-area backbone network that serves your company’s many offices across the country. Messages travel through the network by being routed from point to point until they reach their destination. Each node in the network therefore acts as a switching station to forward messages to other nodes according to a routing table maintained at each node. Some connections in the network carry heavy traffic, while others are less used. Traffic may vary with the time of day; in addition, new nodes occasionally come on line and existing nodes may go off line. Therefore you must periodically provide each node with updated information so that it can forward messages along the most efficient (that is, the least heavily traveled) route.

How can you compute the routing table for each node?

You realize that you can represent the network as a weighted graph, where the arcs are the connections between nodes and the weights of the arcs represent traffic on the connections. The routing problem then becomes one of finding the shortest path in the graph from any node to any other node. Dijkstra’s algorithm can be used to give the shortest path from any one node to all other nodes (see Exercise 9), so you could use the algorithm repeatedly with different start nodes. Or you could use Floyd’s algorithm. Discuss the advantages and disadvantages of each approach, including an analysis of the order of magnitude of each approach.

SECTION 7.4 TRAVERSAL ALGORITHMS

So far this chapter has considered various path questions about a graph G . Is there a path in G from node x to node y ? Is there a path through G that uses each arc once? Is there a path through G that ends where we started and uses each node once? What is the minimum-weight path between x and y ? In this section we deal with a simpler problem—we only want to write down all the nodes of a simple, connected graph G in some orderly way. This means we must find a path that visits each node at least once, but we can visit it more than once if we don't write it down again. We can also retrace arcs on the graph if necessary, and clearly this would in general be necessary if we were to visit each node in a tree. This process is called **graph traversal**. We already have several mechanisms for tree traversal (Section 6.2). The two algorithms in this section generalize traversal to apply to any simple, connected graph.

Depth-First Search

In the **depth-first search** algorithm for graph traversal, we begin at an arbitrary node a of the graph, mark it visited, and write it down. We then strike out on a path away from a , visiting and writing down nodes, proceeding as far as possible until there are no more unvisited nodes on that path. We then back up the path, at each node exploring any new side paths, until finally we retreat back to a . We then explore any new paths remaining from a . Figure 7.13 shows a graph after the first few nodes (marked by circles) have been visited using depth-first search.

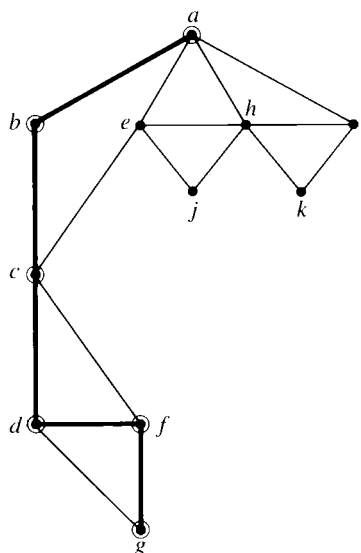


Figure 7.13

For a more formal description of the depth-first search algorithm we will use recursion, where the algorithm invokes itself in the course of its execution. In the following algorithm, the input is a simple, connected graph G and a specified node a ; the output is a list of all nodes in G in depth-first order from a .

ALGORITHM *DEPTHFIRST*

```

DepthFirst(simple, connected graph  $G$ ; node  $a$ )
//Writes nodes in graph  $G$  in depth-first order from node  $a$ 

    mark  $a$  visited
    write ( $a$ )
    for each node  $n$  adjacent to  $a$  do
        if  $n$  not visited then
            DepthFirst( $G, n$ )
        end if
    end for
end DepthFirst

```

In the recursive step, the algorithm is invoked with a new node specified as the starting point. We have not indicated here how to mark visited nodes or how to find those nodes n that are adjacent to a .

EXAMPLE 11

We will apply depth-first search to the graph of Figure 7.13, where a is the initial node. We first mark that we have visited a (it's helpful in tracing the execution of the algorithm to circle a visited node), and then we write out a . Next we search the nodes adjacent to a for an unvisited node. We have a choice here (b, e, h , and i); let us select node b . (Just so we all get the same answers, let's agree to choose the node that is alphabetically first when we have a choice; in practice, the choice would be determined by how the vertices were stored in the graph representation.) Then we invoke the depth-first search algorithm beginning with node b .

We go back to the beginning of the algorithm, where the specified node is now b rather than a . Thus we first mark b visited and write it out. Then we search through nodes adjacent to b to find an unmarked node. Nodes a and c are adjacent to b , but node a is marked as already visited. Node c will do, and we invoke the depth-first search algorithm beginning with node c .

Node c is marked and written out, and we look for unmarked nodes adjacent to c . By our alphabetical convention, we select node d from the set $\{d, f\}$. Continuing in this fashion, after visiting node d we next visit node f and then node g . When we get to node g , we have reached a dead end because there are no unvisited adjacent nodes. Thus the **for** loop of the instance of the algorithm invoked with node g is complete. (The graph at this point looks like Figure 7.13.)

We are therefore done with the algorithm for node g , but node g was (possibly one of) the unmarked nodes adjacent to node f , and we are still in the **for** loop for the instance of the algorithm invoked with node f . As it happens, g is the only unvisited node when we are processing f , therefore we complete the **for** loop and thus the algorithm for node f . Similarly, backing up to node d , the algorithm finds no other adjacent unmarked nodes, and it backs up again to the instance of the algorithm invoked with node c . Thus, after processing node d and everything that came after it until the dead end, we are still in the **for** loop for the algorithm applied to node c . We look for other unmarked nodes adjacent to c and find one—node e . Therefore we apply depth-first search to node e , which leads to nodes h, i , and k before another dead end is reached. Backing up, we have a final new path to try from node h , which leads to node j . The complete list of the nodes, in the order in which they would be written out, is

$a, b, c, d, f, g, e, h, i, k, j$

REMINDER

In a depth-first search, go as far as possible, then back up, catching any paths missed on the way down.

Example 11 makes the depth-first search process sound very complex, but it is much easier to carry out than to write down, as you will see in Practice 14.

PRACTICE 14 Write the nodes in a depth-first search of the graph in Figure 7.14. Begin with node a .

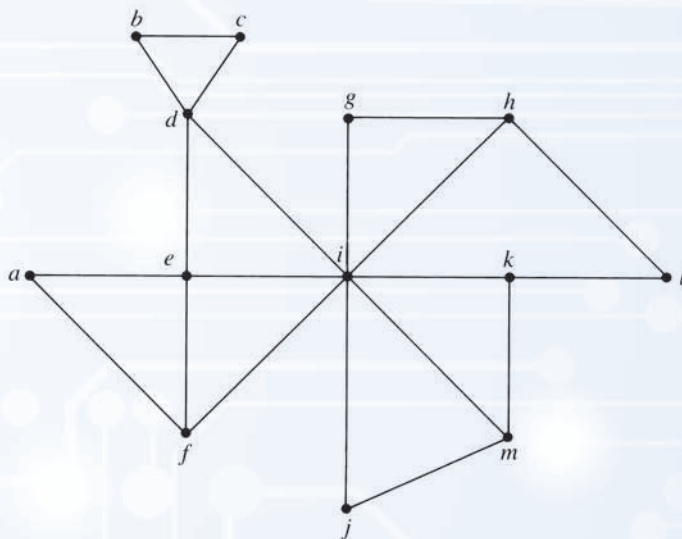


Figure 7.14

Breadth-First Search

In **breadth-first search**, beginning at an arbitrary node a , we first fan out from node a to visit nodes that are adjacent to a ; then we fan out from those nodes, and so on, almost like the concentric circles of ripples in a pond. Figure 7.15 shows the first few nodes visited in the same graph as Figure 7.13, this time using breadth-first search.

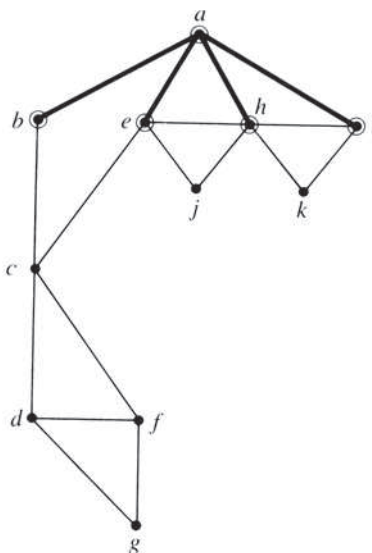


Figure 7.15

To write the breadth-first search algorithm in an elegant fashion, we will use a **queue** structure. A queue is simply a line in which new arrivals are added at the back and departures take place at the front. A checkout line in a grocery store is an example of a queue of customers—a new customer joins the line at the back and departures take place from the front of the line as customers are checked through. The addition of an entry at the back of a queue is called an **enqueue** operation, and a departure from the front of the queue is called a **dequeue** operation. Thus the notation $\text{enqueue}(a, Q)$ denotes adding a to the end of a queue called Q , and $\text{dequeue}(Q)$ denotes removal of the entry currently at the front of Q . We'll also use a function $\text{front}(Q)$, which returns the value of the entry currently at the front of Q but does not remove that entry. In the following algorithm, the input is a simple, connected graph G and a specified node a ; the output is a list of all nodes in G in breadth-first order from a .

ALGORITHM *BREADTHFIRST*

```

BreadthFirst(simple, connected graph  $G$ ; node  $a$ );
//writes nodes in graph  $G$  in breadth first order from node  $a$ 
Local Variable:
queue of nodes  $Q$ 

    initialize  $Q$  to be empty
    mark  $a$  visited
    write( $a$ )
    enqueue( $a, Q$ )
    while  $Q$  is not empty do
        for each node  $n$  adjacent to  $\text{front}(Q)$  do
            if  $n$  not visited then
                mark  $n$  visited
                write( $n$ )
                enqueue( $n, Q$ )
            end if
        end for
        dequeue( $Q$ )
    end while
end BreadthFirst

```

EXAMPLE 12

Let's walk through the algorithm for a breadth-first search of the graph of Figure 7.15 beginning at node a (this is the same graph on which we did the depth-first search in Example 11). We begin by initializing an empty queue Q , marking node a as visited, writing it out, and adding it to the queue. When we first reach the **while** loop, the queue is not empty and a is the entry at the front of the queue. In the **for** loop, we look for unvisited nodes adjacent to a to visit, write them out, and add them to the back of the queue. We may have a choice of nodes to visit here; as before, and purely as a convention, we will agree to visit them in alphabetical order. Thus the first time we complete the **for** loop, we have visited and written out b , e , h , and i , in that order, and added them to the queue. The graph at this point looks

like Figure 7.15. We then remove a from the front of the queue, which as a result contains (from front to back)

$$b, e, h, i$$

In the next iteration of the **while** loop, b is the front element in the queue, and the **for** loop searches for unvisited nodes adjacent to b . The only previously unvisited node here is c , which gets marked as visited, written out and added to the queue. After removing b , the queue contains

$$e, h, i, c$$

Performing the **while** loop again, e is at the front of the queue. A search of the nodes adjacent to node e produces one new node, node j . The graph now looks like Figure 7.16, and after removing e the queue contains

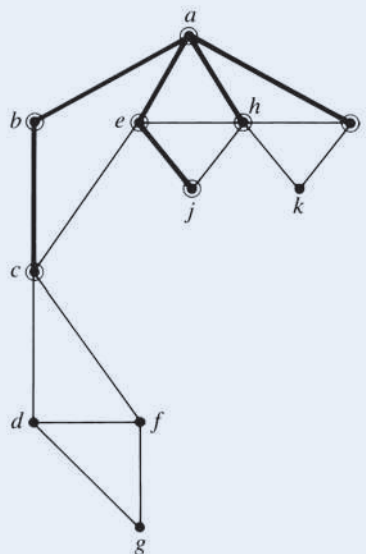
$$h, i, c, j$$


Figure 7.16

When searching for nodes adjacent to h , we pick up one new node, node k . When searching for nodes adjacent to i , no new nodes are added to the queue. When c becomes the first element in the queue, a search for nodes adjacent to c turns up two new nodes, d and f . After adding these to the queue (and removing c), the queue contains

$$j, k, d, f$$

Looking for nodes adjacent to j and then to k adds no new nodes to the queue. When the front of the queue is d , a new node g is found, and the queue (after removing d) is

$$f, g$$

Processing f and then g yields no new nodes. After g is removed from the queue, the queue is empty. The **while** loop—as well as the algorithm—terminates. The list of nodes written out by this process, that is, the nodes in breadth-first order from a , are

$a, b, e, h, i, c, j, k, d, f, g$

Like the depth-first search, the breadth-first search is not difficult to trace; one must just keep track of the nodes that have been visited and the current contents of the queue.

PRACTICE 15 Write the nodes in a breadth-first search of the graph in Figure 7.14, beginning with node a .

Analysis

How much work do the depth-first search and breadth-first searches perform? Both algorithms look for all unvisited nodes adjacent to a given node. Suppose the graph contains n nodes and m arcs. One of the advantages of representing a graph as an adjacency list rather than an adjacency matrix is that this particular operation is more efficient; to find nodes adjacent to node i requires traversing i 's adjacency list, which may be short, rather than traversing row i of the adjacency matrix, which must contain n entries. Therefore we will assume an adjacency list representation of the graph.

In breadth-first search, the algorithm searches all at one time the entire adjacency list of the node at the front of the queue, marking, writing out, and enqueueing the unvisited nodes found. In the depth-first search, the algorithm may be interrupted many times while traversing the adjacency list of a given node to go off (by virtue of the recursion) and process sections of the adjacency lists of other nodes. Eventually, however, every adjacency list is completely covered.

Traversing the adjacency lists of the graph drives the amount of work done in either search. There are n adjacency lists, so the amount of work is at least $\Theta(n)$ because each adjacency list must be checked. Because there are m arcs, the work in traversing the total length of all the adjacency lists is at least $\Theta(m)$. Therefore both depth-first search and breadth-first search are $\Theta(\max(n, m))$ algorithms. If there are more arcs than nodes (the usual case), then $\Theta(\max(n, m)) = \Theta(m)$.

Applications

Depth-first search and breadth-first search can be used as the basis for performing other graph-related tasks, some of which we have solved before. A nonrooted tree structure that is a subgraph of the original graph can be associated with each search. When traversing node i 's adjacency list, if node j is adjacent to i and is previously unvisited, then the i - j arc is added to this subgraph. Because no arc to a previously visited node is used, cycles are avoided and the subgraph is a nonrooted tree. Because all nodes ultimately are visited (for the first time), these trees are spanning trees for the graph. Each tree has $n - 1$ arcs, the minimal number of arcs to connect n nodes. Here we are assuming that arcs are unweighted, but if we consider them to be weighted arcs, each with weight 1, then these trees are minimal spanning trees.

The dark lines in Figure 7.13 are part of the depth-first search tree associated with the search of Example 11, and the dark lines in Figures 7.15 and 7.16 are part of the breadth-first search tree associated with the search of Example 12.

PRACTICE 16

- Complete the depth-first search tree for Example 11.
- Complete the breadth-first search tree for Example 12. ■

The depth-first search and breadth-first search algorithms apply equally well to directed graphs and in the process yield a new algorithm for reachability. To determine whether node j is reachable from node i , do a depth-first (or breadth-first) search beginning at node i ; when the algorithm terminates, check whether node j has been visited. “All pairs” reachability, that is, which nodes are reachable from which nodes, can thus be determined by running depth-first or breadth-first searches using each node in turn as the source node. This process would require $\Theta(n * \max(n, m))$ work. If the graph is very sparse, in which case we have $\max(n, m) = n$, we would have an $\Theta(n^2)$ algorithm for reachability. Recall that Warshall’s algorithm (Section 7.1) was an $\Theta(n^3)$ algorithm. The improvement comes about because in a sparse graph, most adjacency lists will be short or empty, whereas Warshall’s algorithm processes entries in the adjacency matrix even if those entries are 0s. But if the graph is not sparse, the number of arcs can be $\Theta(n^2)$, in which case $\Theta(n * \min(n, m)) = \Theta(n^3)$, the same as Warshall’s algorithm. In addition, Warshall’s algorithm has the advantage of succinct implementation.

In Section 5.2 we defined a topological sort as a way to extend a partial ordering on a finite set to a total ordering. Let the partially ordered set be represented by a directed graph. The topological sort will be achieved by counting the nodes, so let the initial value of the count be 0. Pick a node as a source node and perform a depth-first search from this node. Whenever the search backs up from a node for the final time, assign that node the next counting number. When the depth-first search algorithm terminates, pick an unvisited node (if one exists) to be the source for another depth-first search, and continue to increment the counting number. Continue this process until there are no unvisited nodes left in the graph. A topological sort results by ordering the nodes in the reverse order of their counting number. This process for topological sorting works because we assign the counting number when we back up from a node for the final time. Its counting number will then be higher than the numbers of all the nodes reachable from it, that is, all the nodes of which it is a predecessor in the partial ordering,

EXAMPLE 13

Figure 7.17a is a directed graph that represents a partial ordering. Choosing d (arbitrarily) as the source node and performing a depth-first search, we visit e and f , at which point we must back up. Node f is assigned the counting number 1, but we are not yet done with e , because we can go on to visit g . Backing up from g , g is assigned the counting number 2. At this point we back up from e for the final time and assign e the number 3 and then d the number 4. We choose a as the source node for another search. We visit node c and then must back up, so c and a are assigned the numbers 5 and 6, respectively. Beginning with b as a source node, there

is nowhere to go, and b is assigned the number 7. There are no unvisited nodes left in the graph, so the process stops. The numbering scheme is shown in Figure 7.17b.

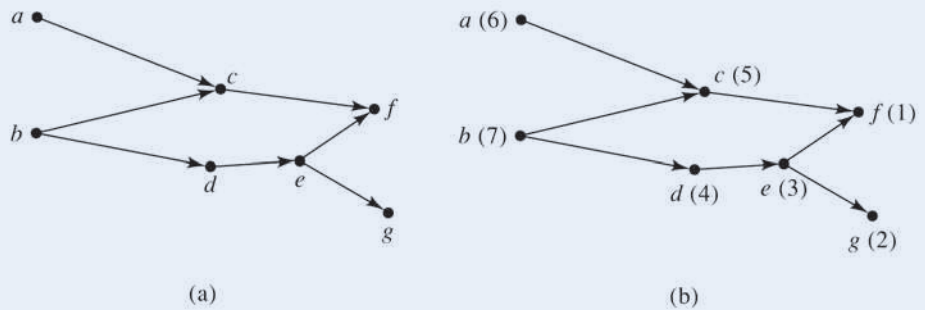


Figure 7.17

In reverse order of the counting numbers, we get

7	6	5	4	3	2	1
b	a	c	d	e	g	f

which is a topological ordering.

PRACTICE 17

Use the depth-first search algorithm to do a topological sort on the graph in Figure 7.18. Indicate the counting numbers on the graph.

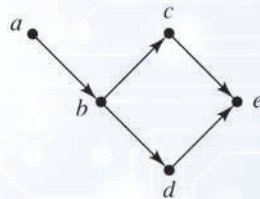


Figure 7.18

Now consider a graph G (undirected) that may not be a connected graph. A **connected component** of G is a subgraph of G that is both connected and not a subgraph of a larger connected subgraph. In Figure 7.19 there are three connected components. Of course, if the original graph is connected, then it has only one connected component.



Figure 7.19

A depth-first or breadth-first search can be used to find the connected components of a graph. We pick an arbitrary node as a source node and then conduct a search. When the algorithm terminates, all visited nodes belong to one component. We then find an unvisited node in the graph to serve as a source for another search, which will produce a second component. We continue this process until there are no unvisited nodes in the graph.

Although we defined reachability only for directed graphs, the concept also makes sense for undirected, unconnected graphs. Let us consider only simple undirected, unconnected graphs but impose the convention that, even though there are no loops, each node is reachable from itself. Reachability then becomes an equivalence relation on the set of nodes of the graph; our convention imposes the reflexive property, and symmetry and transitivity follow because the graph is undirected. This equivalence relation partitions the nodes of the graph into equivalence classes, and each class consists of the nodes in one connected component of the graph. Warshall's algorithm can be applied to undirected graphs as well as directed graphs. Using Warshall's algorithm results in a matrix from which the nodes making up various components of the graph can be determined, but this requires more work than using the depth-first search.

As a final remark about depth-first search, we saw in Section 1.5 that the programming language Prolog, when processing a query based on a recursive definition, pursues a depth-first search strategy (Example 40).

SECTION 7.4 REVIEW

TECHNIQUES

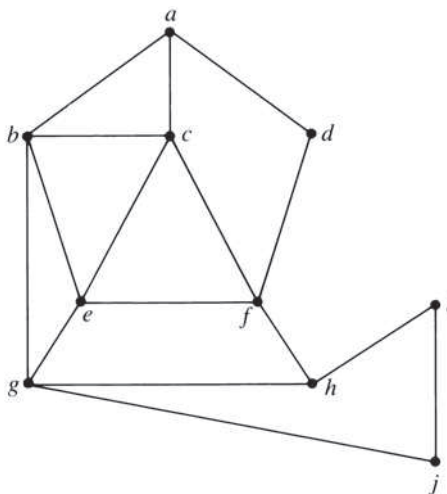
- W Conduct a depth-first search of a graph.
- W Conduct a breadth-first search of a graph.

MAIN IDEAS

- Algorithms exist to visit the nodes of a graph systematically.
- Depth-first and breadth-first searches can serve as a basis for other tasks.

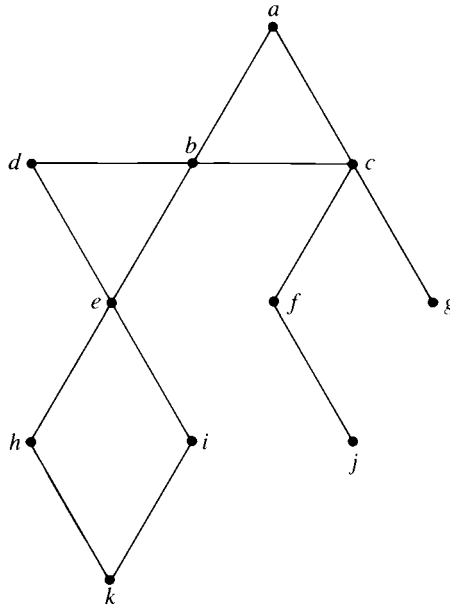
EXERCISES 7.4

For Exercises 1–6, write the nodes in a depth-first search of the following graph, beginning with the node specified.



1. *a*
2. *c*
3. *d*
4. *g*
5. *e*
6. *h*

For Exercises 7–10, write the nodes in a depth-first search of the following graph, beginning with the node specified.



7. *a* 8. *e* 9. *f* 10. *h*

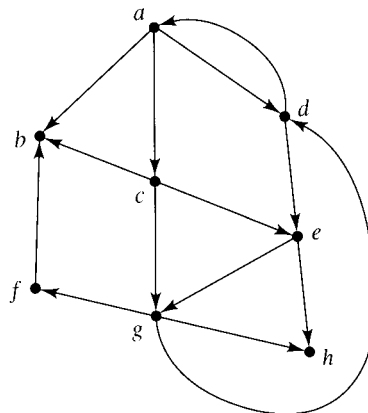
For Exercises 11–16, write the nodes in a breadth-first search of the graph for Exercises 1–6, beginning with the node specified.

11. *a* 12. *c* 13. *d* 14. *g* 15. *e* 16. *h*

For Exercises 17–20, write the nodes in a breadth-first search of the graph for Exercises 7–10, beginning with the node specified.

17. *a* 18. *e* 19. *f* 20. *h*

For Exercises 21–24, write the nodes in a depth-first search of the following graph, beginning with the node specified.

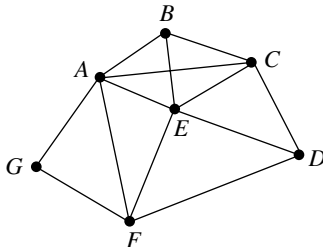


21. *a* 22. *g* 23. *f* 24. *e*

For Exercises 25–28, write the nodes in a breadth-first search of the graph for Exercises 21–24, beginning with the node specified.

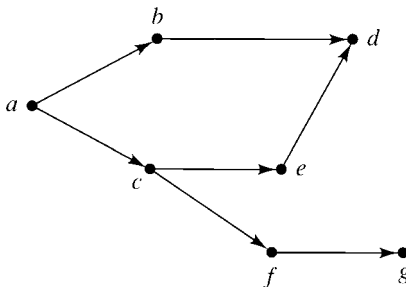
25. a 26. g 27. f 28. e

29. In the computer network in the accompanying figure, the same message is to be broadcast from node C to nodes A , E , F , and G . One way to do this is to find the shortest path from C to each of these nodes and send out multiple copies of the same message. A more efficient approach is to send one copy out from C along a spanning tree for the subgraph containing the nodes involved. Use the depth-first search algorithm to find a spanning tree for the subgraph.

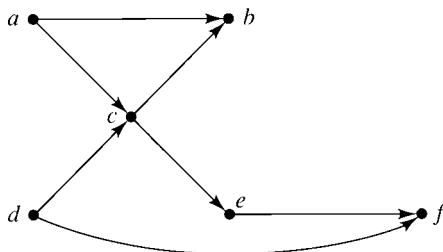


30. Using the graph for Exercise 29, use the breadth-first search algorithm to find a spanning tree for the subgraph.

31. Use the depth-first search algorithm to do a topological sort on the following graph. Indicate the counting numbers on the graph. Also state the starting node or nodes for the search.



32. Use the depth-first search algorithm to do a topological sort on the following graph. Indicate the counting numbers on the graph. Also state the starting node or nodes for the search.



33. The data structure used to implement a breadth-first search is a queue. What is the appropriate data structure to implement a depth-first search?

34. Find a way to traverse a tree in level order, that is, so that all nodes at the same depth are listed from left to right for increasing depth. (*Hint*: We already have a way to do this.)

35. Describe how the depth-first search algorithm can be used in a connected (undirected) graph to detect the presence of cycles in the graph. (While it is simple to look at Figure 7.13 and see that $a-b-c-e-a$, for example, is a cycle, in a huge graph with thousands of node and arcs, a cycle may be less easy to spot, in addition to which you might not even have a visual representation.)

36. a. Describe the order in which nodes are visited in a breadth-first search of the bipartite complete graph $K_{m,n}$.
 b. Describe the order in which nodes are visited in a depth-first search of the bipartite complete graph $K_{m,n}$.

SECTION 7.5 ARTICULATION POINTS AND COMPUTER NETWORKS

The Problem Statement

In a graph that represents a computer network, the nodes denote the communicating entities (end-user computers, servers, routers, and so on) and the arcs denote the communications medium (coaxial cable, fiber optic, and so on). Such a graph should be a connected graph, so that there is a path between every pair of nodes. To minimize the length of cable or wire required, we would choose a minimum spanning tree. However, if an arc in a minimum spanning tree is removed (for example, that section of cable or wire is damaged or broken), then the graph is no longer connected. Each arc becomes a single point of failure for the network. That is why such a network usually contains more arcs than just those of a minimal spanning tree. However, even in a graph sufficiently rich in arcs to withstand the loss of a single arc, a node may be a single point of failure. If the node fails (and thus is logically removed), the arcs of which that node is an endpoint are disabled and the result may be a disconnected graph.

DEFINITION ARTICULATION POINT

A node in a simple, connected graph is an **articulation point** if its removal (along with its attached arcs) causes the remaining graph to be disconnected.

EXAMPLE 14

Node d in the graph of Figure 7.20a is an articulation point. Removing d results in the disconnected graph of Figure 7.20b.

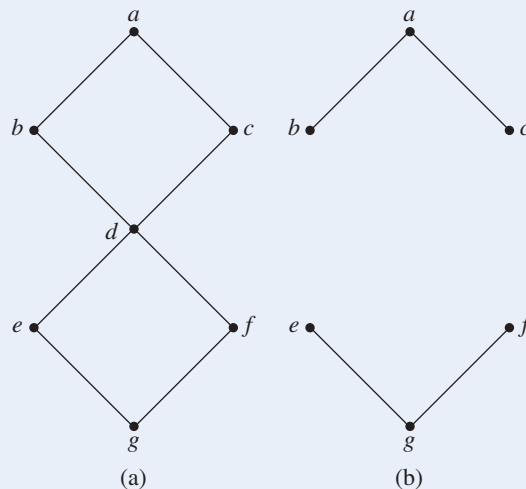


Figure 7.20

DEFINITION BICONNECTED GRAPH

A simple, connected graph is **biconnected** if it has no articulation points.

The presence of articulation points is clearly an undesirable feature of a network. Although it is easy to spot an articulation point in a graph as small as that of Figure 7.20a, we will develop an algorithm that will detect such points no matter how large the graph (and, of course, does not require a visual representation of the graph). Articulation points separate the graph into **biconnected components**, subgraphs that are biconnected and are not subgraphs of larger biconnected subgraphs. In Figure 7.20, $a-b-d-c$ and $d-e-g-f$ are biconnected components.

The Idea behind the Algorithm

The key to this algorithm is depth-first search. We know from the previous section that a depth-first search determines a nonrooted tree. An arc is added to the tree whenever the search progresses to a previously unvisited node. Arcs of the graph belonging to this tree are called **tree arcs**. The remaining arcs in the graph are called **back arcs**.

EXAMPLE 15

In Figure 7.20 a depth-first search from node a visits nodes in the order $a, b, d, c, e, g,$ and f . In Figure 7.21 the tree arcs are dark, and the back arcs are light.

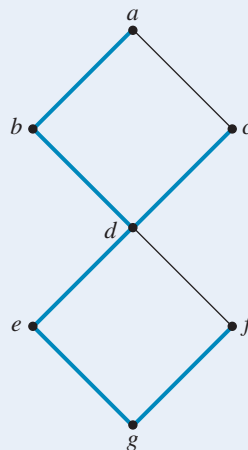


Figure 7.21

The depth-first search tree passes through all nodes. To detect nodes that are articulation points, we examine their relative positions in the tree. First consider the single node that is the starting point of the depth-first search tree. If only one tree arc emanates from the starting node, then as the tree continues, all other nodes in the graph can be reached from the node at the other end of that tree arc. Therefore, removing the starting node will not disconnect the graph. However, if two or more tree arcs emanate from the starting node, then the only way to get from one subtree to another is to pass back through the starting node. In this case, removing the starting node disconnects the graph.

Thus in Figure 7.21 node a is the starting node of the depth-first search tree, and there is a single tree arc emanating from a . Removing node a (and its two arcs) does not disconnect the graph. Had we begun a depth-first search at node d , however, the tree would have looked like Figure 7.22. There would be two tree arcs coming from node d , showing that d is an articulation point.

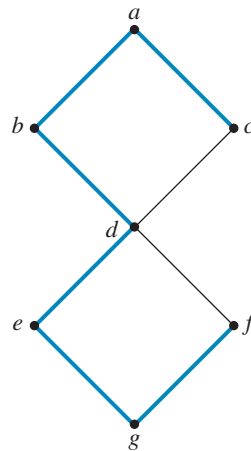


Figure 7.22

Consider any node n that is a leaf of the depth-first search tree (attached to the end of a single tree arc). Such a leaf may be a leaf of the graph itself, that is, a node of degree 1, in which case it is clearly not an articulation point. If not, then the other arcs emanating from n were not used in the depth-first search, so the nodes adjacent to n are reachable through alternative paths that do not go through n . Because n is not needed on a path to any other node, its removal does not disconnect the graph. Therefore no leaves of the depth-first search tree are articulation points. In Figure 7.21 node c , for example, is a leaf of the depth-first search tree; the arc from c to a is a back arc, so node a is accessible through another route that does not require node c . Node c can be removed without disconnecting the graph.

Now consider a node n that is not a leaf in the depth-first search tree and is not the starting node. Because n is not a leaf, there are one or more subtrees below n . Suppose there is a single subtree; let x be a node on this subtree. If x has a back arc to some node that precedes n in the depth-first search (an “ancestor” of n), then this arc provides part of an alternative path for x —and all other nodes in the subtree—to be connected with the rest of the graph without using node n . In this case n is not an articulation point. (See Figure 7.23a, where removing n and its attached arcs does not disconnect the graph.) If there is more than one subtree below n , then n will not be an articulation point if and only if each subtree has such an “escape route” allowing it to connect with the rest of the graph—including the other subtrees—without going through n . (See Figure 6.23b; note that the back arc from y to z does not help because it does not reach back to an ancestor of n .)

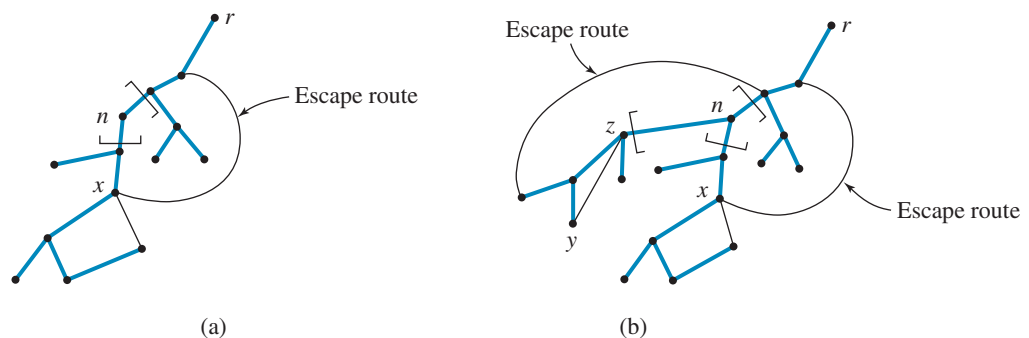


Figure 7.23

The Algorithm Itself

The key to the algorithm, as we may guess from the foregoing discussion, is keeping track of the destinations of the back arcs. We will assign a tree number to each node that corresponds to the order in which that node is visited in the depth-first search. Thus the starting node in a depth-first search has tree number 1, the next node visited has tree number 2, and so on. In addition, we will maintain a “back number” for each node x . The back number will be the minimum tree number of a node (the farthest back node) reachable from x using either back arcs from x or from descendants of x in the subtree. To incorporate information about back arcs of descendants of x , the back number of x is adjusted when the depth-first search backs up to node x from farther down in the tree. Node n is an articulation point whenever a subtree of n has no back arc to an ancestor of n , and this circumstance is detected when the search backs up to n from the subtree. Suppose the search is backing up from x to n . If the back number of x at this point is not smaller than the tree number of n , then n is an articulation point.

The following algorithm carries out the depth-first search and builds the depth-first search tree. It correctly handles both leaves and nonstarting-node non-leaves of the depth-first search tree, leaving only the starting node as a special case. It also assumes that the graph structure itself contains the tree number and back number for each node.

ALGORITHM *ARTPOINT*

```

ArtPoint(graph  $G$ ; node  $n$ ; integer  $TreeNumber$ )
//detects articulation points in  $G$  by depth-first search from  $n$ ; graph  $G$ 
//also maintains a tree number  $TN$  and back number  $BN$  value for each
//node;  $TreeNumber = 0$  when first invoked.

Local variable:
node  $x$  //temporary node

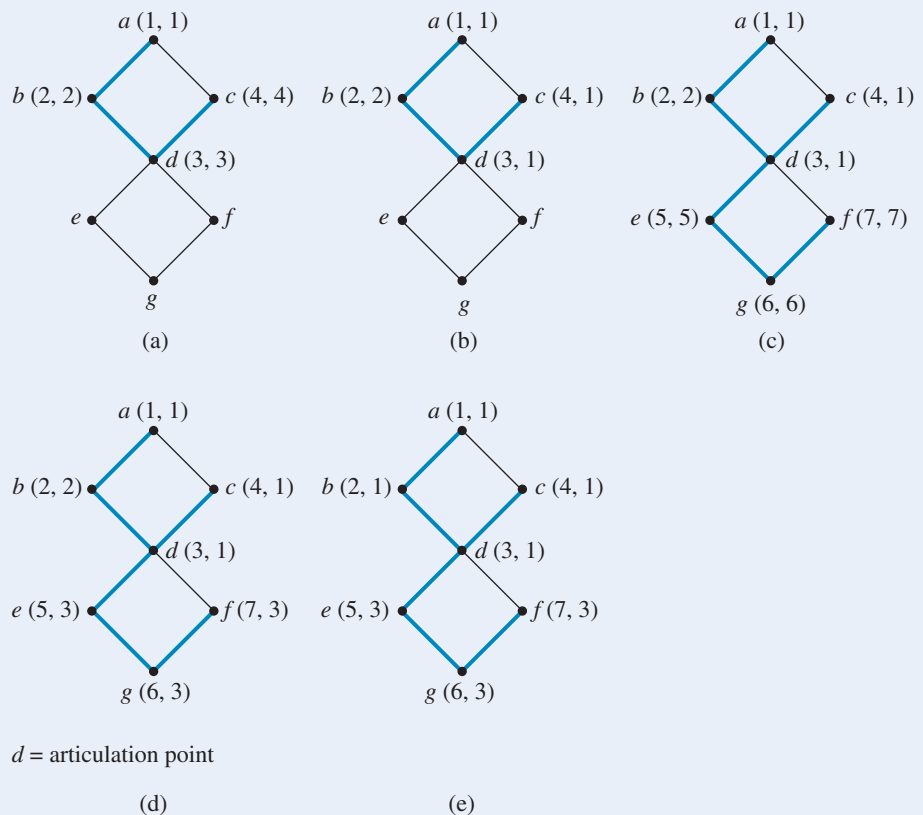
mark  $n$  visited //first encountering  $n$ , assign its numbers
 $TreeNumber = TreeNumber + 1$ 
 $TN[n] = TreeNumber$ 
 $BN[n] = TN[n]$ 
for each node  $x$  adjacent to  $n$  by a nontree edge do
  if  $x$  not visited then
    make  $n-x$  a tree edge
    ArtPoint( $G, x, TreeNumber$ )
    //depth-first search now backing up to  $n$  from  $x$ 
    if  $BN[x] \geq TN[n]$  then //line 1
      write(“ $n$  is an articulation point”)
    else
      //adjust back number of  $n$ 
       $BN[n] = \min(BN[n], BN[x])$  //line 2
    end if
  else
    //arc  $n-x$  is a back edge, adjust  $BN[n]$ 
     $BN[n] = \min(BN[n], TN[x])$  //line 3
  end if
end for
end ArtPoint

```

EXAMPLE 16

We will trace the articulation point algorithm on the graph of Figure 7.20a, where a is the starting node. The tree begins with arcs $a-b$, $b-d$, and $d-c$. Each new node is numbered with a consecutive tree number, and its back number is set equal to its tree number (Figure 7.24a; in the figure, the numbers in parentheses are the tree number and the back number, respectively). While processing node c , the back edge to a is discovered, and the back number of c is adjusted to 1, the tree number of a (line 3 in the algorithm description). This action completes the processing of node c , and the depth-first search backs up to d . The back number of c is less than the tree number of d , so the back number of d is adjusted to equal that of c (line 2). The situation at this point is shown in Figure 7.24b.

The depth-first search moves on to nodes e , g , and f (Figure 7.24c). At f the back arc to d is found, and the back number of f is set equal to the tree number of d (line 3). Backing up from f to g , the back number of g is adjusted to equal the back number of f (line 2) and similarly for e (line 2 again). (See Figure 7.24d.)

**Figure 7.24**

Finally, in backing up from e to d , the back number of e is greater than or equal to the tree number of d , so d is declared an articulation point (line 1). The recursion backs up to node b , adjusting the back number of b , and then node a , at which point line 1 would seem to apply (Figure 7.24e). But a is the starting node of the search and so is not an articulation point, because there is only one tree arc from a .

PRACTICE 18

In Figure 7.25, the depth-first search began at node a . Explain why each node is marked as it is and how it is concluded that c is an articulation point.

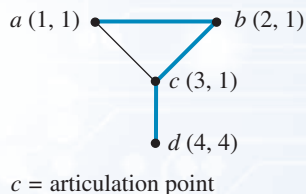


Figure 7.25

SECTION 7.5 REVIEW

TECHNIQUE

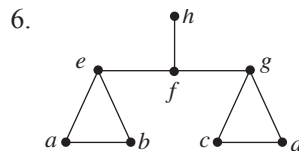
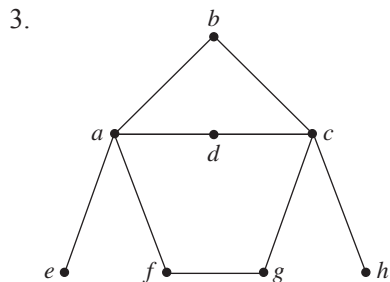
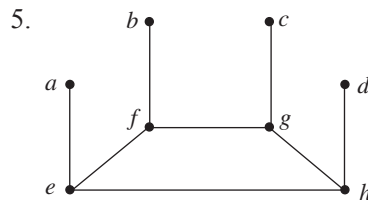
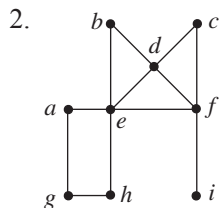
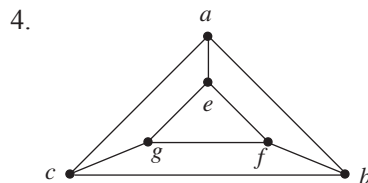
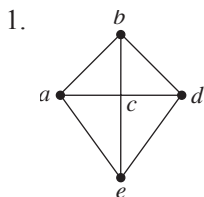
W Find articulation points in a simple, connected graph (using algorithm *ArtPoint*)

MAIN IDEA

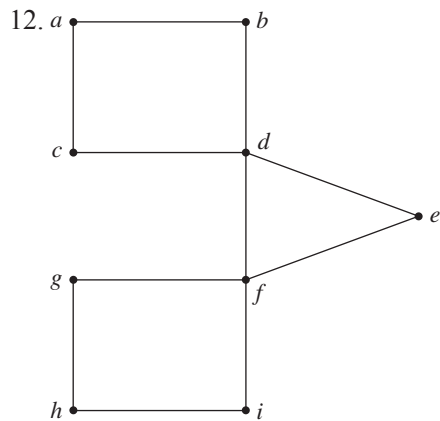
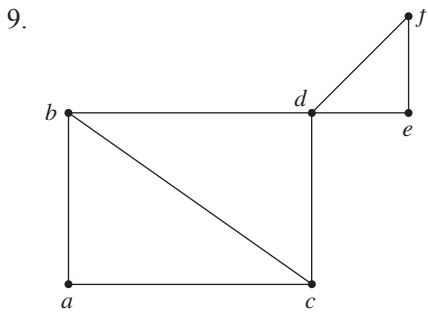
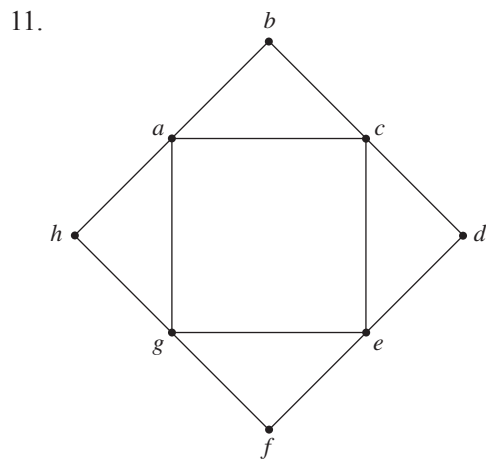
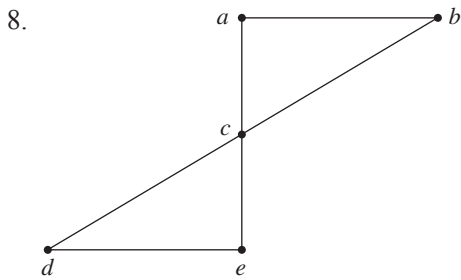
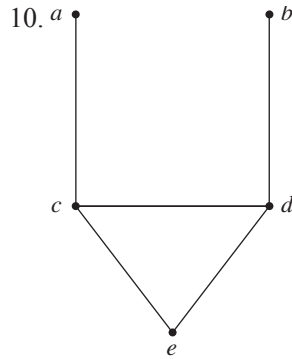
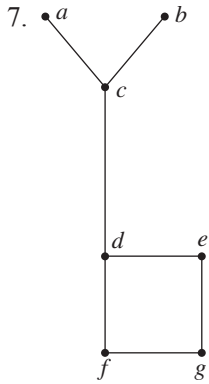
• Articulation points represent single points of failure in a computer network, but an algorithm exists to detect their presence.

EXERCISES 7.5

For Exercises 1–6, draw the depth-first search trees, where node a is the starting node of the depth-first search. Identify the back arcs.



For Exercises 7–12, use algorithm *ArtPoint* to find the articulation points. Label the tree number and back number for each node, both as first assigned and as changed. Draw the biconnected components of the graph.



CHAPTER 7 REVIEW

TERMINOLOGY

adjacency relation (p. 555)	dequeue (p. 599)	odd node (p. 572)
articulation point (p. 607)	enqueue (p. 599)	queue (p. 599)
back arc (p. 608)	Euler path (p. 572)	reachable node (p. 557)
biconnected component (p. 608)	even node (p. 572)	reachability matrix (p. 559)
biconnected graph (p. 607)	graph traversal (p. 596)	spanning tree (p. 587)
breadth-first search (p. 598)	greedy algorithm (p. 586)	tree arc (p. 608)
connected component (p. 603)	Hamiltonian circuit (p. 576)	
depth-first search (p. 596)	minimal spanning tree (p. 587)	

SELF-TEST

Answer the following true–false questions.

Section 7.1

- Any binary relation on a set N has an associated adjacency matrix.
- Transitive closure is the adjacency relation equivalent of reachability.
- The reachability matrix \mathbf{R} for a directed graph G is computed by taking the powers of the adjacency matrix up to n^2 .
- Warshall's algorithm proceeds by computing, in turn, the number of paths of length 1, then length 2, and so on, between nodes.
- Warshall's algorithm computes symmetric closure in the case of a symmetric adjacency relation.

Section 7.2

- A graph with four odd nodes can still be a connected graph.
- An Euler path exists in any graph with an even number of odd nodes.
- An $\Theta(n^2)$ algorithm exists to test the existence of an Euler path in a graph with n nodes.
- A Hamiltonian circuit uses each arc and node of the graph exactly once except for the starting and ending node.
- No algorithm to solve the Hamiltonian circuit problem is known.

Section 7.3

- Dijkstra's algorithm for the shortest path in a graph maintains a set IN and adds at each step the node closest to a node in IN .
- A greedy algorithm is one that recursively divides the problem into as many subproblems as possible.
- The minimal spanning tree for a graph may not be unique.

- Using a linked-list representation for nodes not in IN does not improve the order of magnitude of the worst-case work done by Dijkstra's algorithm.
- The collection of all arcs that are not in a minimal spanning tree for a graph will also form a spanning tree, but it may not be minimal.

Section 7.4

- The depth-first search visits nodes at the bottom of the graph first.
- In a breadth-first search beginning with node i , all the nodes adjacent to i are visited in order.
- An analysis of the depth-first search and the breadth-first search shows them to be algorithms of the same order of magnitude.
- Preorder traversal is the tree equivalent of the breadth-first search, using the root as the starting node.
- Topological sorting can be done by a succession of breadth-first searches on a directed graph.

Section 7.5

- If node n is an articulation point in a connected graph, then any path between any two nodes in the graph must pass through n .
- A biconnected graph is a simple, connected graph with no articulation points.
- When a node n is first reached during a depth-first search, any other arcs from n to previously visited nodes are back arcs.
- A node n where every subtree of n in the depth-first search tree has a back arc to a predecessor of n is not an articulation point.
- The root of a depth-first search is always an articulation point in the graph because any node's back number will be greater than or equal to its tree number.

ON THE COMPUTER

For Exercises 1–5, write a computer program that produces the desired output from the given input.

1. *Input:* Adjacency matrix \mathbf{A} for a directed graph
Output: Reachability matrix \mathbf{R} for the graph, computed from the formula $\mathbf{R} = \mathbf{A} \vee \mathbf{A}^{(2)} \vee \cdots \vee \mathbf{A}^{(n)}$
2. *Input:* Adjacency matrix \mathbf{A} for a directed graph
Output: Reachability matrix \mathbf{R} for the graph, computed by using Warshall's algorithm
3. *Input:* Adjacency matrix \mathbf{A} for a graph
Output: Message indicating whether the graph has an Euler path
4. *Input:* Adjacency matrix \mathbf{A} for a simple weighted graph or directed graph and two nodes in the graph
Output: Distance for the shortest path between the two nodes or a message that no path exists; vertices in the shortest path if one exists (*Hint:* You will need to find some way of denoting which vertices are currently in IN .)
5. *Input:* Adjacency matrix \mathbf{A} for a simple, weighted, connected graph
Output: Arcs (as ordered pairs) in a minimal spanning tree

For Exercises 6–8, first write a function that collects information from the user about a graph and builds an adjacency list representation of the graph; incorporate this function in the programs requested.

6. *Input:* Information about a graph (see instructions above) and a node in the graph
Output: Nodes in a depth-first search of the graph beginning with the given node
7. *Input:* Information about a graph (see instructions above) and a node in the graph
Output: Nodes in a breadth-first search of the graph beginning with the given node
8. *Input:* Information about a graph (see instructions above)
Output: Articulation points in the graph

This page intentionally left blank

Boolean Algebra and Computer Logic

CHAPTER OBJECTIVES

After studying this chapter, you will be able to:

- Determine whether a given mathematical structure is a Boolean algebra.
- Prove properties about Boolean algebras.
- Understand what it means for an isomorphism function to preserve the effects of a binary operation or other property.
- Draw a logic network to represent a Boolean expression.
- Write a Boolean expression to represent a logic network.
- Write the truth function for a Boolean expression or logic network.
- Write a Boolean expression in canonical sum-of-products form for a given truth function.
- Use NAND and NOR gates as well as AND, OR, and NOT gates to build logic networks.
- Write a truth function from a description of a logical control device.
- Simplify Boolean expressions and logic networks using Karnaugh maps.
- Simplify Boolean expressions and logic networks using the Quine–McCluskey method.

You have been hired by Rats R Us to build the control logic for the production facilities for a new anticancer chemical compound being tested on rats. The control logic must manage the opening and closing of two valves, A and B, downstream of the mixing vat. Valve A is to open whenever the pressure in the vat exceeds 50 psi (pounds per square inch) and the salinity of the mixture exceeds 45 g/L (grams per liter). Valve B is to open whenever valve A is closed and the temperature exceeds 53°C and the acidity falls below 7.0 pH (lower pH values mean more acidity).

Question: How many and what type of logic gates will be needed in the circuit?

The answer to this electronics problem lies, surprisingly, in a branch of mathematics developed around 1850 by George Boole, an English mathematician. Boole was interested in developing rules of “algebra” for logical thinking, similar to the rules of algebra for numerical thinking. Derided at the time as useless, if harmless, Boole’s work is the foundation for the electronics found in computers today.

In Section 8.1 we define Boolean algebra as a mathematical model of both propositional logic and set theory. The definition requires every Boolean algebra to have certain properties, from which many additional properties can be derived. This section also discusses what it means for two instances of a Boolean algebra to be isomorphic.

Section 8.2 establishes a relationship between the Boolean algebra structure and the wiring diagrams for the electronic circuits in computers, calculators, industrial control devices, telephone systems, and so forth. Indeed, we will see that truth functions, expressions made up of variables and the operations of Boolean algebra, and these wiring diagrams are all related. As a result, we can effectively pass from one formulation to another and still preserve characteristic behavior with respect to truth values. We will also find that we can simplify wiring diagrams by using properties of Boolean algebras. In Section 8.3 we will look at two other procedures for simplifying wiring diagrams.

SECTION 8.1 BOOLEAN ALGEBRA STRUCTURE

Let us revisit the wffs of propositional logic and associate with them a certain type of function. Suppose a propositional wff P has n statement letters. Then each row of the truth table for that wff associates a value of T or F with an n -tuple of T–F values. The entire truth table defines a function f such that $f: \{T, F\}^n \rightarrow \{T, F\}$. The function associated with a tautology maps $\{T, F\}^n \rightarrow \{T\}$, and the function associated with a contradiction maps $\{T, F\}^n \rightarrow \{F\}$.

EXAMPLE 1

We've seen this idea before. From Example 31 in Chapter 5, the function $f: \{T, F\}^2 \rightarrow \{T, F\}$ for the wff $A \vee B'$ is given by the following truth table.

A	B	B'	$A \vee B'$
T	T	F	T
T	F	T	T
F	T	F	F
F	F	T	T

Here $f(T, F) = T$ and $f(F, T) = F$.

Suppose we agree, for any propositional wff P with n statement letters, to let the symbol P denote not only the wff but also the corresponding function defined by the truth table. If P and Q are equivalent wffs, then they have the same truth tables and therefore define the same function. Then we can write $P = Q$ rather than $P \Leftrightarrow Q$. This simply confirms that a given function has multiple names, although a given wff defines a unique function.

With this agreement, the short list of tautological equivalences from Section 1.1 can be written as follows, where \vee and \wedge denote disjunction and conjunction, respectively, A' denotes the negation of a statement A , 0 stands for any contradiction, and 1 stands for any tautology:

1a. $A \vee B = B \vee A$	1b. $A \wedge B = B \wedge A$	(commutative properties)
2a. $(A \vee B) \vee C =$ $A \vee (B \vee C)$	2b. $(A \wedge B) \wedge C =$ $A \wedge (B \wedge C)$	(associative properties)
3a. $A \vee (B \wedge C) =$ $(A \vee B) \wedge (A \vee C)$	3b. $A \wedge (B \vee C) =$ $(A \wedge B) \vee (A \wedge C)$	(distributive properties)
4a. $A \vee 0 = A$	4b. $A \wedge 1 = A$	(identity properties)
5a. $A \vee A' = 1$	5b. $A \wedge A' = 0$	(complement properties)

Switching gears a bit, in Section 4.1 we studied set identities among the subsets of a set S (the elements of $\wp(S)$). We found the following list of set identities, where \cup and \cap denote the union and intersection of sets, respectively, A' is the complement of a set A , and \emptyset is the empty set:

1a. $A \cup B = B \cup A$	1b. $A \cap B = B \cap A$	(commutative properties)
2a. $(A \cup B) \cup C =$ $A \cup (B \cup C)$	2b. $(A \cap B) \cap C =$ $A \cap (B \cap C)$	(associative properties)
3a. $A \cup (B \cap C) =$ $(A \cup B) \cap (A \cup C)$	3b. $A \cap (B \cup C) =$ $(A \cap B) \cup (A \cap C)$	(distributive properties)
4a. $A \cup \emptyset = A$	4b. $A \cap S = A$	(identity properties)
5a. $A \cup A' = S$	5b. $A \cap A' = \emptyset$	(complement properties)

These two lists of properties are similar. The disjunction of statements and the union of sets seem to play the same roles in their respective environments. So do the conjunction of statements and the intersection of sets. A contradiction seems to correspond to the empty set and a tautology to S . What should we make of this resemblance?

Models or Abstractions

We seem to have found two different examples—propositional logic and set theory—that share some common properties. One of the hallmarks of scientific thought is to look for patterns or similarities among various observed phenomena. Are these similarities manifestations of some underlying general principle? Can the principle itself be identified and studied? Could this research shed light on the behavior of various instances of this principle? Sometimes, as seems to be the case with propositional logic and set theory, similar mathematical properties or behavior can be seen in different contexts. A mathematical structure is a formal model that serves to embody or explain this commonality, just as in physics the law of gravity is a formal model of why apples fall, the ocean has tides, and the planets revolve around the sun.

Mathematical principles are models or abstractions intended to capture properties that may be common to different instances or manifestations. These principles are sometimes expressed as *mathematical structures*—abstract sets of objects, together with operations on or relationships among those objects that obey certain rules. (This concept may give you a clue about why this book is titled as it is.)

We can liken a mathematical structure to a human skeleton. We can think of the skeleton as the basic structure of the human body. People may be thin or fat, short or tall, black or white, and so on, but stripped down to skeletons they all look pretty much alike. Although the outward appearances differ, the inward structure, the shape and arrangement of the bones, is the same. Similarly, mathematical

structures represent the underlying sameness in situations that may appear outwardly different.

It appears reasonable to abstract the common properties (tautological equivalences and set identities) for propositional wffs and set theory. Thus we will soon define a mathematical structure called a Boolean algebra that incorporates these properties. First, however, we note that modeling or abstracting is not an entirely new idea to us.

1. We used predicate logic to model reasoning and formally defined an interpretation as a specific instance of predicate logic (Section 1.3).
2. We defined the abstract ideas of partial ordering and equivalence relation, and considered a number of specific instances that could be modeled as posets or sets on which an equivalence relation is defined (Section 5.1).
3. We noted that graph and tree structures can model a great variety of instances (Sections 6.1 and 6.2).

Boolean algebra is just another model or abstraction for which we already have two instances.

Definition and Properties

Let us characterize formally the similarities between propositional logic and set theory. In each case we are talking about items from a set: a set of wffs or a set of subsets of a set S . In each case we have two binary operations and one unary operation on the members of the set: disjunction/conjunction/negation or union/intersection/complementation. In each case there are two distinguished elements of the set: 0/1 or \emptyset/S . Finally, there are the 10 properties that hold in each case. Whenever all these features are present, we say that we have a Boolean algebra.

DEFINITION BOOLEAN ALGEBRA

A **Boolean algebra** is a set B on which are defined two binary operations $+$ and \cdot and one unary operation $'$ and in which there are two distinct elements 0 and 1 such that the following properties hold for all $x, y, z \in B$:

1a. $x + y = y + x$	1b. $x \cdot y = y \cdot x$	(commutative properties)
2a. $(x + y) + z =$ $x + (y + z)$	2b. $(x \cdot y) \cdot z =$ $x \cdot (y \cdot z)$	(associative properties)
3a. $x + (y \cdot z) =$ $(x + y) \cdot (x + z)$	3b. $x \cdot (y + z) =$ $(x \cdot y) + (x \cdot z)$	(distributive properties)
4a. $x + 0 = x$	4b. $x \cdot 1 = x$	(identity properties)
5a. $x + x' = 1$	5b. $x \cdot x' = 0$	(complement properties)

What, then, is the Boolean algebra structure? It is a formalization that abstracts, or models, the two cases we have considered (and perhaps others as well). There is a subtle philosophical distinction between the formalization itself, the *idea* of the Boolean algebra structure, and any instance of the formalization, such as these two cases. Nevertheless, we will often use the term *Boolean algebra* to describe both the idea and its occurrences. This usage should not be confusing. We often have a mental idea (“chair,” for example), and whenever we encounter a

concrete example of the idea, we also call it by our word for the idea (this object is a “chair”).

The formalization helps us focus on the essential features common to all examples of Boolean algebras, and we can use these features—these facts from the definition of a Boolean algebra—to prove other facts about Boolean algebras. Then these new facts, once proved in general, hold in any particular instance of a Boolean algebra. To use our analogy, if we ascertain that in a typical human skeleton, “the thighbone is connected to the kneebone,” then we don’t need to reconfirm the fact in every person we meet.

We denote a Boolean algebra by $[B, +, \cdot, ', 0, 1]$.

EXAMPLE 2

Let $B = \{0, 1\}$ (the set of integers 0 and 1) and define binary operations $+$ and \cdot on B by $x + y = \max(x, y)$, $x \cdot y = \min(x, y)$. Then we can illustrate the operations of $+$ and \cdot by the following tables.

$+$	0	1
0	0	1
1	1	1

\cdot	0	1
0	0	0
1	0	1

A unary operation $'$ can be defined by means of a table, as follows, instead of by a verbal description.

$'$	
0	1
1	0

Thus $0' = 1$ and $1' = 0$. Then $[B, +, \cdot, ', 0, 1]$ is a Boolean algebra. We can verify the 10 properties by checking all possible cases. Thus, for property 2b, the associativity of \cdot , we show that

$$\begin{aligned} (0 \cdot 0) \cdot 0 &= 0 \cdot (0 \cdot 0) = 0 \\ (0 \cdot 0) \cdot 1 &= 0 \cdot (0 \cdot 1) = 0 \\ (0 \cdot 1) \cdot 0 &= 0 \cdot (1 \cdot 0) = 0 \\ (0 \cdot 1) \cdot 1 &= 0 \cdot (1 \cdot 1) = 0 \\ (1 \cdot 0) \cdot 0 &= 1 \cdot (0 \cdot 0) = 0 \\ (1 \cdot 0) \cdot 1 &= 1 \cdot (0 \cdot 1) = 0 \\ (1 \cdot 1) \cdot 0 &= 1 \cdot (1 \cdot 0) = 0 \\ (1 \cdot 1) \cdot 1 &= 1 \cdot (1 \cdot 1) = 1 \end{aligned}$$

For property 4a, we show that

$$\begin{aligned} 0 + 0 &= 0 \\ 1 + 0 &= 1 \end{aligned}$$

Bear in mind that Example 2 illustrates a particular instance of the Boolean algebra structure. Any Boolean algebra must, by the definition, have at least two elements, a 0 element and a 1 element. The specific Boolean algebra in Example 2 has just those two elements, the integers 0 and 1. But if you want to do a proof about Boolean algebras in general, you cannot assume that 0 and 1 are the only elements. This means that if you know that x and y are two elements in an arbitrary Boolean algebra and that $x \neq y$, that does not mean that $y = x'$.

EXAMPLE 3 Let $S = \{a, b, c\}$. Then $\wp(S)$ has eight elements:

$$\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}$$

so using those eight elements together with the operations of union, intersection, and complementation gives an eight-element Boolean algebra. The empty set is the 0 element, and $\{1, 2, 3\}$ is the 1 element. We could show the tables for union and intersection, but let's just look at complementation. Remember that $x \cup x' = \{a, b, c\}$.

\emptyset	$\{a, b, c\}$
$\{a\}$	$\{b, c\}$
$\{b\}$	$\{a, c\}$
$\{c\}$	$\{a, b\}$
$\{a, b\}$	$\{c\}$
$\{a, c\}$	$\{b\}$
$\{b, c\}$	$\{a\}$
$\{a, b, c\}$	\emptyset

Here $\{a, c\}$ and $\{c\}$ are two distinct elements of this Boolean algebra, but $\{a, c\}$ is not $\{c\}'$. ●

There are many other properties that hold in any Boolean algebra. We can prove these additional properties by using the properties in the definition.

EXAMPLE 4 The **idempotent** (pronounced eye'-dem-po-tent) **property**

$$x + x = x$$

holds in any Boolean algebra because

$$x + x = (x + x) \cdot 1 \quad (4b)$$

$$= (x + x) \cdot (x + x') \quad (5a)$$

$$= x + (x \cdot x') \quad (3a)$$

$$= x + 0 \quad (5b)$$

$$= x \quad (4a)$$
●

REMINDER

A Boolean algebra property may be applied only when your expression exactly matches the pattern of one side of the property.

Although ordinary arithmetic of integers has many of the properties of a Boolean algebra, the idempotent property should convince you that arithmetic is not a Boolean algebra. The property $x + x = x$ does not hold for ordinary numbers and ordinary addition unless x is zero.

In the proof of Example 4, we used property 5a to replace 1 with $x + x'$. The properties of Boolean algebra are equalities, and either side of an equal sign can be replaced with the other side. The Boolean algebra properties (rules) are like the equivalence rules in logic; to apply the rule, your situation must match exactly the pattern of the rule. For example, it is legal to replace

$$(y \cdot z) + x$$

with

$$x + (y \cdot z)$$

using property 1a, because $(y \cdot z) + x$ matches the right side of 1a where y is the Boolean algebra element $y \cdot z$, and $x + (y \cdot z)$ matches the left side of 1a under the same interpretation of y . We cannot say

$$x + (y \cdot z) = (x \cdot y) + (x \cdot z)$$

using either property 3a or 3b because we have mixed up the two properties. And, strictly speaking, we cannot replace

$$(y \cdot z) + x$$

with

$$(y + x) \cdot (z + x)$$

and claim that we are using property 3a because in property 3a the addition is to the left of the multiplication. We must reason as follows:

$$\begin{aligned} (y \cdot z) + x &= x + (y \cdot z) && (1a) \\ &= (x + y) \cdot (x + z) && (3a) \\ &= (y + x) \cdot (z + x) && (1a \text{ twice}) \end{aligned}$$

However, we will sometimes make implicit use of the associative property and write

$$x + y + z$$

with no parentheses.

Each property in the definition of a Boolean algebra has its dual as part of the definition, where the **dual** is obtained by interchanging $+$ and \cdot , and 1 and 0. For example, $x + 0 = x$ and $x \cdot 1 = x$ are duals of each other. Therefore, every time a new property P about Boolean algebras is proved, each step in that proof can be replaced by the dual of that step. The result is a proof of the dual of P . Thus, once we have proved P , we know that the dual of P also holds. It's a two-for-one deal!

EXAMPLE 5

The dual of the idempotent property (Example 4), $x \cdot x = x$, is true in any Boolean algebra. ■

PRACTICE 2

- What does the idempotent property of Example 4 become in the context of propositional logic?
- What does it become in the context of set theory? ■

Once a property about Boolean algebra is proved, we can use it to prove new properties.

PRACTICE 3

- Prove that the **universal bound property** $x + 1 = 1$ holds in any Boolean algebra. Give a reason for each step.
- What is the dual property? ■

More properties of Boolean algebras appear in the exercises at the end of this section. The most important of these properties are double negation and De Morgan's laws:

$$\begin{array}{ll} (x')' = x & \text{(double negation—Exercise 7)} \\ (x + y)' = x' \cdot y' & (x \cdot y)' = x' + y' \quad \text{(De Morgan's laws—Exercise 8)} \end{array}$$

Table 8.1 suggests hints that may help when trying to prove a Boolean algebra property of the form

$$\text{some expression} = \text{some other expression}$$

TABLE 8.1**Hints for Proving Boolean Algebra Equalities**

Usually the best approach is to start with the more complicated expression and try to show that it reduces to the simpler expression.

Think of adding some form of 0 (like $x \cdot x'$) or multiplying by some form of 1 (like $x + x'$).

Remember property 3a, the distributive property of addition over multiplication—it is easy to forget because it doesn't look like arithmetic.

Remember the idempotent property $x + x = x$ and its dual $x \cdot x = x$.

Remember the universal bound property $x + 1 = 1$ and its dual $x \cdot 0 = 0$.

EXAMPLE 6

Prove that $x' \cdot y = x' \cdot y + x' \cdot y \cdot z$ in any Boolean algebra.

Following the suggestion in Table 8.1, we start with the more complicated expression, $x' \cdot y + x' \cdot y \cdot z$. There is a “common factor” of $x' \cdot y$, so we should

use a distributive property, although it's not clear at this point how the z is going to disappear.

$$\begin{aligned}
 x' \cdot y + x' \cdot y \cdot z &= x' \cdot y \cdot 1 + x' \cdot y \cdot z && (4b) \\
 &= x' \cdot y \cdot (1 + z) && (3b) \\
 &= x' \cdot y \cdot (z + 1) && (1a) \\
 &= x' \cdot y \cdot (1) && (\text{universal bound—and that's} \\
 & && \text{how } z \text{ disappears)} \\
 &= x' \cdot y && (4b)
 \end{aligned}$$

For x an element of a Boolean algebra B , the element x' is called the **complement** of x (picking up the terminology from set theory). The complement of x satisfies

$$x + x' = 1 \quad \text{and} \quad x \cdot x' = 0$$

Indeed, x' is the unique element with these two properties. To prove it, suppose x_1 is an element of B with these same properties,

$$x + x_1 = 1 \quad \text{and} \quad x \cdot x_1 = 0$$

Then

$$\begin{aligned}
 x_1 &= x_1 \cdot 1 && (4b) \\
 &= x_1 \cdot (x + x') && (x + x' = 1) \\
 &= (x_1 \cdot x) + (x_1 \cdot x') && (3b) \\
 &= (x \cdot x_1) + (x' \cdot x_1) && (1b) \\
 &= 0 + (x' \cdot x_1) && (x \cdot x_1 = 0) \\
 &= (x \cdot x') + (x' \cdot x_1) && (x \cdot x' = 0) \\
 &= (x' \cdot x) + (x' \cdot x_1) && (1b) \\
 &= x' \cdot (x + x_1) && (3b) \\
 &= x' \cdot 1 && (x + x_1 = 1) \\
 &= x' && (4b)
 \end{aligned}$$

Thus $x_1 = x'$, and x' is unique. (Uniqueness in the context of propositional logic means that the truth table is unique, but there can be many different wffs associated with any particular truth table.)

The following theorem summarizes our observations.

REMINDER

To prove that something is unique, assume that there are two of them and prove that they must be the same.

REMINDER

If it walks like a duck and it quacks like a duck, it must be a duck. If it has the two properties of the complement, then by uniqueness it must be the complement.

THEOREM ON THE UNIQUENESS OF COMPLEMENTS

For any x in a Boolean algebra, if an element x_1 exists such that

$$x + x_1 = 1 \quad \text{and} \quad x \cdot x_1 = 0$$

then $x_1 = x'$.

PRACTICE 4 Prove that $0' = 1$ and $1' = 0$. (*Hint:* $1' = 0$ will follow by duality from $0' = 1$. To show $0' = 1$, use the theorem on the uniqueness of complements.) ■

There are many ways to define a Boolean algebra. Indeed, in our definition of Boolean algebra, we could have omitted the associative properties, since these can be derived from the remaining properties of the definition. It is much more convenient, however, to include them.

Isomorphic Boolean Algebras

What Is Isomorphism?

Two instances of a structure are **isomorphic** if there is a bijection (called an **isomorphism**) that maps the elements of one instance onto the elements of the other so that important properties are preserved. (Isomorphic graphs were discussed in Section 6.1.) If two instances of a structure are isomorphic, each is a mirror image of the other, with the elements simply relabeled. The two instances are essentially the same. Therefore, we can use the idea of isomorphism to classify instances of a structure, lumping together those that are isomorphic.

EXAMPLE 7 Consider the two partially ordered sets

$$S_1 = \{1, 2, 3, 5, 6, 10, 15, 30\}; x \rho y \leftrightarrow x \text{ divides } y$$

$$S_2 = \wp(\{1, 2, 3\}); A \sigma B \leftrightarrow A \subseteq B$$

The Hasse diagram of each partially ordered set appears in Figure 8.1. These two diagrams certainly appear to be mirror images of each other; just by looking at the diagrams, an obvious relabeling of the nodes, as shown in Figure 8.2, suggests itself. The important properties of a partially ordered set are which elements are related, and the Hasse diagram displays this information. For example, Figure 8.1a shows that 1, because of its position at the bottom of the graph, is related to every element in S_1 . Is this property preserved under the relabeling of Figure 8.2? Yes, because \emptyset is the image of 1 under that relabeling, and \emptyset is related to every element in S_2 . Similarly, all the other “is related to” properties are preserved under the relabeling.

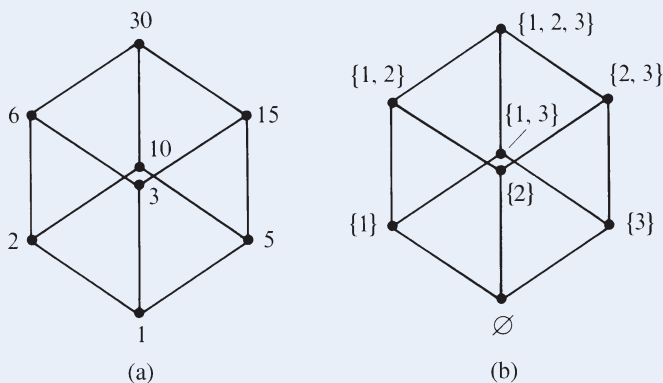


Figure 8.1

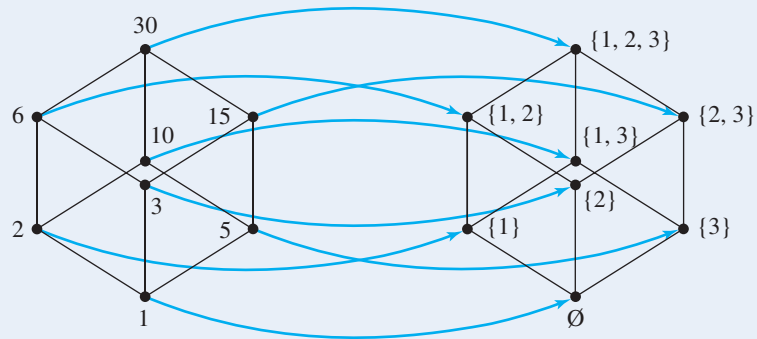


Figure 8.2

More formally, the relabeling is accomplished by the following bijection f from the set of nodes in Figure 8.1a onto the set of nodes in Figure 8.1b.

$$\begin{array}{llll} f(1) = \emptyset & f(2) = \{1\} & f(3) = \{2\} & f(5) = \{3\} \\ f(6) = \{1, 2\} & f(10) = \{1, 3\} & f(15) = \{2, 3\} & f(30) = \{1, 2, 3\} \end{array}$$

The bijection f is an isomorphism from poset (S_1, ρ) to poset (S_2, σ) . Because this isomorphism exists, the posets (S_1, ρ) and (S_2, σ) are isomorphic. (The function f^{-1} would be an isomorphism from (S_2, σ) to (S_1, ρ) .)

In Example 7 it was relatively easy to find an isomorphism because of the visual representation that captured the important properties (which elements are related). Suppose that instead of a partially ordered set, we have a structure (like a Boolean algebra) where binary or unary operations are defined on a set. Then the important properties pertain to how these operations act. An isomorphism must preserve the effects of performing these operations. Each instance of two such structures that are isomorphic must be the mirror image of the other in the sense that “operate and then map” must equal “map and then operate.”

Figure 8.3 illustrates this general idea for a binary operation. In Figure 8.3a, the binary operation is performed on a and b , resulting in c , then c is mapped to d . In Figure 8.3b, a and b are mapped to e and f , on which a binary operation is performed, resulting in the same element d as before. Remember,

$$\text{operate and map} = \text{map and operate}$$

Still another view of this little equation appears in the commutative diagram of Figure 8.4.

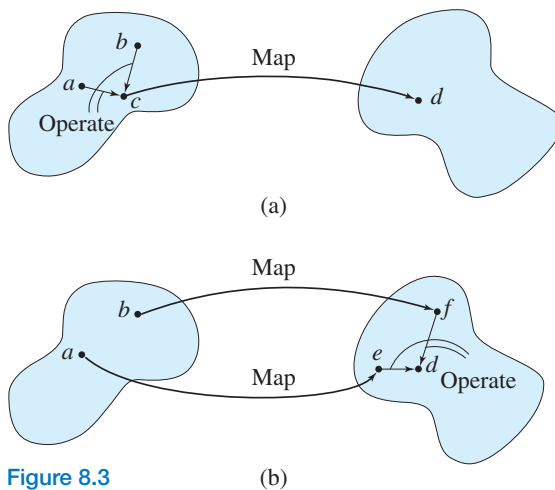


Figure 8.3

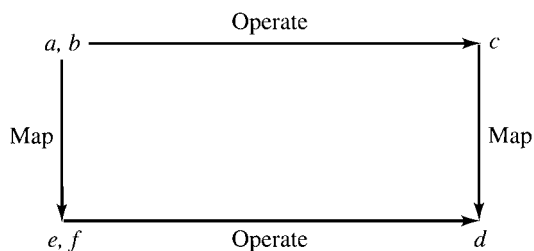


Figure 8.4

Isomorphism as Applied to Boolean Algebra

Now let's determine specifically what is involved when two instances of a Boolean algebra are isomorphic. Suppose we have two Boolean algebras, $[B, +, \cdot, ', 0, 1]$ and $[b, \&, *, ", \phi, \lambda]$. This notation means that, for example, if x is in B , x' is the result of performing on x the unary operation defined in B , and if z is an element of b , z'' is the result of performing on z the unary operation defined in b . How would we define isomorphism between these two Boolean algebras? First, we would need a bijection f from B onto b . Then f must preserve in b the effects of the various operations in B . There are three operations, so we use three equations to express these preservations. To preserve the operation $+$, we want to be able to operate using $+$ on two elements in B and then map the result to b , or to map the two elements to b and operate using the corresponding operation $\&$ on the results there. (Think "operate and map = map and operate.") Thus, for x and y in B , we require

$$f(x + y) = f(x) \& f(y)$$

PRACTICE 5

- Write the equation requiring f to preserve the effect of the binary operation \cdot .
- Write the equation requiring f to preserve the effect of the unary operation $'$.

Here is the definition of an isomorphism for Boolean algebras.

DEFINITION ISOMORPHISM FOR BOOLEAN ALGEBRAS

Let $[B, +, \cdot, ', 0, 1]$ and $[b, \&, *, ', \phi, \chi]$ be Boolean algebras. A function $B \rightarrow b$ is an **isomorphism** from $[B, +, \cdot, ', 0, 1]$ to $[b, \&, *, ', \phi, \chi]$ if

1. f is a bijection
2. $f(x + y) = f(x) \& f(y)$
3. $f(x \cdot y) = f(x) * f(y)$
4. $f(x') = (f(x))'$

PRACTICE 6 Illustrate properties 2, 3, and 4 in the definition by commutative diagrams.

We already know (it was one of our original inspirations) that for any set S , $\wp(S)$ under the operations of union, intersection, and complementation constitutes a Boolean algebra. Example 3 talks about this type of Boolean algebra where $S = \{a, b, c\}$. If we pick $S = \{1, 2\}$, then the elements of $\wp(S)$ are $\emptyset, \{1\}, \{2\}$, and $\{1, 2\}$. The operations are given by the following tables.

\cup	\emptyset	$\{1, 2\}$	$\{1\}$	$\{2\}$
\emptyset	\emptyset	$\{1, 2\}$	$\{1\}$	$\{2\}$
$\{1, 2\}$	$\{1, 2\}$	$\{1, 2\}$	$\{1, 2\}$	$\{1, 2\}$
$\{1\}$	$\{1\}$	$\{1, 2\}$	$\{1\}$	$\{1, 2\}$
$\{2\}$	$\{2\}$	$\{1, 2\}$	$\{1, 2\}$	$\{2\}$

\cap	\emptyset	$\{1, 2\}$	$\{1\}$	$\{2\}$	$'$	
\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	$\{1, 2\}$
$\{1, 2\}$	\emptyset	$\{1, 2\}$	$\{1\}$	$\{2\}$	$\{1, 2\}$	\emptyset
$\{1\}$	\emptyset	$\{1\}$	$\{1\}$	\emptyset	$\{1\}$	$\{2\}$
$\{2\}$	\emptyset	$\{2\}$	\emptyset	$\{2\}$	$\{2\}$	$\{1\}$

A Boolean algebra can be defined on the set $B = \{0, 1, a, a'\}$ where the operations of $+$, \cdot , and $'$ are defined by the following tables (see Exercise 1).

$+$	0	1	a	a'	\cdot	0	1	a	a'	$'$	0	1
0	0	1	a	a'	0	0	0	0	0	0	0	1
1	1	1	1	1	1	0	1	a	a'	1	1	0
a	a	1	a	1	a	0	a	a	0	a	a	a'
a'	a'	1	1	a'	a'	0	a'	0	a'	a'	a'	a

We claim that the mapping $f: B \rightarrow \wp(S)$ given by

$$\begin{aligned} f(0) &= \emptyset \\ f(1) &= \{1, 2\} \\ f(a) &= \{1\} \\ f(a') &= \{2\} \end{aligned}$$

is an isomorphism. Certainly it is a bijection. For $x, y \in B$, we can verify each of the equations

$$\begin{aligned} f(x + y) &= f(x) \cup f(y) \\ f(x \cdot y) &= f(x) \cap f(y) \\ f(x') &= (f(x))' \end{aligned}$$

by examining all possible cases. Thus, for example,

$$f(a \cdot 1) = f(a) = \{1\} = \{1\} \cap \{1, 2\} = f(a) \cap f(1)$$

PRACTICE 7 | Verify the following equations.

- $f(0 + a) = f(0) \cup f(a)$
- $f(a + a') = f(a) \cup f(a')$
- $f(a \cdot a') = f(a) \cap f(a')$
- $f(1') = (f(1))'$

The remaining cases also hold. Even without testing all cases, it is pretty clear here that f is going to work because it merely relabels the entries in the tables for B so that they resemble the tables for $\wp(S)$. In general, however, it may not be so easy to decide whether a given f is an isomorphism between two instances of a structure. Even harder to answer is the question of whether two given instances of a structure are isomorphic; we must either think up a function that works or show that no such function exists. One case where no such function exists is when the sets involved are not the same size; we cannot have a four-element Boolean algebra isomorphic to an eight-element Boolean algebra.

We just showed that a particular four-element Boolean algebra is isomorphic to $\wp(\{1, 2\})$. It turns out that any finite Boolean algebra is isomorphic to the Boolean algebra of a power set. Although we state this as a theorem, we will not prove it.

THEOREM ON FINITE BOOLEAN ALGEBRAS

Let B be any Boolean algebra with n elements. Then $n = 2^m$ for some m , and B is isomorphic to $\wp(\{1, 2, \dots, m\})$.

This theorem gives us two pieces of information. The number of elements in a finite Boolean algebra must be a power of 2. Also we learn that finite Boolean algebras that are power sets are—in our lumping together of isomorphic things—

really the only kinds of finite Boolean algebras. In a sense we have come full circle. We defined a Boolean algebra to represent many kinds of situations; now we find that (for the finite case) the situations, except for the labels of objects, are the same anyway!

SECTION 8.1 REVIEW

TECHNIQUES

- Decide whether something is a Boolean algebra.
- **W** Prove properties about Boolean algebras.
- Write the equation meaning that a function f preserves an operation from one instance of a structure to another, and verify or disprove such an equation.

MAIN IDEAS

- Mathematical structures serve as models or abstractions of common properties found in diverse situations.
- If there is an isomorphism (a bijection that preserves properties) from A to B , where A and B are instances of a structure, then except for labels, A and B are the same.
- All finite Boolean algebras are isomorphic to Boolean algebras that are power sets.

EXERCISES 8.1

1. Let $B = \{0, 1, a, a'\}$, and let $+$ and \cdot be binary operations on B . The unary operation $'$ is defined by the table

0	1
1	0
a	a'
a'	a

Suppose you know that $[B, +, \cdot, ', 0, 1]$ is a Boolean algebra. Making use of the properties that must hold in any Boolean algebra, fill in the following tables defining the binary operations $+$ and \cdot :

$+$	0	1	a	a'
0				
1				
a				
a'				

\cdot	0	1	a	a'
0				
1				
a				
a'				

2. a. What does the universal bound property (Practice 3) become in the context of propositional logic?
 b. What does it become in the context of set theory?
3. Define two binary operations $+$ and \cdot on the set \mathbb{Z} of integers by $x + y = \max(x, y)$ and $x \cdot y = \min(x, y)$.
 - a. Show that the commutative, associative, and distributive properties of a Boolean algebra hold for these two operations on \mathbb{Z} .
 - b. Show that no matter what element of \mathbb{Z} is chosen to be 0, the property $x + 0 = x$ of a Boolean algebra fails to hold.

4. Let $M_2(\mathbb{Z})$ denote the set of 2×2 matrices with integer entries, and let $+$ denote matrix addition and \cdot denote matrix multiplication. Given

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \text{ then } \mathbf{A}' = \begin{bmatrix} -a & -b \\ -c & -d \end{bmatrix}.$$

Using $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ as the 0 element and the 1 element, respectively, either prove that

$[M_2(\mathbb{Z}), +, \cdot, ', 0, 1]$ is a Boolean algebra or give a reason why it is not.

5. Let S be the set $\{0, 1\}$. Then S^2 is the set of all ordered pairs of 0s and 1s; $S^2 = \{(0, 0), (0, 1), (1, 0), (1, 1)\}$. Consider the set B of all functions mapping S^2 to S . For example, one such function, $f(x, y)$, is given by

$$f(0, 0) = 0$$

$$f(0, 1) = 1$$

$$f(1, 0) = 1$$

$$f(1, 1) = 1$$

- a. How many elements are in B ?
 b. For f_1 and f_2 members of B and $(x, y) \in S^2$, define

$$(f_1 + f_2)(x, y) = \max(f_1(x, y), f_2(x, y))$$

$$(f_1 \cdot f_2)(x, y) = \min(f_1(x, y), f_2(x, y))$$

$$f'(x, y) = \begin{cases} 1 & \text{if } f_1(x, y) = 0 \\ 0 & \text{if } f_1(x, y) = 1 \end{cases}$$

Suppose

$$f_1(0, 0) = 1 \quad f_2(0, 0) = 1$$

$$f_1(0, 1) = 0 \quad f_2(0, 1) = 1$$

$$f_1(1, 0) = 1 \quad f_2(1, 0) = 0$$

$$f_1(1, 1) = 0 \quad f_2(1, 1) = 0$$

What are the functions $f_1 + f_2$, $f_1 \cdot f_2$, and f_1' ?

- c. Prove that $[B, +, \cdot, ', 0, 1]$ is a Boolean algebra where the functions 0 and 1 are defined by

$$0(0, 0) = 0 \quad 1(0, 0) = 1$$

$$0(0, 1) = 0 \quad 1(0, 1) = 1$$

$$0(1, 0) = 0 \quad 1(1, 0) = 1$$

$$0(1, 1) = 0 \quad 1(1, 1) = 1$$

6. Let n be a positive integer whose decomposition into prime factors has no repeated prime. Let $B = \{x \mid x \text{ is a divisor of } n\}$. For example, if $n = 21 = 3 \cdot 7$, then $B = \{1, 3, 7, 21\}$. Let the following operations be defined on B :

$$x + y = \text{lcm}(x, y) \quad x \cdot y = \text{gcd}(x, y) \quad x' = n/x$$

Then $+$ and \cdot are binary operations on B and $'$ is a unary operation on B .

a. For $n = 21$, find

(i) $3 \cdot 7$

(ii) $7 \cdot 21$

(iii) $1 + 3$

(iv) $3 + 21$

(v) $3'$

b. Prove that the commutative, associative, and distributive properties hold for both $+$ and \cdot .

c. Find the value of the “0” element and the “1” element, then prove properties 4 and 5 for both $+$ and \cdot .

d. Consider a value for n whose decomposition has repeated primes. In particular, let $n = 12 = 2 \cdot 2 \cdot 3$. Prove that, using the above definitions for $+$ and \cdot , it's not possible to define a complement for 6 in the set $\{1, 2, 3, 4, 6, 12\}$. Therefore a Boolean algebra cannot be constructed with $n = 12$ using the process described.

7. Prove the following property of Boolean algebras. Give a reason for each step. (*Hint*: Remember the uniqueness of the complement.)

$$(x')' = x \quad (\text{double negation})$$

8. Prove the following property of Boolean algebras. Give a reason for each step. (*Hint*: Remember the uniqueness of the complement.)

$$(x + y)' = x' \cdot y' \quad (x \cdot y)' = x' + y' \quad (\text{De Morgan's laws})$$

9. Prove the following properties of Boolean algebras. Give a reason for each step.

a. $x + (x \cdot y) = x$ (absorption properties)

$$x \cdot (x + y) = x$$

b. $x \cdot [y + (x \cdot z)] = (x \cdot y) + (x \cdot z)$ (modular properties)

$$x + [y \cdot (x + z)] = (x + y) \cdot (x + z)$$

c. $(x + y) \cdot (x' + y) = y$

$$(x \cdot y) + (x' \cdot y) = y$$

d. $(x + (y \cdot z))' = x' \cdot y' + x' \cdot z'$

$$(x \cdot (y + z))' = (x' + y') \cdot (x' + z')$$

e. $(x + y) \cdot (x + 1) = x + (x \cdot y) + y$

$$(x \cdot y) + (x \cdot 0) = x \cdot (x + y) \cdot y$$

10. Prove the following properties of Boolean algebras. Give a reason for each step.

a. $(x + y) + (y \cdot x') = x + y$

b. $(y + x) \cdot (z + y) + x \cdot z \cdot (z + z') = y + x \cdot z$

c. $(y' \cdot x) + x + (y + x) \cdot y' = x + (y' \cdot x)$

d. $(x + y') \cdot z = [(x' + z') \cdot (y + z')]'$

e. $(x \cdot y) + (x' \cdot z) + (x' \cdot y \cdot z') = y + (x' \cdot z)$

11. Prove the following properties of Boolean algebras. Give a reason for each step.
- $x + y' = x + (x' \cdot y + x \cdot y)'$
 - $[(x \cdot y) \cdot z] + (y \cdot z) = y \cdot z$
 - $x \cdot y + y \cdot x' = x \cdot y + y$
 - $(x + y)' \cdot z + x' \cdot z \cdot y = x' \cdot z$
 - $(x \cdot y') + (y \cdot z') + (x' \cdot z) = (x' \cdot y) + (y' \cdot z) + (x \cdot z')$
12. Prove the following properties of Boolean algebras. Give a reason for each step.
- $(x + y \cdot x)' = x'$
 - $x \cdot (z + y) + (x' + y)' = x$
 - $(x \cdot y)' + x' \cdot z + y' \cdot z = x' + y'$
 - $x \cdot y + x' = y + x' \cdot y'$
 - $x \cdot y + y \cdot z \cdot x' = y \cdot z + y \cdot x \cdot z'$
13. Prove that in any Boolean algebra, $x \cdot y' + x' \cdot y = y$ if and only if $x = 0$.
14. Prove that in any Boolean algebra, $x \cdot y' = 0$ if and only if $x \cdot y = x$.
15. A new binary operation \oplus in a Boolean algebra (*exclusive OR*) is defined by

$$x \oplus y = x \cdot y' + y \cdot x'$$

Prove that

- $x \oplus y = y \oplus x$
 - $x \oplus x = 0$
 - $0 \oplus x = x$
 - $1 \oplus x = x'$
16. Prove that for any Boolean algebra:
- If $x + y = 0$, then $x = 0$ and $y = 0$.
 - $x = y$ if and only if $x \cdot y' + y \cdot x' = 0$.
17. Prove that the 0 element in any Boolean algebra is unique; prove that the 1 element in any Boolean algebra is unique.
18. a. Find an example of a Boolean algebra with elements $x, y,$ and z for which $x + y = x + z$ but $y \neq z$. (Here is further evidence that ordinary arithmetic of integers is not a Boolean algebra.)
 b. Prove that in any Boolean algebra, if $x + y = x + z$ and $x' + y = x' + z$, then $y = z$.
19. Let (S, \leq) and (S', \leq') be two partially ordered sets. (S, \leq) is isomorphic to (S', \leq') if there is a bijection $f: S \rightarrow S'$ such that for x, y in S , $x < y \rightarrow f(x) <' f(y)$ and $f(x) <' f(y) \rightarrow x < y$.
- Show that there are exactly two nonisomorphic, partially ordered sets with two elements (use diagrams).
 - Show that there are exactly five nonisomorphic, partially ordered sets with three elements.
 - How many nonisomorphic, partially ordered sets with four elements are there?
20. Find an example of two partially ordered sets (S, \leq) and (S', \leq') and a bijection $f: S \rightarrow S'$ where, for x, y in S , $x < y \rightarrow f(x) <' f(y)$ but $f(x) <' f(y) \nrightarrow x < y$.

21. Let $S = \{0, 1\}$ and let a binary operation \cdot be defined on S by

\cdot	0	1
0	1	0
1	0	1

Let $T = \{5, 7\}$, and let a binary operation $+$ be defined on T by

$+$	5	7
5	7	5
7	5	7

Consider $[S, \cdot]$ and $[T, +]$ as mathematical structures.

- a. If a function f is an isomorphism from $[S, \cdot]$ to $[T, +]$, what two properties must f satisfy?
 - b. Define a function $f: S \rightarrow T$ and prove that it is an isomorphism from $[S, \cdot]$ to $[T, +]$.
22. Consider the four-element Boolean algebra defined in Exercise 6 with $n = 21$. Find an isomorphism from this Boolean algebra to the four-element Boolean algebra with set $\wp(\{1, 2\})$ that was defined in this section.
23. Let \mathbb{R} denote the real numbers and \mathbb{R}^+ the positive real numbers. Addition is a binary operation on \mathbb{R} , and multiplication is a binary operation on \mathbb{R}^+ . Consider $[\mathbb{R}, +]$ and $[\mathbb{R}^+, \cdot]$ as mathematical structures.
- a. Prove that the function f defined by $f(x) = 2^x$ is a bijection from \mathbb{R} to \mathbb{R}^+ .
 - b. Write the equation that an isomorphism from $[\mathbb{R}, +]$ to $[\mathbb{R}^+, \cdot]$ must satisfy.
 - c. Prove that the function f of part (a) is an isomorphism from $[\mathbb{R}, +]$ to $[\mathbb{R}^+, \cdot]$.
 - d. What is f^{-1} for this function?
 - e. Prove that f^{-1} is an isomorphism from $[\mathbb{R}^+, \cdot]$ to $[\mathbb{R}, +]$.
24. An isomorphism from the Boolean algebra with set $B = \{0, 1, a, a'\}$ to the Boolean algebra with set $\wp(\{1, 2\})$ was defined in this section. Because the two Boolean algebras are essentially the same, an operation in one can be simulated by mapping to the other, operating there, and mapping back.
- a. Use the Boolean algebra on $\wp(\{1, 2\})$ to simulate the computation $1 \cdot a'$ in the Boolean algebra on B .
 - b. Use the Boolean algebra on $\wp(\{1, 2\})$ to simulate the computation $(a)'$ in the Boolean algebra on B .
 - c. Use the Boolean algebra on B to simulate the computation $\{1\} \cup \{2\}$ in the Boolean algebra on $\wp(\{1, 2\})$.
25. Consider the set B of all functions mapping $\{0, 1\}^2$ to $\{0, 1\}$. We can define operations of $+$, \cdot , and $'$ on B by

$$(f_1 + f_2)(x, y) = \max(f_1(x, y), f_2(x, y))$$

$$(f_1 \cdot f_2)(x, y) = \min(f_1(x, y), f_2(x, y))$$

$$f_1'(x, y) = \begin{cases} 1 & \text{if } f_1(x, y) = 0 \\ 0 & \text{if } f_1(x, y) = 1 \end{cases}$$

Then $[B, +, \cdot, ', 0, 1]$ is a Boolean algebra of 16 elements (see Exercise 5). The following table assigns names to these 16 functions.

(x, y)	0	1	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
(0, 0)	0	1	1	1	1	1	1	1	0	0	0	1	0	0	0	0
(0, 1)	0	1	0	1	1	0	1	0	1	1	1	0	0	1	0	0
(1, 0)	0	1	1	0	1	0	0	1	1	1	0	0	1	0	1	0
(1, 1)	0	1	0	0	0	0	1	1	1	0	1	1	1	0	0	1

According to the theorem on finite Boolean algebras, this Boolean algebra is isomorphic to $[\wp(\{1, 2, 3, 4\}), \cup, \cap, ', \emptyset, \{1, 2, 3, 4\}]$. Complete the following definition of an isomorphism from B to $\wp(\{1, 2, 3, 4\})$.

$$\begin{aligned} 0 &\rightarrow \emptyset \\ 1 &\rightarrow \{1, 2, 3, 4\} \\ f_4 &\rightarrow \{1\} \\ f_{12} &\rightarrow \{2\} \\ f_{13} &\rightarrow \{3\} \\ f_{14} &\rightarrow \{4\} \end{aligned}$$

26. Let P , Q , and R be three statements in propositional logic with statement letters A and B . P , Q , and R define the following three functions from $\{T, F\}^2$ to $\{T, F\}$. Also shown are the contradiction 0 and the tautology 1.

A	B	P	Q	R	0	1
T	T	T	F	F	F	T
T	F	F	T	F	F	T
F	T	F	F	T	F	T
F	F	T	F	F	F	T

- Let $B = \{P, P', Q, Q', R, R', 0, 1\}$. Then $[B, \vee, \wedge, ', 0, 1]$ is a Boolean algebra. Write the 8×8 tables for the \vee and \wedge operations and the 8×1 table for the $'$ operation.
 - $[\wp(\{1, 2, 3\}), \cup, \cap, ', \emptyset, \{1, 2, 3\}]$ is a Boolean algebra. Write the 8×8 tables for the \cup and \cap operations and the 8×1 table for the $'$ operation.
 - Find an isomorphism from the Boolean algebra of part (a) to the Boolean algebra of part (b).
27. Suppose that $[B, +, \cdot, ', 0, 1]$ and $[b, \&, *, ', \phi, \chi]$ are isomorphic Boolean algebras and that f is an isomorphism from B to b .
- Prove that $f(0) = \phi$.
 - Prove that $f(1) = \chi$.
28. According to the theorem on finite Boolean algebras, which we did not prove, any finite Boolean algebra must have 2^m elements for some m . Prove the weaker statement that no Boolean algebra can have an odd number of elements. (Note that in the definition of a Boolean algebra, 0 and 1 are distinct elements of B , so B has at least two elements. Arrange the remaining elements of B so that each element is paired with its complement.)

29. A Boolean algebra may also be defined as a partially ordered set with certain additional properties. Let (B, \leq) be a partially ordered set. For any $x, y \in B$, we define the *least upper bound* of x and y as an element z such that $x \leq z, y \leq z$, and if there is any element z^* with $x \leq z^*$ and $y \leq z^*$, then $z \leq z^*$. The *greatest lower bound* of x and y is an element w such that $w \leq x, w \leq y$, and if there is any element w^* with $w^* \leq x$ and $w^* \leq y$, then $w^* \leq w$. A *lattice* is a partially ordered set in which every two elements x and y have a least upper bound, denoted by $x + y$, and a greatest lower bound, denoted by $x \cdot y$.

a. Prove that in any lattice

- (i) $x \cdot y = x$ if and only if $x \leq y$
- (ii) $x + y = y$ if and only if $x \leq y$

b. Prove that in any lattice

- (i) $x + y = y + x$
- (ii) $x \cdot y = y \cdot x$
- (iii) $(x + y) + z = x + (y + z)$
- (iv) $(x \cdot y) \cdot z = x \cdot (y \cdot z)$

c. A lattice L is *complemented* if there exists a least element 0 and a greatest element 1 , and for every $x \in L$ there exists $x' \in L$ such that $x + x' = 1$ and $x \cdot x' = 0$. Prove that in a complemented lattice L ,

$$x + 0 = x \quad \text{and} \quad x \cdot 1 = x$$

for all $x \in L$.

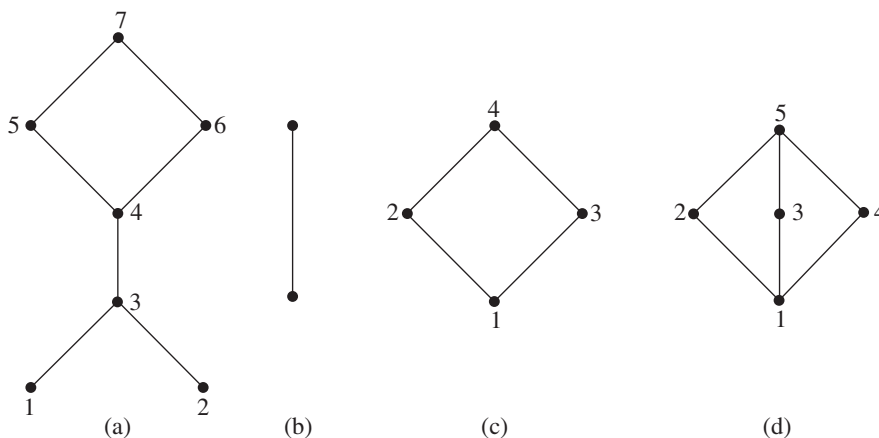
d. A lattice L is *distributive* if

$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

and

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$

for every $x, y, z \in L$. By parts (b) and (c), a complemented, distributive lattice is a Boolean algebra. Which of the following Hasse diagrams of partially ordered sets do not represent Boolean algebras? Why? (*Hint*: In a Boolean algebra, the complement of an element is unique.)



30. a. Let n be a positive integer and consider B to be the set of all positive integer divisors of n . Prove that (B, \leq) is a partially ordered set where $x \leq y$ means $x|y$.

In the terminology of Exercise 29, the least upper bound of x and y is the least common multiple of x and y , and the greatest lower bound is the greatest common divisor. (B, \leq) is a distributive lattice.

- b. Prove that for $n = 6$, (B, \leq) is a Boolean algebra. (*Hint*: 1 is the least element and 6 is the greatest element).
- c. For $n = 8$, (B, \leq) is not a Boolean algebra.
- Show that this is true by using the definition of a Boolean algebra.
 - Show that this is true by using Exercise 6.

SECTION 8.2 LOGIC NETWORKS

Combinational Networks

Basic Logic Elements

In 1938 the American mathematician Claude Shannon perceived the parallel between propositional logic and circuit logic and realized that Boolean algebra could play a part in systematizing this new realm of electronics.

Let us imagine that the electrical voltages carried along wires fall into one of two ranges, high or low, which we represent by 1 and 0, respectively. Voltage fluctuations within these ranges are ignored, so we are forcing a discrete, indeed binary, mask on an analog phenomenon. We also suppose that switches can be wired so that a signal of 1 causes the switch to be closed and a signal of 0 causes the switch to be open (Figure 8.5). Now we combine two such switches, controlled by lines x_1 and x_2 , in parallel. If either or both lines carry a 1 value, one or both of the switches will be closed, and the output line will have a value of 1. However, values of $x_1 = 0$ and $x_2 = 0$ will cause both switches to be open and thus break the circuit, so that the voltage level on the output line will be 0. Figure 8.6 illustrates the various cases.

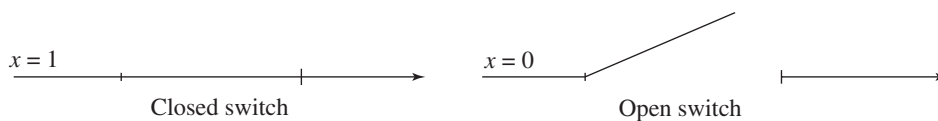


Figure 8.5

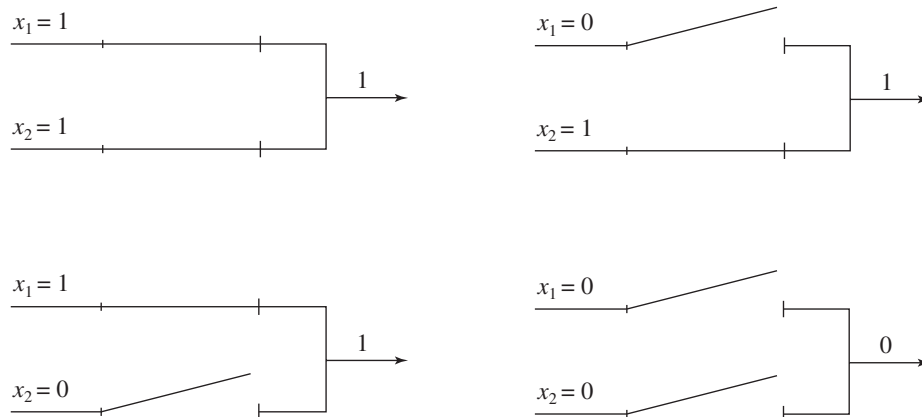


Figure 8.6

x_1	x_2	Output
1	1	1
1	0	1
0	1	1
0	0	0

Table 8.2 summarizes the behavior of the circuit. Substituting T for 1 and F for 0 in the table results in the truth table for the logical connective of disjunction. Disjunction is an example of the Boolean algebra operation $+$ in the realm of propositional logic. Thus we may think of the circuit more abstractly as an electronic device that performs the Boolean operation $+$. Similarly, conjunction and negation are examples of the Boolean algebra operations \cdot and $'$, respectively, in the realm of propositional logic. Other devices perform these Boolean operations. For example, switches connected in series would serve to implement the \cdot operation; both switches must be closed ($x_1 = 1$ and $x_2 = 1$) in order to have an output of 1. However, we'll ignore the details of implementing the devices; suffice it to say that technology has progressed from mechanical switches through vacuum tubes and then transistors to integrated circuits, and now even bacteria and DNA. We will simply represent these devices by their standard symbols.

The **OR gate**, Figure 8.7a, behaves like the Boolean operation $+$. The **AND gate**, Figure 8.7b, represents the Boolean operation \cdot . Figure 8.7c shows an **inverter**, corresponding to the unary Boolean operation $'$. Because of the associativity property for $+$ and \cdot , the OR and AND gates can have more than two inputs.

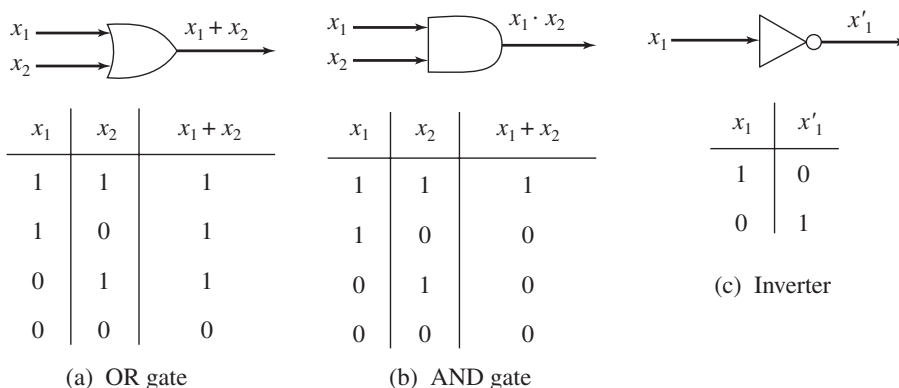


Figure 8.7

Boolean Expressions

DEFINITION **BOOLEAN EXPRESSION**

A **Boolean expression** in n variables, x_1, x_2, \dots, x_n , is any finite string of symbols formed by applying the following rules:

1. x_1, x_2, \dots, x_n , are Boolean expressions.
2. If P and Q are Boolean expressions, so are $(P + Q)$, $(P \cdot Q)$, and (P') .

(The definition of a Boolean expression is another example of a recursive definition; rule 1 is the basis step and rule 2 the inductive step.) When there is no chance of confusion, we can omit the parentheses introduced by rule 2. In addition, we define \cdot to take precedence over $+$ and $'$ to take precedence over $+$ or \cdot ; so $x_1 + x_2 \cdot x_3$ stands for $x_1 + (x_2 \cdot x_3)$ and $x_1 + x_2'$ stands for $x_1 + (x_2')$; this convention also allows us to remove some parentheses. Finally, we will generally omit the symbol \cdot and use juxtaposition, so $x_1 \cdot x_2$ is written x_1x_2 .

EXAMPLE 8 x_3 , $(x_1 + x_2)'x_3$, $(x_1x_3 + x_4')x_2$, and $(x_1'x_2)'x_1$ are all Boolean expressions. ●

Truth Functions

● **DEFINITION TRUTH FUNCTION**

A **truth function** is a function f such that $f: \{0, 1\}^n \rightarrow \{0, 1\}$ for some integer $n \geq 1$.

The notation $\{0, 1\}^n$ denotes the set of all n -tuples of 0s and 1s. A truth function thus associates a value of 0 or 1 with each such n -tuple.

EXAMPLE 9 The truth table for the Boolean operation $+$ describes a truth function f with $n = 2$. The domain of f is $\{(1, 1), (1, 0), (0, 1), (0, 0)\}$, and $f(1, 1) = 1, f(1, 0) = 1, f(0, 1) = 1$, and $f(0, 0) = 0$. Similarly, the Boolean operation \cdot describes a different truth function with $n = 2$, and the Boolean operation $'$ describes a truth function for $n = 1$. ●

PRACTICE 8

- If we are writing a truth function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ in tabular form (like a truth table), how many rows will the table have?
- How many different truth functions are there that take $\{0, 1\}^2 \rightarrow \{0, 1\}$?
- How many different truth functions are there that take $\{0, 1\}^n \rightarrow \{0, 1\}$? ■

Any Boolean expression defines a unique truth function, just as do the simple Boolean expressions $x_1 + x_2$, x_1x_2 , and x_1' .

EXAMPLE 10 The Boolean expression $x_1x_2' + x_3$ defines the truth function given in Table 8.3. (This is just like doing the truth tables of Section 1.1.)

x_1	x_2	x_3	$x_1x_2' + x_3$
1	1	1	1
1	1	0	0
1	0	1	1
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

Networks and Expressions

It's time to see how these ideas of logic gates, Boolean expressions, and truth functions are related. By combining AND gates, OR gates, and inverters, we can construct a logic network representing a given Boolean expression that produces the same truth function as that expression.

EXAMPLE 11

The logic network for the Boolean expression $x_1x_2' + x_3$ is shown in Figure 8.8.

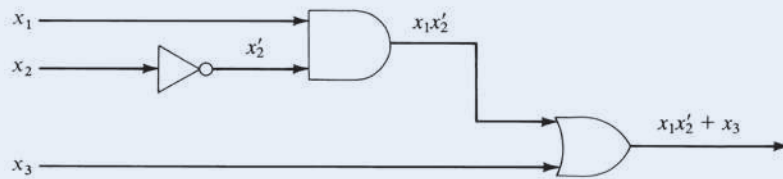


Figure 8.8

PRACTICE 9

Design the logic network for the following Boolean expressions.

- $x_1 + x_2'$
- $x_1(x_2 + x_3)'$

Conversely, if we have a logic network, we can write a Boolean expression with the same truth function.

EXAMPLE 12

A Boolean expression for the logic network in Figure 8.9 is

$$(x_1x_2 + x_3)' + x_3$$

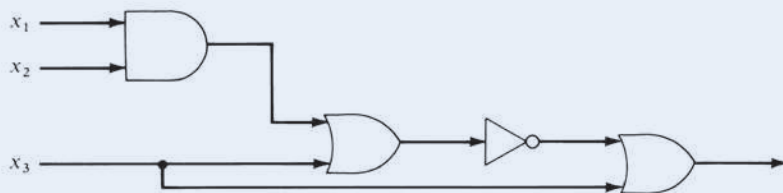


Figure 8.9

PRACTICE 10

- Write a Boolean expression for the logic network in Figure 8.10.

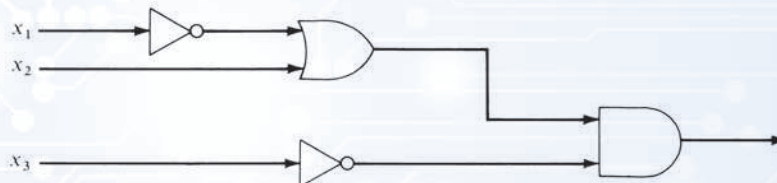


Figure 8.10

- Write the truth function (in table form) for the network (and expression) of part (a).

Logic networks constructed of AND gates, OR gates, and inverters are also called **combinational networks**. They have several features that we should note. First, input or output lines are not tied together except by passing through gates. Lines can be split, however, to serve as input to more than one device. There are no loops where the output of an element is part of the input to that same element. Finally, the output of a network is an instantaneous function of the input; there are no delay elements that capture and remember input signals. Notice also that the picture of any network is, in effect, a directed graph.

Canonical Form

Here is the situation so far (arrows indicate a procedure that we can carry out):

$$\text{truth function} \leftarrow \text{Boolean expression} \leftrightarrow \text{logic network}$$

We can write a unique truth function from either a network or an expression. Given an expression, we can find a network with the same truth function, and conversely. The last part of the puzzle concerns how to get from an arbitrary truth function to an expression (and hence a network) having that truth function. An algorithm to solve this problem is explained in the next example.

EXAMPLE 13

Suppose we want to find a Boolean expression for the truth function f of Table 8.4. There are four rows in the table (rows 1, 3, 4, and 7) for which f is 1. The basic form of our expression will be a sum of four terms

$$() + () + () + ()$$

such that the first term has the value 1 for the input values of row 1 and for no others, the second term has the value 1 for the input values of row 3 and for no others, and so on. Thus, the entire expression has the value 1 for these inputs and for no others—precisely what we want. (Other inputs cause each term in the sum, and hence the sum itself, to be 0.)

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	1
1	1	0	0
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	1
0	0	0	0

Each term in the sum will be a product of the form $\alpha\beta\gamma$ where α is either x_1 or x'_1 , β is either x_2 or x'_2 , and γ is either x_3 or x'_3 . If the input value of x_i , $i = 1, 2, 3$, in the row we are working on is 1, then x_i itself is used; if the input value of x_i in the row we are working on is 0, then x'_i is used. These values will force $\alpha\beta\gamma$ to be 1 for that row and 0 for all other rows. Thus, we have

row 1: $x_1x_2x_3$

row 3: $x_1x'_2x_3$

row 4: $x_1x_2x'_3$

row 7: $x'_1x'_2x_3$

The final expression is

$$(x_1x_2x_3) + (x_1x'_2x_3) + (x_1x_2x'_3) + (x'_1x'_2x_3)$$

The procedure described in Example 13 always leads to an expression that is a sum of products, called the **canonical sum-of-products form**, or the **disjunctive normal form**, for the given truth function. The only case not covered by this procedure is when the function has a value of 0 everywhere. Then we use an expression such as

$$x_1x'_1$$

which is also a sum (one term) of products. Therefore, we can find a sum-of-products expression to represent any truth function. A pseudocode description of the algorithm is given in the accompanying box. For this algorithm, the input is a truth table representing a truth function on n variables x_1, x_2, \dots, x_n ; the output is a Boolean expression in disjunctive normal form with the same truth function.

ALGORITHM *SUM-OF-PRODUCTS*

Sum-Of-Products (truth table; integer n)
 //the truth table represents a truth function with n arguments;
 //result is the canonical sum-of-products expression for this truth function

Local variables:

sum //sum-of-products expression
product //single term in sum, a product
i //index for the columns of the table
row //index for the rows of the table

sum = empty

```

for row = 1 to  $2^n$  do
  if truth value for row is 1 then
    initialize product;
    for  $i = 1$  to  $n$  do
      if  $x_i = 1$  then
        put  $x_i$  in product
      else
        put  $x'_i$  in product
      end if
    end for
     $sum = sum + product$ 
  end if
end for
if sum is empty then
   $sum = x_1x'_1$ 
end if
write ("The canonical sum-of-products expression for this truth function is", sum)
end Sum-Of-Products

```

Because any expression has a corresponding network, any truth function has a logic network representation. Furthermore, the AND gate, OR gate, and inverter are the only devices needed to construct the network. Thus, we can build a network for any truth function with only three kinds of parts—and lots of wire! Later we will see that it is necessary to stock only one kind of part.

Given a truth function, the canonical sum-of-products form just described is one expression that has this truth function, but it is not the only possible one. A method for obtaining a different expression for any truth function is given in Exercise 25 at the end of this section.

EXAMPLE 14

The network for the canonical sum-of-products form of Example 13 is shown in Figure 8.11. We have drawn the inputs to each AND gate separately because it looks neater, but actually a single x_1 , x_2 , or x_3 input can be split as needed.

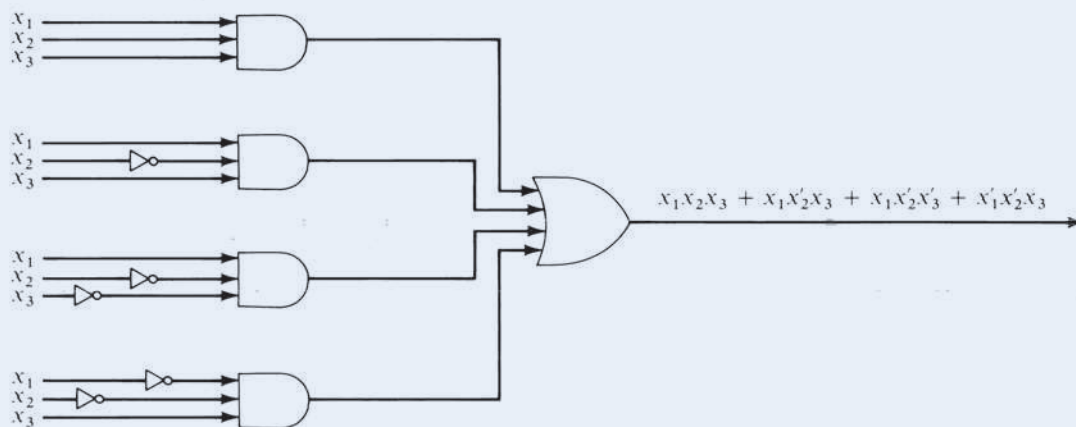


Figure 8.11

PRACTICE 11

- a. Find the canonical sum-of-products form for the truth function of Table 8.5.
 b. Draw the network for the expression of part (a).

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	1
1	1	0	0
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	1
0	0	0	1

Minimization

As already noted, a given truth function may be represented by more than one Boolean expression and hence by more than one logic network composed of AND gates, OR gates, and inverters.

EXAMPLE 15

The Boolean expression

$$x_1x_3 + x_2'$$

has the truth function of Table 8.5. The logic network corresponding to this expression is given by Figure 8.12. Compare this with your network in Practice 11(b)!

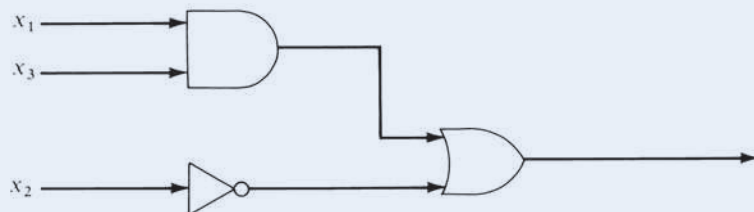


Figure 8.12

DEFINITION EQUIVALENT BOOLEAN EXPRESSIONS

Two Boolean expressions are **equivalent** if they have the same truth functions.

We know that

$$x_1x_2x_3 + x_1x_2'x_3 + x_1x_2'x_3' + x_1'x_2'x_3 + x_1'x_2'x_3'$$

and

$$x_1x_3 + x_2'$$

for example, are equivalent Boolean expressions.

Clearly, equivalence of Boolean expressions is an equivalence relation on the set of all Boolean expressions in n variables. Each equivalence class is associated with a distinct truth function. Given a truth function, algorithm *Sum-Of-Products* produces one particular member of the class associated with that function, namely, the canonical sum-of-products form. However, if we are trying to design the logic network for that function, we want to find a member of the class that is as simple as possible. We would rather build the network of Figure 8.12 than the one for Practice 11(b).

How can we reduce a Boolean expression to an equivalent, simpler expression? We can use the properties of a Boolean algebra because they express the equivalence of Boolean expressions. If P is a Boolean expression containing the subexpression $(x_1 + x_2)(x_1 + x_3)$, for example, and Q is the expression obtained from P by replacing $(x_1 + x_2)(x_1 + x_3)$ with the equivalent expression $x_1 + (x_2x_3)$, then P and Q are equivalent and Q is simpler than P .

EXAMPLE 16

Using the properties of Boolean algebra, we can reduce

$$x_1x_2x_3 + x_1x_2'x_3 + x_1x_2'x_3' + x_1'x_2'x_3 + x_1'x_2'x_3'$$

to

$$x_1x_3 + x_2'$$

as follows:

$$\begin{aligned} & x_1x_2x_3 + x_1x_2'x_3 + x_1x_2'x_3' + x_1'x_2'x_3 + x_1'x_2'x_3' \\ &= x_1x_2x_3 + x_1x_2'x_3 + x_1x_2'x_3' + x_1'x_2'x_3 + x_1'x_2'x_3' && \text{(idempotent)} \\ &= x_1x_3x_2 + x_1x_3x_2' + x_1x_2'x_3 + x_1x_2'x_3' + x_1'x_2'x_3 + x_1'x_2'x_3' && \text{(1b)} \\ &= x_1x_3(x_2 + x_2') + x_1x_2'(x_3 + x_3') + x_1'x_2'(x_3 + x_3') && \text{(3b)} \\ &= x_1x_3 \cdot 1 + x_1x_2' \cdot 1 + x_1'x_2' \cdot 1 && \text{(5a)} \\ &= x_1x_3 + x_1x_2' + x_1'x_2' && \text{(4b)} \\ &= x_1x_3 + x_2'x_1 + x_2'x_1' && \text{(1b)} \\ &= x_1x_3 + x_2'(x_1 + x_1') && \text{(3b)} \\ &= x_1x_3 + x_2' \cdot 1 && \text{(5a)} \\ &= x_1x_3 + x_2' && \text{(4b)} \end{aligned}$$

Unfortunately, one must be fairly clever to apply Boolean algebra properties to simplify an expression. In Section 8.3 we will discuss more systematic approaches to this minimization problem that require less ingenuity. For now, we should say

a bit more about why we want to minimize. When logic networks were built from separate gates and inverters, the cost of these elements was a considerable factor in the design, and it was desirable to have as few elements as possible. Now, however, most networks are built using integrated circuit technology, a development that began in the early 1960s. An integrated circuit is itself a logic network representing a certain truth function or functions, just as if some gates and inverters had been combined in the appropriate arrangement inside a package. These integrated circuits are then combined as needed to produce the desired result. Because the integrated circuits are extremely small and relatively inexpensive, it might seem pointless to bother minimizing a network. However, minimization is still important because the reliability of the final network is inversely related to the number of connections between the integrated circuit packages.

Moreover, the designers of integrated circuits are highly interested in the minimization problem. Integrated circuits are embedded into a substrate of silicon or other semiconductor material. The resulting chips may be tiny, yet they can contain the equivalent of 3 billion transistors for implementing truth functions. The distance between two gates may be as small as 45 nanometers (about 1/1000 of the width of a human hair). Minimizing the number of components and the amount of wiring required to realize a desired truth function makes it possible to embed more functions in a single chip.

Programmable Logic Devices

Instead of designing a custom chip to implement particular truth functions, a **PLD** (*programmable logic device*) can be used. A PLD is a chip that is already implanted with an array of AND gates and an array of OR gates, together with a rectangular grid of wiring channels and some inverters. Once Boolean expressions in sum-of-products form have been determined for the truth functions, the required components in the PLD are activated. Although this chip is not very efficient and is practical only for smaller-scale circuit logic, on the order of hundreds of gates, the PLD can be mass-produced, and only a small amount of time (i.e., money) is then required to “program” it for the desired functions. A **FPGA** (*field-programmable gate array*) is a big brother to the PLD. The term “field-programmable” suggests that, like a PLD, the user can configure the chip for a specific purpose. An FPGA basically connects a number of PLDs in a reconfigurable way, and it can support thousands of gates. Often the FPGA also contains hardwired components such as multipliers or even processors and memory, thus producing a small reconfigurable computer that is “programmed” in hardware rather than software.

EXAMPLE 17

Figure 8.13a shows a PLD for the three inputs $x_1, x_2,$ and x_3 . There are four output lines, so four functions can be programmed in this PLD. When the PLD is programmed, the horizontal line going into an AND gate will pick up certain inputs, and the AND gate will form the product of these inputs. The vertical line going into an OR gate will, when programmed, allow the OR gate to form the sum of certain inputs. Figure 8.13b shows the same PLD programmed to produce the truth functions f_1 from Example 13 ($x_1x_2x_3 + x_1x_2'x_3 + x_1x_2'x_3' + x_1'x_2x_3$) and f_2 from Practice 11 ($x_1x_2x_3 + x_1x_2'x_3 + x_1x_2'x_3' + x_1'x_2x_3 + x_1'x_2'x_3'$). The dots represent activation points.

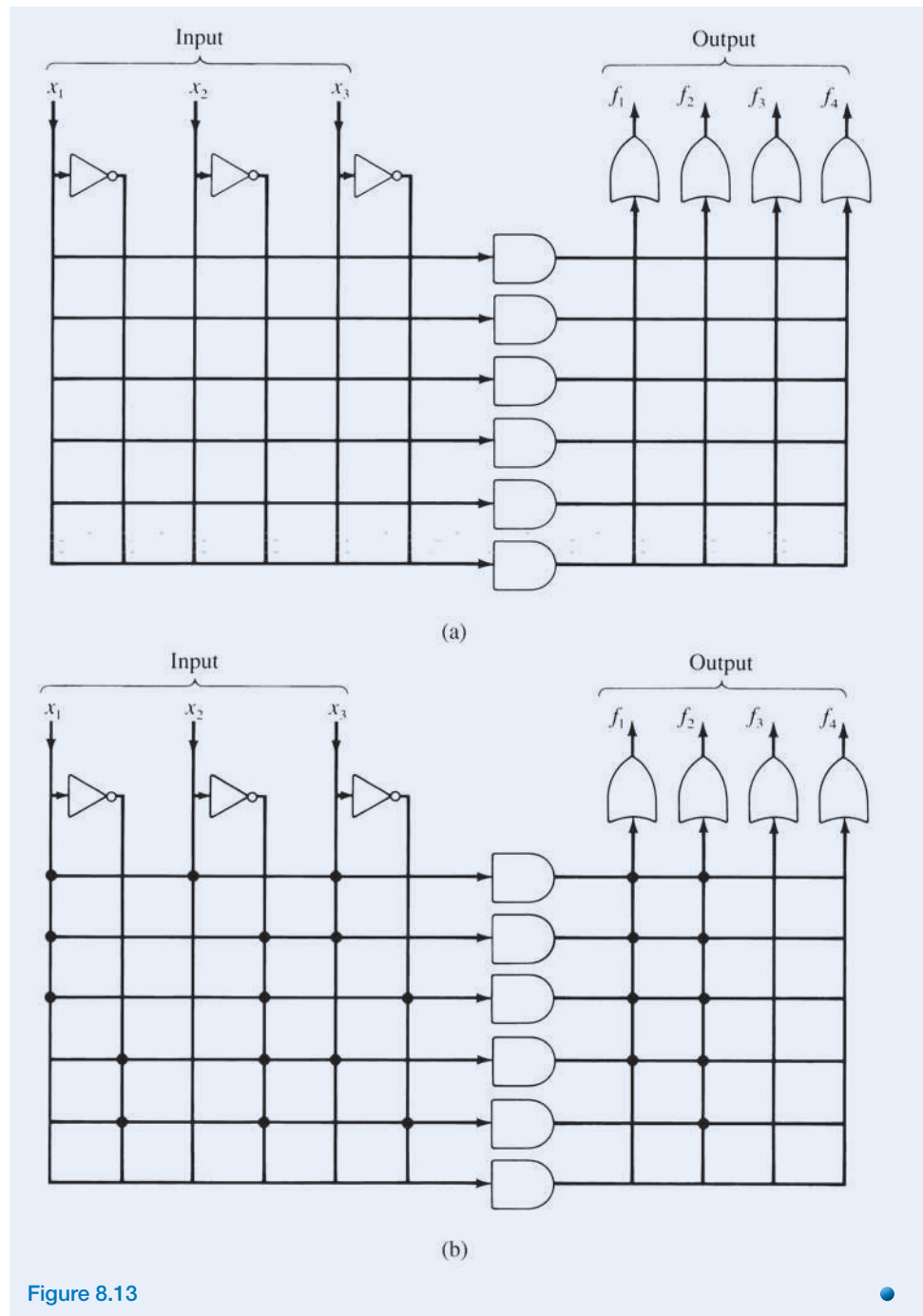


Figure 8.13

A Useful Network

We can design a network that adds binary numbers, a basic operation that a computer must be able to perform. The rules for adding two one-bit numbers are summarized in Table 8.6.

TABLE 8.6		
x_1	x_2	Sum
1	1	10
1	0	1
0	1	1
0	0	0

TABLE 8.7		
x_1	x_2	s
1	1	0
1	0	1
0	1	1
0	0	0

TABLE 8.8		
x_1	x_2	c
1	1	1
1	0	0
0	1	0
0	0	0

We can express the sum as a single sum bit s (the right-hand bit of the actual sum) together with a single carry bit c ; doing this gives us the two truth functions of Tables 8.7 and 8.8, respectively. The canonical sum-of-products form for each truth function is

$$s = x_1'x_2 + x_1x_2'$$

$$c = x_1x_2$$

An equivalent Boolean expression for s is

$$s = (x_1 + x_2)(x_1x_2)'$$

Figure 8.14a shows a network with inputs x_1 and x_2 and outputs s and c . This device, for reasons that will be clear shortly, is called a **half-adder**.

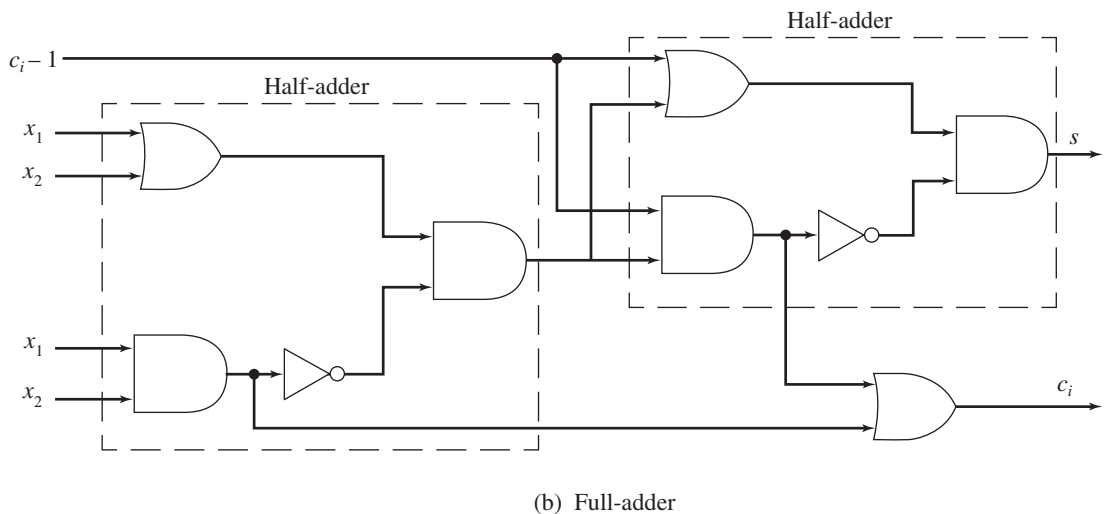
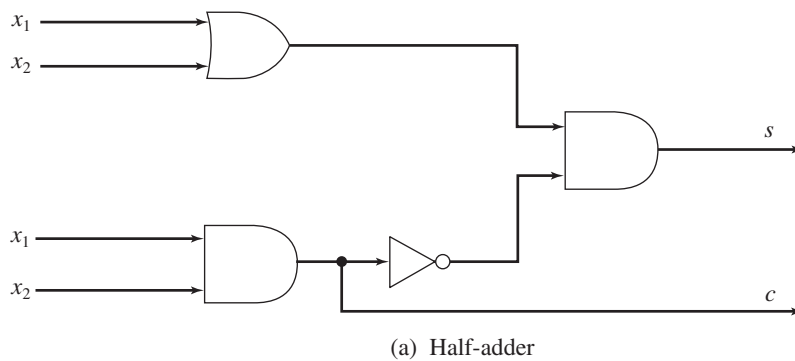


Figure 8.14

To add two n -bit binary numbers, we add column by column from the low-order to the high-order bits. The i th column (except for the very first column) has as input its two bits x_i and x_{i-1} plus the carry bit from the addition of column $i - 1$ to its right. Thus we need a device incorporating the previous carry bit as input. Such a device can be accomplished by adding x_i and x_{i-1} with a half-adder and then adding the previous carry bit c_{i-1} (using another half-adder) to the result. Again, a sum bit s_i and final carry bit c_i are output, where c_i is 1 if either half-adder produces a 1 as its carry bit. The **full-adder** is shown in Figure 8.14b. The full-adder is thus composed of two half-adders and an additional OR gate.

To add two n -bit binary numbers, the two low-order bits, where there is no input carry bit, can be added with a half-adder. The remaining bits are added with full-adders. All are chained together. Figure 8.15 shows the modules required to add two 3-bit binary numbers $z_1y_1x_1$ and $z_2y_2x_2$, resulting in the answer $a_3a_2a_1a_0$, where the leading bit a_3 is the final carry bit and could be 0 (leading 0s are usually not written) or 1.

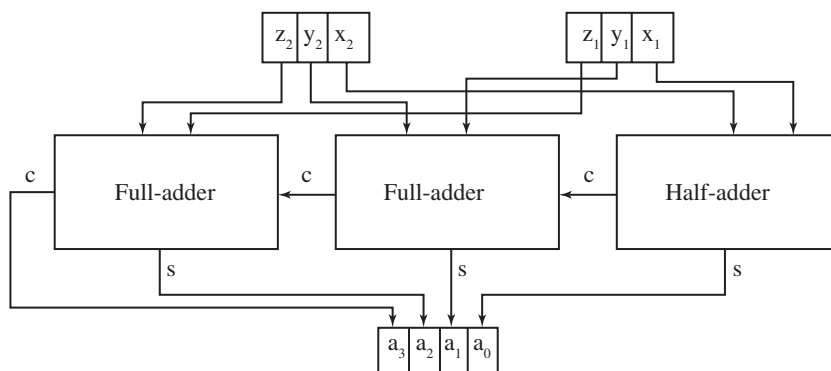


Figure 8.15

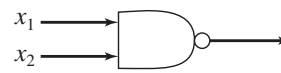
The adder circuit shown in Figure 8.15 is called a “ripple-carry adder” because the carry bits have to propagate right to left through each adder in turn. Although we have assumed that gates output instantaneously, there is in fact a small time delay in an n -bit adder due to this ripple effect that can be appreciable for large n . Variations on the basic circuitry that speed up the addition process depend on anticipating the higher-order carry bits.

PRACTICE 12

Trace the operation of the circuit in Figure 8.15 as it adds 101 and 111.

Other Logic Elements

The basic elements used in integrated circuits are not really AND and OR gates and inverters, but NAND and NOR gates. Figure 8.16 shows the standard symbol for the **NAND gate** (the NOT AND gate) and its truth function. The NAND gate alone is sufficient to realize any truth function because networks using only NAND gates can do the job of inverters, OR gates, and AND gates. Figure 8.17 shows these networks.



x_1	x_2	$(x_1 + x_2)'$
1	1	0
1	0	1
0	1	1
0	0	1

Figure 8.16

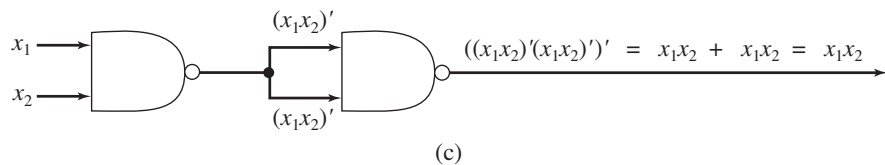
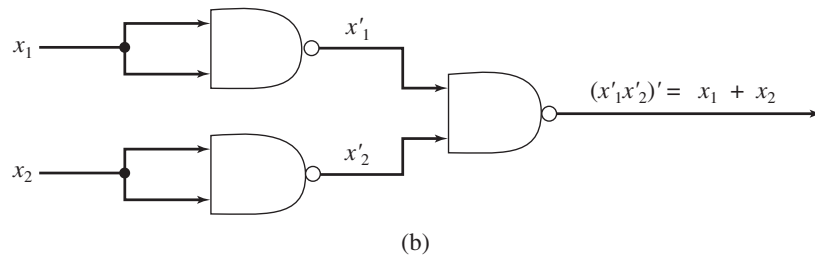
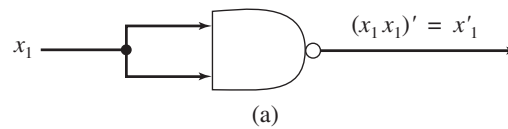


Figure 8.17

The **NOR gate** (the NOT OR gate) and its truth function appear in Figure 8.18. An exercise at the end of this section asks you to construct networks using only NOR gates for inverters, OR gates, and AND gates.¹

¹Exercises 51 and 52 of Section 1.1 give truth tables for binary connectives that agree with the NAND and NOR truth functions. There you were asked to prove that either of these two connectives is sufficient to write any propositional wff; that is, you can write \vee , \wedge , and $'$ in terms of one of these connectives.



x_1	x_2	$(x_1 + x_2)'$
1	1	0
1	0	0
0	1	0
0	0	1

Figure 8.18

Although we can construct a NAND network for a truth function by replacing AND gates, OR gates, and inverters in the canonical form or a minimized form with the appropriate NAND networks, we can often obtain a simpler network by using the properties of NAND elements directly.

PRACTICE 13

- Rewrite the network of Figure 8.12 with NAND elements by directly replacing the AND gate, OR gate, and inverter, as in Figure 8.17.
- Rewrite the Boolean expression $x_1x_3 + x_2'$ for Figure 8.12 using De Morgan's laws, and then construct a network using only two NAND elements. ■

Constructing Truth Functions

We know how to write a Boolean expression and construct a network from a given truth function. Often the truth function itself must first be deduced from the description of the actual problem.

EXAMPLE 18

At a mail-order cosmetics firm, an automatic control device is used to supervise the packaging of orders. The firm sells lipstick, perfume, makeup, and nail polish. As a bonus item, shampoo is included with any order that includes perfume or any order that includes lipstick, makeup, and nail polish. How can we design the logic network that controls whether shampoo is packaged with an order?

The inputs to the network will represent the four items that can be ordered. We label the items

$$\begin{aligned} x_1 &= \text{lipstick} \\ x_2 &= \text{perfume} \\ x_3 &= \text{makeup} \\ x_4 &= \text{nail polish} \end{aligned}$$

The value of x_i will be 1 when that item is included in the order and 0 otherwise. The output from the network should be 1 if shampoo is to be packaged with the order and 0 otherwise. The truth table for the circuit appears in Table 8.9. The canonical sum-of-products form for this truth function is lengthy, but the expression

$x_1x_3x_4 + x_2$ also represents the function. Figure 8.19 shows the logic network for this expression.

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
1	1	1	1	1
1	1	1	0	1
1	1	0	1	1
1	1	0	0	1
1	0	1	1	1
1	0	1	0	0
1	0	0	1	0
1	0	0	0	0
0	1	1	1	1
0	1	1	0	1
0	1	0	1	1
0	1	0	0	1
0	0	1	1	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	0

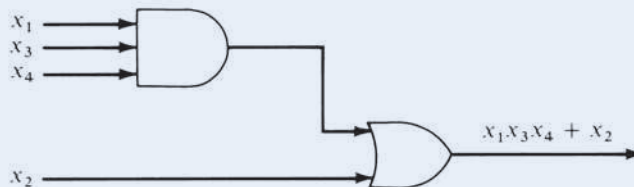


Figure 8.19

PRACTICE 14

A hall light is controlled by two light switches, one at each end of the hall. Find (a) a truth function, (b) a Boolean expression, and (c) a logic network that allows the light to be switched on or off by either switch.

In some problems the corresponding truth functions have certain undefined values because certain combinations of input cannot occur (see Exercise 35 at the end of this section). Under these “don’t-care” conditions, any value may be assigned to the output.

In a programming language where the Boolean operators AND, OR, and NOT are available, designing the logic of a computer program may consist in part of choosing appropriate truth functions and their corresponding Boolean expressions (see Exercise 36 of Section 1.1).

Pruning Chips and Programs

Computer computations (barring coding errors) are generally viewed as the height of accuracy. Translating truth functions into circuits that produce correct outputs for given inputs has been the theme of this chapter. But some computer applications do not require 100% accuracy and can tolerate a certain amount of error. Image processing is one such application because small variations in an image are imperceptible to the human eye. (This feature of human eyesight is used to advantage in JPEG lossy image compression, as discussed in Section 6.4.)

Recently, researchers looking for increased efficiency in chip design have “pruned” traditional circuits, essentially cutting away the parts that are seldom used. Pruned chips consume less energy and are both smaller and faster than the complete chip. We can lump these three factors—energy, space, and time—together under the general term “efficiency.” Experimental pruning of a chip’s addition circuit produced a 7.5% gain in overall efficiency. But of course there is a tradeoff; some percentage of error is introduced in the chip’s operation. The 7.5% efficiency gain came at the cost of a 0.25% error rate. For specialized applications such as image processing, where small amounts of error can be tolerated, the tradeoff seems to be a good thing. Developers of this technology are looking toward applications such as hearing aids, cameras, or even tablet computers that could run on solar power.

The same approach is also being tried with software. For example, “loop perforation” skips some of the iterations in a loop. This is a particularly evocative name—“perforating a loop” sounds very much like “pruning a chip.” Other approaches may skip entire tasks or randomly discard some input values. “Relaxed program” is a general title for software that has built-in nondeterminism that can dynamically prune (skip) some of its instructions or data. The benefit is increased runtime efficiency. The penalty is some percentage of incorrect results. And the trick, of course, is to ensure—and formally verify—that such techniques, while increasing efficiency, keep the output within an acceptable error range.

“Inexact Design—Beyond Fault Tolerance,” Anthes, G., *Communications of the ACM*, April, 2013.

<http://news.rice.edu/2012/05/17/computing-experts-unveil-superefficient-inexact-chip/>

<http://web.mit.edu/newsoffice/2010/fuzzy-logic-0103.html>

“Proving Acceptability Properties of Relaxed Nondeterministic Approximate Programs,” Carbin, M., Kim, D., Misailovic, S., Rinard, M., ACM Conference on Programming Language Design and Implementation, June 11–16, 2012, Beijing, China.

<http://web.mit.edu/newsoffice/2012/loop-perforation-0522.html>

SECTION 8.2 REVIEW

TECHNIQUES

- Find the truth function corresponding to a given Boolean expression or logic network.
- **W** Construct a logic network with the same truth function as a given Boolean expression.
- Write a Boolean expression with the same truth function as a given logic network.
- **W** Write the Boolean expression in canonical sum-of-products form for a given truth function.
- Find a network composed only of NAND gates that has the same truth function as a given network with AND gates, OR gates, and inverters.
- Find a truth function that satisfies the description of a particular problem.

MAIN IDEAS

- We can effectively convert information from any of the following three forms to any other form:

$$\text{truth function} \leftrightarrow \text{Boolean expression} \leftrightarrow \text{logic network}$$

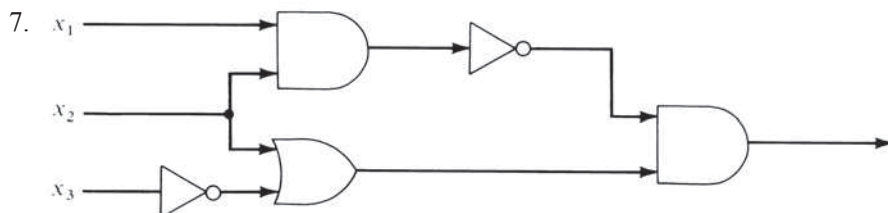
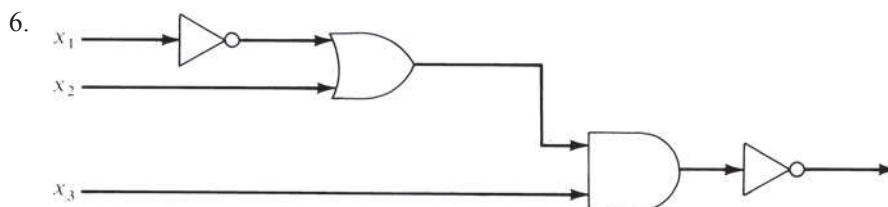
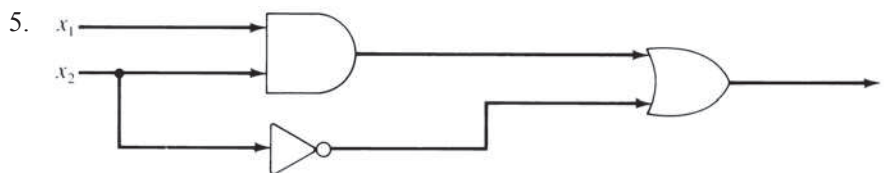
- A Boolean expression can sometimes be converted to a simpler, equivalent expression using the properties of Boolean algebra, thus producing a simpler network for a given truth function.

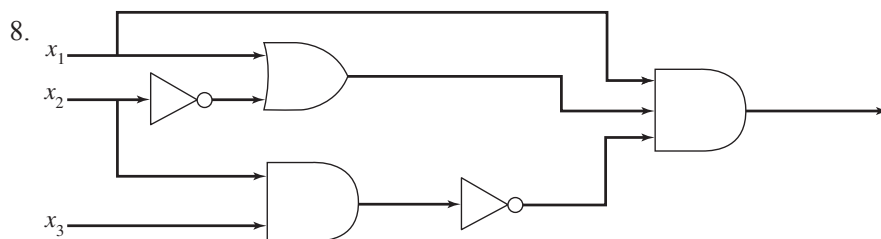
EXERCISES 8.2

For Exercises 1–4, write a truth function and construct a logic network using AND gates, OR gates, and inverters for each of the given Boolean expressions.

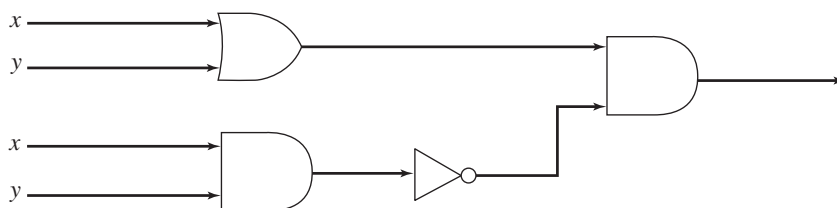
1. $(x'_1 + x_2)x_3$
2. $(x_1 + x'_2) + x'_1x_3$
3. $x'_1x_2 + (x_1x_2)'$
4. $(x_1 + x_2)'x_3 + x'_3$

For Exercises 5–8, write a Boolean expression and a truth function for each of the logic networks shown.





9. a. Write the truth function for the Boolean operation $x \oplus y = xy' + yx'$.
 b. Draw the logic network for $x \oplus y$.
 c. Show that the network of the accompanying figure also represents $x \oplus y$. Explain why the network illustrates that \oplus is the **exclusive OR** operation. (Recall that a bitwise exclusive-OR operation is used in DES encoding, as discussed in Section 5.6.)



10. a. Write the truth function for the Boolean expression

$$(xy)'(yx)'$$

- b. Draw the logic network for this expression.
 c. By looking at either the truth function or the logic network, what propositional logic connective does this Boolean expression represent?

For Exercises 11–20, find the canonical sum-of-products form for the truth functions in the given tables.

11.

x_1	x_2	$f(x_1, x_2)$
1	1	0
1	0	0
0	1	0
0	0	1

12.

x_1	x_2	$f(x_1, x_2)$
1	1	1
1	0	0
0	1	1
0	0	0

13.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

14.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	1

15.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	0
1	1	0	0
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	0

16.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	0

17.

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
1	1	1	1	1
1	1	1	0	0
1	1	0	1	1
1	1	0	0	0
1	0	1	1	1
1	0	1	0	0
1	0	0	1	1
1	0	0	0	0
0	1	1	1	0
0	1	1	0	0
0	1	0	1	0
0	1	0	0	0
0	0	1	1	1
0	0	1	0	1
0	0	0	1	0
0	0	0	0	0

18.

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
1	1	1	1	1
1	1	1	0	0
1	1	0	1	1
1	1	0	0	0
1	0	1	1	1
1	0	1	0	1
1	0	0	1	0
1	0	0	0	0
0	1	1	1	1
0	1	1	0	0
0	1	0	1	1
0	1	0	0	0
0	0	1	1	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	0

19.

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
1	1	1	1	0
1	1	1	0	0
1	1	0	1	0
1	1	0	0	0
1	0	1	1	0
1	0	1	0	1
1	0	0	1	0
1	0	0	0	0
0	1	1	1	1
0	1	1	0	0
0	1	0	1	1
0	1	0	0	0
0	0	1	1	1
0	0	1	0	1
0	0	0	1	1
0	0	0	0	0

20.

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
1	1	1	1	1
1	1	1	0	1
1	1	0	1	0
1	1	0	0	1
1	0	1	1	1
1	0	1	0	0
1	0	0	1	0
1	0	0	0	0
0	1	1	1	0
0	1	1	0	1
0	1	0	1	0
0	1	0	0	1
0	0	1	1	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	0

21. a. Find the canonical sum-of-products form for the truth function in the accompanying table.
 b. Draw the logic network for the expression of part (a).
 c. Use properties of a Boolean algebra to reduce the expression of part (a) to an equivalent expression whose network requires only two logic elements. Draw the network.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

22. a. Find the canonical sum-of-products form for the truth function in the accompanying table.
 b. Draw the logic network for the expression of part (a).
 c. Use properties of a Boolean algebra to reduce the expression of part (a) to an equivalent expression whose network requires only three logic elements. Draw the network.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	0
0	0	0	0

23. a. Show that the two Boolean expressions

$$(x_1 + x_2)(x_1' + x_3)(x_2 + x_3)$$

and

$$(x_1x_3) + (x_1'x_2)$$

are equivalent by writing the truth function for each.

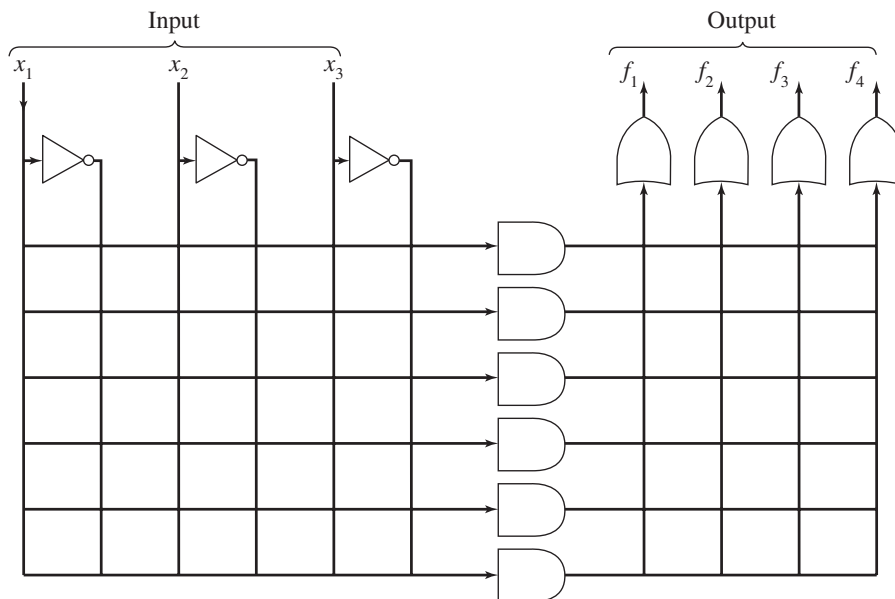
b. Write the canonical sum-of-products form equivalent to the two expressions of part (a).

c. Use properties of a Boolean algebra to reduce one of the expressions of part (a) to the other.

24. The accompanying figure shows an unprogrammed PLD for three inputs, x_1 , x_2 , and x_3 . Program this PLD to generate the truth functions f_1 and f_3 represented by

$$f_1: x_1x_2x_3 + x_1'x_2x_3' + x_1'x_2'x_3$$

$$f_3: x_1x_2'x_3' + x_1'x_2'x_3 + x_1'x_2x_3'$$



25. There is also a *canonical product-of-sums form (conjunctive normal form)* for any truth function. This expression has the form

$$()() \cdots ()$$

with each factor a sum of the form

$$\alpha + \beta + \cdots + \omega$$

where $\alpha = x_1$ or x_1' , $\beta = x_2$ or x_2' , and so on. Each factor is constructed to have a value of 0 for the input values of exactly one of the rows of the truth function having value 0. Thus, the entire expression has value 0 for these inputs and no others. Find the canonical product-of-sums form for the truth functions of Exercises 11–15.

26. Consider the truth function given here. Because there are many more 1s than 0s, the canonical sum-of-products form might seem tedious to compute. Instead, create a Boolean sum-of-products expression using the same formula as before but on the 0 rows instead of the 1 rows. This expression will give outputs of 1s for those rows and no others, exactly the opposite of what you want. Then complement the expression (equivalent to sticking an inverter on the end of the network).

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	0
0	0	0	1

- Use this approach to create a Boolean expression for this truth function.
 - Prove that the resulting expression is equivalent to the canonical product-of-sums form described in Exercise 25.
27. The *2's complement* of an n -bit binary number p is an n -bit binary number q such that $p + q$ equals an n -bit representation of zero (any carry bit to column $n + 1$ is ignored). Thus 01110 is the 2's complement of 10010 because

$$\begin{array}{r} 10010 \\ + 01110 \\ \hline (1)00000 \end{array}$$

The 2's complement idea can be used to represent negative integers in binary form. After all, the negative of p is by definition a number that, when added to p , results in zero.

Given a binary number p , the 2's complement of p is found by scanning p from low-order to high-order bits (right to left). As long as bit i of p is 0, bit i of q is 0. When the first 1 of p is encountered, say at bit j , then bit j of q is 1, but for the remaining bits, $j < i \leq n$, $q_i = p'_i$. For $p = 10010$, for instance, the

rightmost 0 bit of p stays a 0 bit in q , and the first 1 bit stays a 1 bit. The remaining bits of q , however, are the reverse of the bits in p .

$$\begin{array}{r}
 \text{First 1} \\
 \swarrow \\
 p = 100 \quad | \quad 10 \\
 q = 011 \quad | \quad 10 \\
 q_i = p'_i \quad | \quad q_i = p_i
 \end{array}$$

For each binary number p , find the 2's complement of p , namely q , and then calculate $p + q$.

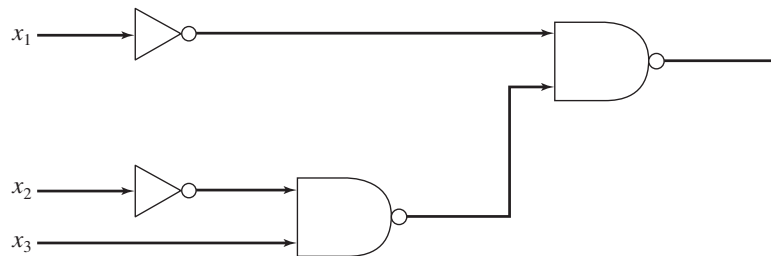
- a. 1100 b. 1001 c. 001

28. For any bit x_i in a binary number p , let r_i be the corresponding bit in q , the 2's complement of p (see Exercise 27). The value of r_i depends on the value of x_i and also on the position of x_i relative to the first 1 bit in p . For the i th bit, let c_{i-1} denote a 0 if the bits p_j , $1 \leq j \leq i-1$, are 0 and a 1 otherwise. A value c_i must be computed to move on to the next bit.
- Give a truth function for r_i with inputs x_i and c_{i-1} . Give a truth function for c_i with inputs x_i and c_{i-1} .
 - Write Boolean expressions for the truth functions of part (a). Simplify as much as possible.
 - Design a circuit module to output r_i and c_i from inputs x_i and c_{i-1} .
 - Using the modules of part (c), design a circuit to find the 2's complement of a three-bit binary number zyx . Trace the operation of the circuit in computing the 2's complement of 110.
29. a. Construct a network for the following expression using only NAND elements. Replace the AND and OR gates and inverters with the appropriate NAND networks.

$$x'_3x_1 + x'_2x_1 + x'_3$$

- Use the properties of a Boolean algebra to reduce the expression of part (a) to one whose network would require only three NAND gates. Draw the network.

30. Replace the following network with an equivalent network using one AND gate, one OR gate, and one inverter.

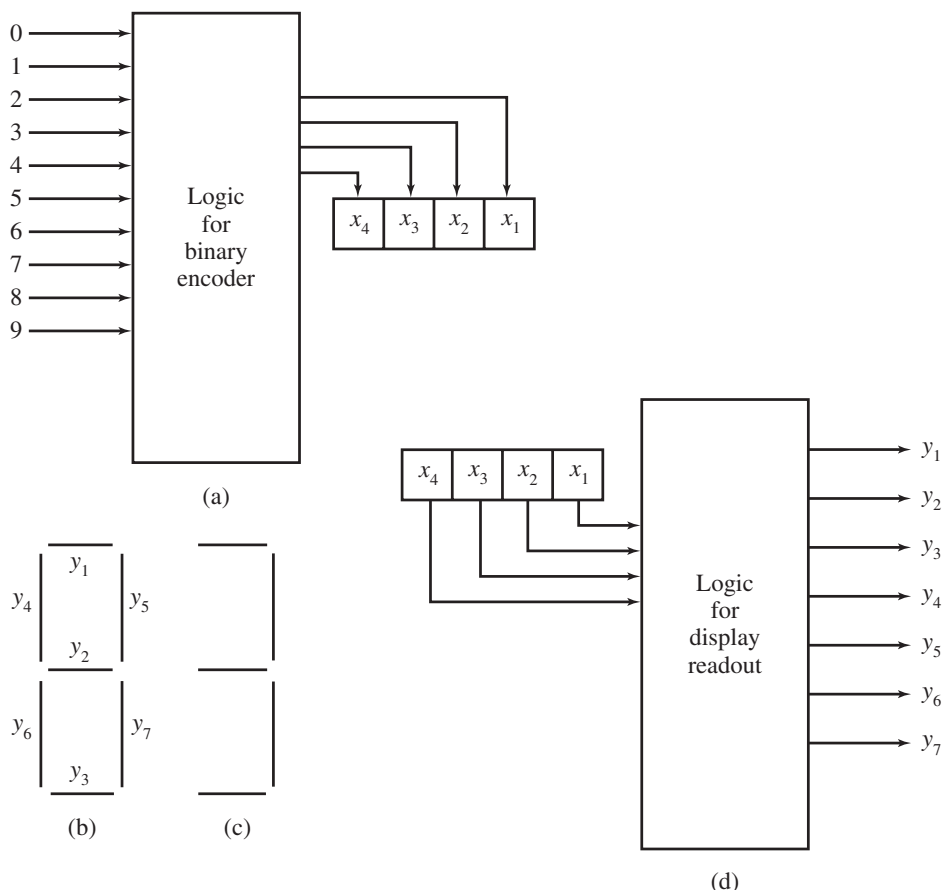


- Using only NOR elements, construct networks that can replace (a) an inverter, (b) an OR gate, and (c) an AND gate.
- Find an equivalent network for the half-adder module that uses exactly five NAND gates. Draw the network.
- A thermostat controls a heating system that should raise the temperature above 67°F during working hours (between 7:00 AM and 6:00 PM). The input values are

$$\begin{array}{l}
 x_1 = 1 \text{ when the temperature is less than } 67^\circ\text{F} \\
 x_1 = 0 \text{ otherwise} \\
 x_2 = 1 \text{ when the time is less than 7:00 AM or greater than 6:00 PM} \\
 x_2 = 0 \text{ otherwise}
 \end{array}$$

Find a truth function, a Boolean expression, and a logic network for when the heating system should be on (value 1) or off (value 0).

34. The space shuttle was controlled by three on-board computers; binary output from these three computers was compared and a majority vote was required for certain actions to take place. Find a truth function, a Boolean expression, and a logic network that outputs the majority vote of the three input values.
35. You have just been hired at Mercenary Motors. Your job is to design a logic network so that a car can be started only when the automatic transmission is in neutral or park and the driver's seat belt is fastened. Find a truth function, a Boolean expression, and a logic network. (There is a don't-care condition to the truth function, since the car cannot be in both neutral and park.)
36. Mercenary Motors has expanded into the calculator business. You need to design the circuitry for the display readout on a new calculator. This design involves a two-step process.
- Any digit 0, 1, ..., 9 put into the calculator is first converted to binary form. Part (a) of the accompanying figure illustrates this conversion, which involves four separate networks, one each for x_1 to x_4 . Each network has 10 inputs, but only 1 input can be on at any given moment. Write a Boolean expression and then draw a network for x_2 .
 - The binary form of the digit is then converted into a visual display by activating a pattern of seven outputs arranged as shown in figure (b). To display the digit 3, for example, $y_1, y_2, y_3, y_5,$ and y_7 must be on, as in figure (c). Thus, the second step of the process can be represented by figure (d), which involves seven separate networks, one each for y_1 to y_7 , each with four inputs, x_1 to x_4 . Write a truth function, a Boolean expression, and a network for y_5 and for y_6 .



37. A *multiplexor* is a control circuit with 2^n input lines (numbered 0 through $2^n - 1$), n selector lines, and exactly one output line. The signal on the output line is to match the signal on one of the input lines; the selector lines determine which of the input line signals will be propagated to the output line. Signals on the n selector lines determine an n -bit binary number, which can range in value from 0 to $2^n - 1$; hence the numeric value on the selector lines identifies exactly one input line. The arithmetic-logic unit of a computer processor contains circuits for various operations (addition, subtraction, comparison, and so forth), and when an arithmetic or comparison operation is to be done, the ALU may activate all these circuit. A multiplexor then picks out the one desired result.
- Write a truth function for a multiplexor where $n = 1$; that is, there are 2 input lines, 1 selector line and, of course, one output line.
 - Draw the logic network (as simple as possible).
38. A *decoder* is a control circuit with n input lines and 2^n output lines (numbered 0 through $2^n - 1$). The pattern of n bits on the input lines represents a binary number between 0 and $2^n - 1$. A decoder activates the output line with the corresponding identification number by putting a 1 output on that line and 0 outputs on all other output lines. A decoder in a computer can, for example, read input lines that represent a binary memory address and then activate the line to that memory cell for a read operation.
- Write truth functions for a decoder where $n = 2$, that is, there are 2 input lines and $2^2 = 4$ output lines (hence four truth functions).
 - Draw the logic network that incorporates all four truth functions.
39. At the beginning of this chapter, you were

... hired by Rats R Us to build the control logic for the production facilities for a new anticancer chemical compound being tested on rats. The control logic must manage the opening and closing of two valves, A and B , downstream of the mixing vat. Valve A is to open whenever the pressure in the vat exceeds 50 psi (pounds per square inch) and the salinity of the mixture exceeds 45 g/L (grams per liter). Valve B is to open whenever valve A is closed and the temperature exceeds 53°C and the acidity falls below 7.0 pH (lower pH values mean more acidity).

How many and what type of logic gates will be needed in the circuit?

Answer this question by finding the canonical sum-of-products form for the logic circuits to control A and B .

SECTION 8.3 MINIMIZATION

Minimization Process

Remember from Section 8.2 that a given truth function is associated with an equivalence class of Boolean expressions. If we want to design a logic network for the function, the ideal would be to have a procedure that chooses the simplest Boolean expression from the class. What we consider simple will depend on the technology employed in building the network, what kind of logic elements are available, and so on. At any rate, we probably want to minimize the total number of connections that must be made and the total number of logic elements used. (As we discuss minimization procedures, keep in mind that other factors may influence the economics of the situation. If a network is to be built only once, the time spent on minimization is costlier than building the network. But if the network is to be mass-produced, then the cost of minimization time may be worthwhile.)

We have had some experience in simplifying Boolean expressions by applying the properties of Boolean algebra. However, we had no procedure to use. We simply had to guess, attacking each problem individually. What we want now is a mechanical procedure that we can use without having to be clever or insightful. Unfortunately, we won't develop the ideal procedure. However, we already know how to select the canonical sum-of-products form from the equivalence class of expressions for a given truth function. In this section we will discuss two procedures to reduce a canonical sum-of-products form to a minimal sum-of-products form. Therefore, we can minimize within the framework of a sum-of-products form and reduce, if not completely minimize, the number of elements and connections required.

EXAMPLE 19

The Boolean expression

$$x_1x_2x_3 + x_1'x_2x_3 + x_1'x_2x_3'$$

is in sum-of-products form. An equivalent minimal sum-of-products form is

$$x_2x_3 + x_2x_1'$$

Implementing a network for this form would require two AND gates, one OR gate, and an inverter. Using one of the distributive laws of Boolean algebra, this expression reduces to

$$x_2(x_3 + x_1')$$

which requires only one AND gate, one OR gate, and an inverter, but it is no longer in sum-of-products form. Thus, a minimal sum-of-products form may not be minimal in an absolute sense. ●

There are two extremely useful equivalences in minimizing a sum-of-products form. They are

$$x_1x_2 + x_1'x_2 = x_2$$

and

$$x_1 + x_1'x_2 = x_1 + x_2$$

PRACTICE 15 Use properties of Boolean algebra to reduce the following expressions as indicated:

- $x_1x_2 + x_1'x_2$ to x_2
- $x_1 + x_1'x_2$ to $x_1 + x_2$

The equivalence $x_1x_2 + x_1'x_2 = x_2$ means, for example, that the expression $x_1'x_2x_3x_4 + x_1x_2'x_3x_4$ reduces to $x_1'x_3x_4$. Thus, when we have a sum of two products that differ in only one factor, we can eliminate that factor. However, the canonical

sum-of-products form for a truth function of, say, four variables might be quite long and require some searching to locate two product terms differing by only one factor. To help us in this search, we can use the *Karnaugh map*. The Karnaugh map is a visual representation of the truth function so that terms in the canonical sum-of-products form that differ by only one factor can be matched quickly.

Karnaugh Map

In the canonical sum-of-products form for a truth function, we are interested in values of the input variables that produce outputs of 1. The Karnaugh map records the 1s of the function in an array that forces products of inputs differing by only one factor to be adjacent. The array form for a two-variable function is given in Figure 8.20. Notice that the square corresponding to x_1x_2 , the upper left-hand square, is adjacent to squares $x_1'x_2$ and x_1x_2' , which differ in one factor from x_1x_2 ; however, it is not adjacent to the $x_1'x_2'$ square, which differs in two factors from x_1x_2 .

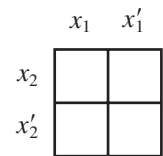


Figure 8.20

EXAMPLE 20

The truth function of Table 8.10 is represented by the Karnaugh map of Figure 8.21. At once we can observe 1s in two adjacent squares, so there are two terms in the canonical sum-of-products form differing by one variable; again from the map, we see that the variable that changes is x_1 . It can be eliminated. We conclude that the function can be represented by x_2 . Indeed, the canonical sum-of-products form for the function is $x_1x_2 + x_1'x_2$, which, by our basic reduction rule, reduces to x_2 . However, we did not have to write the canonical form—we only had to look at the map.

x_1	x_2	$f(x_1, x_2)$
1	1	1
1	0	0
0	1	1
0	0	0

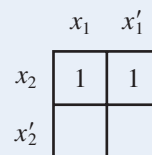


Figure 8.21

PRACTICE 16

Draw the Karnaugh map and use it to find a reduced expression for the function in Table 8.11.

x_1	x_2	$f(x_1, x_2)$
1	1	0
1	0	0
0	1	1
0	0	1

Maps for Three and Four Variables

The array forms for functions of three and four variables are shown in Figure 8.22. In these arrays, adjacent squares also differ by only one variable. However, in Figure 8.22a, the leftmost and rightmost squares in a row also differ by one variable, so we consider them adjacent. (They would in fact be adjacent if we wrapped the map around a cylinder and glued the left and right edges together.) In Figure 8.22b, the leftmost and rightmost squares in a row are adjacent (differ by exactly one variable), and also the top and bottom squares in a column are adjacent.

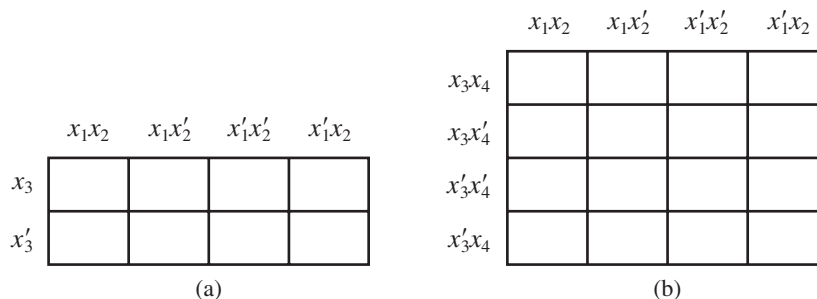


Figure 8.22

REMINDER

It is crucial to label the Karnaugh map so that adjacent squares differ by one variable.

In three-variable maps, when two adjacent squares are marked with 1, one variable can be eliminated; when four adjacent squares are marked with 1 (either in a single row or arranged in a square), two variables can be eliminated.

The labelings for two-, three-, and four-variable maps can be done in various ways, but they must be done so that adjacent squares differ by one variable. It's probably best to just memorize the labeling schemes we've used here.

EXAMPLE 21

In the map of Figure 8.23, the squares that combine for a reduction are shown as a block. These four adjacent squares reduce to x_3 (eliminate the changing variables x_1 and x_2). The reduction uses our basic reduction rule more than once:

$$\begin{aligned}
 x_1x_2x_3 + x_1x'_2x_3 + x'_1x'_2x_3 + x'_1x_2x_3 &= x_1x_3(x_2 + x'_2) + x'_1x_3(x'_2 + x_2) \\
 &= x_1x_3 + x'_1x_3 \\
 &= x_3(x_1 + x'_1) \\
 &= x_3
 \end{aligned}$$

But, again, you don't have to go through this process; you can just look at the Karnaugh map in Figure 8.23.

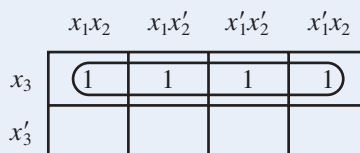


Figure 8.23

In four-variable maps, when two adjacent squares are marked with 1, one variable can be eliminated; when four adjacent squares are marked with 1, two variables can be eliminated; when eight adjacent squares are marked with 1, three variables can be eliminated.

Figure 8.24 illustrates some instances of two adjacent marked squares, Figure 8.25 illustrates some instances of four adjacent marked squares, and Figure 8.26 shows instances of eight.

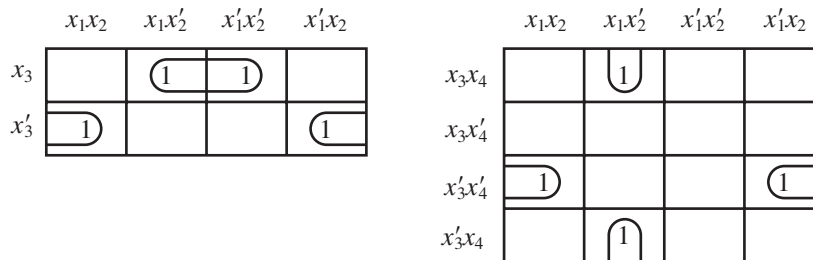


Figure 8.24

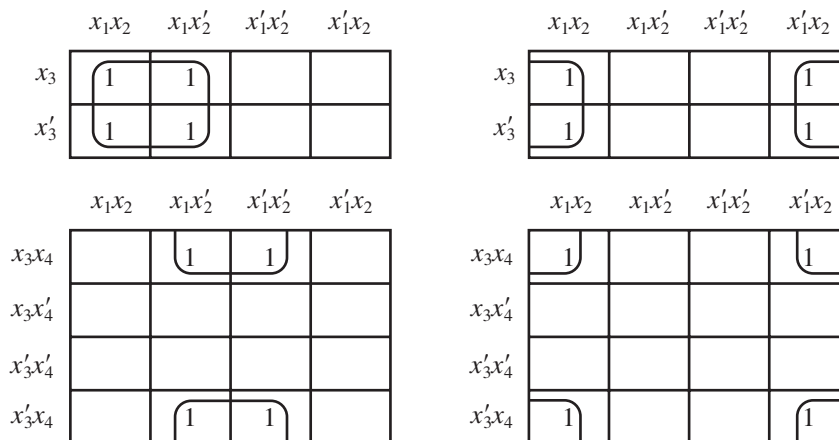


Figure 8.25

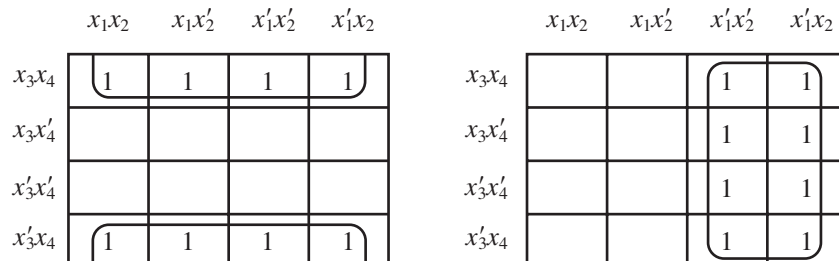


Figure 8.26

EXAMPLE 22

In the map of Figure 8.27, the four outside corners reduce to x_2x_4 and the inside square reduces to $x_2'x_4'$.

	x_1x_2	x_1x_2'	$x_1'x_2$	$x_1'x_2'$
x_3x_4	1			1
x_3x_4'		1	1	
$x_3'x_4$		1	1	
$x_3'x_4'$	1			1

Figure 8.27

PRACTICE 17

Find the two terms represented by the map in Figure 8.28.

	x_1x_2	x_1x_2'	$x_1'x_2$	$x_1'x_2'$
x_3x_4	1	1		
x_3x_4'	1	1		
$x_3'x_4$				1
$x_3'x_4'$				1

Figure 8.28

Using the Karnaugh Map

How do we find a minimal sum-of-products form from a Karnaugh map (or from a truth function or a canonical sum-of-products form)? We must use every marked square of the map, and we want to include every marked square in the largest combination of marked squares possible, since doing so will reduce the expression as much as possible. Surprisingly, we cannot begin by simply looking for the largest blocks of marked squares on the map.

EXAMPLE 23

In the Karnaugh map of Figure 8.29, if we simply looked for the largest block of marked squares, we would use the column of 1s and reduce it to $x_1'x_2'$. However, we would still have four marked squares unaccounted for. Each of these marked squares can be combined into a two-square block in only one way (Figure 8.30), and each of these blocks has to be included. But when this adjustment is made, every square in the column of 1s is used, and the term $x_1'x_2'$ is superfluous. The minimal sum-of-products form for this map becomes

$$x_2'x_3x_4 + x_1'x_3x_4' + x_2'x_3'x_4' + x_1'x_3'x_4$$

	x_1x_2	$x_1x'_2$	$x'_1x'_2$	x'_1x_2
x_3x_4		1	1	
$x_3x'_4$			1	1
$x'_3x'_4$		1	1	
x'_3x_4			1	1

Figure 8.29

	x_1x_2	$x_1x'_2$	$x'_1x'_2$	x'_1x_2
x_3x_4		1	1	
$x_3x'_4$			1	1
$x'_3x'_4$		1	1	
x'_3x_4			1	1

Figure 8.30

To avoid the redundancy illustrated by Example 23, we analyze the map as follows. First, we form terms for those marked squares that cannot be combined with anything. Then we use the remaining marked squares to find those that can be combined only into two-square blocks and in only one way. Then among the unused marked squares—that is, those not already assigned to a block—we find those that can be combined only into four-square blocks and in only one way; then we look for any unused squares that go uniquely into eight-square blocks. At each step, if an unused marked square can go into more than one block, we do nothing with it. Finally, we take any unused marked squares that are left (for which there was a choice of blocks) and select blocks that include them in the most efficient manner.

Table 8.12 shows the steps involved. Note, however, that this procedure for handling Karnaugh maps is not, strictly speaking, an algorithm because it doesn't always produce the correct result. If there are many 1s in the map, thus allowing many different blockings, even this procedure may not lead to a minimal form (see Example 28).

TABLE 8.12**Steps in Using Karnaugh Maps**

1. Set up the grid, using correct labeling for the number of Boolean variables.
2. Insert 1s in the table for the terms in the canonical sum-of-products expression.
3. Form terms for any isolated marked squares.
4. Combine squares uniquely into two-square blocks, if possible.
5. Combine squares uniquely into four-square blocks, if possible.
6. Combine squares uniquely into eight-square blocks, if possible.
7. Combine any remaining unused marked squares into blocks as efficiently as possible.

EXAMPLE 24

In Figure 8.31a we show the only square that cannot be combined into a larger block. In Figure 8.31b, we have formed the unique two-square block for the $x_1x_2x'_3$ square and the unique two-square block for the $x'_1x'_2x_3$ square. All marked squares are covered. The minimal sum-of-products expression is

$$x_1x_2x_3 + x'_2x'_3 + x'_1x'_2$$

Formally, the last two terms are obtained by expanding $x_1'x_2'x_3'$ into $x_1'x_2'x_3 + x_1'x_2'x_3'$ and then combining it with each of its neighbors.

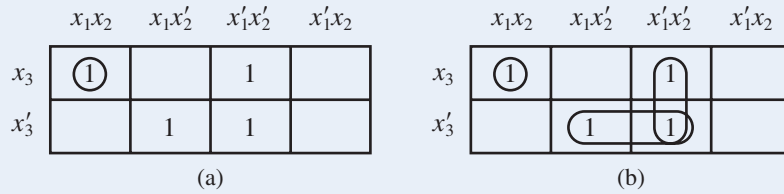


Figure 8.31

EXAMPLE 25

Figure 8.32a shows the unique two-square blocks for the $x_1'x_2'x_3x_4$ square and the $x_1x_2x_3'x_4'$ square. In Figure 8.32b the two unused squares have been combined into a unique four-square block. The minimal sum-of-products expression is

$$x_1x_3 + x_2'x_3x_4 + x_1x_2'x_4'$$

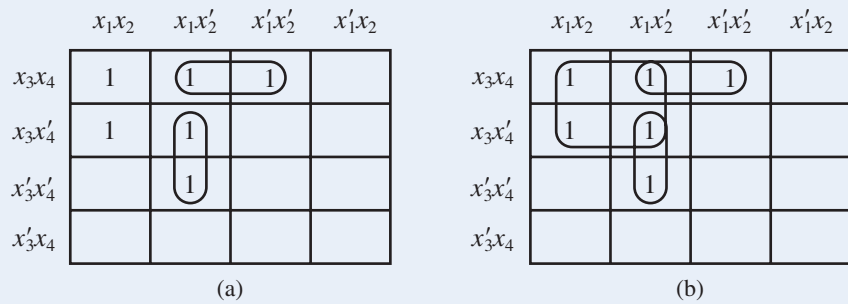


Figure 8.32

EXAMPLE 26

Figure 8.33a shows the unique two-square blocks. We can assign the remaining unused marked square to either of two different two-square blocks; these blocks are shown in Figure 8.33b. There are two minimal sum-of-products forms,

$$x_1x_2'x_4' + x_1'x_2x_3 + x_2'x_3x_4'$$

and

$$x_1x_2'x_4' + x_1'x_2x_3 + x_1'x_3x_4'$$

Either can be used, as they are equally efficient.

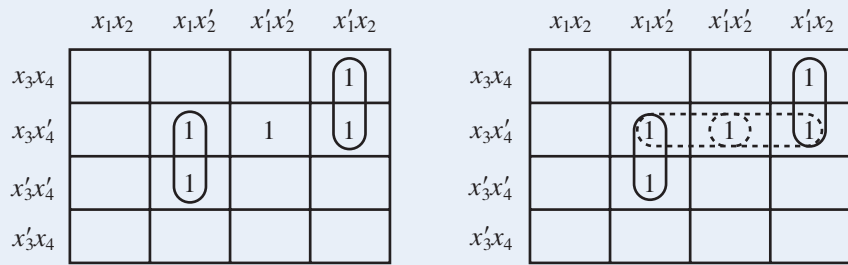


Figure 8.33

EXAMPLE 27

Figure 8.34a shows the unique two-square and four-square blocks. The remaining two unused marked squares can be assigned to two-square blocks in two different ways, as shown in parts (b) and (c). Assigning them together to a single two-square block is more efficient because it produces a sum-of-products form with three terms rather than four. The minimal sum-of-products expression is

$$x_1x_3 + x_1'x_2x_3' + x_2'x_3'x_4'$$

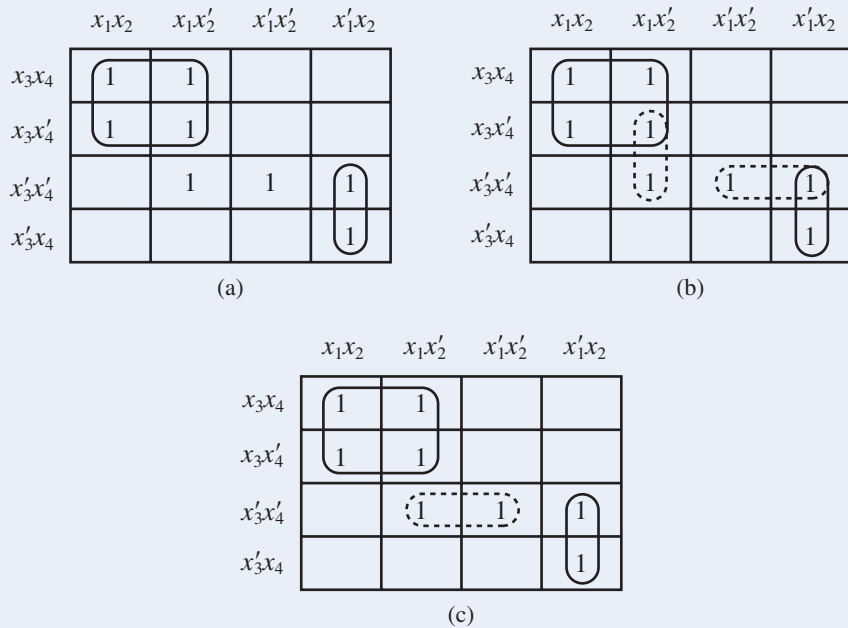


Figure 8.34

EXAMPLE 28

Consider the map of Figure 8.35a. Here the two unique four-square blocks determined by the squares with * have been chosen. In Figure 8.35b, the remaining unmarked squares, for which there was a choice of blocks, are combined into blocks as efficiently as possible. The resulting sum-of-products form is

$$x_1x_3 + x_1'x_3' + x_3x_4 + x_1'x_2 + x_1x_2'x_4'$$

Yet in Figure 8.35c, choosing a different four-square block at the top leads to a simpler sum-of-products form,

$$x_2x_3 + x_1'x_3' + x_3x_4 + x_1x_2'x_4'$$

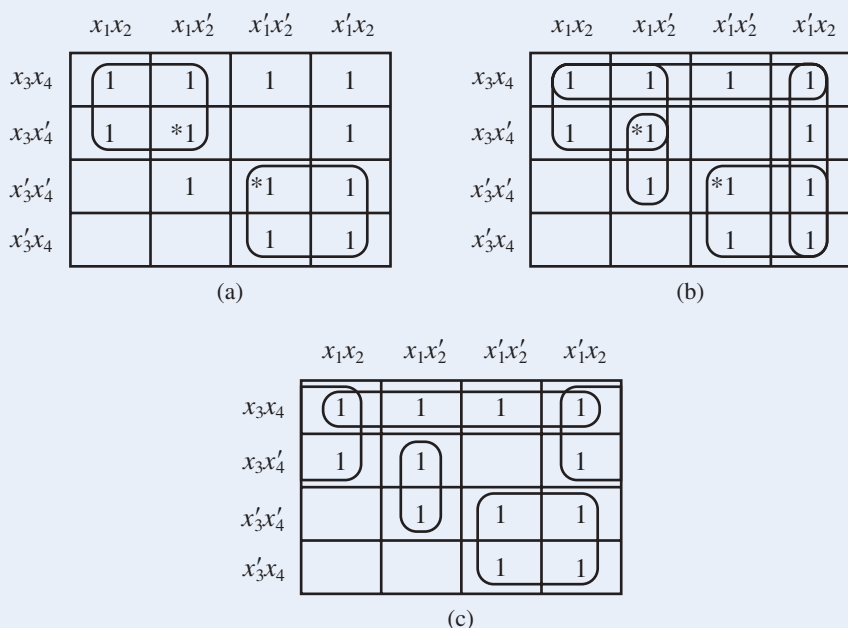


Figure 8.35

PRACTICE 18 Write the minimal sum-of-products expression for the map shown in Figure 8.36.

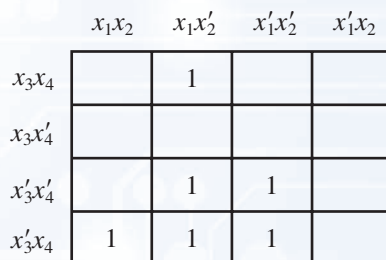


Figure 8.36

If the Karnaugh map corresponds to a function with don't-care conditions, then the don't-care squares on the map can be left blank or assigned the value 1, whichever aids the minimization process.

We have used Karnaugh maps for functions of two, three, and four variables. By using three-dimensional drawings or overlapping transparency sheets, Karnaugh maps for functions of five, six, or even more variables can be constructed, but the visualization gets too complicated to be worthwhile. The next procedure works for any number of variables.

Quine–McCluskey Procedure

Remember that the key to reducing the canonical sum-of-products form for a truth function lies in recognizing terms of the sum that differ in only one factor. In the Karnaugh map, we see where such terms occur. A second method of reduction, the *Quine–McCluskey procedure*, organizes information from the canonical sum-of-products form into a table to simplify the search for terms differing by only one factor.

The procedure is a two-step process paralleling the use of the Karnaugh map. First we find groupings of terms (just as we looped together marked squares in the Karnaugh map); then we eliminate redundant groupings and make choices for terms that can belong to several groups.

EXAMPLE 29

Let's illustrate the Quine–McCluskey procedure by using the truth function for Example 23. We did not write the actual truth function there, but the information is contained in the Karnaugh map. The truth function is shown in Table 8.13. The eight 4-tuples of 0s and 1s producing a function value of 1 are listed in Table 8.14, which is separated into four groupings according to the number of 1s. Note that terms of the canonical sum-of-products form differing by only one factor must be in adjacent groupings, which simplifies the search for such terms.

TABLE 8.13

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
1	1	1	1	0
1	1	1	0	0
1	1	0	1	0
1	1	0	0	0
1	0	1	1	1
1	0	1	0	0
1	0	0	1	0
1	0	0	0	1
0	1	1	1	0
0	1	1	0	1
0	1	0	1	1
0	1	0	0	0
0	0	1	1	1
0	0	1	0	1
0	0	0	1	1
0	0	0	0	1

TABLE 8.14

Number of 1s	x_1	x_2	x_3	x_4
Three	1	0	1	1
Two	0	1	1	0
	0	1	0	1
	0	0	1	1
One	1	0	0	0
	0	0	1	0
	0	0	0	1
None	0	0	0	0

We compare the first term, 1011, with each of the three terms of the second group, 0110, 0101, and 0011, to locate terms differing by only one factor. Such a term is 0011. The combination 1011 and 0011 reduces to -011 when the changing variable x_1 is eliminated. We will write this reduced term with a dash in the x_1 position in the first row of a new table. The new table is Table 8.15b, where we've just seen how the first row was obtained. We've rewritten the original Table 8.14 as Table 8.15a, but we have also marked the two terms 1011 and 0011 in this table with a superscript 1. This superscript 1 is a pointer that indicates the row number of the reduced term in Table 8.15b that is formed from these two terms (numbering terms corresponds to putting loops in the Karnaugh map).

Number of 1s	x_1	x_2	x_3	x_4		Number of 1s	x_1	x_2	x_3	x_4
Three	1	0	1	1	¹	Two	-	0	1	1
Two	0	1	1	0	²	One	0	-	1	0
	0	1	0	1	³		0	-	0	1
	0	0	1	1	^{1, 4, 5}		0	0	1	-
One	1	0	0	0	⁶		0	0	-	1
	0	0	1	0	^{2, 4, 7}	None	-	0	0	0
	0	0	0	1	^{3, 5, 8}		0	0	-	0
None	0	0	0	0	^{6, 7, 8}		0	0	0	-

(a)

(b)

We continue this process with all the terms in Table 8.15a. A numbered term may still be used in other combinations, just as a marked square in a Karnaugh map can be in more than one loop. When we are done, the result is the completed Table 8.15b shown, where the terms in this table are again grouped by the number of 1s.

We now build still another table by processing the terms in Table 8.15b. Here not only the groupings but also the dashes help organize the search process, since terms differing by only one variable must have dashes in the same location. Tables 8.16a and 8.16b are the same as Tables 8.15a and 8.15b, and Table 8.16c is the new table. Again, numbers on terms in Table 8.16b that combine serve as pointers to the reduced terms in Table 8.16c. When we have processed all the terms in Table 8.16b, the reduction process cannot be continued. The unnumbered terms are irreducible, so they represent the possible maximum-sized loops on a Karnaugh map.

TABLE 8.16

Number of 1s	x_1	x_2	x_3	x_4	
Three	1	0	1	1	¹
Two	0	1	1	0	²
	0	1	0	1	³
	0	0	1	1	^{1, 4, 5}
One	1	0	0	0	⁶
	0	0	1	0	^{2, 4, 7}
	0	0	0	1	^{3, 5, 8}
None	0	0	0	0	^{6, 7, 8}

(a)

Number of 1s	x_1	x_2	x_3	x_4	
Two	–	0	1	1	
One	0	–	1	0	
	0	–	0	1	
	0	0	1	–	¹
	0	0	–	1	¹
None	–	0	0	0	
	0	0	–	0	¹
	0	0	0	–	¹

(b)

Number of 1s	x_1	x_2	x_3	x_4
None	0	0	–	–

(c)

For the second step of the process, we compare the original terms with the irreducible terms. We form a table with the original terms as column headers and the irreducible terms (the unnumbered terms in the reduction tables just constructed) as row labels. A check in the comparison table (Table 8.17) indicates that the original term in that column eventually led to the irreducible term in that row, which can be determined by following the pointers.

TABLE 8.17

	1011	0110	0101	0011	1000	0010	0001	0000
–011	✓			✓				
0–10		✓				✓		
0–01			✓				✓	
–000					✓			✓
00–				✓		✓	✓	✓

If a column in the comparison table has a check in only one row, the irreducible term for that row is the only one covering the original term, so it is an essential term and must appear in the final sum-of-products form. Thus, we see from Table 8.17 that the terms –011, 0–10, 0–01, and –000 are essential and must be in the final expression. We also note that all columns with a check in row 5 also have checks in another row and so are covered by an essential reduced term already in the expression. Thus, 00– is redundant. As in Example 23, the minimal sum-of-products form is

$$x_2'x_3x_4 + x_1'x_3x_4' + x_1'x_3'x_4 + x_2'x_3'x_4'$$

In situations where there is more than one minimal sum-of-products form, the comparison table will have nonessential, nonredundant reduced terms. A selection must be made from these reduced terms to cover all columns not covered by essential terms.

EXAMPLE 30

We will use the Quine–McCluskey procedure on the problem presented in Example 26. The reduction tables are given in Table 8.18, and the comparison table appears in Table 8.19.

Number of 1s	x_1	x_2	x_3	x_4		Number of 1s	x_1	x_2	x_3	x_4
Three	0	1	1	1	¹	Two	0	1	1	–
Two	1	0	1	0	^{2,3}	One	–	0	1	0
	0	1	1	0	^{1,4}		1	0	–	0
One	0	0	1	0	^{2,4}		0	–	1	0
	1	0	0	0	³					

(a)

(b)

	0111	1010	0110	0010	1000
011–	✓		✓		
–010		✓		✓	
10–0		✓			✓
0–10			✓	✓	

We see from the comparison table that 011– and 10–0 are essential reduced terms and that there are no redundant terms. The only original term not covered by essential terms is 0010, column 4, and the choice of the reduced term for row 2 or for row 4 will cover it. Thus, the minimal sum-of-products form, as before, is

$$x_1'x_2x_3 + x_1x_2'x_4' + x_2'x_3x_4'$$

or

$$x_1'x_2x_3 + x_1x_2'x_4' + x_1'x_3x_4'$$

PRACTICE 19 Use the Quine–McCluskey procedure to find a minimal sum-of-products form for the truth function in Table 8.20.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	1
0	0	0	1

While the Quine–McCluskey procedure applies to truth functions with any number of input variables, for a large number of variables, the procedure is extremely tedious to do by hand. However, it is exactly the kind of systematic, mechanical process that lends itself to a computerized solution. In contrast, Karnaugh maps make use of the human ability to quickly recognize visual patterns.

If the truth function f has few 0 values and a large number of 1 values, it may be simpler to implement the Quine–McCluskey procedure for the complement of the function, f' , which will have 1 values where f has 0 values, and vice versa. Once a minimal sum-of-products expression is obtained for f' , it can be complemented to obtain an expression for f , although the new expression will not be in sum-of-products form. (In fact, by De Morgan's laws, it will be equivalent to a product-of-sums form.) We can obtain the network for f from the sum-of-products network for f' by tacking an inverter on the end.

The whole object of minimizing a network is to simplify the internal configuration while preserving the external behavior. In Chapter 9 we will attempt the same sort of minimization on finite-state machine structures.

SECTION 8.3 REVIEW

TECHNIQUES

- W Minimize the canonical sum-of-products form for a truth function by using a Karnaugh map.
- W Minimize the canonical sum-of-products form for a truth function by using the Quine–McCluskey procedure.

MAIN IDEA

- Algorithms exist for reducing a canonical sum-of-products form to a minimized sum-of-products form.

EXERCISES 8.3

For Exercises 1–8, write the minimal sum-of-products form for the Karnaugh maps of the given figures.

1.

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3			1	1
x_3'	1	1		1

2.

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3	1			1
x_3'		1		

3.

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3	1	1	1	1
x_3'	1			1

4.

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3				
x_3'	1		1	1

5.

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3x_4		1		
x_3x_4'		1	1	1
$x_3'x_4'$	1	1	1	
$x_3'x_4$		1		

6.

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3x_4				
x_3x_4'	1	1		1
$x_3'x_4'$	1			1
$x_3'x_4$				

7.

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3x_4		1		
x_3x_4'		1	1	1
$x_3'x_4'$				
$x_3'x_4$		1		

8.

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3x_4				1
x_3x_4'	1	1		
$x_3'x_4'$		1	1	
$x_3'x_4$			1	

For Exercises 9 and 10, use a Karnaugh map to find the minimal sum-of-products form for the truth functions shown.

9.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

10.

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
1	1	1	1	1
1	1	1	0	1
1	1	0	1	1
1	1	0	0	1
1	0	1	1	0
1	0	1	0	1
1	0	0	1	0
1	0	0	0	1
0	1	1	1	1
0	1	1	0	1
0	1	0	1	1
0	1	0	0	1
0	0	1	1	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	0

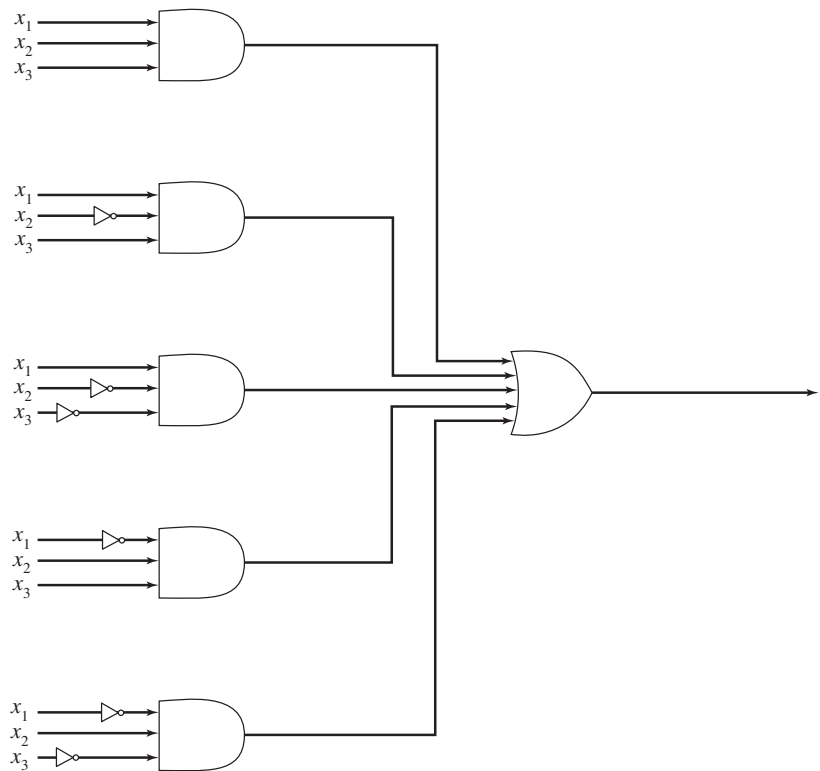
11. Use a Karnaugh map to find the minimal sum-of-products form for the truth function of Exercise 17, Section 8.2.
12. Use a Karnaugh map to find the minimal sum-of-products form for the truth function of Exercise 18, Section 8.2.
13. a. Use a Karnaugh map to find the minimal sum-of-products form for the truth function of Exercise 19, Section 8.2.
b. Draw the logic network for the reduced expression of part (a).
14. a. Use a Karnaugh map to find the minimal sum-of-products form for the truth function of Exercise 20, Section 8.2.
b. Draw the logic network for the reduced expression of part (a).
15. Use a Karnaugh map to find the minimal sum-of-products form for the following Boolean expression.

$$x_1'x_2'x_3x_4 + x_1x_2x_3'x_4 + x_1'x_2x_3'x_4 + x_1x_2'x_3x_4' + x_1'x_2x_3x_4 + x_1'x_2x_3'x_4 + x_1'x_2'x_3x_4'$$

16. Use a Karnaugh map to find the minimal sum-of-products form for the following Boolean expression.

$$x_1'x_2'x_3'x_4' + x_1x_2x_3'x_4 + x_1'x_2'x_3'x_4 + x_1x_2x_3'x_4' + x_1'x_2x_3x_4 + x_1x_2'x_3'x_4'$$

17. Use a Karnaugh map to find a minimal sum-of-products expression for the network of three variables shown in the figure. Sketch the new network.



18. At Rats R Us, you found a standard sum-of-products form for the logic to control valves A and B (Exercise 39, Section 8.2). Now earn yourself a raise by using Karnaugh maps to minimize these expressions.
19. Use a Karnaugh map to find a minimal sum-of-products form for the truth function in the table. Don't-care conditions are shown by dashes.

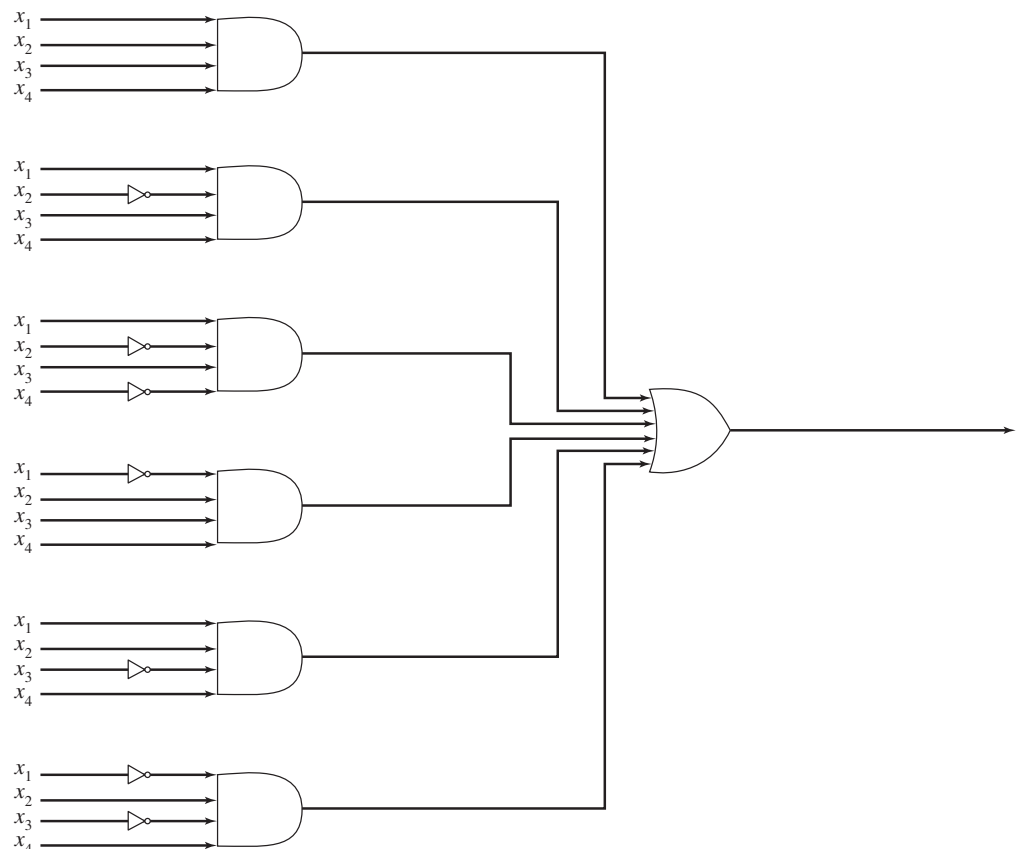
x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
1	1	1	1	0
1	1	1	0	1
1	1	0	1	0
1	1	0	0	–
1	0	1	1	0
1	0	1	0	–
1	0	0	1	0
1	0	0	0	0
0	1	1	1	0
0	1	1	0	1
0	1	0	1	0
0	1	0	0	1
0	0	1	1	1
0	0	1	0	0
0	0	0	1	–
0	0	0	0	0

20. Use a Karnaugh map to find a minimal sum-of-products form for the truth function in the table. Don't-care conditions are shown by dashes.

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
1	1	1	1	0
1	1	1	0	1
1	1	0	1	0
1	1	0	0	–
1	0	1	1	–
1	0	1	0	0
1	0	0	1	0
1	0	0	0	0
0	1	1	1	1
0	1	1	0	0
0	1	0	1	1
0	1	0	0	0
0	0	1	1	1
0	0	1	0	0
0	0	0	1	–
0	0	0	0	0

21. Use the Quine–McCluskey procedure to find a minimal sum-of-products form for the truth function illustrated by the map for Exercise 3.

22. Use the Quine–McCluskey procedure to find a minimal sum-of-products form for the network in the figure. Sketch the new network.



For Exercises 23 and 24, use the Quine–McCluskey procedure to find the minimal sum-of-products form for the truth functions in the given tables.

23.

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
1	1	1	1	0
1	1	1	0	1
1	1	0	1	0
1	1	0	0	0
1	0	1	1	0
1	0	1	0	1
1	0	0	1	1
1	0	0	0	1
0	1	1	1	0
0	1	1	0	0
0	1	0	1	0
0	1	0	0	1
0	0	1	1	1
0	0	1	0	1
0	0	0	1	0
0	0	0	0	1

24.

x_1	x_2	x_3	x_4	$f(x_1, x_2, x_3, x_4)$
1	1	1	1	1
1	1	1	0	0
1	1	0	1	1
1	1	0	0	0
1	0	1	1	1
1	0	1	0	0
1	0	0	1	1
1	0	0	0	1
0	1	1	1	1
0	1	1	0	0
0	1	0	1	1
0	1	0	0	1
0	0	1	1	1
0	0	1	0	0
0	0	0	1	1
0	0	0	0	1

In Exercises 25–28, use the Quine–McCluskey procedure to find the minimal sum-of-products form for the Boolean expressions.

25. $x_1x_2'x_3x_4' + x_1'x_2'x_3x_4 + x_1'x_2x_3x_4 + x_1'x_2'x_3'x_4' + x_1'x_2x_3'x_4' + x_1'x_2'x_3'x_4$

26. $x_1x_2x_3x_4 + x_1x_2'x_3x_4 + x_1x_2x_3x_4' + x_1x_2'x_3x_4' + x_1'x_2x_3x_4' + x_1x_2x_3'x_4' + x_1'x_2x_3'x_4 + x_1'x_2x_3x_4$

27. $x_1x_2x_3x_4 + x_1x_2x_3x_4' + x_1'x_2x_3x_4' + x_1x_2x_3'x_4' + x_1'x_2'x_3'x_4' + x_1'x_2x_3'x_4 + x_1'x_2'x_3x_4 + x_1x_2x_3'x_4$

28. $x_1'x_2x_3'x_4x_5' + x_1'x_2x_3x_4'x_5 + x_1x_2x_3x_4x_5 + x_1'x_2'x_3x_4'x_5 + x_1x_2'x_3x_4x_5 + x_1'x_2'x_3'x_4'x_5 + x_1x_2'x_3x_4'x_5' + x_1x_2x_3'x_4x_5' + x_1'x_2'x_3'x_4x_5'$

29. Use the Quine–McCluskey procedure to find a minimal sum-of-products form for the truth function illustrated by the map in Figure 8.34.

CHAPTER 8 REVIEW

TERMINOLOGY

AND gate (p. 639)	double negation (for a Boolean algebra) (p. 624)	isomorphic instances of a structure (p. 626)
Boolean algebra (p. 620)	dual (of a Boolean algebra property) (p. 623)	isomorphism (p. 626)
Boolean expression (p. 639)	equivalent Boolean expressions (p. 645)	isomorphism for Boolean algebras (p. 629)
canonical sum-of-products form (p. 643)	FPGA (p. 647)	NAND gate (p. 650)
combinational network (p. 642)	full-adder (p. 650)	NOR gate (p. 651)
complement (of a Boolean algebra element) (p. 625)	half-adder (p. 649)	OR gate (p. 639)
De Morgan's laws (for a Boolean algebra) (p. 624)	idempotent property (of a Boolean algebra) (p. 622)	PLD (p. 647)
disjunctive normal form (p. 643)	inverter (p. 639)	truth function (p. 640)
		universal bound property (of a Boolean algebra) (p. 624)

SELF-TEST

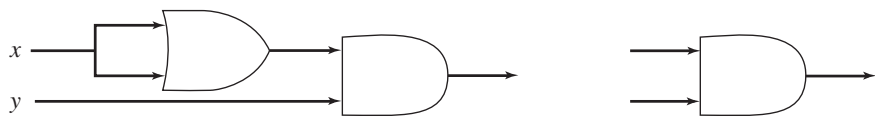
Answer the following true-false questions.

Section 8.1

1. In any Boolean algebra, $x + x' = 0$.
2. Set theory is an instance of a Boolean algebra in which $+$ is set union and \cdot is set intersection.
3. In any Boolean algebra, $x + (y + x \cdot z) = x + y$.
4. The dual of the equation in the previous statement is $x \cdot [y \cdot (x + z)] = x \cdot y$.
5. Any two Boolean algebras with 16 elements are isomorphic.

Section 8.2

1. A logic network for the Boolean expression $(x + y)'$ could be built using one AND gate and two inverters.
2. The canonical sum-of-products form for a truth function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ has n terms.
3. Two single-bit binary numbers can be added using a network consisting of two half-adders.
4. The following two logic networks represent the same truth function:



5. The most efficient way to construct a logic network for a given truth function using only NAND gates is to construct the logic network using AND, OR, and NOT gates and then replace each of these elements with its equivalent form in NAND gates.

Section 8.3

1. A Karnaugh map is a device to help change a canonical sum-of-products form to a reduced sum-of-products form.
2. The 1s in a Karnaugh map correspond to the 1 values of the truth function.
3. When using a Karnaugh map to reduce a Boolean expression, the largest possible blocks should be determined first because they provide the greatest reduction.
4. In the Quine–McCluskey procedure, terms that combine must have dashes in the same locations.
5. In the Quine–McCluskey procedure, a check in some row of the comparison table indicates that the term for that row is an essential term that must appear in the reduced expression.

ON THE COMPUTER

For Exercises 1–4, write a computer program that produces the desired output from the given input.

1. *Input:* n and tables defining two binary operations and one unary operation on a set of n objects
Output: Indication of whether the structure is a Boolean algebra
Algorithm: Testing the 10 properties for all cases
2. *Input:* n , tables defining a binary operation on each of two sets of n elements, and a table defining a bijection from one set to the other
Output: Indication of whether the function is an isomorphism
Algorithm: Testing all possible cases
3. *Input:* n and a table representing a truth function with n arguments
Output: Canonical sum-of-products Boolean expression for the truth function
4. *Input:* n and a table representing a truth function with n arguments
Output: Minimal sum-of-products Boolean expression for the truth function
Algorithm: Using the Quine–McCluskey procedure

Modeling Arithmetic, Computation, and Languages

CHAPTER OBJECTIVES

After studying this chapter, you will be able to:

- See how algebraic structures, finite-state machines, and Turing machines are all models of various kinds of computation, and how formal languages attempt to model natural languages.
- Recognize certain well-known group structures.
- Prove some properties about groups.
- Understand what it means for groups to be isomorphic.
- Be able to construct group codes for single-error correction of binary m -tuples.
- Be able to decode received n -tuples for a single-error correcting perfect code.
- Trace the operation of a given finite-state machine on an input string.
- Construct finite-state machines to recognize certain sets.
- For a given finite-state machine, find an equivalent machine with fewer states if one exists.
- Build a circuit for a finite-state machine.
- Trace the operation of a given Turing machine on an input tape.
- Construct Turing machines to perform certain recognition or computation tasks.
- Understand the Church–Turing thesis and what it implies for the Turing machine as a model of computation.
- Be aware of the $P = NP$ question regarding computational complexity.
- Given a grammar G , construct the derivation of strings in $L(G)$.
- Understand the relationship between different classes of formal languages and different computational devices.

Your team at Babel, Inc., is writing a compiler for a new programming language, currently code-named ScrubOak after a tree outside your office window. During the first phase of compilation (called the lexical analysis phase) the compiler must break down statements into individual units called tokens. In particular, the compiler must be able to recognize identifiers in the language, which are strings of letters, and also recognize the two keywords in the language, which are **if** and **in**.

Question: How can the compiler recognize the individual tokens in a statement?

A mathematical structure, as discussed in Chapter 8, is a formal model intended to capture common properties or behavior found in different contexts. A structure consists of an abstract set of objects, together with operations on or relationships among those objects that obey certain rules. The Boolean algebra structure of Chapter 8 is a model of the properties and behavior common to both propositional logic and set theory. As a formal model, it is an abstract entity, an idea; propositional logic and set theory are two instances, or realizations, of this idea.

In this chapter we study other structures. In Section 9.1, algebraic structures are defined that model various types of arithmetic such as addition of integers and multiplication of positive real numbers. As an aside, section 9.2 looks at coding theory, an important application of one of these algebraic structures. This type of encoding (and decoding) is not about secrecy but about detecting and perhaps correcting bit errors in data transmission or storage.

The “arithmetics” of Section 9.1 represent a limited form of computation, but we will see models of much broader forms of computation in Sections 9.3 and 9.4. Our initial choice for such a model, the finite-state machine, is a useful device for certain tasks, such as the lexical analysis task facing your team at Babel. But the finite-state machine is ultimately too limited to model computation in the general sense. For a model that captures the notion of computation in all its generality, we turn to the Turing machine. Using the Turing machine as a model of computation will reveal that some well-defined tasks are not computable at all.

Finally, Section 9.5 discusses formal grammars and languages, which were developed as attempts to model natural languages such as English. While less than completely successful in this regard, formal grammars and languages do serve to model many constructs in programming languages and play an important role in compiler theory.

SECTION 9.1 ALGEBRAIC STRUCTURES

Definitions and Examples

We begin by analyzing a simple form of arithmetic, namely the addition of integers. There is a set \mathbb{Z} of objects (the integers) and a binary operation on those objects (addition). Recall from Section 4.1 that a binary operation on a set must be *well defined* (giving a unique answer whenever it is applied to any two members of the set) and that the set must be *closed* under the operation (the answer must be a member of the set). The notation $[\mathbb{Z}, +]$ will denote the set together with the binary operation on that set.

In $[\mathbb{Z}, +]$, an equation such as

$$2 + (3 + 5) = (2 + 3) + 5$$

is true. On each side of the equation the integers remain in the same order, but the grouping of those integers, which indicates the order in which the additions are performed, changes. Changing the grouping has no effect on the answer. Another type of equation that holds in $[\mathbb{Z}, +]$ is

$$2 + 3 = 3 + 2$$

Changing the order of the integers being added has no effect on the answer. Equations such as

$$\begin{aligned}2 + 0 &= 2 \\0 + 3 &= 3 \\-125 + 0 &= -125\end{aligned}$$

are also true. Adding zero to any integer does not change the value of that integer. Finally, equations such as

$$\begin{aligned}2 + (-2) &= 0 \\5 + (-5) &= 0 \\-20 + 20 &= 0\end{aligned}$$

are true; adding the negative of an integer to the integer gives 0 as a result.

These equations represent four properties that occur so often they each have a name.

DEFINITIONS PROPERTIES OF BINARY OPERATIONS

Let S be a set and let \cdot denote a binary operation on S . (Here \cdot does *not* necessarily denote multiplication but simply any binary operation.)

1. The operation \cdot is **associative** if

$$(\forall x)(\forall y)(\forall z)[x \cdot (y \cdot z) = (x \cdot y) \cdot z]$$

Associativity allows us to write $x \cdot y \cdot z$ without using parentheses because grouping does not matter.

2. The operation \cdot is **commutative** if

$$(\forall x)(\forall y)(x \cdot y = y \cdot x)$$

3. $[S, \cdot]$ has an **identity element** if

$$(\exists i)(\forall x)(x \cdot i = i \cdot x = x)$$

4. If $[S, \cdot]$ has an identity element i , then each element in S has an **inverse** with respect to \cdot if

$$(\forall x)(\exists x^{-1})(x \cdot x^{-1} = x^{-1} \cdot x = i)$$

In the statements of the properties, the universal quantifiers range over the set S ; if the associative property holds, the equation $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ is true for any $x, y, z \in S$, and similarly for the commutative property. The existential quantifier also applies to the set S , so an identity element i , if it exists, must be an element of S , and an inverse element x^{-1} , if it exists, must be an element of S . Note the order of the quantifiers: In the definition of an identity, the existential quantifier comes first—there must be one identity element i that satisfies the equation $x \cdot i = i \cdot x = x$ for every x in S , just like the integer 0 in $[\mathbb{Z}, +]$. In the definition of the inverse element, the existential quantifier comes second—for each x , there is an x^{-1} , and if x is changed, then x^{-1} can change, just like the inverse of 2 in $[\mathbb{Z}, +]$ is -2 and the inverse of 5 is -5 . If there is no identity element, then it does not make sense to talk about inverse elements.

● **DEFINITIONS** **GROUP, COMMUTATIVE GROUP**

$[S, \cdot]$ is a **group** if S is a nonempty set and \cdot is a binary operation on S such that

1. \cdot is associative.
2. an identity element exists (in S).
3. each element in S has an inverse (in S) with respect to \cdot .

A group in which the operation \cdot is commutative is called a **commutative group**.

Once again, the dot in the definitions is a generic symbol representing a binary operation. In any specific case, the particular binary operation has to be defined. If the operation is addition, for example, then the $+$ sign replaces the generic symbol, as in $[\mathbb{Z}, +]$. As an analogy with programming, we can think of the generic symbol as a formal parameter to be replaced by an actual argument—the specific operation—when its value becomes known. If it is clear what the binary operation is, we may refer to “the group S ” rather than “the group $[S, \cdot]$.”

From the discussion, it should be clear that $[\mathbb{Z}, +]$ is a commutative group, with an identity element of 0. The idea of a group would not be useful if there were not a number of other instances. (Again as an analogy with programming, one can think of a group as an abstract data type—a pattern—with many possible instances of that data type.)

EXAMPLE 1

Let \mathbb{R}^+ denote the positive real numbers, and let \cdot denote real-number multiplication, which is a binary operation on \mathbb{R}^+ . Then $[\mathbb{R}^+, \cdot]$ is a commutative group. Multiplication is associative and commutative. The positive real number 1 serves as an identity because

$$x \cdot 1 = 1 \cdot x = x$$

for every positive real number x . Every positive real number x has an inverse with respect to multiplication, namely the positive real number $1/x$, because

$$x \cdot 1/x = 1/x \cdot x = 1$$

PRACTICE 1

The set in Example 1 is limited to the positive real numbers. Is $[\mathbb{R}, \cdot]$ a commutative group? Why or why not? ■

EXAMPLE 2

Let $M_2(\mathbb{Z})$ denote the set of 2×2 matrices with integer entries, and let $+$ denote matrix addition. Then $+$ is a binary operation on $M_2(\mathbb{Z})$ (note that closure holds). This is a commutative group because the integers are a commutative group, so each corner of the matrix behaves properly. For example, matrix addition is commutative because

$$\begin{aligned} \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} + \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} &= \begin{bmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} \end{bmatrix} \\ &= \begin{bmatrix} b_{1,1} + a_{1,1} & b_{1,2} + a_{1,2} \\ b_{2,1} + a_{2,1} & b_{2,2} + a_{2,2} \end{bmatrix} \\ &= \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} + \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \end{aligned}$$

The matrix

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

is an identity. The matrix

$$\begin{bmatrix} 1 & -4 \\ 2 & 5 \end{bmatrix}$$

is an inverse of the matrix

$$\begin{bmatrix} -1 & 4 \\ -2 & -5 \end{bmatrix}$$

REMINDER

It's important to understand all this new terminology. You can't prove that $P \rightarrow Q$ if you don't know what you're starting with or where you want to go.

A structure called a **monoid** results from dropping the inverse property in the definition of a group; thus a monoid has an associative operation and an identity element, but in a monoid that is not also a group, at least one element has no inverse. A **semigroup** results from dropping the identity property and the inverse property in the definition of a group; thus a semigroup has an associative operation, but in a semigroup that is not also a monoid, no identity element exists. Many familiar forms of arithmetic are instances of semigroups, monoids, and groups.

EXAMPLE 3

Consider $[M_2(\mathbb{Z}), \cdot]$ where \cdot denotes matrix multiplication. Closure holds. It can be shown (Exercise 2) that matrix multiplication is associative. The matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

serves as an identity because

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Thus $[M_2(\mathbb{Z}), \cdot]$ is at least a monoid.

PRACTICE 2 Prove that $[M_2(\mathbb{Z}), \cdot]$ is not a commutative monoid.

PRACTICE 3 Prove that $[M_2(\mathbb{Z}), \cdot]$ is not a group.

Although the requirements for a structure to be a semigroup are relatively modest, not every arithmetic structure qualifies.

PRACTICE 4 Prove that $[\mathbb{Z}, -]$ is not a semigroup, where $-$ denotes integer subtraction. ■

PRACTICE 5 Let S be the set of noninteger rational numbers, and let \cdot denote multiplication. Is $[S, \cdot]$ a semigroup? ■

EXAMPLE 4 Each of the following is an instance of a commutative semigroup. You should be able to verify closure, associativity, and commutativity for each:

$$[\mathbb{N}, +], [\mathbb{N}, \cdot], [\mathbb{Q}, \cdot], [\mathbb{R}^+, +], [\mathbb{R}, +]$$

EXAMPLE 5 For any Boolean algebra $[B, +, \cdot, ', 0, 1]$, $[B, +]$ and $[B, \cdot]$ are commutative semigroups. Therefore for any set S , $[\wp(S), \cup]$ and $[\wp(S), \cap]$ are commutative semigroups. ■

Because the requirements that must be satisfied in going from semigroup to monoid to group keep getting stiffer, we expect some examples to drop out, but those remaining should have richer and more interesting personalities.

PRACTICE 6 Which of the following semigroups are monoids? Name the identities. ■

$$[\mathbb{N}, +], [\mathbb{N}, \cdot], [\mathbb{Q}, \cdot], [\mathbb{R}^+, +], [\mathbb{R}, +], [\wp(S), \cup], [\wp(S), \cap]$$

PRACTICE 7 Which of the monoids from the list in Practice 6 are groups? ■

Next we look at a selection of other examples of semigroups, monoids, and groups where the elements are not just simple numbers or where the operations are less familiar.

EXAMPLE 6 An expression of the form

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$$

where $a_i \in \mathbb{R}$, $i = 0, 1, \dots, n$, and $n \in \mathbb{N}$ is a **polynomial in x with real-number coefficients** (or a **polynomial in x over \mathbb{R}** .) For each i , a_i is the **coefficient** of x^i .

If i is the largest integer greater than 0 for which $a_i \neq 0$, the polynomial is of **degree** i ; if no such i exists, the polynomial is of **zero degree**. Terms with zero coefficients are generally not written. Thus, $\pi x^4 - 2/3x^2 + 5$ is a polynomial of degree 4, and the constant polynomial 6 is of zero degree. The set of all polynomials in x over \mathbb{R} is denoted by $\mathbb{R}[x]$.

We define binary operations of $+$ and \cdot in $\mathbb{R}[x]$ to be the familiar operations of polynomial addition and multiplication. For polynomials $f(x)$ and $g(x)$ members of $\mathbb{R}[x]$, the products $f(x) \cdot g(x)$ and $g(x) \cdot f(x)$ are equal because the coefficients are real numbers, and we can use all the properties of real numbers under multiplication and addition (properties such as commutativity and associativity). Similarly, for $f(x)$, $g(x)$, and $h(x)$ members of $\mathbb{R}[x]$, $(f(x) \cdot g(x)) \cdot h(x) = f(x) \cdot (g(x) \cdot h(x))$. The constant polynomial 1 is an identity because $1 \cdot f(x) = f(x) \cdot 1 = f(x)$ for every $f(x) \in \mathbb{R}[x]$. Thus, $[\mathbb{R}[x], \cdot]$ is a commutative monoid. It fails to be a group because only the nonzero constant polynomials have inverses. For example, there is no polynomial $g(x)$ such that $g(x) \cdot x = x \cdot g(x) = 1$, so the polynomial x has no inverse. (Note that while $x \cdot (1/x) = 1$, $1/x = x^{-1}$ is not a polynomial.) However, $[\mathbb{R}[x], +]$ is a commutative group. ●

PRACTICE 8

- For $f(x), g(x), h(x) \in \mathbb{R}[x]$, write the equations saying that $\mathbb{R}[x]$ under $+$ is commutative and associative.
- What is an identity element in $[\mathbb{R}[x], +]$?
- What is an inverse of $7x^4 - 2x^3 + 4$ in $[\mathbb{R}[x], +]$? ■

Polynomials play a special part in the history of group theory (the study of groups) because much research in group theory was prompted by the very practical problem of solving polynomial equations of the form $f(x) = 0, f(x) \in \mathbb{R}[x]$. The quadratic formula provides an algorithm for finding solutions for every $f(x)$ of degree 2, and the algorithm uses only the algebraic operations of addition, subtraction, multiplication, division, and taking roots. Other such algorithms exist for polynomials of degrees 3 and 4. One of the highlights of abstract algebra is the proof that no algorithm using only these operations exists for every $f(x)$ of degree 5. (Notice that this statement is much stronger than simply saying that no algorithm has yet been found; it says to stop looking for one.)

The next example uses modular arithmetic. You may recall from Section 5.1 (Example 15 and the subsequent discussion) that each computer has some limit on the size of the integers that it can store. Although we would like a computer to be able to exhibit the behavior of $[\mathbb{Z}, +]$, the best we can obtain is some finite approximation. The approximation is achieved by performing addition modulo n . The “answer” to the computation $x + y$ for $x, y \in \mathbb{Z}$ is then either the actual value $x + y$ if this value falls within the limit that can be stored or a remainder value obtained by doing modular arithmetic, which is equivalent to $x + y$ under the equivalence relation of congruence modulo n .

EXAMPLE 7

Let $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$ and define *addition modulo 5*, denoted by $+_5$, on \mathbb{Z}_5 by $x +_5 y = r$, where r is the remainder when $x + y$ is divided by 5. In other words, $x +_5 y = (x + y) \bmod 5$. For example, $1 +_5 2 = 3$ and $3 +_5 4 = 2$. *Multiplication modulo 5* is defined by $x \cdot_5 y = (x \cdot y) \bmod 5$. Thus, $2 \cdot_5 3 = 1$ and $3 \cdot_5 4 = 2$. Then $[\mathbb{Z}_5, +_5]$ is a commutative group, and $[\mathbb{Z}_5, \cdot_5]$ is a commutative monoid. ■

PRACTICE 9

a. Complete the following tables defining $+_5$ and \cdot_5 on \mathbb{Z}_5 .

$+_5$	0	1	2	3	4
0					
1			3		
2					
3					
4					

\cdot_5	0	1	2	3	4
0					
1					
2				1	
3					2
4					

- b. What is an identity in $[\mathbb{Z}_5, +_5]$? In $[\mathbb{Z}_5, \cdot_5]$?
 c. What is an inverse of 2 in $[\mathbb{Z}_5, +_5]$?
 d. Which elements in $[\mathbb{Z}_5, +_5]$ have inverses? ■

As we did on \mathbb{Z}_5 , we can define operations of **addition modulo n** and **multiplication modulo n** on the set $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$ where n is any positive integer. Again $[\mathbb{Z}_n, +_n]$ is a commutative group and $[\mathbb{Z}_n, \cdot_n]$ is a commutative monoid. (See Exercise 47 for the relationship between the group $[\mathbb{Z}_n, +_n]$ and the set of equivalence classes under the binary relation of congruence modulo n .)

PRACTICE 10

- a. Give the table for \cdot_6 on \mathbb{Z}_6 .
 b. Which elements in $[\mathbb{Z}_6, \cdot_6]$ have inverses? ■

Notice that when we use a table to define an operation on a finite set, it is easy to check for commutativity by looking for symmetry around the main diagonal. It is also easy to find an identity element because its row looks like the top of the table and its column looks like the side. And it is easy to locate an inverse of an element. Look along the row until you find a column where the identity appears; then check to see that changing the order of the elements still gives the identity. However, associativity (or the lack of it) is not immediately apparent from the table.

EXAMPLE 8

Let Z_2^n be the set of all binary n -tuples with $n \geq 1$. We'll usually use a little shorthand and write, for example, 1101 instead of $(1, 1, 0, 1)$. Using componentwise addition modulo 2, $+_2$, it's easy to see that the addition operation is associative and commutative and that the n -tuple of all 0s is an identity. Each n -tuple has an inverse. Therefore $[Z_2^n, +_2]$ is a commutative group. ●

REMINDER

In the group $[Z_2, +_2]$, the set is $\{0, 1\}$ and the operation is addition modulo 2. In the group $[Z_2^n, +_2]$ for any n , the set is all binary n -tuples and the operation is componentwise addition modulo 2.

PRACTICE 11 In the group $[Z_2^5, +_2]$, what is

- $01101 +_2 11011?$
- $-10100?$

The next two examples give us algebraic structures where the elements are *functions*, mappings from a domain to a codomain.

EXAMPLE 9

Let A be a set and consider the set S of all functions f such that $f: A \rightarrow A$. The binary operation is function composition, denoted by \circ . Note that S is closed under \circ and that function composition is associative (see Practice 12). Thus $[S, \circ]$ is a semigroup, called the **semigroup of transformations on A** . Actually $[S, \circ]$ is a monoid because the identity function i_A that takes each member of A to itself has the property that for any $f \in S$,

$$f \circ i_A = i_A \circ f = f$$

PRACTICE 12 Prove that function composition on the set S just defined is associative. ■

EXAMPLE 10

Again let A be a set and consider the set S_A of all bijections f such that $f: A \rightarrow A$ (permutations of A). Bijectionness is preserved under function composition, function composition is associative, the identity function i_A is a permutation, and for any $f \in S_A$, the inverse function f^{-1} exists and is a permutation. Furthermore,

$$f \circ f^{-1} = f^{-1} \circ f = i_A$$

Thus, $[S_A, \circ]$ is a group, called the **group of permutations on A** . ●

If $A = \{1, 2, \dots, n\}$ for some positive integer n , then S_A is called the **symmetric group of degree n** and denoted by S_n . Thus, S_3 , for example, is the set of all permutations on $\{1, 2, 3\}$. There are $3! = 6$ such permutations, which we name as follows (using the cycle notation of Section 5.4):

$$\begin{array}{lll} \alpha_1 = i & \alpha_2 = (1, 2) & \alpha_3 = (1, 3) \\ \alpha_4 = (2, 3) & \alpha_5 = (1, 2, 3) & \alpha_6 = (1, 3, 2) \end{array}$$

Recall that the notation $(1, 2)$, for example, means that 1 maps to 2, 2 maps to 1, and unnamed elements map to themselves. The composition $(1, 2) \circ (1, 3)$ is done from right to left, so

By $(1, 3)$ By $(1, 2)$

$$1 \rightarrow 3 \rightarrow 3$$

$$2 \rightarrow 2 \rightarrow 1$$

$$3 \rightarrow 1 \rightarrow 2$$

resulting in $(1, 3, 2)$. Thus $\alpha_2 \circ \alpha_3 = (1, 2) \circ (1, 3) = (1, 3, 2) = \alpha_6$.

PRACTICE 13

a. Complete the group table for $[S_3, \circ]$.

\circ	α_1	α_2	α_3	α_4	α_5	α_6
α_1						
α_2			α_6			
α_3						
α_4						
α_5						
α_6						

b. Is $[S_3, \circ]$ a commutative group? ■

$[S_3, \circ]$ is our first example of a noncommutative group (although $[M_2(\mathbb{Z}), \cdot]$ was a noncommutative monoid).

The next example is very simple but particularly appropriate because it appears in several areas of computer science, including formal language theory and automata theory.

EXAMPLE 11

Let A be a finite set; its elements are called **symbols** and A itself is called an **alphabet**. A^* denotes the set of all finite-length **strings**, or **words**, over A . A^* can be defined recursively (as in Example 6 in Chapter 3), where \cdot denotes **concatenation** of strings:

- The **empty string** λ (the string with no symbols) belongs to A^* .
- Any single member of A belongs to A^* .
- If x and y are strings in A^* , so is $x \cdot y$.

Thus, if $A = \{a, b\}$, then $abbaa$, $bbbbba$, and a are all strings over A , and $abbaa \cdot a$ gives the string $abbaaa$. From the recursive definition, any string over A contains only a finite number of symbols. The number of symbols in a string is called its **length**. The empty string λ is the only zero-length string.

The empty string λ should not be confused with the empty set \emptyset ; even if A itself is \emptyset , then $A^* = \{\lambda\}$. If A is nonempty, then whatever the size of A , A^* is a denumerable (countably infinite) set. If A contains only one element, say $A = \{a\}$, then $\lambda, a, aa, aaa, \dots$, is an enumeration of A^* . If A contains more than one element, then a lexicographical (alphabetical) ordering can be imposed on the elements of A . An enumeration of A^* is then obtained by counting the empty string first, then lexicographically ordering all strings of length 1 (there is a finite number of these), then lexicographically ordering all strings of length 2 (there is a finite number of these), and so forth. Note also that if A is nonempty, strings of arbitrary length can be found in A^* .

Concatenation is a binary operation on A^* , and it is associative. The empty string λ is an identity because for any string $x \in A^*$,

$$x \cdot \lambda = \lambda \cdot x = x$$

Therefore, $[A^*, \cdot]$ is a monoid, called the **free monoid generated by A** . •

PRACTICE 14 For $A = \{a, b\}$

- Is $[A^*, \cdot]$ a commutative monoid?
- Is $[A^*, \cdot]$ a group?

Basic Results about Groups

We will now prove some basic theorems about groups. There are hundreds of theorems about groups and many books devoted exclusively to group theory, so we are barely scratching the surface here. The results we will prove follow almost immediately from the definitions involved.

By definition, a group $[G, \cdot]$ (or a monoid) has an identity element, and we have tried to be careful to refer to *an* identity element rather than *the* identity element. However, it is legal to say *the* identity because there is only one. To prove that the identity element is unique, suppose that i_1 and i_2 are both identity elements. Then

$$i_1 = i_1 \cdot i_2 = i_2$$

REMINDER

To prove that something is unique ...

PRACTICE 15 Justify the foregoing equality signs. ■

Because $i_1 = i_2$, the identity element is unique. Thus, we have proved the following theorem.

• **THEOREM ON THE UNIQUENESS OF THE IDENTITY IN A GROUP**
In any group (or monoid) $[G, \cdot]$, the identity element i is unique.

Each element x in a group $[G, \cdot]$ has an inverse element x^{-1} . Therefore, G contains many different inverse elements, but for each x , the inverse is unique.

THEOREM ON THE UNIQUENESS OF INVERSES IN A GROUP

For each x in a group $[G, \cdot]$, x^{-1} is unique.

PRACTICE 16

Prove the preceding theorem. (*Hint*: Assume two inverses for x , namely y and z , and let i be the identity. Then $y = y \cdot i = y \cdot (x \cdot z) = \dots$)

If x and y belong to a group $[G, \cdot]$, then $x \cdot y$ belongs to G and must have an inverse element in G . Naturally, we expect that inverse to have some connection with x^{-1} and y^{-1} , which we know exist in G . We can show that $(x \cdot y)^{-1} = y^{-1} \cdot x^{-1}$; thus the inverse of a product is the product of the inverses in reverse order.

THEOREM ON THE INVERSE OF A PRODUCT

For x and y members of a group $[G, \cdot]$, $(x \cdot y)^{-1} = y^{-1} \cdot x^{-1}$.

REMINDER

If it walks like a duck ...

Proof: We will show that $y^{-1} \cdot x^{-1}$ has the two properties required of $(x \cdot y)^{-1}$. Then, because inverses are unique, $(y^{-1} \cdot x^{-1})$ must be $(x \cdot y)^{-1}$.

$$\begin{aligned} (x \cdot y) \cdot (y^{-1} \cdot x^{-1}) &= x \cdot (y \cdot y^{-1}) \cdot x^{-1} \\ &= x \cdot i \cdot x^{-1} \\ &= x \cdot x^{-1} \\ &= i \end{aligned}$$

Similarly, $(y^{-1} \cdot x^{-1}) \cdot (x \cdot y) = i$. Notice how associativity and the meaning of i and inverses all come into play in this proof. *End of Proof*.

PRACTICE 17

Write 10 as $7 +_{12} 3$ and use the theorem on the inverse of a product to find $(10)^{-1}$ in the group $[\mathbb{Z}_{12}, +_{12}]$.

We know that many familiar number systems such as $[\mathbb{Z}, +]$ and $[\mathbb{R}, +]$ are groups. We make use of group properties when we do arithmetic or algebra in these systems. In $[\mathbb{Z}, +]$, for example, if we see the equation $x + 5 = y + 5$, we conclude that $x = y$. We are making use of the right cancellation law, which, we will soon see, holds in any group.

DEFINITION CANCELLATION LAWS

A set S with a binary operation \cdot satisfies the **right cancellation law** if for $x, y, z \in S$, $x \cdot z = y \cdot z$ implies $x = y$. It satisfies the **left cancellation law** if $z \cdot x = z \cdot y$ implies $x = y$.

Suppose that $x, y,$ and z are members of a group $[G, \cdot]$ and that $x \cdot z = y \cdot z$. To conclude that $x = y$, we take advantage of z^{-1} . Thus,

$$x \cdot z = y \cdot z$$

implies

$$(x \cdot z) \cdot z^{-1} = (y \cdot z) \cdot z^{-1}$$

$$x \cdot (z \cdot z^{-1}) = y \cdot (z \cdot z^{-1})$$

$$x \cdot i = y \cdot i$$

$$x = y$$

Hence, G satisfies the right cancellation law.

PRACTICE 18 Show that any group $[G, \cdot]$ satisfies the left cancellation law. ■

We have proved the following theorem.

THEOREM ON CANCELLATION IN A GROUP
Any group $[G, \cdot]$ satisfies the left and right cancellation laws.

EXAMPLE 12 We know that $[\mathbb{Z}_6, \cdot_6]$ is not a group. Here the equation

$$4 \cdot_6 2 = 1 \cdot_6 2$$

holds, but of course $4 \neq 1$. ●

Again, working in $[\mathbb{Z}, +]$, we would solve the equation $6 + x = 13$ by adding -6 to both sides, producing a unique answer of $x = (-6) + 13 = 7$. The property of being able to solve linear equations for unique solutions holds in all groups. Consider the equation $a \cdot x = b$ in the group $[G, \cdot]$ where a and b belong to G and x is to be found. Then $x = a^{-1} \cdot b$ is an element of G satisfying the equation. Should x_1 and x_2 both be solutions to the equation $ax = b$, then $a \cdot x_1 = a \cdot x_2$ and, by left cancellation, $x_1 = x_2$. Similarly, the unique solution to $x \cdot a = b$ is $x = b \cdot a^{-1}$.

THEOREM ON SOLVING LINEAR EQUATIONS IN A GROUP
Let a and b be any members of a group $[G, \cdot]$. Then the linear equations $a \cdot x = b$ and $x \cdot a = b$ have unique solutions in G .

PRACTICE 19 Solve the equation $x +_8 3 = 1$ in $[\mathbb{Z}_8, +_8]$. ■

The theorem on solving linear equations tells us something about tables for finite groups. As we look along row a of the group operation table, does element b

appear twice? If so, then the table says that there are two distinct elements x_1 and x_2 of the group such that $a \cdot x_1 = b$ and $a \cdot x_2 = b$. But by the theorem on solving linear equations, this double occurrence cannot happen. Thus, a given element of a finite group appears at most once in a given row of the group table. However, to complete the row, each element must appear at least once. A similar result holds for columns. Therefore, in a group table, each element appears exactly once in each row and each column. This property alone, however, is not sufficient to insure that a table represents a group; the operation must also be associative (see Exercise 31).

PRACTICE 20

Assume that \circ is an associative binary operation on $\{1, a, b, c, d\}$. Complete the following table to define a group with identity 1,

\circ	1	a	b	c	d
1	1				
a			c	d	1
b		c	d		
c		d		a	
d				b	c

If $[G, \cdot]$ is a group where G is finite with n elements, then n is said to be the **order of the group**, denoted by $|G|$. If G is an infinite set, the group is of infinite order.

PRACTICE 21

- Name a commutative group of order 18.
- Name a noncommutative group of order 6.

More properties of groups appear in the exercises at the end of this section.

Subgroups

We know what groups are and we know what subsets are, so it should not be hard to guess what a subgroup is. However, we will look at an example before we give the definition. We know that $[\mathbb{Z}, +]$ is a group. Now let A be any nonempty subset of \mathbb{Z} . For any x and y in A , x and y are also in \mathbb{Z} , so $x + y$ exists and is unique. The set A “inherits” a well-defined operation, $+$, from $[\mathbb{Z}, +]$. The associativity property is also inherited, because for any $x, y, z \in A$, it is also true that $x, y, z \in \mathbb{Z}$ and the equation

$$(x + y) + z = x + (y + z)$$

holds. Perhaps A under the inherited operation has all the structure of $[\mathbb{Z}, +]$ and is itself a group. Whether this is true depends on A .

Suppose that $A = E$, the set of even integers. E is closed under addition, E contains 0 (the identity element), and the inverse of every even integer (its negative) is an even integer. $[E, +]$ is thus a group. But suppose that $A = O$, the set of odd integers. $[O, +]$ fails to be a group for several reasons. For one thing, it is not closed—adding two odd integers produces an even integer. (Closure depends on the set as well as the operation, so it is not an inherited property). For another, a subgroup must have an identity with respect to addition; 0 is the only integer that will serve, and 0 is not an odd integer.

● **DEFINITION SUBGROUP**

Let $[G, \cdot]$ be a group and $A \subseteq G$. Then $[A, \cdot]$ is a **subgroup** of $[G, \cdot]$ if $[A, \cdot]$ is itself a group.

For $[A, \cdot]$ to be a group, it must have an identity element, which we'll denote by i_A . Of course G also has an identity element, which we'll denote by i_G . It turns out that $i_A = i_G$, but this equation does not follow from the uniqueness of a group identity because the element i_A , as far as we know, may not be an identity for all of G , and we cannot yet say that i_G is an element of A . However, $i_A = i_A \cdot i_A$ because i_A is the identity for $[A, \cdot]$, and $i_A = i_A \cdot i_G$ because i_G is the identity for $[G, \cdot]$. Because of the left cancellation law holding in the group $[G, \cdot]$, it follows that $i_A = i_G$.

To test whether $[A, \cdot]$ is a subgroup of $[G, \cdot]$, we can assume the inherited properties of a well-defined operation and associativity, and we check for the three remaining properties required.

● **THEOREM ON SUBGROUPS**

For $[G, \cdot]$ a group with identity i and $A \subseteq G$, $[A, \cdot]$ is a subgroup of $[G, \cdot]$ if it meets the following three tests:

1. A is closed under \cdot .
2. $i \in A$.
3. Every $x \in A$ has an inverse element in A .

PRACTICE 22

The definition of a group requires that the set be nonempty. In the theorem on subgroups, why isn't there a specific test that $A \neq \emptyset$?

EXAMPLE 13

- a. $[\mathbb{Z}, +]$ is a subgroup of the group $[\mathbb{R}, +]$. \mathbb{Z} is closed under addition, $0 \in \mathbb{Z}$, and the negative of every integer is an integer.
- b. $[\{1, 4\}, \cdot_5]$ is a subgroup of the group $[\{1, 2, 3, 4\}, \cdot_5]$. Closure holds:

\cdot_5	1	4
1	1	4
4	4	1

The identity $1 \in \{1, 4\}$, and $1^{-1} = 1, 4^{-1} = 4$.

PRACTICE 23

- a. Show that $[\{0, 2, 4, 6\}, +_8]$ is a subgroup of the group $[\mathbb{Z}_8, +_8]$.
 b. Show that $[\{1, 2, 4\}, \cdot_7]$ is a subgroup of the group $[\{1, 2, 3, 4, 5, 6\}, \cdot_7]$. ■

If $[G, \cdot]$ is a group with identity i , then it is true that $[\{i\}, \cdot]$ and $[G, \cdot]$ are subgroups of $[G, \cdot]$. These somewhat trivial subgroups of $[G, \cdot]$ are called **improper subgroups**. Any other subgroups of $[G, \cdot]$ are **proper subgroups**.

PRACTICE 24

Find all the proper subgroups of S_3 , the symmetric group of degree 3. (You can find them by looking at the group table; see Practice 13.) ■

One point of confusing terminology: The set of *all* bijections on a set A into itself under function composition (like S_3) is called *the group of permutations on A* , and any subgroup of this set (such as those in Practice 24) is called a **permutation group**. The distinction is that *the* group of permutations on a set A includes all bijections on A into itself, but *a* permutation group may not. Permutation groups are of particular importance, not only because they were the first groups to be studied, but also because they are the only groups if we consider isomorphic structures to be the same. We will see this result shortly.

There is an interesting subgroup we can always find in the symmetric group S_n for $n > 1$. We know that every member of S_n can be written as a composition of cycles, but it is also true that each cycle can be written as the composition of cycles of length 2, called **transpositions**. In S_7 , for example, $(5, 1, 7, 2, 3, 6) = (5, 6) \circ (5, 3) \circ (5, 2) \circ (5, 7) \circ (5, 1)$. We can verify this by computing $(5, 6) \circ (5, 3) \circ (5, 2) \circ (5, 7) \circ (5, 1)$. Working from right to left,

$$1 \rightarrow 5 \rightarrow 7 \rightarrow 7 \rightarrow 7 \rightarrow 7$$

so 1 maps to 7. Similarly,

$$7 \rightarrow 7 \rightarrow 5 \rightarrow 2 \rightarrow 2 \rightarrow 2$$

so 7 maps to 2, and so on, resulting in $(5, 1, 7, 2, 3, 6)$. It is also true that $(5, 1, 7, 2, 3, 6) = (1, 5) \circ (1, 6) \circ (1, 3) \circ (1, 2) \circ (2, 4) \circ (1, 7) \circ (4, 2)$.

For any $n > 1$, the identity permutation i in S_n can be written as $i = (a, b) \circ (a, b)$ for any two elements a and b in the set $\{1, 2, \dots, n\}$. This equation also shows that the inverse of the transposition (a, b) in S_n is (a, b) . Now we borrow (without proof) one more fact: Even though there are various ways to write a cycle as the composition of transpositions, for a given cycle the number of transpositions will either always be even or always be odd. Consequently, we classify any permutation in S_n , $n > 1$, as **even** or **odd** according to the number of transpositions in any representation of that permutation. For example, in S_7 , $(5, 1, 7, 2, 3, 6)$ is odd. If we denote by A_n the set of all even permutations in S_n , then A_n determines a subgroup of $[S, \circ]$. The composition of even permutations produces an even permutation, and $i \in A_n$. If $\alpha \in A_n$, and α as a product of transpositions is $\alpha = \alpha_1 \circ \alpha_2 \circ \dots \circ \alpha_k$, then $\alpha^{-1} = \alpha_k^{-1} \circ \alpha_{k-1}^{-1} \circ \dots \circ \alpha_1^{-1}$. Each inverse of a transposition is a transposition, so α^{-1} is also even.

The order of the group $[S_n, \circ]$ (the number of elements) is $n!$ What is the order of the subgroup $[A, \circ]$? We might expect half the permutations in S_n to be even and half to be odd. Indeed, this is the case. If we let O_n denote the set of odd permutations in S_n (which is not closed under function composition), then the mapping $f: A_n \rightarrow O_n$ defined by $f(\alpha) = \alpha \circ (1, 2)$ is a bijection.

PRACTICE 25 Prove that $f: A_n \rightarrow O_n$, given by $f(\alpha) = \alpha \circ (1, 2)$ is one-to-one and onto. ■

Because there is a bijection from A_n onto O_n , each set has the same number of elements. But $A_n \cap O_n = \emptyset$ and $A_n \cup O_n = S_n$, so $|A_n| = |S_n|/2 = n!/2$.

● **THEOREM ON ALTERNATING GROUPS**

For $n \in \mathbb{N}$, $n > 1$, the set A_n of even permutations determines a subgroup, called the **alternating group**, of $[S_n, \circ]$ of order $n!/2$.

We have now seen several examples of subgroups of finite groups. In Example 13b and Practice 23, there were three such examples, and the orders of the groups and subgroups were

Group of order 4, subgroup of order 2

Group of order 8, subgroup of order 4

Group of order 6, subgroup of order 3

The theorem on alternating groups says that a particular group of order $n!$ has a subgroup of order $n!/2$.

Based on these examples, one might conclude that subgroups are always half the size of the parent group. This conclusion is not always true, but there is a relationship between the size of a group and the size of a subgroup. This relationship is stated in Lagrange's theorem, proved by the great French mathematician Joseph-Louis Lagrange in 1771 (we will omit the proof here).

● **THEOREM LAGRANGE'S THEOREM**

The order of a subgroup of a finite group divides the order of the group.

Lagrange's theorem helps us narrow down the possibilities for subgroups of a finite group. If $|G| = 12$, for example, we would not look for any subgroups of order 7 because 7 does not divide 12. Also, the fact that 6 divides 12 does not imply the existence of a subgroup of G of order 6. In fact, A_4 is a group of order $4!/2 = 12$, but it can be shown that A_4 has no subgroups of order 6. Therefore the converse to Lagrange's theorem does not always hold. In certain cases the converse can be shown to be true—for example, in finite commutative groups (note that A_4 is not commutative).

Finally, we consider subgroups of the group $[\mathbb{Z}, +]$. For n any fixed element of \mathbb{N} , the set $n\mathbb{Z}$ is defined as the set of all integral multiples of n ; $n\mathbb{Z} = \{nz \mid z \in \mathbb{Z}\}$. Thus, for example, $3\mathbb{Z} = \{0, \pm 3, \pm 6, \pm 9, \dots\}$.

PRACTICE 26 Show that for any $n \in \mathbb{N}$, $[n\mathbb{Z}, +]$ is a subgroup of $[\mathbb{Z}, +]$. ■

Not only is $[n\mathbb{Z}, +]$ a subgroup of $[\mathbb{Z}, +]$ for any fixed n , but sets of the form $n\mathbb{Z}$ are the only subgroups of $[\mathbb{Z}, +]$. To illustrate, let $[S, +]$ be any subgroup of $[\mathbb{Z}, +]$. If $S = \{0\}$, then $S = 0\mathbb{Z}$. If $S \neq \{0\}$, let m be a member of S , $m \neq 0$. Either m is positive or, if m is negative, $-m \in S$ and $-m$ is positive. The subgroup S , therefore, contains at least one positive integer. Let n be the smallest positive integer in S (which exists by the principle of well-ordering). We will now see that $S = n\mathbb{Z}$.

First, since 0 , n , and $-n$ are members of S and S is closed under $+$, $n\mathbb{Z} \subseteq S$. To obtain inclusion in the other direction, let $s \in S$. Now we divide the integer s by the integer n to get an integer quotient q and an integer remainder r with $0 \leq r < n$. Thus, $s = nq + r$. Solving for r , $r = s + (-nq)$. But $nq \in S$; therefore $-nq \in S$, and $s \in S$, so by closure of S under $+$, $r \in S$. If r is positive, we have a contradiction of the definition of n as the smallest positive number in S . Therefore, $r = 0$ and $s = nq + r = nq$. We now have $S \subseteq n\mathbb{Z}$, and thus $S = n\mathbb{Z}$, which completes the proof of the following theorem.

● **THEOREM ON SUBGROUPS OF $[\mathbb{Z}, +]$**
Subgroups of the form $[n\mathbb{Z}, +]$ for $n \in \mathbb{N}$ are the only subgroups of $[\mathbb{Z}, +]$.

Isomorphic Groups

Suppose that $[S, \cdot]$ and $[T, +]$ are isomorphic groups; what would this mean? From the discussion of isomorphism in Section 8.1, isomorphic structures are the same except for relabeling. There must be a bijection from S to T that accomplishes the relabeling. This bijection must also preserve the effects of the binary operation; that is, it must be true that “operate and map” yields the same result as “map and operate.” The following definition is more precise.

● **DEFINITION GROUP ISOMORPHISM**
Let $[S, \cdot]$ and $[T, +]$ be groups. A mapping $f: S \rightarrow T$ is an **isomorphism** from $[S, \cdot]$ to $[T, +]$ if

1. the function f is a bijection.
2. for all $x, y \in S$, $f(x \cdot y) = f(x) + f(y)$.

Property (2) is expressed by saying that f is a **homomorphism**.

PRACTICE 27 Illustrate the homomorphism property of the definition of group isomorphism by a commutative diagram. ■

If isomorphic groups are really the same except for the relabeling accomplished by the bijection, then we would expect that the identity of one group maps

to the identity of the other, that inverses map to inverses, and that if one group is commutative, so is the other. Indeed, we can prove that these expectations are correct. (The proofs do not make use of the one-to-one property of the isomorphism, so an onto homomorphism also maps the identity to the identity, inverses to inverses, and preserves commutativity.)

Suppose, then, that f is an isomorphism from the group $[S, \cdot]$ to the group $[T, +]$ and that i_S and i_T are the identities in the respective groups. Under the function f , i_S maps to an element $f(i_S)$ in T . Let t be any element in T . Then, because f is an onto function, $t = f(s)$ for some $s \in S$. It follows that

$$\begin{aligned} f(i_S) + t &= f(i_S) + f(s) \\ &= f(i_S \cdot s) && \text{(because } f \text{ is a homomorphism)} \\ &= f(s) && \text{(because } i_S \text{ is the identity in } S) \\ &= t \end{aligned}$$

Therefore

$$f(i_S) + t = t$$

Similarly,

$$t + f(i_S) = t$$

The element $f(i_S)$ acts like an identity element in $[T, +]$, and because the identity is unique, $f(i_S) = i_T$.

PRACTICE 28

Prove that if f is an isomorphism from the group $[S, \cdot]$ to the group $[T, +]$, then for any $s \in S$, $f(s^{-1}) = -f(s)$ (inverses map to inverses). (*Hint:* Show that $f(s^{-1})$ acts like the inverse of $f(s)$.)

PRACTICE 29

Prove that if f is an isomorphism from the commutative group $[S, \cdot]$ to the group $[T, +]$, then $[T, +]$ is a commutative group.

EXAMPLE 14

$[\mathbb{R}^+, \cdot]$ and $[\mathbb{R}, +]$ are both groups. Let b be a positive real number, $b \neq 1$, and let f be the function from \mathbb{R}^+ to \mathbb{R} defined by

$$f(x) = \log_b x$$

Then f is an isomorphism. To prove it, we must show that f is a bijection (one-to-one and onto) and that f is a homomorphism (preserves the operation). We can show that f is onto: For $r \in \mathbb{R}$, $b^r \in \mathbb{R}^+$ and $f(b^r) = \log_b b^r = r$. Also, f is one-to-one: If $f(x_1) = f(x_2)$, then $\log_b x_1 = \log_b x_2$. Let $p = \log_b x_1 = \log_b x_2$. Then $b^p = x_1$ and $b^p = x_2$, so $x_1 = x_2$. Finally f is a homomorphism: For $x_1, x_2 \in \mathbb{R}^+$, $f(x_1 \cdot x_2) = \log_b(x_1 \cdot x_2) = \log_b x_1 + \log_b x_2 = f(x_1) + f(x_2)$. Note that $\log_b 1 = 0$, so f maps 1, the identity of $[\mathbb{R}^+, \cdot]$ to 0, the identity of $[\mathbb{R}, +]$.

Also note that

$$\log_b(1/x) = \log_b 1 - \log_b x = 0 - \log_b x = -\log_b x = -f(x)$$

so f maps the inverse of x in $[\mathbb{R}^+, \cdot]$ to the inverse of $f(x)$ in $[\mathbb{R}, +]$. Finally, both groups are commutative. ●

Because the two groups in Example 14 are isomorphic, each is the mirror image of the other, and each can be used to simulate a computation in the other. Suppose, for example, that $b = 2$. Then $[\mathbb{R}, +]$ can be used to simulate the computation $64 \cdot 512$ in $[\mathbb{R}^+, \cdot]$. First, map from \mathbb{R}^+ to \mathbb{R} :

$$\begin{aligned} f(64) &= \log_2 64 = 6 \\ f(512) &= \log_2 512 = 9 \end{aligned}$$

Now in $[\mathbb{R}, +]$ perform the computation

$$6 + 9 = 15$$

Finally, use f^{-1} to map back to \mathbb{R}^+ :

$$f^{-1}(15) = 2^{15} = 32,768$$

(In the age BC—before calculators and computers—large numbers were multiplied by using tables of common logarithms, where $b = 10$, to convert a multiplication problem to an addition problem, as addition is less prone to human error.) Either of two isomorphic groups can always simulate computations in the other, just as in Example 14.

EXAMPLE 15

Consider the two groups $[S, \cdot]$ and $[T, +]$ as defined by the following tables:

\cdot	2	5	9
2	9	2	5
5	2	5	9
9	5	9	2

$+$	0	1	4
0	0	1	4
1	1	4	0
4	4	0	1

Both are groups of order 3, so an isomorphism is certainly possible. If f is to be an isomorphism, it must map i_S to i_T . Looking at the operation tables, $i_S = 5$ and $i_T = 0$, so let $f(5) = 0$. As a guess, let $f(2) = 1$ and $f(9) = 4$. Now let's reorganize the $[T, +]$ table:

$+$	1	0	4
1	4	1	0
0	1	0	4
4	0	4	1

This table contains exactly the same data that were in the original T table; the rows and columns have just been shuffled. Written in this form, it's clear that the T table is just a relabeling (using f) of the S table, so f is indeed an isomorphism. Note that inverses map to inverses:

$$\begin{aligned} f(2^{-1}) &= f(9) = 4 \text{ and } -f(2) = -1 = 4 \\ f(9^{-1}) &= f(2) = 1 \text{ and } -f(9) = -4 = 1 \end{aligned}$$

And we can simulate the computation $9 \cdot 2 = 5$ in S by mapping to T , applying the $+$ operation, and mapping back to S :

$$\begin{aligned} f(9) &= 4, f(2) = 1 \\ 4 + 1 &= 0 \\ f^{-1}(0) &= 5, \text{ which is } 9 \cdot 2 \end{aligned}$$

EXAMPLE 16 Let $f: M_2(\mathbb{Z}) \rightarrow M_2(\mathbb{Z})$ be given by

$$f\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

To show that f is one-to-one, let

$$f\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = f\left(\begin{bmatrix} e & f \\ g & h \end{bmatrix}\right)$$

Then

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} = \begin{bmatrix} e & g \\ f & h \end{bmatrix}$$

so $a = e$, $c = g$, $b = f$, and $d = h$, or

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

To show that f is onto, let

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \in M_2(\mathbb{Z})$$

Then

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix} \in M_2(\mathbb{Z}) \quad \text{and} \quad f\left(\begin{bmatrix} a & c \\ b & d \end{bmatrix}\right) = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Also, f is a homomorphism from $[M_2(\mathbb{Z}), +]$ to $[M_2(\mathbb{Z}), +]$ because

$$\begin{aligned} f\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e & f \\ g & h \end{bmatrix}\right) &= f\left(\begin{bmatrix} a+e & b+f \\ c+g & d+h \end{bmatrix}\right) = \begin{bmatrix} a+e & c+g \\ b+f & d+h \end{bmatrix} \\ &= \begin{bmatrix} a & c \\ b & d \end{bmatrix} + \begin{bmatrix} e & g \\ f & h \end{bmatrix} = f\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) + f\left(\begin{bmatrix} e & f \\ g & h \end{bmatrix}\right) \end{aligned}$$

The function f is therefore an isomorphism from $[M_2(\mathbb{Z}), +]$ to $[M_2(\mathbb{Z}), +]$. ●

PRACTICE 30

Let $5\mathbb{Z} = \{5z \mid z \in \mathbb{Z}\}$. Then $[5\mathbb{Z}, +]$ is a group. Show that $f: \mathbb{Z} \rightarrow 5\mathbb{Z}$ given by $f(x) = 5x$ is an isomorphism from $[\mathbb{Z}, +]$ to $[5\mathbb{Z}, +]$. ■

If f is an isomorphism from $[S, \cdot]$ to $[T, +]$, then f^{-1} exists and is a bijection. Further, f^{-1} is also a homomorphism, this time from T to S . To see this, let t_1 and t_2 belong to T and consider $f^{-1}(t_1 + t_2)$. Because $t_1, t_2 \in T$ and f is onto, $t_1 = f(s_1)$ and $t_2 = f(s_2)$ for some s_1 and s_2 in S . Thus,

$$\begin{aligned} f^{-1}(t_1 + t_2) &= f^{-1}(f(s_1) + f(s_2)) \\ &= f^{-1}(f(s_1 \cdot s_2)) \\ &= (f^{-1} \circ f)(s_1 \cdot s_2) \\ &= s_1 \cdot s_2 \\ &= f^{-1}(t_1) \cdot f^{-1}(t_2) \end{aligned}$$

Therefore we can speak of S and T as being simply isomorphic, denoted by $S \simeq T$, without having to specify that the isomorphism is from S to T or vice versa.

Checking whether a given function is an isomorphism from S to T , as in Practice 30, is not hard. Deciding whether S and T are isomorphic may be harder. To prove that they are isomorphic, we must produce a function. To prove that they are not isomorphic, we must show that no such function exists. Since we can't try all possible functions, we use ideas such as the following: There is no one-to-one correspondence between S and T , S is commutative but T is not, and so on.

We have noted that isomorphic groups are alike except for relabeling and that each can be used to simulate the computations in the other. Isomorphism of groups is really an equivalence relation, as Practice 31 shows; thus isomorphic groups belong to the same equivalence class. Thinking of isomorphic groups as “alike except for labeling” is consistent with the idea that elements in an equivalence class represent different names for the same thing.

PRACTICE 31

- Let $f: S \rightarrow T$ be an isomorphism from the group $[S, \cdot]$ to the group $[T, +]$ and $g: T \rightarrow U$ be an isomorphism from $[T, +]$ to the group $[U, *]$. Show that $g \circ f$ is an isomorphism from S to U .
- Let \mathcal{T} be a collection of groups and define a binary relation ρ on \mathcal{T} by $S \rho T \leftrightarrow S \simeq T$. Show that ρ is an equivalence relation on \mathcal{T} . ■

We will finish this section by looking at some equivalence classes of groups under isomorphism. Often we pick out one member of an equivalence class and note that it is the typical member of that class and that all other groups in the class look just like it (with different names).

A result concerning the nature of very small groups follows immediately from Exercise 24 at the end of this section.

THEOREM ON SMALL GROUPS

Every group of order 2 is isomorphic to the group whose group table is

·	1	a
1	1	a
a	a	1

Every group of order 3 is isomorphic to the group whose group table is

·	1	a	b
1	1	a	b
a	a	b	1
b	b	1	a

Every group of order 4 is isomorphic to one of the two groups whose group tables are

·	1	a	b	c
1	1	a	b	c
a	a	1	c	b
b	b	c	1	a
c	c	b	a	1

·	1	a	b	c
1	1	a	b	c
a	a	b	c	1
b	b	c	1	a
c	c	1	a	b

We can also prove that any group is essentially a permutation group. Suppose $[G, \cdot]$ is a group. We want to establish an isomorphism from G to a permutation group; each element g of G must be associated with a permutation α_g on some set. In fact, the set will be G itself; for any $x \in G$, we define $\alpha_g(x)$ to be $g \cdot x$. We must show that $\{\alpha_g \mid g \in G\}$ forms a permutation group and that this permutation group is isomorphic to G . First we need to show that for any $g \in G$, α_g is indeed a permutation on G . From the definition $\alpha_g(x) = g \cdot x$, it is clear that $\alpha_g: G \rightarrow G$, but it must be shown that α_g is a bijection.

PRACTICE 32 Show that α_g as just defined is a permutation (bijection) on G .

Now we consider $P = \{\alpha_g \mid g \in G\}$ and show that P is a group under function composition. P is nonempty because G is nonempty, and associativity always holds for function composition. We must show that P is closed and has an identity and that each $\alpha_g \in P$ has an inverse in P . To show closure, let α_g and $\alpha_h \in P$. For any $x \in G$, $(\alpha_g \circ \alpha_h)(x) = \alpha_g(\alpha_h(x)) = \alpha_g(h \cdot x) = g \cdot (h \cdot x) = (g \cdot h) \cdot x$. Thus, $\alpha_g \circ \alpha_h = \alpha_{g \cdot h}$ and $\alpha_{g \cdot h} \in P$.

PRACTICE 33

- Let 1 denote the identity of G . Show that α_1 is an identity for P under function composition.
- For $\alpha_g \in P, \alpha_{g^{-1}} \in P$; show that $\alpha_{g^{-1}} = (\alpha_g)^{-1}$. ■

We now know that $[P, \circ]$ is a permutation group, and it only remains to show that the function $f: G \rightarrow P$ given by $f(g) = \alpha_g$ is an isomorphism. Clearly, f is an onto function.

PRACTICE 34 Show that $f: G \rightarrow P$ defined by $f(g) = \alpha_g$ is

- one-to-one.
- a homomorphism. ■

We have now proved the following theorem, first stated and proved by the English mathematician Arthur Cayley in the mid-1800s.

THEOREM CAYLEY'S THEOREM
Every group is isomorphic to a permutation group.

SECTION 9.1 REVIEW**TECHNIQUES**

- W** Test whether a given set and operation have the properties necessary to form a semigroup, monoid, or group structure.
- Test whether a given subset of a group is a subgroup.
 - Test whether a given function from one group to another is an isomorphism.
 - Decide whether two groups are isomorphic.

MAIN IDEAS

- Many elementary arithmetic systems are instances of a semigroup, monoid, or group structure.
- In any group structure, the identity and inverse elements are unique, cancellation laws hold, and linear equations are solvable; these and other properties follow from the definitions involved.
- A subset of a group may itself be a group under the inherited operation.

- The order of a subgroup of a finite group divides the order of the group.
- The only subgroups of the group $[\mathbb{Z}, +]$ are of the form $[n\mathbb{Z}, +]$, where $n\mathbb{Z}$ is the set of all integral multiples of a fixed $n \in \mathbb{N}$.
- If f is an isomorphism from one group to another, f maps the identity to the identity and inverses to inverses, and it preserves commutativity.
- If S and T are isomorphic groups, they are identical except for relabeling, and each simulates any computation in the other.
- Isomorphism is an equivalence relation on groups.
- To within an isomorphism, there is only one group of order 2, one group of order 3, and two groups of order 4.
- Every group is essentially a permutation group.

EXERCISES 9.1

- A binary operation \cdot is defined on the set $\{a, b, c, d\}$ by the table on the left. Is \cdot commutative? Is \cdot associative?
 - Let $S = \{p, q, r, s\}$. An associative binary operation \cdot is partly defined on S by the table on the right. Complete the table to preserve associativity. Is \cdot commutative?

\cdot	a	b	c	d
a	a	c	d	a
b	b	c	a	d
c	c	a	b	d
d	d	b	a	c

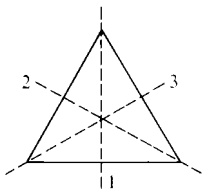
\cdot	p	q	r	s
p	p	q	r	s
q	q	r	s	p
r			p	
s	s		q	r

2. Show that matrix multiplication on $M_2(\mathbb{Z})$ is associative.
3. Each of the following cases defines a binary operation, denoted by \cdot , on a given set. Which are associative? Which are commutative?
 - a. On \mathbb{Z} : $x \cdot y = \begin{cases} x & \text{if } x \text{ is even} \\ x + 1 & \text{if } x \text{ is odd} \end{cases}$
 - b. On \mathbb{N} : $x \cdot y = (x + y)^2$
 - c. On \mathbb{R}^+ : $x \cdot y = x^4$
 - d. On \mathbb{Q} : $x \cdot y = xy/2$
 - e. On \mathbb{R}^+ : $x \cdot y = 1/(x + y)$
4. Define binary operations on the set \mathbb{N} that are
 - a. commutative but not associative.
 - b. associative but not commutative.
 - c. neither associative nor commutative.
 - d. both associative and commutative.

For Exercises 5–7, determine whether the structures $[S, \cdot]$ are semigroups, monoids, groups, or none of these. Name the identity element in any monoid or group structure.

5.
 - a. $S = \mathbb{N}$; $x \cdot y = \min(x, y)$
 - b. $S = \mathbb{R}$; $x \cdot y = (x + y)^2$
 - c. $S = \{a\sqrt{2} \mid a \in \mathbb{N}\}$; $\cdot =$ multiplication
 - d. $S = \{a + b\sqrt{2} \mid a, b \in \mathbb{Z}\}$; $\cdot =$ multiplication
 - e. $S = \{a + b\sqrt{2} \mid a, b \in \mathbb{Q}, a \text{ and } b \text{ not both } 0\}$; $\cdot =$ multiplication
 - f. $S = \{1, -1, i, -i\}$; $\cdot =$ multiplication (where $i^2 = -1$)
6.
 - a. $S = \{1, 2, 4\}$; $\cdot = \cdot_6$
 - b. $S = \{1, 2, 3, 5, 6, 10, 15, 30\}$; $x \cdot y =$ least common multiple of x and y
 - c. $S = \mathbb{N} \times \mathbb{N}$; $(x_1, y_1) \cdot (x_2, y_2) = (x_1, y_2)$
 - d. $S = \mathbb{N} \times \mathbb{N}$; $(x_1, y_1) \cdot (x_2, y_2) = (x_1 + x_2, y_1 y_2)$
 - e. $S =$ set of even integers; $\cdot =$ addition
 - f. $S =$ set of odd integers; $\cdot =$ addition
7.
 - a. $S =$ set of all polynomials in $\mathbb{R}[x]$ of degree ≤ 3 ; $\cdot =$ polynomial addition
 - b. $S =$ set of all polynomials in $\mathbb{R}[x]$ of degree ≤ 3 ; $\cdot =$ polynomial multiplication
 - c. $S = \left\{ \begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix} \mid z \in \mathbb{Z} \right\}$; $\cdot =$ matrix multiplication
 - d. $S = \{1, 2, 3, 4\}$; $\cdot = \cdot_5$

- e. $S = \mathbb{R} - \{-1\}; x \cdot y = x + y + xy$
 f. $S = \{f | f: \mathbb{N} \rightarrow \mathbb{N}\}; \cdot = \text{function addition, that is, } (f + g)(x) = f(x) + g(x)$
8. Let $A = \{1, 2\}$.
- Describe the elements and write the table for the semigroup of transformations on A .
 - Describe the elements and write the table for the group of permutations on A .
9. Given an equilateral triangle, six permutations can be performed on the triangle that will leave its image in the plane unchanged. Three of these permutations are clockwise rotations in the plane of 120° , 240° , and 360° about the center of the triangle; these permutations are denoted R_1, R_2 , and R_3 , respectively. The triangle can also be flipped about any of the axes 1, 2, and 3 (see the accompanying figure); these permutations are denoted F_1, F_2 , and F_3 , respectively. During any of these permutations, the axes remain fixed in the plane. Composition of permutations is a binary operation on the set D_3 of all six permutations. For example, $F_3 \circ R_2 = F_2$. The set D_3 under composition is a group, called the *group of symmetries of an equilateral triangle*. Complete the group table below for $[D_3, \circ]$. What is an identity element in $[D_3, \circ]$? What is an inverse element for F_1 ? For R_2 ?



\circ	R_1	R_2	R_3	F_1	F_2	F_3
R_1						
R_2						
R_3						
F_1						
F_2						
F_3					F_2	

10. The set S_3 , the symmetric group of degree 3, is isomorphic to D_3 , the group of symmetries of an equilateral triangle (see Exercise 9). Find a bijection from the elements of S_3 to the elements of D_3 that preserves the operation. (*Hint*: R_1 of D_3 may be considered a permutation in S_3 sending 1 to 2, 2 to 3, and 3 to 1.)
11. In each case, decide whether the structure on the left is a subgroup of the group on the right. If not, why not? (Note that here S^* denotes $S - \{0\}$.)
- $[\mathbb{Z}_5^*, \cdot_5]; [\mathbb{Z}_5, +_5]$
 - $[P, +]; [\mathbb{R}[x], +]$ where P is the set of all polynomials in x over \mathbb{R} of degree ≥ 3
 - $[\mathbb{Z}^*, \cdot]; [\mathbb{Q}^*, \cdot]$
 - $[K, +]; [\mathbb{R}[x], +]$ where K is the set of all polynomials in x over \mathbb{R} of degree $\leq k$ for some fixed k
12. In each case, decide whether the structure on the left is a subgroup of the group on the right. If not, why not?
- $[A, \circ]; [S, \circ]$ where S is the set of all bijections on \mathbb{N} and A is the set of all bijections on \mathbb{N} mapping 3 to 3
 - $[\mathbb{Z}, +]; [M_2(\mathbb{Z}), +]$
 - $[\{0, 3, 6\}, +_8]; [\mathbb{Z}_8, +_8]$
 - $[A, +_2]; [\mathbb{Z}_2^5, +_2]$ where $A = \{00000, 01111, 10101, 11010\}$
13. Find all the distinct subgroups of $[\mathbb{Z}_{12}, +_{12}]$.
14. a. Show that the subset

$$\begin{aligned} \alpha_1 &= i & \alpha_3 &= (1, 4) \circ (2, 3) \\ \alpha_2 &= (1, 2) \circ (3, 4) & \alpha_4 &= (1, 3) \circ (2, 4) \end{aligned}$$

forms a subgroup of the symmetric group S_4 .

b. Show that the subset

$$\begin{array}{ll} \alpha_1 = i & \alpha_5 = (1, 2) \circ (3, 4) \\ \alpha_2 = (1, 2, 3, 4) & \alpha_6 = (1, 4) \circ (2, 3) \\ \alpha_3 = (1, 3) \circ (2, 4) & \alpha_7 = (2, 4) \\ \alpha_4 = (1, 4, 3, 2) & \alpha_8 = (1, 3) \end{array}$$

forms a subgroup of the symmetric group S_4 .

15. Find the elements of the alternating group A_4 .
16. Let $A = \{p, q, r\}$. Then $[A^*, \cdot]$ is the free monoid generated by A .
- What is $ppqrp \cdot qpr$?
 - Let $B =$ the set of all strings over A with an even number of q 's. Then $B \subseteq A$. Prove that $[B, \cdot]$ is also a monoid.
17. In each case, decide whether the given function is a homomorphism from the group on the left to the one on the right. Are any of the homomorphisms also isomorphisms?
- $[\mathbb{Z}, +], [\mathbb{Z}, +]; f(x) = 2$
 - $[\mathbb{R}, +], [\mathbb{R}, +]; f(x) = |x|$
 - $[\mathbb{R}^*, \cdot], [\mathbb{R}^*, \cdot]$ (where \mathbb{R}^* denotes the set of nonzero real numbers); $f(x) = |x|$
18. In each case, decide whether the given function is a homomorphism from the group on the left to the one on the right. Are any of the homomorphisms also isomorphisms?
- $[\mathbb{R}[x], +], [\mathbb{R}, +]; f(a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0) = a_n + a_{n-1} + \cdots + a_0$
 - $[S_3, \circ], [\mathbb{Z}_2, +_2]; f(\alpha) = \begin{cases} 1 & \text{if } \alpha \text{ is an even permutation} \\ 0 & \text{if } \alpha \text{ is an odd permutation} \end{cases}$
 - $[\mathbb{Z} \times \mathbb{Z}, +]$ where $+$ denotes componentwise addition, $[\mathbb{Z}, +]; f(x, y) = x + 2y$
19. In each case, decide whether the given groups are isomorphic. If they are, produce an isomorphism function. If they are not, give a reason why they are not.
- $[\mathbb{Z}, +], [12\mathbb{Z}, +]$ (where $12\mathbb{Z} = \{12z \mid z \in \mathbb{Z}\}$)
 - $[\mathbb{Z}_5, +_5], [5\mathbb{Z}, +]$
 - $[5\mathbb{Z}, +], [12\mathbb{Z}, +]$
 - $[S_3, \circ], [\mathbb{Z}_6, +_6]$
20. In each case, decide whether the given groups are isomorphic. If they are, produce an isomorphism function. If they are not, give a reason why they are not.
- $[\{a_1 x + a_0 \mid a_1, a_0 \in \mathbb{R}\}, +], [\mathbb{C}, +]$
 - $[\mathbb{Z}_6, +_6], [S_6, \circ]$
 - $[\mathbb{Z}_2, +_2], [S_2, \circ]$
 - $[\mathbb{Z}_2^3, +_2], [\mathbb{Z}_8, +_8]$
21. Let $M_2^0(\mathbb{Z})$ be the set of all 2×2 matrices of the form

$$\begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix}$$

where $z \in \mathbb{Z}$.

- Show that $[M_2^0(\mathbb{Z}), \cdot]$ is a group, where \cdot denotes matrix multiplication.

b. Let a function $f: M_2^0(\mathbb{Z}) \rightarrow \mathbb{Z}$ be defined by

$$f\left(\begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix}\right) = z$$

Prove that f is an isomorphism from $[M_2^0(\mathbb{Z}), \cdot]$ to $[\mathbb{Z}, +]$

c. Use $[\mathbb{Z}, +]$ to simulate the computation

$$\begin{bmatrix} 1 & 7 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -3 \\ 0 & 1 \end{bmatrix}$$

in $[M_2^0(\mathbb{Z}), \cdot]$.

d. Use $[M_2^0(\mathbb{Z}), \cdot]$ to simulate the computation $2 + 3$ in $[\mathbb{Z}, +]$.

22. a. Let $S = \{1, -1\}$. Show that $[S, \cdot]$ is a group where \cdot denotes ordinary integer multiplication.

b. Let f be the function from the group $[S_n, \circ]$ to the group $[S, \cdot]$ given by

$$f(\alpha) = \begin{cases} 1 & \text{if } \alpha \text{ is even} \\ -1 & \text{if } \alpha \text{ is odd} \end{cases}$$

Prove that f is a homomorphism.

23. In any group $[G, \cdot]$, show that

a. $i^{-1} = i$

b. $(x^{-1})^{-1} = x$ for any $x \in G$

24. a. Show that any group of order 2 is commutative by constructing a group table on the set $\{1, a\}$ with 1 as the identity.

b. Show that any group of order 3 is commutative by constructing a group table on the set $\{1, a, b\}$ with 1 as the identity. (You may assume associativity.)

c. Show that any group of order 4 is commutative by constructing a group table on the set $\{1, a, b, c\}$ with 1 as the identity. (You may assume associativity.) There will be four such tables, but three of them are isomorphic because the elements have simply been relabeled from one to the other. Find these three groups and indicate the relabeling. Thus, there are two essentially different groups of order 4, and both of these are commutative.

25. Let $[G, \cdot]$ be a group and let $x, y \in G$. Define a relation ρ on G by $x \rho y \leftrightarrow g \cdot x \cdot g^{-1} = y$ for some $g \in G$.

a. Prove that ρ is an equivalence relation on G .

b. Prove that for each $x \in G$, $[x] = \{x\}$ if and only if G is commutative.

26. For x a member of a group $[G, \cdot]$, we can define x^n for any positive integer n by $x^1 = x$, $x^2 = x \cdot x$, and $x^n = x^{n-1} \cdot x$ for $n > 2$. Prove that in a finite group $[G, \cdot]$, for each $x \in G$ there is a positive integer k such that $x^k = i$.

27. Let $[S, \cdot]$ be a semigroup. An element $i_L \in S$ is a *left identity* element if for all $x \in S$, $i_L \cdot x = x$. An element $i_R \in S$ is a *right identity* element if for all $x \in S$, $x \cdot i_R = x$.

a. Prove that if a semigroup $[S, \cdot]$ has both a left identity element and a right identity element, then $[S, \cdot]$ is a monoid.

b. Give an example of a finite semigroup with two left identities and no right identity.

c. Give an example of a finite semigroup with two right identities and no left identity.

d. Give an example of a semigroup with neither a right nor a left identity.

28. Let $[S, \cdot]$ be a monoid with identity i , and let $x \in S$. An element x_L^{-1} in S is a *left inverse* of x if $x_L^{-1} \cdot x = i$. An element x_R^{-1} in S is a *right inverse* of x if $x \cdot x_R^{-1} = i$.
- Prove that if every element in a monoid $[S, \cdot]$ has both a left inverse and a right inverse, then $[S, \cdot]$ is a group.
 - Let S be the set of all functions f such that $f: \mathbb{N} \rightarrow \mathbb{N}$. Then S under function composition is a monoid. Define a function $f \in S$ by $f(x) = 2x, x \in \mathbb{N}$. Then define a function $g \in S$ by

$$g(x) = \begin{cases} x/2 & \text{if } x \in \mathbb{N}, x \text{ even} \\ 1 & \text{if } x \in \mathbb{N}, x \text{ odd} \end{cases}$$
 Prove that g is a left inverse for f . Also prove that f has no right inverse.
29. Let $[S, \cdot]$ be a semigroup having a left identity i_L (see Exercise 27) and the property that for every $x \in S$, x has a left inverse y such that $y \cdot x = i_L$. Prove that $[S, \cdot]$ is a group. (*Hint*: y also has a left inverse in S .)
30. An element of a semigroup $[S, \cdot]$ is idempotent if $x \cdot x = x$. Prove that a group contains exactly one idempotent element.
31. Prove that if $[S, \cdot]$ is a semigroup in which the linear equations $a \cdot x = b$ and $x \cdot a = b$ are solvable for any $a, b \in S$, then $[S, \cdot]$ is a group. (*Hint*: Use Exercise 29.)
32. Prove that a finite semigroup that satisfies the left and right cancellation laws is a group. (*Hint*: Use Exercise 29.)
33. Prove that a group $[G, \cdot]$ is commutative if and only if $(x \cdot y)^2 = x^2 \cdot y^2$ for each $x, y \in G$.
34. Prove that a group $[G, \cdot]$ in which $x \cdot x = i$ for each $x \in G$ is commutative.
35. Let $[G, \cdot]$ be a commutative group with identity i . For a fixed positive integer k , let $B_k = \{x \mid x \in G, x^k = i\}$. Prove that $[B_k, \cdot]$ is a subgroup of $[G, \cdot]$.
36. Let $[G, \cdot]$ be a commutative group with subgroups $[S, \cdot]$ and $[T, \cdot]$. Let $ST = \{s \cdot t \mid s \in S, t \in T\}$. Prove that $[ST, \cdot]$ is a subgroup of $[G, \cdot]$.
37. a. Let $[G, \cdot]$ be a group and let $[S, \cdot]$ and $[T, \cdot]$ be subgroups of $[G, \cdot]$. Prove that $[S \cap T, \cdot]$ is a subgroup of $[G, \cdot]$.
- b. Will $[S \cup T, \cdot]$ be a subgroup of $[G, \cdot]$? Prove or give a counterexample.
38. For any group $[G, \cdot]$, the *center* of the group is $A = \{x \in G \mid x \cdot g = g \cdot x \text{ for all } g \in G\}$.
- Prove that $[A, \cdot]$ is a subgroup of $[G, \cdot]$
 - Find the center of the group of symmetries of an equilateral triangle, $[D_3, \circ]$ (see Exercise 9).
 - Prove that G is commutative if and only if $G = A$.
 - Let x and y be members of G with $x \cdot y^{-1} \in A$. Prove that $x \cdot y = y \cdot x$.
39. a. Let S_A denote the group of permutations on a set A , and let a be a specific element of A . Prove that the set H_a of all permutations in S_A that map a to a forms a subgroup of S_A .
- b. If A has n elements, what is $|H_a|$?
40. a. Let $[G, \cdot]$ be a group and $A \subseteq G, A \neq \emptyset$. Prove that $[A, \cdot]$ is a subgroup of $[G, \cdot]$ if for each $x, y \in A, x \cdot y^{-1} \in A$. This subgroup test is sometimes more convenient to use than the theorem on subgroups.
- b. Use the test of part (a) to work Exercise 35.
41. a. Let $[G, \cdot]$ be any group with identity i . For a fixed $a \in G, a^0$ denotes i and a^{-n} means $(a^n)^{-1}$. Let $A = \{a^z \mid z \in \mathbb{Z}\}$. Prove that $[A, \cdot]$ is a subgroup of G . (*Hint*: Use Exercise 40.)
- b. The group $[G, \cdot]$ is a *cyclic group* if for some $a \in G, A = \{a^z \mid z \in \mathbb{Z}\}$ is the entire group G . In this case, a is a *generator* of $[G, \cdot]$. For example, 1 is a generator of the group $[\mathbb{Z}, +]$; remember that the operation is addition. Thus, $1^0 = 0, 1^1 = 1, 1^2 = 1 + 1 = 2, 1^3 = 1 + 1 + 1 = 3 \dots$; $1^{-1} = (1)^{-1} = -1; 1^{-2} = (1^2)^{-1} = -2; 1^{-3} = (1^3)^{-1} = -3, \dots$. Every integer can be written as an integral

- “power” of 1, and $[\mathbb{Z}, +]$ is cyclic with generator 1. Prove that the group $[\mathbb{Z}_7, +_7]$ is cyclic with generator 2.
- Prove that 5 is also a generator of the cyclic group $[\mathbb{Z}_7, +_7]$.
 - Prove that 3 is a generator of the cyclic group $[\mathbb{Z}_4, +_4]$.
42. Let $[G, \cdot]$ be a cyclic group with generator a (see Exercise 41). Show that G is commutative.
43. a. Let $[S, \cdot]$ be a semigroup. An isomorphism from S to S is called an *automorphism* on S . Let $\text{Aut}(S)$ be the set of all automorphisms on S , and prove that $\text{Aut}(S)$ is a group under function composition.
- b. For the group $[\mathbb{Z}_4, +_4]$, find the set of automorphisms and show its group table under \circ .
44. Let $[G, \cdot]$ be a commutative group with identity i . Prove that the function $f: G \rightarrow G$ given by $f(x) = x^{-1}$ is an isomorphism.
45. Let f be a homomorphism from a group G onto a group H . Show that f is an isomorphism if and only if the only element of G that is mapped to the identity of H is the identity of G .
46. Let $[G, \cdot]$ be a group and g a fixed element of G . Define $f: G \rightarrow G$ by $f(x) = g \cdot x \cdot g^{-1}$ for any $x \in G$. Prove that f is an isomorphism from G to G .
47. a. Consider the equivalence relation on the integers of congruence modulo n defined in Section 5.1. If $n = 5$, there are 5 equivalence classes:

$$[0] = \{\dots, -10, -5, 0, 5, 10, \dots\}$$

$$[1] = \{\dots, -9, -4, 1, 6, 11, \dots\}$$

$$[2] = \{\dots, -8, -3, 2, 7, 12, \dots\}$$

$$[3] = \{\dots, -7, -2, 3, 8, 13, \dots\}$$

$$[4] = \{\dots, -6, -1, 4, 9, 14, \dots\}$$

Let $E_5 = \{[0], [1], [2], [3], [4]\}$. An operation $+$ is defined on E_5 by

$$[x] + [y] = [x + y]$$

For example, $[2] + [4] = [2 + 4] = [6] = [1]$ (recall that an equivalence class can be named by any of its elements). Prove that $[E_5, +]$ is a commutative group.

- $[\mathbb{Z}_5, +_5]$ is a commutative group with elements $\{0, 1, 2, 3, 4\}$ (see Example 7). Prove that $[\mathbb{Z}_5, +_5]$ is isomorphic to the group $[E_5, +]$.
- Results *a* and *b* hold for any value of n . In the group E_{14} , what is the inverse of $[10]$? What is the preimage of $[21]$ under the isomorphism from \mathbb{Z}_{14} to E_{14} ?

SECTION 9.2 CODING THEORY

Introduction

We talked about cryptographic codes (codes for secrecy) in Chapter 5. The codes we will talk about in this section are designed not to keep data secret but to cope with degraded data. Data can degrade over time in a storage device or can be corrupted over space, that is, during a transmission from one site to another over some medium. Bits are changed from 0 to 1 or vice versa through interference (“noise”), hardware failures, media damage, and so forth. The goal is to be able to detect, perhaps even to correct, such errors.

As an analogy, consider voice transmission over a poor-quality cell phone connection. The speaker says *the black hat* but the receiver hears *the black cat*. Because the received message makes sense and could have been the transmitted message, there is no way to detect that an error has occurred. An alternative is to encode the message to be transmitted by repeating it. Thus the code word for *the black hat* would be *the black hat the black hat*, and the code word for *the black cat* would be *the black cat the black cat*. A received message of *the black hat the black cat* would alert the receiver that an error has occurred in transmission. However, the received message is equally close to either of the two code words, so there is no way to guess the correct code word. Note that two errors could still go undetected.

Now let's make the code word three copies of the message: *the black hat the black hat the black hat* and *the black cat the black cat the black cat*. A received message of *the black hat the black cat the black hat* would signal that either one or two errors has occurred. If we assume it to be more likely that only one error has occurred, we decode the message to the closest code word, *the black hat the black hat the black hat*. This process is called **maximum likelihood decoding** and gives us the correct code word for a received message where no more than one error has occurred. Because we can detect up to two errors and correct the effects of one error, we have designed a **double-error detecting, single-error correcting code**.

Here we've seen three ideas central to coding theory: redundancy in coding, maximum-likelihood decoding, and distance between code words. Redundancy uses additional bandwidth or storage space, and it also increases exposure to errors. Yet, as we've seen from our little example, it also increases the capability for detecting and perhaps correcting such errors. You are probably familiar with the idea of a parity bit, where at the end of a binary string one extra bit is added so that the total number of 1s in the string, including the parity bit, is even (an even parity bit scheme). A single bit error is detectable because it would result in an odd number of 1s. But there's no way to tell which bit is in error. An even number of errors is undetectable, and any odd number of errors is indistinguishable from a single error. This code is single-error detecting.

The codes we will examine are generalizations of the parity-bit code. A parity-bit code adds one bit to the end of an m -tuple message to turn it into an n -tuple code word where $n = m + 1$. Our code words will add additional bits to the end of an m -tuple message to create an n -tuple code word where the additional $n - m$ bits are all special sorts of parity bits. These codes also rely on further results from group theory (indeed they are called *group codes*), so we need some additional background first.

Background: Homomorphisms and Cosets

According to the definition, for f to be an isomorphism it must be both a bijection and a homomorphism.

EXAMPLE 17

Consider the following functions from \mathbb{Z} to \mathbb{Z} :

$$\begin{aligned} f(x) &= 0 \\ g(x) &= x + 1 \end{aligned}$$

The function f is a homomorphism from the group $[\mathbb{Z}, +]$ to the group $[\mathbb{Z}, +]$ because $f(x + y) = 0 = 0 + 0 = f(x) + f(y)$. However, f is not a bijection,

so it is not an isomorphism. The function g is a bijection because $g(x) = g(y)$ implies $x + 1 = y + 1$, or $x = y$, so g is one-to-one; g is also onto because for any $z \in \mathbb{Z}$, $z - 1 \in \mathbb{Z}$ and $g(z - 1) = z$. But g is not a homomorphism because $g(x + y) = (x + y) + 1 \neq (x + 1) + (y + 1) = g(x) + g(y)$. Hence g is not an isomorphism. ●

What can be said about functions from a group to a group that are homomorphisms but not isomorphisms? More specifically, let $[G, \cdot]$ be a group with identity i_G , $[H, +]$ be a group with identity i_H , and f be a homomorphism, $f: G \rightarrow H$. The range of f , $f(G)$, is a subset of H (remember that f might not be an onto function, so it isn't necessarily true that $f(G) = H$). In fact, $f(G)$ is actually a subgroup of H . It is easy enough to prove the three properties needed for $f(G)$ to be a subgroup, and the proof will show that $f(i_G) = i_H$ and $-f(x) = f(x^{-1})$ (Exercise 1). We can define a binary relation ρ on G by

$$x \rho y \leftrightarrow f(x) = f(y)$$

and ρ is an equivalence relation.

PRACTICE 35 Prove that ρ is an equivalence relation on G . ■

If f is a one-to-one function, $x \rho y$ would mean that $f(x) = f(y)$ and therefore $x = y$. The equivalence classes formed by ρ would then be trivial, each containing only a single element of G . But in general, because f is not a bijection, it might not be a one-to-one function. Let $[i_G]$ be the equivalence class determined by ρ that contains the identity of G . Because $f(i_G) = i_H$, $[i_G] = \{x \in G \mid f(x) = i_H\}$. This set is called the **kernel** of the homomorphism f .

The kernel K is a subset of G , and in addition it is a subgroup of G . We need to prove the three properties necessary for a subgroup:

REMINDER

Given a homomorphism f from $[G, \cdot]$ to $[H, +]$, the range $f(G)$ is a subgroup of $[H, +]$ and the kernel K is a subgroup of $[G, \cdot]$.

1. K is closed: Let x and y be elements in K . Then $f(x) = f(y) = i_H$. And $f(x \cdot y) = f(x) + f(y) = i_H + i_H = i_H$, so $x \cdot y$ is in K .
2. i_G belongs to K : Yes, because $f(i_G) = i_H$.
3. Let x be an element in K : Then $f(x) = i_H$. Also $f(x^{-1}) = -f(x) = -i_H = i_H$ so x^{-1} is in K .

EXAMPLE 18 Let $\mathbb{R}^* = \mathbb{R} - \{0\}$ and let \mathbb{R}^+ be the set of all positive real numbers. Then the function f defined by $f(x) = |x|$ is a homomorphism from the group $[\mathbb{R}^*, \cdot]$ to the group $[\mathbb{R}^+, \cdot]$. The kernel K of f is $\{x \in \mathbb{R}^* \mid f(x) = |x| = 1\}$. Therefore $K = \{1, -1\}$. ●

PRACTICE 36 The function f defined by $f(x) = x \cdot_3 1$ is a homomorphism from the group $[\mathbb{Z}, +]$ to the group $[\mathbb{Z}_3, +_3]$. Find the kernel K . ■

For our discussion of cosets, we begin with a definition.

● **DEFINITION COSET**

Let $[S, +]$ be a subgroup of a group $[G, +]$. Then for $x \in G$, sets of the form $x + S = \{x + s \mid s \in S\}$ are called **left cosets** of S in G . Sets of the form $S + x = \{s + x \mid s \in S\}$ are called **right cosets** of S in G .

Of course, if G is commutative, then left cosets and right cosets are identical.

EXAMPLE 19

Let $S = \{0, 2, 4, 6\}$. Then $[S, +_8]$ is a subgroup of $[\mathbb{Z}_8, +_8]$. The left coset $5 +_8 S = \{5 +_8 0, 5 +_8 2, 5 +_8 4, 5 +_8 6\} = \{5, 7, 1, 3\}$. ●

PRACTICE 37

- Verify that $5 +_8 S = 1 +_8 S = 3 +_8 S = 7 +_8 S$.
- Compute the left coset $2 +_8 S$. What are its other names? ■

For $[S, +]$ any subgroup of a group $[G, +]$, we can define a binary relation ρ on G as follows:

$$x \rho y \leftrightarrow y \text{ belongs to } x + S, x\text{'s left coset of } S \text{ in } G$$

Then $x \rho y$, which is equivalent to y belonging to $x + S$, means y can be written as $x + s$ for some element $s \in S$. It turns out that ρ is an equivalence relation on G :

Reflexive property: $x \rho x$ because $x = x + i_G$. (Because S is a subgroup of G , the identity i_G of G is an element of S .)

Symmetric property: if $x \rho y$ then $y = x + s$ for some $s \in S$. Then, because S is a subgroup, $-s$ is in S and $x = y + (-s)$, which puts x in the same coset as y , that is $y \rho x$.

Transitive property: if $x \rho y$ and $y \rho z$ then $y = x + s_1$ for some $s_1 \in S$ and $z = y + s_2$ for some $s_2 \in S$. Therefore, $z = y + s_2 = (x + s_1) + s_2 = x + (s_1 + s_2)$, which means z is in x 's left coset so $x \rho z$.

We have proved the following result.

● **THEOREM ON COSET PARTITIONS**

Let $[S, +]$ be a subgroup of a group $[G, +]$. Then the set of left cosets of S in G forms a partition of G . One of these cosets is $i_G + S = S$.

This coset partition will play a key role in decoding our group codes. But first, we need to see how we actually create group codes.

Generating Group Codes

Suppose we know how to turn any m -bit binary message into an n -bit code word where $m < n$, and we also know how to reverse this process. There is then a set of n -bit code words, one for each m -bit binary string. An n -bit code word is transmitted and the received message is also a binary n -tuple. Two cases can occur:

1. The received n -tuple is identical to one of the code words. In this case, maximum-likelihood decoding will assume that no errors have occurred. This is actually based on another assumption, namely that the probability of errors occurring is very low, so that the probability of no errors is very high. Therefore the probability that a received code word is the same code word that was originally transmitted is more likely than that the transmitted code word was scrambled into a different code word.
2. The received n -tuple does not match any code word. Maximum-likelihood decoding suggests that the received n -tuple should be decoded as the closest code word, as this would be the result of the fewest bit errors and therefore has the highest probability of giving the correct result.

Keep in mind throughout this section that maximum-likelihood decoding does not guarantee 100% accurate results; it guarantees only results with the highest probability of being accurate.

To determine the “closest” code word, we need to define what we mean by the distance between binary n -tuples. Hamming distance, defined here, is named for Richard W. Hamming, an American mathematician who pioneered the study of error-detecting and error-correcting codes in 1950.

● **DEFINITION** **HAMMING DISTANCE**

Let X and Y be binary n -tuples. The **Hamming distance** between X and Y , $H(X, Y)$, is the number of components in which X and Y differ. The **minimum distance** of a code is the minimum Hamming distance between all possible pairs of distinct code words.

PRACTICE 38 For $X = 01011$ and $Y = 11001$, what is $H(X, Y)$?

Each error that occurs in the transmission of a code word adds one unit to the Hamming distance between that code word and the received word. Suppose we picture the code words as specific binary n -tuples distinguished from the set S of all binary n -tuples, as in Figure 9.1. Suppose also that the minimum distance of the code is at least $d + 1$. Then any time a code word X is corrupted by d or fewer errors, it will be changed to an n -tuple X' that is not another code word, and the occurrence of errors can be detected. Conversely, if any combination of d or fewer errors can be detected, code words must be at least $d + 1$ apart.

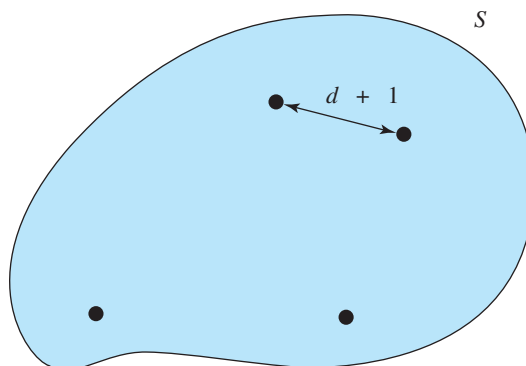


Figure 9.1

Now suppose that the minimum distance of the code is at least $2d + 1$. Then any time a code word X is corrupted by d or fewer errors, the received word X' will be such that $H(X, X') \leq d$ but for any other code word Y , $H(X', Y) \geq d + 1$. Therefore X' will be correctly decoded as X , the closest code word. Conversely, to correct any received word with d or fewer errors, the minimum distance of the code must be at least $2d + 1$ so that neighborhoods of radius d around code words do not intersect (Figure 9.2).

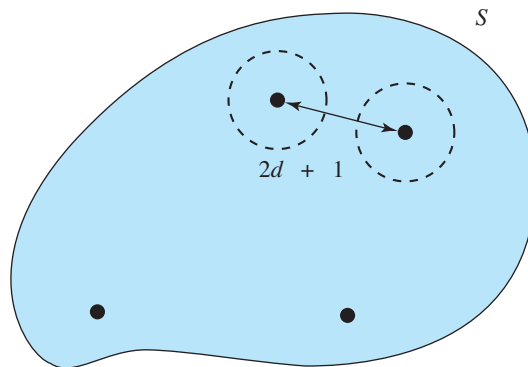


Figure 9.2

As a result, we see that a code is d -error detecting if and only if its minimum distance is at least $d + 1$, and it is d -error correcting if and only if its minimum distance is at least $2d + 1$. As one would expect, it requires a stronger condition to correct errors than to merely detect them.

EXAMPLE 20

Suppose a code has a minimum distance of 6. It can then detect any combination of ≤ 5 errors. It can correct any combination of ≤ 2 errors. If code words X and Y are such that $H(X, Y) = 6$, then there will be a word X' produced by 4 errors on X that will be incorrectly decoded as Y ; there will be a word X'' produced by 3 errors on X that can be arbitrarily decoded correctly as X or incorrectly as Y . This code is double-error correcting, 5-error detecting. ●

The set of code words will be a subset of the set of all binary n -tuples, Z_2^n . We want these code words to be sufficiently widely scattered in Z_2^n so that the minimum distance is large enough to allow some error correction. The minimum distance of the code is easy to compute if the code words form a subgroup of the group $[Z_2^n, +_2]$. In this case we have a **group code**; the identity is the n -tuple of all 0s (we'll call this 0_n), and each code word is its own inverse. Denote by $W(X)$ the **weight** of a code word X , meaning the number of 1s it contains.

THEOREM ON MINIMUM DISTANCE OF GROUP CODE

The minimum distance of a group code equals the minimum weight of all the nonzero code words.

To prove this theorem, let d be the minimum distance of a group code; then there are two distinct code words X and Y with $H(X, Y) = d$. Because it's a group code, closure holds and $X +_2 Y = Z$ is a code word. $Z \neq 0$ because X and Y are distinct. Z will have 1s in exactly those components where X and Y differ, so

$W(Z) = H(X, Y) = d$. Thus the minimum weight of the code is $\leq d$. If the minimum weight is $< d$, let M be a nonzero code word with $W(M)$ the minimum weight. Then (remember that 0_n is a code word) $H(M, 0_n) = W(M) < d$, which contradicts the fact that d is the minimum distance of the code. Therefore the minimum distance equals the minimum weight.

EXAMPLE 21

The set $\{00000, 01111, 10101, 11010\}$ is a group code (you can check closure, identity, and inverses) in $[Z_2^5, +_2]$. The minimum distance of the code is 3, so it is a single-error correcting code ($3 = 2 * 1 + 1$). ●

All well and good, but how can we produce subgroups of Z_2^n to use as code words, and how can we control the minimum distance of the code? Algebraic ideas again come to our rescue. Let \mathbf{H} be any $n \times r$ binary matrix with $r < n$. If $X \in Z_2^n$, we can perform the matrix multiplication $X \cdot \mathbf{H}$, where all additions are done modulo 2. The result of this multiplication is a binary r -tuple.

EXAMPLE 22

Let $r = 3$, $n = 5$, and

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

Then

$$(11101) \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} = (110)$$

We can think of multiplication by \mathbf{H} as a mapping from the group $[Z_2^n, +_2]$ to the group $[Z_2^r, +_2]$. Moreover, this mapping is a homomorphism (see Exercise 13 at the end of this section). We know that the kernel K of this homomorphism is a subgroup of $[Z_2^n, +_2]$ and it consists of all those X in Z_2^n such that $X \cdot \mathbf{H} = 0_r$, the zero of the group $[Z_2^r, +_2]$. We'll take K as the set of code words. Then we can easily determine the minimum weight (minimum distance) of the code simply by looking at \mathbf{H} . If \mathbf{H} has d distinct rows that add to 0_r in $[Z_2^r, +_2]$, say i_1, \dots, i_d , we can choose an X in Z_2^n having 1s exactly in the i_1, \dots, i_d , components. Then $X \cdot \mathbf{H} = 0_r$, so that X is a code word, and $W(X) = d$. On the other hand, if X is a code word with $W(X) = d$ and X has 1's exactly in components i_1, \dots, i_d , then the equation $X \cdot \mathbf{H} = 0_r$ forces rows i_1, \dots, i_d , of \mathbf{H} to sum to 0_r . Therefore, the minimum weight of the code equals the minimum number of distinct rows of \mathbf{H} that add to 0_r . In particular, to produce a single-error correcting code, we must have minimum distance at least 3, so we would have to choose an \mathbf{H} with no row consisting of all 0s and no two rows that are alike (these would add to 0_r).

EXAMPLE 23

The code words of Example 21 were generated using the matrix \mathbf{H} where

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

\mathbf{H} has no row of all 0s and no two rows that are alike, but rows 1, 2, and 4 add to $(0,0,0)$. Again, we see that the minimum distance of this code is 3. ●

PRACTICE 39

For each code word X of Example 21, verify that $X \cdot \mathbf{H} = 0_3$, where \mathbf{H} is given in Example 23. ■

From now on, we will assume that the matrix \mathbf{H} has the form

$$\mathbf{H} = \begin{bmatrix} \mathbf{B} \\ \mathbf{I}_r \end{bmatrix} \quad (1)$$

where \mathbf{I}_r is the $r \times r$ identity matrix and \mathbf{B} is an arbitrary $(n - r) \times r$ binary matrix. In computing the product $X \cdot \mathbf{H}$ for some $X \in Z_2^n$, we multiply elements of X by corresponding elements of the columns of \mathbf{H} , and then sum modulo 2. For each column of \mathbf{H} , the pattern of 1s in the column determines which components of X contribute to the sum. If the sum is to be 0 (as is true when $X \cdot \mathbf{H} = 0_r$), then those selected components of X must sum to 0 and therefore must consist of an even number of 1s. The \mathbf{I}_r portion of \mathbf{H} has the effect that each column of \mathbf{H} selects a distinct component from among the last r components of X . Each of the last r components of X therefore controls the even parity check for one of the r multiplications that are done. A matrix \mathbf{H} that matches Equation (1) is called a **canonical parity-check matrix**.

For X to be a code word, the first $n - r$ components of X can be arbitrary, but the final r components will then be determined. The maximum number of code words is therefore the maximum number of ways to select binary $(n - r)$ -tuples, or 2^{n-r} . Let $m = n - r$. We can code all members of Z_2^m in Z_2^n by leaving the first m components alone and then choosing the last r components so that the even parity check works for each column of \mathbf{H} . Such a code is called an (n, m) code. The first m components of a code word are the **information bits**, and the last r components are the **check bits**.

EXAMPLE 24

The matrix \mathbf{H} of Example 23 is a canonical parity-check matrix where $n = 5$, $r = 3$, and $m = n - r = 2$. \mathbf{H} can thus generate $2^m = 2^2 = 4$ code words. The four members of Z_2^2 are 00, 01, 10, and 11. Each can be coded as a member of Z_2^5 by keeping the first two digits and adding the appropriate check digits. To code 10, for instance, we have

$$(10C_1C_2C_3) \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = (000)$$

Multiplication by the first column of \mathbf{H} gives $1 \cdot 1 + 1 \cdot C_1$; therefore C_1 must equal 1 to give the sum value of 0. Similarly $C_2 = 0$ and $C_3 = 1$. We encode 10 as 10101. ●

PRACTICE 40 Use the encoding procedure of Example 24 to code 00, 01, and 11 in Z_2^5 . Compare the results with Example 21. ■

For a given $n \times r$ canonical parity-check matrix \mathbf{H} , we know how to encode all of $Z_2^m = Z_2^{n-r}$ as a subgroup of Z_2^n , and we also know how to determine from \mathbf{H} the minimum distance of the resulting code. Now let's turn the problem around. Suppose we want to encode Z_2^m for some m as, say, a single-error correcting code. How big will the code words have to be (what is n) or, equivalently, how many check bits must be added (what is r)? Once we know the dimensions of a parity-check matrix \mathbf{H} , how can we find a canonical \mathbf{H} that generates the code? We know that to produce a single-error correcting code, \mathbf{H} must have no row consisting of all 0s and no two rows that are alike. That means that in the canonical form $\mathbf{H} = \begin{bmatrix} \mathbf{B} \\ \mathbf{I}_r \end{bmatrix}$, \mathbf{B} cannot contain the row of all 0s nor can it contain any row with a single 1 because this would match one of the rows in \mathbf{I}_r . The rows are r -tuples, so from the 2^r possible r -tuples, we have to subtract those that cannot occur in \mathbf{B} . \mathbf{B} can have at most $2^r - 1 - r$ rows so m , the number of rows in \mathbf{B} , must be $\leq 2^r - 1 - r$. If m and r are such that

$$m = 2^r - 1 - r$$

the resulting code is called a **perfect code**.

EXAMPLE 25 A $(7, 4)$ code is a perfect code. Here $n = 7$, $m = 4$, $r = n - m = 3$, and $4 = 2^3 - 1 - 3$. The matrix \mathbf{H} will be a 7×3 matrix of the form

$$\begin{bmatrix} \mathbf{B} \\ \mathbf{I}_3 \end{bmatrix}$$

where the $m = 4$ rows of \mathbf{B} are all of the 3-tuples with at least two 1s. ●

PRACTICE 41

- Write an \mathbf{H} matrix for the $(7, 4)$ perfect code.
- A $(7, 4)$ code can encode all of Z_2^4 in Z_2^7 . Using your \mathbf{H} from part (a), write the set of binary 4-tuples that \mathbf{H} encodes and write the code word for each one. ■

Decoding Group Codes

Now suppose that some encoding scheme has been used to encode all members of Z_2^m in Z_2^n . There will be 2^m of these code words scattered among the 2^n binary n -tuples. These code words are known to us (remember that secrecy is not the issue here). When an n -tuple X is received, we have a process for decoding: we decode X as the closest code word in terms of Hamming distance, making the assumption that the fewest errors have occurred. This process is not truly an algorithm, however; there may not be a unique closest code word, and even if there is, we might still decode incorrectly if enough errors have occurred.

Nonetheless, let's concentrate on how to find the closest code word(s) for a received word X . There is certainly a brute force approach; we can create an array of all 2^m code words and when an X is received, we just compare X to each one in turn to find the closest code word. But wait—this requires storage for an exponential array size as well as an exponential sequential search process. Even for a relatively modest m -value of 32, 2^{32} is a very large number. But, suppose our code is a group code generated by an $n \times r$ canonical parity-check matrix \mathbf{H} . Then we will ultimately be able to decode by searching an array of only 2^r elements. In a single-error correcting code with $m = 32$, r can be as low as 6 ($m \leq 2^r - 1 - r$), and $2^6 = 64$ is an acceptable search array.

Here's how this decoding works. Recall that the set of code words equals the kernel K of the homomorphism \mathbf{H} induces from $[Z_2^n, +_2]$ to the group $[Z_2^r, +_2]$. We therefore know that K is a subgroup of $[Z_2^n, +_2]$ and we also know that the set of left cosets of K in $[Z_2^n, +_2]$ partitions the set Z_2^n . Cosets have the form $X +_2 K$ where $X \in Z_2^n$. Any given coset

$$X +_2 K = \{X +_2 C_i \mid C_i \in K\}$$

K has 2^m elements, so the size of each coset is 2^m . Because the set of left cosets partitions Z_2^n , a set of size 2^n , there must be $2^n/2^m = 2^{n-m} = 2^r$ distinct cosets.

When a word X is received, X belongs to the coset $X +_2 K$. As we just noted, each element E_i of this coset has the form $X +_2 C_i$ where C_i is a code word. Both E_i and C_i are binary n -tuples, so they are each self-inverse ($-E_i = E_i$ and $-C_i = C_i$). Therefore the equation

$$E_i = X +_2 C_i \tag{2}$$

can be written as

$$X +_2 E_i = C_i \tag{3}$$

From (2) we can see that 1's in E_i occur in exactly those components where X and C_i differ. Thus the weight of E_i equals the distance between X and C_i , and the closest code word to X is the one for which the corresponding E_i has minimum weight. To decode X , look for the element in the coset of X having minimum weight and, by (3), add that element to X . The result is the code word to which we decode X . The coset element having minimum weight is called the **coset leader**, and it may not be unique. If there are two "minimum-weight" elements in a given coset, one is chosen arbitrarily as the coset leader. This just means that no word in this particular coset can be accurately decoded because it has too many errors.

Summary: To decode a received n -tuple X , find the coset to which X belongs and add that coset's leader to X .

EXAMPLE 26

Consider the code of Example 21. Here $n = 5$ and $K = \{00000, 01111, 10101, 11010\}$. Suppose the 5-tuple $X = 11011$ is received. Because the set of code words is so small, we can easily pick out the closest code word, which is 11010. We would decode X as 11010. (From Example 23 we know that the canonical matrix \mathbf{H} for this code is 5×3 , so $r = 3$. Knocking off the three check bits, the original information bits were 11.)

Now let's try our decoding procedure. The elements of X 's coset are

$$\begin{aligned} 11011 +_2 00000 &= 11011 \\ 11011 +_2 01111 &= 10100 \\ 11011 +_2 10101 &= 01110 \\ 11011 +_2 11010 &= 00001 \end{aligned}$$

The coset leader (the element of minimum weight in this coset) is clearly 00001.

Adding this to X we get

$$11011 +_2 00001 = 11010$$

and we decode X to 11010, as before. ●

But we haven't really solved the efficiency problem. To find all the elements of X 's coset requires adding all the code words to X , which still requires knowing all 2^m code words. For a better approach, we need one new idea.

DEFINITION SYNDROME

In the group code generated by an $n \times r$ parity-check matrix \mathbf{H} , for any $X \in \mathbb{Z}_2^n$, the r -tuple $X \cdot \mathbf{H}$ is the **syndrome** of X .

The syndrome is useful because of the following theorem.

THEOREM ON SYNDROMES AND COSETS

Let \mathbf{H} be an $n \times r$ parity-check matrix generating a group code K . Then for $X, Y \in \mathbb{Z}_2^n$, X and Y are in the same left coset of K in $[\mathbb{Z}_2^n, +_2]$ if and only if X and Y have the same syndrome.

Proof: Suppose X and Y are in the same left coset of K in $[\mathbb{Z}_2^n, +_2]$. Then $Y = X +_2 C_i$ for some $C_i \in K$, and $Y \cdot \mathbf{H} = (X +_2 C_i) \cdot \mathbf{H} = X \cdot \mathbf{H} +_2 C_i \cdot \mathbf{H}$ (because multiplication by \mathbf{H} is a homomorphism) $= X \cdot \mathbf{H} +_2 0_r$ (because C_i is in the kernel of this homomorphism) $= X \cdot \mathbf{H}$. Therefore $Y \cdot \mathbf{H} = X \cdot \mathbf{H}$ and X and Y have the same syndrome.

Now suppose that $Y \cdot \mathbf{H} = X \cdot \mathbf{H}$. Then $Y \cdot \mathbf{H} +_2 X \cdot \mathbf{H} = 0_r$, or $(Y +_2 X) \cdot \mathbf{H} = 0_r$, which makes $Y +_2 X$ a code word C_i in K . If $Y +_2 X = C_i$, then $Y = X +_2 C_i$ and Y and X are in the same coset. *End of proof.*

PRACTICE 42

Example 26 shows four members of one coset. The parity-check matrix that generated the code for this example is

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Compute the syndrome for each member of the coset. ■

Now suppose that we somehow have available a list of the 2^r coset leaders. It is then easy to find which coset leader corresponds to X by making use of the fact from the previous theorem that X and the coset leader of X 's coset will have the same syndrome.

EXAMPLE 27

In the $(5, 2)$ code of Example 26, $n = 5$, $m = 2$, and $r = 3$. The size of each coset is $2^m = 4$, and there are $2^r = 8$ distinct cosets. Here is a list of the coset leaders and their corresponding syndromes. The syndromes were computed using the matrix \mathbf{H} of Practice 42, and the coset leaders were found by brute force.

Coset leaders	Syndromes
00000	000
00001	001
00010	010
00011	011
00100	100
10000	101
00110	110
01000	111

A received word of 10101 is decoded by computing its syndrome: $(10101)\mathbf{H} = 000$. The coset leader sharing this same syndrome is 00000. The received word is decoded as

$$10101 +_2 00000 = 10101$$

The received word is a code word, so it is assumed that no errors have occurred.

A received word of 11000 is decoded by computing its syndrome: $(11000)\mathbf{H} = 010$. The coset leader sharing this same syndrome is 00010, so the received word is decoded as

$$11000 +_2 00010 = 11010$$

assuming that a single error has occurred.

A received word of 10011 has a syndrome of 110, so it can be decoded as

$$10011 +_2 00110 = 10101$$

But because its coset leader has weight 2, we might instead generate a flag noting that at least two errors have occurred and that decoding cannot be done with certainty. ●

At this point we have traded one difficulty for another, namely, How do we find the list of 2^r coset leaders? This isn't always easy to do. In Example 27, the "brute force" involved writing all $2^n = 32$ binary 5-tuples, computing the 32 syndromes to group the 5-tuples into 8 cosets, and then reviewing the 4 members of each coset to find the coset leader. In two cosets, there was a tie for coset leader, and one value was picked arbitrarily in each case.

But—if the code is a perfect single-error correcting code, then the coset leaders are easy to find. Recall that in a perfect code, $m = 2^r - 1 - r$, and because $r = n - m$, it is also true that $n = 2^r - 1$. The n rows of the matrix \mathbf{H} are r -tuples that are the binary representations of the numbers $1, 2, \dots, 2^r - 1$ (there's no 0_r row in \mathbf{H}). The code word 0_n is the coset leader corresponding to the syndrome 0_r . Any other syndrome is a binary r -tuple representing a digit d , $1 \leq d \leq 2^r - 1$. The value d is also represented by a row of \mathbf{H} , say, row q . The coset leader for this syndrome is the binary n -tuple with a 1 in component q and 0s elsewhere. In this case there is no ambiguity about the coset leader, and every received word is at most distance 1 from a code word.

EXAMPLE 28

A $(7, 4)$ code is a perfect code for which $r = 3$. One matrix generating such a code is

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Given the syndrome 101, the row of \mathbf{H} that represents this binary number is row 2. The coset leader for this syndrome is therefore 0100000 (a 1 in component 2, 0s elsewhere). You can check that $(0100000)\mathbf{H} = 101$. The table of coset leaders and the corresponding syndromes is partially filled in here.

Coset leaders	Syndromes
0000000	000
	001
	010
	011
	100
0100000	101
	110
	111

A received word of 1001100 has a syndrome of $(1001100)\mathbf{H} = 101$, so it is decoded as

$$1001100 +_2 0100000 = 1101100$$

SECTION 9.2 REVIEW

TECHNIQUES

- Find the kernel of a given homomorphism from a group $[G, \cdot]$ to a group $[H, +]$.
- W Given a canonical $n \times r$ parity-check matrix \mathbf{H} , write the set of binary m -tuples that \mathbf{H} encodes and write the code word for each one.
- Given m such that Z_2^m is to be encoded as a single-error correcting code, find a canonical parity-check matrix to generate the code.
- W Given a canonical $n \times r$ parity-check matrix \mathbf{H} for a perfect code, be able to decode a received n -tuple by computing its syndrome and finding its coset leader.

MAIN IDEAS

- If f is a homomorphism from a group $[G, \cdot]$ to a group $[H, +]$, then $f(G)$ is a subgroup of $[H, +]$ and the kernel K (the set of all elements in G mapping to i_H) is a subgroup of $[G, \cdot]$.

- If $[S, +]$ is a subgroup of the group $[G, +]$, then the set of left cosets of S in G forms a partition of G .
- The error-detecting and error-correcting capabilities of a binary code are functions of the minimum distance of the code.
- In a group code, the minimum distance is the minimum weight of the nonzero code words.
- A parity-check matrix \mathbf{H} can be used to generate a group code, in which case the minimum distance of the code can be determined from \mathbf{H} .
- A canonical $n \times r$ parity-check matrix provides an easy procedure to encode Z_2^m in Z_2^n where $m = n - r$.
- For a group code generated by an $n \times r$ parity-check matrix \mathbf{H} , each word X in Z_2^n is decoded by using its syndrome to locate the coset in Z_2^n to which X belongs and adding the coset leader to X . If the code is a perfect code, the coset leader can be determined from \mathbf{H} .

EXERCISES 9.2

1. Let f be a homomorphism from a group $[G, \cdot]$ to a group $[H, +]$. Prove that $[f(G), +]$ is a subgroup of $[H, +]$.
2. Let f be a homomorphism from the group $[\mathbb{Z}, +]$ to the group $[\mathbb{Z}, +]$ given by $f(x) = 2x$.
 - a. Verify that f is a homomorphism.
 - b. Is f an isomorphism? Prove or disprove.
 - c. What is the subgroup $[f(\mathbb{Z}), +]$ of $[\mathbb{Z}, +]$?
3. The function f defined by $f(x) = x \cdot_8 2$ is a homomorphism from $[\mathbb{Z}, +]$ to $[\mathbb{Z}_8, +_8]$. Find the kernel K .
4. The function f defined by $f(x) = x \cdot_8 4$ is a homomorphism from $[\mathbb{Z}_{12}, +_{12}]$ to $[\mathbb{Z}_8, +_8]$. Find the kernel K .
5. A function $f: \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$ is defined by $f(x, y) = x + y$.
 - a. Prove that f is a homomorphism from the group $[\mathbb{Z} \times \mathbb{Z}, +]$ (where $+$ means componentwise addition) to the group $[\mathbb{Z}, +]$.
 - b. Find the kernel K .
6. Let F be the set of all functions $f: \mathbb{R} \rightarrow \mathbb{R}$. For $f, g \in F$, let $f + g$ be defined as follows: For $x \in \mathbb{R}$, $(f + g)(x) = f(x) + g(x)$.
 - a. Prove that $[F, +]$ is a group.
 - b. Let $a \in \mathbb{R}$. Define a function $\alpha: F \rightarrow F$ by $\alpha(f) = f(a)$. Prove that α is a homomorphism from $[F, +]$ to $[\mathbb{R}, +]$.
 - c. Find the kernel K .
7. Let $S = \{0, 4, 8\}$. Then $[S, +_{12}]$ is a subgroup of $[\mathbb{Z}_{12}, +_{12}]$. Find the members of the left coset $7 +_{12} S$.
8. Let $S = \{i, (2, 3)\}$. Then $[S, \circ]$ is a subgroup of the symmetric group $[S_3, \circ]$. Find the members of the left coset $(1, 2, 3) \circ S$ and the right coset $S \circ (1, 2, 3)$. Explain this result.

9. Consider the canonical parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Prove that the code generated by \mathbf{H} is single-error correcting.
- Write the set of binary m -tuples \mathbf{H} encodes and write the code word for each one.

10. Consider the canonical parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Prove that the code generated by \mathbf{H} is single-error correcting.
- Write the set of binary m -tuples \mathbf{H} encodes and write the code word for each one.

- Give an example of a canonical parity-check matrix that will generate a single-error correcting code for the set of words in Z_2^6 .
- Give a canonical parity-check matrix for a single-error correcting (15, 11) code.
- Let \mathbf{H} be an $n \times r$ binary matrix mapping $[Z_2^n, +_2]$ to $[Z_2^r, +_2]$ by the operation $X \cdot \mathbf{H}$ for X in Z_2^n . Prove that this function is a homomorphism by showing that for X, Y in $Z_2^n, (X +_2 Y) \cdot \mathbf{H} = X \cdot \mathbf{H} +_2 Y \cdot \mathbf{H}$.
- Which of the following are perfect codes? Which are single-error correcting?
 - (5, 3)
 - (12, 7)
 - (15, 11)
- Complete the coset leader/syndrome table of Example 28.
- Use your table from Exercise 15 and the matrix \mathbf{H} from Example 28 to decode the following received words.
 - 1010011
 - 0001110
 - 0101101

For Exercises 17 and 18, use a canonical parity-check matrix for the (15, 11) perfect code (see Exercise 12) to decode the given received words. (*Hint:* Because this is a perfect code, you do not need to generate the entire coset leader/syndrome table to solve these problems.)

- 011000010111001
- 110111001010011

SECTION 9.3 FINITE-STATE MACHINES

The algebraic structures of Section 9.1 served as models for various simple arithmetic systems. However, we would surely agree that computation should go beyond mere arithmetic. We would like a model that captures the general nature of computation. Perhaps looking at a simplified version of a modern digital computer would be a start.

A computer stores information internally in binary form. At any instant, the computer contains certain information, so its internal storage is set in certain patterns of binary digits, which we'll call the state of the computer at that instant. Because a computer contains a finite amount of storage, there is a finite (although large) number of different states that the computer can assume. An internal clock synchronizes

the actions of the computer. On a clock pulse, input can be read, which can change some of the storage locations and thus change the state of the machine to a new state. What the new state is will depend on what the input was, as well as what the previous state was. If these two factors are known, the change is predictable and nonrandom. Because the contents of certain storage cells are available as output, the state of the machine determines its output. In this way, over a succession of clock pulses, the machine produces a sequence of outputs in response to a sequence of inputs.

Definition

The finite-state machine is a model that captures the characteristics of a computer. As you read the definition, look for the following properties in the behavior of our abstract machine:

1. Operations of the machine are *synchronized* by discrete clock pulses.
2. The machine proceeds in a *deterministic* fashion; that is, its actions in response to a given sequence of inputs are completely predictable.
3. The machine responds to *inputs*.
4. There is a *finite number of states* that the machine can attain. At any given moment, the machine is in exactly one of these states. Which state it will be in next is a function of both the present state and the present input. The present state, however, depends on the previous state and input, while the previous state depends on its previous state and input, and so on, going all the way back to the initial configuration. Thus, the state of the machine at any moment serves as a form of memory of past inputs.
5. The machine is capable of *output*. The nature of the output is a function of the present state of the machine, meaning that it also depends on past inputs.

DEFINITION FINITE-STATE MACHINE

$M = [S, I, O, f_S, f_O]$ is a **finite-state machine** if S is a finite set of states, I is a finite set of input symbols (the **input alphabet**), O is a finite set of output symbols (the **output alphabet**), and f_S and f_O are functions where $f_S: S \times I \rightarrow S$ and $f_O: S \rightarrow O$. The machine is always initialized to begin in a fixed starting state s_0 .

The function f_S is the **next-state function**. It maps a (state, input) pair to a state. Thus, the state at clock pulse t_{i+1} , $\text{state}(t_{i+1})$, is obtained by applying the next-state function to the state at time t_i and the input at time t_i :

$$\text{state}(t_{i+1}) = f_S(\text{state}(t_i), \text{input}(t_i))$$

The next-state function is indeed a function, so for any (state, input) pair, there is a unique next state. The function f_O is the **output function**. When f_O is applied to a state at time t_i , we get the output at time t_i :

$$\text{output}(t_i) = f_O(\text{state}(t_i))$$

Notice that the effect of applying function f_O is available instantly, but the effect of applying function f_S is not available until the next clock pulse.

Examples of Finite-State Machines

To describe a particular finite-state machine, we have to define the three sets and two functions involved.

EXAMPLE 29

A finite-state machine M is described as follows: $S = \{s_0, s_1, s_2\}$, $I = \{0, 1\}$, $O = \{0, 1\}$. Because the two functions f_S and f_O act on finite domains, they can be defined by a **state table**, as in Table 9.1. The machine M begins in state s_0 , which has an output of 0. If the first input symbol is a 0, the next state of the machine is then s_1 , which has an output of 1.

Present state	Next state		Output
	Present input 0	1	
s_0	s_1	s_0	0
s_1	s_2	s_1	1
s_2	s_2	s_0	1

If the next input symbol is a 1, the machine stays in state s_1 with an output of 1. By continuing this analysis, we see that an input sequence consisting of the characters 01101 (read left to right) would produce the following effect:

Time	t_0	t_1	t_2	t_3	t_4	t_5
Input	0	1	1	0	1	–
State	s_0	s_1	s_1	s_1	s_2	s_0
Output	0	1	1	1	1	0

The initial 0 of the output string is spurious—it merely reflects the starting state, not the result of any input. In a similar way, the input sequence 1010 produces an output of 00111.

Another way to define the functions f_S and f_O (in fact all of M) is by a directed graph called a state graph. Each state of M with its corresponding output is the label of a node of the graph. The next-state function is given by directed arcs of the graph, each arc showing the input symbol(s) that produces that particular state change. The state graph for M appears in Figure 9.3.

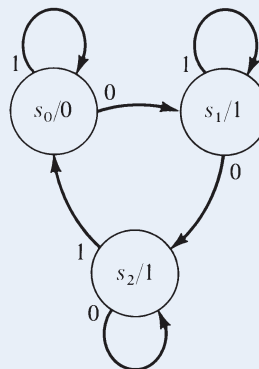


Figure 9.3

REMINDER

Each input symbol must appear on one and only one transition arc from each state.

PRACTICE 43 For the machine M of Example 29, what output sequence is produced by the input sequence 11001?

PRACTICE 44 A machine M is given by the state graph of Figure 9.4. Give the state table for M .

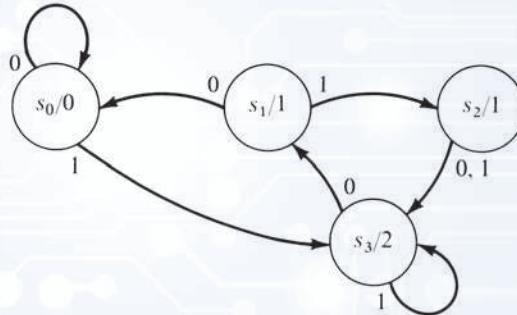


Figure 9.4

PRACTICE 45 A machine M is described by the state table shown in Table 9.2.

- Draw the state graph for M .
- What output corresponds to an input sequence of 2110?

Present state	Next state			Output
	Present input			
	0	1	2	
s_0	s_0	s_1	s_1	0
s_1	s_1	s_0	s_0	1

The machine of Example 29 is not particularly interesting. If finite-state machines model real-world computers, they should be able to do something. Let's try to build a finite-state machine that will add two binary numbers. The input will consist of a sequence of pairs of binary digits, each of the form 00, 01, 10, or 11. Each pair represents one column of digits of the two numbers to be added, least significant digits first (so in this particular case, read the inputs column by column right to left). Thus to add the two numbers

$$\begin{array}{r} 011 \\ 101 \end{array}$$

the number pairs are 11, 10, and 01. The output gives the least significant digits of the answer first. Recall the basic facts of binary addition:

$$\begin{array}{r} 0 \quad 0 \quad 1 \quad 1 \\ 0 \quad 1 \quad 0 \quad 1 \\ \hline 0 \quad 1 \quad 1 \quad 10 \end{array}$$

(Note that in the fourth addition a carry to the next column takes place.)

A moment's thought shows us that we can encounter four cases in adding the digits in any given column, and we will use states of the machine to represent these cases.

- The output should be 0 with no carry—state s_0 .
- The output should be 0 but with a carry to the next column—state s_1 .
- The output should be 1 with no carry—state s_2 .
- The output should be 1 with a carry to the next column—state s_3 .

State s_0 , as always, is the starting state. We have already indicated the output for each state, but we need to determine the next state based on the present state and the input. For example, suppose we are in state s_1 and the input is 11. The output for the present state is 0, but there is a carry, so in the next column we are adding $1 + 1 + 1$, which results in an output of 1 and a carry. The next state is s_3 .

PRACTICE 46 In the binary adder under construction:

- What is the next state if the present state is s_2 and the input is 11?
- What is the next state if the present state is s_3 and the input is 10?

After considering all possible cases, we have the complete state graph of Figure 9.5.

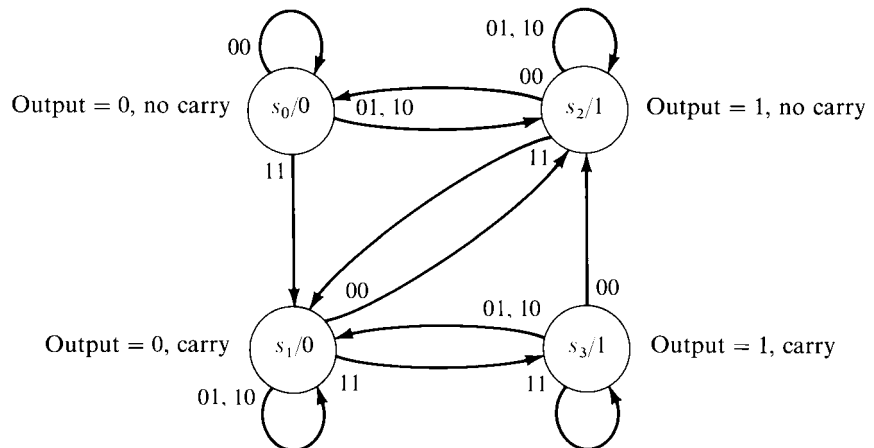


Figure 9.5

The operation of this machine in adding the two numbers 011 and 101 (read right to left, that is, low-order digits first) can be traced as follows:

Time	t_0	t_1	t_2	t_3	t_4
Input	11	10	01	00	–
State	s_0	s_1	s_1	s_1	s_2
Output	0	0	0	0	1

The output (written right to left) is 1000 when we ignore the initial 0, which does not reflect the action of any input. Converting this arithmetic to decimal form, we have computed $3 + 5 = 8$. Note the symmetry of this machine with respect to the inputs of 10 and 01, reflecting that binary addition is commutative.

PRACTICE 47

Compute the sum of 01110110 and 01010101 by using the binary adder machine of Figure 9.5.

Recognition

We have already noted that a given input signal may affect the behavior of a finite-state machine for longer than just one clock pulse. Because of the (limited) memory of past inputs represented by the states of a machine, we can use these machines as *recognizers*. A machine can be built to recognize, say by producing an output of 1, when the input it has received matches a certain description. We will soon discuss more fully the capabilities of finite-state machines as recognizers. Here we will simply construct some examples.

EXAMPLE 30

When binary data are transmitted (or stored) as strings of bits, an extra bit is often tacked onto the end of each string as a parity bit. Under an even parity scheme, the parity bit (a 0 or a 1) is chosen so that the total number of 1s in the string, including the parity bit, is an even number. Under an odd parity scheme, the parity bit is chosen so that the total number of 1s in the string is odd. When the string is received, its parity is checked. Assuming an even parity scheme, if the parity of the received string is not even, then an error has occurred in transmitting the string, and a request for retransmission can be made. A parity bit therefore serves as a simple single-error-detection mechanism. (Note that a parity bit does not detect errors in 2 bits, although it does detect errors in 3 bits.) The machine described in Figure 9.6 is a parity check machine. When the input received through time t_i contains an even number of 1s, then the output at time t_{i+1} is 1; otherwise, the output is 0.

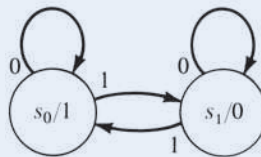


Figure 9.6

EXAMPLE 31

Suppose we want to design a machine having an output of 1 exactly when the input string received to that point ends in 101. As a special case, an input sequence consisting of just 101 could be handled by progressing directly from state s_0 to states s_1 , s_2 , and s_3 with outputs of 0 except for s_3 , which has an output of 1. This much of the design results in Figure 9.7a. This figure shows that we want to be in state s_2 whenever the input has been such that one more 1 takes us to s_3 (with an output

of 1); thus we should be in s_2 whenever the two most recent input symbols were 10, regardless of what came before. In particular, a string of 1010 should put us in s_2 ; hence, the next-state function for s_3 with an input of 0 is s_2 . Similarly, we can use s_1 to “remember” that the most recent input symbol received was 1, and that a 01 will take us to s_3 . In particular, 1011 should put us in s_1 ; hence, the next-state function for s_3 with an input of 1 is s_1 . The rest of the next state function can be determined the same way; Figure 9.7b shows the complete state graph.

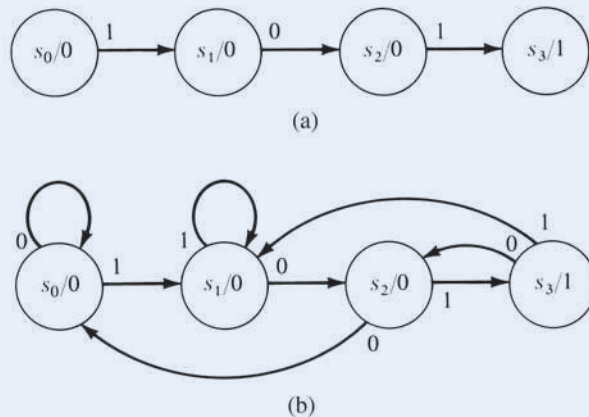


Figure 9.7

Notice that the machine is in state s_2 at the end of an input of 0110 and at the end of an input of 011010—in fact, at the end of any input ending in 10; yet s_2 cannot distinguish between these inputs. *Each state of M represents a class of indistinguishable input histories, s_3 being the state representing all inputs ending in 101.*

PRACTICE 48 Draw the state graph for a machine producing an output of 1 exactly when the input string received to that point ends in 00.

Now we want to see exactly what sets finite-state machines can recognize. Remember that recognition is possible because machine states have a limited memory of past inputs. Even though the machine is finite, a particular input signal can affect the behavior of a machine “forever.” However, not every input signal can do so, and some classes of inputs require remembering so much information that no machine can detect them.

To avoid writing down outputs, we will designate those states of a finite-state machine with an output of 1 as **final states** and denote them in the state graph with a double circle. Then we can give the following formal definition of recognition, where I^* denotes the set of finite-length strings over the input alphabet.

DEFINITION FINITE-STATE MACHINE RECOGNITION
 A finite-state machine M with input alphabet I **recognizes** a subset S of I^* if M , beginning in state s_0 and processing an input string α , ends in a final state if and only if $\alpha \in S$.

PRACTICE 49 Describe the sets recognized by the machines in Figure 9.8.

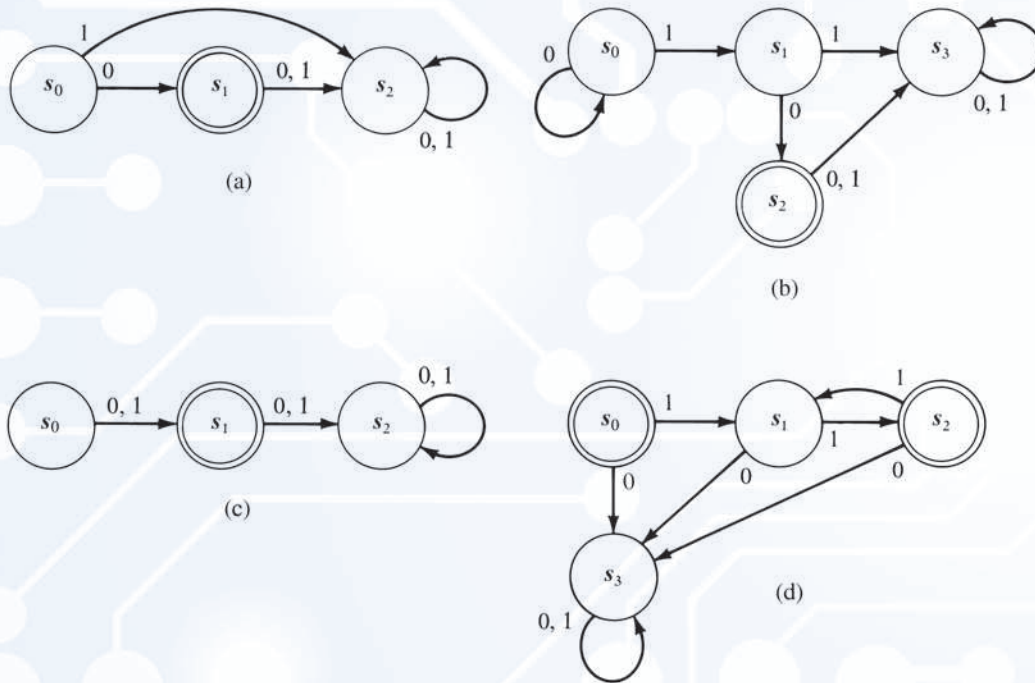


Figure 9.8

In Practice 49d, state s_3 is a “dead” state from which there is no recovery. The appearance of a 0 in states s_0, s_1 , or s_2 irrevocably ruins the pattern the machine is able to recognize. In contrast, the machine of Example 31 (or any machine that recognizes strings with a certain ending) will never have a dead state because there is always hope that subsequent input will match the ending pattern.

Regular Sets and Kleene’s Theorem

We want a compact, symbolic way to describe sets such as those appearing in the answer to Practice 49. We will describe such sets by using *regular expressions*; each regular expression describes a particular set. First, we define what regular expressions are; then we will see how a regular expression describes a set. We assume here that I is some finite set of symbols; later I will be the input alphabet for a finite-state machine.

DEFINITION **REGULAR EXPRESSIONS OVER I**
Regular expressions over I are the

1. symbol \emptyset and the symbol λ .
2. symbol i for any $i \in I$.
3. expressions (AB) , $(A \vee B)$, and $(A)^*$ if A and B are regular expressions.

(This definition of a regular expression over I is still another example of a recursive definition.)

● **DEFINITION** **REGULAR SET**

Any set represented by a regular expression according to the following conventions is a **regular set**:

1. \emptyset represents the empty set.
2. λ represents the set $\{\lambda\}$ containing the empty string.
3. i represents the set $\{i\}$.
4. For regular expressions A and B ,
 - a. (AB) represents the set of all elements of the form $\alpha\beta$ where α belongs to the set represented by A and β belongs to the set represented by B .
 - b. $(A \vee B)$ represents the union of A 's set and B 's set.
 - c. $(A)^*$ represents the set of all concatenations of members of A 's set.

In our discussion, we will be a little sloppy and say things like “the regular set AB ” instead of “the set represented by the regular expression AB .” Informally, an element in AB is an item from A followed by an item from B . An element in $A \vee B$ is a single item chosen from either A or B . An element in $(A)^*$ is zero or more repetitions of elements from A . We note that λ , the empty string, is a member of the set represented by A^* for any A because it is the case of zero repetitions of elements from A . In writing regular expressions, we can eliminate parentheses when no ambiguity results. The regular expression $0^* \vee 10$ therefore consists of $\lambda, 0, 00, 000, 0000, \dots, 10$.

EXAMPLE 32

Here are some regular expressions and a description of the set each one represents.

- | | |
|---------------------------|--|
| a. $1^*0(01)^*$ | Any number (including none) of 1s, followed by a single 0, followed by any number (including none) of 01 pairs. |
| b. $0 \vee 1^*$ | A single 0 or any number (including none) of 1s. |
| c. $(0 \vee 1)^*$ | Any string of 0s or 1s, including λ . |
| d. $11((10)^*11)^*(00)^*$ | A nonempty string of pairs of 1s interspersed with any number (including none) of 10 pairs, followed by at least one 0. One can see how awkward this verbal description is and appreciate the compactness of a regular expression description. |

PRACTICE 50

Which strings belong to the set described by the regular expression?

- a. 10100010; $(0^*10)^*$
- b. 011100; $(0 \vee (11)^*)^*$
- c. 000111100; $((011 \vee 11)^*(00)^*)^*$

PRACTICE 51

Write regular expressions for the sets recognized by the machines of Practice 49.

A regular set may be described by more than one regular expression. For example, the set of all strings of 0s and 1s, which we already know from Example 32(c) to be described by $(0 \vee 1)^*$, is also described by the regular expression $[(0 \vee 1^*)^* \vee (01)^*]^*$. We might, therefore, write the equation

$$(0 \vee 1)^* = [(0 \vee 1^*)^* \vee (01)^*]^*$$

Although we may be quite willing to accept this particular equation, it can be difficult to decide in general whether two regular expressions are equal, that is, whether they represent the same set. An efficient algorithm that will make this decision for any two regular expressions has not been found.

We have introduced regular sets because, as it turns out, these are exactly the sets finite-state machines are capable of recognizing. This result was first proved by the American mathematician Stephen Kleene in 1956. We state his theorem here without proof.

● **THEOREM** **KLEENE'S THEOREM**

Any set recognized by a finite-state machine is regular, and any regular set can be recognized by some finite-state machine.

Kleene's theorem outlines the limitations as well as the capabilities of finite state machines—there are certainly many sets that are not regular. For example, consider $S = \{0^n 1^n \mid n \geq 0\}$ where a^n stands for a string of n copies of a . Strings in S have some number of 0s followed by the same number of 1s. S is not regular. (Notice that 0^*1^* does not do the job.) By Kleene's theorem, there is no finite-state machine capable of recognizing S . Yet S seems like such a reasonable set, and surely we humans could count a string of 0s followed by 1s and see whether we had the same number of 1s as 0s. This lapse suggests some deficiency in our use of a finite-state machine as a model of computation. We will investigate this further in Section 9.4.

Machine Minimization

Although we have treated finite-state machines as abstractions, circuits that act like finite-state machines can be built from electronic devices like the logic elements of Section 8.2 and others. If we wish to construct a physical machine, the number of internal states is a factor in the cost of construction. Minimization is the process of finding, for a given finite-state machine M , a machine M' with two properties:

1. If M and M' are both begun in their respective start states and are given the same sequence of input symbols, they will produce identical output sequences.
2. M' has, if possible, fewer states than M (if this is not possible, then M is already a minimal machine and cannot be further reduced).

Unreachable States

First, let's observe that we can remove any **unreachable states** of M , those states that cannot be attained from the starting state no matter what input sequence occurs.

EXAMPLE 33

Let M be given by the state table of Table 9.3. Although the state table contains the same information as the state graph (Figure 9.9), the graph shows us at a glance that state s_2 can never be reached from the starting state s_0 . If we simply remove state s_2 and its associated arcs, we have the state graph of Figure 9.10 for a machine M' with one less state than M that behaves exactly like M ; that is, it gives the same output as M for any input string.

Present state	Next state		Output
	Present input 0	1	
s_0	s_1	s_3	0
s_1	s_3	s_0	0
s_2	s_1	s_3	1
s_3	s_0	s_1	1

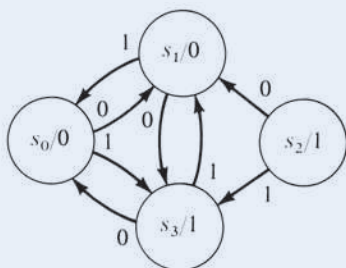


Figure 9.9

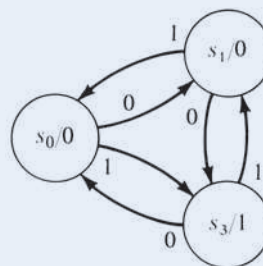


Figure 9.10

PRACTICE 52

What state(s) is/are unreachable from s_0 in the machine of Table 9.4? Try to get your answer directly from the state table.

Present state	Next state		Output
	Present input 0	1	
s_0	s_1	s_4	0
s_1	s_4	s_1	1
s_2	s_2	s_2	1
s_3	s_3	s_1	0
s_4	s_0	s_0	1

Because the state graph of a finite-state machine is a directed graph, it has an associated adjacency matrix. Warshall's algorithm (Section 7.1) can be used to detect unreachable states.

Minimization Procedure

Assuming now that all unreachable states have been removed from M , we will continue to look for a reduced machine M' . The key to finding a reduced M' , if one exists, lies in the notion of equivalent states.

DEFINITION EQUIVALENT STATES

Two states s_i and s_j of M are **equivalent** if for any $\alpha \in I^*$, $f_O(s_i, \alpha) = f_O(s_j, \alpha)$, where I^* again denotes the set of finite-length strings over the input alphabet.

In this definition of equivalent states, the definition of the output function has been extended to denote the sequence of output symbols obtained by repeatedly applying f_O to a sequence α of input symbols. Thus equivalent states of a machine produce identical output strings for any input string.

PRACTICE 53 Prove that state equivalence is an equivalence relation on the states of a machine. ■

For the time being, we will postpone the problem of how to identify equivalent states in a given machine M . Let's simply assume that we have somehow found which states are equivalent and have partitioned the states of M into the corresponding equivalence classes. These classes have two properties: (1) All states in the same class have the same output, and (2) for each input symbol, all states in the same class proceed under the next-state function to states that are all in the same class.

PRACTICE 54 Show that properties 1 and 2 are satisfied when M is partitioned into classes of equivalent states. ■

We define a machine M' whose states are the equivalence classes of M . M' has the same input and output alphabet as M , and its start state is the class to which s_0 , the start state of M , belongs. The output of a class is the output symbol common to all states of M in that class (property 1). The next state of class X under an input symbol is that class to which all states of M in X proceed under that input symbol (property 2). M' is a well-defined machine. M' produces the same output strings when processing a given input string as does M . Also, the number of states of M' (equivalence classes of M) will be no greater than the number of states of M .

The minimization problem for M thus boils down to finding the equivalent states of M . Perhaps we should note first that the obvious approach of directly trying to satisfy the definition of equivalent states will not work. Given two states s_i and s_j of M , we cannot actually compare the outputs corresponding to each possible input string. Fortunately, the problem is not as infinite as it sounds; we only need to identify k -equivalent states.

DEFINITION K-EQUIVALENT STATES

Two states s_i and s_j of M are **k -equivalent** if for any $\alpha \in I^*$ where α has no more than k symbols, $f_O(s_i, \alpha) = f_O(s_j, \alpha)$.

It is not hard to see that k -equivalence is an equivalence relation on the states of M (check the reflexive, symmetric and transitive properties). It is possible to test two states of M for k -equivalence directly, since we can actually produce the finite number of input strings having no more than k symbols. However, it turns out that we don't have to do this. We can begin by finding 0-equivalent states. These are states producing the same output for 0-length input strings, that is, states having the same associated output symbol. Thus, we can identify the 0-equivalence classes directly from the description of M .

EXAMPLE 34

Let M be defined by the state table of Table 9.5. (Here we've started writing 0, 1, 2 ... for states instead of s_0, s_1, s_2, \dots .) The 0-equivalence classes of the states of M are

$$\{0, 2, 5\} \text{ and } \{1, 3, 4, 6\}$$

Present state	Next state		Output
	Present input		
	0	1	
0	2	3	0
1	3	2	1
2	0	4	0
3	1	5	1
4	6	5	1
5	2	0	0
6	4	0	1

Our procedure to find k -equivalent states is a recursive one; we know how to find 0-equivalent states, and we will show how to find k -equivalent states once we have identified states that are $(k - 1)$ -equivalent. Suppose, then, that we already know which states are $(k - 1)$ -equivalent. If states s_i and s_j are k -equivalent, they must produce the same output strings for any input string of length k or less, which includes any string of length $k - 1$ or less. Thus, s_i and s_j must at least be $(k - 1)$ -equivalent. But they also must produce the same output strings for any k -length input string.

An arbitrary k -length input string consists of a single arbitrary input symbol followed by an arbitrary $(k - 1)$ -length input string. If we apply such a k -length string to states s_i and s_j (which themselves have the same output symbol), the single input symbol moves s_i and s_j to next states s'_i and s'_j ; then s'_i and s'_j must produce identical output strings for the remaining, arbitrary $(k - 1)$ -length string, which will surely happen if s'_i and s'_j are $(k - 1)$ -equivalent. Therefore, to find k -equivalent states, look for $(k - 1)$ -equivalent states whose next states under any input symbol are $(k - 1)$ -equivalent.

EXAMPLE 35

Consider again the machine M of Example 34. We know the 0-equivalent states. To find 1-equivalent states, we look for 0-equivalent states with 0-equivalent next states. For example, the states 3 and 4 are 0-equivalent; under the input symbol 0, they proceed to states 1 and 6, respectively, which are 0-equivalent states, and under the input symbol 1 they both proceed to 5, which of course is 0-equivalent to itself. Therefore, states 3 and 4 are 1-equivalent. Similarly, states 1 and 3 are 1-equivalent and states 4 and 6 are 1-equivalent. Therefore the class $\{1, 3, 4, 6\}$ of 1-equivalent states is unchanged from the class of 0-equivalent states. But states 0 and 5, themselves 0-equivalent, proceed under the input symbol 1 to states 3 and 0, respectively, which are not 0-equivalent states. So states 0 and 5 are not 1-equivalent; the input string 1 will produce an output string of 01 from state 0 and of 00 from state 5. States 0 and 2 are 1-equivalent. Therefore the 1-equivalence classes for M are

$$\{0, 2\}, \{5\}, \{1, 3, 4, 6\}$$

To find 2-equivalent states, we look for 1-equivalent states with 1-equivalent next states. States 1 and 3, although 1-equivalent, proceed under input 1 to states 2 and 5, respectively, which are not 1-equivalent states. Therefore, states 1 and 3 are not 2-equivalent. The states 0 and 2, 1 and 6, and 3 and 4, respectively, are 2-equivalent. The 2-equivalence classes for M are

$$\{0, 2\}, \{5\}, \{1, 6\}, \{3, 4\}$$

The 3-equivalence classes for M are the same as the 2-equivalence classes. ●

DEFINITION PARTITION REFINEMENT

Given two partitions π_1 and π_2 of a set S , π_1 is a **refinement** of π_2 if each block of π_1 is a subset of a block of π_2 .

In Example 35 each successive partition of the states of M into equivalence classes is a refinement of the previous partition. This refinement will always happen; k -equivalent states must also be $(k - 1)$ -equivalent, so the blocks of the $(k - 1)$ -partition can only be further subdivided. However, the subdivision process cannot continue indefinitely (at worst it can go on only until each partition block contains only one state); there will eventually be a point where $(k - 1)$ -equivalent states and k -equivalent states coincide. (In Example 35, 2-equivalent and 3-equivalent states coincide.) Once this happens, all next states for members of a partition block under any input symbol fall within a partition block. Thus, k -equivalent states are also $(k + 1)$ -equivalent and $(k + 2)$ -equivalent, and so on. Indeed, these states are equivalent.

The total procedure for finding equivalent states is to start with 0-equivalent states, then 1-equivalent states, and so on, until the partition no longer subdivides. A pseudocode description of this algorithm is given. It isn't nearly as complex as it looks, but it does involve checking many pairs of states, just as in Example 35.

ALGORITHM MINIMIZE

```

Minimize (finite-state machine table  $M$ )
//produces a minimized version of  $M$ 
Local variable:
boolean  $flag$  //flag for loop exit when nonequivalent states found

find 0-equivalent states of  $M$ 
repeat
  while untested equivalence classes remain do
    select untested equivalence class
    while untested state pairs in current class remain do
      select untested state pair in current class
       $flag = false$ 
      while untried input symbols remain and not  $flag$  do
        select untried input symbol
        for both states in current pair, find next state
          under current input symbol
        if next states not equivalent then
           $flag = true$ 
        end if
      end while
    end while
    if  $flag$  then
      mark current states for different classes;
    end if
  end while
  form new equivalence classes
end while
until set of new equivalence classes = set of old equivalence classes
end Minimize

```

EXAMPLE 36

For the machine M of Examples 34 and 35, the reduced machine M' will have states

$$A = \{0, 2\}$$

$$B = \{5\}$$

$$C = \{1, 6\}$$

$$D = \{3, 4\}$$

The state table for M' (Table 9.6) is obtained from that for M . Machine M' (starting state A) will reproduce M 's output for any input string, but it has four states instead of seven.

Present state	Next state		Output
	Present input		
	0	1	
A	A	D	0
B	A	A	0
C	D	A	1
D	C	B	1

EXAMPLE 37

We will minimize M where M is given by the state table of Table 9.7.

Present state	Next state		Output
	Present input		
	0	1	
0	3	1	1
1	4	1	0
2	3	0	1
3	2	3	0
4	1	0	1

The 0-equivalence classes of M are

$$\{0, 2, 4\}, \{1, 3\}$$

States 0 and 2 under input 1 go to states 1 and 0, which are not 0-equivalent. The 1-equivalence classes of M are

$$\{0\}, \{2, 4\}, \{1, 3\}$$

No further refinement is possible. Let

$$A = \{0\}$$

$$B = \{2, 4\}$$

$$C = \{1, 3\}$$

The reduced machine is shown in Table 9.8.

Present state	Next state		Output
	Present input		
	0	1	
A	C	C	1
B	C	A	1
C	B	C	0

PRACTICE 55 Minimize the machines whose state tables are shown in Tables 9.9 and 9.10.

Present state	Next state		Output
	Present input		
	0	1	
0	2	1	1
1	2	0	1
2	4	3	0
3	2	3	1
4	0	1	0

Present state	Next state		Output
	Present input		
	0	1	
0	1	3	1
1	2	0	0
2	1	3	0
3	2	1	0

Sequential Networks and Finite-State Machines

The output of a finite-state machine is a function of its present state, and the present state of the machine is a function of past inputs. Thus, the states of a machine have certain memory capabilities. In the combinational networks of Chapter 8, which use AND gates, OR gates, and inverters, the output is virtually instantaneous and a function only of the present input. To build a finite-state machine we need one additional element that provides the memory missing from our previous logic networks.

A **delay element** is the simplest of a class of elements known as *flip-flops*. It is regulated by a clock, has a single binary input, and its output at time $t + 1$ is the input signal it received at time t . The delay element is therefore a “memory device” that captures input for the duration of one clock pulse. Figure 9.11 represents the delay element at time $t + 1$, with the signal propagating from right to left. When the delay element is receiving an input of $d(t + 1)$, the output is the previous input $d(t)$.



Figure 9.11

When one or more delay elements are introduced into a combinational network, the network is known as a **sequential network**. Unlike a combinational network, loops (where output from a circuit becomes part of its input) are allowed, provided that at least one delay element is incorporated into the loop. The delay element prevents the confusion that results when a circuit tries to act on its current output. Input sequences can be run through sequential networks (hence the name) provided that the clock pulse synchronizes input signals as well as delay elements.

EXAMPLE 38

The delay element in Figure 9.12 feeds the output from the terminal AND gate of the network back into the initial OR gate at the next clock pulse. The initial output of the delay element is assumed to be 0. The input sequences $x_1 = 10010$ and $x_2 = 11000$ (read left to right) produce the effect shown in the table.

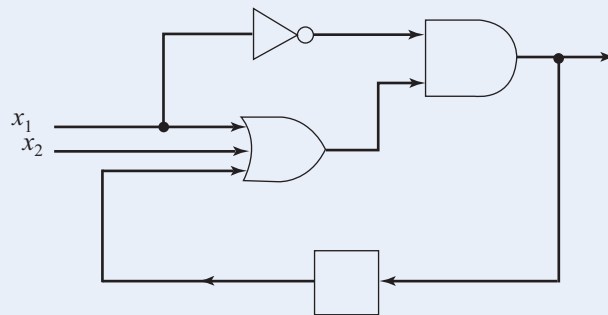


Figure 9.12

Time	t_0	t_1	t_2	t_3	t_4
x_1	1	0	0	1	0
x_2	1	1	0	0	0
Delay output	0	0	1	1	0
Circuit output	0	1	1	0	0

Any finite-state machine can be built using a sequential network. (We'll assume that all input and output values are binary; if not, they can be encoded in binary form.) The general structure of such a network is shown in Figure 9.13. It consists of two parts: (1) a combinational network (no delay elements) and (2) some loops containing all the delay elements.

To build the network for a given finite-state machine, we represent each machine state as a binary number, beginning with zero (0000, 0001, 0010, 0011, 0100, and so on); the assignment of states to numbers is arbitrary. At any moment, each delay element in the network has a 0 or 1 signal on its output line, so the collection of these output signals is a binary number and therefore represents one of the states of the machine. As these signals feed into the combinational network, they represent the current state; circuits within the combinational network compute the next state, which is the pattern of 0s and 1s on the input lines to the delay elements.

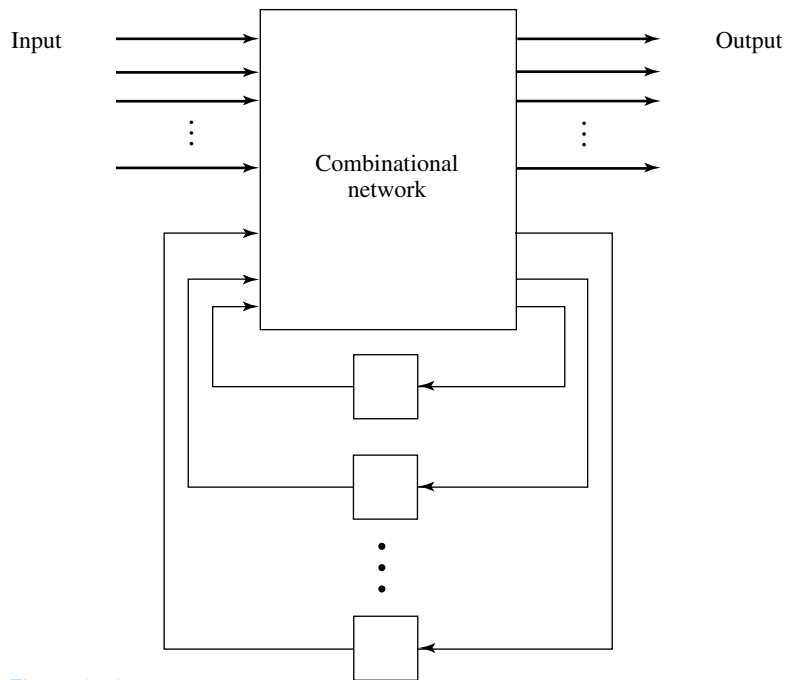


Figure 9.13

If the finite-state machine has q states, the binary numbering of the states requires a certain number of bits. For example, to represent six different states requires 3-bit numbers ranging from 000 to 101. To represent q different states requires $\lceil \log_2 q \rceil$ bits. Each bit in the binary number is the signal from a delay element, so the network will require $\lceil \log_2 q \rceil$ delay elements. The essence of the construction is to translate the state table of the finite-state machine into truth functions for the outputs and delay elements, and then to construct the logic network for each truth function, as we did in Chapter 8.

EXAMPLE 39

Consider the finite-state machine of Example 29, with state table as shown in Table 9.11.

Present state	Next state		Output
	Present input		
	0	1	
s_0	s_1	s_0	0
s_1	s_2	s_1	1
s_2	s_2	s_0	1

Because there are only two input symbols and two output symbols, only one input line x (taking on values of 0 or 1) and one output line y are needed for the network. There are three states, so we need $\lceil \log_2 3 \rceil = 2$ delay elements. We arbitrarily associate states with binary numbers (see Table 9.12) represented as signals on the inputs or outputs of the delay elements.

	d_1	d_2
s_0	0	0
s_1	0	1
s_2	1	0

Now we use the information contained in the state table to write three truth functions. One truth function describes the behavior of the output $y(t)$; it is a function of the two variables $d_1(t)$ and $d_2(t)$ representing the present state. The other two truth functions describe the behavior of $d_1(t + 1)$ and $d_2(t + 1)$, representing the next state; these are functions of $x(t)$, $d_1(t)$, and $d_2(t)$, the present input and the present state. Table 9.13 shows these truth functions.

$x(t)$	$d_1(t)$	$d_2(t)$	$y(t)$	$d_1(t + 1)$	$d_2(t + 1)$
0	0	0	0	0	1
1	0	0	0	0	0
0	0	1	1	1	0
1	0	1	1	0	1
0	1	0	1	1	0
1	1	0	1	0	0

In constructing the third line of Table 9.13, for example, $x(t) = 0$, $d_1(t) = 0$, and $d_2(t) = 1$, meaning that the present input is 0 and the present state is s_1 . The output associated with state s_1 is 1, so $y(t) = 1$. The next state associated with input 0 and present state s_1 is s_2 , so $d_1(t + 1) = 1$ and $d_2(t + 1) = 0$. Notice that there are some don't-care conditions for these functions because the configuration $d_1(t) = 1$ and $d_2(t) = 1$ does not occur.

The canonical sum-of-products form for each of these truth functions is

$$\begin{aligned}
 y(t) &= d_1'd_2 + d_1d_2' && (y \text{ is not a function of } x) \\
 d_1(t + 1) &= x'd_1'd_2 + x'd_1d_2' \\
 d_2(t + 1) &= x'd_1'd_2' + xd_1'd_2
 \end{aligned}$$

Using a Karnaugh map and appropriate choices for the don't-care conditions, these expressions can be simplified to

$$\begin{aligned}
 y(t) &= d_1 + d_2 \\
 d_1(t + 1) &= x'd_1 + x'd_2 = x'(d_1 + d_2) \\
 d_2(t + 1) &= xd_2 + x'd_1d_2'
 \end{aligned}$$

The logic networks for these expressions go into the “combinational network” box in Figure 9.13. Thus Figure 9.14 is a wiring diagram for the finite-state machine.

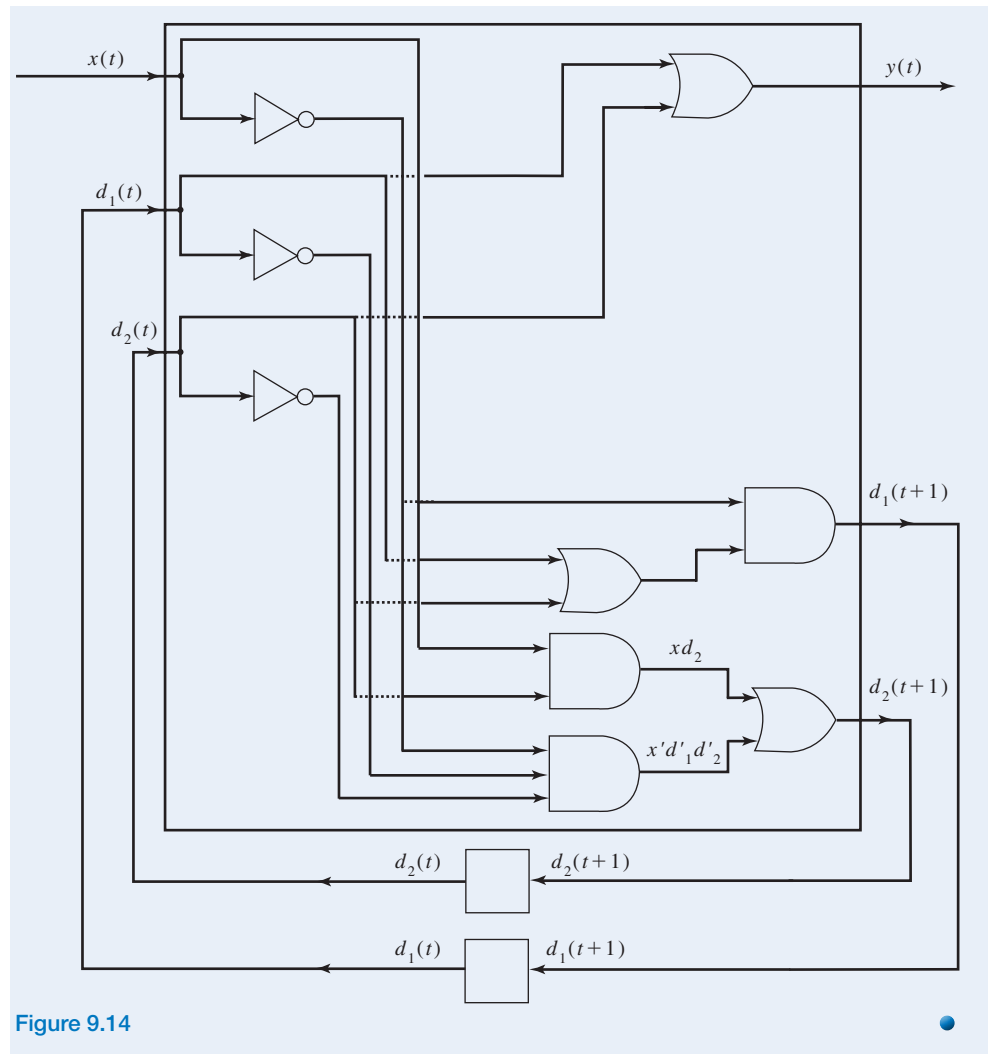


Figure 9.14

PRACTICE 56 Construct a sequential network for the parity check machine of Example 30.

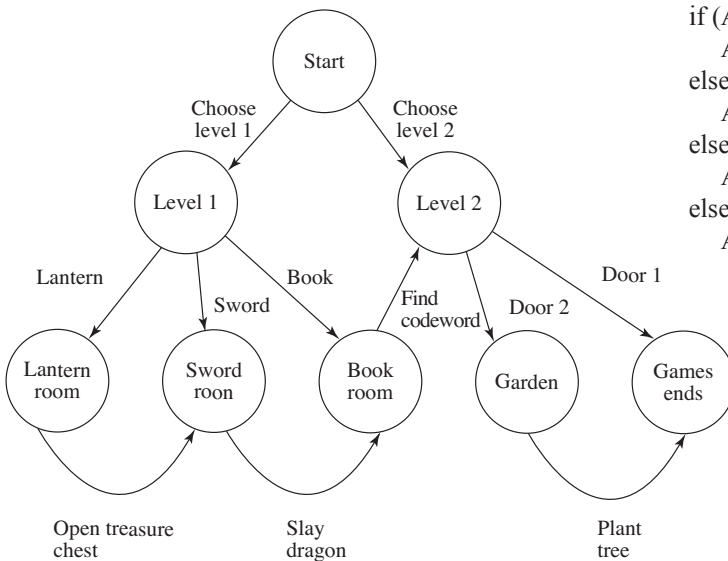
FSMs Behind the Game

Everyone is familiar with the use of storyboards in designing a movie. Storyboards are sequences of illustrated panels that set the scene, sketch the action, and outline the dialogue. Thumbing rapidly through pages of such panels creates a “flipbook,” a primitive animation of the movie. So these panels, laid end-to-end, are a one-dimensional layout of the movie.

What differentiates a video game from a movie? A movie (a film) is static in the sense that it always comes out the same way; there are no variations in the scenes or changes in the ending. A video game incorporates user interaction, and based on user input, different outcomes can occur at various points throughout the game. Design of a video game requires a two-dimensional approach.

If you freeze the video game at some point where it is waiting for user input, there is a certain scene on the screen with particular characters in place, and these characters may have certain attributes at this point in time. We can consider this to be the “state” of the game at the moment. User input is provided, and based on that input and the current state, something happens to move the game to a predefined new state. Sound familiar? Yes, this is essentially a finite-state machine. “Finite state machines are the nuts and bolts of game AI.”¹

Here’s an outline, written as an English paragraph, of a simple video game: The user starts the game by



choosing to play at level 1 or level 2. In this simple game, the only entity is the user’s avatar, call it object *A*. A user who chooses level 1 sends *A* into a room where *A* must pick up one of three objects: a book, a sword, or a lantern. If the book is chosen, *A* enters a room where the book can be searched for a secret code word. When the code word is found, *A* moves on to the opening screen for level 2. If the sword is chosen, *A* enters a room with a fearsome dragon. When *A* slays the dragon, *A* receives the book and enters the book room to search for the code word. If the lantern is chosen, *A* enters a room to search for a treasure chest. When the treasure chest is opened, it contains a sword and *A* moves on to the room with the fearsome dragon. At level 2, *A* chooses between door 1 and door 2. Choosing door 1 ends the game. Choosing door 2 moves *A* into a garden, where *A* must find and plant a lilac tree. Upon planting the tree, the game ends.

A state graph for a finite-state machine to model the game behavior described above looks something like the diagram shown.

Ultimately this design has to be translated into computer code. Details will vary with the programming language used, of course, but in any event there must be a way to make multiple choices from a given state depending on the user action:

```

if (A.state = level1 and A.pick = lantern)
    A.state = LanternRoom
else if (A.state = level1 and A.pick = sword)
    A.state = SwordRoom
else if (A.state = level1 and A.pick = book)
    A.state = BookRoom
else if (A.state = level2 and A.pick = Door1)
    A.state = GameEnds
  
```

and so forth.

In a more complex video game there will be many entities, and the computer code must go through a similar decision process for each one to program its correct action based on its current state and user input. The environment itself may be an entity, requiring changes of background (from the lantern room to the sword room, for example).

¹AI for Game Development, David M Bourg and Glenn Seemann, O’Reilly Media, Inc., 2004, ISBN-13: 978-0-596-00555-9

SECTION 9.3 REVIEW

TECHNIQUES

- Compute the output string for a given finite-state machine and a given input string.
- Draw a state graph from a state table and vice versa.
- **W** Construct a finite-state machine to act as a recognizer for a certain type of input.
- Find a regular expression given the description of a regular set.
- Decide whether a given string belongs to a given regular set.
- Minimize finite-state machines.
- Construct sequential networks for finite-state machines.

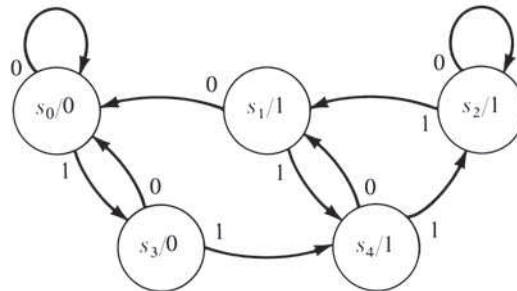
MAIN IDEAS

- Finite-state machines have a synchronous, deterministic mode of operation and limited memory capabilities.
- The class of sets that finite-state machines can recognize is the class of all regular sets; hence, their recognition capabilities are limited.
- Unreachable states can be removed from a machine.
- After unreachable states have been removed from a machine, a minimized version of that machine can be found that produces the same output strings for all input strings.
- Any finite-state machine can be built using a network of AND gates, OR gates, inverters, and delay elements.

EXERCISES 9.3

1. For each input sequence and machine given, compute the corresponding output sequence (starting state is always s_0).

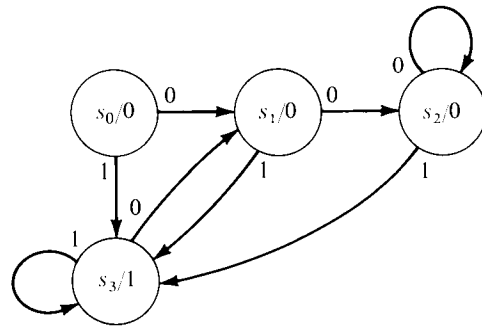
a. 011011010



b. *abccaab*

Present state	Next state			Output
	Present input			
	<i>a</i>	<i>b</i>	<i>c</i>	
s_0	s_2	s_0	s_3	<i>a</i>
s_1	s_0	s_2	s_3	<i>b</i>
s_2	s_2	s_0	s_1	<i>a</i>
s_3	s_1	s_2	s_0	<i>c</i>

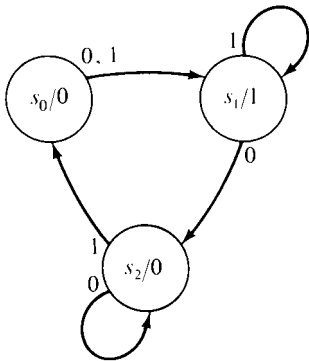
c. 0100110



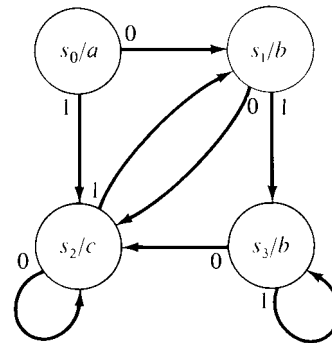
2. a. For the machine described in Exercise 1a, find all input sequences yielding an output sequence of 0011110.
- b. For the machine described in Exercise 1b, find all input sequences yielding an output sequence of *abaaca*.
- c. For the machine described in Exercise 1c, what will be the output for an input sequence $a_1a_2a_3a_4a_5$ where $a_i \in \{0, 1\}$, $1 \leq i \leq 5$?

In Exercises 3–6, write the state table for the machine, and compute the output sequence for the given input sequence.

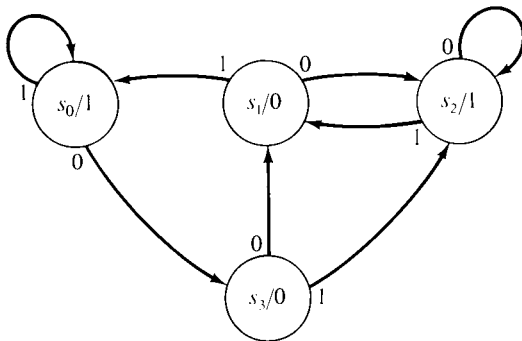
3. 00110



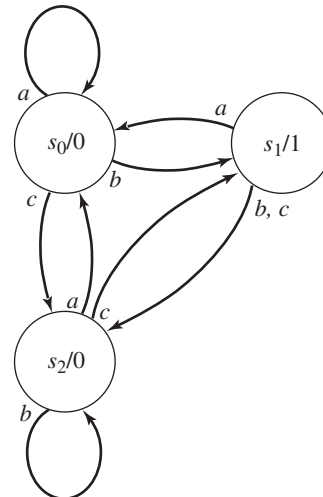
5. 01011



4. 1101100



6. *acbabc*



In Exercises 7–10, draw the state graph for the machine, and compute the output sequence for the given input sequence.

7. 10001

Present state	Next state		Output
	Present input		
	0	1	
s_0	s_0	s_2	1
s_1	s_1	s_0	0
s_2	s_0	s_1	0

9. *acbbca*

Present state	Next state			Output
	Present input			
	<i>a</i>	<i>b</i>	<i>c</i>	
s_0	s_1	s_1	s_1	0
s_1	s_2	s_2	s_1	0
s_2	s_0	s_2	s_1	1

8. 0011

Present state	Next state		Output
	Present input		
	0	1	
s_0	s_2	s_3	0
s_1	s_0	s_1	1
s_2	s_1	s_3	0
s_3	s_1	s_2	1

10. 21021

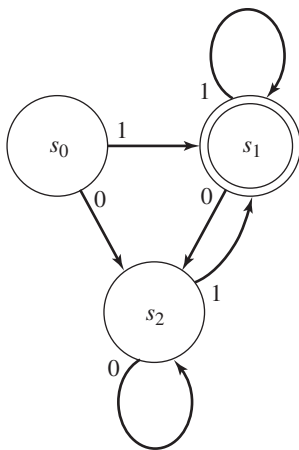
Present state	Next state			Output
	Present input			
	0	1	2	
s_0	s_3	s_1	s_2	1
s_1	s_3	s_0	s_1	2
s_2	s_2	s_1	s_1	0
s_3	s_1	s_4	s_0	0
s_4	s_1	s_4	s_2	2

11. a. Construct a finite-state machine that complements each bit of the binary input string (read left to right).
b. Write the output for the input sequence 01011.
12. a. Construct a finite-state machine that will compute $x + 1$ where x is the input given in binary form, least significant digit first (in this case, read the input right to left). You could use the binary adder of Figure 9.5 by writing 1 as $00 \dots 01$ with the correct number of leading 0s, but that's much too complicated.
b. Write the output for $x = 1101$.
13. a. Construct a finite-state machine that will compute the bitwise AND of two binary input strings.
b. Write the output for the input sequence consisting of the strings 11011 and 10010 (read left to right).
14. a. Construct a finite-state machine that will compute the bitwise OR of two binary input strings.
b. Write the output for the input sequence consisting of the strings 11011 and 10010 (read left to right).
15. a. Construct a delay machine having input and output alphabet $\{0, 1\}$ that, for any input sequence $a_1a_2a_3 \dots$ produces an output sequence of $00a_1a_2a_3 \dots$.
b. Explain (intuitively) why a finite-state machine cannot be built that, for any input sequence $a_1a_2a_3 \dots$, produces the output sequence $0a_10a_20a_3 \dots$.
16. a. Construct a finite-state machine that will compute the 2's complement of p where p is a binary number input with the least significant digit first. (See Exercise 27, Section 8.2.) (In this case, read the input right to left.)
b. Use the machine of part (a) to find the 2's complement of 1100 and of 1011.
17. You are designing a Windows-based, event-driven program to handle customers for a small business. You design the user interface with three screens. The opening screen contains an exit button to quit the program and displays a list box of customer names. Double-clicking on one of the entries in the list box brings up a second screen showing complete data for that customer. This screen contains a button to get back to the opening screen. The opening screen also contains a button that brings up a form to enter the data for a new customer. Draw the state graph for a finite-state machine that describes the high-level user interaction with the program.

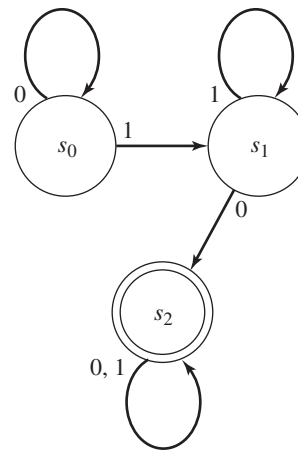
18. Whenever a video disk is inserted into a DVR, the machine automatically turns on and plays the disk. At the end of the recorded part of the disk, the machine turns off. To program the DVR, you must manually turn it on and then select the menu function; when you are finished, you turn the machine off, but its timer is set. At the appropriate time, the machine records, then at the appropriate time it turns itself completely off. Draw the state graph for a finite-state machine that describes the behavior of the DVR.
19. You have an account at First National Usury Trust (FNUT) and a card to operate their ATM (automated teller machine). Once you have inserted your card, the ATM will allow you to process a transaction only if you enter your correct code number, which is 417. Draw the state graph for a finite-state machine designed to recognize this code number. The output alphabet should have three symbols: “bingo” (correct code), “wait” (correct code so far), and “dead” (incorrect code). The input alphabet is $\{0, 1, 2, \dots, 9\}$. To simplify notation, you may designate an arc by $I-\{3\}$, for example, meaning that the machine will take this path for an input symbol that is any digit except 3. (At FNUT, you get only one chance to enter the code correctly.)
20. An elevator in a three-story building services floors 1, 2, and 3. Input consists of a signal to the elevator of an up-or-down request (U or D) together with the floor from which the signal originates. The elevator responds to an input signal by moving to the correct floor. For example, when the elevator is on floor 1 and receives a $D-3$ signal, it moves to floor 3. Draw a state graph for a finite-state machine that describes the elevator behavior.

For Exercises 21–24, determine whether the given machine recognizes the given input string.

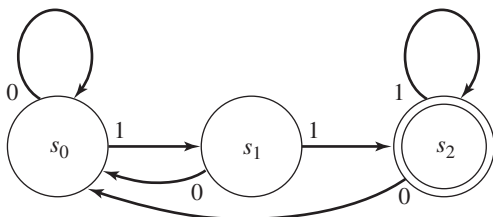
21. 11010



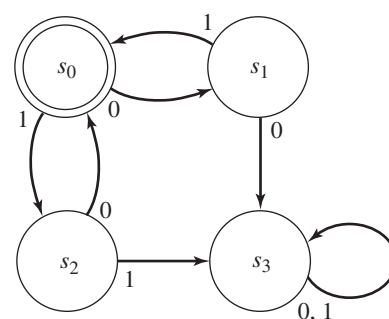
23. 0101



22. 01110111



24. 01101



For Exercises 25–28, construct finite-state machines that act as recognizers for the input described by producing an output of 1 exactly when the input received to that point matches the description. The input and output alphabet in each case is $\{0, 1\}$.

25. a. set of all strings containing an even number of 0s
 b. set of all strings consisting of two or more 1s followed by a 0
 c. set of all strings containing two consecutive 0s and the rest 1s
26. a. set of all strings ending with one or more 0s
 b. set of all strings where the number of 0s is a multiple of 3
 c. set of all strings containing at least four 1s
27. a. set of all strings containing exactly one 1
 b. set of all strings beginning with 000
 c. set of all strings where the second input is 0 and the fourth input is 1
28. a. set of all strings consisting entirely of any number (including none) of 01 pairs or consisting entirely of two 1s followed by any number (including none) of 0s
 b. set of all strings ending in 110
 c. set of all strings containing 00
29. A paragraph of English text is to be scanned and the number of words beginning with “con” counted. Design a finite-state machine that will output a 1 each time such a word is encountered. The output alphabet is $\{0, 1\}$. The input alphabet is the 26 letters of the English alphabet, a finite number of punctuation symbols (period, comma, and so on), and a special character β for blank. To simplify your description, you may use $I - \{m\}$, for example, to denote any input symbol not equal to m .
30. a. In many computer languages, any decimal number N must be presented in one of the following forms:

$$sd^* \quad sd^*.d^* \quad d^* \quad d^*.d^* \quad (1)$$

where s denotes the sign ($s \in \{+, -\}$), d is a digit ($d \in \{0, 1, 2, \dots, 9\}$), and d^* denotes a string of digits where the string may be of any length, including length zero (the empty string). Thus, the following would be examples of valid decimal numbers:

$$+2.74 \quad -.58 \quad 129 \quad +$$

Design a finite-state machine that recognizes valid decimal numbers by producing an output of 1. The input symbols are $+$, $-$, $.$, and the 10 digits. To simplify notation, you may use d to denote any digit input symbol.

- b. Modify the machine of part (a) to recognize any sequence of decimal numbers as defined in part (a) separated by commas. For example, such a machine would recognize

$$+2.74, -.58, 129, +$$

The input alphabet should be the same as for the machine of part (a) with the addition of the symbol c for comma.

- c. Suppose a decimal number must be presented in a form similar to that for part (a) except that any decimal point that appears must have at least one digit before it and after it. Write an expression similar to expression (1) in part (a) to describe the valid form for a decimal number. How would you modify the machine of part (a) to recognize such a number?
31. Let M be a finite-state machine with n states. The input alphabet is $\{0\}$. Show that for any input sequence that is long enough, the output of M must eventually be periodic. What is the maximum number of inputs before periodic output begins? What is the maximum length of a period?

32. At the beginning of the chapter, we learn:

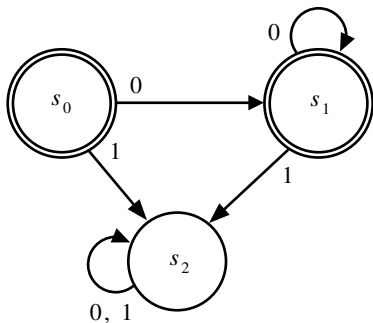
Your team at Babel, Inc., is writing a compiler for a new programming language, currently code-named ScrubOak after a tree outside your office window. During the first phase of compilation (called the lexical analysis phase) the compiler must break down statements into individual units called tokens. In particular, the compiler must be able to recognize identifiers in the language, which are strings of letters, and also recognize the two keywords in the language, which are **if** and **in**.

How can the compiler recognize the individual tokens in a statement?

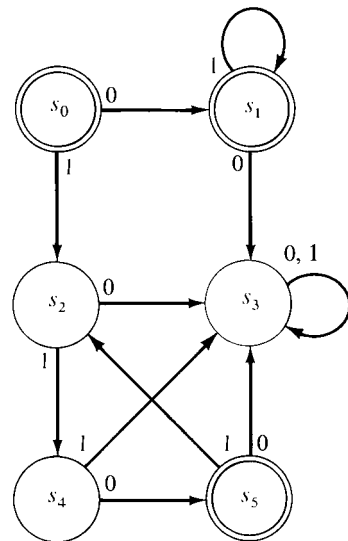
Construct a finite-state machine that operates on a stream of characters and moves into one of two final states representing that a complete keyword has just been processed or that a complete nonkeyword identifier has just been processed. Use β to denote a separating blank between tokens.

For Exercises 33–38, give a regular expression for the set recognized by the finite-state machine.

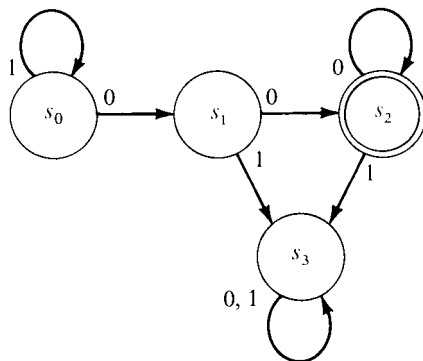
33.



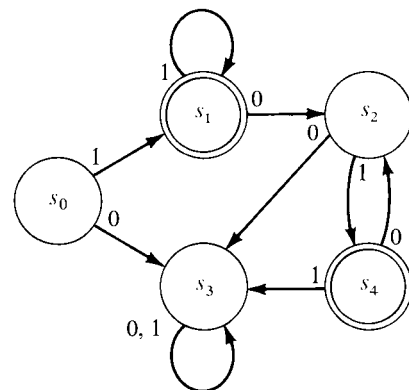
35.

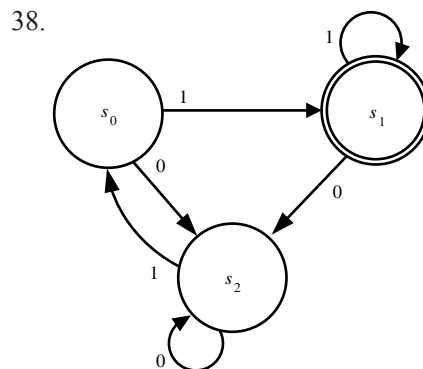
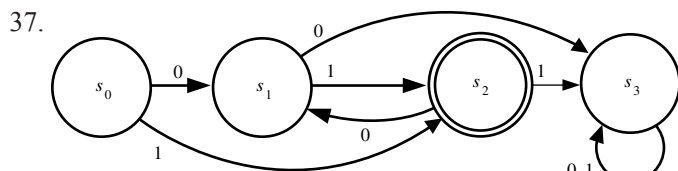


34.



36.





For Exercises 39–42, give a regular expression for the set recognized by the finite-state machine.

39.

Present state	Next state		Output
	Present input		
	0	1	
s_0	s_3	s_1	0
s_1	s_1	s_2	0
s_2	s_3	s_3	1
s_3	s_3	s_3	0

41.

Present state	Next state		Output
	Present input		
	0	1	
s_0	s_2	s_1	1
s_1	s_3	s_1	1
s_2	s_3	s_4	0
s_3	s_3	s_3	0
s_4	s_5	s_3	0
s_5	s_2	s_3	1

40.

Present state	Next state		Output
	Present input		
	0	1	
s_0	s_3	s_1	1
s_1	s_1	s_2	1
s_2	s_2	s_2	0
s_3	s_0	s_2	0

42.

Present state	Next state		Output
	Present input		
	0	1	
s_0	s_4	s_1	1
s_1	s_4	s_2	0
s_2	s_4	s_3	0
s_3	s_3	s_1	1
s_4	s_4	s_4	0

43. Give a regular expression for each of the following sets.

- set of all strings of 0s and 1s beginning with 0 and ending with 1
- set of all strings of 0s and 1s having an odd number of 0s
- $\{101, 1001, 10001, 100001, \dots\}$

44. Give a regular expression for each of the following sets.

- set of all strings of 0s and 1s containing at least one 0
- set of all strings of a 's and b 's where each a is followed by two b 's
- set of all strings of 0s and 1s containing exactly two 0s

45. Does the given string belong to the given regular set?
- 01110111; $(1^*01)^*(11 \vee 0^*)$
 - 11100111; $[(1^*0)^* \vee 0^*11]^*$
 - 011100101; $01^*10^*(11^*0)^*$
46. Does the given string belong to the given regular set?
- 1000011; $(10^* \vee 11)^*(0^*1)^*$
 - 011110; $0^*11(1^* \vee 10)$
 - 101110; $[(101)^*10^*]^*$
47. Write a regular expression for the set of all arithmetic expressions indicating the addition or subtraction of two positive integers.
48. Write a regular expression for the set of all alphanumeric strings beginning with a letter, which is the set of legal identifiers in some programming languages.
49. Write regular expressions for each of the strings described in Exercise 25.
50. Write regular expressions for each of the strings described in Exercise 26.
51. Write regular expressions for each of the strings described in Exercise 27.
52. Write regular expressions for each of the strings described in Exercise 28.
53. a. Prove that if A is a regular set, then the set A^R consisting of the reverse of all strings in A is also regular.
 b. For any string α , let α^R be the reverse string. Do you think the set $\{\alpha\alpha^R \mid \alpha \in I^*\}$ is regular?
54. Prove that if A is a regular set whose symbols come from the alphabet I , then $I^* - A$ is a regular set.

A number of programming languages define “regular expressions” somewhat differently than we have done in this chapter. In these instances, the regular expression is meant to describe a pattern for a set of strings so that an arbitrary string can be matched against the pattern to see whether the string belongs in the set. Examples are (a) searching a string to see whether it matches the format of a valid e-mail address and (b) extracting all instances of the form href = “...” from an HTML document. Perl (Practical Extraction and Report Language) is a language strong in text processing; following are some of the syntax rules for regular expressions in Perl.

- * repeat the preceding character or group 0 or more times [this is familiar]
- ? repeat the preceding character or group 0 or 1 times
- + repeat the preceding character or group 1 or more times
- .
- (period)—matches any single character
- .* (period asterisk)—matches arbitrary string of any length
- \s (backslash lowercase s)—matches any whitespace character (space, tab, newline)
- \S (backslash uppercase s)—matches any nonwhitespace character

For Exercises 55–62, decide which of the given strings match the given regular expression.

- | | |
|---|---------------------------------------|
| 55. Regular expression: <code>bet?er</code> | Strings: beer, beter, better, bettter |
| 56. Regular expression: <code>bet*er</code> | Strings: beer, beter, better, bettter |
| 57. Regular expression: <code>bet+er</code> | Strings: beer, beter, better, bettter |
| 58. Regular expression: <code>b.?t</code> | Strings: bit, but, beet, bt |
| 59. Regular expression: <code>b.+t</code> | Strings: bit, but, beet, bt |
| 60. Regular expression: <code>b\St</code> | Strings: bit, but, beet, b t |
| 61. Regular expression: <code>b\st</code> | Strings: bit, but, beet, b t |
| 62. Regular expression: <code>b\s*t</code> | Strings: bit, bt, b t, b t |

63. Identify any unreachable states of M .

Present state	Next state		Output
	Present input		
	0	1	
s_0	s_2	s_0	0
s_1	s_2	s_1	1
s_2	s_2	s_0	1

64. Identify any unreachable states of M .

Present state	Next state			Output
	Present input			
	a	b	c	
s_0	s_1	s_0	s_3	0
s_1	s_1	s_3	s_0	1
s_2	s_3	s_2	s_1	0
s_3	s_1	s_1	s_0	0

For Exercises 65–74, minimize the given machine.

65.

Present state	Next state		Output
	Present input		
	0	1	
0	3	6	1
1	4	2	0
2	4	1	0
3	2	0	1
4	5	0	1
5	3	5	0
6	4	2	1

66.

Present state	Next state		Output
	Present input		
	0	1	
0	5	3	1
1	5	2	0
2	1	3	0
3	2	4	1
4	2	0	1
5	1	4	0

67.

Present state	Next state		Output
	Present input		
	0	1	
0	1	2	0
1	2	3	1
2	3	4	0
3	2	1	1
4	5	4	1
5	6	7	0
6	5	6	1
7	8	1	0
8	7	3	0

68.

Present state	Next state		Output
	Present input		
	0	1	
0	7	1	1
1	0	3	1
2	5	1	0
3	7	6	1
4	5	6	0
5	2	3	0
6	3	0	1
7	4	0	0

69.

Present state	Next state		Output
	Present input		
	0	1	
0	1	3	0
1	2	4	1
2	5	4	0
3	1	2	2
4	2	1	1
5	4	0	2

70.

Present state	Next state		Output
	Present input		
	0	1	
0	1	3	0
1	2	0	0
2	0	3	0
3	2	1	0

71. Present state

	Next state			Output
	Present input			
	a	b	c	
0	1	4	0	1
1	4	2	3	0
2	3	4	2	1
3	4	0	1	0
4	1	0	2	0

72. Present state

	Next state		Output
	Present input		
	0	1	
0	1	3	1
1	2	0	0
2	4	3	1
3	0	1	1
4	2	4	0

73. Present state

	Next state		Output
	Present input		
	0	1	
0	3	0	0
1	4	3	1
2	1	4	0
3	0	4	1
4	5	2	0
5	2	3	1

74. Present state

	Next state		Output
	Present input		
	0	1	
0	3	5	1
1	1	6	1
2	0	4	0
3	1	6	1
4	5	3	0
5	4	1	0
6	2	5	1

75. Construct a sequential network for the finite-state machine of Exercise 8.

76. Construct a sequential network for the finite-state machine of Exercise 1a. Make use of don't-care conditions to simplify the network.

SECTION 9.4 TURING MACHINES

In Section 9.3, we noted that because $S = \{0^n 1^n \mid n \geq 0\}$ is not a regular set, Kleene's theorem tells us that it is not recognizable by any finite-state machine. We didn't actually prove that S is not a regular set, however; we only noted that we were not able to come up with a regular expression for it. Let's take a slightly different approach.

Suppose S is recognized by a finite-state machine M with m states. Then all strings from S and only strings from S lead M from its start state to a final state. Now let us run M a number of times on successive input strings of $\lambda, 0, 0^2, 0^3, \dots, 0^m$. At the end of processing each of these $m + 1$ strings, M will be in some state. Because M has only m distinct states, there must be two strings from this list, say 0^v and 0^w , $v \neq w$, each of which lead M from the start state to the same state. (This is actually a result of the pigeonhole principle of Chapter 4, where the items are the input strings and the bins into which we put the items are the states M is in after processing the strings.) Because M recognizes S , the input string $0^v 1^v$ will cause M to end in a final state. But because M is in the same state after processing 0^w as after processing 0^v , the string $0^w 1^v$, which does not belong to S , will take M to the same final state. This contradiction proves that no finite-state machine can recognize S .

We probably consider ourselves to be finite-state machines and imagine that our brains, being composed of a large number of cells, can take on only a finite, although immensely large, number of configurations, or states. We feel sure, however, that if someone presented us with an arbitrarily long string of 0s followed by an arbitrarily long string of 1s, we could detect whether the number of 0s and 1s was the same. Let's think of some techniques we might use.

For small strings of 0s and 1s, we could just look at the strings and decide. Thus, we can tell without great effort that $000111 \in S$ and that $00011 \notin S$. However, for the string

00000000000000001111111111111111

we must devise another procedure, probably resorting to counting. We would count the number of 0s received, and when we got to the first 1, we would write the number of 0s down (or remember it) for future reference; then we would begin counting 1s. (This process is what we did mentally for smaller strings.)

However, we have now made use of some extra memory, because when we finished counting 1s, we would have to retrieve the number representing the total number of 0s to make a comparison. But such information retrieval is what the finite-state machine cannot do; its only capacity for remembering input is to have a given input symbol send it to a particular state. We have already seen that no finite-state machine can “remember” 0^n for arbitrarily large n because it runs out of distinct states. In fact, if we try to solve this problem on a real computer, we encounter the same difficulty. If we set a counter as we read in 0s, we might get an overflow because our counter can go only so high. To process $0^n 1^n$ for arbitrarily large n requires an unlimited auxiliary memory for storing the value of our counter, which in practice cannot exist.

Another way we humans might attack the problem of recognizing S is to wait until the entire string has been presented. Then we would go to one end of the string and cross out a 0, go to the other end and cross out a 1, and then continue this back-and-forth operation until we ran out of 0s or 1s. The string belongs to S if and only if we run out of both at the same time. Although this approach sounds rather different from the first one, it still requires remembering each of the inputs, since we must go back and read them once the string is complete. The finite-state machine, of course, cannot reread input.

We have come up with two computational procedures—algorithms—to decide, given a string of 0s and 1s, whether that string belongs to $S = \{0^n 1^n \mid n \geq 0\}$. Both require some form of additional memory unavailable in a finite-state machine. Evidently, the finite-state machine is not a model of the most general form of computational procedure.

Definition

To simulate more general computational procedures than the finite-state machine can handle, we use a *Turing machine*, proposed by the British mathematician Alan M. Turing in 1936. A Turing machine is essentially a finite-state machine with the added ability to reread its input and also to erase and write over its input. It also has unlimited auxiliary memory. Thus, the Turing machine overcomes the deficiencies we noted in finite-state machines. Unlimited auxiliary memory makes the Turing machine a hypothetical “machine”—a model—not a real device.

A Turing machine consists of a finite-state machine and an unlimited tape divided into cells, each cell containing at most one symbol from an allowable finite alphabet. At any one instant, only a finite number of cells on the tape are nonblank. We use the special symbol b to denote a blank cell. The finite-state unit, through its read–write head, reads one cell of the tape at any given moment (Figure 9.15).

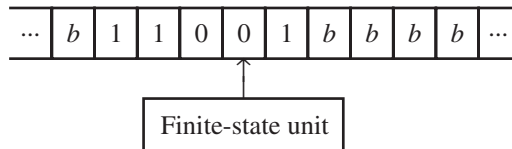
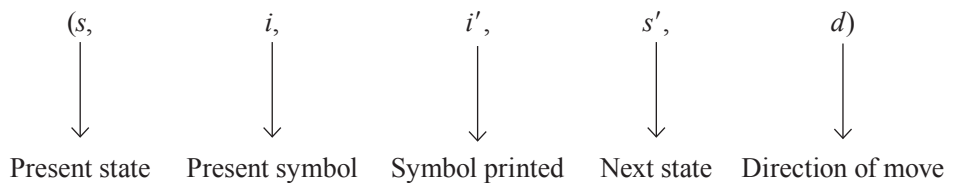


Figure 9.15

By the next clock pulse, depending on the present state of the unit and the symbol read, the unit either does nothing (halts) or completes three actions:

1. Print a symbol from the alphabet on the cell read (it might be the same symbol that’s already there).
2. Go to the next state (it might be the same state as before).
3. Move the read–write head one cell left or right.

We can describe the actions of any particular Turing machine by a set of quintuples of the form (s, i, i', s', d) , where s and i indicate the present state and the tape symbol being read, i' denotes the symbol printed, s' denotes the new state, and d denotes the direction in which the read–write head moves (R for right, L for left).



Thus, a machine in the configuration illustrated by Figure 9.16a, if acting according to the instructions contained in the quintuple $(2, 1, 0, 1, R)$, would move to

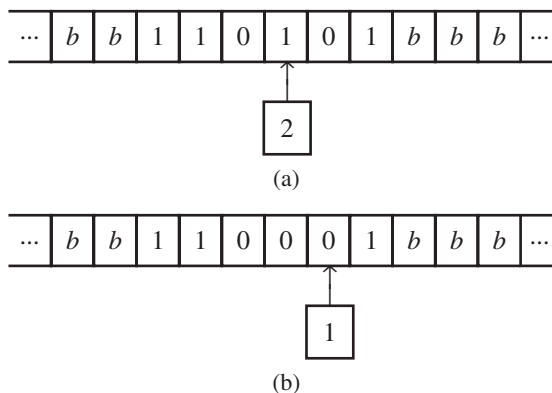


Figure 9.16

the configuration illustrated in Figure 9.16b. The symbol 1 being read on the tape has been changed to a 0, the state of the unit has been changed from 2 to 1, and the head has moved one cell to the right.

The term “Turing machine” is used both in the generic sense and also as the collection of quintuples that describe the actions of any particular machine. This is the same double usage for both the name of the abstraction and any instance of that abstraction that we mentioned for Boolean algebra in Chapter 8.

● **DEFINITION TURING MACHINE**

Let S be a finite set of states and I a finite set of tape symbols (the **tape alphabet**) including a special symbol b . A **Turing machine** is a set of quintuples of the form (s, i, i', s', d) where $s, s' \in S$; $i, i' \in I$; and $d \in \{R, L\}$ and no two quintuples begin with the same s and i symbols.

The restriction that no two quintuples begin with the same s and i symbols ensures that the action of the Turing machine is deterministic and completely specified by its present state and symbol read. If a Turing machine gets into a configuration for which its present state and symbol read are not the first two symbols of any quintuple, the machine halts.

Just as in the case of ordinary finite-state machines, we specify a fixed starting state, denoted by 0, in which the machine begins any computation. We also assume an initial configuration for the read–write head, namely, a position over the farthest left nonblank symbol on the tape. (If the tape is initially all blank, the read–write head can be positioned anywhere to start.)

EXAMPLE 40

A Turing machine is defined by the set of quintuples:

$$(0, 0, 1, 0, R)$$

$$(0, 1, 0, 0, R)$$

$$(0, b, 1, 1, L)$$

$$(1, 0, 0, 1, R)$$

$$(1, 1, 0, 1, R)$$

The action of this Turing machine when processing a particular initial tape is shown by the sequence of configurations in Figure 9.17, which also shows the quintuple that applies at each step. Again, which quintuple applies is determined by the present state and present symbol; as a result, the order in which quintuples are applied has nothing to do with the order in which they are presented in the machine’s definition, quintuples can be used more than once, or may not be used at all.

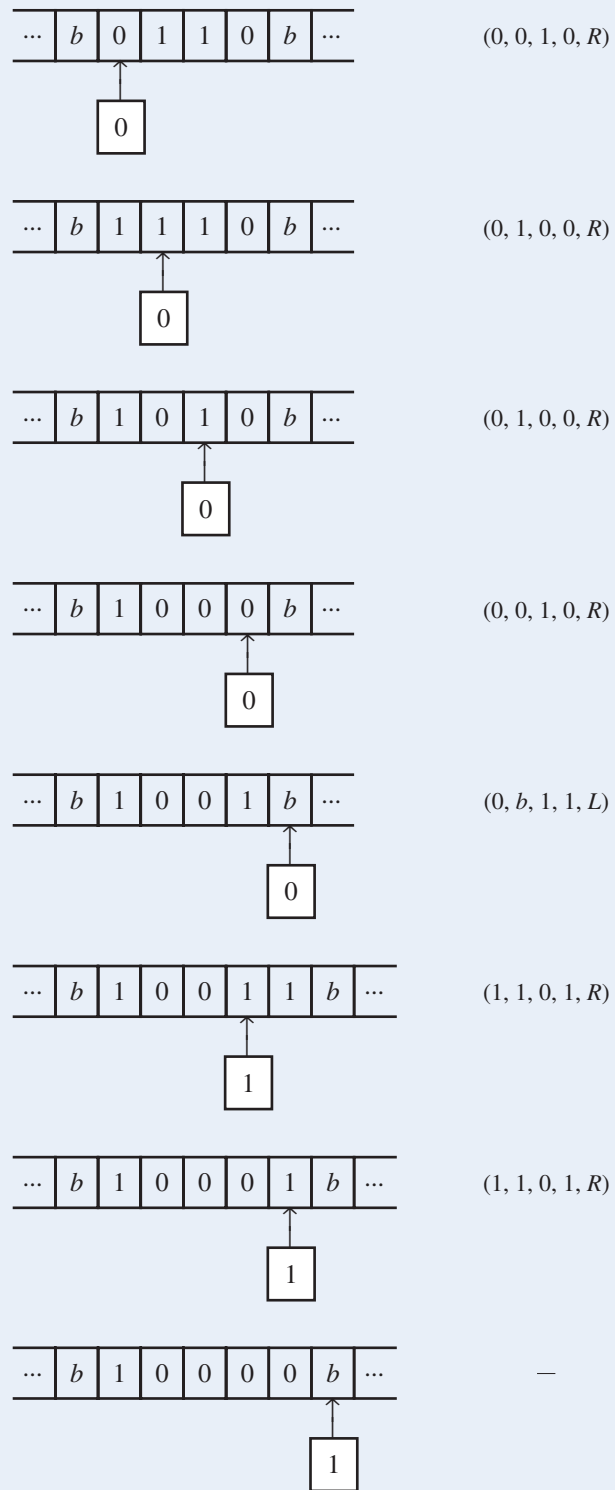
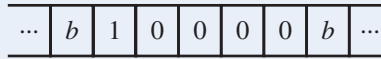


Figure 9.17

Since there are no quintuples defining the action to be taken when in state 1 reading b , the machine halts with final tape:



The tape serves as a memory medium for a Turing machine, and in general, the machine can reread cells of the tape. Since it can also write on the tape, the nonblank portion of the tape can be as long as desired, although there are still only a finite number of nonblank cells at any time. Hence the machine has available an unbounded, though finite, amount of storage. Because Turing machines overcome the limitations of finite-state machines, Turing machines should have considerably higher capabilities. In fact, a finite-state machine is a very special case of a Turing machine, one that always prints the old symbol on the cell read, always moves to the right, and always halts on the symbol b .

PRACTICE 57 Consider the following Turing machine:

$(0, 0, 0, 1, R)$

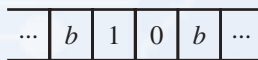
$(0, 1, 0, 0, R)$

$(0, b, b, 0, R)$

$(1, 0, 1, 0, R)$

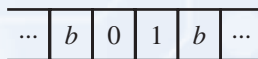
$(1, 1, 1, 0, L)$

- a. What is the final tape, given the initial tape



(Since it is tedious to draw all the little squares, you don't need to do so; just write down the contents of the final tape.)

- b. Describe the behavior of the machine when started on the tape



- c. Describe the behavior of the machine when started on the tape



Parts (b) and (c) of Practice 57 illustrate two ways in which a Turing machine can fail to halt: by endlessly cycling or by moving forever along the tape.

Turing Machines as Set Recognizers

Although the Turing machine computations we have seen so far are not particularly meaningful, we will use the Turing machine to do two kinds of jobs. First, we'll

use it as a recognizer, much as we considered finite-state machines as recognizers in the previous section. We can even give a very similar definition, provided we first define a final state for a Turing machine. A **final state** in a Turing machine is one that is not the first symbol in any quintuple. Thus, on entering a final state, whatever the symbol read, the Turing machine halts.

• **DEFINITION TURING MACHINE RECOGNITION (ACCEPTANCE)**

A Turing machine T with tape alphabet I **recognizes (accepts)** a subset S of I^* if T , beginning in standard initial configuration on a tape containing a string α of tape symbols, halts in a final state if and only $\alpha \in S$.

Note that our definition of acceptance leaves open two possible behaviors for T when applied to a string α of tape symbols not in S . T may halt in a nonfinal state, or T may fail to halt at all.

We can now build a Turing machine to recognize our old friend $S = \{0^n 1^n \mid n \geq 0\}$. The machine is based on our second approach to this recognition problem, sweeping back and forth across the input and crossing out 0–1 pairs.

EXAMPLE 41

We want to build a Turing machine that will recognize $S = \{0^n 1^n \mid n \geq 0\}$. We will use one additional special symbol, call it X , to mark out (“erase”) the 0s and 1s already examined. Thus the tape alphabet is $I = \{0, 1, b, X\}$. State 6 is the only final state. The quintuples making up T are given below. Each quintuple or group of quintuples has a “comment” describing its function. Just like well-written comments in computer code, this makes the Turing instructions much easier to understand. Be sure to include comments in any Turing machine definition.

$(0, b, b, 6, R)$	Recognizes the empty tape, which is in S .
$(0, 0, X, 1, R)$	Erases the leftmost 0 and begins to move right.
$(1, 0, 0, 1, R)$	} Moves right in state 1 until it reaches the end of the binary string; then moves left in state 2.
$(1, 1, 1, 1, R)$	
$(1, b, b, 2, L)$	
$(1, X, X, 2, L)$	
$(2, 1, X, 3, L)$	
$(3, 1, 1, 3, L)$	Moves left over 1s.
$(3, 0, 0, 4, L)$	Goes to state 4 if more 0s are left.
$(3, X, X, 5, R)$	Goes to state 5 if no more 0s in string.
$(4, 0, 0, 4, L)$	Moves left over 0s.
$(4, X, X, 0, R)$	Finds left end of binary string and begins sweep again.
$(5, X, X, 6, R)$	No more 1s in string; machine accepts.

Reading down the columns in Figure 9.18, we can see the key configurations (skipping some routine steps) in the machine’s behavior on the tape

...	b	0	0	0	1	1	1	b	...
-----	-----	---	---	---	---	---	---	-----	-----

which, of course, it should accept.

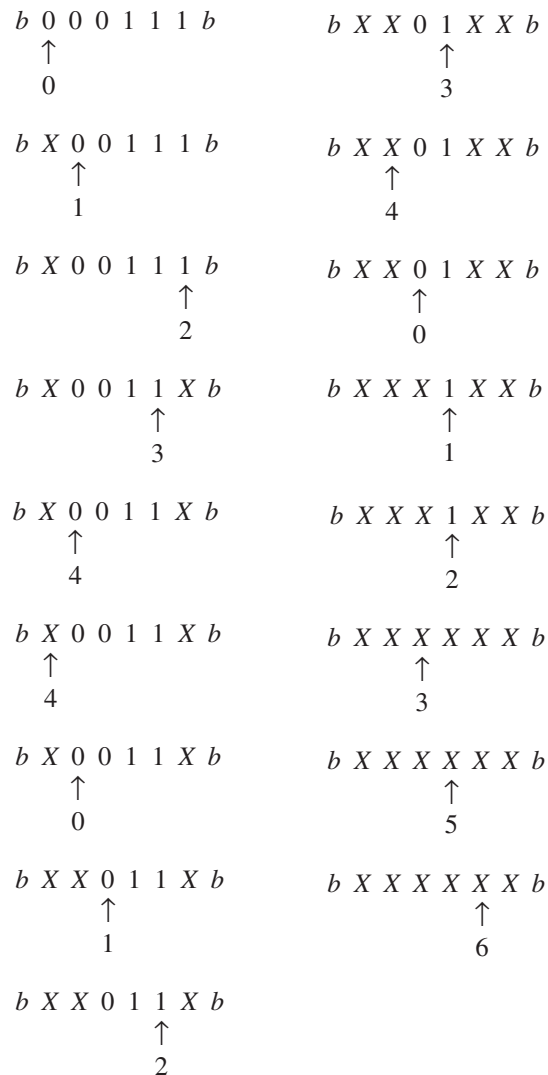


Figure 9.18

PRACTICE 58

For the Turing machine of Example 41, describe the final configuration after processing the following input tapes:

- a.

...	<i>b</i>	0	0	1	1	1	<i>b</i>	...
-----	----------	---	---	---	---	---	----------	-----
- b.

...	<i>b</i>	0	0	0	1	1	<i>b</i>	...
-----	----------	---	---	---	---	---	----------	-----
- c.

...	<i>b</i>	0	0	0	0	1	1	<i>b</i>	...
-----	----------	---	---	---	---	---	---	----------	-----

Notice how each state of the Turing machine in Example 41 is designed to accomplish a certain task, as indicated by the comments. The job of state 1, for

REMINDER

Give the states of your Turing machine big enough jobs to do so that the machine will work in general, not just for special cases. Test using a variety of input tapes.

example, is to move right until the end of the binary string is found, marked by either a blank or an X , then turn the computation over to state 2. A change of state should occur only when something significant happens. For example, a Turing machine cannot pass right over an indeterminate number of 1s by changing state at each move because its behavior would then be tied to a specific input tape. On the other hand, if the machine needs to count over a certain fixed number of 1s, then changing states at each move would accomplish this.

PRACTICE 59

Design a Turing machine to recognize the set of all strings of 0s and 1s ending in 00. (This set can be described by the regular expression $(0 \vee 1)^*00$, so you should be able to use a Turing machine that changes no tape symbols and always moves to the right.) Be sure to include comments.

PRACTICE 60

Modify the Turing machine of Example 41 to recognize $\{0^n 1^{2n} \mid n \geq 0\}$.

Turing Machines as Function Computers

The second job for which we will use the Turing machine is to compute functions. Given a particular Turing machine T and a string α of tape symbols, we begin T in standard initial configuration on a tape containing α . If T eventually halts with a string β on the tape, we may consider β as the value of a function evaluated at α . Using function notation, $T(\alpha) = \beta$. The domain of the function T consists of all strings α for which T eventually halts. We can also think of T as computing **number-theoretic functions**, functions from a subset of \mathbb{N}^k into \mathbb{N} for any $k \geq 1$. We will think of a string of 1s of length $n + 1$ as the unary representation of the nonnegative integer n ; we'll denote this encoding of n by \bar{n} . (The extra 1 in the encoding enables us to distinguish 0 from a blank tape.) Then a tape containing the string $\bar{n}_1 * \bar{n}_2 * \dots * \bar{n}_k$ can be thought of as the representation of the k -tuple (n_1, n_2, \dots, n_k) of nonnegative integers. If T begun in the standard initial configuration on such a tape eventually halts with a final tape that is the representation \bar{m} of a nonnegative integer m , then T has acted as a k -variable function T^k , where $T^k(n_1, n_2, \dots, n_k) = m$. If T begun in standard initial configuration on such a tape either fails to halt or halts with the final tape not a representation of a nonnegative integer, then the function T^k is undefined at (n_1, n_2, \dots, n_k) . There is no need to identify final states when using the Turing machine as a function computer.

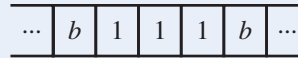
There is thus an infinite sequence $T^1, T^2, \dots, T^k, \dots$ of number-theoretic functions computed by T associated with each Turing machine T . For each k , the function T^k is a **partial function** on \mathbb{N}^k , meaning that its domain may be a proper subset of \mathbb{N}^k . A special case of a partial function on \mathbb{N}^k is a **total function** on \mathbb{N}^k , where the function is defined for all k -tuples of nonnegative integers.

EXAMPLE 42

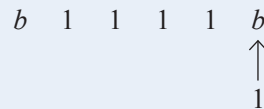
Let a Turing machine T be given by the quintuples

$$\begin{aligned} &(0, 1, 1, 0, R) \\ &(0, b, 1, 1, R) \end{aligned}$$

If T is begun in standard initial configuration on the tape



then T will halt with final configuration



Therefore, T defines a one-variable function T^1 that maps $\bar{2}$ to $\bar{3}$. In general, T maps \bar{n} to $n + 1$, so $T^1(n) = n + 1$, a total function of one variable. ●

In Example 42, we began with a Turing machine and observed a particular function it computed, but we can also begin with a number-theoretic function and try to find a Turing machine to compute it.

● **DEFINITION** **TURING-COMPUTABLE FUNCTION**

A **Turing-computable function** is a number-theoretic function computed by some Turing machine.

A Turing-computable function f can in fact be computed by an infinite number of Turing machines. Once a machine T is found to compute f , we can always include extraneous quintuples in T , producing other machines that also compute f .

EXAMPLE 43

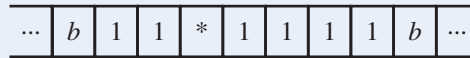
We want to find a Turing machine that computes the function f defined as follows:

$$f(n_1, n_2) = \begin{cases} n_2 - 1 & \text{if } n_2 \neq 0 \\ \text{undefined} & \text{if } n_2 = 0 \end{cases}$$

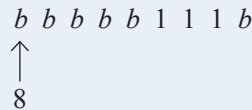
Thus f is a partial function of two variables. Let's consider the Turing machine given by the following quintuples.

(0, 1, 1, 0, R)	}	Passes right over \bar{n}_1 to \bar{n}_2 .
(0, *, *, 1, R)		
(1, 1, 1, 2, R)		
(2, b , b , 3, R)	}	Counts first 1 in \bar{n}_2 . $n_2 = 0$; halts.
(2, 1, 1, 4, R)		
(4, 1, 1, 4, R)	}	Finds the right end of \bar{n}_2 .
(4, b , b , 5, L)		
(5, 1, b , 6, L)		
(6, 1, 1, 6, L)	}	Erases last 1 in \bar{n}_2 .
(6, *, b , 7, L)		
(7, 1, b , 7, L)		
(7, b , b , 8, L)	}	Passes left to \bar{n}_1 , erasing *. Erases \bar{n}_1 . \bar{n}_1 erased; halts with $\overline{n_2 - 1}$ on tape.

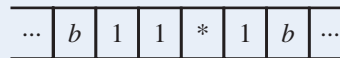
If T is begun on the tape



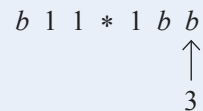
then T will halt with final configuration



This configuration agrees with the requirement that $f(1, 3) = 2$. If T is begun on the tape



then T will halt with final configuration



Because the final tape is not \bar{m} for any nonnegative integer m , the function computed by T is undefined at $(1, 0)$ —just as we want. It is easy to see that this Turing machine computes f and that f is therefore a Turing-computable function. ●

PRACTICE 61 Design a Turing machine to compute the function

$$f(n) = \begin{cases} n - 2 & \text{if } n \geq 2 \\ 1 & \text{if } n < 2 \end{cases}$$

Church–Turing Thesis

In this chapter we have talked about models of “computation” or of “computational procedures.” Although we have not defined the term, by a computational procedure we mean an algorithm. We have talked about algorithms often in this book and have given a number of algorithms for various tasks. Recall that our (somewhat intuitive) definition of an algorithm is a set of instructions that can be mechanically executed in a finite amount of time in order to solve some problem. Given input appropriate to the task, the algorithm must eventually stop (halt) and produce the correct answer if an answer exists. (If no answer exists, let us agree that the algorithm can either halt and declare that no answer exists, or it can go on indefinitely searching for an answer.)

Now we ask: Is the Turing machine a better model of a computational procedure than the finite-state machine? We are quite likely to agree that any Turing computable function f is a function whose values can be found by a computational

procedure or algorithm. In fact, if f is computed by the Turing machine T , then the set of quintuples of T is itself the algorithm; as a list of instructions that can be carried out mechanically, it satisfies the various properties in our notion of an algorithm. Therefore, we are probably willing to accept the proposal illustrated by Figure 9.19. The figure shows “computable by algorithm” as a “cloudy,” intuitive idea and “Turing computable” as a mathematically precise, well-defined idea. The arrow asserts that any Turing-computable function is computable by an algorithm.

Given the simplicity of the definition of a Turing machine, it is a little startling to contemplate Figure 9.20, which asserts that any function computable by any means we might consider to be an algorithm is also Turing computable. Combining Figures 9.19 and 9.20, we get the Church–Turing thesis (Figure 9.21), named after Turing and another well-known mathematician, Alonzo Church.

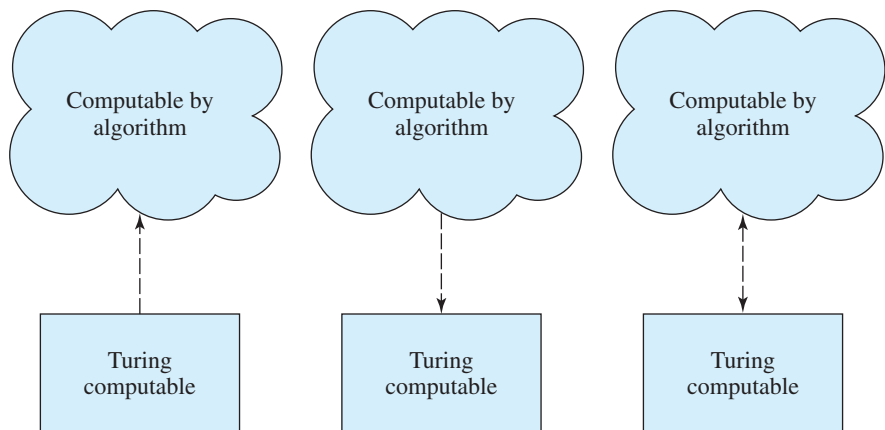


Figure 9.19

Figure 9.20

Figure 9.21

• THESIS CHURCH–TURING THESIS

A number-theoretic function is computable by an algorithm if and only if it is Turing computable.

Because the Church–Turing thesis equates an intuitive idea with a mathematical idea, it can never be formally proved and must remain a thesis, not a theorem. What, then, is its justification?

One piece of evidence is that whenever a procedure generally agreed to be an algorithm has been proposed to compute a function, someone has been able to design a Turing machine to compute that function. (Of course, there is always the nagging thought that someday this might not happen.)

Another piece of evidence is that other mathematicians, several of them at about the same time Turing developed the Turing machine, proposed other models of a computational procedure. On the surface, each proposed model seemed quite unrelated to any of the others. However, because all the models were formally defined, just as Turing computability is, it was possible to determine on a formal, mathematical basis whether any of them were equivalent. All the models, as well as Turing computability, were proved equivalent; that is, they all defined the same class of functions, which suggests that Turing computability embodies everyone’s

concept of an algorithm. Figure 9.22 illustrates what has been done; here solid lines represent mathematical proofs and dashed lines correspond to the Church–Turing thesis. The dates indicate when the various models were proposed.

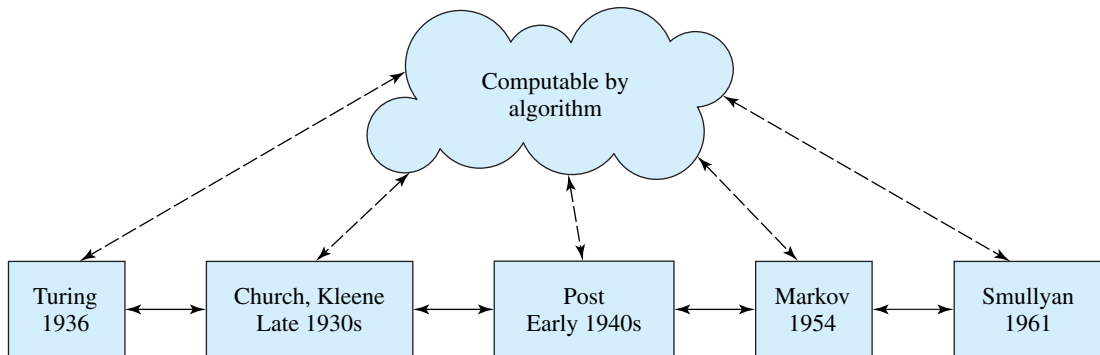


Figure 9.22

The Church–Turing thesis is now widely accepted as a working tool by researchers dealing with computational procedures. If, in a research paper, a method is set forth for computing a function and the method intuitively seems to be an algorithm, then the Church–Turing thesis is invoked and the function is declared to be Turing computable (or one of the names associated with one of the equivalent formulations of Turing computability). This invocation means that the author presumably could, if pressed, produce a Turing machine to compute the function, but again, the Church–Turing thesis is so universally accepted that no one bothers with these details anymore.

Although the Church–Turing thesis is stated in terms of number-theoretic functions, it can be interpreted more broadly. Any algorithm in which a finite set of symbols is manipulated can be translated into a number-theoretic function by a suitable encoding of the symbols as nonnegative integers, much as input to a real computer is encoded and stored in binary form. Thus, by the Church–Turing thesis we can say that if there is an algorithm to do a symbol manipulation task, there is a Turing machine to do it.

By accepting the Church–Turing thesis, we have accepted the Turing machine as the ultimate model of a computational procedure. Turing machine capabilities exceed those of any actual computer, which, after all, does not have the unlimited tape storage of a Turing machine. It is remarkable that Turing proposed this concept in 1936, well before the advent of the modern computer.

Decision Problems and Uncomputability

We have spent quite a bit of time discussing what Turing machines can do. By the Church–Turing thesis, they can do a great deal indeed, although not very efficiently. It is even more important, however, to consider what Turing machines *cannot* do. Because a Turing machine’s abilities to perform tasks exceed those of an actual computer, if we find something no Turing machine can do, then a real computer cannot do it either. In fact, by invoking the Church–Turing thesis, no algorithm exists to do it, and the task is not computable. The type of task we have in mind here is generally that of determining the truth value of each of a number of related statements.

● **DEFINITION** **DECISION PROBLEM**

A **decision problem** asks if an algorithm exists to decide whether individual statements from some large class of statements are true.

The solution to a decision problem answers the question of whether an algorithm exists. A **positive solution** consists of proving that an algorithm exists, and it is generally given by actually producing an algorithm that works. A **negative solution** consists of proving that no algorithm exists. Note that this statement is much stronger than simply saying that a lot of people have tried but no one has come up with an algorithm—this might simply mean that the algorithm is hard. It must be shown that it is impossible for anyone ever to come up with an algorithm. When a negative solution to a decision problem is found, the problem is said to be **unsolvable**, **uncomputable**, or **undecidable**. This terminology can be confusing because the decision problem itself—the question of whether an algorithm exists to do a task—has been solved; what must forever be unsolvable is the task itself.

Examples of Decision Problems

We will look at some decision problems that have been answered.

EXAMPLE 44

Does an algorithm exist to decide, given integers a , b , and c , whether $a^2 = b^2 + c^2$? Clearly, this question is a solvable decision problem. The algorithm consists of multiplying b by itself, multiplying c by itself, adding the two results, and comparing the sum with the result of multiplying a by itself. ●

Obviously, Example 44 is a rather trivial decision problem. Historically, much of mathematics has concerned itself at least indirectly with finding positive solutions to decision problems, that is, producing algorithms. Negative solutions to decision problems arose only in the twentieth century.

EXAMPLE 45

One of the earliest decision problems to be formulated was Hilbert's tenth problem, tenth in a list of problems David Hilbert posed to the International Congress of Mathematicians in 1900. The problem is: Does an algorithm exist to decide for any polynomial equation $P(x_1, x_2, \dots, x_n) = 0$ with integral coefficients whether it has integral solutions? For polynomial equations of the form $ax + by + c = 0$, where a , b , and c are integers, it is known that integer solutions exist if and only if the greatest common divisor of a and b also divides c . Thus, for particular subclasses of polynomial equations, there might be algorithms to decide whether integer solutions exist, but the decision problem as stated applies to the whole class of polynomial equations with integer coefficients. When this problem was posed and for some time after, the general belief was that surely an algorithm existed and the fact that no one had found such an algorithm merely implied that it must be difficult. In the mid-1930s, a startling result by Kurt Gödel, described in the next example, began to cast doubt on this view. It was not until 1970, however, that this problem was finally shown to be unsolvable. ●

EXAMPLE 46

The decision problem for propositional wffs asks whether an algorithm exists to decide whether any given propositional wff is a tautology. This is a solvable decision problem; the solution algorithm consists of constructing and examining the truth table for the wff. The decision problem for predicate wffs asks whether an algorithm exists to decide the validity of any given predicate wff. This is an undecidable problem; such an algorithm does not exist, which is exactly why we resorted to the formal derivation rules of predicate logic to help establish validity for any given wff. Because of the completeness and correctness of predicate logic, a wff is valid if and only if we can produce a proof sequence for it. However, this has only shifted one decision problem to an equivalent one—there is no algorithm to decide whether a proof sequence exists for any given predicate wff, much less a mechanical way to know what steps to use to produce a proof sequence if one does exist.

An alternative formulation to the logic systems we discussed in Chapter 1 is to identify certain strings of symbols as *axioms* and to give *rules of inference* whereby a new string can be obtained from old strings. Any string that is the last one in a finite list of strings consisting of either axioms or strings obtainable by the rules of inference from earlier strings in the list is said to be a *theorem*. The decision problem for such a formal theory is: Does an algorithm exist to decide whether a given string in the formal theory is a theorem of the theory?

The work of Church and the famous twentieth-century logician Kurt Gödel showed that any formal theory that axiomatizes properties of arithmetic (making commutativity of addition an axiom, for example) and is not completely trivial (not everything is a theorem) is undecidable. Their work can be considered good news for working mathematicians because it means that ingenuity in answering questions in number theory will never be replaced by a mechanical procedure. ●

EXAMPLE 47

A particular Turing machine T begun on a tape containing a string α will either eventually halt or never halt. The **halting problem** for Turing machines is a decision problem: Does an algorithm exist to decide, given a Turing machine T and string α , whether T begun on a tape containing α will eventually halt? Turing proved the unsolvability of the halting problem in the late 1930s. ●

Halting Problem

We will prove the unsolvability of the halting problem after two observations. First, it might occur to us that “run T on α ” would constitute an algorithm to see whether T halts on α . If within 25 steps of T ’s computation T has halted, then we know T halts on α . But if within 25,000 steps T has not halted, what can we conclude? T may still eventually halt. How long should we wait? This so-called algorithm will not always give us the answer to our question.

A second observation is that the halting problem asks for one algorithm to be applied to a large class of statements. The halting problem asks, Does an algorithm exist to decide, for any given (T, α) pair, whether T halts when begun on a tape containing α ? The algorithm comes first, and that single algorithm has to give the correct answer for all (T, α) pairs. In the notation of predicate logic, the halting problem asks about the truth value of a statement in the form

$$(\exists \text{ algorithm})(\forall (T, \alpha))(\dots)$$

Consider the following statement, which seems very similar: Given a particular (T, α) pair, does an algorithm exist to decide whether T halts when begun on a tape containing α ? Here, the (T, α) pair comes first and an algorithm is chosen based on the particular (T, α) ; for a different (T, α) , there can be a different algorithm. The statement has become $(\forall(T, \alpha))(\exists \text{ algorithm})(\dots)$. This problem is solvable. Suppose someone gives us a (T, α) . Two algorithms are (1) “say yes” and (2) “say no.” Since T acting on α either does or does not halt, one of these two algorithms correctly answers the question. This solution may seem trivial or even sneaky, but consider again the problem statement: Given a particular (T, α) pair, *does an algorithm exist* to decide, and so forth. Such an algorithm *does exist*; it is either to say yes or to say no—we are not required to choose which one is correct!

This turnabout of words changes the unsolvable halting problem into a trivially solvable problem. It also points out the character of a decision problem, asking whether a single algorithm exists to solve a large class of problems. An unsolvable problem has both a good side and a bad side. That no algorithm exists to solve a large class of problems guarantees jobs for creative thinkers who cannot be replaced by Turing machines. But that the class of problems considered is so large might make the result too general to be of interest.

We will state the halting problem again and then prove its unsolvability.

DEFINITION HALTING PROBLEM

The halting problem asks: Does an algorithm exist to decide, given any Turing machine T and string α , whether T begun on a tape containing α will eventually halt?

THEOREM ON THE HALTING PROBLEM

The halting problem is unsolvable.

Proof: We want to prove that something *does not exist*, a situation made to order for proof by contradiction. Therefore we assume that the halting problem is solvable and that a single algorithm exists that can act on any (T, α) pair as input and eventually decide whether T running on α halts. We are asking this algorithm to solve a task of symbol manipulation, since we can imagine the set of quintuples of T encoded as some unique string s_T of symbols; we’ll use (s_T, α) to denote the string s_T concatenated with the string α . The task then becomes transforming the string (s_T, α) into a string representing a yes (the Turing machine with description s_T halts when begun on a tape containing α) or a no (the Turing machine with description s_T does not halt when begun on a tape containing α). By the Church–Turing thesis, because we have assumed the existence of an algorithm that performs this task, we can assume the existence of a single Turing machine X that performs this task. Thus X acts on a tape containing (s_T, α) for any T and α and eventually halts, at the same time telling us whether T on α halts. To be definite, suppose that X begun on (s_T, α) halts with a 1 left on the tape if and only if T begun on α halts, and X begun on (s_T, α) halts with a 0 left on the tape if and only if T begun on α fails to halt; these are the only two possibilities. Figure 9.23 illustrates Turing machine X .

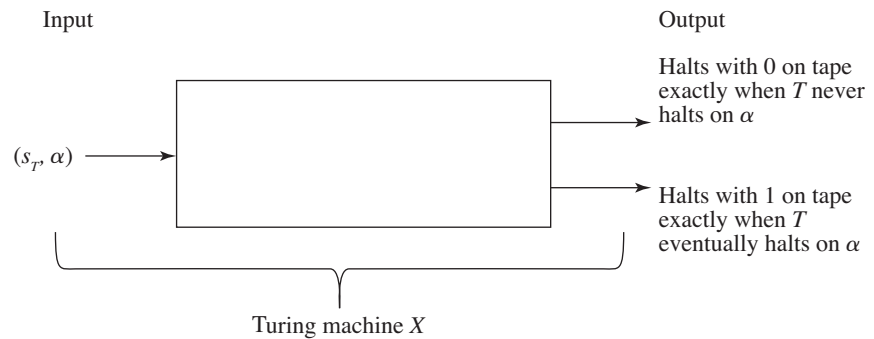


Figure 9.23

Now we add to X 's quintuples to create a new machine Y . Machine Y modifies X 's behavior so that whenever X halts with a 1 on its tape, Y goes to a state that moves Y endlessly to the right so that it never halts. If X halts with a 0 on its tape, so does Y . Figure 9.24 illustrates the behavior of Y .

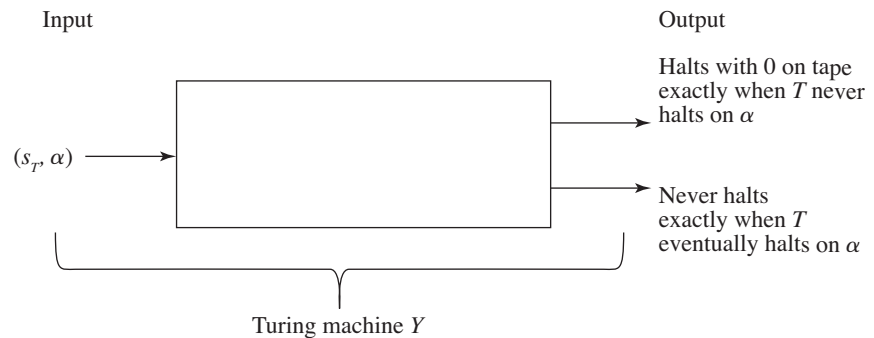


Figure 9.24

Finally, we modify Y to get a new machine Z that acts on any input β by first copying β (see, for example, Exercise 16) and then turning the computation over to Y so that Y acts on (β, β) . What happens if we run Z on its own description, s_Z ? This situation is shown in Figure 9.25.

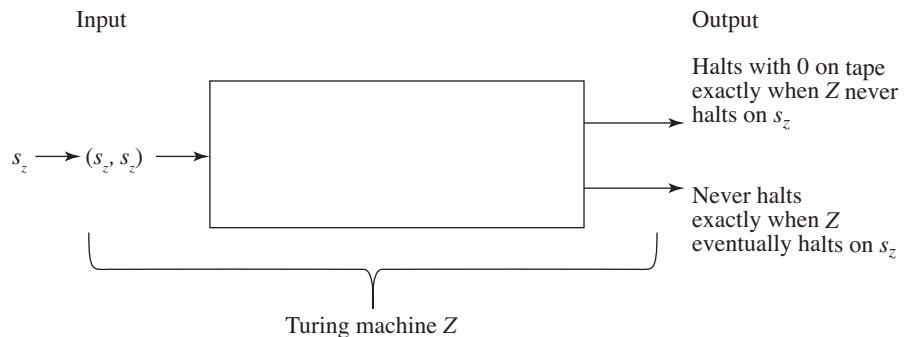


Figure 9.25

By the way Z is constructed, if Z acting on s_Z halts, it is because Y acting on (s_Z, s_Z) halts, and that happens because X acting on (s_Z, s_Z) halts with a 0 on the tape; but if this happens, it implies that Z begun on s_Z fails to halt! Therefore,

$$Z \text{ on } s_Z \text{ halts} \rightarrow Z \text{ on } s_Z \text{ fails to halt} \quad (1)$$

This implication is very strange; let's see what happens if Z on s_Z does not halt. By the way Z is constructed, if Z acting on s_Z does not halt, neither does Y acting on (s_Z, s_Z) . Now Y acting on (s_Z, s_Z) fails to halt exactly when X acting on (s_Z, s_Z) halts with a 1 on the tape; but this result implies that Z begun on s_Z halts! Therefore

$$Z \text{ on } s_Z \text{ fails to halt} \rightarrow Z \text{ on } s_Z \text{ halts} \quad (2)$$

Together, implications (1) and (2) provide an airtight contradiction, so our assumption that the halting problem is solvable is incorrect. *End of Proof.*

The proof of the unsolvability of the halting problem depends on two ideas. One is that of encoding a Turing machine into a string description, and the other is that of having a machine look at and act on its own description. Notice also that neither (1) nor (2) alone in the proof is sufficient to prove the result. Both are needed to contradict the original assumption of the solvability of the halting problem.

We have previously encountered another proof of this nature, where the observation that makes the proof work is self-contradictory. You might want to review here the proof of Cantor's theorem in Chapter 5.

Computational Complexity

As a model of computation, the Turing machine has provided us with a way to prove the existence of unsolvable (uncomputable) problems. Not only does the Turing machine help us find the limits of computability, but it can also help us classify problems that are computable—that have an algorithm for their solution—by the amount of work required to carry out the algorithm.

Finding the amount of work required to carry out an algorithm sounds like analysis of algorithms. We have analyzed a number of real algorithms in this book and classified them as $\Theta(\log n)$, $\Theta(n)$, $\Theta(n^2)$, or what have you. By the Church–Turing thesis, any algorithm can be expressed in Turing machine form. In this form, the amount of work is the number of Turing machine steps (one per clock pulse) required before the Turing machine halts. (We assume here that we are considering only tasks that “have answers” so that the Turing machine halts on all appropriate input.)

Turing machine computations are quite inefficient. Therefore if algorithms A and A' both solve the same problem, but A is expressed as a description of a Turing machine and A' as pseudocode for instructions in a high-level programming language, then comparing the number of operations each algorithm performs is rather meaningless. Therefore we will assume that all algorithms are expressed in Turing machine form so that we can readily compare the efficiency of different algorithms.

Rather than discuss whether a Turing machine algorithm is $\Theta(n)$ or $\Theta(n^2)$, let us simply note whether it is a polynomial-time algorithm. (Only quite trivial algorithms can be better than polynomial time, because it takes a Turing machine n steps just to examine its tape.) Problems for which no polynomial-time algorithms exist are called **intractable**. Such problems may be solvable, but only by inefficient algorithms.

DEFINITION P

P is the collection of all sets recognizable by Turing machines in polynomial time.

Consideration of set recognition in our definition of P is not as restrictive as it may seem. Because the Turing machine halts on all appropriate input, it actually decides, by halting in a final or nonfinal state, whether the initial string was or was not a member of the set. Many problems can be posed as set decision problems by suitably encoding the objects involved in the problem.

For example, consider the Hamiltonian circuit problem (Section 7.2) of whether a graph has a cycle that uses every node of the graph. We may define some encoding process to represent any graph as a string of symbols. Strings that are the representations of graphs become appropriate input, and we want to decide, given such a string, whether it belongs to the set of strings whose associated graphs have Hamiltonian circuits. If we can build a Turing machine to make this decision in polynomial time, then the Hamiltonian circuit problem belongs to P .

We noted in Section 6.2 that the Hamiltonian circuit problem is solvable by the brute-force approach of tracing all possible paths, but this is an exponential solution because of the number of paths. We said that there is no known efficient (polynomial) algorithm to solve the Hamiltonian circuit problem, so we have no proof that the Hamiltonian circuit problem belongs to P . But there is also no proof that the Hamiltonian circuit problem does not belong to P . Might a clever, efficient algorithm someday be found? To see why this is unlikely, we'll consider a new kind of Turing machine.

Ordinary Turing machines act deterministically, due to our restriction that no two quintuples begin with the same present state/present symbol pair. A relaxation of this requirement results in a **nondeterministic Turing machine**, which may have a choice of actions at any step. A nondeterministic Turing machine recognizes a string on its tape if some sequence of actions leads to halting in a final state.

DEFINITION NP

NP is the collection of all sets recognizable by nondeterministic Turing machines in polynomial time. (NP comes from *nondeterministic polynomial time*.)

Although a set in P requires that a deterministic Turing machine be able to make a decision (in polynomial time) about whether some string on its tape does or does not belong to the set, a set in NP requires only that a nondeterministic Turing machine be able to verify (in polynomial time) by a fortuitous choice of actions that an input string is in the set. Given a graph that has a Hamiltonian circuit, for example, this fact can be confirmed in polynomial time by a nondeterministic Turing machine that picks the correct path, so the Hamiltonian circuit problem belongs to NP . Another way to think about a nondeterministic Turing machine is to imagine “parallel processing.” At every clock pulse where there is a choice of action, new versions of the machine are created, one for each possible action. For the Hamiltonian circuit problem, there may be an exponential number of versions created, but the one version tracing the Hamiltonian circuit can complete its path through the graph in polynomial time.

If a Turing machine can decide in polynomial time whether an arbitrary string belongs to a set, it can surely use the same process to verify a member of the set in polynomial time. Therefore $P \subseteq NP$. However, it is not known whether this

inclusion is proper, that is, whether $P \subset NP$ so that there could be NP problems—including perhaps the Hamiltonian circuit problem—that are intractable.

The Hamiltonian circuit problem belongs to a third class of problems known as **NP-complete problems**, meaning that not only are they in NP , but if a polynomial-time decision algorithm were ever found for any one of them, that is, if any of them were ever found to be in P , then indeed we would have $P = NP$. A large number of problems from many different fields have been found to be NP-complete since this idea was formulated in 1971.

EXAMPLE 48

The problem of deciding, for an arbitrary propositional wff, whether it is a tautology is NP-complete. No efficient algorithm has been found for its solution. A brute-force algorithm would explore each of the possible truth assignments to the statement letters. Like the Hamiltonian circuit problem, we see the exponential (nonpolynomial) nature of this inefficient approach—a wff with n statement letters has a truth table with 2^n rows. A related problem called the satisfiability problem asks for a decision as to whether there exists any truth assignment that can “satisfy” the wff—make it true. This is also an NP-complete problem, and in fact was the first problem discovered to be NP-complete. Its brute-force solution algorithm also relies on testing all truth value assignments.

The graph-coloring problem (given an arbitrary graph and a positive integer k , color the nodes of the graph using k colors so that all adjacent nodes are different colors) is NP-complete. Again, no efficient algorithm has been found; the brute-force approach says to assign colors to nodes so that adjacent nodes are different colors, and if you run into a place where this becomes impossible, backtrack and modify your color assignments. Basically, this says to try all possible color assignments (similar to trying all truth assignments, but with k^n possibilities).

The general Sudoku puzzle consists of an $n^2 \times n^2$ grid made up of $n \times n$ blocks where each row, each column, and each block must contain exactly one of the digits 1 through n^2 . (See Exercise 53 in Section 2.4 for an example of the popular $n = 3$ version.) This is an NP-complete problem. No efficient solution algorithm is known; a brute-force solution tries all possible number assignments.

Once again, no polynomial-time decision algorithm has been found for any of these or the many other NP-complete problems, and if an efficient procedure could be found to solve any one of them, such a procedure would exist for all other problems in NP. Therefore it is now suspected that $P \subset NP$ and that all these problems are intractable, but to prove this remains a tantalizing goal in computer science research. ●

SECTION 9.4 REVIEW**TECHNIQUES**

- Describe the action of a given Turing machine on a given initial tape.
- Ⓜ Construct a Turing machine to recognize a given set.
- Ⓜ Construct a Turing machine to compute a given number-theoretic function.

MAIN IDEAS

- Turing machines have a deterministic mode of operation, the ability to reread and rewrite input, and an unbounded auxiliary memory.
- A finite-state machine is a special case of a Turing machine.

- Turing machines can be used as set recognizers and as function computers.
- The Church–Turing thesis equates a function computable by an algorithm with a Turing-computable function. Because this thesis expresses a relationship between an intuitive idea and a formally defined one, it can never be proved but has nonetheless been widely accepted.
- A decision problem asks if an algorithm exists to decide whether individual statements from a large class of statements are true; if no algorithm exists, the decision problem is unsolvable.
- The halting problem is unsolvable.
- $P \subseteq NP$, but it is unknown whether $P \subset NP$.

EXERCISES 9.4

For Exercises 3–26, be sure to include comments with any Turing machine definition.

1. Consider the Turing machine

(0, 0, 0, 0, L)
 (0, 1, 0, 1, R)
 (0, *b*, *b*, 0, L)
 (1, 0, 0, 1, R)
 (1, 1, 0, 1, R)

- a. What is its behavior when started on the tape

...	<i>b</i>	1	0	0	1	1	<i>b</i>	...
-----	----------	---	---	---	---	---	----------	-----

- b. What is its behavior when started on the tape

...	<i>b</i>	0	0	1	1	1	<i>b</i>	...
-----	----------	---	---	---	---	---	----------	-----

2. Consider the Turing machine

(0, 1, 1, 0, R)
 (0, 0, 0, 1, R)
 (1, 1, 1, 1, R)
 (1, *b*, 1, 2, L)
 (2, 1, 1, 2, L)
 (2, 0, 0, 2, L)
 (2, *b*, 1, 0, R)

- a. What is its behavior when started on the tape

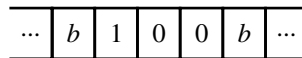
...	<i>b</i>	1	0	1	0	<i>b</i>	...
-----	----------	---	---	---	---	----------	-----

- b. What is its behavior when started on the tape

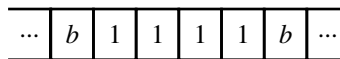
...	<i>b</i>	1	0	1	<i>b</i>	...
-----	----------	---	---	---	----------	-----

3. Find a Turing machine that recognizes the set of all unary strings consisting of an even number of 1s (this includes the empty string).

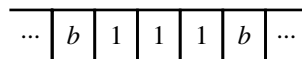
4. Find a Turing machine that recognizes the set of all strings of 0s and 1s containing at least one 1.
5. Find a Turing machine that recognizes 0^*10^*1 .
6. Find a Turing machine to accept the set of nonempty strings of well-balanced parentheses. (Note that $((()(()))$ is well balanced and $((()(($) is not.)
7. Find a Turing machine that recognizes $\{0^{2^n}1^n2^{2^n} \mid n \geq 0\}$.
8. Find a Turing machine that recognizes $\{w * w^R \mid w \in \{0, 1\}^* \text{ and } w^R \text{ is the reverse of the string } w\}$.
9. Find a Turing machine that recognizes $\{w_1 * w_2 \mid w_1, w_2 \in \{0, 1\}^* \text{ and } w_1 \neq w_2\}$.
10. Find a Turing machine that recognizes the set of palindromes on $\{0, 1\}^*$, that is, the set of all strings in $\{0, 1\}^*$ that read the same forward and backward, such as 101.
11. Find a bit-inverter Turing machine that replaces every 0 in a string of 0s and 1s with a 1 and every 1 with a 0.
12. Find a Turing machine that changes a unary string to a string of the same length with alternating 1s and 0s.
13. Find a nonhalting Turing machine that begins with a single 1 on its tape and successively generates strings of the form 0^n10^n , $n \geq 1$, that is, such strings appear every so often on the tape.
14. Find a Turing machine that, given an initial tape containing a (possibly empty) string of 1s, adds a single 0 to the left end of the string if the number of 1s is even and adds two 0s to the left end of the string if the number of 1s is odd.
15. Find a Turing machine that converts a string of 0s and 1s representing a nonzero binary number into a string of that number of 1s. As an example, the machine should, when started on a tape containing



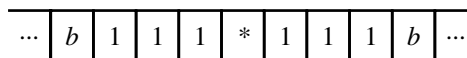
halt on a tape containing



16. Find a Turing machine that, given an initial tape containing a nonempty string of 1s, marks the right end of the string with a * and puts a copy of the string to the right of the *. As an example, the machine should, when started on a tape containing



halt on a tape containing



17. What number-theoretic function of three variables is computed by the following Turing machine?

$(0, 1, b, 0, R)$

$(0, *, b, 1, R)$

$(1, 1, 1, 2, R)$

$(2, *, *, 3, R)$

$(3, 1, 1, 2, L)$

$(2, 1, 1, 4, R)$

$(4, 1, 1, 4, R)$

$(4, *, 1, 5, R)$

$(5, 1, b, 5, R)$

$(5, b, b, 6, R)$

18. What number-theoretic function of one variable is computed by the following Turing machine?

(0, 1, 1, 1, R)
 (1, b, b, 9, R)
 (1, 1, 1, 2, R)
 (2, b, b, 3, L)
 (3, 1, b, 9, L)
 (2, 1, 1, 4, R)
 (4, b, b, 5, L)
 (5, 1, b, 3, L)
 (4, 1, 1, 6, L)
 (6, 1, 1, 6, L)
 (6, b, 1, 7, L)
 (7, b, 1, 8, L)
 (8, b, 1, 9, L)

19. Find a Turing machine to compute the function

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ 2 & \text{if } n \neq 0 \end{cases}$$

20. Find a Turing machine to compute the function

$$f(n) = \begin{cases} n & \text{if } n \text{ is even} \\ n + 1 & \text{if } n \text{ is odd} \end{cases}$$

21. Find a Turing machine to compute the function

$$f(n) = 2n$$

22. Find a Turing machine to compute the function

$$f(n) = \begin{cases} n/3 & \text{if 3 divides } n \\ \text{undefined} & \text{otherwise} \end{cases}$$

23. Find a Turing machine to compute the function

$$f(n_1, n_2) = n_1 + n_2$$

24. Find a Turing machine to compute the function

$$f(n_1, n_2) = \begin{cases} n_1 & \text{if } n_1 \text{ is even} \\ n_2 & \text{if } n_1 \text{ is odd} \end{cases}$$

25. Find a Turing machine to compute the function

$$f(n_1, n_2) = \begin{cases} n_1 - n_2 & \text{if } n_1 \geq n_2 \\ 0 & \text{otherwise} \end{cases}$$

26. Find a Turing machine to compute the function

$$f(n_1, n_2) = \max(n_1, n_2)$$

27. Do Exercise 21 again, this time making use of the machines T_1 and T_2 of Exercises 16 and 23, respectively, as “functions.” (Formally, the states of these machines would have to be renumbered as the quintuples are inserted into the “main program,” but you may omit this tiresome detail and merely “invoke T_1 ” or “invoke T_2 .”)
28. Describe verbally the actions of a Turing machine that computes the function $f(n_1, n_2) = n_1 \cdot n_2$, that is, design the algorithm but do not bother to create all the necessary quintuples. You may make use of Exercises 16 and 23.

SECTION 9.5 FORMAL LANGUAGES

Suppose we come upon the English language sentence, “The walrus talks loudly.” Although we might be surprised at the meaning, or *semantics*, of the sentence, we accept its form, or *syntax*, as valid in the language, meaning that the various parts of speech (noun, verb, and so on) are strung together in a reasonable way. In contrast, we reject, “Loudly walrus the talks,” as an illegal combination of parts of speech, or as syntactically incorrect and not part of the language. We must also worry about correct syntax in programming languages, but in these, unlike natural languages (English, French, and so on), legal combinations of symbols are specified in detail. Let’s give a formal definition of *language*; the definition will be general enough to include both natural languages and programming languages.

DEFINITIONS ALPHABET, VOCABULARY, WORD, LANGUAGE

An **alphabet** or **vocabulary** V is a finite, nonempty set of symbols. A **word** over V is a finite-length string of symbols from V . The set V^* is the set of all words over V . (See Example 6 in Chapter 3 for a recursive definition of V^* .) A **language** over V is any subset of V^* .

Viewing syntactically correct English language as a subset L of the set of all strings over the usual alphabet, we feel that, “The walrus talks loudly,” belongs to L while, “Loudly walrus the talks,” does not.

For any given language L , how can we describe L , that is, specify exactly those words belonging to L ? If L is finite, we can just list its members, but if L is infinite, can we find a finite description of L ? Not always—there are many more languages than possible finite descriptions. Although we will consider only languages that can be finitely described, we can still think of two possibilities. We may be able to describe an algorithm to *decide* membership in L ; that is, given any word in V^* , we could apply our algorithm and receive a yes or no answer as to whether the word belongs to L . Or we may be able to describe a procedure allowing us only to *generate* members of L , that is, crank out one at a time a list of all the members of L . We will settle for languages for which this second option is possible and describe such a language by defining its generative process, or giving a *grammar* for the language.

Before we give a formal definition of what constitutes a grammar, let's look again at why, "The walrus talks loudly," seems to be an acceptable sentence by seeing how it could be generated. Starting from the notion of sentence, we would agree that one legitimate form for a sentence is a noun phrase followed by a verb phrase. Symbolically,

$$\text{sentence} \rightarrow \text{noun-phrase verb-phrase}$$

A legitimate form of noun phrase is an article followed by a noun,

$$\text{noun-phrase} \rightarrow \text{article noun}$$

and a legitimate form of verb phrase is a verb followed by an adverb,

$$\text{verb-phrase} \rightarrow \text{verb adverb}$$

We would also agree with the substitutions

$$\begin{aligned} \text{article} &\rightarrow \text{the} \\ \text{noun} &\rightarrow \text{walrus} \\ \text{verb} &\rightarrow \text{talks} \\ \text{adverb} &\rightarrow \text{loudly} \end{aligned}$$

Thus we can generate the sentence, "The walrus talks loudly," by making successive substitutions:

$$\begin{aligned} \text{sentence} &\Rightarrow \text{noun-phrase verb-phrase} \\ &\Rightarrow \text{article noun verb-phrase} \\ &\Rightarrow \text{the noun verb-phrase} \\ &\Rightarrow \text{the walrus verb-phrase} \\ &\Rightarrow \text{the walrus verb adverb} \\ &\Rightarrow \text{the walrus talks adverb} \\ &\Rightarrow \text{the walrus talks loudly} \end{aligned}$$

The foregoing boldface terms are those for which further substitutions can be made. The nonboldface terms stop or terminate the substitution process. These ideas are incorporated in the next definition.

● **DEFINITION PHRASE-STRUCTURE (TYPE 0) GRAMMAR**

A **phrase-structure grammar (type 0 grammar)** G is a 4-tuple, $G = (V, V_T, S, P)$, where

V = vocabulary

V_T = nonempty subset of V called the set of **terminals**

S = element of $V - V_T$ called the **start symbol**

P = finite set of **productions** of the form $\alpha \rightarrow \beta$ where α is a word over V containing at least one nonterminal symbol and β is a word over V

EXAMPLE 49

Here is a very simple grammar: $G = (V, V_T, S, P)$ where $V = \{0, 1, S\}$, $V_T = \{0, 1\}$, and $P = \{S \rightarrow 0S, S \rightarrow 1\}$.

The productions of a grammar allow us to transform some words over V into others; the productions can be called rewriting rules.

DEFINITION GENERATIONS (DERIVATIONS) IN A LANGUAGE

Let G be a grammar, $G = (V, V_T, S, P)$, and let w_1 and w_2 be words over V . Then w_1 **directly generates (directly derives)** w_2 , written $w_1 \Rightarrow w_2$, if $\alpha \rightarrow \beta$ is a production of G , w_1 contains an instance of α , and w_2 is obtained from w_1 by replacing that instance of α with β . If w_1, w_2, \dots, w_n are words over V and $w_1 \Rightarrow w_2, w_2 \Rightarrow w_3, \dots, w_{n-1} \Rightarrow w_n$, then w_1 **generates (derives)** w_n , written $w_1 \xRightarrow{*} w_n$. (By convention, $w_1 \xRightarrow{*} w_1$.)

EXAMPLE 50

In the grammar of Example 49, $00S \Rightarrow 000S$ because the production $S \rightarrow 0S$ has been used to replace the S in $00S$ with $0S$. Also $00S \xRightarrow{*} 00000S$.

PRACTICE 62

Show that in the grammar of Example 49, $0S \xRightarrow{*} 00001$.

DEFINITION LANGUAGE GENERATED BY A GRAMMAR

Given a grammar G , the **language L generated by G** , sometimes denoted $L(G)$, is the set

$$L = \{w \in V_T^* \mid S \xRightarrow{*} w\}$$

In other words, L is the set of all strings of terminals generated from the start symbol.

Notice that once a string w of terminals has been obtained, no productions can be applied to w , and w cannot generate any other words.

The following procedure generates a list of the members of L : Begin with the start symbol S and systematically apply some sequence of productions until a string w_1 of terminals has been obtained; then $w_1 \in L$. Go back to S and repeat this procedure using a different sequence of productions to generate another word $w_2 \in L$, and so forth. Actually, this procedure doesn't quite work because we might start on an infinite sequence of direct derivations that never leads to a string of terminals and thus never contributes a word to our list. Instead, we need to run a number of derivations from S simultaneously (parallel processing), checking on each one after each step and adding the final word to the list of members of L for any that terminate. That way we cannot get stuck waiting indefinitely while unable to do anything else.

REMINDER

Productions may be used in any order. The only requirement is that the left side of the production must appear in the string you are processing.

PRACTICE 63 Describe the language generated by the grammar G of Example 49. ■

Languages derived from grammars such as we have defined are called **formal languages**. If the grammar is defined first, the language will follow as an outcome of the definition. Alternatively, the language, as a well-defined set of strings, may be given first, and we then seek a grammar that generates it.

EXAMPLE 51

Let L be the set of all nonempty strings consisting of an even number of 1s. Then L is generated by the grammar $G = (V, V_T, S, P)$ where $V = \{1, S\}$, $V_T = \{1\}$, and $P = \{S \rightarrow SS, S \rightarrow 11\}$. A language can be generated by more than one grammar. L is also generated by the grammar $G' = (V', V'_T, S', P')$ where $V' = \{1, S\}$, $V'_T = \{1\}$, and $P' = \{S \rightarrow 1S1, S \rightarrow 11\}$. ■

PRACTICE 64

- Find a grammar that generates the language $L = \{0^n 10^n \mid n \geq 0\}$.
- Find a grammar that generates the language $L = \{0^n 10^n \mid n \geq 1\}$. ■

Trying to describe concisely the language generated by a given grammar and defining a grammar to generate a given language can both be quite difficult tasks. We'll look at another example where the grammar is a bit more complicated than any we've seen so far. Don't worry about how you might think up this grammar; just convince yourself that it works.

EXAMPLE 52

Let $L = \{a^n b^n c^n \mid n \geq 1\}$. A grammar generating L is $G = (V, V_T, S, P)$ where $V = \{a, b, c, S, B, C\}$, $V_T = \{a, b, c\}$, and P consists of the following productions:

- | | | | |
|-------------------------|------------------------|------------------------|------------------------|
| 1. $S \rightarrow aSBC$ | 3. $CB \rightarrow BC$ | 5. $bB \rightarrow bb$ | 7. $cC \rightarrow cc$ |
| 2. $S \rightarrow aBC$ | 4. $aB \rightarrow ab$ | 6. $bC \rightarrow bc$ | |

It is fairly easy to see how to generate any particular member of L using these productions. Thus, a derivation of the string $a^2 b^2 c^2$ is

$$\begin{aligned}
 S &\Rightarrow aSBC \\
 &\Rightarrow aaBCBC \\
 &\Rightarrow aaBBCC \\
 &\Rightarrow aabBCC \\
 &\Rightarrow aabbCC \\
 &\Rightarrow aabbcC \\
 &\Rightarrow aabbcc
 \end{aligned}$$

In general, $L \subseteq L(G)$ where the outline of a derivation for any $a^n b^n c^n$ is given below; the numbers refer to the productions used.

$$\begin{aligned}
S &\stackrel{*}{\Rightarrow}_1 a^{n-1}S(BC)^{n-1} \\
&\Rightarrow_2 a^n(BC)^n \\
&\stackrel{*}{\Rightarrow}_3 a^nB^nC^n \\
&\Rightarrow_4 a^nbB^{n-1}C^n \\
&\stackrel{*}{\Rightarrow}_5 a^nb^nC^n \\
&\Rightarrow_6 a^nb^nC^{n-1} \\
&\stackrel{*}{\Rightarrow}_7 a^nb^nc^n
\end{aligned}$$

We must also show that $L(G) \subseteq L$, which involves arguing that some productions must be used before others and that the general derivation shown above is the only sort that will lead to a string of terminals. ●

REMINDER

“Throwing in” productions to get the language L you want often means that you generate more than L . The productions must generate exactly the strings in L .

In trying to invent a grammar to generate the L of Example 52, we might first try to use productions of the form $B \rightarrow b$ and $C \rightarrow c$ instead of productions 4 through 7. Then we would indeed have $L \subseteq L(G)$, but $L(G)$ would also include words such as $a^n(bc)^n$.

Formal languages were developed in the 1950s by linguist Noam Chomsky in an attempt to model natural languages, such as English, with an eye toward automatic translation. However, since a natural language already exists and is quite complex, defining a formal grammar to generate a natural language is very difficult. Attempts to do this for English have been only partially successful.

EXAMPLE 53

We can describe a formal grammar that will generate a very restricted class of English sentences. The terminals in the grammar are the words “the,” “a,” “river,” “walrus,” “talks,” “flows,” “loudly,” and “swiftly,” and the nonterminals are the words **sentence**, **noun-phrase**, **verb-phrase**, **article**, **noun**, **verb**, and **adverb**. The start symbol is **sentence** and the productions are

sentence \rightarrow **noun-phrase verb-phrase**
noun-phrase \rightarrow **article noun**
verb-phrase \rightarrow **verb adverb**
article \rightarrow the
article \rightarrow a
noun \rightarrow river
noun \rightarrow walrus
verb \rightarrow talks
verb \rightarrow flows
adverb \rightarrow loudly
adverb \rightarrow swiftly

We know how to derive, “The walrus talks loudly,” in this grammar. Here is a derivation of, “A river flows swiftly”:

sentence \Rightarrow **noun-phrase verb-phrase**
 \Rightarrow **article noun verb-phrase**
 \Rightarrow **a noun verb-phrase**
 \Rightarrow **a river verb-phrase**
 \Rightarrow **a river verb adverb**
 \Rightarrow **a river flows adverb**
 \Rightarrow **a river flows swiftly**

A few other sentences making various degrees of sense, such as, “a walrus flows loudly,” are also part of the language defined by this grammar. The difficulty of specifying a grammar for English as a whole becomes more apparent when we consider that a phrase such as, “time flies,” can be an instance of either a noun followed by a verb or of a verb followed by a noun. This phrase is “ambiguous” (see Exercises 35 and 36 in this section). ●

Programming languages are less complex than natural languages, and their syntax often can be described successfully using formal language notation.

EXAMPLE 54

A section of formal grammar to generate identifiers in some programming language could be presented as follows:

identifier \rightarrow **letter**
identifier \rightarrow **identifier letter**
identifier \rightarrow **identifier digit**
letter \rightarrow **a**
letter \rightarrow **b**
 \vdots
letter \rightarrow **z**
digit \rightarrow **0**
digit \rightarrow **1**
 \vdots
digit \rightarrow **9**

Here the set of terminals is $\{a, b, \dots, z, 0, 1, \dots, 9\}$ and **identifier** is the start symbol. ●

EXAMPLE 55

A shorthand that can avoid a long listing of productions is called **Backus-Naur form (BNF)**. The productions of Example 54 can be given in BNF by three lines (as in Example 7 of Chapter 3):

$\langle \text{identifier} \rangle ::= \langle \text{letter} \rangle \mid \langle \text{identifier} \rangle \langle \text{letter} \rangle \mid \langle \text{identifier} \rangle \langle \text{digit} \rangle$
 $\langle \text{letter} \rangle ::= a \mid b \mid c \mid \dots \mid z$
 $\langle \text{digit} \rangle ::= 0 \mid 1 \mid \dots \mid 9$

In BNF, nonterminals are identified by $\langle \rangle$, the production arrow becomes $::=$, and $|$ stands for “or,” identifying various productions having the same left-hand symbol.

In modern times, BNF notation was originally used to define the programming language Algol (ALGOrithmic Language) in the early 1960s. However, it appears that a similar notation was used between 400 BCE and 200 BCE to describe the rules of Sanskrit grammar.²

EXAMPLE 56

The markup language HTML (Hypertext Markup Language) is a specific language for writing documents that will be transmitted across a network and displayed by a Web browser. The browser parses and interprets the various tags that identify legal elements of an HTML document and displays the elements accordingly. The structure of each HTML element, such as a TABLE element, is fixed; someone writing an HTML document using TABLE elements must follow the structure of a TABLE element as defined in HTML. XML (Extensible Markup Language) is a generalized language for defining specific markup languages, that is, it’s a language for defining languages. The elements of an XML document can have whatever structure the user chooses; the user defines these elements in an associated DTD (Document Type Definition). The DTD can be part of the XML document, or it can be in a separate file referenced by the XML document (and as such, available for a number of XML documents to use). The following DTD gives a grammar for an XML document about manufactured parts, and defines the structure of a “PARTSLIST” element, an “ITEM” element, and so forth.

```
<!DOCTYPE PARTSLIST {
  <!ELEMENT PARTSLIST (ITEM+)>
  <!ELEMENT ITEM (PARTNUMBER, USEDINLIST)>
  <!ELEMENT PARTNUMBER (#PCDATA)>
  <!ELEMENT USEDINLIST (USEDIN*)>
  <!ELEMENT USEDIN (MAKE, MODEL)>
  <!ELEMENT MAKE (#PCDATA)>
  <!ELEMENT MODEL (#PCDATA)>
}>
```

The DTD uses notation similar to that for regular expressions, where $*$ means zero or more instances, $+$ means one or more instances, and $?$ means zero instances or one instance. So in the preceding DTD, a PARTSLIST consists of one or more ITEMS, and a USEDINLIST consists of zero or more USEDINs. The notation #PCDATA stands for a “parsed character string”. In BNF notation, the grammar becomes

```
<PARTSLIST> ::= <ITEM>|<ITEM><PARTSLIST>
<ITEM> ::= <PARTNUMBER> <USEDINLIST>
<PARTNUMBER> ::= <identifier>
<USEDINLIST> ::=  $\lambda$  | <USEDIN>|<USEDIN><USEDINLIST>
<USEDIN> ::= <MAKE><MODEL>
<MAKE> ::= <identifier>
<MODEL> ::= <identifier>
<identifier> ::= [as defined in Example 55]
```

²“Panini-Backus Form Suggested,” Ingerman, P. Z. ., *Communications of the ACM*, vol. 10, No. 3, 1967.

A Web browser parses the XML document to determine whether it is a valid instance of the language defined by its associated DTD, and if so, displays it. ●

Classes of Grammars

Before we identify some types of grammars, let's look at one more example.

EXAMPLE 57

Let L be the empty string λ together with the set of all strings consisting of an odd number n of 0s, $n \geq 3$. The grammar $G = (V, V_T, S, P)$ generates L where $V = \{0, A, B, E, F, W, X, Y, Z, S\}$, $V_T = \{0\}$, and the productions are

$$\begin{array}{lll} S \rightarrow FA & 0X \rightarrow X0 & 0Z \rightarrow Z0 \\ S \rightarrow FBA & Y0 \rightarrow 0Y & WBZ \rightarrow EB \\ FB \rightarrow FOEB0 & FX \rightarrow F0W & F \rightarrow \lambda \\ EB \rightarrow 0 & YA \rightarrow Z0A & A \rightarrow \lambda \\ EB \rightarrow XBY & W0 \rightarrow 0W & \end{array}$$

The derivation $S \Rightarrow FA \xRightarrow{*} \lambda\lambda = \lambda$ produces λ . The derivation

$$\begin{array}{l} S \Rightarrow FBA \\ \Rightarrow FOEB0A \\ \Rightarrow FOXBY0A \\ \xRightarrow{*} FX0B0YA \\ \xRightarrow{*} F0W0B0Z0A \\ \xRightarrow{*} F00WBZ00A \\ \Rightarrow F00EB00A \\ \Rightarrow F00000A \\ \xRightarrow{*} 00000 \end{array}$$

produces five 0s. Notice how X and Y , and also W and Z , march back and forth across the strings of 0s, adding one more 0 on each side. This activity is highly reminiscent of a Turing machine read-write head sweeping back and forth across its tape and enlarging the printed portion. ●

The preceding grammar allows the erasing productions $F \rightarrow \lambda$ and $A \rightarrow \lambda$. To generate any language containing λ , we have to be able to erase somewhere. In the following grammar types, we will limit erasing, if it occurs at all, to a single production of the form $S \rightarrow \lambda$, where S is the start symbol; if this production occurs, we will not allow S to appear on the right-hand side of any other productions. This restriction allows us to crank out λ from S as a special case and then get on with other derivations, none of which allow any erasing. Let's call this the **erasing convention**. The following definition defines three special types of grammars by further restricting the productions allowed.

REMINDER

The erasing convention says how to do erasing IF erasing is to be done. It does not require the production $S \rightarrow \lambda$ in every grammar.

● **DEFINITIONS** **CONTEXT-SENSITIVE, CONTEXT-FREE, AND REGULAR GRAMMARS; CHOMSKY HIERARCHY**

A grammar G is **context-sensitive (type 1)** if it obeys the erasing convention and if, for every production $\alpha \rightarrow \beta$ (except $S \rightarrow \lambda$), the word β is at least as long as the word α . A grammar G is **context-free (type 2)** if it obeys the erasing convention and for every production $\alpha \rightarrow \beta$, α is a single nonterminal. A grammar G is **regular (type 3)** if it obeys the erasing convention and for every production $\alpha \rightarrow \beta$ (except $S \rightarrow \lambda$), α is a single nonterminal and β is of the form t or tW , where t is a terminal symbol and W is a nonterminal symbol. This hierarchy of grammars, from type 0 to type 3, is called the **Chomsky hierarchy**.

In a context-free grammar, a single nonterminal symbol on the left of a production can be replaced wherever it appears by the right side of the production. In a context-sensitive grammar, a given nonterminal symbol can perhaps be replaced only if it is part of a particular string (context)—hence the names *context-free* and *context-sensitive*. It is clear that any regular grammar is also context-free, and any context-free grammar is also context-sensitive. The grammar of Example 49 is regular (the two productions have the single nonterminal S on the left, and on the right either 1—a terminal—or $0S$ —a terminal followed by a nonterminal). Both grammars of Example 51 are context-free but not regular (again the single nonterminal S appears on the left of all productions, but the right sides consist of three symbols or two nonterminals or two terminals, respectively). The grammar of Example 52 is context-sensitive but not context-free (the productions do not shrink any strings, but some left sides have multiple symbols). The grammars of Example 53 and Example 54 are context-free but not regular (for example, the first three productions of Example 54 violate the requirement for a regular grammar). Finally, the grammar of Example 57 is a type 0 grammar, but it is not context-sensitive (for example, the production $EB \rightarrow 0$ is a “shrinking” production; also, the erasing convention is violated).

● **DEFINITION** **LANGUAGE TYPES**

A language is **type 0 (context-sensitive, context-free, or regular)** if it can be generated by a type 0 (context-sensitive, context-free, or regular) grammar.

Because of the relationships among the four grammar types, we can classify languages as shown in Figure 9.26. Thus, any regular language is also context-free because any regular grammar is also a context-free grammar, and so on. However, although it turns out to be true, we do not know from what we have done that these sets are properly contained in one another. For example, the language L described in Example 57 was generated in that example by a grammar that was type 0 but not context-sensitive, but that does not imply that L itself falls into that category. Different grammars can generate the same language.

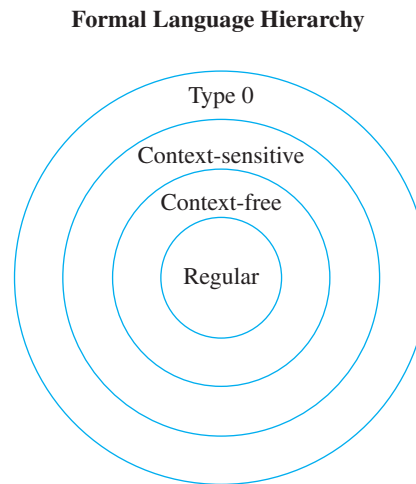


Figure 9.26

● **DEFINITION EQUIVALENT GRAMMARS**
Two grammars are **equivalent** if they generate the same language.

EXAMPLE 58

Example 57 gave a grammar G to generate a language L , which is λ together with all odd-length strings of 0s of length at least 3. We will now give three more grammars equivalent to G . (See if you agree that each of these grammars also generates L .)

$G_1 = (V, V_T, S, P)$ where $V = \{0, A, B, S\}$, $V_T = \{0\}$, and the productions are

$$\begin{array}{ll} S \rightarrow \lambda & AB \rightarrow 00 \\ S \rightarrow ABA & 0A \rightarrow 000A \\ & A \rightarrow 0 \end{array}$$

G_1 is context-sensitive but not context-free.

$G_2 = (V, V_T, S, P)$ where $V = \{0, A, S\}$, $V_T = \{0\}$, and the productions are

$$\begin{array}{ll} S \rightarrow \lambda & A \rightarrow 00A \\ S \rightarrow 00A & A \rightarrow 0 \end{array}$$

G_2 is context-free but not regular.

$G_3 = (V, V_T, S, P)$ where $V = \{0, A, B, C, S\}$, $V_T = \{0\}$, and the productions are

$$\begin{array}{ll} S \rightarrow \lambda & B \rightarrow 0 \\ S \rightarrow 0A & B \rightarrow 0C \\ A \rightarrow 0B & C \rightarrow 0B \end{array}$$

G_3 is regular.

Thus, when all is said and done, L is a regular language. ●

PRACTICE 65 Give the derivation of 00000 in G_1 , G_2 , and G_3 .

This is a somewhat confusing point and worth repeating. Because a given language can be generated by multiple grammars, if you find a grammar G for a language L that is type x but not type y , that does not necessarily make L type x but not type y . There might be a different grammar that also generates L and is a type y grammar.

Formal Languages and Computational Devices

The language L of Example 57 can be described by the regular expression $\lambda \vee (000)(00)^*$, so L is a regular set. From Example 58, L is also a regular language. It is not coincidental that a regular set turned out to be a regular language. It can be shown that for any finite-state machine, the set it recognizes is a regular language. It can also be shown that for any regular language, there is a finite-state machine that recognizes exactly that language. (In the proofs of these results, the productions of a regular grammar correspond to the state transitions of a finite-state machine.) Hence those sets recognized by finite-state machines—the regular sets—correspond to regular languages. Therefore the class of sets recognized by a computational device of limited capacity coincides with the most restricted class of languages.

On the other end of the spectrum, the most general computational device is the Turing machine and the most general language is a type 0 language. As it happens, the sets recognized by Turing machines correspond to type 0 languages.

There are computational devices with capabilities midway between those of finite-state machines and those of Turing machines. These devices recognize exactly the context-free languages and the context-sensitive languages, respectively.

The type of device that recognizes the context-free languages is called a **pushdown automaton**, or **pda**. A pda consists of a finite-state unit that reads input from a tape and controls activity in a stack. Symbols from some alphabet can be pushed onto or popped off of the top of the stack. The finite-state unit in a pda, as a function of the input symbol read, the present state, and the top symbol on the stack, has a finite number of possible next moves. The moves are of the following types:

1. Go to a new state, pop the top symbol off the stack, and read the next input symbol.
2. Go to a new state, pop the top symbol off the stack, push a finite number of symbols onto the stack, and read the next input symbol.
3. Ignore the input symbol being read, manipulate the stack as above, but do not read the next input symbol.

A pda has a choice of next moves, and it recognizes the set of all inputs for which some sequence of moves exists that causes it to empty its stack. It can be shown that any set recognized by a pda is a context-free language, and conversely.

The type of device that recognizes the context-sensitive languages is called a **linear bounded automaton**, or **lba**. An lba is a Turing machine whose read-write head is restricted to a portion of the tape that is no longer than a constant multiple of the length of the original input; in addition, at each step it has a choice of possible next moves. An lba recognizes the set of all inputs for which some sequence

of moves exists that causes it to halt in a final state. Any set recognized by an lba can be shown to be a context-sensitive language, and conversely.

Figure 9.27 shows the relationship between the hierarchy of languages and the hierarchy of computational devices.

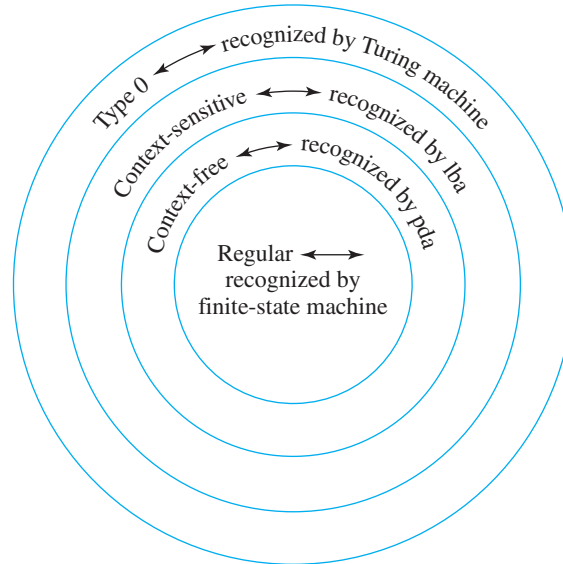


Figure 9.27

Context-Free Grammars

Context-free grammars are important for three reasons. Context-free grammars seem to be the easiest to work with, since they allow replacing only one symbol at a time. Furthermore, many programming languages are defined such that sections of syntax, if not the whole language, can be described by context-free grammars. Finally, a derivation in a context-free grammar has a nice graphical representation called a **parse tree**.

EXAMPLE 59

The grammar of Example 54 is context-free. The word $d2q$ can be derived as follows: **identifier** \Rightarrow **identifier letter** \Rightarrow **identifier digit letter** \Rightarrow **letter digit letter** \Rightarrow **d digit letter** \Rightarrow **d2 letter** \Rightarrow **d2q**. We can represent this derivation as a tree with the start symbol for the root. When a production is applied to a node, that node is replaced at the next lower level of the tree by the symbols in the right-hand side of the production used. A tree for the derivation appears in Figure 9.28.

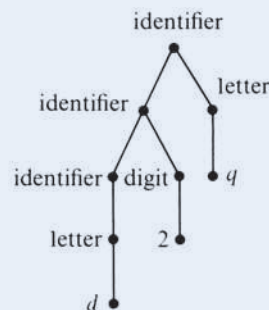


Figure 9.28

PRACTICE 66 Draw a parse tree for the word $m34s$ in the grammar of Example 54. ■

Suppose that a context-free grammar G describes a programming language. The programmer uses the rules of G to generate legitimate strings of symbols, that is, words in the language. Here we may think of a word as corresponding to a program instruction. Thus, a word consists of various subwords, for example, identifiers, operators, and key words for the language. The program instructions are fed into the compiler for the language so that the program can be translated into machine language code for the computer. The compiler must decide whether each program instruction is legitimate in the language. In a sense, the compiler has to undo the process that the programmer used to construct the statement; that is, the compiler must start with the statement and decompose it to see whether it follows the rules of the language. This really entails two questions: Are the subwords themselves legitimate strings? Is the program instruction a legitimate way of grouping the subwords together?

Usually the set of legitimate subwords of a language can be described by a regular expression, and then a finite-state machine can be used to detect the subwords; the lexical analysis or scanner portion of the compiler handles this phase of compilation. (See Exercise 32 of Section 9.3 for the lexical analysis of ScrubOak.) If all goes well, the scanner then passes the program instruction, in the form of a string of legitimate subwords, to the syntax analyzer. The syntax analyzer determines whether the string is correct by trying to parse it (construct its parse tree).

Various parsing techniques, which we won't go into, have been devised. Given a string to be tested, one approach is to construct a tree by beginning with the start symbol, applying productions (keeping an eye on the "goal"—the given string) and ending with the goal string. This procedure is called **top-down parsing**. The alternative is to begin with the string, see what productions were used to create it, apply productions "backwards," and end with the start symbol. This process is called **bottom-up parsing**. The trick to either approach is to decide exactly which productions should be used.

EXAMPLE 60

Consider the context-free grammar G given by $G = (V, V_T, S, P)$ where $V = \{a, b, c, A, B, C, S\}$, $V_T = \{a, b, c\}$, and the productions are

$$\begin{array}{lll} S \rightarrow B & B \rightarrow C & A \rightarrow abc \\ S \rightarrow A & B \rightarrow ab & C \rightarrow c \end{array}$$

Suppose we want to test the string abc . A derivation for abc is $S \Rightarrow A \Rightarrow abc$. If we try a top-down parse, we might begin with

$$\begin{array}{c} S \\ \downarrow \\ B \end{array}$$

Then we have to detect that this will not work and try something else. If we try a bottom-up parse, we might begin with



Then we have to detect that this will not work and try something else. Parsing techniques automate this process and attempt to minimize the amount of false starts and backtracking required. ●

Notice the distinction between *generating* members of a set, which the programmer does, and *deciding* membership in a set, which the compiler does. Since we ask the compiler to decide membership in a set, a decision algorithm must exist for the set. It turns out that decision algorithms do exist for context-free languages, another point in their favor.

SECTION 9.5 REVIEW

TECHNIQUES

- Describe $L(G)$ for a given grammar G .
- W Define a grammar to generate a given language L .
- Construct parse trees in a context-free grammar.

MAIN IDEAS

- A grammar G is a generating mechanism for its language $L(G)$.
- Formal languages were developed in an attempt to describe correct syntax for natural languages; although this attempt has largely failed because of the complexity of natural languages, it has been quite successful for high-level programming languages.

- Special classes of grammars are defined by restricting the allowable productions.
- The various types of formal languages correspond to the sets recognized by various automata; in particular, (1) regular languages are the sets recognized by finite-state machines, (2) context-free languages are the sets recognized by pushdown automata, (3) context-sensitive languages are the sets recognized by linear bounded automata, and (4) type 0 languages are the sets recognized by Turing machines.
- Derivations in context-free grammars can be illustrated by parse trees.
- A compiler for a context-free programming language checks correct syntax by parsing.

EXERCISES 9.5

1. Describe $L(G)$ for the grammar $G = (V, V_T, S, P)$ where $V = \{a, A, B, C, S\}$, $V_T = \{a\}$, and P consists of

$$\begin{array}{ll} S \rightarrow A & B \rightarrow A \\ A \rightarrow BC & aC \rightarrow \lambda \\ A \rightarrow a & \end{array}$$

What type of grammar is this?

2. Describe $L(G)$ for the grammar $G = (V, V_T, S, P)$ where $V = \{0, 1, A, B, S\}$, $V_T = \{0, 1\}$, and P consists of

$$\begin{array}{lll} S \rightarrow 0A & A \rightarrow 1BB & B \rightarrow 01 \\ S \rightarrow 1A & & B \rightarrow 11 \end{array}$$

What type of grammar is this?

3. Describe $L(G)$ for the grammar $G = (V, V_T, S, P)$ where $V = \{0, 1, A, B, S\}$, $V_T = \{0, 1\}$, and P consists of

$$\begin{array}{lll} S \rightarrow 0 & A \rightarrow 1B & B \rightarrow 0A \\ S \rightarrow 0A & & B \rightarrow 0 \end{array}$$

What type of grammar is this?

4. Describe $L(G)$ for the grammar $G = (V, V_T, S, P)$ where $V = \{0, 1, A, S\}$, $V_T = \{0, 1\}$, and P consists of

$$\begin{array}{ll} S \rightarrow 0S & A \rightarrow 1A \\ S \rightarrow 11A & A \rightarrow 1 \end{array}$$

What type of grammar is this?

5. Find a regular grammar that generates the language of Exercise 1.
 6. Find a regular grammar that generates the language of Exercise 2.
 7. Find a regular grammar that generates the language of Exercise 3.
 8. Find a regular grammar that generates the language of Exercise 4.
 9. Describe $L(G)$ for the grammar $G = (V, V_T, S, P)$ where $V = \{a, b, A, B, S\}$, $V_T = \{a, b\}$, and P consists of

$$\begin{array}{l} S \rightarrow aA \\ S \rightarrow \lambda \\ A \rightarrow bS \end{array}$$

10. Describe $L(G)$ for the grammar $G = (V, V_T, S, P)$ where $V = \{0, 1, A, S\}$, $V_T = \{0, 1\}$, and P consists of

$$\begin{array}{l} S \rightarrow 0 \\ S \rightarrow ASA \\ A \rightarrow 1 \end{array}$$

11. Describe $L(G)$ for the grammar $G = (V, V_T, S, P)$ where $V = \{a, b, A, B, S\}$, $V_T = \{a, b\}$, and P consists of

$$\begin{array}{ll} S \rightarrow AB & AB \rightarrow AAB \\ A \rightarrow a & AB \rightarrow ABB \\ B \rightarrow b & \end{array}$$

What type of grammar is G ? Find a regular grammar G' that generates $L(G)$.

12. Describe $L(G)$ for the grammar $G = (V, V_T, S, P)$ where $V = \{a, b, c, A, B, C, R, T, S\}$, $V_T = \{a, b, c\}$, and P consists of

$$\begin{array}{ll} S \rightarrow ARBT & T \rightarrow cT \\ R \rightarrow ARB & T \rightarrow c \\ RB \rightarrow ACBB & A \rightarrow a \\ CB \rightarrow BC & B \rightarrow b \\ CT \rightarrow cT & \end{array}$$

What type of grammar is G ? Find a context-free grammar G' that generates $L(G)$. Explain why it is not possible to find a regular grammar that generates $L(G)$.

13. Write the productions of the following grammars in BNF:
- G in Exercise 2
 - G in Exercise 3
 - G in Exercise 4
14. Write the productions of the following grammars in BNF:
- G in Exercise 10
 - G_3 in Example 58
15. A grammar G is described in BNF as

$$\langle S \rangle ::= 1|1\langle S \rangle|\langle S \rangle 00$$

- Find a regular expression for $L(G)$.
 - Find a regular grammar for $L(G)$.
16. A grammar G is described in BNF as

$$\langle S \rangle ::= 01|0\langle S \rangle|\langle S \rangle 1$$

- Find a regular expression for $L(G)$.
 - Find a regular grammar for $L(G)$.
17. Find a grammar that generates the set of all strings of well-balanced parentheses.
18. English words are translated into “pig latin” by the following two rules:
- If a word begins with a vowel, add the suffix “yay”.
 - If a word begins with a string of one or more consonants, move the string of leading consonants to the back of the word and then add the suffix “ay”.
- What is the pig latin translation of the word “apple”?
 - What is the pig latin translation of the word “monkey”?
 - What is the pig latin translation of the word “chain”?
 - Find a grammar that generates all legal pig latin words.
 - Using your grammar from part d, generate the pig latin word that was the answer to part c.
19. A word w in V^* is a palindrome if $w = w^R$, where w^R is the reverse of the string w . A language L is a palindrome language if L consists entirely of palindromes. Find a grammar that generates the set of all palindromes over the alphabet $\{a, b\}$.
20. a. Let L be a palindrome language (see Exercise 19). Prove that $L^R = \{w^R | w \in L\}$ is a palindrome language.
 b. Let w be a palindrome. Prove that the language described by the regular expression w^* is a palindrome language.
21. Find a regular grammar that generates the language $L = 11(0 \vee 1)^*$.
22. Find a regular grammar that generates the language $L = (0 \vee 1)^*01$.
23. Find a grammar that generates the language $L = \{1^{2^n} | n \geq 0\}$.
24. Find a regular expression for the language of Exercise 23. Is your grammar from Exercise 23 regular? If not, write a regular grammar that generates L .
25. Find a context-free grammar that generates the language $L = \{0^n 1^n | n \geq 0\}$.

26. Find a context-free grammar that generates the language $L = \{ww^R \mid w \in \{0, 1\}^*\}$ and w^R is the reverse of the string w .
27. Find a context-free grammar that generates the language L where L consists of the set of all nonempty strings of 0s and 1s with an equal number of 0s and 1s.
28. Find a context-free grammar that generates the language L where L consists of the set of all nonempty strings of 0s and 1s with twice as many 0s as 1s.
29. Find a grammar that generates the language $L = \{0^{2^i} \mid i \geq 0\}$.
30. Find a grammar that generates the language $L = \{0^n 1^{2^n} 0^n \mid n \geq 0\}$.
31. Find a grammar that generates the language $L = \{ww \mid w \in \{0, 1\}^*\}$.
32. Find a grammar that generates the language $L = \{a^n \mid n \geq 1\}$. (By Exercise 38, L is not a context-free language, so your grammar cannot be too simple.)
33. Draw parse trees for the following words:
- 111111 in the grammar G of Example 51
 - 111111 in the grammar G' of Example 51
34. Draw parse trees for the following words:
- 011101 in the grammar G of Exercise 2
 - 00111111 in the grammar G of Exercise 4
35. Consider the context-free grammar $G = (V, V_T, S, P)$ where $V = \{0, 1, A, S\}$, $V_T = \{0, 1\}$, and P consists of

$$S \rightarrow A1A$$

$$A \rightarrow 0$$

$$A \rightarrow A1A$$

Draw two distinct parse trees for the word 01010 in G . A grammar in which a word has more than one parse tree is *ambiguous*.

36. Ambiguity in a grammar (see Exercise 35) that describes a programming language is an undesirable trait because the compiler is unable to uniquely parse the programming instruction. Ambiguity in a natural language can be similarly confusing. The following two instructions were posted beside an escalator.

“Shoes must be worn.”

“Dogs must be carried.”

Give two possible interpretations for each instruction.

37. Show that for any context-free grammar G there exists a context-free grammar G' in which for every production $\alpha \rightarrow \beta$, β is a longer string than α , $L(G') \subseteq L(G)$ and $L(G) - L(G')$ is a finite set.
38. Following is the *pumping lemma* for context-free languages. Let L be any context-free language. Then there exists some constant k such that for any word w in L with $|w| \geq k$, w can be written as the string $w_1 w_2 w_3 w_4 w_5$ with $|w_2 w_3 w_4| \leq k$ and $|w_2 w_4| \geq 1$. Furthermore, the word $w_1 w_2^i w_3 w_4^i w_5 \in L$ for each $i \geq 0$.
- Use the pumping lemma to show that $L = \{a^n b^n c^n \mid n \geq 1\}$ is not context-free.
 - Use the pumping lemma to show that $L = \{a^n \mid n \geq 1\}$ is not context-free.

CHAPTER 9 REVIEW

TERMINOLOGY

- addition modulo n (p. 692)
 alphabet (pp. 694, 782)
 alternating group (p. 701)
 associative binary operation (p. 687)
 Backus Naur form (BNF) (p. 787)
 bottom-up parsing (p. 794)
 cancellation laws (p. 696)
 canonical parity-check matrix (p. 721)
 check bits in a code word (p. 721)
 Chomsky hierarchy (p. 790)
 coefficient (p. 690)
 commutative binary operation (p. 687)
 commutative group (p. 688)
 concatenation (p. 694)
 context-free (type 2) grammar (p. 790)
 context-free language (p. 790)
 context-sensitive (type 1) grammar (p. 790)
 context-sensitive language (p. 790)
 coset leader (p. 723)
 cosets of a subgroup in a group (p. 717)
 decision problem (p. 772)
 degree of a polynomial (p. 691)
 delay element (p. 744)
 direct generation (derivation) of a word (p. 784)
 double-error detecting code (p. 715)
 empty string (p. 694)
 equivalent grammars (p. 791)
 equivalent states (p. 739)
 erasing convention (p. 789)
 even and odd permutations (p. 700)
 final state (pp. 734, 765)
 finite-state machine (p. 729)
 formal language (p. 785)
 free monoid generated by a set A (p. 695)
 generation (derivation) of a word (p. 784)
 group (p. 688)
 group code (p. 719)
 group of permutations on a set A (p. 693)
 halting problem (p. 774)
 Hamming distance (p. 718)
 homomorphism (p. 702)
 identity element (p. 687)
 improper subgroup (p. 700)
 information bits in a code word (p. 721)
 input alphabet (p. 729)
 intractable problem (p. 776)
 inverse element (p. 687)
 isomorphism (p. 702)
 k -equivalent states (p. 739)
 kernel of a homomorphism (p. 716)
 language (p. 782)
 language generated by a grammar G (p. 784)
 left cancellation law (p. 696)
 left cosets (p. 717)
 length of a string (p. 694)
 linear bounded automaton (lba) (p. 792)
 maximum likelihood decoding (p. 715)
 minimum distance of a code (p. 718)
 monoid (p. 689)
 multiplication modulo n (p. 692)
 negative solution to a decision problem (p. 772)
 next-state function (p. 729)
 nondeterministic Turing machine (p. 777)
 NP (p. 777)
 NP -complete problem (p. 778)
 number-theoretic function (p. 767)
 order of a group (p. 698)
 output alphabet (p. 729)
 output function (p. 729)
 P (p. 777)
 parse tree (p. 793)
 partial function (p. 767)
 partition refinement (p. 741)
 perfect code (p. 722)
 permutation group (p. 700)
 phrase-structure (type 0) grammar (p. 783)
 polynomial in x with real number coefficients (polynomial in x over R) (p. 690)
 polynomial of zero degree (p. 691)
 positive solution to a decision problem (p. 772)
 production (p. 783)
 proper subgroup (p. 700)
 pushdown automaton (pda) (p. 792)
 recognition by a finite-state machine (p. 734)
 recognition (acceptance) by a Turing machine (p. 765)
 refinement (p. 741)
 regular expression (p. 735)
 regular (type 3) grammar (p. 790)
 regular language (p. 790)
 regular set (p. 736)
 right cancellation law (p. 696)
 right cosets (p. 717)
 semigroup (p. 689)
 semigroup of transformations on a set A (p. 693)
 sequential network (p. 745)
 single-error correcting code (p. 715)
 start symbol (p. 783)
 state graph (p. 730)
 state table (p. 730)
 string (p. 694)
 subgroup (p. 699)
 symbol (p. 694)
 symmetric group of degree n (p. 693)
 syndrome of a binary n -tuple in a group code (p. 724)

tape alphabet (p. 762)
 terminal (p. 783)
 top-down parsing (p. 794)
 total function (p. 767)
 transposition (p. 700)

Turing-computable function
 (p. 768)
 Turing machine (p. 762)
 type 0 language (p. 790)
 unreachable state (p. 737)

unsolvable (undecidable) decision
 problem (p. 772)
 vocabulary (p. 782)
 weight of a code word (p. 719)
 word (pp. 694, 782)

SELF-TEST

Answer the following true–false questions.

Section 9.1

1. A binary operation is associative if the order of the elements being operated upon does not matter.
2. The identity i in a group $[G, \cdot]$ has the property that $x^{-1} \cdot i = i \cdot x^{-1} = x^{-1}$ for all x in G .
3. Every group is also a monoid.
4. A group of order 10 cannot have a subgroup of order 6.
5. If $[S, \cdot]$ and $[T, +]$ are two groups, then a function $f: S \rightarrow T$ for which $f(x \cdot y) = f(x) + f(y)$ is an isomorphism.

Section 9.2

1. A binary single-error correcting code must have minimum distance at least 3.
2. In a canonical $n \times r$ parity-check matrix H , the bottom r rows form the $r \times r$ identity matrix.
3. A canonical $n \times r$ parity-check matrix H maps all of Z_2^m to code words in Z_2^n .
4. If the syndrome of a received word X in Z_2^n is 0_r , then X is assumed to be a code word.
5. A perfect code is “perfect” because no bit errors will occur during transmission.

Section 9.3

1. The next state of a finite-state machine is determined by its present state and the present input symbol.
2. The set of all binary strings ending in two 0s is regular.
3. A finite-state machine cannot get to a state from which there is no exit.
4. According to Kleene’s theorem, a set that cannot be described by a regular expression cannot be recognized by a finite-state machine.

5. In a finite-state machine, k -equivalent states are also $(k + 1)$ -equivalent.

Section 9.4

1. A Turing machine halts if and only if it enters a final state.
2. A Turing machine that computes the function $f(n) = n + 1$, given input n , will halt with $(n + 1)$ 1s on its tape.
3. Church’s thesis says that the halting problem is unsolvable.
4. The halting problem says that, given a Turing machine and its input, there is no algorithm to decide whether the Turing machine halts when run on that input.
5. A set in P is recognizable by a Turing machine in no more than a polynomial number of steps.

Section 9.5

1. The language generated by a type 0 grammar G is the set of all strings of terminals generated from the start symbol by applying G ’s productions a finite number of times.
2. Beginning at the start symbol and applying the productions of a grammar G eventually leads to a string of terminals.
3. A language generated by a grammar that is context-sensitive but not context-free is a context-sensitive but not context-free language.
4. Any regular set is a regular language.
5. A parse tree will have as many leaves as terminal symbols in the word being derived.

ON THE COMPUTER

For Exercises 1–11, write a computer program that produces the desired output from the given input.

1. *Input:* Two words from an alphabet A
Output: Their concatenation
2. *Input:* Positive integer n and finite alphabet A
Output: All words over A of length $\leq n$
3. *Input:* Positive integer n
Output: Tables for addition and multiplication modulo n
4. *Input:* Positive integer n
Output: The $n!$ elements of S_n expressed both in array form and in cycle notation, the group table for $[S_n, \circ]$, and the group table for $[A_n, \circ]$
5. *Input:* $n \times n$ array, $n \leq 10$, that purports to represent a binary operation on the finite set of integers from 1 to n
Output: Determination of whether the set under this operation is a commutative group
6. *Input:* Two $n \times n$ arrays, $n \leq 10$, that represent two groups and an array that represents a function from the first group to the second
Output: Determination of whether the function is an isomorphism
7. *Input:* A canonical $n \times r$ parity-check matrix H for a single-error correcting code with $r \leq 4$ and $n \leq 2^r - 1$
Output: The set of binary m -tuples H encodes and the code word for each one
8. *Input:* A canonical parity-check matrix for a perfect single-error correcting code where $r \leq 5$ and any binary n -tuple X
Output: The binary n -tuple to which X is decoded
9. *Input:* Positive integer n , $n \leq 50$, representing the number of states of a finite-state machine with input alphabet = output alphabet = $\{0, 1\}$ and an $n \times 3$ array representing the state table description of such a machine
Output: List of any states unreachable from the start state s_0
10. *Input:* Positive integer n , $n \leq 50$, representing the number of states of a finite state machine with input alphabet = output alphabet = $\{0, 1\}$ and an $n \times 3$ array representing the state table description of such a machine
Output: $m \times 3$ array representing the state table of a minimized version of M
11. *Input:* Set of terminals in a grammar and a description of the productions in a grammar; ability for the user to set a maximum number of steps for any derivation
Output: List of words in the language that can be derived within that maximum
12. Write a finite-state machine simulator. That is, given
 - a positive integer n , $n \leq 50$, representing the number of states of a finite state machine with input alphabet = output alphabet = $\{0, 1\}$
 - an $n \times 3$ array representing the state table description of such a machine
 your program should request input strings and write the corresponding output strings as long as the user wishes.
13. Write a Turing machine simulator. That is, given a set of quintuples describing a Turing machine, your program should request the initial tape contents and write out a sequence of successive tape configurations. Assume that there are at most 100 quintuples and that the number of cells used on the tape is at most 70, and allow the user to set a maximum number of steps in case the computation does not halt before then

This page intentionally left blank



Derivation Rules for Propositional and Predicate Logic

EQUIVALENCE RULES		
Expression	Equivalent to	Name/Abbreviation for Rule
$P \vee Q$ $P \wedge Q$	$Q \vee P$ $Q \wedge P$	Commutative—comm
$(P \vee Q) \vee R$ $(P \wedge Q) \wedge R$	$P \vee (Q \vee R)$ $P \wedge (Q \wedge R)$	Associative—ass
$(P \vee Q)'$ $(P \wedge Q)'$	$P' \wedge Q'$ $P' \vee Q'$	De Morgan's laws—De Morgan
$P \rightarrow Q$	$P' \vee Q$	Implication—imp
P	$(P')'$	Double negation—dn
$[(\exists x)A(x)]'$	$(\forall x)[A(x)]'$	Negation—neg

INFERENCE RULES		
From	Can Derive	Name/Abbreviation for Rule
$P, P \rightarrow Q$	Q	Modus ponens—mp
$P \rightarrow Q, Q'$	P'	Modus tollens—mt
P, Q	$P \wedge Q$	Conjunction—con
$P \wedge Q$	P, Q	Simplification—sim
P	$P \vee Q$	Addition—add
$P \rightarrow Q, Q \rightarrow R$	$P \rightarrow R$	Hypothetical syllogism—hs
$P \vee Q, P'$	Q	Disjunctive syllogism—ds
$P \rightarrow Q$	$Q' \rightarrow P'$	Contraposition—cont
$Q' \rightarrow P'$	$P \rightarrow Q$	Contraposition—cont
P	$P \wedge P$	Self-reference—self
$P \vee P$	P	Self-reference—self
$(P \wedge Q) \rightarrow R$	$P \rightarrow (Q \rightarrow R)$	Exportation—exp
P, P'	Q	Inconsistency—inc
$P \wedge (Q \vee R)$	$(P \wedge Q) \vee (P \wedge R)$	Distributive—dist
$P \vee (Q \wedge R)$	$(P \vee Q) \wedge (P \vee R)$	Distribution—dist

INFERENCE RULES (CONTINUED)			
From	Can Derive	Name/Abbreviation for Rule	Restrictions on Use
$(\forall x)P(x)$	$P(t)$, where t is a variable or constant symbol	Universal instantiation—ui	If t is a variable, it must not fall within the scope of a quantifier for t .
$(\exists x)P(x)$	$P(a)$ where a is a constant symbol not previously used in proof sequence	Existential instantiation—ei	Must be the first rule used that introduces a .
$P(x)$	$(\forall x)P(x)$	Universal generalization—ug	$P(x)$ has not been deduced from any hypotheses in which x is a free variable nor has $P(x)$ been deduced by ei from any wff in which x is a free variable.
$P(x)$ or $P(a)$ where a is a constant symbol	$(\exists x)P(x)$	Existential generalization—eg	To go from $P(a)$ to $(\exists x)P(x)$, x must not appear in $P(a)$.



Summation and Product Notation

Summation notation is a shorthand way of writing certain expressions that are sums of terms. As an example, consider the sum of the integers from 1 to 5:

$$1 + 2 + 3 + 4 + 5$$

This expression can be thought of in the following way: Suppose we have some quantity i that initially has the value 1 and then takes on successive values of 2, 3, 4, and 5. The expression is the sum of i at all its different values. The summation notation is

$$\sum_{i=1}^5 i$$

The uppercase Greek letter sigma, Σ , denotes summation. Here the number 1 is the *lower limit of summation*, and the number 5 is the *upper limit of summation*. The variable i is called the *index of summation*. The index of summation is initially set equal to the lower limit and then keeps increasing its value by 1 until it reaches the upper limit. All the values that the index of summation takes on are added together. Thus

$$\sum_{i=1}^5 i = 1 + 2 + 3 + 4 + 5 = 15$$

Similarly

$$\sum_{i=1}^3 i = 1 + 2 + 3 = 6$$

And

$$\sum_{i=4}^8 i = 4 + 5 + 6 + 7 + 8 = 30$$

In these examples, the expression after the summation sign is just i , the index of summation. However, what appears after the summation sign can be any expression, and the successive values of the index are simply substituted into the expression. Thus

$$\sum_{i=1}^5 i^2 = 1^2 + 2^2 + 3^2 + 4^2 + 5^2 = 55$$

A way to symbolize summation in general is

$$\sum_{i=p}^q a_i$$

Here the lower limit, upper limit, and expression behind the summation are not specifically given but merely symbolized. The notation a_i is a reminder that the expression will be evaluated at the different values i takes on in going from the lower to the upper limit.

There are three special cases to consider:

$$1. \sum_{i=p}^q 0 = 0$$

Here the expression behind the summation is the constant 0, which has the value 0 no matter what the value of the index of summation. The sum of any number of 0s is 0.

$$2. \sum_{i=1}^n 1 = n$$

Here again the expression behind the summation is a constant, and the summation says to add n copies of 1, which results in n .

$$3. \sum_{i=1}^0 a_i = 0$$

Here the upper limit is smaller than the lower limit; the usual interpretation of summation does not apply, but by convention the summation is assigned the value 0.

The index of summation is a *dummy variable*, meaning that it merely acts as a placeholder and that a different variable could be used without changing the value of the summation. Thus

$$\sum_{i=1}^3 i = \sum_{j=1}^3 j = 6$$

It may be convenient to change the limits on a summation, which is legitimate as long as the final value of the summation remains the same. For example,

$$\sum_{i=1}^3 i = \sum_{i=0}^2 (i + 1)$$

since both have the value

$$1 + 2 + 3 = 6$$

Finally, the following three rules hold, as we will see shortly.

Rules of Summation

1. $\sum_{i=p}^q (a_i + b_i) = \sum_{i=p}^q a_i + \sum_{i=p}^q b_i$
2. $\sum_{i=p}^q (a_i - b_i) = \sum_{i=p}^q a_i - \sum_{i=p}^q b_i$
3. $\sum_{i=p}^q ca_i = c \sum_{i=p}^q a_i$ where c is a constant

To prove rule 1, note that

$$\begin{aligned} a_p + b_p + a_{p+1} + b_{p+1} + \cdots + a_q + b_q \\ = a_p + a_{p+1} + \cdots + a_q + b_p + b_{p+1} + \cdots + b_q \end{aligned}$$

because of the commutative property of addition. The proof of rule 2 is similar.

To prove rule 3, note that

$$ca_p + ca_{p+1} + \cdots + ca_q = c(a_p + a_{p+1} + \cdots + a_q)$$

because of the distributive property. This rule allows a constant to be “moved through” a summation.

Sometimes a summation can be represented by an even shorter expression that does not involve adding separate terms. For example, according to Exercise 7 of Section 2.2,

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6} \quad (1)$$

so that the value of $\sum_{i=1}^5 i^2$ can be found by substituting the upper limit, 5, into the right side of (1), giving

$$\frac{5(5+1)(2 \cdot 5+1)}{6} = 55$$

as before. Section 2.2 and its exercises give a number of other “closed form” expressions for certain summations, all of them provable by mathematical induction.

Product notation is a shorthand way of writing certain expressions that are products of factors. Product notation is very similar to summation notation, except it uses the uppercase Greek letter pi, Π and the various items are multiplied rather than added. There is an index of multiplication, a lower limit, and an upper limit. For example,

$$\prod_{i=3}^7 i = (3)(4)(5)(6)(7) = 2520$$

and

$$\prod_{i=1}^2 (2i + 5) = (2 \cdot 1 + 5)(2 \cdot 2 + 5) = (7)(9) = 63$$

Product notation occurs less often than summation notation, but there are still special cases to note:

1. $\prod_{i=p}^q 0 = 0$ (the product of all 0s equals 0)

2. $\prod_{i=p}^q 1 = 1$ (the product of all 1s equals 1)



The Logarithm Function

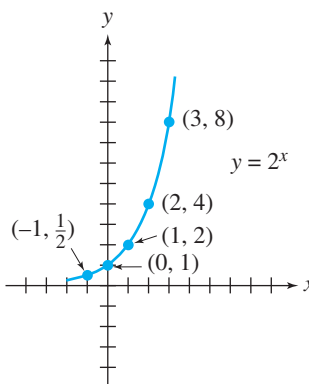
The logarithm function is closely related to the *exponential function*

$$y = b^x$$

where b , the *base*, is a constant greater than 1. (Actually b can be any positive number, but the only interesting cases occur when $b > 1$.) Recall the following rules of exponents:

1. $b^n b^m = b^{n+m}$ (when you multiply, you add exponents)
2. $b^n / b^m = b^{n-m}$ (when you divide, you subtract exponents)
3. $(b^n)^m = b^{nm}$ (when you raise a power to a power, you multiply exponents)

If we select a specific base, $b = 2$ for example, we can plot $y = 2^x$ for various values of x and fill in the remaining values, getting the graph



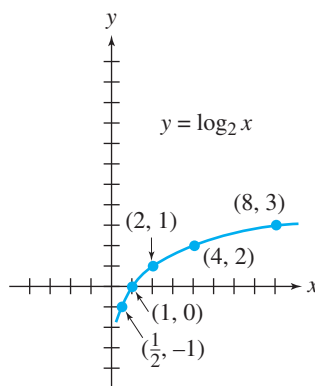
In this function, x can take on any real value, and y will always be positive. Another way to say this is that the domain of the function is the set \mathbb{R} of real numbers, and the range is the set \mathbb{R}^+ of positive real numbers.

A related function (in fact, the inverse function) is the *logarithm function*, defined by

$$y = \log_b x \quad \text{meaning} \quad b^y = x$$

Therefore $\log_2 16 = 4$, for example, because $2^4 = 16$. These two equations are the logarithmic form and exponential form of the same fact. Similarly, $\log_2 8 = 3$ and $\log_2 2 = 1$.

Following is a graph of $y = \log_2 x$.



Because the logarithm function $y = \log_b x$ is the inverse of the exponential function, its domain (the values x can take on) is the set \mathbb{R}^+ of positive real numbers and the range (the values y can take on) is the set \mathbb{R} of real numbers. The logarithm function for any base $b > 1$ has a domain, range, and shape similar to the case for $b = 2$.

Certain properties about the logarithm function are true either because of its definition or because of corresponding properties about the exponential function. We'll list all the properties of the logarithm function, and then prove them.

Properties of the Logarithm Function $y = \log_b x$

1. If $p < q$ then $\log_b p < \log_b q$ (the log function is strictly increasing)
2. If $\log_b p = \log_b q$ then $p = q$ (the log function is one to one)
3. $\log_b 1 = 0$
4. $\log_b b = 1$
5. $\log_b (b^p) = p$
6. $b^{\log_b p} = p$
7. $\log_b (pq) = \log_b p + \log_b q$ (the log of a product equals the sum of the logs)
8. $\log_b (p/q) = \log_b p - \log_b q$ (the log of a quotient equals the difference of the logs)
9. $\log_b (p^q) = q(\log_b p)$ (the log of something to a power equals the power times the log)
10. $\log_a p = \log_b p / \log_b a$ (change-of-base formula)
11. $p^{\log_b q} = q^{\log_b p}$

In proving the properties of the logarithm function, we first note that many of the properties involve the quantities $\log_b p$ and $\log_b q$. Let us name these quantities r and s , respectively, so that

$$\log_b p = r \text{ and } \log_b q = s$$

This in turn means that

$$b^r = p \text{ and } b^s = q$$

1. Because $b > 1$, the larger the power to which b is raised, the larger the result. Thus if $p < q$, then $b^r < b^s$, so $r < s$ and therefore $\log_b p < \log_b q$.
2. If $\log_b p = \log_b q$, then $r = s$, so $b^r = b^s$ and $p = q$.
3. $\log_b 1 = 0$ because $b^0 = 1$.
4. $\log_b b = 1$ because $b^1 = b$.
5. $\log_b(b^p) = p$ because (translating this equation to its exponential form) $b^p = b^p$.
6. $b^{\log_b p} = p$ because (translating this equation to its logarithmic form) $\log_b p = \log_b p$.
7. $\log_b(pq) = \log_b p + \log_b q = r + s$ because it is true that $b^{r+s} = b^r b^s = pq$, which is the exponential form of the equation we are trying to prove.
8. $\log_b(p/q) = \log_b p - \log_b q = r - s$ because it is true that $b^{r-s} = b^r/b^s = p/q$ which is the exponential form of the equation we are trying to prove.
9. $\log_b(p^q) = q(\log_b p) = qr$ because it is true that $b^{qr} = (b^r)^q = p^q$, which is the exponential form of the equation we are trying to prove.
10. $\log_a p = \log_b p / \log_b a$

Let $\log_a p = w$. Then $a^w = p$. Now take the logarithm to the base b of both sides of this equation:

$$\log_b(a^w) = w(\log_b a) = \log_b p$$

or

$$w = \log_b p / \log_b a$$

which is the desired result.

$$\begin{aligned} 11. \quad p^{\log_b q} &= q^{\log_b p} \\ \log_b(p^{\log_b q}) &= (\log_b q)(\log_b p) \text{ by Property 9} \\ &= (\log_b p)(\log_b q) \\ &= \log_b(q^{\log_b p}) \text{ by Property 9} \end{aligned}$$

Therefore $p^{\log_b q} = q^{\log_b p}$ by Property 2

The three most useful bases for logarithms are

$b = 10$ (common logarithm)

$b = e$, $e \sim 2.7183$ (natural logarithm)

$b = 2$ (what we use throughout this book)

Common logarithms were used as computational aids before calculators and computers became commonplace. Property 7 of the logarithm function says that to multiply two numbers, one can take the logarithm of each number, add the results, and then find the number with that logarithm value. Addition was easier than multiplication, and tables of common logarithms allowed one to look up a number and find its logarithm, or vice versa.

Natural logarithms are useful in calculus and are often written “ $\ln p$ ” rather than “ $\log_e p$.” Base 2 logarithms are sometimes denoted by “ $\lg p$ ” rather than “ $\log_2 p$.” In this book, all logarithms are base 2, so we use $\log p$ to denote $\log_2 p$.

A final inequality involving base 2 logarithms (used in Section 3.3) is

$$1 + \log n < n \text{ for } n \geq 3$$

To prove this, note that

$$n < 2^{n-1} \text{ for } n \geq 3$$

so by property 1 of logarithms,

$$\log n < \log 2^{n-1}$$

By property 5 of logarithms, $\log 2^{n-1} = n - 1$. Therefore

$$\log n < n - 1$$

or

$$1 + \log n < n \text{ for } n \geq 3$$

Answers to Practice Problems

Note to student: Finish all parts of a practice problem before turning to the answers.

CHAPTER 1

1. False, false, false

2.

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

3.

A	B	$A \rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

4.

A	A'
T	F
F	T

5. a. Antecedent: The rain continues Consequent: The river will flood
 b. Antecedent: The central switch goes down Consequent: Network failure
 c. Antecedent: The avocados are ripe Consequent: They are dark and soft
 d. Antecedent: A healthy cat Consequent: A good diet
6. Answer d. This is negation of $A \wedge B$, the same as the negation of "Peter is tall and thin."

7. a.

A	B	$A \rightarrow B$	$B \rightarrow A$	$(A \rightarrow B) \leftrightarrow (B \rightarrow A)$
T	T	T	T	T
T	F	F	T	F
F	T	T	F	F
F	F	T	T	T

b.

A	B	A'	B'	$A \vee A'$	$B \wedge B'$	$(A \vee A') \rightarrow (B \wedge B')$
T	T	F	F	T	F	F
T	F	F	T	T	F	F
F	T	T	F	T	F	F
F	F	T	T	T	F	F

c.

A	B	C	B'	A ∨ B'	C'	(A ∧ B') → C'	[(A ∧ B') → C']'
T	T	T	F	F	F	T	F
T	T	F	F	F	T	T	F
T	F	T	T	T	F	F	T
T	F	F	T	T	T	T	F
F	T	T	F	F	F	T	F
F	T	F	F	F	T	T	F
F	F	T	T	F	F	T	F
F	F	F	T	F	T	T	F

d.

A	B	A'	B'	A → B	B' → A'	(A → B) ↔ (B' → A')
T	T	F	F	T	T	T
T	F	F	T	F	F	T
F	T	T	F	T	T	T
F	F	T	T	T	T	T

8.

A	1	A'	A ∨ A'	A ∨ A' ↔ 1
T	T	F	T	T
F	T	T	T	T

9. To prove $(P \rightarrow Q) \leftrightarrow (P' \vee Q)$, just construct a truth table:

P	Q	P → Q	P'	P' ∨ Q	(P → Q) ↔ (P' ∨ Q)
T	T	T	F	T	T
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

10. $(A \wedge B)'$ 1, 2, mt

11. $[(A \vee B') \rightarrow C] \wedge (C \rightarrow D) \wedge A \rightarrow D$

1. $(A \vee B') \rightarrow C$ hyp

2. $C \rightarrow D$ hyp

3. A hyp

4. $A \vee B'$ 3, add

5. C 1, 4, mp

6. D 2, 5, mp

12. $(A \rightarrow B) \wedge (B \rightarrow C) \rightarrow (A \rightarrow C)$

1. $A \rightarrow B$ hyp

2. $B \rightarrow C$ hyp

3. A hyp

4. B 1, 3, mp

5. C 2, 4, mp

13. $(A \rightarrow B) \wedge (C' \vee A) \wedge C \rightarrow B$
1. $A \rightarrow B$ hyp
 2. $C' \vee A$ hyp
 3. C hyp
 4. $C \rightarrow A$ 2, imp
 5. $C \rightarrow B$ 1, 4, hs
 6. B 3, 5, mp
14. The argument is $(S \rightarrow R) \wedge (S' \rightarrow B) \rightarrow (R' \rightarrow B)$. A proof sequence is
1. $S \rightarrow R$ hyp
 2. $S' \rightarrow B$ hyp
 3. R' hyp
 4. S' 1, 3, mt
 5. B 2, 4, mp
15. a. True (all daffodils are yellow)
 b. False (not true that all flowers are yellow)
 c. True (all flowers are plants)
 d. False (zero is neither positive nor negative)
16. For example:
- a. The domain is the collection of licensed drivers in the United States; $P(x)$ is the property that x is older than 14.
 - b. The domain is the collection of all fish; $P(x)$ is the property that x weighs more than 3 pounds.
 - c. No; if all objects in the domain have property P , then (since the domain must contain objects) there is an object in the domain with property P .
 - d. The domain is all the people who live in Boston; $P(x)$ is the property that x is a male. (Not every person who lives in Boston is a male, but someone is.)
17. Let $x = 1$; then x is positive and any integer less than x is ≤ 0 , so the truth value of the statement is true. For the second interpretation, let $A(x)$ be “ x is even,” $B(x, y)$ be “ $x < y$,” and $C(y)$ be “ y is odd”; the statement is false because no even integer has the property that all larger integers are odd.
18. a. $(\forall x)[S(x) \rightarrow I(x)]$
 b. $(\exists x)[I(x) \wedge S(x) \wedge M(x)]$
 c. $(\forall x)(M(x) \rightarrow S(x) \wedge [I(x)]')$
 d. $(\forall x)(M(x) \rightarrow S(x) \wedge I(x))$
19. a. $(\exists x)[V(x) \wedge (\forall y)(F(y) \rightarrow S(x, y))]$
 b. $(\forall x)[F(x) \rightarrow (\forall y)(V(y) \rightarrow S(x, y))]$
 c. $(\forall x)[F(x) \rightarrow (\exists y)(V(y) \wedge S(x, y))]$
 d. $(\forall x)(\forall y)[V(y) \wedge S(x, y) \rightarrow F(x)]$
20. Answer d. If $L(x, y, t)$ means “ x loves y at time t ,” the original statement is

$$(\forall x)(\exists y)(\exists t)L(x, y, t)$$

and the negation is

$$\begin{aligned} [(\forall x)(\exists y)(\exists t)L(x, y, t)]' &\leftrightarrow (\exists x)[(\exists y)(\exists t)L(x, y, t)]' \\ &\leftrightarrow (\exists x)(\forall y)[(\exists t)L(x, y, t)]' \\ &\leftrightarrow (\exists x)(\forall y)(\forall t)[L(x, y, t)]' \end{aligned}$$

or, “There is some person who, for all persons and all times, dislikes those persons at those times” or “Somebody hates everybody all the time.” Answers (a) and (c) can be eliminated because they begin with a universal quantifier instead of an existential quantifier; answer (b) is wrong because “loves” has not been negated.

- 21.** Invalid. In the interpretation where the domain consists of the integers, $P(x)$ is “ x is odd” and $Q(x)$ is “ x is even,” the antecedent is true (every integer is even or odd), but the consequent is false (it is not the case that every integer is even or that every integer is odd).
- 22.** $(\forall x)[P(x) \rightarrow R(x)] \wedge [R(y)]' \rightarrow [P(y)]'$
1. $(\forall x)[P(x) \rightarrow R(x)]$ hyp
 2. $[R(y)]'$ hyp
 3. $P(y) \rightarrow R(y)$ 1, ui
 4. $[P(y)]'$ 2, 3, mt
- 23.** $(\forall x)[P(x) \wedge Q(x)] \rightarrow (\forall x)[Q(x) \wedge P(x)]$
1. $(\forall x)[P(x) \wedge Q(x)]$ hyp
 2. $P(x) \wedge Q(x)$ 1, ui
 3. $Q(x) \wedge P(x)$ 2, comm
 4. $(\forall x)[Q(x) \wedge P(x)]$ 3, ug
- 24.** $(\forall y)[P(x) \rightarrow Q(x, y)] \rightarrow [P(x) \rightarrow (\forall y)Q(x, y)]$
1. $(\forall y)[P(x) \rightarrow Q(x, y)]$ hyp
 2. $P(x)$ hyp
 3. $P(x) \rightarrow Q(x, y)$ 1, ui
 4. $Q(x, y)$ 2, 3, mp
 5. $(\forall y)Q(x, y)$ 4, ug
- 25.** $(\forall x)[(B(x) \vee C(x)) \rightarrow A(x)] \rightarrow (\forall x)[B(x) \rightarrow A(x)]$
1. $(\forall x)[(B(x) \vee C(x)) \rightarrow A(x)]$ hyp
 2. $(B(x) \vee C(x)) \rightarrow A(x)$ 1, ui
 3. $B(x)$ temporary hyp
 4. $B(x) \vee C(x)$ 3, add
 5. $A(x)$ 2, 4, mp
 6. $B(x) \rightarrow A(x)$ temporary hyp discharged
 7. $(\forall x)[B(x) \rightarrow A(x)]$ 6, ug
- 26.** $(\exists x)R(x) \wedge [(\exists x)[R(x) \wedge S(x)]]' \rightarrow (\exists x)[S(x)]'$
- The argument is valid. If something has property R but nothing has both property R and property S , then something fails to have property S . A proof sequence is
1. $(\exists x)R(x)$ hyp
 2. $[(\exists x)[R(x) \wedge S(x)]]'$ hyp

3. $(\forall x)[R(x) \wedge S(x)]'$ 2, neg
 4. $R(a)$ 1, ei
 5. $[R(a) \wedge S(a)]'$ 3, ui
 6. $[R(a)]' \vee [S(a)]'$ 5, De Morgan
 7. $[[R(a)]']'$ 4, dn
 8. $[S(a)]'$ 6, 7, ds
 9. $(\exists x)[S(x)]'$ 8, eg
- 27.** The argument is $(\forall x)[R(x) \rightarrow L(x)] \wedge (\exists x)R(x) \rightarrow (\exists x)L(x)$
1. $(\forall x)[R(x) \rightarrow L(x)]$ hyp
 2. $(\exists x)R(x)$ hyp
 3. $R(a)$ 2, ei
 4. $R(a) \rightarrow L(a)$ 1, ui
 5. $L(a)$ 3, 4, mp
 6. $(\exists x)L(x)$ 5, eg
- 28.** deer grass (deer eat grass and grass is a plant)
- 29.** a. $\text{predator}(X) \leq \text{eat}(X, Y)$ **and** $\text{animal}(Y)$
 b. bear
 fish
 raccoon
 bear
 bear
 fox
 bear
 wildcat
- 30.** Responses 7–9 result from $\text{in-food-chain}(\text{raccoon}, Y)$; responses 10 and 11 result from $\text{in-food-chain}(\text{fox}, Y)$; response 12 results from $\text{in-food-chain}(\text{deer}, Y)$.
- 31.** $x - 2 = y$, or $x = y + 2$
- 32.** Working backwards from the postcondition using the assignment rule,

$$\begin{aligned} &\{x + 4 = 7\} \\ &\quad y = 4 \\ &\{x + y = 7\} \\ &\quad z = x + y \\ &\{z = 7\} \end{aligned}$$

The first assertion, $x + 4 = 7$, is equivalent to the precondition, $x = 3$. The assignment rule, applied twice, proves the program segment correct.

- 33.** The two implications to prove are

$$\begin{aligned} &\{x = 4 \text{ and } x < 5\} y = x - 1 \{y = 3\} \\ &\{x = 4 \text{ and } x \geq 5\} y = 7 \{y = 3\} \end{aligned}$$

The first implication is true by the assignment rule. Working backwards from the postcondition,

$$\begin{aligned} &\{x - 1 = 3\} \\ &\quad y = x - 1 \\ &\{y = 3\} \\ &x - 1 = 3 \leftrightarrow x = 4 \leftrightarrow x = 4 \text{ and } x < 5 \end{aligned}$$

The second implication is true because the antecedent is false. The program segment is correct by the conditional rule.

CHAPTER 2

1. Possible answers:
 - a. A whale
 - b. The integer 4. Four is less than 10, but it is not bigger than 5.
2. a. Show that the conjecture is true for all cases:

n	n^2	$10 + 5n$
1	1	15
2	4	20
3	9	25
4	16	30
5	25	35

- b. For $n = 7$, n^2 is 49 but $10 + 5n$ is only 45.
3. Let x be divisible by 6. Then $x = 6k$ where k is an integer, and $2x = 2(6k) = 12k = 4(3k)$. Since $3k$ is an integer, $2x$ is divisible by 4.
4.
 - a. If the river will not flood, then the rain will not continue.
 - b. If there is not a network failure, then the central switch does not go down.
 - c. If the avocados are not dark or not soft, then they are not ripe.
 - d. If the diet is not good, the cat is not healthy.
5.
 - a. If the river will flood, then the rain will continue.
 - b. If there is network failure, then the central switch goes down.
 - c. If the avocados are dark and soft, then they are ripe.
 - d. If the diet is good, then the cat is healthy.
6. Let $x = 2m + 1$ and $y = 2n + 1$ where m and n are integers, and assume that xy is even. Then

$$xy = 2k \text{ for some integer } k$$

or

$$(2m + 1)(2n + 1) = 2k$$

Multiplying out the left side,

$$4mn + 2m + 2n + 1 = 2k$$

Rearranging terms in the equation,

$$1 = 2k - 4mn - 2m - 2n$$

Factoring out 2 on the right side,

$$1 = 2(k - 2mn - m - n) \quad \text{where } k - 2mn - m - n \text{ is an integer}$$

This is a contradiction since 1 is not even.

7. $P(1)$: $1 = 1(1 + 1)/2$, true

Assume $P(k)$: $1 + 2 + \cdots + k = k(k + 1)/2$

Show $P(k + 1)$: $1 + 2 + \cdots + (k + 1) \stackrel{?}{=} \frac{(k + 1)[(k + 1) + 1]}{2}$

$$\begin{aligned} 1 + 2 + \cdots + (k + 1) &= 1 + 2 + \cdots + k + (k + 1) \\ &= \frac{k(k + 1)}{2} + (k + 1) = (k + 1)\left(\frac{k}{2} + 1\right) \\ &= (k + 1)\left(\frac{k + 2}{2}\right) = \frac{(k + 1)[(k + 1) + 1]}{2} \end{aligned}$$

8. The base case is $n = 2$.

$P(2)$: $2^{2+1} < 3^2$, or $8 < 9$, true

Assume $P(k)$: $2^{k+1} < 3^k$ and $k > 1$

Show $P(k + 1)$: $2^{k+2} \stackrel{?}{<} 3^{k+1}$

$$\begin{aligned} 2^{k+2} &= 2(2^{k+1}) \\ &< 2(3^k) && \text{(by the inductive hypothesis)} \\ &< 3(3^k) && \text{(since } 2 < 3) \\ &= 3^{k+1} \end{aligned}$$

9. a. To verify $P(k + 1)$ in implication 2', we subtract 3 from $k + 1$. For the inductive hypothesis to hold, it must be the case that $(k + 1) - 3 \geq 8$, so $k + 1$ must be ≥ 11 . Therefore implication 2' cannot be used to verify $P(9)$ or $P(10)$.

b. The truth of $P(k + 1)$ cannot be verified from the truth of $P(k)$. For example, in trying to express 11 as a sum of 3's, and 5's, knowing that $10 = 5 + 5$ is no help. However, knowing that $8 = 3 + 5$ is helpful because adding one more 3 gives $11 = 2 * 3 + 5$.

10. $Q(0)$: $j_0 = x + i_0$ true since $j = x$, $i = 0$ before the loop is entered

Assume $Q(k)$: $j_k = x + i_k$

Show $Q(k + 1)$: $j_{k+1} \stackrel{?}{=} x + i_{k+1}$

$$\begin{aligned} j_{k+1} &= j_k + 1 && \text{(by the assignment } j = j + 1) \\ &= (x + i_k) + 1 && \text{(by inductive hypothesis)} \\ &= x + (i_k + 1) \\ &= x + i_{k+1} && \text{(by the assignment } i = i + 1) \end{aligned}$$

Upon loop termination, $i = y$ and $j = x + y$.

11. a. If $d|a$ then $a = n_1d$ where n_1 is a positive integer. If $d|b$ then $b = n_2d$ where n_2 is a positive integer. Therefore

$$c = ia + jb = i(n_1d) + j(n_2d) = (in_1 + jn_2)d$$

where $in_1 + jn_2$ is an integer and $d|c$.

- b. If $d|c$ then $c = nd$ where $n \geq 1$ because both c and d are positive, so $c \geq d$.

12. From the Euclidean algorithm to find $\gcd(21, 16)$,

$$\begin{aligned} 1 &= 16 - 3 \cdot 5 \\ 5 &= 21 - 1 \cdot 16 \end{aligned}$$

from which

$$1 = 16 - 3 \cdot (21 - 1 \cdot 16) = 4 \cdot 16 - 3 \cdot 21$$

so $i = -3, j = 4$.

13. Do a proof by mathematical induction.

Base case: ($k = 1$). If $p|a_1$ then $p|a_1$

Assume that if $p|a_1a_2 \dots a_k$ then $p|a_j$ for some $j, 1 \leq j \leq k$

Let $p|a_1a_2 \dots a_k a_{k+1} = (a_1a_2 \dots a_k)a_{k+1}$. Using the theorem on division by prime numbers, either $p|a_1a_2 \dots a_k$ or $p|a_{k+1}$. If $p|a_1a_2 \dots a_k$ then by the assumption $p|a_j$ for some $j, 1 \leq j \leq k$. Therefore $p|a_j$ for some $j, 1 \leq j \leq k + 1$.

14. $1176 = 2^3 \cdot 3 \cdot 7^2$
 15. $420 = 2^2 \cdot 3 \cdot 5 \cdot 7$ and $66 = 2 \cdot 3 \cdot 11$, so $\gcd(420, 66) = 2 \cdot 3 = 6$
 16. Because p is a prime number, it has no factors other than itself and 1. Therefore every positive integer less than p has only the factor 1 in common with p , so it is relatively prime to p . Therefore $\varphi(p) = p - 1$.
 17. $\varphi(n) = 3^3 \cdot 7[\varphi(3)\varphi(5)\varphi(7)] = 27 \cdot 7 \cdot 2 \cdot 4 \cdot 6 = 9072$

CHAPTER 3

- 1, 4, 7, 10, 13
- 1, 1, 2, 3, 5, 8, 13, 21
- In proving the $k + 1$ case, the terms $F(k - 1)$ and $F(k)$ are used. If $k + 1 = 2$, then the value at 2 positions back, $F(k - 1)$, is undefined. Therefore in the inductive step, we must have $k + 1 \geq 3$ and the case $n = 2$ must be done separately. Put another way, the inductive step does not demonstrate the truth of the $n = 2$ case from the truth of the $n = 1$ case.
- A, B , and C are wffs by rule 1. By rule 2, (B') is a wff, and so then is $(A \vee (B'))$ and $((A \vee (B')) \rightarrow C)$. This can be written as $(A \vee B') \rightarrow C$.
- Every parent of an ancestor of James is an ancestor of James.
- 1011001, 0011011, 00110111011

7. 1, 0, and 1 are binary palindromes.
2. If x is a binary palindrome, so are $0x0$ and $1x1$

8. 1. $x^1 = x$
2. $x^n = x^{n-1}x$ for $n > 1$

9. **if** $n = 1$ **then**
 return 1
 else
 return $T(n - 1) + 3$
 end if

10. 10, 7, 8

11. $T(n) = T(n - 1) + 3$
 $= [T(n - 2) + 3] + 3 = T(n - 2) + 2 * 3$
 $= [T(n - 3) + 3] + 2 * 3 = T(n - 3) + 3 * 3$
 \vdots

In general, we guess that

$$T(n) = T(n - k) + k * 3$$

When $n - k = 1$, that is, $k = n - 1$,

$$T(n) = T(1) + (n - 1) * 3 = 1 + (n - 1) * 3$$

Now prove by induction that $T(n) = 1 + (n - 1) * 3$.

$T(1)$: $T(1) = 1 + (1 - 1) * 3 = 1$, true

Assume $T(k)$: $T(k) = 1 + (k - 1) * 3$

Show $T(k + 1)$: $T(k + 1) \stackrel{?}{=} 1 + k * 3$

$$\begin{aligned} T(k + 1) &= T(k) + 3 && \text{(by the recurrence relation)} \\ &= 1 + (k - 1) * 3 + 3 && \text{(by the inductive hypothesis)} \\ &= 1 + k * 3 \end{aligned}$$

12. The recurrence relation matches equation (6) with $c = 1$ and $g(n) = 3$. From equation (8), the closed-form solution is

$$\begin{aligned} T(n) &= 1^{n-1}(1) + \sum_{i=2}^n 1^{n-i}(3) \\ &= 1 + \sum_{i=2}^n 3 \\ &= 1 + (n - 1)3 \end{aligned}$$

13. a. From the base cases and the recurrence relation, the first five terms of the sequence are

$$\begin{aligned} S(1) &= 3, S(2) = 1, S(3) = 2S(2) + 3S(1) = 11, \\ S(4) &= 2S(3) + 3S(2) = 25, S(5) = 2S(4) + 3S(3) = 83 \end{aligned}$$

- b. The formula $S(n) = 3^{n-1} + 2(-1)^{n-1}$ generates 3, 1, 11, 25, 83 for $n = 1, 2, 3, 4, 5$.

14. $c_1 = 6$ and $c_2 = -5$, so the characteristic equation is

$$t^2 - 6t + 5 = 0$$

which has roots $r_1 = 1, r_2 = 5$. The solution has the form

$$T(n) = p + q(5)^{n-1}$$

where

$$\begin{aligned} p + q &= 5 \\ p + q(5) &= 13 \end{aligned}$$

Solving this system of equations, $p = 3, q = 2$ and the solution formula is

$$T(n) = 3 + 2(5)^{n-1}$$

15. This is in the form of equation (16) with $c = 2$ and $g(n) = 1$. By equation (21), the solution is

$$\begin{aligned} 2^{\log n}(1) + \sum_{i=1}^{\log n} 2^{(\log n)-i}(1) &= 2^{\log n} + 2^{(\log n)-1} + 2^{(\log n)-2} + \dots + 2^0 \\ &= 2^{(\log n)+1} - 1 \\ &= (2)2^{\log n} - 1 = 2n - 1 \end{aligned}$$

16.

n	Sequential Search	Binary Search
64	64	7
1024	1024	11
32768	32768	16

CHAPTER 4

- $\{4, 5, 6, 7\}$
 - $\{\text{April, June, September, November}\}$
 - $\{\text{Washington, D.C.}\}$
- $\{x|x \text{ is one of the first four perfect squares}\}$
 - $\{x|x \text{ is one of the Three Men in a Tub in the children's nursery rhyme}\}$
 - $\{x|x \text{ is a prime number}\}$
- $A = \{x|x \in \mathbb{N} \text{ and } x \geq 5\}$
 - $B = \{3, 4, 5\}$
- $x \in B$
- $A \subset B$ means $(\forall x)(x \in A \rightarrow x \in B) \wedge (\exists y)[y \in B \wedge (y \in A)']$
- a, b, d, e, h, i, l

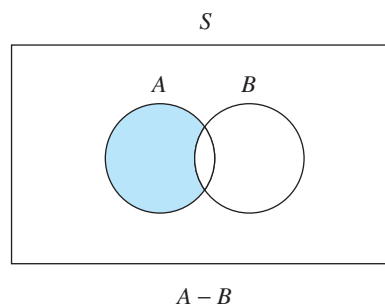
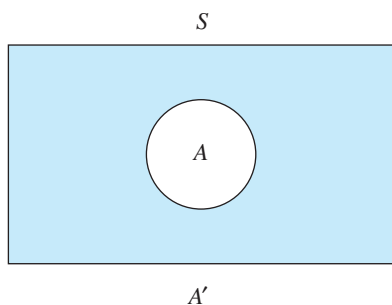
7. Let $x \in A$. Then $x \in \mathbb{R}$ and $x^2 - 4x + 3 = 0$, or $(x - 1)(x - 3) = 0$, which gives $x = 1$ or $x = 3$. In either case, $x \in \mathbb{N}$ and $1 \leq x \leq 4$, so $x \in B$. Therefore $A \subseteq B$. The value 4 belongs to B but not to A , so $A \subset B$.
8. $\wp(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$.
9. 2^n
10. By the definition of equality for ordered pairs,

$$2x - y = 7 \text{ and } x + y = -1$$

Solving the system of equations,

$$x = 2, y = -3$$

11. (3, 3), (3, 4), (4, 3), (4, 4)
12. a. S is not closed under division. ($3 \div 4$ is not a positive integer)
 c. 0^0 is not defined.
 f. $x^\#$ is not unique for, say, $x = 4$ ($2^2 = 4$ and $(-2)^2 = 4$).
13. Yes; if $x \in A \cap B$, then $x \in A$ (and $x \in B$, but we don't need this fact), so $x \in A \cup B$.
14. $A' = \{x \mid x \in S \text{ and } x \notin A\}$ 15. $A - B = \{x \mid x \in A \text{ and } x \notin B\}$



16. a. $\{1, 2, 3, 4, 5, 7, 8, 9, 10\}$
 b. $\{1, 2, 3\}$
 c. $\{1, 3, 5, 10\}$
17. a. $A \times B = \{(1, 3), (1, 4), (2, 3), (2, 4)\}$
 b. $B \times A = \{(3, 1), (3, 2), (4, 1), (4, 2)\}$
 c. $A^2 = \{(1, 1), (1, 2), (2, 1), (2, 2)\}$
 d. $A^3 = \{(1, 1, 1), (1, 1, 2), (1, 2, 1), (1, 2, 2), (2, 1, 1), (2, 1, 2), (2, 2, 1), (2, 2, 2)\}$
18. Show set inclusion in each direction. To show $A \cup \emptyset \subseteq A$, let $x \in A \cup \emptyset$. Then $x \in A$ or $x \in \emptyset$, but since \emptyset has no elements, $x \in A$. To show $A \subseteq A \cup \emptyset$, let $x \in A$. Then $x \in A$ or $x \in \emptyset$, so $x \in A \cup \emptyset$.
19. a. $[C \cap (A \cup B)] \cup [(A \cup B) \cap C']$
 $= [(A \cup B) \cap C] \cup [(A \cup B) \cap C']$ (1b)
 $= (A \cup B) \cap (C \cup C')$ (3b)
 $= (A \cup B) \cap S$ (5a)
 $= A \cup B$ (4b)
- b. $[C \cup (A \cap B)] \cap [(A \cap B) \cup C'] = A \cap B$

20. An enumeration of the even positive integers is 2, 4, 6, 8, 10, 12, ...

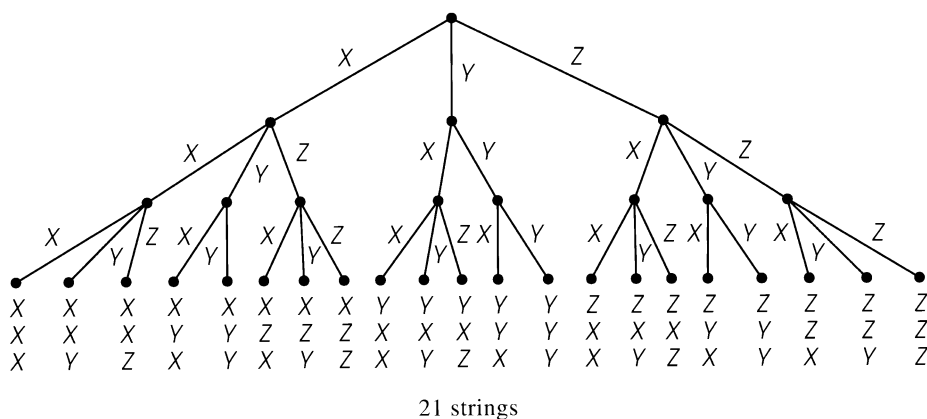
21. $1/5, 5/1$

22. $4(8)(5) = 160$

23. $7(5) + 9 = 44$

24. Although the problem consists of successive events—the five tosses—the number of outcomes of each event is not constant but varies between one and two depending on the outcome of the preceding event.

25.



26. $A \cup B$

27. Equation (2) gives the result of Example 31 because if A and B are disjoint, then $|A \cap B| = 0$.

28. The reasons for the equalities are
 set union is associative
 equation (2)
 equation (2) and set identity 3b (distributive property)
 equation (2)
 rearranging terms

29. 7 (The bins are the 6 possible values.)

32. $C(12, 3) = \frac{12!}{3!9!} = 220$

30. $P(20, 2) = \frac{20!}{18!} = 380$

33. a. 18 b. 24

31. $6! = 720$

34. $\frac{9!}{3!2!}$

35. Here $r = 6$ and $n = 3$, with repetitions. $C(r + n - 1, r) = C(8, 6) = \frac{8!}{6!2!} = 28$

36. cysar, scary, scrva, scyra, yarcs, yarsc

37. Here five digits are being permuted, so $n = 5$. At this point

$$d_1 = 5, d_2 = 1, d_3 = 4, d_4 = 3, d_5 = 2$$

In the for loop that generates all permutations after the first, set $i = 4, j = 5$. Consider pairs of adjacent values from right to left as long as $d_i > d_j$:

$$\begin{array}{lll} d_4 > d_5 & 3 > 2 & \text{true} \\ d_3 > d_4 & 4 > 3 & \text{true} \\ d_2 > d_3 & 1 > 4 & \text{false} \end{array}$$

The value of i at this point is 2, and $d_i = d_2 = 1$. Set $j = 5$; consider d_j values from right to left as long as $d_i > d_j$.

$$d_2 > d_5 \quad 1 > 2 \quad \text{false}$$

The value of j at this point is 5. Swap d_2 and d_5 , giving 52431. Take the descending sequence right of d_2 , namely 431, and swap pairs of values from the outside in to reverse the sequence. Swap 4 and 1, giving 52134. The indices i from the left and j from the right meet in the middle, at which point the reversal—and the new permutation—is complete.

- 38.** The rightmost non-max element is 5, which becomes incremented to 6. The two digits to its right are reset to their minimum values of 78. The next combination is therefore 24678.

39. $(a + b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$

Coefficients: 1 3 3 1, which is row $n = 3$ in Pascal's triangle

$$(a + b)^4 = a^4 + 4a^3b + 6a^2b^2 + 4ab^3 + b^4$$

Coefficients: 1 4 6 4 1, which is row $n = 4$ in Pascal's triangle

40. $(x + 1)^5 = C(5, 0)x^5 + C(5, 1)x^4 + C(5, 2)x^3 + C(5, 3)x^2 + C(5, 4)x + C(5, 5)$
 $= x^5 + 5x^4 + 10x^3 + 10x^2 + 5x + 1$

41. $C(7, 4)x^3y^4$

- 42.** The sample space consists of all the cards in the deck, so $|S| = 52$. The event of interest is the set of aces, so $|E| = 4$. $P(E) = 4/52 = 1/13$.

43. $16/42 = 8/21$; $28/42 = 14/21$

44. a. $p(c) = 1 - (p(a) + p(b)) = 1 - (0.2 + 0.3) = 0.5$

b. $p(a) + p(c) = 0.2 + 0.5 = 0.7$

45. $E_1 = \{HH, HT, TH\}$, $E_2 = \{HH\}$, $E_1 \cap E_2 = \{HH\}$

$$P(E_2|E_1) = \frac{P(E_1 \cap E_2)}{P(E_1)} = \frac{1/4}{3/4} = 1/3$$

- 46.** Let

E_1 be the event that the package came from Supplier A

E_2 be the event that the package came from Supplier B

F be the event that the package was lettuce

By Bayes' theorem,

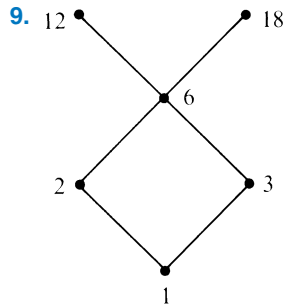
$$P(E_1|F) = \frac{P(F|E_1)P(E_1)}{P(F|E_1)P(E_1) + P(F|E_2)P(E_2)} = \frac{(57/100)(1/2)}{(57/100)(1/2) + (39/100)(1/2)} \cong 0.594$$

47. $E(X) = 5(2/8) + 2(3/8) + 3(2/8) + 7(1/8) = 29/8 = 3.625$

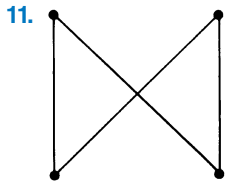
- 48.** This is still a Bernoulli trial where $n = 200$ and $p = 1/2$. From the binomial distribution, the probability of 100 heads is $C(200, 100)(1/2)^{100}(1/2)^{100} \cong 0.056$.

CHAPTER 5

1.
 - a. $(3, 2) \in \rho$
 - b. $(2, 4), (2, 6) \in \rho$
 - c. $(3, 4), (5, 6) \in \rho$
 - d. $(2, 1), (5, 2) \in \rho$
2.
 - a. Many-to-one
 - b. One-to-one
 - c. Many-to-many
3.
 - a. $x(\rho \cup \sigma)y \leftrightarrow x \leq y$
 - b. $x \rho' y \leftrightarrow x \neq y$
 - c. $x \sigma' y \leftrightarrow x \geq y$
 - d. $\rho \cap \sigma = \emptyset$
4.
 - a. $(1, 1), (2, 2), (3, 3)$
 - b. Knowing that a relation is symmetric does not by itself give information about any of the ordered pairs that might belong to ρ . If we know that a relation is symmetric and we know some ordered pairs that belong to the relation, then we know certain other pairs that must belong to the relation (see part (c)).
 - c. (b, a)
 - d. $a = b$
 - e. The transitive property says $(x, y) \in \rho \wedge (y, z) \in \rho \rightarrow (x, z) \in \rho$. In this case $(1, 2)$ is the only element of ρ and $(2, z) \notin \rho$ for any z in S . Therefore the antecedent of the implication is always false, and the implication is true; ρ is transitive.
5.
 - a. Reflexive, symmetric transitive
 - b. Reflexive, antisymmetric, transitive
 - c. Reflexive, symmetric, transitive
 - d. Antisymmetric
 - e. Reflexive, symmetric, antisymmetric, transitive
 - f. Antisymmetric (recall the truth table for implication), transitive
 - g. Reflexive, symmetric, transitive
 - h. Reflexive, symmetric, transitive
6. No. If the relation has the antisymmetry property, then it is its own antisymmetric closure. If the relation is not antisymmetric, there must be two ordered pairs (x, y) and (y, x) in the relation with $x \neq y$. Extending the relation by adding more ordered pairs will not change this situation, so no extension will be antisymmetric.
7. Reflexive closure: $\{(a, a), (b, b), (c, c), (a, c), (a, d), (b, d), (c, a), (d, a), (d, d)\}$
 Symmetric closure: $\{(a, a), (b, b), (c, c), (a, c), (a, d), (b, d), (c, a), (d, a), (d, b)\}$
 Transitive closure: $\{(a, a), (b, b), (c, c), (a, c), (a, d), (b, d), (c, a), (d, a), (d, d), (d, c), (b, a), (b, c), (c, d)\}$
8.
 - a. $(1, 1), (1, 2), (2, 2), (1, 3), (3, 3), (1, 6), (6, 6), (1, 12), (12, 12), (1, 18), (18, 18), (2, 6), (2, 12), (2, 18), (3, 6), (3, 12), (3, 18), (6, 12), (6, 18)$
 - b. 1, 2, 3
 - c. 2, 3



10. $y \in S$ is a greatest element if $x \leq y$ for all $x \in S$.
 $y \in S$ is a maximal element if there is no $x \in S$ with $y < x$.



12. Let $q \in [x]$. Then $x \rho q$. Because $x \rho z$, by symmetry $z \rho x$. By transitivity, $z \rho x$ together with $x \rho q$ gives $z \rho q$. Therefore, $q \in [z]$.

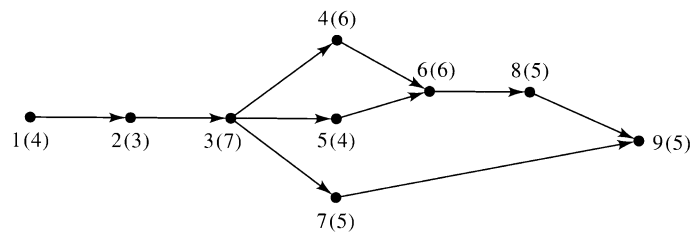
13. Reflexive: For any $x \in S$, x is in the same block as itself, so $x \rho x$.
 Symmetric: If $x \rho y$, then x is in the same block as y , so y is in the same block as x , or $y \rho x$ and ρ is symmetric.
 Transitive: If $x \rho y$ and $y \rho z$, then x is in the same block as y and y is in the same block as z , so x is in the same block as z , or $x \rho z$.

Therefore ρ is an equivalence relation.

14. a. The equivalence classes are sets consisting of lines in the plane with the same slope.
 b. $[n] = \{n\}$; the equivalence classes are all of the singleton sets of elements of \mathbb{N} .
 c. $[1] = [2] = \{1, 2\}$, $[3] = \{3\}$
15. $[0] = \{\dots, -15, -10, -5, 0, 5, 10, 15, \dots\}$
 $[1] = \{\dots, -14, -9, -4, 1, 6, 11, 16, \dots\}$
 $[2] = \{\dots, -13, -8, -3, 2, 7, 12, 17, \dots\}$
 $[3] = \{\dots, -12, -7, -2, 3, 8, 13, 18, \dots\}$
 $[4] = \{\dots, -11, -6, -1, 4, 9, 14, 19, \dots\}$

16. 2

17.



18. Minimum time to completion is 36 days. Critical path is 1, 2, 3, 4, 6, 8, 9.
 19. For example: 1, 3, 2, 6, 7, 5, 4, 8, 9, 10, 11, 12
 20. For example: 1, 2, 3, 7, 5, 4, 6, 8, 9

21.

Locale	
Name	State
Patrick, Tom	FL
Smith, Mary	IL
Collier, Jon	IL
Jones, Kate	OH
Smith, Bob	MA
White, Janet	GA
Garcia, Maria	NY

22. a. **project(join(restrict PetOwner where PetType = "Dog") and Person over Name) over City giving Result**

b. **SELECT City FROM Person, PetOwner WHERE Person.Name = PetOwner.Name AND PetType = "Dog"**

c. Range of x is Person
Range of y is PetOwner
{ x .City | **exists** y (y .Name = x .Name **and** y .PetType = "Dog")}

23. a. Not a function; $2 \in S$ has two values associated with it.

b. Function

c. Not a function; for values 0, 1, 2, 3 of the domain, the corresponding $h(x)$ values fall outside the codomain.

d. Not a function; not every member of S owns an automobile.

e. Function (not every value in the codomain need be used)

f. Function

g. Function

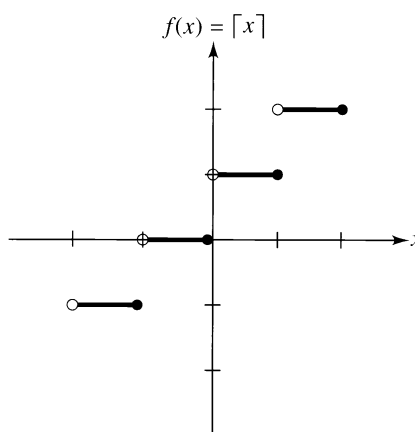
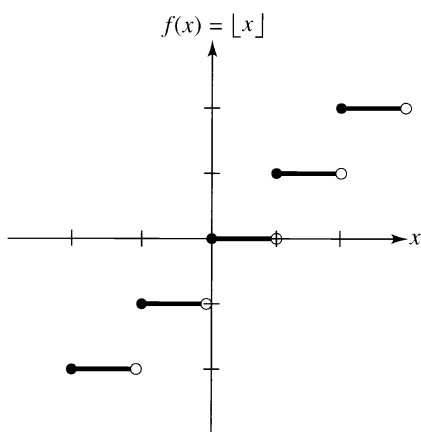
h. Not a function; $5 \in \mathbb{N}$ has two values associated with it.

24. a. 16

b. ± 3

25. T, F

26.



27. f and g have the same domain and codomain, so we must show that each function has the same effect on each member of the domain.

$$f(1) = 1 \quad g(1) = \frac{\sum_{k=1}^1 (4k - 2)}{2} = \frac{4 \cdot 1 - 2}{2} = \frac{2}{2} = 1$$

$$f(2) = 4 \quad g(2) = \frac{\sum_{k=1}^2 (4k - 2)}{2} = \frac{(4 \cdot 1 - 2) + (4 \cdot 2 - 2)}{2} = \frac{2 + 6}{2} = 4$$

$$f(3) = 9 \quad g(3) = \frac{\sum_{k=1}^3 (4k - 2)}{2} = \frac{(4 \cdot 1 - 2) + (4 \cdot 2 - 2) + (4 \cdot 3 - 2)}{2}$$

$$= \frac{2 + 6 + 10}{2} = 9$$

Therefore $f = g$.

28. b, f, g

29. If P is either a tautology or a contradiction.

30. e, g

31. $(g \circ f)(2.3) = g(f(2.3)) = g((2.3)^2) = g(5.29) = \lfloor 5.29 \rfloor = 5$

$$(f \circ g)(2.3) = f(g(2.3)) = f(\lfloor 2.3 \rfloor) = f(2) = 2^2 = 4$$

32. Let $(g \circ f)(s_1) = (g \circ f)(s_2)$. Then $g(f(s_1)) = g(f(s_2))$ and because g is one-to-one, $f(s_1) = f(s_2)$. Because f is one-to-one, $s_1 = s_2$.

33. Let $t \in T$. Then $(f \circ g)(t) = f(g(t)) = f(s) = t$.

34. $f^{-1}: \mathbb{R} \rightarrow \mathbb{R}, f^{-1}(x) = (x - 4)/3$

35. a. $(1, 4, 5) = (4, 5, 1) = (5, 1, 4)$

b. $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 4 & 2 & 5 & 3 \end{pmatrix}$

36. a. $g \circ f = (1, 3, 5, 2, 4) = (3, 5, 2, 4, 1) = \dots$

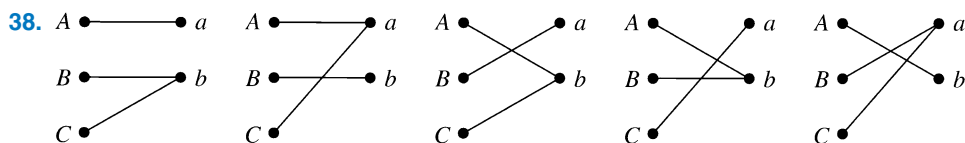
$$f \circ g = (1, 5, 2, 3, 4) = (5, 2, 3, 4, 1) = \dots$$

b. $g \circ f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 2 & 5 & 1 & 3 \end{pmatrix}$

$$f \circ g = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 3 & 5 & 4 \end{pmatrix}$$

c. $g \circ f = f \circ g = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 5 & 1 & 4 & 2 \end{pmatrix}$

37. $(1, 2, 4) \circ (3, 5)$ or $(3, 5) \circ (1, 2, 4)$



39. One possibility: $\{(0, 0), (1, 1), (-1, 2), (2, 3), (-2, 4), (3, 5), (-3, 6), \dots\}$

40. a. $10.87 \leq 12 \leq 1087$

$$22.27 \leq 27 \leq 2227$$

$$37.67 \leq 48 \leq 3767$$

$$57.07 \leq 75 \leq 5707$$

b. No

c. $n_0 = 1, c_1 = 1/200, c_2 = 1$

41. a. Let $f \rho g$. Then there are positive constants $n_0, c_1,$ and c_2 with $c_1 g(x) \leq f(x) \leq c_2 g(x)$ for $x \geq n_0$. Then for $x \geq n_0$, it is true that $(1/c_2)f(x) \leq g(x) \leq (1/c_1)f(x)$, so $g \rho f$.

b. Let $f \rho g$ and $g \rho h$. Then there are positive constants n_0, n_1, c_1, c_2, d_1 , and d_2 with $c_1 g(x) \leq f(x) \leq c_2 g(x)$ for $x \geq n_0$ and $d_1 h(x) \leq g(x) \leq d_2 h(x)$ for $x \geq n_1$. Then for $x \geq \max(n_0, n_1)$, $c_1 d_1 h(x) \leq f(x) \leq c_2 d_2 h(x)$ so $f \rho h$.

42. $3x^2 = \Theta(x^2)$ using constants $n_0 = 1, c_1 = c_2 = 3$.

$200x^2 + 140x + 7 = \Theta(x^2)$ using constants $n_0 = 2, c_1 = 1, c_2 = 300$.

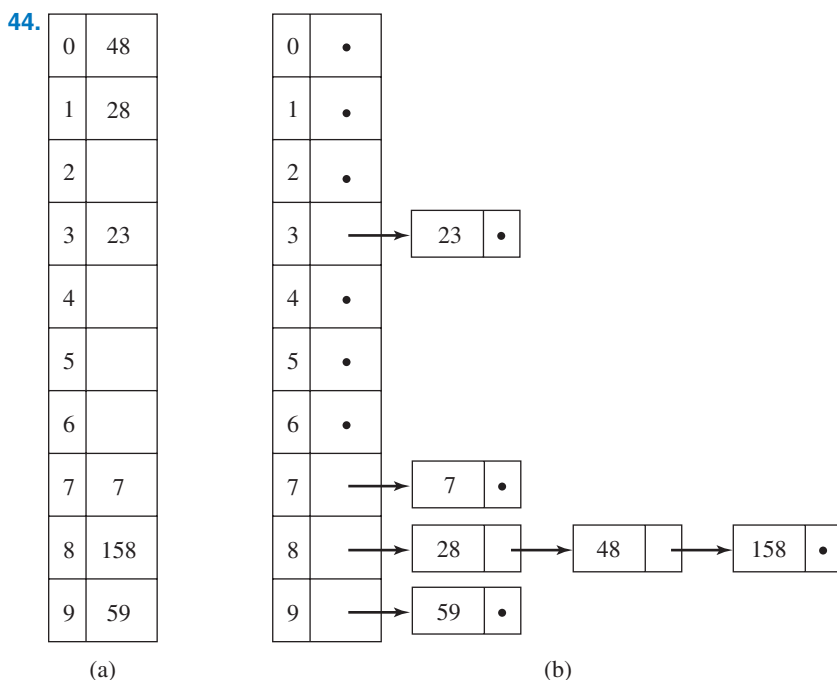
43. a. If $x \equiv y \pmod{n}$ then $x - y = kn$ for some integer k , so $x - y = kn + 0$ and $(x - y) \bmod n = 0$.

Conversely, if $(x - y) \bmod n = 0$ then $x - y = kn + 0$ for some integer k , or $x - y = kn$ so $x \equiv y \pmod{n}$.

b. Let $x = q_1 n + r_1, 0 \leq r_1 < n$ and $y = q_2 n + r_2, 0 \leq r_2 < n$, so $x \bmod n = r_1$ and $y \bmod n = r_2$. Then $x - y = (q_1 n + r_1) - (q_2 n + r_2) = (q_1 - q_2)n + (r_1 - r_2)$ with $-n < r_1 - r_2 < n$.

If $x \bmod n = y \bmod n$, then $r_1 = r_2$ so $r_1 - r_2 = 0$ and $x - y = (q_1 - q_2)n$ where $q_1 - q_2$ is an integer, so $x \equiv y \pmod{n}$.

Conversely, if $x \equiv y \pmod{n}$, then $x - y = kn$ for some integer k . Because $x - y = (q_1 - q_2)n + (r_1 - r_2)$ with $-n < r_1 - r_2 < n, r_1 - r_2 = 0$ and $x \bmod n = y \bmod n$.



45. THE CAT IN THE HAT

46. $x = 1011, p = x \bmod 2^3 = 0011, q = p \cdot 2 = 0110, s = x \oplus p = 1000, t = s \cdot 2^{-3} = 0001, y = q + t = 0111$

47. $(166)^{35} \bmod 221 = (166^2)^{17} \cdot 166 \bmod 221 = (27556)^{17} \cdot 166 \bmod 221$
 $= (152)^{17} \cdot 166 \bmod 221 = (152^2)^8 \cdot 152 \cdot 166 \bmod 221 = (23104)^8 \cdot 152 \cdot 166 \bmod 221$
 $= (120)^8 \cdot 152 \cdot 166 \bmod 221 = (120^2)^4 \cdot 152 \cdot 166 \bmod 221$
 $= (14400)^4 \cdot 152 \cdot 166 \bmod 221 = (35)^4 \cdot 152 \cdot 166 \bmod 221$
 $= 35^2 \cdot 35^2 \cdot 152 \cdot 166 \bmod 221 = 120 \cdot 120 \cdot 152 \cdot 166 \bmod 221$
 $= 14400 \cdot 25232 \bmod 221 = 35 \cdot 38 \bmod 221 = 4$

48. a. 3

b. X

49. temp = 375
 ones = temp mod 10 = 5
 temp = (375 - 5)/10 = 370/10 = 37
 tens = temp mod 10 = 37 mod 10 = 7
 temp = (37 - 7)/10 = 30/10 = 3
 hundreds = temp = 3

50. $a_{23} = 1, a_{24} = -7, a_{13} = -6$

51.
$$2\mathbf{A} + \mathbf{B} = \begin{bmatrix} 6 & 14 \\ 3 & 10 \\ 9 & 16 \end{bmatrix}$$

52.
$$\mathbf{A} \cdot \mathbf{B} = \begin{bmatrix} 15 & 22 \\ 12 & 28 \end{bmatrix}$$

$$\mathbf{B} \cdot \mathbf{A} = \begin{bmatrix} 39 & 0 \\ 27 & 4 \end{bmatrix}$$

53.
$$\mathbf{I} \cdot \mathbf{A} = \begin{bmatrix} 1(a_{11}) + 0(a_{21}) & 1(a_{12}) + 0(a_{22}) \\ 0(a_{11}) + 1(a_{21}) & 0(a_{12}) + 1(a_{22}) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \mathbf{A}$$

Similarly, $\mathbf{A} \cdot \mathbf{I} = \mathbf{A}$.

54.
$$\mathbf{A} \cdot \mathbf{B} = \begin{bmatrix} -1 & 2 & -3 \\ 2 & 1 & 0 \\ 4 & -2 & 5 \end{bmatrix} \begin{bmatrix} -5 & 4 & -3 \\ 10 & -7 & 6 \\ 8 & -6 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{B} \cdot \mathbf{A} = \begin{bmatrix} -5 & 4 & -3 \\ 10 & -7 & 6 \\ 8 & -6 & 5 \end{bmatrix} \begin{bmatrix} -1 & 2 & -3 \\ 2 & 1 & 0 \\ 4 & -2 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

55. The augmented matrix is

$$\begin{bmatrix} 3 & -5 & 5 \\ 7 & 1 & 37 \end{bmatrix}$$

Multiply row 1 by 1/3:

$$\frac{1}{3} \begin{bmatrix} 3 & -5 & 5 \\ 7 & 1 & 37 \end{bmatrix}$$

Then multiply row 1 by -7 and add it to row 2:

$$-7 \begin{matrix} \curvearrowright \\ \curvearrowleft \end{matrix} \begin{bmatrix} 1 & -5/3 & 5/3 \\ 7 & 1 & 37 \end{bmatrix} \text{ resulting in } \begin{bmatrix} 1 & -5/3 & 5/3 \\ 38/3 & 76/3 & 76/3 \end{bmatrix}$$

Using the second row,

$$(38/3)y = 76/3 \text{ or } y = 2$$

Using the first row,

$$x - (5/3)y = 5/3 \text{ or } x - (5/3)(2) = 5/3 \text{ or } x = 5.$$

The solution is $x = 5, y = 2$.

56.

x	y	$x \wedge y$
1	1	1
1	0	0
0	1	0
0	0	0

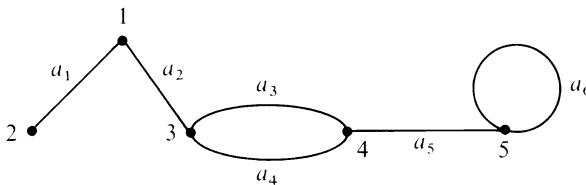
x	y	$x \vee y$
1	1	1
1	0	1
0	1	1
0	0	0

57. No, $\mathbf{A} \cdot \mathbf{B} = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

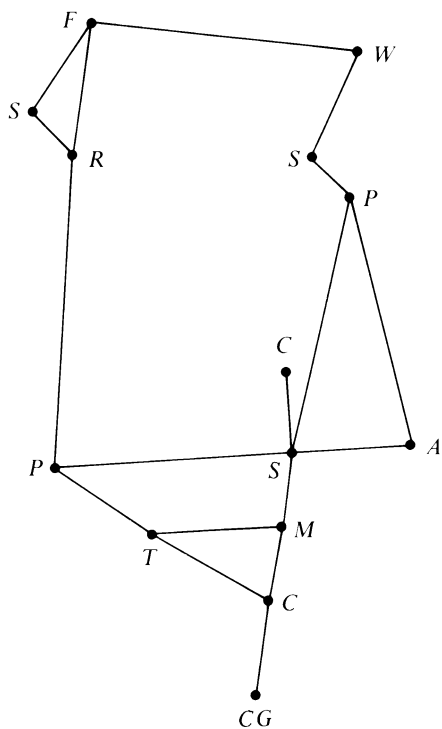
58. $\mathbf{B} \times \mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

CHAPTER 6

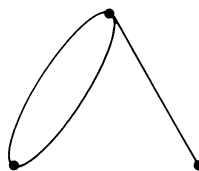
1. One possible picture:



2. a.



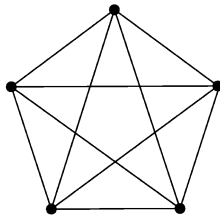
b.



3. Possible answers:

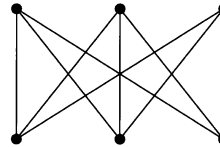
- | | |
|--------------------|---|
| a. 2 and 3 | f. 2, a_1 , 1, a_2 , 3, a_3 , 4, a_4 , 3, a_3 , 4 |
| b. 5 | g. 3, a_3 , 4, a_4 , 3 |
| c. a_6 | h. no |
| d. a_3 and a_4 | i. yes |
| e. 3 | |

4.



K_5

5.



$K_{3,3}$

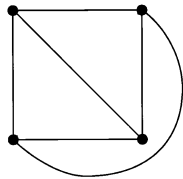
6. a. In a complete graph, any two distinct nodes are adjacent, so there is a path of length 1 from any node to any other node; hence the graph is connected.
 b. For example, the graph of Figure 6.10b.

7. f_2 : $a_4 \rightarrow e_3$
 $a_5 \rightarrow e_8$
 $a_6 \rightarrow e_7$
 $a_7 \rightarrow e_5$ (or e_6)
 $a_8 \rightarrow e_6$ (or e_5)

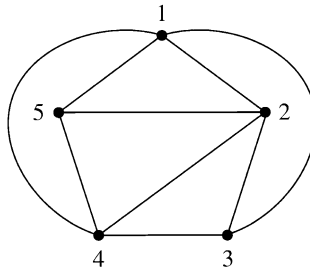
8. f : $1 \rightarrow d$
 $2 \rightarrow e$
 $3 \rightarrow f$
 $4 \rightarrow c$
 $5 \rightarrow b$
 $6 \rightarrow a$

9. The graph on the left in Figure 6.19 has two nodes of degree 2, but the graph on the right does not; or the graph on the left has parallel arcs, but the graph on the right does not.

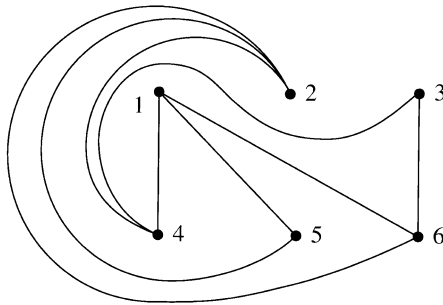
10. K_4 can be represented as



11. Making 1–3 and 1–4 exterior arcs leads to the graph below, where it is still impossible to make 3 and 5 adjacent while preserving planarity.

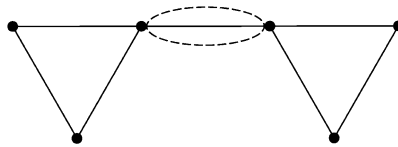


12. An attempt to construct $K_{3,3}$ as a planar graph leads to the graph below; there is no way to connect nodes 3 and 5. Any other construction leads to a similar difficulty.



13. $n = 6$, $a = 7$, $r = 3$, and $6 - 7 + 3 = 2$

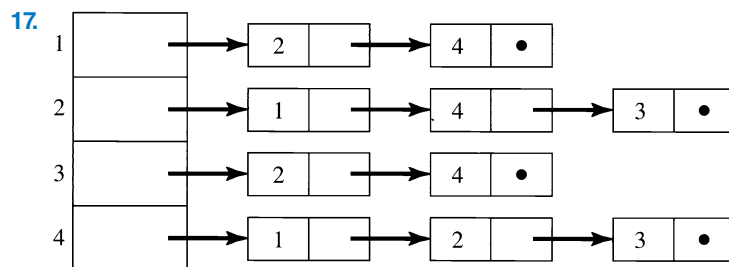
14. Without this condition on the arc, a figure such as the one below could result. Then the graph would be split into two disconnected subgraphs and the inductive hypothesis would not apply. Also the number of regions would not change.



15. In $K_{3,3}$, $a = 9$, $n = 6$, and $9 \leq 3(6) - 6$.

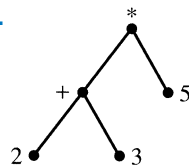
16.

$$A = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \\ 1 & 0 & 2 & 0 \end{bmatrix}$$



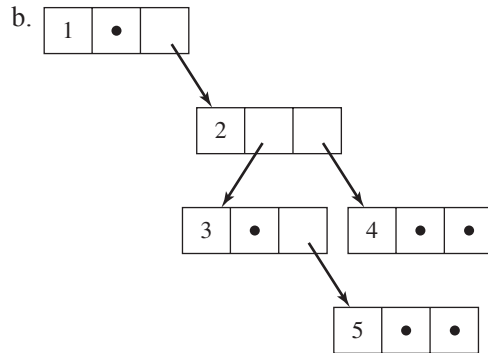
18. a. 2 b. 4 c. 2

19.



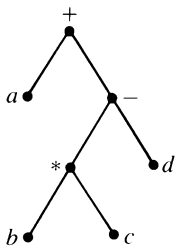
20. a.

	Left child	Right child
1	0	2
2	3	4
3	0	5
4	0	0
5	0	0



21. $a, b, e, f, c, d, g, i, h$
 $e, b, f, a, c, i, g, d, h$
 $e, f, b, c, i, g, h, d, a$

22.



Prefix notation: $+ a - * b c d$

Postfix notation: $a b c * d - +$

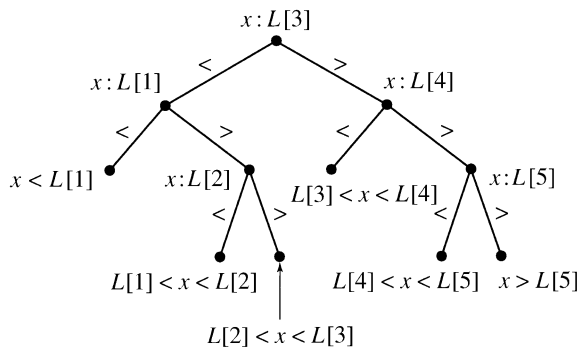
23. For the base case, $n = 1$, the tree consists of a single node and no arcs, therefore no arc ends. The number of arc ends is $0 = 2(1) - 2$. Assume that any tree with k nodes has a total number of arc ends of $2k - 2$. Consider a tree with $k + 1$ nodes, and show that the number of arc ends is $2(k + 1) - 2$. In this tree, remove a leaf node and the arc to that node's parent. This leaves a tree with k nodes and, by the inductive hypothesis, $2k - 2$ arc ends. The original graph had one more arc, and two more arc ends, so it had $2k - 2 + 2 = 2k = 2(k + 1) - 2$ arc ends.
24. The base case is the same as in Practice 23. Assume that tree T is constructed from subtrees T_1, \dots, T_t and that any subtree T_i with n_i nodes has $(2n_i - 2)$ arc ends. Let n equal the number of nodes in T . Then, as in Example 30,

$$n = 1 + \sum_{i=1}^t n_i \text{ so } 2n = 2 + 2 \sum_{i=1}^t n_i$$

The number N of arc ends in T is $2t + \sum_{i=1}^t$ (number of arc ends in T_i). (The extra $2t$ counts the number of arc ends contributed by the t arcs from the root of T to the t subtrees.) Then

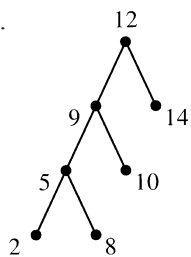
$$N = 2t + \sum_{i=1}^t (2n_i - 2) = 2t + 2 \sum_{i=1}^t n_i - 2t = 2 \sum_{i=1}^t n_i = 2n - 2$$

25. a.



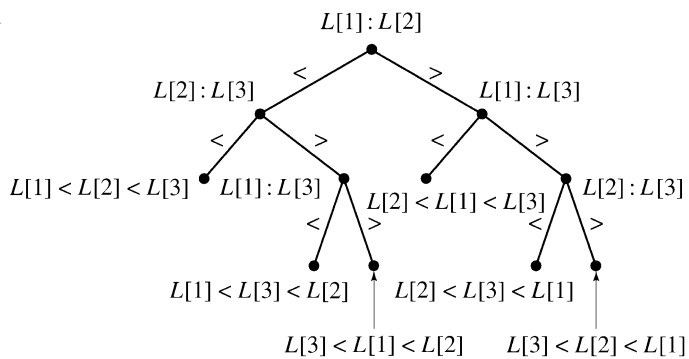
b. Depth of tree = 3 = 1 + $\lceil \log 5 \rceil$

26. a.



b. $d = 3$

27. a.

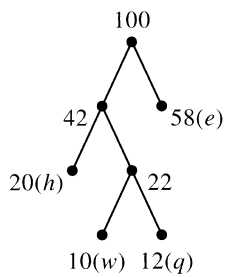


28. a. *ppca?*

b. *cagak*

c. *?kac?*

29.



30. w : 010

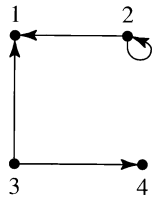
q : 011

h : 00

e : 1

CHAPTER 7

- 1.
- $\{(2, 1), (2, 2), (3, 1), (3, 4)\}$



2. There are two distinct nodes, 3 and 4, with
- $3 \rho 4$
- and
- $4 \rho 3$
- .

3.
$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{A}^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$\mathbf{A}^{(2)}[2, 1] = 1$ because there is a path from 2 to 1 of length 2 (2–4–1).

4. There is a length-4 path (2–4–1–4–1) from 2 to 1, so
- $\mathbf{A}^{(4)}[2, 1]$
- should be 1.

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{A}^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A}^{(3)} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{A}^{(4)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 5.
- $\mathbf{R} = \mathbf{A} \vee \mathbf{A}^{(2)} \vee \mathbf{A}^{(3)} \vee \mathbf{A}^{(4)}$
- , so performing the Boolean
- or**
- of the four matrices from Practice 4 gives

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

Column 2 is all 0s, so 2 is not reachable from any node.

6.
$$\mathbf{M}_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{M}_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{M}_2 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M}_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{M}_4 = \mathbf{R} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

7. a. No

- b. Yes

8. a. No, four odd nodes b. Yes, no odd nodes

9. No, all four nodes are odd nodes.

10.
$$\begin{matrix} A & B & C & D \\ A & \begin{bmatrix} 0 & 2 & 2 & 1 \end{bmatrix} \\ B & \begin{bmatrix} 2 & 0 & 0 & 1 \end{bmatrix} \\ C & \begin{bmatrix} 2 & 0 & 0 & 1 \end{bmatrix} \\ D & \begin{bmatrix} 1 & 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

After row *C*, *total* = 3, the loop terminates, and there is no path.

11. a. No b. Yes

12.

$$IN = \{x\}$$

	<i>x</i>	1	2	3	<i>y</i>
<i>d</i>	0	1	∞	4	∞
<i>s</i>	–	<i>x</i>	<i>x</i>	<i>x</i>	<i>x</i>

$$p = 1$$

$$IN = \{x, 1\}$$

	<i>x</i>	1	2	3	<i>y</i>
<i>d</i>	0	1	4	2	6
<i>s</i>	–	<i>x</i>	1	1	1

$$p = 3$$

$$IN = \{x, 1, 3\}$$

	<i>x</i>	1	2	3	<i>y</i>
<i>d</i>	0	1	4	2	5
<i>s</i>	–	<i>x</i>	1	1	3

$$p = 2$$

$$IN = \{x, 1, 3, 2\}$$

	<i>x</i>	1	2	3	<i>y</i>
<i>d</i>	0	1	4	2	5
<i>s</i>	–	<i>x</i>	1	1	3

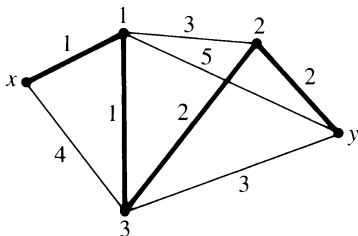
$$p = y$$

$$IN = \{x, 1, 3, 2, y\}$$

	<i>x</i>	1	2	3	<i>y</i>
<i>d</i>	0	1	4	2	5
<i>s</i>	–	<i>x</i>	1	1	3

Path: *x*, 1, 3, *y* Distance = 5

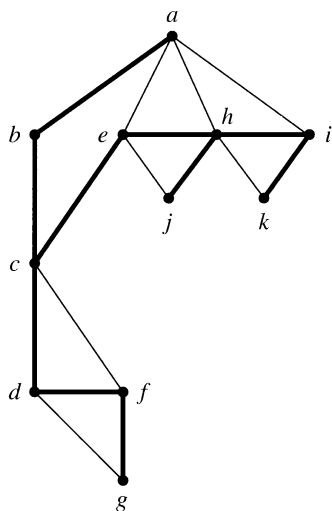
13.



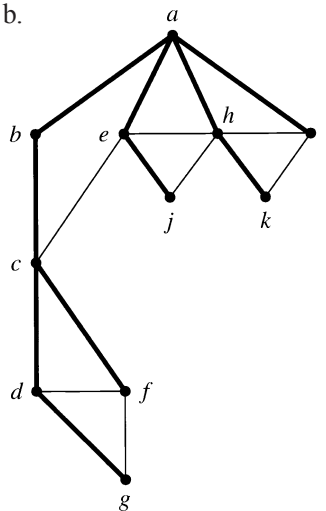
14. *a*, *e*, *d*, *b*, *c*, *i*, *f*, *g*, *h*, *l*, *k*, *m*, *j*

15. *a*, *e*, *f*, *d*, *i*, *b*, *c*, *g*, *h*, *j*, *k*, *m*, *l*

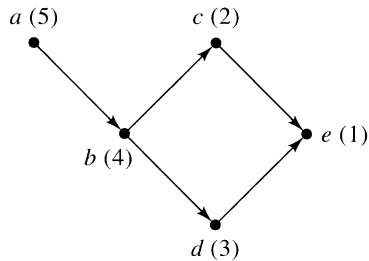
16. a.



b.



17. Choosing node c (arbitrarily) as the start node, we visit e , then have to back up to c . There's nowhere else to go, so start over from node a (again, an arbitrary choice from the unvisited nodes). Traveling from a to b to d and then backing up produces the rest of the node numbers.



Topological sort: a, b, d, c, e

18. The depth-first search progresses from node a to b and then c , with *TreeNumbers* and *BackNumbers* assigned in sequence. At node c , the back arc to node a causes *BackNumber* of c to be changed to *TreeNumber* of a . The search progresses to d , which has a sequential *TreeNumber* and *BackNumber* assigned. The search then backs up to node c . Because $BackNumber(d) > TreeNumber(c)$, c is recognized as an articulation point. The search backs up to node b , and b 's *BackNumber* is reduced to that of c . The search backs up to a , but a is not an articulation point because it is the starting node with only one tree arc.

CHAPTER 8

1. $0 \cdot 1 = 0$
 $1 \cdot 1 = 1$
2. a. $A \vee A = A$
 b. $A \cup A = A$
3. a. $x + 1 = x + (x + x')$ (5a, complement property)
 $= (x + x) + x'$ (2a, associative property)
 $= x + x'$ (idempotent property)
 $= 1$ (5a, complement property)
- b. $x \cdot 0 = 0$
4. To prove that $0' = 1$, show that 1 has the two properties of the complement of 0.

$$0 + 1 = 1 \quad (\text{universal bound})$$

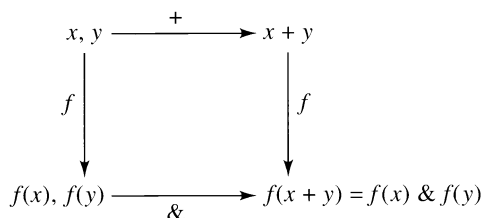
$$0 \cdot 1 = 1 \cdot 0 \quad (1b)$$

$$= 0 \quad (\text{dual of universal bound})$$

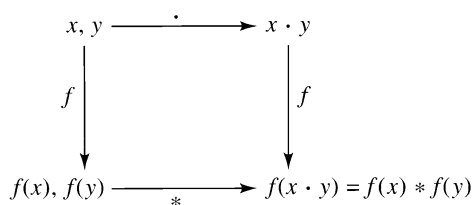
Therefore $1 = 0'$ by the theorem on the uniqueness of complements.

5. a. $f(x \cdot y) = f(x) * f(y)$
 b. $f(x') = [f(x)]''$

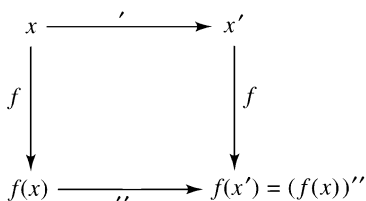
6. Property 2:



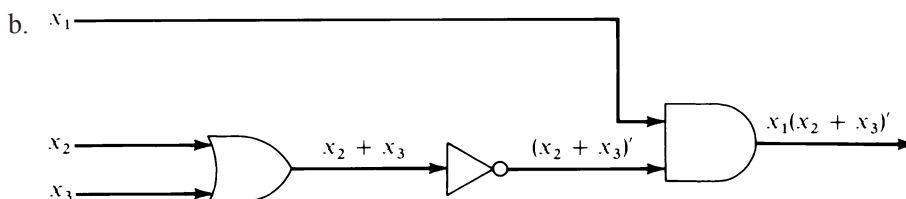
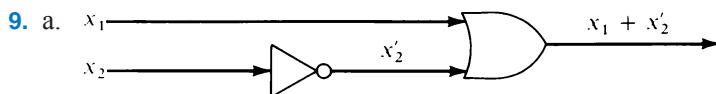
Property 3:



Property 4:

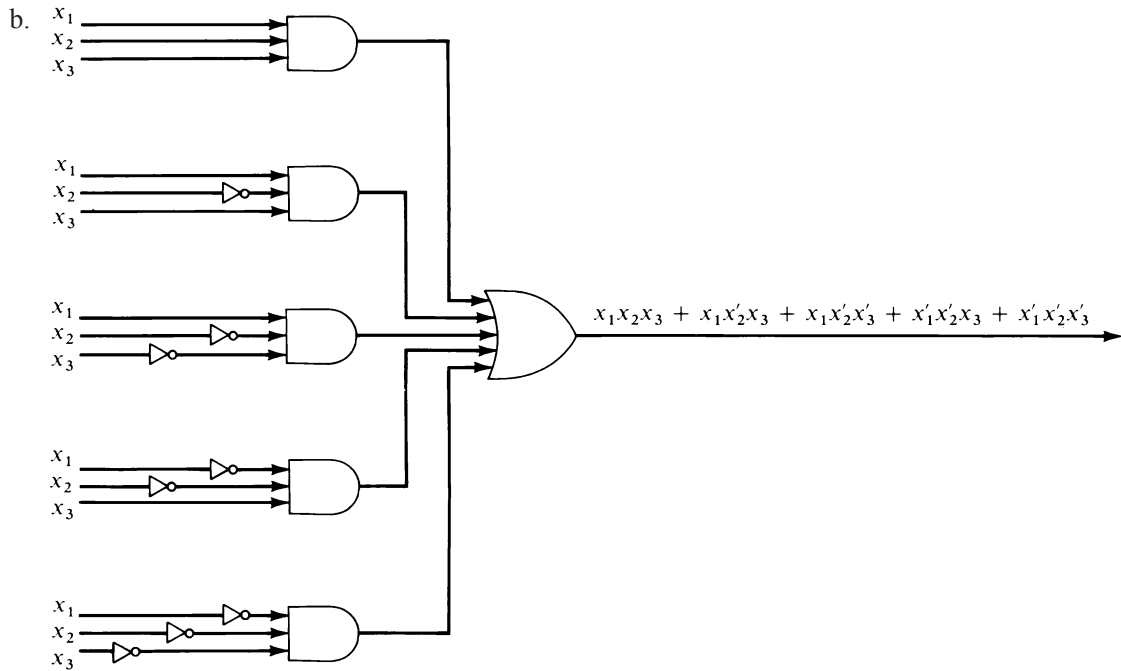


7. a. $f(0 + a) = f(a) = \{1\} = \emptyset \cup \{1\} = f(0) \cup f(a)$
 b. $f(a + a') = f(1) = \{1, 2\} = \{1\} \cup \{2\} = f(a) \cup f(a')$
 c. $f(a \cdot a') = f(0) = \emptyset = \{1\} \cap \{2\} = f(a) \cap f(a')$
 d. $f(1') = f(0) = \emptyset = \{1, 2\}' = (f(1))'$
8. a. A single truth function on $\{0, 1\}^n$ must map each of the elements in the domain to a 0 or a 1, and there are $2^n n$ -tuples in the domain $\{0, 1\}^n$. Hence the table for the function will have 2^n rows.
 b. Any truth function must fill 4 “slots” (corresponding to the $2^2 = 4$ domain elements) with one of 2 values, a 0 or a 1. There are $2^4 = 16$ different ways to do this.
 c. Any truth function must fill 2^n “slots” (corresponding to the 2^n domain elements) with one of 2 values, a 0 or a 1. There are 2^{2^n} different ways to do this.

10. a. $(x_1' + x_2)x_3'$

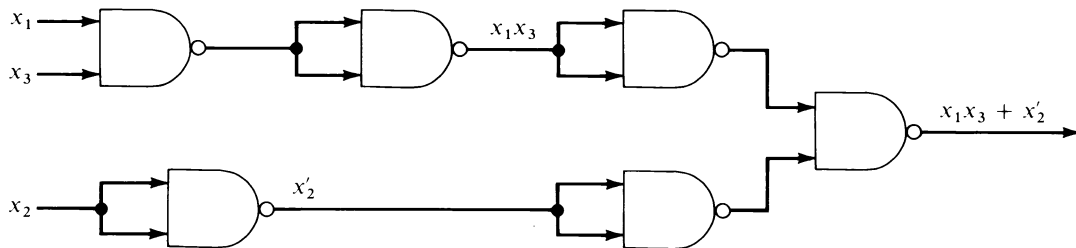
x_1	x_2	x_3	$(x_1' + x_2)x_3'$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	0
0	1	0	1
0	0	1	0
0	0	0	1

11. a. $x_1x_2x_3 + x_1x_2'x_3 + x_1x_2'x_3' + x_1'x_2'x_3 + x_1'x_2'x_3'$

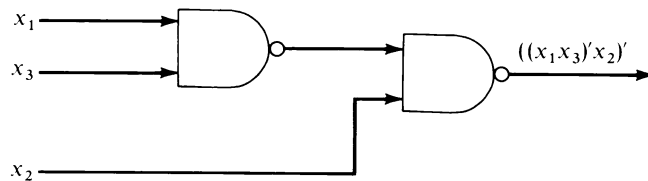


12.
$$\begin{array}{r} 101 \\ 111 \\ (1)100 \end{array}$$

13. a.



b. $x_1x_3 + x_2' = ((x_1x_3)'x_2)'$

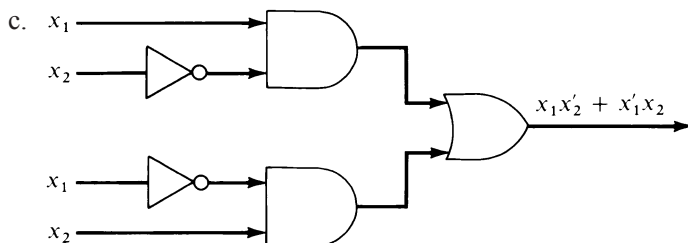


14. a

x_1	x_2	$f(x_1, x_2)$
1	1	0
1	0	1
0	1	1
0	0	0

Here $x_i = 0$ when the switch is in the “off” position and $f(x_1, x_2) = 0$ when the light is off. The last row of the truth table says that when both switches are off, the light is off. Rows 2 and 3 say that when one or the other switch is turned on, the light goes on. But the top row says that if the light is on (because one of the switches has been turned on) then flipping the second switch turns the light off.

b. One possibility is the canonical sum-of-products form, $x_1x_2' + x_1'x_2$



15. a. $x_1x_2 + x_1'x_2 = x_2x_1 + x_2x_1'$
 $= x_2(x_1 + x_1')$
 $= x_2 \cdot 1$
 $= x_2$

b. $x_1 + x_1'x_2 = x_1 \cdot 1 + x_1'x_2$
 $= x_1(1 + x_2) + x_1'x_2$ (universal bound)
 $= x_1 + x_1x_2 + x_1'x_2$
 $= x_1 + x_2(x_1 + x_1')$
 $= x_1 + x_2 \cdot 1$
 $= x_1 + x_2$

16.

	x_1	x_1'
x_2		1
x_2'		1

The reduced expression is x_1'

17. x_1x_3 (4 squares) and $x_1'x_2x_3'$ (2 squares)

18. $x_1x_2'x_4 + x_1x_3'x_4 + x_2'x_3'$

	x_1x_2	x_1x_2'	$x_1'x_2'$	$x_1'x_2$
x_3x_4		1		
x_3x_4'				
$x_3'x_4'$		1	1	
$x_3'x_4$	1	1	1	

19. The reduction table follows.

Number of 1s	x_1	x_2	x_3	
Three	1	1	1	1
Two	1	1	0	1,2
One	1	0	0	2,3
	0	0	1	4
None	0	0	0	3,4

Number of 1s	x_1	x_2	x_3
Two	1	1	–
One	1	–	0
None	–	0	0
	0	0	–

The comparison table follows.

	111	110	100	001	000
11-	✓	✓			
1-0		✓	✓		
-00			✓		✓
00-				✓	✓

Essential terms are 11- and 00-. Either 1-0 or -00 can be used as the third reduced term. The minimal sum-of-products form is

$$x_1x_2 + x_1'x_2' + x_1x_3' \quad \text{or} \quad x_1x_2 + x_1'x_2' + x_2'x_3'$$

CHAPTER 9

1. Multiplication in \mathbb{R} is associative and commutative, and 1 is an identity. But $[\mathbb{R}, \cdot]$ is not a commutative group because $0 \in \mathbb{R}$ does not have an inverse with respect to multiplication; there is no real number y such that $0 \cdot y = y \cdot 0 = 1$.
2. See Practice 52 of Chapter 5.
3. Many elements of $M_2(\mathbb{Z})$ do not have inverses under matrix multiplication. For example, if

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

is to have an inverse

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

under multiplication, then

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

By the definition of matrix multiplication, the only matrix that satisfies this equation would have $2a = 1$ or $a = \frac{1}{2}$ and thus would not be a member of $M_2(\mathbb{Z})$.

4. Subtraction is not associative; for example, $5 - (3 - 1) = 3$ but $(5 - 3) - 1 = 1$.
5. No, S is not closed under multiplication; for example, $\frac{2}{3} \cdot \frac{3}{2} = 1$.
6. All except $[\mathbb{R}^+, +]$ (which has no identity) are monoids; the identities are, respectively, 0, 1, 1, 0, \emptyset , S .
7. $[\mathbb{R}, +]$
8. a. $f(x) + g(x) = g(x) + f(x)$
 $[f(x) + g(x)] + h(x) = f(x) + [g(x) + h(x)]$
 b. the zero polynomial, 0
 c. $-7x^4 + 2x^3 - 4$

9. a.

$+_5$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

\cdot_5	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

b. 0; 1
c. 3
d. all except 0

10. a.

\cdot_6	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	1	2	3	4	5
2	0	2	4	0	2	4
3	0	3	0	3	0	3
4	0	4	2	0	4	2
5	0	5	4	3	2	1

b. 1 and 5

11. a. $01101 +_2 11011 = 10110$

b. $-10100 = 10100$ because $10100 +_2 10100 = 00000$

12. Let $f, g, h \in S$. Then for any $x \in A$, $[(f \circ g) \circ h](x) = (f \circ g)(h(x)) = f(g[h(x)])$ and $[f \circ (g \circ h)](x) = f[(g \circ h)(x)] = f(g[h(x)])$. Hence, $(f \circ g) \circ h = f \circ (g \circ h)$.

13. a.

\circ	α_1	α_2	α_3	α_4	α_5	α_6
α_1	α_1	α_2	α_3	α_4	α_5	α_6
α_2	α_2	α_1	α_6	α_5	α_4	α_3
α_3	α_3	α_5	α_1	α_6	α_2	α_4
α_4	α_4	α_6	α_5	α_1	α_3	α_2
α_5	α_5	α_3	α_4	α_2	α_6	α_1
α_6	α_6	α_4	α_2	α_3	α_1	α_5

b. No, because $\alpha_2 \circ \alpha_3 = \alpha_6$ but $\alpha_3 \circ \alpha_2 = \alpha_5$.

14. a. No, because $ab \cdot a = aba$ but $a \cdot ab = aab$.

b. No, because no nonempty string has an inverse; for example, there is no string to concatenate with a to get λ .

15. $i_1 = i_1 \cdot i_2$ because i_2 is an identity $i_1 \cdot i_2 = i_2$ because i_1 is an identity

16. Let y and z both be inverses of x . Let i be the identity. Then $y = y \cdot i = y \cdot (x \cdot z) = (y \cdot x) \cdot z = i \cdot z = z$.

17. $7^{-1} = 5, 3^{-1} = 9$; so $10^{-1} = (7 +_{12} 3)^{-1} = 3^{-1} +_{12} 7^{-1} = 9 +_{12} 5 = 2$

18. $z \cdot x = z \cdot y$ implies

$$z^{-1} \cdot (z \cdot x) = z^{-1} \cdot (z \cdot y)$$

$$(z^{-1} \cdot z) \cdot x = (z^{-1} \cdot z) \cdot y$$

$$i \cdot x = i \cdot y$$

$$x = y$$

20.

$*$	1	a	b	c	d
1	1	a	b	c	d
a	a	b	c	d	1
b	b	c	d	1	a
c	c	d	1	a	b
d	d	1	a	b	c

19. $x = 1 +_8(3)^{-1} = 1 +_8 5 = 6$

21. a. $[\mathbb{Z}_{18}, +_{18}]$

b. $[S_3, \circ]$

22. Requirement 2, $i \in A$, ensures that $A \neq \emptyset$.

23. a. Closure holds:

$+_8$	0	2	4	6
0	0	2	4	6
2	2	4	6	0
4	4	6	0	2
6	6	0	2	4

$0 \in \{0, 2, 4, 6\}$; $0^{-1} = 0$, $4^{-1} = 4$, and 2 and 6 are inverses of each other.

b. Closure holds:

$+_7$	1	2	4
1	1	2	4
2	2	4	1
4	4	1	2

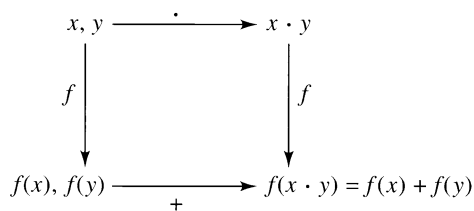
$1 \in \{1, 2, 4\}$; $1^{-1} = 1$, and 2 and 4 are inverses of each other.

24. $[\{\alpha_1, \alpha_5, \alpha_6\}, \circ] [\{\alpha_1, \alpha_2\}, \circ] [\{\alpha_1, \alpha_3\}, \circ] [\{\alpha_1, \alpha_4\}, \circ]$

25. To show that f is one-to-one, let α and β belong to A_n and suppose $f(\alpha) = f(\beta)$. Then $\alpha \circ (1, 2) = \beta \circ (1, 2)$. By the cancellation law available in the group S_n , $\alpha = \beta$. To show that f is onto, let $\gamma \in O_n$. Then $\gamma \circ (1, 2) \in A_n$ and $f(\gamma \circ (1, 2)) = \gamma \circ (1, 2) \circ (1, 2) = \gamma \circ i = \gamma$.

26. For $nz_1, nz_2 \in n\mathbb{Z}$, $nz_1 + nz_2 = n(z_1 + z_2) \in n\mathbb{Z}$, so closure holds; $0 = n \cdot 0 \in n\mathbb{Z}$; for $nz \in n\mathbb{Z}$, $-nz = n(-z) \in n\mathbb{Z}$.

27.



28. $f(s) + f(s^{-1}) = f(s \cdot s^{-1}) = f(i_S) = i_T$. Similarly, $f(s^{-1}) + f(s) = i_T$. Therefore $f(s^{-1})$ acts like the inverse of $f(s)$ in T , and since inverses are unique, $f(s^{-1}) = -f(s)$.

29. Let t_1 and t_2 be members of T . Because f is onto, $t_1 = f(s_1)$ and $t_2 = f(s_2)$ for some $s_1, s_2 \in S$. Then

$$\begin{aligned}
 t_1 + t_2 &= f(s_1) + f(s_2) = f(s_1 \cdot s_2) = f(s_2 \cdot s_1) && \text{(because } [S, \cdot] \text{ is commutative)} \\
 &= f(s_2) + f(s_1) = t_2 + t_1
 \end{aligned}$$

so $[T, +]$ is commutative.

30. Clearly f is onto. f is also on-to-one: Let $f(x) = f(y)$. Then $5x = 5y$ and $x = y$. f is a homomorphism: For $x, y \in \mathbb{Z}$, $f(x + y) = 5(x + y) = 5x + 5y = f(x) + f(y)$.

31. a. Composition of bijections is a bijection, and for $x, y \in S$, $(g \circ f)(x \cdot y) = g(f(x \cdot y)) = g(f(x) + f(y)) = g(f(x)) * g(f(y)) = (g \circ f)(x) * (g \circ f)(y)$

b. $S \cong S$ by the identity mapping. If f is an isomorphism from S to T , then f^{-1} is an isomorphism from T to S . If $S \cong T$ and $T \cong V$, then by part (a), $S \cong V$.

- 32.** To show that α_g is an onto function, let $y \in G$. Then $g^{-1} \cdot y$ belongs to G and $\alpha_g(g^{-1} \cdot y) = g(g^{-1} \cdot y) = (g \cdot g^{-1})y = y$. To show that α_g is one-to-one, let $\alpha_g(x) = \alpha_g(y)$. Then $g \cdot x = g \cdot y$, and by cancellation, $x = y$.
- 33.** a. For $\alpha_g \in P$, $\alpha_g \circ \alpha_1 = \alpha_{g \cdot 1} = \alpha_g$ and $\alpha_1 \circ \alpha_g = \alpha_{1 \cdot g} = \alpha_g$
 b. $\alpha_g \circ \alpha_{g^{-1}} = \alpha_{g \cdot g^{-1}} = \alpha_1$ and $\alpha_{g^{-1}} \circ \alpha_g = \alpha_{g^{-1} \cdot g} = \alpha_1$
- 34.** a. Let $f(g) = f(h)$. Then $\alpha_g = \alpha_h$ and, in particular, $\alpha_g(1) = \alpha_h(1)$, or $g \cdot 1 = h \cdot 1$ and $g = h$.
 b. For $g, h \in G$, $f(g \cdot h) = \alpha_{g \cdot h} = \alpha_g \circ \alpha_h = f(g) \circ f(h)$.
- 35.** $x \rho x$ because $f(x) = f(x)$.
 $x \rho y \rightarrow y \rho x$ because if $f(x) = f(y)$ then $f(y) = f(x)$.
 $x \rho y$ and $y \rho z \rightarrow x \rho z$ because if $f(x) = f(y)$ and $f(y) = f(z)$, then $f(x) = f(z)$.
- 36.** $K = \{x \in \mathbb{Z} \mid f(x) = x \cdot 3 \equiv 0\}$. Therefore $K = \{0, \pm 3, \pm 6, \pm 9, \dots\} = 3\mathbb{Z}$.
- 37.** a. $1 +_8 S = \{1 +_8 0, 1 +_8 2, 1 +_8 4, 1 +_8 6\} = \{1, 3, 5, 7\}$
 $3 +_8 S = \{3 +_8 0, 3 +_8 2, 3 +_8 4, 3 +_8 6\} = \{3, 5, 7, 1\}$
 $7 +_8 S = \{7 +_8 0, 7 +_8 2, 7 +_8 4, 7 +_8 6\} = \{7, 1, 3, 5\}$
 b. $2 +_8 S = \{2 +_8 0, 2 +_8 2, 2 +_8 4, 2 +_8 6\} = \{2, 4, 6, 0\}$
 $2 +_8 S = 0 +_8 S = 4 +_8 S = 6 +_8 S$
- 38.** $H(X, Y) = 2$
- 39.** Straightforward matrix multiplication using addition modulo 2. For example,

$$(01111) \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = (000)$$

- 40.** $00 \rightarrow 00000$
 $01 \rightarrow 01111$
 $11 \rightarrow 11010$

Together with $10 \rightarrow 10101$, these are the four code words given in Example 21.

- 41.** a. For example,

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(There are other possibilities because the order of the rows in \mathbf{B} does not matter.)

- b. $0000 \rightarrow 0000000$
 $0001 \rightarrow 0001110$
 $0010 \rightarrow 0010011$

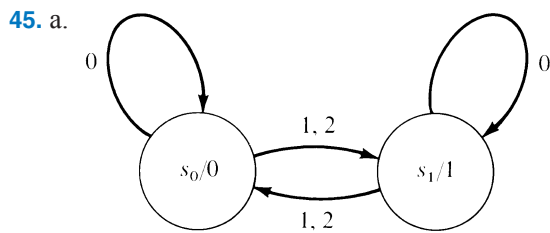
- 0011 → 0011101
- 0100 → 0100111
- 0101 → 0101001
- 0110 → 0110100
- 0111 → 0111010
- 1000 → 1000101
- 1001 → 1001011
- 1010 → 1010110
- 1011 → 1011000
- 1100 → 1100010
- 1101 → 1101100
- 1110 → 1110001
- 1111 → 1111111

- 42. (11011)**H** = 001
- (10100)**H** = 001
- (01110)**H** = 001
- (00001)**H** = 001

43. 000110

44.

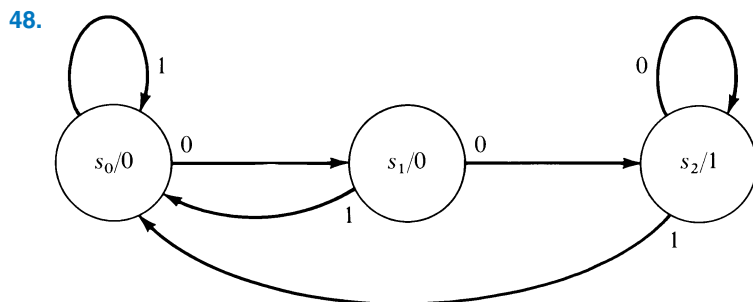
Present state	Next state		Output
	Present input 0	Present input 1	
s_0	s_0	s_3	0
s_1	s_0	s_2	1
s_2	s_3	s_3	1
s_3	s_1	s_3	2



b. 01011

46. a. s_j b. s_j

47. 11001011



49. a. Set consisting of a single 0
 b. Set consisting of any number of 0s (including none) followed by 10
 c. Set consisting of a single 0 or a single 1
 d. Set consisting of any number (including none) of pairs of 1s
50. The string in part (b) does not belong.
51. a. 0 b. 0^*10 c. $0 \vee 1$ d. $(11)^*$
52. s_2, s_3
53. A state s produces the same output as itself for any input. If s_i produces the same output as s_j , then s_j produces the same output as s_i . Transitivity is equally clear.
54. Property 1 is satisfied because all states in the same class have the same output strings for any input string, including the empty input string. To see that property 2 is satisfied, assume s_i and s_j are equivalent states proceeding under the input symbol i to states s'_i and s'_j that are not equivalent. Then there is an input string α such that $f_O(s'_i, \alpha) \neq f_O(s'_j, \alpha)$. Thus, for the input string $i\alpha$, s_i and s_j produce different output strings, contradicting the equivalence of s_i and s_j .
55. Equivalent states of M in Table 9.9 are $A = \{0,1,3\}$, $B = \{2\}$, and $C = \{4\}$. The reduced machine is

Present state	Next state		Output
	Present input		
	0	1	
A	B	A	1
B	C	A	0
C	A	A	0

Equivalent states of M in Table 9.10 are $\{0\}$, $\{1\}$, $\{2\}$, and $\{3\}$. M is already minimal.

56. First, write the state table:

Present state	Next state		Output
	Present input		
	0	1	
s_0	s_0	s_1	1
s_1	s_1	s_0	0

The states can be encoded by a single delay element, as shown:

	d
s_0	0
s_1	1

The truth functions are

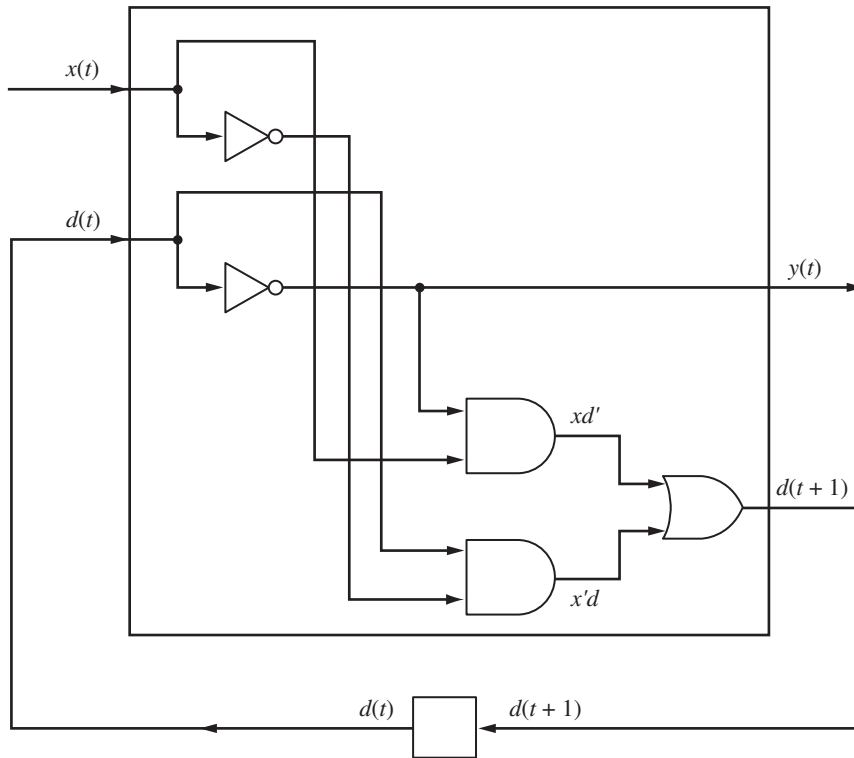
$x(t)$	$d(t)$	$y(t)$	$d(t + 1)$
0	0	1	0
1	0	1	1
0	1	0	1
1	1	0	0

The canonical sum-of-products forms are

$$y(t) = d'$$

$$d(t + 1) = xd' + x'd$$

and the sequential network is



57. a. ...b 0 0 b...
 b. The machine cycles endlessly over the two nonblank tape squares.
 c. The machine changes the two nonblank squares to 0 1 and then moves endlessly to the right.

58. a. b X X 1 X X b halts without accepting



b. b X X X X X b halts without accepting



c. b X X X 0 X X b halts without accepting



59. State 3 is the only final state.

$(0, 1, 1, 0, R)$	move right, ignoring 1s
$(0, 0, 0, 1, R)$	a 0 has been read, change state
$(1, 0, 0, 2, R)$	a second 0 in succession has been read
$(1, 1, 1, 0, R)$	no second 0, start over
$(2, 0, 0, 2, R)$	0s continue to be read
$(2, 1, 1, 0, R)$	string of 0s broken before end of input, start over
$(2, b, b, 3, R)$	end of input follows string of at least two 0s, accept

If we were not constrained to move only to the right, we could use the following machine, where state 4 is the only final state.

$(0, 0, 0, 1, R)$	} 0 reads the first symbol so the tape is not blank
$(0, 1, 1, 1, R)$	
$(1, 1, 1, 1, R)$	
$(1, 0, 0, 1, R)$	} state 1 reads to end of input, moves left
$(1, b, b, 2, L)$	
$(2, 0, 0, 3, L)$	reads one 0
$(3, 0, 0, 4, L)$	reads second 0, accepts

60. Change $(2, 1, X, 3, L)$ to $(2, 1, X, 7, L)$ and add $(7, 1, X, 3, L)$.

61. One machine that works, together with a description of its actions:

$(0, 1, 1, 1, R)$	reads first 1
$(1, b, 1, 6, R)$	$n = 0$, changes to 1 and halts
$(1, 1, 1, 2, R)$	reads second 1
$(2, b, b, 6, R)$	$n = 1$, halts
$(2, 1, 1, 3, R)$	$n \geq 2$
$(3, 1, 1, 3, R)$	} finds right end of \bar{n}
$(3, b, b, 4, L)$	
$(4, 1, b, 5, L)$	} erases two 1s from \bar{n} and halts
$(5, 1, b, 6, L)$	

62. $0S \Rightarrow 00S \Rightarrow 000S \Rightarrow 0000S \Rightarrow 00001$

63. $L = \{0^n 1 \mid n \geq 0\}$

64. For example:

a. $G(V, V_T, S, P)$ where $V = \{0, 1, S\}$, $V_T = \{0, 1\}$, and $P = \{S \rightarrow 1, S \rightarrow 0S0\}$

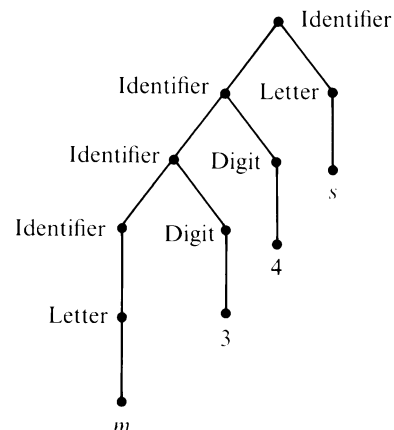
b. $G = (V, V_T, S, P)$ where $V = \{0, 1, S, M\}$, $V_T = \{0, 1\}$, and $P = \{S \rightarrow 0M0, M \rightarrow 0M0, M \rightarrow 1\}$

65. In G_1 : $S \Rightarrow ABA \Rightarrow 00A \Rightarrow 0000A \Rightarrow 00000$

In G_2 : $S \Rightarrow 00A \Rightarrow 0000A \Rightarrow 00000$

In G_3 : $S \Rightarrow 0A \Rightarrow 00B \Rightarrow 000C \Rightarrow 0000B \Rightarrow 00000$

66.



Answers to Odd-Numbered Exercises

(Note that these are answers, not necessarily complete solutions; your instructor may require more explanation or justification, as well as a different format, for some of these exercises.)

CHAPTER 1

EXERCISES 1.1

1. a, c, d, e, f
3. a. T b. T c. T d. F
5. a. If there is sufficient water, then there is healthy plant growth.
b. If there are further technological advances, then there is increased availability of information.
c. If errors were introduced, then there was a modification of the program.
d. If there is fuel savings, then there is good insulation or storm windows throughout.
7. a. $A \vee B$ b. $A' \wedge B'$
9. a. 1 and 3 b. 2 c. 4
11. a. The food is good but the service is poor.
b. The food is poor and so is the service.
c. Either the food is poor or the service is poor, but the price is low.
d. Either the food is good or the service is excellent.
e. The price is high but either the food is poor or the service is poor.
13. a. $[A \rightarrow B \wedge C] \wedge (C' \rightarrow B)$
b. $[(A \vee B) \rightarrow C] \wedge (C \rightarrow B)'$
c. $(A \vee B) \wedge (A \wedge B)'$
d. $(A \vee B) \rightarrow C$
e. $A \vee (B \rightarrow C)$
15. a. $A \wedge B$
b. $A \wedge (B \vee C)$
c. $B \rightarrow (A \wedge C)$
d. $A \rightarrow (B' \vee C')$
e. $A \wedge [C' \rightarrow (B' \vee C)]$
17. a. Violets are blue or sugar is sour.
b. Violets are not blue or, if roses are red, then sugar is sweet.
c. Sugar is sweet and roses are not red, if and only if violets are blue.

- d. Sugar is sweet, and roses are not red if and only if violets are blue.
 e. If it is false that both violets are blue and sugar is sour, then roses are red.
 f. Roses are red, or violets are blue and sugar is sour.
 g. Roses are red or violets are blue, and sugar is sour.

19. a. $H \rightarrow K$ b. $K \rightarrow (H \wedge A)$ c. $K \rightarrow H$ d. $K \leftrightarrow A$ e. $(A \vee H) \rightarrow K$

21. a. $F \rightarrow B$ b. $B \rightarrow F$ c. $B' \rightarrow (F' \wedge S)$ d. $S \rightarrow B'$ e. $S \leftrightarrow F'$

23. a.

A	B	$A \rightarrow B$	A'	$A' \vee B$	$(A \rightarrow B) \leftrightarrow A' \vee B$
T	T	T	F	T	T
T	F	F	F	F	T
F	T	T	T	T	T
F	F	T	T	T	T

Tautology

b.

A	B	C	$A \wedge B$	$(A \wedge B) \vee C$	$B \vee C$	$A \wedge (B \vee C)$	$(A \wedge B) \vee C \rightarrow A \wedge (B \vee C)$
T	T	T	T	T	T	T	T
T	T	F	T	T	T	T	T
T	F	T	F	T	T	T	T
T	F	F	F	F	F	F	T
F	T	T	F	T	T	F	F
F	T	F	F	F	T	F	T
F	F	T	F	T	T	F	F
F	F	F	F	F	F	F	T

c.

A	B	A'	B'	$A' \vee B'$	$(A' \vee B)'$	$A \wedge (A' \vee B)'$
T	T	F	F	F	T	T
T	F	F	T	T	F	F
F	T	T	F	T	F	F
F	F	T	T	T	F	F

d.

A	B	A'	$A \wedge B$	$A \wedge B \rightarrow A'$
T	T	F	T	F
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

e.

A	B	C	$A \rightarrow B$	$A \vee C$	$B \vee C$	$(A \vee C) \rightarrow (B \vee C)$	$(A \rightarrow B) \rightarrow [(A \vee C) \rightarrow (B \vee C)]$
T	T	T	T	T	T	T	T
T	T	F	T	T	T	T	T
T	F	T	F	T	T	T	T
T	F	F	F	T	F	F	T
F	T	T	T	T	T	T	T
F	T	F	T	F	T	T	T
F	F	T	T	T	T	T	T
F	F	F	T	F	F	T	T

Tautology

3b.	A	B	C	$B \vee C$	$A \wedge (B \vee C)$	$A \wedge B$	$A \wedge C$	$(A \wedge B) \vee (A \wedge C)$	$A \wedge (B \vee C) \leftrightarrow (A \wedge B) \vee (A \wedge C)$
	T	T	T	T	T	T	T	T	T
	T	T	F	T	T	T	F	T	T
	T	F	T	T	T	F	T	T	T
	T	F	F	F	F	F	F	F	T
	F	T	T	T	F	F	F	F	T
	F	T	F	T	F	F	F	F	T
	F	F	T	T	F	F	F	F	T
	F	F	F	F	F	F	F	F	T

4a.	A	0	$A \vee 0$	$A \vee 0 \leftrightarrow A$
	T	F	T	T
	F	F	F	T

5b.	A	A'	$A \wedge A'$	0	$A \wedge A' \leftrightarrow 0$
	T	F	F	F	T
	F	T	F	F	T

27. a. $(A \wedge B') \wedge C \Leftrightarrow A \wedge (B' \wedge C)$ by 2b $\Leftrightarrow A \wedge (C \wedge B')$ by 1b $\Leftrightarrow (A \wedge C) \wedge B'$ by 2b

b. $(A \vee B) \wedge (A \vee B') \Leftrightarrow A \vee (B \wedge B')$ by 3a $\Leftrightarrow A \vee 0$ by 5b $\Leftrightarrow A$ by 4a

c. $A \vee (B \wedge A') \Leftrightarrow (A \vee B) \wedge (A \vee A')$ by 3a $\Leftrightarrow (A \vee B) \wedge 1$ by 5a $\Leftrightarrow A \vee B$ by 4b

29. If A is F and both B and C are T, then $(A \wedge B) \vee C$ is T but $A \wedge (B \vee C)$ is F. These two wffs are not equivalent.

31. dogs AND NOT retrievers

33. (novels OR plays) AND AIDS

35. 1.0, 2.4, 7.2, 5.3

37. if not ($Value1 < Value2$) then statement1 else statement2 end if

39.	A	B	$A \rightarrow B$	A'	$A' \vee B$	$A \rightarrow B \leftrightarrow A' \vee B$
	T	T	T	F	T	T
	T	F	F	F	F	T
	F	T	T	T	T	T
	F	F	T	T	T	T

41. a. Assign $B' \wedge (A \rightarrow B)$ true and A' false. From the second assignment, A is true. From the first assignment, B' is true (so B is false), and $A \rightarrow B$ is true. If $A \rightarrow B$ is true and A is true, then B is true. B is thus both true and false, and $[B' \wedge (A \rightarrow B)] \rightarrow A'$ is a tautology.

b. Assign $(A \rightarrow B) \wedge A$ true and B false. From the first assignment, A is true and $A \rightarrow B$ is true. If $A \rightarrow B$ is true and A is true, then B is true. B is thus both true and false, and $[(A \rightarrow B) \wedge A] \rightarrow B$ is a tautology.

c. Assign $(A \vee B) \wedge A'$ true and B false. From the first assignment, A' is true (and A is false), and $A \vee B$ is true. If $A \vee B$ is true and A is false, then B is true. B is thus both true and false, and $(A \vee B) \wedge A' \rightarrow B$ is a tautology.

43. $2^{2^5} = 2^{32}$

45.	A	B	$A \oplus B$
	T	T	F
	T	F	T
	F	T	T
	F	F	F

47. a.

A	B	$A \vee B$	A'	B'	$A' \wedge B'$	$(A' \wedge B')'$	$A \vee B \leftrightarrow (A' \wedge B')'$
T	T	T	F	F	F	T	T
T	F	T	F	T	F	T	T
F	T	T	T	F	F	T	T
F	F	F	T	T	T	F	T

b.

A	B	B'	$A \wedge B'$	$(A \wedge B')'$	$A \rightarrow B$	$A \rightarrow B \leftrightarrow (A \wedge B')'$
T	T	F	F	T	T	T
T	F	T	T	F	F	T
F	T	F	F	T	T	T
F	F	T	F	T	T	T

49. $A \wedge B$ is equivalent to $(A \rightarrow B)'$

A	B	$A \wedge B$	B'	$A \rightarrow B'$	$(A \rightarrow B)'$	$A \wedge B \leftrightarrow (A \rightarrow B)'$
T	T	T	F	F	T	T
T	F	F	T	T	F	T
F	T	F	F	T	F	T
F	F	F	T	T	F	T

$A \vee B$ is equivalent to $A' \rightarrow B$

A	B	$A \vee B$	A'	$A' \rightarrow B$	$A \vee B \leftrightarrow A' \rightarrow B$
T	T	T	F	T	T
T	F	T	F	T	T
F	T	T	T	T	T
F	F	F	T	F	T

51. $A \wedge B$ is equivalent to $(A|B)|(A|B)$

A	B	$A \wedge B$	$A B$	$(A B) (A B)$	$A \wedge B \leftrightarrow (A B) (A B)$
T	T	T	F	T	T
T	F	F	T	F	T
F	T	F	T	F	T
F	F	F	T	F	T

A' is equivalent to $A|A$

A	A'	$A A$	$A' \leftrightarrow A A$
T	F	F	T
F	T	T	T

53. a. For $A \wedge B$ to be true, we would want to know that both parts are true; if one part has an unknown truth value then it is unknown whether this is the case. For $A \vee B$ to be true, we would want at least one part to be true; if one part is false and the other part has an unknown truth value, then it is unknown whether this is the case. Finally, if the truth value of A is unknown, then the truth value of A' is also unknown.
- b. N c. F d. T
55. 3^n
57. Machine D is either clean or infected. In either case, by statements 3 and 1, respectively, C is infected. Because C is infected, then A is infected by statement 2. By statement 4, B is infected (because C is not clean). By statement 3, because B is infected, D is not clean. The conclusion is that all four machines are infected.
59. This could include everyone because members are welcome and nonmembers are welcome. Or it could include no one because no one is both a member and a nonmember.
61. If Percival is a liar, then his statement is false. Therefore it is false that there is at least one liar, and both Percival and Llewellyn must be truth-tellers. But this is impossible because we assumed Percival is a liar. Therefore Percival is a truth-teller, and his statement is true. Because he said, "At least one of us is a liar," Llewellyn must be a liar. Therefore Percival is a truth-teller and Llewellyn is a liar.
63. Rothwold's statement is of the form $A \vee B$, where A stands for "I am a liar" and B stands for "Grymlin is a truth-teller." If Rothwold is a liar, then his statement $A \vee B$ is false, and the statement $(A \vee B)'$ must be true. By De Morgan's laws, A' and B' must both be true. But A' is the statement that Rothwold is a truth-teller, which is not true. Therefore Rothwold must be a truth-teller, and his statement $A \vee B$ is true. Statement A, however, is false because it says that Rothwold is a liar. So statement B must be true, and Grymlin is a truth-teller. Both are truth-tellers.

EXERCISES 1.2

1. mt
3. sim
5. By mt, the conclusion is that the car was not involved in the hit-and-run.
7. By simplification, the conclusion is that you will be paid tomorrow.
9. 1. hyp
2. hyp
3. hyp (deduction method)
4. 2, 3, mp
5. 1, 4, con
11. 1. hyp
2. hyp
3. hyp
4. 2, 3, con
5. 4, De Morgan
6. 1, 5, mt
13. 1. $(A \vee B)'$ hyp 5. A' 4, sim
2. $(B \rightarrow C)$ hyp 6. B 4, sim
3. $A' \wedge (B)'$ 1, DeMorgan 7. C 2, 6 mp
4. $A' \wedge B$ 3, dn 8. $A' \wedge C$ 5, 7, con
15. 1. $A \rightarrow B$ hyp 4. B 1, 3, mp
2. $A \rightarrow (B \rightarrow C)$ hyp 5. $B \rightarrow C$ 2, 3, mp
3. A hyp 6. C 4, 5, mp

- | | |
|--|---|
| <p>17. 1. A' hyp
 2. $A \vee B$ hyp
 3. $(A')' \vee B$ 2, dn</p> | <p>4. $A' \rightarrow B$ 3, imp
 5. B 1, 4, mp</p> |
| <p>19. 1. $A' \rightarrow B'$ hyp
 2. B hyp
 3. $A \rightarrow C$ hyp
 4. $(B')'$ 2, dn</p> | <p>5. $(A')'$ 1, 4, mt
 6. A 5, dn
 7. C 3, 6, mp</p> |
| <p>21. 1. $A \rightarrow (B \rightarrow C)$ hyp
 2. B hyp
 3. A hyp (using deduction method again)</p> | <p>4. $B \rightarrow C$ 1, 3, mp
 5. C 2, 4, mp</p> |
| <p>23. 1. $A \rightarrow C$ hyp
 2. $C \rightarrow B'$ hyp
 3. B hyp</p> <p>or</p> <p>1. $A \rightarrow C$ hyp
 2. $C \rightarrow B'$ hyp
 3. B hyp</p> | <p>4. $(B')'$ 3, dn
 5. C' 2, 4, mt
 6. A' 1, 5, mt</p> <p>4. $A \rightarrow B'$ 1, 2, hs
 5. $(B')'$ 3, dn
 6. A' 4, 5, mt</p> |
| <p>25. 1. $P \vee Q$ hyp
 2. P' hyp
 3. $(P')' \vee Q$ 1, dn</p> | <p>4. $P' \rightarrow Q$ 3, imp
 5. Q 2, 4, mp</p> |
| <p>27. 1. $Q' \rightarrow P'$ hyp
 2. P hyp
 3. $(P')'$ 2, dn</p> | <p>4. $(Q')'$ 1, 3, mt
 5. Q 4, dn</p> |
| <p>29. 1. $P' \rightarrow P' \wedge P'$
 2. $P' \rightarrow (P \vee P)'$
 3. $[P' \rightarrow (P \vee P)'] \rightarrow [(P \vee P) \rightarrow P]$
 4. $P \vee P \rightarrow P$</p> | <p>Exercise 28
 1, De Morgan
 Exercise 27
 2, 3, mp</p> |
| <p>31. 1. P hyp
 2. P' hyp
 3. $P \vee Q$ 1, add
 4. $Q \vee P$ 3, comm</p> | <p>5. $(Q')' \vee P$ 4, dn
 6. $Q' \rightarrow P$ 5, imp
 7. $(Q')'$ 2, 6, mt
 8. Q 7, dn</p> |

33. Prove

$$P \vee (Q \wedge R) \rightarrow (P \vee Q)$$

Rewriting the conclusion, the argument is

$$P \vee (Q \wedge R) \rightarrow ((P')' \vee Q) \text{ by dn}$$

or

$$P \vee (Q \wedge R) \rightarrow (P' \rightarrow Q) \text{ by imp}$$

- | | |
|--|--|
| <p>1. $P \vee (Q \wedge R)$ hyp
 2. P' hyp
 3. $(P')' \vee (Q \wedge R)$ 1, dn
 4. $P' \rightarrow (Q \wedge R)$ 3, imp
 5. $Q \wedge R$ 2, 4, mp
 6. Q 5, sim</p> | <p>hyp
 hyp
 1, dn
 3, imp
 2, 4, mp
 5, sim</p> |
|--|--|

The proof for

$$P \vee (Q \wedge R) \rightarrow (P \vee R)$$

is similar.

- 35.**
- | | | | |
|------------------------|----------|-------------------|---------|
| 1. $P \rightarrow Q$ | hyp | 5. $(Q')' \vee Q$ | imp |
| 2. $P' \rightarrow Q$ | hyp | 6. $Q \vee Q$ | 5, dn |
| 3. $Q' \rightarrow P'$ | 1, cont | 7. Q | 6, self |
| 4. $Q' \rightarrow Q$ | 2, 3, hs | | |
- 37.**
- | | | | |
|-----------------------|-----|-----------------------|----------|
| 1. $A' \rightarrow B$ | hyp | 4. $A' \rightarrow C$ | 1, 2, hs |
| 2. $B \rightarrow C$ | hyp | 5. $A' \rightarrow D$ | 3, 4, hs |
| 3. $C \rightarrow D$ | hyp | | |
- 39.**
- | | | | |
|-----------------------|-----|---------|-----------|
| 1. $Y \rightarrow Z'$ | hyp | 4. Z' | 1, 3, mp |
| 2. $Y \rightarrow Z$ | hyp | 5. Z | 2, 3, mp |
| 3. Y | hyp | 6. W | 4, 5, inc |
- 41.**
- | | | | |
|-----------------------|--------------|---|--------------|
| 1. $(A \wedge B)'$ | hyp | 8. $C' \rightarrow A'$ | 7, imp |
| 2. $(C' \wedge A)'$ | hyp | 9. $C' \vee (B')'$ | 3, De Morgan |
| 3. $(C \wedge B')'$ | hyp | 10. $(B')' \vee C'$ | 9, comm |
| 4. $A' \vee B'$ | 1, De Morgan | 11. $B' \rightarrow C'$ | 10, imp |
| 5. $B' \vee A'$ | 4, comm | 12. $B' \rightarrow A'$ | 8, 11, hs |
| 6. $B \rightarrow A'$ | 5, imp | 13. $(B \rightarrow A') \wedge (B' \rightarrow A')$ | 6, 12, con |
| 7. $(C')' \vee A'$ | 2, De Morgan | 14. A' | Exercise 35 |
- 43.** The argument is $(E \rightarrow Q) \wedge (E \vee B) \wedge Q' \rightarrow B$
A proof sequence is
- | | | | |
|------------------------|---------|-----------------------|----------|
| 1. $E \rightarrow Q$ | hyp | 5. E' | 3, 4, mp |
| 2. $E \vee B$ | hyp | 6. $(E')' \vee B$ | 2, dn |
| 3. Q' | hyp | 7. $E' \rightarrow B$ | 6, imp |
| 4. $Q' \rightarrow E'$ | 1, cont | 8. B | 5, 7, mp |

45. The argument is

$$(C \rightarrow F') \wedge (F \vee S) \rightarrow (C \rightarrow S)$$

A proof sequence is

- | | | | |
|-----------------------|-----|---------|----------|
| 1. $C \rightarrow F'$ | hyp | 4. F' | 1, 3, mp |
| 2. $F \vee S$ | hyp | 5. S | 2, 4, ds |
| 3. C | hyp | | |

47. The argument is

$$[(A \rightarrow S) \wedge (A \vee C) \wedge S'] \rightarrow C$$

A proof sequence is

- | | | | |
|----------------------|-----|---------|----------|
| 1. $A \rightarrow S$ | hyp | 4. A' | 1, 3, mt |
| 2. $A \vee C$ | hyp | 5. C | 2, 4, ds |
| 3. S' | hyp | | |

49. The argument is

$$[(R \wedge (F' \vee N)) \wedge N' \wedge (A' \rightarrow F)] \rightarrow (A \wedge R)$$

A proof sequence is:

1. $R \wedge (F' \vee N)$	hyp	7. F'	2, 6, ds
2. N'	hyp	8. $F' \rightarrow (A)'$	3, cont
3. $A' \rightarrow F$	hyp	9. $(A)'$	7, 8, mp
4. R	1, sim	10. A	9, dn
5. $F' \vee N$	1, sim	11. $A \wedge R$	4, 10, con
6. $N \vee F'$	5, comm		

51. The argument is

$$[((J \vee L) \rightarrow C) \wedge T' \wedge (C \rightarrow T)] \rightarrow J'$$

A proof sequence is

1. $(J \vee L) \rightarrow C$	hyp	6. $C' \rightarrow (J \vee L)'$	1, cont
2. T'	hyp	7. $(J \vee L)'$	5, 6, mp
3. $C \rightarrow T$	hyp	8. $J' \wedge L'$	7, De Morgan
4. $T' \rightarrow C'$	3, cont	9. J'	8, sim
5. C'	2, 4, mp		

or

1. $(J \vee L) \rightarrow C$	hyp	5. $(J \vee L)'$	1, 4, mt
2. T'	hyp	6. $J' \wedge L'$	5, De Morgan
3. $C \rightarrow T$	hyp	7. J'	6, sim
4. C'	2, 3, mt		

53. The argument is

$$(D \rightarrow T) \wedge (D \vee B) \rightarrow (T' \rightarrow B)$$

A proof sequence is

1. $D \rightarrow T$	hyp	4. D'	1, 3, mt
2. $D \vee B$	hyp	5. B	2, 4, ds
3. T'	hyp		

55. a.

A	B	C	$B \rightarrow C$	$A \rightarrow (B \rightarrow C)$	$A \wedge B$	$(A \wedge B) \rightarrow C$	$A \rightarrow (B \rightarrow C) \leftrightarrow (A \wedge B) \rightarrow C$
T	T	T	T	T	T	T	T
T	T	F	F	F	T	F	T
T	F	T	T	T	F	T	T
T	F	F	T	T	F	T	T
F	T	T	T	T	F	T	T
F	T	F	F	T	F	T	T
F	F	T	T	T	F	T	T
F	F	F	T	T	F	T	T

b. $A \rightarrow (B \rightarrow C) \Leftrightarrow A \rightarrow (B' \vee C) \Leftrightarrow A' \vee (B' \vee C) \Leftrightarrow (A' \vee B') \vee C \Leftrightarrow (A \wedge B)' \vee C \Leftrightarrow (A \wedge B) \rightarrow C$

c. By part (a) (or (b)), $[P_1 \wedge P_2 \wedge \dots \wedge P_n] \rightarrow (R \rightarrow S) \Leftrightarrow (P_1 \wedge P_2 \wedge \dots \wedge P_n \wedge R) \rightarrow S$, which says to take each of P_1, P_2, \dots, P_n, R as hypotheses and deduce S .

EXERCISES 1.3

1. a. T b. F c. F d. T
3. a. T b. T c. T d. F e. F f. T g. T h. F
5. a. F b. T c. T d. F e. T f. T
7. a. true: domain is the integers, $A(x)$ is “ x is even”, $B(x)$ is “ x is odd”, false: domain is the positive integers, $A(x)$ is “ $x > 0$ ”, $B(x)$ is “ $x \geq 1$ ”
 b. true: domain is the collection of lines in the plane, $P(x, y)$ is “ x is parallel to y ”, false: domain is the integers, $P(x, y)$ is “ $x < y$ ”
 c. true: domain is the integers, $P(x)$ is “ x is even”, $Q(x, y)$ is “ $y|x$ ” (y divides x), false: domain is the collection of all people, $P(x)$ is “ x is male”, $Q(x, y)$ is “ y is a brother of x ”
9. a. scope of $(\forall x)$ is $P(x) \rightarrow Q(y)$; y is a free variable
 b. scope of $(\exists x)$ is $A(x) \wedge (\forall y)B(y)$; scope of $(\forall y)$ is $B(y)$; no free variables
 c. scope of $(\exists x)$ is $(\forall y)P(x, y) \wedge Q(x, y)$; scope of $(\forall y)$ is $P(x, y)$; y is a free variable
 d. scope of $(\exists x)$ is $(\exists y)[A(x, y) \wedge B(y, z) \rightarrow A(a, z)]$; scope of $(\exists y)$ is $A(x, y) \wedge B(y, z) \rightarrow A(a, z)$; z is a free variable
11. b and c

Many parts of Exercises 13–24 have multiple equivalent answers, some of which are shown here.

13. a. $(\forall x)(D(x) \rightarrow S(x))$
 b. $(\exists x)[D(x) \wedge (R(x))']$ or $[(\forall x)(D(x) \rightarrow R(x))']$
 c. $(\forall x)[D(x) \wedge S(x) \rightarrow (R(x))']$
 d. $(\exists x)[D(x) \wedge S(x) \wedge R(x)]$
 e. $(\forall x)[D(x) \rightarrow (S(x) \wedge R(x))']$
 f. $(\forall x)[D(x) \wedge S(x) \rightarrow D(x) \wedge R(x)]$
 g. $(\forall x)[D(x) \rightarrow (S(x))']$
 h. $S(M) \rightarrow (\forall x)(D(x) \rightarrow S(x))$
 i. $R(M) \wedge R(T)$
 j. $(\exists x)(D(x) \wedge R(x)) \rightarrow (\forall x)(D(x) \rightarrow S(x))$
15. a. $(\forall x)(M(x) \rightarrow T(x))$
 b. $(\exists x)(W(x) \wedge T(x))$
 c. $(\forall x)(M(x) \rightarrow T(x)) \wedge (\forall x)(W(x) \rightarrow [T(x)]')$
 d. $(\forall x)(T(x) \rightarrow W(x))$
 e. $(\forall x)[M(x) \rightarrow (T(x))']$
 f. $(\forall x)(M(x) \rightarrow T(x)) \rightarrow (\forall x)(W(x) \rightarrow T(x))$
 g. $(\exists x)[W(x) \wedge (T(x))']$
 h. $(\forall x)[M(x) \rightarrow (T(x))'] \rightarrow (\exists x)[W(x) \wedge (T(x))']$
17. a. $(\exists x)[P(x) \wedge (\forall y)(T(y) \rightarrow F(x, y))]$
 b. $(\forall x)[P(x) \rightarrow (\exists y)(T(y) \wedge F(x, y))]$
 c. $(\exists x)(\exists y)(P(x) \wedge T(y) \wedge (F(x, y))')$
19. a. $(\forall x)(\forall y)(M(x) \wedge G(y) \rightarrow F(x, y))$
 b. $[(\exists x)(G(x) \wedge (\forall y)(M(y) \rightarrow F(x, y)))]'$ or $(\forall x)(G(x) \rightarrow (\exists y)(M(y) \wedge [F(x, y)]'))'$
 c. $(\forall x)(\forall y)(M(y) \wedge F(x, y) \rightarrow G(x))$
 d. $(\forall x)(G(x) \rightarrow (\exists y)(M(y) \wedge F(x, y)))$ or $(\forall x)(\exists y)(G(x) \rightarrow (M(y) \wedge F(x, y)))$
21. a. $(\exists x)(W(x) \wedge L(x) \wedge C(x))$
 b. $(\forall x)[W(x) \rightarrow (L(x) \wedge C(x))']$
 c. $(\exists x)[L(x) \wedge (\forall y)(A(x, y) \rightarrow J(y))]$ or $(\exists x)(\forall y)[L(x) \wedge (A(x, y) \rightarrow J(y))]$

- d. $(\forall x)[J(x) \rightarrow (\forall y)(A(x, y) \rightarrow J(y))]$ or $(\forall x)(\forall y)[J(x) \rightarrow (A(x, y) \rightarrow J(y))]$ or $(\forall x)(\forall y)[J(x) \wedge A(x, y) \rightarrow J(y)]$
- e. $(\forall x)(\forall y)[(J(y) \wedge A(x, y)) \rightarrow J(x)]$
- f. $(\forall x)([W(x) \wedge L(x)] \rightarrow (\exists y)[J(y) \wedge A(x, y)])$ or $(\forall x)(\exists y)([W(x) \wedge L(x)] \rightarrow [J(y) \wedge A(x, y)])$
- g. $(\exists x)(W(x) \wedge (\forall y)[L(y) \rightarrow (A(x, y))'])$ or $(\exists x)(W(x) \wedge (\forall y)[A(x, y) \rightarrow (L(y))'])$ or $(\exists x)(\forall y)(W(x) \wedge [L(y) \rightarrow (A(x, y))'])$
- 23.** a. $(\forall x)[B(x) \rightarrow (\forall y)(F(y) \rightarrow L(x, y))]$ or $(\forall x)(\forall y)[(B(x) \wedge F(y)) \rightarrow L(x, y)]$
 b. $(\exists x)[B(x) \wedge (\forall y)(F(y) \rightarrow L(x, y))]$
 c. $(\forall x)[B(x) \rightarrow (\exists y)(F(y) \wedge L(x, y))]$
 d. $(\forall x)[B(x) \rightarrow (\forall y)((L(x, y))' \rightarrow F(y))]$
 e. $(\forall y)[F(y) \rightarrow (\forall x)(L(x, y) \rightarrow B(x))]$ or $(\forall y)(\forall x)[(F(y) \wedge L(x, y)) \rightarrow B(x)]$
 f. $(\forall x)[B(x) \rightarrow (\forall y)(L(x, y) \rightarrow F(y))]$
 g. $[(\exists x)[B(x) \wedge (\forall y)(L(x, y) \rightarrow F(y))]]'$ or $(\forall x)[B(x) \rightarrow (\exists y)(L(x, y) \wedge (F(y))')]$
 h. $(\exists x)[B(x) \wedge (\exists y)(F(y) \wedge L(x, y))]$ or $(\exists x)(\exists y)[B(x) \wedge F(y) \wedge L(x, y)]$
 i. $((\exists x)[B(x) \wedge (\forall y)(L(x, y) \rightarrow F(y))])'$
 j. $(\forall x)[B(x) \rightarrow (\exists y)(F(y) \wedge (L(x, y))')]$
 k. $(\forall x)[B(x) \rightarrow (\forall y)(F(y) \rightarrow (L(x, y))')]$ or $(\forall x)(\forall y)[(B(x) \wedge F(y)) \rightarrow (L(x, y))']$
 l. $[(\exists x)[B(x) \wedge (\forall y)(F(y) \rightarrow (L(x, y))')]]'$ or $(\forall x)[B(x) \rightarrow (\exists y)(F(y) \wedge L(x, y))]$
- 25.** a. John is handsome and Kathy loves John.
 b. All men are handsome.
 c. All women love only handsome men.
 d. A handsome man loves Kathy.
 e. Some pretty woman loves only handsome men.
 f. John loves all pretty women.
- 27.** a. 2 b. 3 c. 3 d. 1
- 29.** a. No Web site features audio.
 b. Some Web site does not have audio or does not have video.
 c. Some Web site has neither audio nor video.
 d. Every Web site has either audio or video.
 e. Some Web site does not have text and also either doesn't have audio or doesn't have video.
- 31.** a. Every farmer grows something besides corn.
 b. Some farmer does not grow corn.
 c. Someone besides a farmer grows corn.
- 33.** a. Both sides are true exactly when $A(x, y)$ holds for all x, y pairs.
 b. Both sides are true exactly when some x, y pair satisfies the property $A(x, y)$.
 c. If there is a single x that is in relation P to all y , then for every y an x exists (this same x) that is in relation P to y .
 d. If a has property A , then something in the domain has property A .
 e. If any member of the domain that has property A also has property B , then if all members of the domain have property A , all have property B .
- 35.** a. valid: there is an x in the domain with property A says it is false that everything in the domain fails to have property A .
 b. not valid: domain is the integers, $P(x)$ is "x is even", $Q(x)$ is "x is prime". Because there are prime integers, $(\exists x)Q(x)$ and therefore $(\forall x)P(x) \vee (\exists x)Q(x)$ is true. But it is false that every integer is even or prime, so the implication is false.
- 37.** If something in the domain has either property P or property Q , then something has property P or something has property Q , and vice versa.

EXERCISES 1.4

1. The conclusion is that pansies are plants. The hypotheses have the form $(\forall x)(F(x) \rightarrow P(x)) \wedge F(p)$. By universal instantiation, $F(p) \rightarrow P(p)$, then by modus ponens, $P(p)$.
3. The conclusion is that pansies are red. The hypotheses have the form $(\forall x)[F(x) \rightarrow (R(x) \vee P(x))] \wedge F(p) \wedge [P(p)]'$. By universal instantiation, $F(p) \rightarrow (R(p) \vee P(p))$, then by modus ponens, $R(p) \vee P(p)$, and finally by disjunctive syllogism, $R(p)$.
5. No conclusion is possible. Just because pansies are flowers, it does not make them either red or purple. The hypotheses have the form $(\exists x)(F(x) \wedge R(x))$, $(\exists x)(F(x) \wedge P(x))$, $F(p)$. But existential instantiation does not allow us to use p in removing the existential quantifiers, so we can say nothing further about pansies.
7.
 1. hyp
 2. 1, ei
 3. hyp
 4. 3, ui
 5. 2, 4, mp
 6. 5, eg
9.
 - a. The domain is the set of integers, $P(x, y)$ is " $x < y$ ", and $Q(x, y)$ is " $x > y$ "; for every integer x , there is some integer that is larger and there is some integer that is smaller. But it is false that for every integer x there is some one integer that is both larger and smaller than x .
 - b. To get to step 2, ei was performed on two different existential quantifiers, neither of which was in front with the whole rest of the wff as its scope. Also, both existential quantifiers were removed at once, with the same constant a substituted for the variable in each case; this should be done in two steps, and the second would then have to introduce a new constant not previously used in the proof. And at step 3, the existential quantifier was not inserted at the front of the wff.
11.

1. $(\forall x)P(x)$	hyp		
2. $P(x)$	1, ui		
3. $P(x) \vee Q(x)$	2, add		
4. $(\forall x)(P(x) \vee Q(x))$	3, ug (note that $P(x) \vee Q(x)$ was deduced from $(\forall x)P(x)$ in which x is not free		
13.

1. $(\exists x)(\exists y)P(x, y)$	hyp	4. $(\exists x)P(x, b)$	3, eg
2. $(\exists y)P(a, y)$	1, ei	5. $(\exists y)(\exists x)P(x, y)$	4, eg
3. $P(a, b)$	2, ei		
15.

1. $(\forall x)P(x)$	hyp	4. $P(a)$	1, ui
2. $(\exists x)[P(x)]'$	hyp	5. $Q(a)$	3, 4, inc
3. $[P(a)]'$	2, ei	6. $(\exists x)Q(x)$	5, eg
17.

1. $(\exists x)(A(x) \wedge B(x))$	hyp	5. $(\exists x)A(x)$	3, eg
2. $A(a) \wedge B(a)$	1, ei	6. $(\exists x)B(x)$	4, eg
3. $A(a)$	2, sim	7. $(\exists x)A(x) \wedge (\exists x)B(x)$	5, 6, con
4. $B(a)$	2, sim		
19. Domain is the integers, $P(x)$ is " x is even", $Q(x, y)$ is " $x = 2y + 1$ " (which means that x is odd). Then $(\exists x)P(x)$ is true ($x = 2$) and $(\exists x)(\exists y)Q(x, y)$ is true ($3 = 2 * 1 + 1$), but $(\exists x)(\exists y)[P(x) \wedge Q(x, y)]$ is false (no x is both even and odd).
21.

1. $(\forall x)(P(x))'$	hyp	5. $P(x) \rightarrow Q(x)$	temporary hyp
2. $(P(x))'$	1, ui		discharged
3. $P(x)$	temporary hyp	6. $(\forall x)(P(x) \rightarrow Q(x))$	5, ug
4. $Q(x)$	2, 3, inc		
23.

1. $(\exists x)(\forall y)Q(x, y)$	hyp	4. $(\exists x)Q(x, y)$	3, eg
2. $(\forall y)Q(a, y)$	1, ei	5. $(\forall y)(\exists x)Q(x, y)$	4, ug
3. $Q(a, y)$	2, ui		

- 25.** 1. $(\forall x)(A(x) \rightarrow B(x))$ hyp
 2. $(\exists x)A(x)$ hyp
 3. $A(a)$ 2, ei
 4. $A(a) \rightarrow B(a)$ 1, ui
 5. $B(a)$ 3,4, mp
 6. $(\exists x)B(x)$ 5, eg
- 27.** 1. $P(x) \rightarrow (\exists y)Q(x, y)$ hyp
 2. $P(x)$ temporary hyp
 3. $(\exists y)Q(x, y)$ 1, 2, mp
 4. $Q(x, a)$ 3, ei
 5. $P(x) \rightarrow Q(x, a)$ temporary hyp discharged
 6. $(\exists y)(P(x) \rightarrow Q(x, y))$ 5, eg
- 29.** 1. $(\exists x)[P(x) \wedge Q(x)]$ hyp
 2. $(\forall y)[Q(y) \rightarrow R(y)]$ hyp
 3. $P(a) \wedge Q(a)$ 1, ei
 4. $P(a)$ 3, sim
 5. $Q(a)$ 3, sim
 6. $Q(a) \rightarrow R(a)$ 2, ui
 7. $R(a)$ 5, 6, mp
 8. $P(a) \wedge R(a)$ 4, 7, con
 9. $(\exists x)[P(x) \wedge R(x)]$ 8, eg
- 31. a.** $(\forall x)(M(x) \rightarrow P(x)) \wedge (\forall x)(S(x) \rightarrow M(x)) \rightarrow (\forall x)(S(x) \rightarrow P(x))$
 1. $(\forall x)(M(x) \rightarrow P(x))$ hyp
 2. $(\forall x)(S(x) \rightarrow M(x))$ hyp
 3. $M(x) \rightarrow P(x)$ 1, ui
 4. $S(x) \rightarrow M(x)$ 2, ui
 5. $S(x) \rightarrow P(x)$ 3, 4 hs
 6. $(\forall x)(S(x) \rightarrow P(x))$ 5, ug
- b.** $(\forall x)(M(x) \rightarrow [P(x)]') \wedge (\forall x)(S(x) \rightarrow M(x)) \rightarrow (\forall x)(S(x) \rightarrow [P(x)]')$
 1. $(\forall x)(M(x) \rightarrow [P(x)]')$ hyp
 2. $(\forall x)(S(x) \rightarrow M(x))$ hyp
 3. $M(x) \rightarrow [P(x)]'$ 1, ui
 4. $S(x) \rightarrow M(x)$ 2, ui
 5. $S(x) \rightarrow [P(x)]'$ 3, 4, hs
 6. $(\forall x)(S(x) \rightarrow [P(x)]')$ 5, ug
- c.** $(\forall x)(M(x) \rightarrow P(x)) \wedge (\exists x)(S(x) \wedge M(x)) \rightarrow (\exists x)(S(x) \wedge P(x))$
 1. $(\forall x)(M(x) \rightarrow P(x))$ hyp
 2. $(\exists x)(S(x) \wedge M(x))$ hyp
 3. $S(a) \wedge M(a)$ 2, ei
 4. $M(a)$ 3, sim
 5. $M(a) \rightarrow P(a)$ 1, ui
 6. $P(a)$ 4, 5, mp
 7. $S(a)$ 3, sim
 8. $S(a) \wedge P(a)$ 6, 7, con
 9. $(\exists x)(S(x) \wedge P(x))$ 8, eg
- d.** $(\forall x)(M(x) \rightarrow [P(x)]') \rightarrow (\exists x)(S(x) \wedge M(x)) \rightarrow (\exists x)(S(x) \wedge [P(x)]')$
 1. $(\forall x)(M(x) \rightarrow [P(x)]')$ hyp
 2. $(\exists x)(S(x) \wedge M(x))$ hyp
 3. $S(a) \wedge M(a)$ 2, ei
 4. $M(a) \rightarrow [P(a)]'$ 1, ui
 5. $M(a)$ 3, sim
 6. $[P(a)]'$ 4, 5, mp
 7. $S(a)$ 3, sim
 8. $S(a) \wedge [P(a)]'$ 6, 7, con
 9. $(\exists x)(S(x) \wedge [P(x)]')$ 8, eg

33. The argument is

$$(\forall x)(\forall y)[C(x) \wedge A(y) \rightarrow B(x, y)] \wedge C(s) \wedge (\exists x)(S(x) \wedge [B(s, x)]') \rightarrow (\exists x)[A(x)]'$$

A proof sequence is

1. $(\forall x)(\forall y)[C(x) \wedge A(y) \rightarrow B(x, y)]$ hyp
2. $C(s)$ hyp
3. $(\forall y)[C(s) \wedge A(y) \rightarrow B(s, y)]$ 1, ui
4. $(\exists x)(S(x) \wedge [B(s, x)]')$ hyp
5. $S(a) \wedge [B(s, a)]'$ 4, ei
6. $C(s) \wedge A(a) \rightarrow B(s, a)$ 3, ui
7. $[B(s, a)]'$ 5, sim
8. $[C(s) \wedge A(a)]'$ 6, 7, mt
9. $[C(s)]' \vee [A(a)]'$ 8, De Morgan
10. $[[C(s)]']'$ 2, dn
11. $[A(a)]'$ 9, 10, ds
12. $(\exists x)[A(x)]'$ 11, eg

35. The argument is

$$(\forall x)(M(x) \rightarrow I(x) \vee G(x)) \wedge (\forall x)(G(x) \wedge L(x) \rightarrow F(x)) \wedge (I(j))' \wedge L(j) \rightarrow [(M(j) \rightarrow F(j))]$$

A proof sequence is

1. $(\forall x)(M(x) \rightarrow I(x) \vee G(x))$	hyp	7. $(I(j))'$	hyp
2. $(\forall x)(G(x) \wedge L(x) \rightarrow F(x))$	hyp	8. $G(j)$	6, 7, ds
3. $M(j) \rightarrow I(j) \vee G(j)$	1, ui	9. $L(j)$	hyp
4. $G(j) \wedge L(j) \rightarrow F(j)$	2, ui	10. $G(j) \wedge L(j)$	8, 9, con
5. $M(j)$	hyp	11. $F(j)$	4, 10, mp
6. $I(j) \vee G(j)$	3, 5, mp		

37. The argument is

$$(\forall x)(R(x) \rightarrow F(x)) \wedge (\exists x)(R(x) \wedge B(x)) \wedge (\forall x)(G(x)' \rightarrow B(x)') \rightarrow (\exists x)(G(x) \wedge F(x))$$

A proof sequence is

1. $(\forall x)(R(x) \rightarrow F(x))$	hyp	8. $F(a)$	5, 6, mp
2. $(\exists x)(R(x) \wedge B(x))$	hyp	9. $G(a)' \rightarrow B(a)'$	3, ui
3. $(\forall x)(G(x)' \rightarrow B(x)')$	hyp	10. $B(a) \rightarrow G(a)$	9, cont
4. $R(a) \wedge B(a)$	2, ei	11. $G(a)$	7, 10, mp
5. $R(a) \rightarrow F(a)$	1, ui	12. $G(a) \wedge F(a)$	8, 11, con
6. $R(a)$	4, sim	13. $(\exists x)(G(x) \wedge F(x))$	12, eg
7. $B(a)$	4, sim		

39. The argument is

$$(\forall x)(C(x) \rightarrow (\exists y)W(x, y)) \wedge (\forall x)(\forall y)(W(x, y) \rightarrow S(x, y)) \wedge C(m) \rightarrow (\exists y)S(m, y)$$

A proof sequence is

1. $(\forall x)(C(x) \rightarrow (\exists y)W(x, y))$	hyp	6. $(\forall y)(W(m, y) \rightarrow S(m, y))$	5, ui
2. $C(m) \rightarrow (\exists y)W(m, y)$	1, ui	7. $W(m, a)$	4, ei
3. $C(m)$	hyp	8. $W(m, a) \rightarrow S(m, a)$	6, ui
4. $(\exists y)W(m, y)$	2, 3, mp	9. $S(m, a)$	7, 8, mp
5. $(\forall x)(\forall y)(W(x, y) \rightarrow S(x, y))$	hyp	10. $(\exists y)S(m, y)$	9, eg

41. The argument is

$$(\exists x)(E(x) \wedge (\forall y)(M(y) \rightarrow A(x, y))) \wedge (\exists x)(M(x) \wedge S(x)) \rightarrow (\exists x)(E(x) \wedge (\exists y)(S(y) \wedge A(x, y)))$$

A proof sequence is

1. $(\exists x)(E(x) \wedge (\forall y)(M(y) \rightarrow A(x, y)))$	hyp
2. $(\exists x)(M(x) \wedge S(x))$	hyp
3. $E(a) \wedge (\forall y)(M(y) \rightarrow A(a, y))$	1, ei
4. $(\forall y)(M(y) \rightarrow A(a, y))$	3, sim
5. $M(b) \wedge S(b)$	2, ei
6. $M(b) \rightarrow A(a, b)$	4, ui
7. $M(b)$	5, sim
8. $A(a, b)$	6, 7, mp
9. $S(b)$	5, sim
10. $S(b) \wedge A(a, b)$	8, 9, con
11. $(\exists y)(S(y) \wedge A(a, y))$	10, eg
12. $E(a)$	3, sim
13. $E(a) \wedge (\exists y)(S(y) \wedge A(a, y))$	11, 12 con
14. $(\exists x)(E(x) \wedge (\exists y)(S(y) \wedge A(x, y)))$	13, eg

43. $[(\exists x)[A(x)]]' \leftrightarrow (\forall x)[[A(x)]]'$ neg, using $[A(x)]'$ for $A(x)$
 $[(\exists x)[A(x)]]' \leftrightarrow (\forall x)A(x)$ dn
 $[(\forall x)A(x)]' \leftrightarrow [(\exists x)[A(x)]]'$ cont (each direction)
 $[(\forall x)A(x)]' \leftrightarrow (\exists x)[A(x)]'$ dn

EXERCISES 1.5

1. yes
 3. no
 5. fish
 7. fox, deer
 9. $herbivore(X) \leq eat(X, Y)$ and $plant(Y)$
 11. fox
 13. a. anita
 b. mike, kim
 c. judith, sam, mike, kim, joan, hamal, enrique, jefferson
 15. a. $?authorof(\text{marktwain}, \text{houndofthebaskervilles})$
 b. $?authorof(\text{williamfaulkner}, X)$
 c. $nonfictionauthor(X) \leq authorof(X, Y)$ and not $fiction(Y)$
 d. $?nonfictionauthor(X)$
 17. a. $fatherof(X, Y) \leq parentof(X, Y)$ and $male(X)$
 b. $daughterof(X, Y) \leq parentof(Y, X)$ and $female(X)$
 c. $ancestorof(X, Y) \leq parentof(X, Y)$, $ancestorof(X, Y) \leq parentof(X, Z)$ and $ancestorof(Z, Y)$
 19. a. $?dry(X)$ and $ingredientof(X, Y)$
 b. $?perishable(Y)$ and $ingredientof(X, Y)$ and $liquid(X)$
 c. $foundin(X, Y) \leq ingredientof(X, Y)$, $foundin(X, Y) \leq ingredientof(X, Z)$ and $foundin(Z, Y)$
 21. The results should agree with the results for Exercises 13 and 14.

EXERCISES 1.6

1. $x + 1 = y - 1 \leftrightarrow x = y - 2$
 3. $3x - 1 = 2y - 1 \leftrightarrow 3x = 2y$
 5. Working backward from the postcondition using the assignment rule,
 $\{x + 3 = 4\} \leftrightarrow x = 1$
 $y = x + 3$
 $\{2y = 8 \text{ or } y = 4\}$
 $y = 2 * y$
 $\{y = 8\}$
 7. Working backward from the postcondition using the assignment rule,
 $\{2x + 1 = 1\} \leftrightarrow x = 0$
 $z = 2x + 1$
 $\{z - 1 = 0 \text{ or } z = 1\}$
 $\{y = 0\}$

9. Working backward from the postcondition using the assignment rule,

$$\{x(x - 1) = x(x - 1)\}$$

$$y = x - 1$$

$$\{xy = x(x - 1)\}$$

$$y = x * y$$

$$\{y = x(x - 1)\}$$

Because the precondition is always true, so is each subsequent assertion, including the postcondition.

11. Using the conditional rule, the two implications to prove are

$$\{y = 0 \text{ and } y < 5\} y = y + 1 \{y = 1\} \text{ and } \{y = 0 \text{ and } y \geq 5\} y = 5 \{y = 1\}$$

The first is true by the assignment rule. Working backward from the postcondition,

$$\{y + 1 = 1\} \leftrightarrow y = 0$$

$$y = y + 1$$

$$\{y = 1\}$$

The second is true because the antecedent is false.

13. Using the conditional rule, the two implications to prove are

$$\{x \neq 0 \text{ and } x > 0\} y = 2 * x \{y > 0\} \text{ and } \{x \neq 0 \text{ and } x \leq 0\} y = (-2) * x \{y > 0\}$$

The first is true by the assignment rule. Working backward from the postcondition,

$$\{2 * x > 0\} \leftrightarrow x > 0 \leftrightarrow x \neq 0 \text{ and } x > 0$$

$$y = 2 * x$$

$$\{y > 0\}$$

The second is true by the assignment rule. Working backward from the postcondition,

$$\{(-2) * x > 0\} \leftrightarrow x < 0 \leftrightarrow x \neq 0 \text{ and } x \leq 0$$

$$y = (-2) * x$$

$$\{y > 0\}$$

15. Using the conditional rule and the definition of absolute value for a nonzero number, the two implications to prove are

$$\{x \neq 0 \text{ and } x \geq 0\} abs = x \{(x > 0 \text{ and } abs = x) \text{ or } (x < 0 \text{ and } abs = -x)\}$$

$$\{x \neq 0 \text{ and } x < 0\} abs = -x \{(x > 0 \text{ and } abs = x) \text{ or } (x < 0 \text{ and } abs = -x)\}$$

Using the assignment rule on the first implication gives the precondition

$$(x > 0 \text{ and } x = x) \text{ or } (x < 0 \text{ and } x = -x) \leftrightarrow (x > 0 \text{ and } x = x) \leftrightarrow (x \neq 0 \text{ and } x \geq 0).$$

Using the assignment rule on the second implication gives the precondition

$$(x > 0 \text{ and } -x = x) \text{ or } (x < 0 \text{ and } -x = -x) \leftrightarrow (x < 0 \text{ and } -x = -x) \leftrightarrow (x \neq 0 \text{ and } x < 0)$$

CHAPTER 2

EXERCISES 2.1

1. a. If there is not healthy plant growth, then there is not sufficient water.
- b. If there is not increased availability of information, then there are no further technological advances.
- c. No modification of the program implies that errors will not be introduced.
- d. Poor insulation and some windows not storm windows implies no fuel savings.

3. For example:
- a nonsquare rectangle
 - 0
 - a short, blue-eyed redhead
 - a redhead who is short
5. Half of this statement is true. If n is an odd integer, $3n + 5$ is an even integer. However, the converse is false. Consider the even integer 6. If $3n + 5 = 6$, then $3n = 1$ and $n = 1/3$, which is not an integer at all, much less an odd integer. See Exercise 25.
7. a. $4 + 6 = 10$; 4 and 6 are even but 10 is not a multiple of 4.
 b. The error lies in choosing both x and y to be equal to $2m$. This makes them the same number, which is a special case.
9. $25 = 5^2 = 9 + 16 = 3^2 + 4^2$, $100 = (10)^2 = 36 + 64 = 6^2 + 8^2$, $169 = (13)^2 = 25 + 144 = 5^2 + (12)^2$
11. $n = 1, n! = 1, 2^n = 2; n = 2, n! = 2, 2^n = 4; n = 3, n! = 6, 2^n = 8$
13. Let $x = 2m, y = 2n$, where m and n are integers. Then $x + y = 2m + 2n = 2(m + n)$, where $m + n$ is an integer, so $x + y$ is even.
15. Let $x = 2m + 1, y = 2n + 1$, where m and n are integers. Then $x + y = (2m + 1) + (2n + 1) = 2m + 2n + 2 = 2(m + n + 1)$, where $m + n + 1$ is an integer, so $x + y$ is even.
17. Let $x = 2m + 1$ and $y = 2n$ where m and n are integers. Then $x - y = 2m + 1 - 2n = 2(m - n) + 1$ where $m - n$ is an integer, so $x - y$ is odd.
19. For two consecutive integers, one is even and one is odd. The product of an even integer and an odd integer is even by the proof of Example 9.
21. Let $x = 2m$ where m is an integer. Then $x^2 = (2m)^2 = 4m^2$, where m^2 is an integer, so x^2 is divisible by 4.
23. The contrapositive is: if $x + 1 \leq 0$, then $x \leq 0$. If $x + 1 \leq 0$, then $x \leq -1 < 0$, so $x < 0$ and therefore $x \leq 0$.
25. If n is odd, then $n = 2k + 1$ for some integer k . Then $3n + 5 = 3(2k + 1) + 5 = 6k + 8$. For the converse, if $3n + 5 = 6k + 8$ for some integer k , then $3n = 6k + 3$ or $3n = 3(2k + 1)$ and $n = 2k + 1$ for some integer k , so n is an odd integer.
27. If $x < y$ then multiplying both sides of the inequality by the positive numbers x and y in turn gives $x^2 < xy$ and $xy < y^2$ and therefore $x^2 < xy < y^2$ or $x^2 < y^2$. For the other direction, if $x^2 < y^2$ then $y^2 - x^2 > 0$ by the definition of $<$, $(y + x)(y - x) > 0$ by factoring, $(y + x) < 0$ and $(y - x) < 0$ or $(y + x) > 0$ and $(y - x) > 0$ because a positive number is the product of two negatives or two positives. But it cannot be that $(y + x) < 0$ because y and x are both positive; therefore $(y + x) > 0$ and $y - x > 0$ and $y > x$.
29. Let n be a prime number with $n = 2k$, where k is an integer. Then both 2 and k divide n . Because n is prime, n is divisible only by itself and 1, so $n = 2$ and $k = 1$. Therefore $n = 2$.
31. Let p and q be divisible by n . Then $p = k_1n$ and $q = k_2n$, where k_1 and k_2 are integers, and $p + q = k_1n + k_2n = (k_1 + k_2)n$, where $k_1 + k_2$ is an integer. Therefore $p + q$ is divisible by n .
33. Because $n|m, m = k_1n$ for some integer k_1 . Because $m|p, p = k_2m$ for some integer k_2 . Then $p = k_2m = k_2(k_1n) = (k_2k_1)n$ where k_2k_1 is an integer, so $n|p$.
35. Let $x = 2n + 1$. Then $x^2 = (2n + 1)^2 = 4n^2 + 4n + 1 = 4n(n + 1) + 1$. But $n(n + 1)$ is even (Exercise 19), so $n(n + 1) = 2k$ for some integer k . Therefore $x^2 = 4(2k) + 1 = 8k + 1$.
37. $m^2n^2 = (mn)^2$
39. Proof by cases, depending on whether x and y are negative. Case 1: $x \geq 0, y \geq 0$. Then $|x| = x, |y| = y$. Also, $x + y \geq 0$ and $|x + y| = x + y$. Therefore $|x + y| = x + y = |x| + |y|$. Case 2: $x \geq 0, y < 0$. Then $|x| = x, |y| = -y$. Subcase a: $x + y \geq 0$. Then $|x + y| = x + y$. Therefore $|x + y| =$

$x + y < x + (-y)$ (remember that y is negative, so $-y$ is positive) $= |x| + |y|$. Subcase b : $x + y < 0$. Then $|x + y| = -(x + y)$. Therefore $|x + y| = -(x + y) = (-x) + (-y) \leq x + (-y)$ (remember that $x \geq 0$, so $-x \leq 0$) $= |x| + |y|$. Case 3: $x < 0, y \geq 0$. Similar to Case 2 with the roles of x and y reversed. Case 4: $x < 0, y < 0$. Then $|x| = -x, |y| = -y$. Also, $x + y < 0$ and $|x + y| = -(x + y)$. Therefore $|x + y| = -(x + y) = (-x) + (-y) = |x| + |y|$.

41. Proof by contradiction. If $x_1 < A, x_2 < A, \dots$, and $x_n < A$, then $x_1 + x_2 + \dots + x_n < A + A + \dots + A = nA$, and $(x_1 + x_2 + \dots + x_n)/n < A$, which contradicts the definition of A as the average of x_1, \dots, x_n .
43. Assume that $\sqrt{3}$ is rational. Then $\sqrt{3} = p/q$ where p and q are integers, $q \neq 0$, and p and q have no common factors (other than ± 1). If $\sqrt{3} = p/q$ then $3 = p^2/q^2$ or $3q^2 = p^2$. Then 3 divides p^2 so 3 divides p . Thus 3 is a factor of p or 9 is a factor of p^2 , and the equation $3q^2 = p^2$ can be written $3q^2 = 9x$ or $q^2 = 3x$. Then 3 divides q^2 so 3 divides q . Therefore 3 is a common factor of p and q , a contradiction.
45. Assume that $\sqrt[3]{2}$ is rational. Then $\sqrt[3]{2} = p/q$ where p and q are integers, $q \neq 0$, and p and q have no common factors (other than ± 1). If $\sqrt[3]{2} = p/q$ then $2 = p^3/q^3$ or $2q^3 = p^3$. Then 2 divides p^3 so 2 divides p . Thus 2 is a factor of p , or 8 is a factor of p^3 , and the equation $2q^3 = p^3$ can be written $2q^3 = 8x$ or $q^3 = 4x$. Then 2 divides $4x$ so 2 divides q^3 or 2 divides q . Therefore 2 is a common factor of p and q , a contradiction.
47. $0 = (0)2$, which is an integral multiple of 2.
49. 297 is a composite number; $297 = 3 \cdot 3 \cdot 3 \cdot 11$
51. Counterexample: $9 - 7 = 2$
53. Proof: If x is even, then $x = 2n$ and $x(x + 1)(x + 2) = (2n)(2n + 1)(2n + 2) = 2[(n)(2n + 1)(2n + 2)]$, which is even. If x is odd, then $x = 2n + 1$ and $x(x + 1)(x + 2) = (2n + 1)(2n + 2)(2n + 3) = 2[(2n + 1)(n + 1)(2n + 3)]$, which is even.
55. Proof: If x is even, then $x = 2n$ and $2n + (2n)^3 = 2n + 8n^3 = 2(n + 4n^3)$, which is even. If x is odd, then $x = 2n + 1$ and $(2n + 1) + (2n + 1)^3 = (2n + 1) + (8n^3 + 12n^2 + 6n + 1) = 8n^3 + 12n^2 + 8n + 2 = 2(4n^3 + 6n^2 + 4n + 1)$, which is even.
57. Counterexample: $3 \times 9 = 27$
59. Let n be odd and m be even. Then n^2 is odd by Example 9 and m^2 is even by Example 5. Then $n^2 + m^2$ is the sum of an odd and an even, so it is odd by Exercise 16.
61. For $n = 1, n + \frac{1}{n} = 1 + \frac{1}{1} = 2$. For $n \geq 2, n + \frac{1}{n} \geq 2 + \frac{1}{n} > 2$ because $1/n$ is a positive number.
63. Counterexample: 5 is prime, but $5 + 4 = 9$ is not prime.
65. Proof: $n^2 - 1 = (n + 1)(n - 1)$ where $n - 1 > 1$ (because $n > 2$), which is a nontrivial factorization, so the number is not prime.
67. Counterexample: $4^2 + 4 + 1 = 21 = 3(7)$, not prime.
69. Proof: Let x and y be rational numbers, $x = p/q, y = r/s$ with p, q, r, s integers and $q, s \neq 0$. Then $x + y = p/q + r/s = (ps + rq)/qs$, where $ps + rq$ and qs are integers with $qs \neq 0$, and any common factors between q and s can be removed. Thus $x + y$ is rational.
71. Counterexample: $\sqrt{2}$ is irrational but $\sqrt{2} \times \sqrt{2} = 2$, which is rational.
73. Angle 6 plus Angle 5 plus the right angle sum to 180° by the first fact. The right angle is 90° by the fourth fact. Therefore Angle 6 plus Angle 5 sum to 90° . Angle 6 is the same size as Angle 3 by the second fact. Therefore Angle 3 plus Angle 5 sum to 90° .
75. Assume that Angle 1 and Angle 5 are the same size. As in Exercise 73, Angle 3 plus Angle 5 sum to 90° . Because Angle 1 and Angle 5 are the same size, Angle 3 plus Angle 1 sum to 90° . Also, Angle 3 plus Angle

1 plus Angle 2 sum to 180° by the first fact. Therefore 90° plus Angle 2 sum to 180° , or Angle 2 = 90° . Angle 2 is a right angle by the fourth fact.

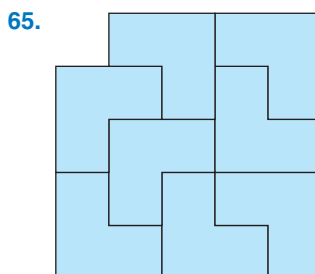
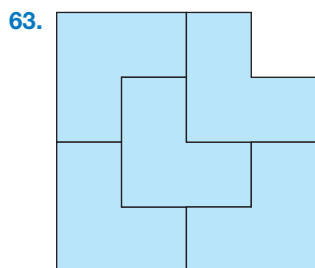
EXERCISES 2.2

1. a. $P(1)$: $4(1) - 2 = 2(1)^2$ or $2 = 2$, (true)
 b. $P(k)$: $2 + 6 + 10 + \cdots + (4k - 2) = 2k^2$
 c. $P(k + 1)$: $2 + 6 + 10 + \cdots + [4(k + 1) - 2] = 2(k + 1)^2$
 d. Left side of $P(k + 1) = 2 + 6 + 10 + \cdots + [4(k + 1) - 2] = 2 + 6 + 10 + \cdots + (4k - 2) + [4(k + 1) - 2]$ (write next-to-last term) $= 2k^2 + 4(k + 1) - 2$ (using $P(k)$) $= 2k^2 + 4k + 2 = 2(k^2 + 2k + 1) = > 2(k + 1)^2$ which is the right side of $P(k + 1)$.
3. Base case: $P(1)$: $1 = 1(2(1) - 1)$, (true). Assume $P(k)$: $1 + 5 + 9 + \cdots + (4k - 3) = k(2k - 1)$. Show $P(k + 1)$: $1 + 5 + 9 + \cdots + [4(k + 1) - 3] = (k + 1)[2(k + 1) - 1]$. Left side of $P(k + 1) = 1 + 5 + 9 + \cdots + [4(k + 1) - 3] = 1 + 5 + 9 + \cdots + (4k - 3) + [4(k + 1) - 3] = k(2k - 1) + 4(k + 1) - 3$ (using $P(k)$) $= 2k^2 - k + 4k + 1 = 2k^2 + 3k + 1 = (k + 1)(2k + 1) = (k + 1)[2(k + 1) - 1]$ which is the right side of $P(k + 1)$.
5. Base case: $P(1)$: $6 - 2 = 1[3(1) + 1]$, (true). Assume $P(k)$: $4 + 10 + 16 + \cdots + (6k - 2) = k(3k + 1)$. Show $P(k + 1)$: $4 + 10 + 16 + \cdots + [6(k + 1) - 2] = (k + 1)[3(k + 1) + 1]$. $4 + 10 + 16 + \cdots + [6(k + 1) - 2] = 4 + 10 + 16 + \cdots + (6k - 2) + [6(k + 1) - 2] = k(3k + 1) + 6(k + 1) - 2$ (using $P(k)$) $= 3k^2 + k + 6k + 4 = 3k^2 + 7k + 4 = (k + 1)(3k + 4) = (k + 1)[3(k + 1) + 1]$ which is the right side of $P(k + 1)$.
7. Base case: $P(1)$: $1^2 = 1(1 + 1)(2 + 1)/6$, (true). Assume $P(k)$: $1^2 + 2^2 + \cdots + k^2 = k(k + 1)(2k + 1)/6$. Show $P(k + 1)$: $1^2 + 2^2 + \cdots + (k + 1)^2 = (k + 1)(k + 2)(2k + 1 + 1)/6$. Left side of $P(k + 1) = 1^2 + 2^2 + \cdots + (k + 1)^2 = 1^2 + 2^2 + \cdots + k^2 + (k + 1)^2 = k(k + 1)(2k + 1)/6 + (k + 1)^2$ (using $P(k)$) $= (k + 1)[k(2k + 1)/6 + k + 1] = (k + 1)[(2k^2 + k + 6k + 6)/6] = (k + 1)(2k^2 + 7k + 6)/6 = (k + 1)(k + 2)(2k + 3)/6 = (k + 1)(k + 2)(2k + 1 + 1)/6$ which is the right side of $P(k + 1)$.
9. Base case: $P(1)$: $1^2 = 1(2 - 1)(2 + 1)/3$, (true). Assume $P(k)$: $1^2 + 3^2 + \cdots + (2k - 1)^2 = k(2k - 1)(2k + 1)/3$. Show $P(k + 1)$: $1^2 + 3^2 + \cdots + [2(k + 1) - 1]^2 = (k + 1)(2(k + 1) - 1)(2(k + 1) + 1)/3$. Left side of $P(k + 1) = 1^2 + 3^2 + \cdots + [2(k + 1) - 1]^2 = 1^2 + 3^2 + \cdots + (2k - 1)^2 + [2(k + 1) - 1]^2 = k(2k - 1)(2k + 1)/3 + [2(k + 1) - 1]^2$ (using $P(k)$) $= k(2k - 1)(2k + 1)/3 + (2k + 1)^2 = (2k + 1)[k(2k - 1)/3 + 2k + 1] = (2k + 1)(2k^2 - k + 6k + 3)/3 = (2k + 1)(2k^2 + 5k + 3)/3 = (2k + 1)(k + 1)(2k + 3)/3 = (k + 1)(2(k + 1) - 1)(2(k + 1) + 1)/3$ which is the right side of $P(k + 1)$.
11. Base case: $P(1)$: $1 \cdot 3 = 1(2)(9)/6$, (true). Assume $P(k)$: $1 \cdot 3 + 2 \cdot 4 + \cdots + k(k + 2) = k(k + 1)(2k + 7)/6$. Show $P(k + 1)$: $1 \cdot 3 + 2 \cdot 4 + \cdots + (k + 1)(k + 3) = (k + 1)(k + 2)(2k + 1 + 7)/6$. Left side of $P(k + 1) = 1 \cdot 3 + 2 \cdot 4 + \cdots + (k + 1)(k + 3) = 1 \cdot 3 + 2 \cdot 4 + \cdots + k(k + 2) + (k + 1)(k + 3) = k(k + 1)(2k + 7)/6 + (k + 1)(k + 3)$ (using $P(k)$) $= (k + 1)[k(2k + 7) + 6(k + 3)]/6 = (k + 1)(2k^2 + 13k + 18)/6 = (k + 1)(k + 2)(2k + 9)/6 = (k + 1)(k + 2)(2(k + 1) + 7)/6$ which is the right side of $P(k + 1)$.
13. Base case: $P(1)$: $1/(1 \cdot 2) = 1/(1 + 1)$, (true). Assume $P(k)$: $1/(1 \cdot 2) + 1/(2 \cdot 3) + \cdots + 1/k(k + 1) = k/(k + 1)$. Show $P(k + 1)$: $1/(1 \cdot 2) + 1/(2 \cdot 3) + \cdots + 1/(k + 1)(k + 2) = (k + 1)/(k + 2)$. Left side of $P(k + 1) = 1/(1 \cdot 2) + 1/(2 \cdot 3) + \cdots + 1/(k + 1)(k + 2) = 1/(1 \cdot 2) + 1/(2 \cdot 3) + \cdots + 1/k(k + 1) + 1/(k + 1)(k + 2) = k/(k + 1) + 1/(k + 1)(k + 2)$ (using $P(k)$) $= [k(k + 2) + 1]/(k + 1)(k + 2) = (k^2 + 2k + 1)/(k + 1)(k + 2) = (k + 1)^2/(k + 1)(k + 2) = (k + 1)/(k + 2)$ which is the right side of $P(k + 1)$.

15. Base case: $P(1): 1^2 = (-1)^2(1)(2)/2$, (true). Assume $P(k): 1^2 - 2^2 + \cdots + (-1)^{k+1}k^2 = (-1)^{k+1}(k)(k+1)/2$. Show $P(k+1): 1^2 - 2^2 + \cdots + (-1)^{k+2}(k+1)^2 = (-1)^{k+2}(k+1)(k+2)/2$. Left side of $P(k+1) = 1^2 - 2^2 + \cdots + (-1)^{k+2}(k+1)^2 = 1^2 - 2^2 + \cdots + (-1)^{k+1}k^2 + (-1)^{k+2}(k+1)^2 = (-1)^{k+1}(k)(k+1)/2 + (-1)^{k+2}(k+1)^2$ (using $P(k)$) $= [(-1)^{k+1}(k)(k+1) + 2(-1)^{k+2}(k+1)^2]/2 = (-1)^{k+2}(k+1)[k(-1)^{-1} + 2(k+1)]/2 = (-1)^{k+2}(k+1)[-k + 2k + 2]/2 = (-1)^{k+2}(k+1)(k+2)/2$ which is the right side of $P(k+1)$.
17. Base case: $P(1): 2^2 = (2)(1)(2)(2+1)/3$ or $4 = (2)(6)/3$, (true). Assume $P(k): 2^2 + 4^2 + \cdots + (2k)^2 = 2k(k+1)(2k+1)/3$. Show $P(k+1): 2^2 + 4^2 + \cdots + [2(k+1)]^2 = 2(k+1)(k+2)[2(k+1)+1]/3$. Left side of $P(k+1) = 2^2 + 4^2 + \cdots + [2(k+1)]^2 = 2^2 + 4^2 + \cdots + (2k)^2 + [2(k+1)]^2 = 2k(k+1)(2k+1)/3 + [2(k+1)]^2$ (using $P(k)$) $= 2(k+1)[k(2k+1)/3 + 2(k+1)] = 2(k+1)[k(2k+1) + 6(k+1)]/3 = 2(k+1)[2k^2 + 7k + 6]/3 = 2(k+1)(k+2)(2k+3)/3 = 2(k+1)(k+2)[2(k+1)+1]/3$ which is the right side of $P(k+1)$.
19. Base case: $P(1): 1 \cdot 2 = (1)(2)(3)/3$, (true). Assume $P(k): 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \cdots + k(k+1) = k(k+1)(k+2)/3$. Show $P(k+1): 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \cdots + (k+1)(k+2) = (k+1)(k+2)(k+3)/3$. Left side of $P(k+1) = 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \cdots + (k+1)(k+2) = 1 \cdot 2 + 2 \cdot 3 + 3 \cdot 4 + \cdots + k(k+1) + (k+1)(k+2) = k(k+1)(k+2)/3 + (k+1)(k+2)$ (using $P(k)$) $= (k+1)(k+2)[k/3 + 1] = (k+1)(k+2)(k+3)/3$ which is the right side of $P(k+1)$.
21. Base case: $P(1): 1/(1 \cdot 4) = 1/(3 \cdot 1 + 1)$, (true). Assume $P(k): 1/(1 \cdot 4) + 1/(4 \cdot 7) + 1/(7 \cdot 10) + \cdots + 1/(3k-2)(3k+1) = k/(3k+1)$. Show $P(k+1): 1/(1 \cdot 4) + 1/(4 \cdot 7) + 1/(7 \cdot 10) + \cdots + 1/(3(k+1)-2)(3(k+1)+1) = (k+1)/[3(k+1)+1]$. Left side of $P(k+1) = 1/(1 \cdot 4) + 1/(4 \cdot 7) + 1/(7 \cdot 10) + \cdots + 1/(3(k+1)-2)(3(k+1)+1) = 1/(1 \cdot 4) + 1/(4 \cdot 7) + 1/(7 \cdot 10) + \cdots + 1/(3k-2)(3k+1) + 1/(3(k+1)-2)(3(k+1)+1) = k/(3k+1) + 1/(3(k+1)-2)(3(k+1)+1)$ (using $P(k)$) $= k/(3k+1) + 1/(3k+1)(3k+4) = [k(3k+4) + 1]/(3k+1)(3k+4) = (3k^2 + 4k + 1)/(3k+1)(3k+4) = (3k+1)(k+1)/(3k+1)(3k+4) = (k+1)/[3(k+1)+1]$ which is the right side of $P(k+1)$.
23. Base case: $P(1): 1 + 4 = (4^2 - 1)/3$ or $5 = 15/3$, (true). Assume $P(k): 1 + 4 + 4^2 + \cdots + 4^k = (4^{k+1} - 1)/3$. Show $P(k+1): 1 + 4 + 4^2 + \cdots + 4^{k+1} = (4^{k+2} - 1)/3$. Left side of $P(k+1) = 1 + 4 + 4^2 + \cdots + 4^{k+1} = 1 + 4 + 4^2 + \cdots + 4^k + 4^{k+1} = (4^{k+1} - 1)/3 + 4^{k+1}$ (using $P(k)$) $= (4^{k+1} - 1 + 3 \cdot 4^{k+1})/3 = (4 \cdot 4^{k+1} - 1)/3 = (4^{k+2} - 1)/3$ which is the right side of $P(k+1)$.
25. Base case: $P(1): 1 = (3 - 1)/2$, (true). Assume $P(k): 1 + 4 + 7 + 10 + \cdots + (3k - 2) = k(3k - 1)/2$. Show $P(k+1): 1 + 4 + 7 + 10 + \cdots + (3(k+1) - 2) = (k+1)(3(k+1) - 1)/2$. Left side of $P(k+1) = 1 + 4 + 7 + 10 + \cdots + (3(k+1) - 2) = 1 + 4 + 7 + 10 + \cdots + (3k - 2) + (3(k+1) - 2) = k(3k - 1)/2 + (3(k+1) - 2)$ (using $P(k)$) $= [k(3k - 1) + 2(3(k+1) - 2)]/2 = (3k^2 - k + 6k + 6 - 4)/2 = (3k^2 + 5k + 2)/2 = (k+1)(3k+2)/2 = (k+1)(3(k+1) - 1)/2$ which is the right side of $P(k+1)$.
27. Base case: $P(1): a = (a - ar)/(1 - r) = a(1 - r)/(1 - r)$, (true). Assume $P(k): a + ar + \cdots + ar^{k-1} = (a - ar^k)/(1 - r)$. Show $P(k+1): a + ar + \cdots + ar^k = (a - ar^{k+1})/(1 - r)$. Left side of $P(k+1) = a + ar + \cdots + ar^k = a + ar + \cdots + ar^{k-1} + ar^k = (a - ar^k)/(1 - r) + ar^k$ (using $P(k)$) $= [a - ar^k + ar^k(1 - r)]/(1 - r) = (a - ar^{k+1})/(1 - r)$ which is the right side of $P(k+1)$.
29. a. 4,882,812
b. 64,592,673,600
c. 225
d. 884
31. Base case: $P(2): 2^2 > 2 + 1$, (true). Assume $P(k): k^2 > k + 1$. Show $P(k+1): (k+1)^2 > k+2$. Left side of $P(k+1) = (k+1)^2 = k^2 + 2k + 1 > (k+1) + 2k + 1$ (using $P(k)$) $= 3k + 2 > k + 2$ which is the right side of $P(k+1)$.

- 33.** Base case: $P(7)$: $7^2 > 5 \cdot 7 + 10$ or $49 > 45$, (true). Assume $P(k)$: $k^2 > 5k + 10$. Show $P(k + 1)$: $(k + 1)^2 > 5(k + 1) + 10 = 5k + 15$. Left side of $P(k + 1) = (k + 1)^2 = k^2 + 2k + 1 > (5k + 10) + 2k + 1$ (using $P(k)$) $= 5k + 2k + 11 > 5k + 12 + 11$ (because $k > 6$) $= 5k + 23 > 5k + 15$ which is the right side of $P(k + 1)$.
- 35.** Base case: $P(4)$: $4! > 4^2$ or $1 \cdot 2 \cdot 3 \cdot 4 = 24 > 16$ (true). Assume $P(k)$: $k! > k^2$. Show $P(k + 1)$: $(k + 1)! > (k + 1)^2$. Left side of $P(k + 1) = (k + 1)! = k!(k + 1) > k^2(k + 1)$ (using $P(k)$) $> (k + 1)(k + 1)$ (by Exercise 31 because $k \geq 4$) $= (k + 1)^2$ which is the right side of $P(k + 1)$.
- 37.** Base case: $P(4)$: $4! > 2^4$ or $24 > 16$, (true). Assume $P(k)$: $k! > 2^k$. Show $P(k + 1)$: $(k + 1)! > 2^{k+1}$. Left side of $P(k + 1) = (k + 1)! = k!(k + 1) > 2^k(k + 1)$ (using $P(k)$) $> 2^k(2)$ (because $k \geq 4$) $= 2^{k+1}$ which is the right side of $P(k + 1)$.
- 39.** Base case: $P(1)$: $1! \geq 2^0$, (true). Assume $P(k)$: $k! \geq 2^{k-1}$. Show $P(k + 1)$: $(k + 1)! \geq 2^k$. Left side of $P(k + 1) = (k + 1)! = k!(k + 1) \geq 2^{k-1}(k + 1)$ (using $P(k)$) $\geq 2^{k-1}(2)$ (because $k \geq 1$ so $k + 1 \geq 2$) $= 2^k$ which is the right side of $P(k + 1)$.
- 41.** Base case: $P(2)$: $(1 + x)^2 > 1 + x^2$ or $1 + 2x + x^2 > 1 + x^2$ (true because $x > 0$ implies $2x > 0$). Assume $P(k)$: $(1 + x)^k > 1 + x^k$. Show $P(k + 1)$: $(1 + x)^{k+1} > 1 + x^{k+1}$. Left side of $P(k + 1) = (1 + x)^{k+1} = (1 + x)^k(1 + x) > (1 + x^k)(1 + x)$ (using $P(k)$) $= 1 + x^k + x + x^{k+1} > 1 + x^{k+1}$ which is the right side of $P(k + 1)$.
- 43.** Base case: $P(2)$: $1 + 2 < 2^2$ or $3 < 4$, (true). Assume $P(k)$: $1 + 2 + \cdots + k < k^2$. Show $P(k + 1)$: $1 + 2 + \cdots + (k + 1) < (k + 1)^2$. Left side of $P(k + 1) = 1 + 2 + \cdots + (k + 1) = 1 + 2 + \cdots + k + (k + 1) < k^2 + k + 1$ (using $P(k)$) $< k^2 + 2k + 1 = (k + 1)^2$ which is the right side of $P(k + 1)$.
- 45. a.** Base case: $P(1)$: $1 + (1/2) < 2$, (true). Assume $P(k)$: $1 + (1/2) + \cdots + (1/2^k) < 2$. Show $P(k + 1)$: $1 + (1/2) + \cdots + (1/2^{k+1}) < 2$. Left side of $P(k + 1) = 1 + (1/2) + \cdots + (1/2^{k+1}) = 1 + (1/2) + \cdots + (1/2^k) + (1/2^{k+1}) < 2 + (1/2^{k+1})$ (using $P(k)$), but $2 + (1/2^{k+1})$ is not less than 2.
- b.** Base case: $P(1)$: $1 + (1/2) = 2 - (1/2)$, (true). Assume $P(k)$: $1 + (1/2) + \cdots + (1/2^k) = 2 - (1/2^k)$. Show $P(k + 1)$: $1 + (1/2) + \cdots + (1/2^{k+1}) = 2 - (1/2^{k+1})$. Left side of $P(k + 1) = 1 + (1/2) + \cdots + (1/2^{k+1}) = 1 + (1/2) + \cdots + (1/2^k) + (1/2^{k+1}) = 2 - (1/2^k) + (1/2^{k+1})$ (using $P(k)$) $= 2 - (2/2^{k+1}) + (1/2^{k+1}) = 2 - (1/2^{k+1})$ which is the right side of $P(k + 1)$.
- 47.** Base case: $P(1)$: $2^3 - 1 = 8 - 1 = 7$ and $7|7$. Assume $P(k)$: $7|2^{3k} - 1$ so $2^{3k} - 1 = 7m$ or $2^{3k} = 7m + 1$ for some integer m . Show $P(k + 1)$: $7|2^{3(k+1)} - 1$. $2^{3(k+1)} - 1 = 2^{3k+3} - 1 = 2^{3k} \cdot 2^3 - 1 = (7m + 1)2^3 - 1 = 7(2^3m) + 8 - 1 = 7(2^3m + 1)$ where $2^3m + 1$ is an integer, so $7|2^{3(k+1)} - 1$.
- 49.** Base case: $P(1)$: $7 - 2 = 5$ and $5|5$. Assume $P(k)$: $5|7^k - 2^k$, so $7^k - 2^k = 5m$ or $7^k = 5m + 2^k$ for some integer m . Show $P(k + 1)$: $5|7^{k+1} - 2^{k+1}$. $7^{k+1} - 2^{k+1} = 7 \cdot 7^k - 2 \cdot 2^k = 7(5m + 2^k) - 2^{k+1} = 5(7m) + 2^k(7 - 2) = 5(7m + 2^k)$ where $7m + 2^k$ is an integer, so $5|7^{k+1} - 2^{k+1}$.
- 51.** Base case: $P(1)$: $2 + (-1)^2 = 2 + 1 = 3$ and $3|3$. Assume $P(k)$: $3|2^k + (-1)^{k+1}$, so $2^k + (-1)^{k+1} = 3m$ or $2^k = 3m - (-1)^{k+1}$ for some integer m . Show $P(k + 1)$: $3|2^{k+1} + (-1)^{k+2}$. $2^{k+1} + (-1)^{k+2} = 2 \cdot 2^k + (-1)^{k+2} = 2(3m - (-1)^{k+1}) + (-1)^{k+2} = 3(2m) - 2(-1)^{k+1} + (-1)^{k+2} = 3(2m) + (-1)^{k+1}(-2 + (-1)) = 3(2m) + (-1)^{k+1}(-3) = 3(2m - (-1)^{k+1})$ where $2m - (-1)^{k+1}$ is an integer, so $3|2^{k+1} + (-1)^{k+2}$.
- 53.** Base case: $P(1)$: $3^{4+2} + 5^{2+1} = 3^6 + 5^3 = 729 + 125 = 854 = 61 \cdot 14$ and $14|61 \cdot 14$. Assume $P(k)$: $14|3^{4k+2} + 5^{2k+1}$, so $3^{4k+2} + 5^{2k+1} = 14m$ or $3^{4k+2} = 14m - 5^{2k+1}$ for some integer m . Show $P(k + 1)$: $14|3^{4(k+1)+2} + 5^{2(k+1)+1}$. $3^{4(k+1)+2} + 5^{2(k+1)+1} = 3^{4k+2} \cdot 3^4 + 5^{2k+3} = (14m - 5^{2k+1})3^4 + 5^{2k+3} = 14(m3^4) - 5^{2k+1} \cdot 3^4 + 5^{2k+1} \cdot 5^2 = 14(m3^4) - 5^{2k+1}(3^4 - 5^2) = 14(m3^4) - 5^{2k+1}(81 - 25) = 14(m3^4) - 5^{2k+1}(56) = 14(m3^4 - 4 \cdot 5^{2k+1})$ where $m3^4 - 4 \cdot 5^{2k+1}$ is an integer, so $14|3^{4(k+1)+2} + 5^{2(k+1)+1}$.

55. Base case: $P(1)$: $10 + 3 \cdot 4^3 + 5 = 10 + 192 + 5 = 207 = 9 \cdot 23$ and $9|9 \cdot 23$. Assume $P(k)$: $9|10^k + 3 \cdot 4^{k+2} + 5$, so $10^k + 3 \cdot 4^{k+2} + 5 = 9m$ or $10^k = 9m - 3 \cdot 4^{k+2} - 5$ for some integer m . Show $P(k+1)$: $9|10^{k+1} + 3 \cdot 4^{k+3} + 5$. $10^{k+1} + 3 \cdot 4^{k+3} + 5 = 10 \cdot 10^k + 3 \cdot 4^{k+3} + 5 = 10(9m - 3 \cdot 4^{k+2} - 5) + 3 \cdot 4^{k+3} + 5 = 9(10m) - 30 \cdot 4^{k+2} - 50 + 3 \cdot 4^{k+2} \cdot 4 + 5 = 9(10m) - 45 - 3 \cdot 4^{k+2}(10 - 4) = 9(10m - 5) - 18 \cdot 4^{k+2} = 9(10m - 5 - 2 \cdot 4^{k+2})$ where $10m - 5 - 2 \cdot 4^{k+2}$ is an integer, so $9|10^{k+1} + 3 \cdot 4^{k+3} + 5$.
57. Base case: $P(1)$: $1^3 + 2(1) = 3$ and $3|3$. Assume $P(k)$: $3|k^3 + 2k$ so $k^3 + 2k = 3m$ for some integer m . Show $P(k+1)$: $3|(k+1)^3 + 2(k+1)$. $(k+1)^3 + 2(k+1) = k^3 + 3k^2 + 3k + 1 + 2k + 2 = k^3 + 2k + 3(k^2 + k + 1) = 3m + 3(k^2 + k + 1) = 3(m + k^2 + k + 1)$ where $m + k^2 + k + 1$ is an integer, so $3|(k+1)^3 + 2(k+1)$. This result also follows directly from Exercise 56: $n^3 + 2n = n^3 - n + 3n = 3m + 3n$ (by Exercise 56) $= 3(m+n)$.
59. Base case: $P(1)$: $\cos \theta + i \sin \theta = \cos \theta + i \sin \theta$. Assume $P(k)$: $(\cos \theta + i \sin \theta)^k = \cos k\theta + i \sin k\theta$. Show $P(k+1)$: $(\cos \theta + i \sin \theta)^{k+1} = \cos(k+1)\theta + i \sin(k+1)\theta$. $(\cos \theta + i \sin \theta)^{k+1} = (\cos \theta + i \sin \theta)^k (\cos \theta + i \sin \theta) = (\cos k\theta + i \sin k\theta)(\cos \theta + i \sin \theta) = \cos k\theta \cos \theta + i \sin k\theta \cos \theta + i \cos k\theta \sin \theta + i^2 \sin k\theta \sin \theta = \cos k\theta \cos \theta - \sin k\theta \sin \theta + i(\sin k\theta \cos \theta + \cos k\theta \sin \theta) = \cos(k\theta + \theta) + i \sin(k\theta + \theta) = \cos(k+1)\theta + i \sin(k+1)\theta$.
61. The statement to be proved is that $n(n+1)(n+2)$ is divisible by 3 for $n \geq 1$. Base case: $P(1)$: $1(1+1)(1+2) = 6$ is divisible by 3, (true). Assume $P(k)$: $k(k+1)(k+2) = 3m$ for some integer m . Show $P(k+1)$: $(k+1)(k+2)(k+3)$ is divisible by 3. $(k+1)(k+2)(k+3) = (k+1)(k+2)k + (k+1)(k+2)3 = 3m + (k+1)(k+2)3 = 3[m + (k+1)(k+2)]$.



67. Proof is by induction on n . $P(1)$ is true because 1 line divides the plane into 2 regions, and $(1^2 + 1 + 2)/2 = 2$. Assume that $P(k)$ is true: k lines divide the plane into $(k^2 + k + 2)/2$ regions. Show $P(k+1)$, that $k+1$ lines divide the plane into $[(k+1)^2 + (k+1) + 2]/2$ regions. A new line creates one more region than the number of lines it crosses. When line $k+1$ is added, it will cross k lines (because no two lines are parallel and have no common intersection points). Therefore $k+1$ new regions are created. The total number of regions is therefore $k+1$ more than the number present with k lines, or $(k^2 + k + 2)/2 + (k+1) = (k^2 + k + 2 + 2(k+1))/2 = (k^2 + 3k + 4)/2 = ((k+1)^2 + (k+1) + 2)/2$.
69. $P(1)$ is $1 = 1 + 1$ which is not true.
71. a. Let $P(n)$ be the property that any word composed of a juxtaposition of n subwords has an even number of o's. Then $P(1)$ is true because the only words with 1 subword are the words moon, noon, and soon, all of which have 2 o's. Assume that $P(k)$ is true and consider $P(k+1)$. For any word composed of $k+1$ subwords, break the word into two parts composed of k subwords and 1 subword. By the inductive hypothesis, the part with k subwords has an even number m of o's. The part with 1 subword has 2 o's. The total number of o's is therefore $m+2$, an even number. This verifies $P(k+1)$ and completes the proof.

- b. Let $P(n)$ be the property that any word composed of a juxtaposition of n subwords has an even number of o's. Then $P(1)$ is true because the only words with 1 subword are the words *moon*, *noon*, and *soon*, all of which have 2 o's. Assume that $P(r)$ is true for all r , $1 \leq r \leq k$, and consider $P(k+1)$. For any word composed of $k+1$ subwords, break the word into two parts composed of r_1 and r_2 subwords, with $1 \leq r_1 \leq k$, $1 \leq r_2 \leq k$, and $r_1 + r_2 = k+1$. By the inductive hypothesis, r_1 contains m_1 o's, an even number, and r_2 contains m_2 o's, an even number. Then the original word contains $m_1 + m_2$ o's, an even number. This verifies $P(k+1)$ and completes the proof.
73. For the base case, a 1-piece puzzle requires 0 steps to assemble. Assume that any block of r pieces, $1 \leq r \leq k$, requires $r-1$ steps to assemble. Now consider a puzzle with $k+1$ pieces. The last step in assembling the puzzle is to fit together two blocks of size r_1 and r_2 with $1 \leq r_1 \leq k$, $1 \leq r_2 \leq k$, and $r_1 + r_2 = k+1$. By the inductive hypothesis these blocks required r_1-1 and r_2-1 steps to assemble, so with the final step, the total number of steps required is $(r_1-1) + (r_2-1) + 1 = (r_1+r_2) - 1 = k$.
75. For the base case, the simplest such wff is a single statement letter, which has 1 symbol; 1 is odd. Assume that for any such wff with r symbols, $1 \leq r \leq k$, r is odd. Consider a wff with $k+1$ symbols. It must have the form $(P) \wedge (Q)$, $(P) \vee (Q)$, or $(P) \rightarrow (Q)$ where P has r_1 symbols, $1 \leq r_1 < k$, and Q has r_2 symbols, $1 \leq r_2 < k$. By the inductive hypothesis, both r_1 and r_2 are odd. The number of symbols in the original wff is then $r_1 + r_2 + 5$ (four parentheses plus one connective), which is odd.
77. $P(2)$ and $P(3)$ are true by the equations $2 = 2$ and $3 = 3$. Now assume that $P(r)$ is true for any r , $2 \leq r \leq k$, and consider $P(k+1)$. We may assume that $k+1 \geq 4$, so that $(k+1) - 2 \geq 2$ and by the inductive hypothesis can be written as a sum of $2s$ and $3s$. Adding an additional 2 gives $k+1$ as a sum of $2s$ and $3s$.
79. $P(14)$, $P(15)$, and $P(16)$ are true by the equations $14 = 2(3) + 8$, $15 = 5(3)$, $16 = 2(8)$. Now assume that $P(r)$ is true for any r , $14 \leq r \leq k$, and consider $P(k+1)$. We may assume that $k+1 \geq 17$, so $(k+1) - 3 \geq 14$ and by the inductive hypothesis can be written as a sum of $3s$ and $8s$. Adding an additional 3 gives $k+1$ as a sum of $3s$ and $8s$.
81. $P(64)$, $P(65)$, $P(66)$, $P(67)$, and $P(68)$ are true by the equations $64 = 6(5) + 2(17)$, $65 = 13(5)$, $66 = 3(5) + 3(17)$, $67 = 10(5) + 17$, $68 = 4(17)$. Now assume that $P(r)$ is true for any r , $64 \leq r \leq k$, and consider $P(k+1)$. We may assume that $k+1 \geq 69$, so $(k+1) - 5 \geq 64$ and by the inductive hypothesis can be written as a sum of $5s$ and $17s$. Adding an additional 5 gives $k+1$ as a sum of $5s$ and $17s$.
83. From Exercise 2,

$$\sum_{m=1}^n 2m = n(n+1) = n^2 + n. \text{ Also, } \int_0^n 2x \, dx = \left. \frac{2x^2}{2} \right|_0^n = n^2$$

and

$$\int_1^{n+1} 2x \, dx = \left. \frac{2x^2}{2} \right|_1^{n+1} = (n+1)^2 - 1 = n^2 + 2n. \text{ It is true that } n^2 \leq n^2 + n \leq n^2 + 2n.$$

EXERCISES 2.3

1. Assume $x_k^2 > x_k + 1$. Then $x_{k+1}^2 = (x_k + 1)^2 = x_k^2 + 2x_k + 1 > x_k^2 + 1 > (x_k + 1) + 1 = x_{k+1} + 1$.
3. $Q(0): j_0 = (i_0 - 1)!$ because $j = 1, i = 2$ before loop is entered. Assume $Q(k): j_k = (i_k - 1)!$. Then $Q(k+1): j_{k+1} = j_k \cdot i_k = (i_k - 1)! i_k = (i_k)! = (i_{k+1} - 1)!$. At loop termination, $j = (i - 1)!$ and $i = x + 1$, so $j = x!$
5. $Q(0): j_0 = x^{i_0}$ because $j = x, i = 1$ before loop is entered. Assume $Q(k): j_k = x^{i_k}$. Then $Q(k+1): j_{k+1} = j_k \cdot x = x^{i_k} \cdot x = x^{i_k+1} = x^{i_{k+1}}$. At loop termination, $j = x^i$ and $i = y$, so $j = x^y$.

7. $\gcd(308, 165) = 11$
9. $\gcd(735, 90) = 15$
11. $\gcd(1326, 252) = 6$
13. You want to divide the 792 bars of soap evenly among x packages. Therefore x must be a divisor of 792. Similarly, you want to divide the 400 shampoo bottles evenly among the x packages. Therefore x must be a divisor of 400. The number of packages is the largest value of x that divides both 792 and 400, which is the definition of $\gcd(792, 400)$.
15. $Q: j = x * y^j$. $Q(0): j_0 = x * y^{j_0}$ because $j = x, i = 0$ before loop is entered. Assume $Q(k): j_k = x * y^{j_k}$. Then $Q(k + 1): j_{k+1} = j_k * y = x * y^{j_k} * y = x * y^{j_k+1} = x * y^{j_{k+1}}$. At loop termination, $j = x * y^j$ and $i = n$, so $j = x * y^n$.
17. $Q: j = (i + 1)^2$. $Q(0): j_0 = (i_0 + 1)^2$ because $j = 4, i = 1$ before loop is entered. Assume $Q(k): j_k = (i_k + 1)^2$. Then $Q(k + 1): j_{k+1} = j_k + 2i_k + 3 = (i_k + 1)^2 + 2i_k + 3 = i_k^2 + 2i_k + 1 + 2i_k + 3 = i_k^2 + 4i_k + 4 = (i_k + 2)^2 = (i_k + 1 + 1)^2 = (i_{k+1} + 1)^2$. At loop termination, $j = (i + 1)^2$ and $i = x$, so $j = (x + 1)^2$.
19. $Q: j = x * i!$. $Q(0): j_0 = x * i_0!$ because $j = x, i = 1$ before loop is entered. Assume $Q(k): j_k = x * (i_k)!$. Then $Q(k + 1): j_{k+1} = j_k * (i_k + 1) = x * (i_k)!(i_k + 1) = x * (i_k + 1)! = x * (i_{k+1})!$. At loop termination, $j = x * i!$ and $i = n$, so $j = x * n!$
21. $Q: j = \max(a[1], \dots, a[i])$. $Q(0): j_0 = \max(a[1], \dots, a[i_0])$ because $i_0 = 1$, so the right side becomes $\max(a[1])$ and $j_0 = a[1]$. Assume $Q(k): j_k = \max(a[1], \dots, a[i_k])$. Then $Q(k + 1): j_{k+1} = \max(j_k, a[i_k + 1]) = \max(\max(a[1], \dots, a[i_k]), a[i_k + 1]) = \max(a[1], \dots, a[i_k + 1]) = \max(a[1], \dots, a[i_{k+1}])$. At loop termination, $j = \max(a[1], \dots, a[i])$ and $i = n$, so $j = \max(a[1], \dots, a[n])$.
23. Suppose there exists an integer d such that $d|a/2, d|b/2$, and $d > c$, from which $2d > 2c$. Then $a/2 = k_1d$ and $b/2 = k_2d$ where k_1, k_2 are integers. Then $a = k_1(2d)$ and $b = k_2(2d)$, which means that $2d$ divides both a and b but is greater than $2c = \gcd(a, b)$, which is a contradiction.
25. If a and b are both odd, then $a - b$ is even, in which case by fact 2, $\gcd(a - b, b) = \gcd((a - b)/2, b)$
27.

308	165	Fact 2
154	165	Fact 2
77	165	Fact 3
77	44	Fact 2
77	22	Fact 2
77	11	Fact 3
33	11	Fact 3
11	11	Fact 3
0	11	

 $\gcd(308, 165) = 11$

EXERCISES 2.4

1. $11 = 7 \cdot 308 - 13 \cdot 165$
3. $15 = 1 \cdot 735 - 8 \cdot 90$
5. $6 = 100 \cdot 252 - 19 \cdot 1326$
7. $1729 = 7 \cdot 13 \cdot 19$
9. Because $\sqrt{1171} \cong 34$, we try the primes 2, 3, 5, 7, 11, 13, 17, 23, 29, 31. None of these divide n , so n is prime.
11. $8712 = 2^3 \cdot 3^2 \cdot 11^2$

13. $308 = 2^2 \cdot 7 \cdot 11$ and $165 = 3 \cdot 5 \cdot 11$, so $\gcd(308, 165) = 11$
15. $735 = 3 \cdot 5 \cdot 7^2$ and $90 = 2 \cdot 3^2 \cdot 5$ so $\gcd(735, 90) = 3 \cdot 5 = 15$
17. $1326 = 2 \cdot 3 \cdot 13 \cdot 17$ and $252 = 2^2 \cdot 3^2 \cdot 7$, so $\gcd(1326, 252) = 2 \cdot 3 = 6$
19. The $\gcd(a, b)$ is the product of primes that appear in both a and b to the lowest power to which they appear. The $\text{lcm}(a, b)$ is the product of primes that appear in either a or b to the highest power to which they appear.
21. $\gcd = 2 \cdot 3 \cdot 11$, $\text{lcm} = 2^2 \cdot 3 \cdot 11^2 \cdot 13$.
23. $\gcd = 3 \cdot 5 \cdot 11$, $\text{lcm} = 3^2 \cdot 5^3 \cdot 11^2 \cdot 17$
25. Let $\gcd(a, b) = c$ and $\gcd(a, a + b) = d$. Because $\gcd(a, b) = c$, $c|a$ and $c|b$, so $a = mc$ and $b = nc$ for some integers m and n . Then $a + b = mc + nc = (m + n)c$, so $c|(a + b)$. Therefore c is a common divisor of a and $a + b$, and must be \leq the greatest common divisor of a and $a + b$, namely d . Because $\gcd(a, a + b) = d$, $d|a$ and $d|(a + b)$, so $a = id$ and $a + b = jd$ for some integers i and j . Then $b = jd - a = jd - id = (j - i)d$, so $d|b$. Therefore d is a common divisor of a and b and must be \leq the greatest common divisor of a and b , namely c . Now $c \leq d$ and $d \leq c$, so $c = d$.
27. For example, $8|24$ and $24 = 12 \cdot 2$ but $8 \nmid 12$ and $8 \nmid 2$. This does not violate the theorem of division by prime numbers because 8 is not prime.
29. 3, 5, 7
31. $\varphi(8) = 4$ (the numbers 1, 3, 5, 7)
33. $\varphi(10) = 4$ (the numbers 1, 3, 7, 9)
35. Let $\varphi(n) = n - 1$. Because n is never relatively prime to n , the numbers being counted in $\varphi(n)$ are $(1, 2, \dots, n - 1)$. Thus every number between 1 and $n - 1$ is relatively prime to n , so only 1 and n divide n , making n prime.
37. $\varphi(2^4) = 2^3 \varphi(2) = 8 \cdot 1 = 8$. The numbers are: 1, 3, 5, 7, 9, 11, 13, 15
39. 35640
41. 1248000
43. To compute $\varphi(pq)$, count the number of positive integers less than or equal to pq , which is pq , and throw out those that are not relatively prime to pq . A positive integer m less than or equal to pq and not relatively prime to pq must either contain at least one factor of p or at least one factor of q . We count how many integral multiples of p ($p, 2p, 3p, \dots, pq$) are less than or equal to pq . This number is $pq/p = q$. Similarly we count how many integral multiples of q ($q, 2q, 3q, \dots, pq$) are less than or equal to pq , of which there are $pq/q = p$. Multiples of p and multiples of q are distinct except for pq , which we have counted twice; to compensate, we will add 1 to the final count. The correct expression is $\varphi(pq) = pq - q - p + 1 = (p - 1)(q - 1) = \varphi(p)\varphi(q)$.
45. Using the fundamental theorem of arithmetic, let $n = p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}$ so that $n^m = p_1^{n_1 m} p_2^{n_2 m} \cdots p_k^{n_k m}$ and by Equation 2,

$$\phi(n) = p_1^{n_1 - 1} p_2^{n_2 - 1} \cdots p_k^{n_k - 1} [\phi(p_1)\phi(p_2) \cdots \phi(p_k)]$$

Then, again by Equation 2,

$$\begin{aligned} \varphi(n^m) &= p_1^{n_1 m - 1} p_2^{n_2 m - 1} \cdots p_k^{n_k m - 1} \varphi(p_1)\varphi(p_2) \cdots \varphi(p_k) = \frac{p_1^{n_1 m} p_2^{n_2 m} \cdots p_k^{n_k m}}{p_1 p_2 \cdots p_k} \varphi(p_1)\varphi(p_2) \cdots \varphi(p_k) \\ &= \frac{p_1^{n_1 m} p_2^{n_2 m} \cdots p_k^{n_k m}}{p_1 p_2 \cdots p_k} \left(\frac{\varphi(n)}{p_1^{n_1 - 1} p_2^{n_2 - 1} \cdots p_k^{n_k - 1}} \right) = \frac{p_1^{n_1 m} p_2^{n_2 m} \cdots p_k^{n_k m}}{p_1^{n_1} p_2^{n_2} \cdots p_k^{n_k}} \varphi(n) \\ &= \frac{n^m}{n} \varphi(n) = n^{m-1} \varphi(n) \end{aligned}$$

47. 5, 7, 31, 127

49. a. $28 = 1 + 2 + 4 + 7 + 14$

b. $28 = 2^2(2^3 - 1)$

51. The original list is

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	

After pass one (crossing out multiples of 2):

1	2	3	5	7	9
11		13	15	17	19
21		23	25	27	29
31		33	35	37	39
41		43	45	47	49
51		53	55	57	59
61		63	65	67	69
71		73	75	77	79
81		83	85	87	89
91		93	95	97	99

After pass two (crossing out multiples of 3):

1	2	3	5	7	
11		13		17	19
		23	25		29
31			35	37	
41		43		47	49
		53	55		59
61			65	67	
71		73		77	79
		83	85		89
91			95	97	

After pass three (crossing out multiples of 5):

1	2	3	5	7	
11		13		17	19
		23			29
31				37	
41		43		47	49
		53			59
61				67	
71		73		77	79
		83			89
91				97	

After pass four (crossing out multiples of 7):

1	2	3	5	7	
11		13		17	19
		23			29
31				37	
41		43		47	
		53			59
61				67	
71		73			79
		83			89
				97	

Because 7 is the largest prime less than $\sqrt{100}$, the process terminates. The remaining numbers (excluding 1) are the primes less than 100.

53.

8	9	5	2	1	7	6	4	3
6	4	7	5	8	3	9	1	2
2	3	1	6	9	4	5	8	7
9	2	4	1	3	5	8	7	6
1	5	8	7	2	6	4	3	9
7	6	3	9	4	8	1	2	5
4	8	2	3	5	9	7	6	1
3	7	9	4	6	1	2	5	8
5	1	6	8	7	2	3	9	4

CHAPTER 3

EXERCISES 3.1

1. 10, 20, 30, 40, 50
3. 2, $1/2$, 2, $1/2$, 2
5. $1, 1 + \frac{1}{2}, 1 + \frac{1}{2} + \frac{1}{3}, 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4}, 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5}$
7. 1, 5, 47, 755, 18879
9. 2, 2, 6, 14, 34
11. 2, 3, 6, 18, 108
13. $F(n+1) + F(n-2) = F(n-1) + F(n) + F(n-2) = [F(n-2) + F(n-1)] + F(n) = F(n) + F(n) = 2F(n)$.
15. $F(n) = F(n-2) + F(n-1) = [F(n-4) + F(n-3)] + [F(n-3) + F(n-2)] = F(n-4) + 2F(n-3) + F(n-2) = F(n-4) + 2F(n-3) + [F(n-4) + F(n-3)] = 3F(n-3) + 2F(n-4)$.
17. $F(n+3) = F(n+2) + F(n+1) = F(n+1) + F(n) + F(n+1) = 2F(n+1) + F(n)$.
19. $n = 1$: $F(1) = F(3) - 1$ or $1 = 2 - 1$, true. Assume true for $n = k$: $F(1) + \cdots + F(k) = F(k+2) - 1$. Then $F(1) + \cdots + F(k+1) = F(1) + \cdots + F(k) + F(k+1) = F(k+2) - 1 + F(k+1) = F(k+3) - 1$.

21. $n = 1$: $F(1) = F(2)$ or $1 = 1$, true. Assume true for $n = k$: $F(1) + F(3) + \cdots + F(2k - 1) = F(2k)$. Then $F(1) + F(3) + \cdots + F(2(k + 1) - 1) = F(1) + F(3) + \cdots + F(2k - 1) + F(2(k + 1) - 1) = F(2k) + F(2(k + 1) - 1) = F(2k) + F(2k + 1) = F(2k + 2) = F(2(k + 1))$.
23. $n = 1$: $F(4) = 2F(2) + F(1)$ or $3 = 2(1) + 1$, true. $n = 2$: $F(5) = 2F(3) + F(2)$ or $5 = 2(2) + 1$, true. Assume for all r , $1 \leq r \leq k$, $F(r + 3) = 2F(r + 1) + F(r)$. Then $F(k + 4) = F(k + 2) + F(k + 3) = 2F(k) + F(k - 1) + 2F(k + 1) + F(k) = 2[F(k) + F(k + 1)] + [F(k - 1) + F(k)] = 2F(k + 2) + F(k + 1)$.
25. $n = 1$: $F(1) < 2$ or $1 < 2$, true. $n = 2$: $F(2) < 2^2$ or $1 < 4$, true. Assume for all r , $1 \leq r \leq k$, $F(r) < 2^r$. Then $F(k + 1) = F(k - 1) + F(k) < 2^{k-1} + 2^k = 2^{k-1}(1 + 2) = 3(2^{k-1}) < 4(2^{k-1}) = 2^2(2^{k-1}) = 2^{k+1}$.

27. F (positive integer n)
 //function that recursively computes the value of
 //the n th Fibonacci number
if $n = 1$ **then**
 return 1
else
 if $n = 2$ **then**
 return 1
 else
 return $F(n - 2) + F(n - 1)$
 end if
end if
end function F

29. a. $i = n$
 b. $p = F(n - 1)$

31. a. $p^2 = \frac{(1 + \sqrt{5})^2}{2^2} = \frac{1 + 2\sqrt{5} + 5}{2} = \frac{6 + 2\sqrt{5}}{4} = \frac{3 + \sqrt{5}}{2} = \frac{2}{2} + \frac{1 + \sqrt{5}}{2} = 1 + p$

The proof that $1 + q = q^2$ is similar.

b. $n = 1$: $F(1) = \frac{p - q}{p - q} = 1$, true.

$$n = 2: F(2) = \frac{p^2 - q^2}{p - q} = \frac{(p - q)(p + q)}{p - q} = p + q = \frac{1 + \sqrt{5}}{2} + \frac{1 - \sqrt{5}}{2} = \frac{2}{2} = 1, \text{ true.}$$

Assume for all r , $1 \leq r \leq k$, $F(r) = \frac{p^r - q^r}{p - q}$. Then

$$\begin{aligned} F(k + 1) = F(k - 1) + F(k) &= \frac{p^{k-1} - q^{k-1}}{p - q} + \frac{p^k - q^k}{p - q} = \frac{p^{k-1}(1 + p) - q^{k-1}(1 + q)}{p - q} \\ &= \frac{p^{k-1}p^2 - q^{k-1}q^2}{p - q} = \frac{p^{k+1} - q^{k+1}}{p - q} \end{aligned}$$

c. From part (b),

$$\begin{aligned}
 F(n) &= \frac{p^n - q^n}{p - q} = \frac{\left(\frac{1 + \sqrt{5}}{2}\right)^n - \left(\frac{1 - \sqrt{5}}{2}\right)^n}{\left(\frac{1 + \sqrt{5}}{2}\right) - \left(\frac{1 - \sqrt{5}}{2}\right)} = \frac{2(1 + \sqrt{5})^n}{2^n(2\sqrt{5})} - \frac{2(1 - \sqrt{5})^n}{2^n(2\sqrt{5})} \\
 &= \frac{1}{\sqrt{5}}\left(\frac{1 + \sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1 - \sqrt{5}}{2}\right)^n = \frac{\sqrt{5}}{5}\left(\frac{1 + \sqrt{5}}{2}\right)^n - \frac{\sqrt{5}}{5}\left(\frac{1 - \sqrt{5}}{2}\right)^n
 \end{aligned}$$

33. Yes.

35. No.

37. a. $F(1) = F(2) = 1$ because the rabbits do not breed until they are 2 months old. $F(n)$ = the number of pairs at the end of n months = (the number of pairs at the end of $n - 1$ months) + (the number of offspring pairs produced during month n , born to the pairs existing at month $n - 2$) = $F(n - 1) + F(n - 2)$

b. $27 = 1 + 5 + 21$, $62 = 2 + 5 + 55$

39. Base cases: $S(1)$, $S(2)$, $S(3)$ are even. Assume for all r , $1 \leq r \leq k$, $S(r)$ is even. Then $S(k + 1) = 3S(k - 2) = 3(\text{even}) = \text{even}$

41. a. $n = 0$: $S(0) = 1$ is odd, $n = 1$: $S(1) = 1$ is odd. Assume for all r , $0 \leq r \leq k$, $S(r)$ is odd. Then $S(k + 1) = 2S(k) + S(k - 1) = 2(\text{odd}) + \text{odd} = \text{even} + \text{odd} = \text{odd}$

b. $n = 4$: $S(4) < 6S(2)$ or $17 < 6(3) = 18$, true. $n = 5$: $S(5) < 6S(3)$ or $41 < 6(7) = 42$, true. Assume for all r , $4 \leq r \leq k$, $S(r) < 6S(r - 2)$. Then $S(k + 1) = 2S(k) + S(k - 1) < 2[6S(k - 2)] + 6S(k - 3) = 6[2S(k - 2) + S(k - 3)] = 6S(k - 1)$

43. $S(1) = a$, $S(n) = rS(n - 1)$ for $n \geq 2$

45. a. $A(1) = 50,000$, $A(n) = 3A(n - 1)$ for $n \geq 2$

b. 4.

47. b and c

49. a , b , and e

51. The basis elements are 0 and 3, each of which is a multiple of 3. Assume that x and y are integers that are multiples of 3, so $x = (n)3$ and $y = (k)3$. Then $x + y = (n)3 + (k)3 = (n + k)3$, which is a multiple of 3.

53. 1. Any unary predicate in x is a wff. 2. If P and Q are unary predicate wffs in x , so are $(P \wedge Q)$, $(P \vee Q)$, $(P \rightarrow Q)$, (P') , $(P \leftrightarrow Q)$, $(\forall x)P$, and $(\exists x)P$.

55. (1) The integer 1 belongs to the set. (2) If x is an odd integer, so are $x + 2$ and $x - 2$.

57. (1) The string 0 belongs to the set. (2) If x is a binary string with an odd number of 0s, so is $1x$, $x1$, and $0x0$.

59. (1) The string 0 belongs to the set. (2) If x is a binary string ending with 0, so is $1x$, $0x$, $x0$.

61. $\langle \text{positive digit} \rangle ::= 1|2|3|4|5|6|7|8|9$, $\langle \text{digit} \rangle ::= 0|\langle \text{positive digit} \rangle$, $\langle \text{positive integer} \rangle ::= \langle \text{positive digit} \rangle|\langle \text{positive integer} \rangle\langle \text{digit} \rangle$

63. (1) $\lambda^R = \lambda$ (2) If x is a string with a single character, $x^R = x$. (3) If $x = yz$, then $x^R = z^R y^R$

65. $1! = 1$, $n! = n(n - 1)!$ for $n \geq 2$

67. a. $\max(a_1, a_2) = \begin{cases} a_1 & \text{if } a_1 \geq a_2 \\ a_2 & \text{if } a_1 < a_2 \end{cases}$ $\max(a_1, \dots, a_n) = \max(\max(a_1, \dots, a_{n-1}), a_n)$ for $n > 2$

b. $\min(a_1, a_2) = \begin{cases} a_1 & \text{if } a_1 \leq a_2 \\ a_2 & \text{if } a_1 > a_2 \end{cases}$ $\min(a_1, \dots, a_n) = \min(\min(a_1, \dots, a_{n-1}), a_n)$ for $n > 2$

69. $A \vee (B_1 \wedge B_2) \Leftrightarrow (A \vee B_1) \wedge (A \vee B_2)$ by equivalence 3a. Assume that $A \vee (B_1 \wedge \cdots \wedge B_k) \Leftrightarrow (A \vee B_1) \wedge \cdots \wedge (A \vee B_k)$. Then $A \vee (B_1 \wedge \cdots \wedge B_{k+1}) = A \vee [(B_1 \wedge \cdots \wedge B_k) \wedge B_{k+1}]$ by Exercise 68 $\Leftrightarrow (A \vee (B_1 \wedge \cdots \wedge B_k)) \wedge (A \vee B_{k+1}) \Leftrightarrow [(A \vee B_1) \wedge \cdots \wedge (A \vee B_k)] \wedge (A \vee B_{k+1}) \Leftrightarrow (A \vee B_1) \wedge \cdots \wedge (A \vee B_{k+1})$ by Exercise 68. The proof of the other statement is similar.
71. **if** $n = 1$ **then**
 return 1
else
 return $3 * S(n - 1)$
end if
73. **if** $n = 1$ **then**
 return 1
else
 return $S(n - 1) + (n - 1)$
end if
75. **if** $n = 1$ **then**
 return a
else
 if $n = 2$ **then**
 return b
 else
 return $S(n - 2) + S(n - 1)$
 end if
end if
77. $\text{Mystery}(n) = n$
79. If the list has 1 element or 0 elements, then we are done; else exchange the first and last element in the list and invoke the algorithm on the list minus its first and last elements.
81. Divide a by b . If the remainder r is 0, then $\text{gcd}(a, b) = b$; else invoke the algorithm on b and r instead of a and b .
83. 4, 10, -6, 2, 5; 4, 5, -6, 2, 10; 4, 2, -6, 5, 10; -6, 2, 4, 5, 10
85. New Orleans, Charlotte, Indianapolis
87. Q : $\text{CurrentValue} = 2^{i-1}$. $Q(0)$: $\text{CurrentValue}_0 = 2^{i_0-1}$ true because $\text{CurrentValue}_0 = 2$ and $i_0 = 2$ and $2 = 2^{2-1}$. Assume $Q(k)$: $\text{CurrentValue}_k = 2^{i_k-1}$. Then $\text{CurrentValue}_{k+1} = 2 * \text{CurrentValue}_k = 2(2^{i_k-1}) = 2^{i_k} = 2^{i_{k+1}-1}$. At termination, $\text{CurrentValue} = 2^{i-1}$ and $i = n + 1$, so $\text{CurrentValue} = 2^{n+1-1} = 2^n$.

EXERCISES 3.2

1. $S(n) = n(5)$
3. $F(n) = n2^n$
5. $A(n) = n(n + 1)/2$
7. $T(n) = \frac{n(n + 1)(2n + 1)}{6}$
9. The solution formula does not apply. Using expand, guess, and verify, $F(n) = n!$
11. The solution formula does not apply. Using expand, guess, and verify, $A(n) = 2^{n-1}(n - 1)!$
13. a. The recurrence relation is $T(n) = 0.95T(n - 1)$ with a base case $T(1) = X$.
 b. $T(n) = (0.95)^{n-1}(X)$.
 c. $T(21) = 0.358(X)$, which is slightly more than one-third the original amount X .

15. a. The recurrence relation is $S(n) = 10S(n - 1)$ with a base case of $S(1) = 1000$.
 b. $S(n) = 10^{n+2}$
 c. At the end of 20 seconds, (the beginning of the 21st second), $S(21) = 10^{23}$ e-mail messages are sent.
17. a. The recurrence relation is $A(n) = (1.01)A(n - 1) - 80$ with a base case of $A(1) = 5000$.
 b. $A(n) = (1.01)^{n-1}(5000) - 80[1 - (1.01)^{n-1}]/[1 - 1.01]$
 c. $A(19) = \$4411.56$
19. a. The recurrence relation is $S(n) = 0.98S(n - 1) - 10,000$ with a base case of $S(1) = 1,000,000$.
 b. $S(n) = (0.98)^{n-1}(1,000,000) - 10,000[1 - (0.98)^{n-1}]/(1 - 0.98)$
 c. $S(10) = 750622$
21. a. The recurrence relation for the total number of infected machines each day is $T(n) = 6T(n - 1) - 6^{n-2}$ with a base case of $T(1) = 3$.
 b. $T(n) = 6^{n-2}[6 * 3 - (n - 1)]$
 c. The virus disappears after 19 days.
23. The recurrence relation is $P(n) = P(n - 1) + n$, with $P(1) = 1$. The solution is $P(n) = n(n + 1)/2$.
25. The recurrence relation is $P(n) = P(n - 1) + 3n - 2$ with $P(1) = 1$. The solution is $P(n) = (n/2)(3n - 1)$.
27. $T(n) = 4(2)^{n-1} + (3)^{n-1}$
29. $S(n) = 2 + 2(-2)^{n-1}$
31. $F(n) = 6 + 2(5)^{n-1}$
33. $B(n) = 3(2)^{n-1} + 4(n - 1)(2)^{n-1}$
35. $A(n) = 4(1 + i)^{n-1} + 4(1 - i)^{n-1}$
37. The characteristic equation is $t^2 - t - 1 = 0$ with roots

$$r_1 = \frac{1 + \sqrt{5}}{2}, r_2 = \frac{1 - \sqrt{5}}{2}$$

The solution is

$$F(n) = \frac{\sqrt{5}}{5} \left(\frac{1 + \sqrt{5}}{2} \right)^n - \frac{\sqrt{5}}{5} \left(\frac{1 - \sqrt{5}}{2} \right)^n$$

which agrees with the expression given in Exercise 31(c) of Section 3.1.

39. a. The recurrence relation is $M(n) - M(n - 1) = (1/2)[M(n - 1) - M(n - 2)]$ for $n \geq 3$, with $M(1) = 200,000$, $M(2) = 250,000$. The solution is $M(n) = 300,000 + (-100,000)(1/2)^{n-1}$.
 b. $M(7) = 298437$, which is within \$2000 of \$300,000.
41. Let $S(n)$ be the number of binary strings of length n with no two consecutive 0s. Such strings can be generated in two ways: (i) Put a 1 on the end of a string of length $n - 1$ that has no two consecutive 0s. There are $S(n - 1)$ of these strings. (ii) Put a 10 on the end of a string of length $n - 2$ that has no two consecutive 0s. There are $S(n - 2)$ of these strings. Therefore $S(n) = S(n - 1) + S(n - 2)$, which is the recurrence relation for the Fibonacci numbers. Also $S(1) = 2$ (both 0 and 1 are 1-length binary strings with no two consecutive 0s), $S(2) = 3$ (01, 10, 11 are 2-length binary strings with no two consecutive 0s). So $S(1) = 2 = F(3)$, $S(2) = 3 = F(4)$, $S(3) = S(2) + S(1) = 5 = F(5)$, etc.
43. Here $c_2 = 0$ so the characteristic equation is $t^2 - c_1t = 0$, which has roots $r_1 = 0$, $r_2 = c_1$. The solution is $S(n) = p(0)^{n-1} + q(c_1)^{n-1} = q(c_1)^{n-1}$ where $p + q = S(1)$, $q(c_1) = S(2)$, so $q = S(2)/c_1$. By the recurrence relation $S(n) = c_1S(n - 1)$, $S(2)/c_1 = S(1)$, so $q = S(1)$. The solution is $S(n) = S(1)(c_1)^{n-1}$, which is the solution given by Equation 8 because $g(n) = 0$.
45. $P(n) = 4n - 3$
47. $S(n) = (1 + \log n)n$

EXERCISES 3.3

1. **for** $i = 1$ to n **do**
 $low = roster[i].quiz[1]$
 $high = roster[i].quiz[1]$
 $sum = roster[i].quiz[1]$
for $j = 2$ to m **do**
 $sum = sum + roster[i].quiz[j]$ //A
if $roster[i].quiz[j] < low$ **then**
 $low = roster[i].quiz[j]$
end if
if $roster[i].quiz[j] > high$ **then**
 $high = roster[i].quiz[j]$
end if
end for
 $sum = sum - low$ //S
 $sum = sum + high$ //A
write("Total for student", i , "is", sum)
end for
3. A total of n^2 additions is done.
5. The overall number of output statements is $n(\log n)$
7. **a.** $factorial(integer\ n)$
integer i
 $factorial = 1$
if $n = 1$ **then**
return 1
else
for $i = 1$ to $n - 1$ **do**
 $factorial = factorial * (i + 1)$
end for
return $factorial$
end if
- b.** $n - 1$ multiplications are done.
9. **a.** c has the value 4. When $i = 1$, product = $1 * 4 = 4$, sum = $-14 + 5 * 4 = 6$. When $i = 2$, product = $4 * 4 = 16$, sum = $6 + (-7) * 16 = -106$. When $i = 3$, product = $16 * 4 = 64$, sum = $-106 + 2 * 64 = 22$, so 22 is the final value, which is correct.
- b.** The total work is $3n$.
11. The best case occurs when the first quiz grade is the lowest quiz grade for each student. Then the condition of the **if** statement is never true and the assignment statement within the **if** statement executes 0 times. The worst case occurs when all the quiz grades go downhill from beginning to end for each student. Then each new quiz grade is lower than the previous one, so the assignment statement within the **if** statement executes each time, or $n(m - 1)$ times. Total assignments and comparisons in the best case = $3n + 2n(m - 1)$ and in the worst case = $3n + 3n(m - 1)$.
13. **a.** After pass 1 the list is 5, 3, 4, 6, 2, 8. After pass 2 the list is 3, 4, 5, 2, 6, 8. After pass 3 the list is 3, 4, 2, 5, 6, 8. After pass 4 the list is 3, 2, 4, 5, 6, 8. After pass 5 the list is 2, 3, 4, 5, 6, 8.
- b.** $B(1) = 0$, $B(n) = (n - 1) + B(n - 1)$ for $n \geq 2$
- c.** $B(n) = (n - 1)n/2$
15. $n - 1$ compares are always needed - every element after the first must be considered a potential new maximum.

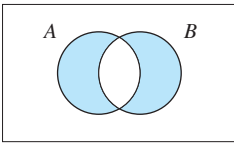
17. $S(n) = (n - 1)n/2$
19. a. The merged list is 1, 4, 5, 6, 8, 9; 3 comparisons
 b. The merged list is 1, 2, 3, 4, 5, 8; 4 comparisons
 c. The merged list is 0, 1, 2, 3, 4, 7, 8, 9, 10; 8 comparisons.
21. $M(1) = 0, M(n) = 2M(n/2) + (n - 1)$ for $n = 2^m, n \geq 2$
- 23.
- | | selectionsort | mergesort |
|----------|---------------|-----------|
| $n = 4$ | 9 | 5 |
| $n = 8$ | 35 | 17 |
| $n = 16$ | 135 | 49 |
| $n = 32$ | 527 | 129 |
25. Original list: 9, 8, 3, 13. After 1st pass: 8, 3, [9], 13. After 2nd pass: 3, [8], [9], 13 – sorted
27. 6
29. $Q(1) = 0, Q(n) = (n - 1) + 2Q(n/2)$ for $n \geq 2$
31. $Q(1) = 0, Q(n) = (n - 1) + Q(n - 1)$ for $n \geq 2$
33. If the original list is sorted in increasing order, then the first element of each sublist is the smallest element, so for the next pass the list of elements smaller than the pivot element will be empty and the list of elements greater than the pivot element will be only one element shorter than the sublist.
- 35.
- | Position at which x occurs | Number of comparisons |
|------------------------------|-----------------------|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| ... | ... |
| n | n |
37. For $m = 1, F(m + 2) = F(3) = 2$ and $F(m + 1) = F(2) = 1$. We need to show that if 1 division is required to find $\text{gcd}(a, b)$, then $a \geq 2$ and $b \geq 1$. Because the Euclidean algorithm applies to positive integers, $b \geq 1$. Because $a > b, a \geq 2$. Assume that if k divisions are required, $a \geq F(k + 2), b \geq F(k + 1)$. Show that if $k + 1$ divisions are required, then $a \geq F(k + 3), b \geq F(k + 2)$. The first step of the algorithm in computing $\text{gcd}(a, b)$ is to divide a by b , so $a = qb + r, 0 \leq r < b$. This is 1 division. The algorithm finishes the computation by finding $\text{gcd}(b, r)$, which will therefore require k divisions. By the inductive hypothesis, $b \geq F(k + 2)$ and $r \geq F(k + 1)$. Then $a = qb + r \geq b + r (q \geq 1 \text{ because } a > b) \geq F(k + 2) + F(k + 1) = F(k + 3)$.
39. From Exercise 38, $\left(\frac{3}{2}\right)^{m+1} < a$, so, taking the logarithm to the 3/2 of both sides, $m + 1 < \log_{1.5} a$ or $m < \log_{1.5} a - 1$.

CHAPTER 4

EXERCISES 4.1

1. a. T b. F c. F d. F
3. Four: $\{2, 3, 4\} = \{x|x \in N \text{ and } 2 \leq x \leq 4\} = \{3, 4, 2\}, \{a, b, c\} = \{x|x \text{ is the first letter of cat, bat, or apple}\}, \emptyset = \{x|x \text{ is the first letter of cat, bat, and apple}\}, \{2, a, 3, b, 4, c\}$

5. a. $\{0, 1, 2, 3, 4\}$
 b. $\{4, 6, 8, 10\}$
 c. $\{\text{Washington, Adams, Jefferson}\}$
 d. \emptyset
 e. $\{\text{Maine, Vermont, New Hampshire, Massachusetts, Connecticut, Rhode Island}\}$
 f. $\{-3, -2, -1, 0, 1, 2, 3\}$
7. a. $\{x|x \in \mathbb{N} \text{ and } 1 \leq x \leq 5\}$
 b. $\{x|x \in \mathbb{N} \text{ and } x \text{ is odd}\}$
 c. $\{x|x \text{ is one of the Three Wise Men}\}$
 d. $\{x|x \text{ is a nonnegative integer written in binary form}\}$
9. If $A = \{x|x = 2^n \text{ for } n \text{ a positive integer}\}$, then $16 \in A$. But if $A = \{x|x = 2 + n(n - 1) \text{ for } n \text{ a positive integer}\}$, then $16 \notin A$. In other words, there is not enough information to answer the question.
11. a. F b. T c. F d. T e. T f. T
13. $b, e,$ and g are true; a is false because $\{1\} \in S$ but $\{1\} \notin R$; c is false because $\{1\} \in S$, but $1 \notin S$; d is false because 1 is not a set (the correct statement is $\{1\} \subseteq U$); f is false because $1 \notin S$
15. $a, b, d, e, g,$ and i are true; c is false because neither member of C is a member of A ; f is false because this 2-element set is not an element of A ; h is false because $a \notin C$
17. Let $(x, y) \in A$. Then (x, y) lies within 3 units of the point $(1, 4)$, so by the distance formula, $\sqrt{(x - 1)^2 + (y - 4)^2} \leq 3$, or $(x - 1)^2 + (y - 4)^2 \leq 9$, which means $(x - 1)^2 + (y - 4)^2 \leq 25$, so $(x, y) \in B$. The point $(6, 4)$ satisfies the inequality $(x - 1)^2 + (y - 4)^2 \leq 25$, so $(6, 4) \in B$, but $(6, 4)$ is not within 3 units of $(1, 4)$, so $(6, 4)$ does not belong to A .
19. a. For $a = 1, b = -2, c = -24$, the quadratic equation is $x^2 - 2x - 24 = 0$ or $(x + 4)(x - 6) = 0$, with solutions 6 and -4 . Each of these is an even integer between -100 and 100 , so each belongs to E .
 b. Here $Q = \{6, -4\}$, but $E = \{-4, -2, 0, 2, 4\}$, and $Q \not\subseteq E$.
21. $a, d,$ and e
23. Let $x \in A$. Then, because $A \subseteq B, x \in B$. Because $B \subseteq C, x \in C$. Thus $A \subseteq C$.
25. The proof uses mathematical induction. $n = 2$: A set with 2 elements has exactly 1 subset with 2 elements, namely the set itself. Putting $n = 2$ into the formula $n(n - 1)/2$ gives the value 1. This proves the base case. Assume that any set with k elements has $k(k - 1)/2$ subsets with exactly 2 elements. Show that any set with $k + 1$ elements has $(k + 1)k/2$ subsets with exactly 2 elements. Let x be a member of a set with $k + 1$ elements. Temporarily removing x from the set gives a set of k elements that, by the inductive hypothesis, has $k(k - 1)/2$ subsets with exactly 2 elements. These are all of the 2-element subsets of the original set that do not include x . All 2-element subsets of the original set that do include x can be found by pairing x in turn with each of the remaining k elements, giving k subsets. The total number of 2-element subsets is therefore $\frac{k(k - 1)}{2} + k = \frac{(k + 1)k}{2}$
27. $\wp(S) = \{\emptyset, \{a\}\}$
29. For this set of four elements, the power set should have $2^4 = 16$ elements.
 $\wp(S) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}, \{1, 2, 3, 4\}\}$
31. $\wp(S) = \{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \{\{\emptyset, \{\emptyset\}\}\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset, \{\emptyset\}\}\}, \{\{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}, \{\emptyset, \{\emptyset, \{\emptyset, \{\emptyset\}\}\}\}$
33. $A = \{x, y\}$
35. Let $x \in A$. Then $\{x\} \in \wp(A)$, so $\{x\} \in \wp(B)$ and $x \in B$. Thus $A \subseteq B$. A similar argument shows that $B \subseteq A$ so that $A = B$.
37. a. $x = 1, y = 5$ b. $x = 8, y = 7$ c. $x = 1, y = 4$

39. a. binary operation
 b. no; $0 \circ 0 \notin \mathbb{N}$, so closure fails
 c. binary operation
 d. no; $\ln x$ is undefined for $x \leq 0$
41. a. no; operation undefined for $x = 0$
 b. binary operation
 c. unary operation
 d. binary operation
43. n^n
45. a. $AB + CD - *$ b. $AB ** CD * -$ c. $AC * BCDB * +/+$
47. a. $\{t\}$ b. $\{p, q, r, s, t, u\}$ c. $\{q, r, v, w\}$ d. \emptyset
49. a. $\{1, 2, 4, 5, 6, 8, 9\}$ e. $\{2, 6, 8\}$
 b. $\{4, 5\}$ f. $\{0, 1, 3, 7, 9\}$
 c. $\{2, 4\}$ g. \emptyset
 d. $\{1, 2, 3, 4, 5, 9\}$
51. a. $\{a\}$ e. $\{a, \{a\}\}$
 b. $\{\emptyset, \{a\}, \{a, \{a\}\}\}$ f. $\{\emptyset, \{a, \{a\}\}\}$
 c. $\{\emptyset, a, \{a\}, \{\{a\}\}, \{a, \{a\}\}\} = S$ g. $\{\emptyset\}$
 d. \emptyset
53. $c, e,$ and f
55. a. B' d. $B' \cap C$
 b. $B \cap C$ e. $B' \cap C'$ or $(B \cup C)'$ or $B' - C$
 c. $A \cap B$
57. a. C' b. $B \cap D$ c. $A \cap B$ d. $A \cap D'$
59. $D \cap R'$ 61. $(N \cup P) \cap A$
63. $a, b, d,$ and f 65. b and c
67. a. $B \subseteq A$ b. $A \subseteq B$ c. $A = \emptyset$ d. $B \subseteq A$ e. $A = B$ f. $A = B$
69. Let $x \in A \cap B$. Then $x \in A$ and $x \in B$, so $x \in A$.
71. Let $C \in \wp(A) \cap \wp(B)$. Then $C \in \wp(A)$ and $C \in \wp(B)$, from which $C \subseteq A$ and $C \subseteq B$, so $C \subseteq A \cap B$ or $C \in \wp(A \cap B)$. Therefore $\wp(A) \cap \wp(B) \subseteq \wp(A \cap B)$. The same argument works in reverse.
73. Suppose $B \neq \emptyset$. Let $x \in B$. Then $x \in A \cup B$ but $x \notin A - B$, which contradicts the equality of $A \cup B$ and $A - B$.
75. Let $x \in C$. Then $x \in B - A = B \cap A'$. Therefore $x \in A'$, and no element that is in C can also be in A , so $A \cap C = \emptyset$.
77. i. Let $A \subseteq B$ and let $x \in A$. Then $x \in B$, so $x \notin B'$ and no element that is in A can also be in B' . Therefore $A \cap B' = \emptyset$. ii. Let $A \cap B' = \emptyset$ and let $x \in A$. Then because $A \cap B' = \emptyset$, $x \notin B'$, so $x \in B$. Therefore any element of A is an element of B , and $A \subseteq B$.
79. a.  b. $\{2, 4, 6, 7, 9\}$
- c. $x \in (A \cup B) - (A \cap B) \leftrightarrow x \in (A \cup B)$ and $x \in (A \cap B)'$ $\leftrightarrow (x \in A \text{ or } x \in B)$ and $x \notin A \cap B \leftrightarrow (x \in A \text{ and } x \notin A \cap B)$ or $(x \in B \text{ and } x \notin A \cap B) \leftrightarrow (x \in A \text{ and } x \notin B)$ or $(x \in B \text{ and } x \notin A) \leftrightarrow x \in (A - B) \cup (B - A)$
81. (1a) $x \in A \cup B \leftrightarrow x \in A \text{ or } x \in B \leftrightarrow x \in B \text{ or } x \in A \leftrightarrow x \in B \cup A$ (1b) $x \in A \cap B \leftrightarrow x \in A \text{ and } x \in B \leftrightarrow x \in B \text{ and } x \in A \leftrightarrow x \in B \cap A$ (2a) $x \in (A \cup B) \cup C \leftrightarrow x \in (A \cup B)$ or $x \in C \leftrightarrow (x \in A \text{ or } x \in B)$ or $x \in C \leftrightarrow x \in A \text{ or } x \in B \text{ or } x \in C \leftrightarrow x \in A \text{ or } (x \in B \text{ or } x \in C) \leftrightarrow x \in A \text{ or } x \in (B \cup C) \leftrightarrow$

$x \in A \cup (B \cup C)$ **(2b)** $x \in (A \cap B \cap C) \leftrightarrow x \in (A \cap B)$ and $x \in C \leftrightarrow (x \in A$ and $x \in B)$ and $x \in C$
 $\leftrightarrow x \in A$ and $x \in B$ and $x \in C \leftrightarrow x \in A$ and $(x \in B$ and $x \in C) \leftrightarrow x \in A$ and $x \in (B \cap C) \leftrightarrow$
 $x \in A \cap (B \cap C)$ **(3b)** $x \in A \cap (B \cup C) \leftrightarrow x \in A$ and $x \in (B \cup C) \leftrightarrow x \in A$ and $(x \in B$ or $x \in C) \leftrightarrow$
 $(x \in A$ and $x \in B)$ or $(x \in A$ and $x \in C) \leftrightarrow x \in (A \cap B)$ or $x \in (A \cap C) \leftrightarrow x \in (A \cap B) \cup (A \cap C)$
(4b) $x \in A \cap S \rightarrow x \in A$ and $x \in S \rightarrow x \in A$. Also $x \in A \rightarrow x \in A$ and $x \in S$ because $A \subseteq S \rightarrow x \in A \cap S$
(5a) $x \in A \cup A' \rightarrow x \in A$ or $x \in A' \rightarrow x \in S$ or $x \in S$ because $A \subseteq S, A' \subseteq S \rightarrow x \in S$. Also $x \in S \rightarrow$
 $(x \in S$ and $x \in A)$ or $(x \in S$ and $x \notin A) \rightarrow x \in A$ or $x \in A' \rightarrow x \in A \cup A'$ **(5b)** For any x such that
 $x \in A \cap A'$, it follows that $x \in A$ and $x \in A'$, or x belongs to A and x does not belong to A . This is a con-
 tradiction; no x belongs to $A \cap A'$, and $A \cap A' = \emptyset$.

$$83. \text{ a. } ((A \cup B) \cap (A \cup B')) = A \cup (B \cap B') \quad (3a)$$

$$= A \cup \emptyset \quad (5b)$$

$$= A \quad (4a)$$

The dual is $(A \cap B) \cup (A \cap B') = A$.

$$\text{ b. } [((A \cap C) \cap B) \cup ((A \cap C) \cap B')] \cup (A \cap C)' \quad (3b)$$

$$= [(A \cap C) \cap (B \cup B')] \cup (A \cap C)' \quad (3b)$$

$$= [(A \cap C) \cap S] \cup (A \cap C)' \quad (5b)$$

$$= (A \cap C) \cup (A \cap C)' \quad (4b)$$

$$= S \quad (5b)$$

The dual is $[((A \cup C) \cup B) \cap ((A \cup C) \cup B')] \cap (A \cup C)' = \emptyset$.

$$\text{ c. } (A \cup C) \cap [(A \cap B) \cup (C' \cap B)] \quad (1b)$$

$$= (A \cup C) \cap [(B \cap A) \cup (B \cap C')] \quad (1b)$$

$$= (A \cup C) \cap [B \cap (A \cup C')] \quad (3b)$$

$$= (A \cup C) \cap [(A \cup C') \cap B] \quad (1b)$$

$$= [(A \cup C) \cap (A \cup C')] \cap B \quad (2b)$$

$$= [A \cup (C \cap C')] \cap B \quad (3a)$$

$$= (A \cup \emptyset) \cap B \quad (5b)$$

$$= A \cap B \quad (4b)$$

The dual is $(A \cap C) \cup [(A \cup B) \cap (C' \cup B)] = A \cup B$.

$$85. \text{ a. } A \cap (B \cup A') = (A \cap B) \cup (A \cap A') \quad (3b)$$

$$= (A \cap B) \cup \emptyset \quad (5b)$$

$$= A \cap B \quad (4a)$$

$$= B \cap A \quad (1b)$$

$$\text{ b. } (A \cup B) - C = (A \cup B) \cap C' \quad (\text{defn. set diff.})$$

$$= C' \cap (A \cup B) \quad (1b)$$

$$= (C' \cap A) \cup (C' \cap B) \quad (3b)$$

$$= (A \cap C') \cup (B \cap C') \quad (1b)$$

$$= (A - C) \cup (B - C) \quad (\text{defn. set diff.})$$

$$\text{ c. } (A - B) - C = (A - B) \cap C' \quad (\text{defn. set diff.})$$

$$= (A \cap B') \cap C' \quad (\text{defn. set diff.})$$

$$= C' \cap (A \cap B') \quad (1b)$$

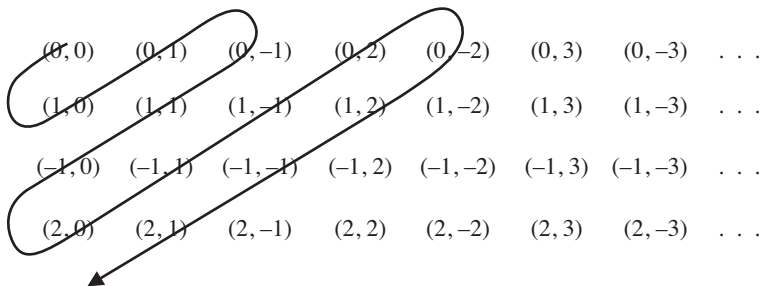
$$= (C' \cap A) \cap B' \quad (2b)$$

$$= (A \cap C') \cap B' \quad (1b)$$

$$= (A - C) \cap B' \quad (\text{defn. set diff.})$$

$$= (A - C) - B \quad (\text{defn. set diff.})$$

87. a. $A_1 \cup A_2 \cup \dots \cup A_n = \{x|x \text{ belongs to some } A_i \text{ for } 1 \leq i \leq n\}$
 b. $A_1 \cup A_2 = \{x|x \in A_1 \text{ or } x \in A_2\}$ for $n = 2$, $A_1 \cup A_2 \cup \dots \cup A_n = (A_1 \cup \dots \cup A_{n-1}) \cup A_n$ for $n > 2$
89. a. $A_1 \cap A_2 \cap \dots \cap A_n = \{x|x \text{ belongs to every } A_i \text{ for } 1 \leq i \leq n\}$
 b. $A_1 \cap A_2 = \{x|x \in A_1 \text{ and } x \in A_2\}$ for $n = 2$, $A_1 \cap A_2 \cap \dots \cap A_n = (A_1 \cap \dots \cap A_{n-1}) \cap A_n$ for $n > 2$
91. a. Proof is by induction on n . For $n = 2$, $B \cup (A_1 \cap A_2) = (B \cup A_1) \cap (B \cup A_2)$ by identity 3a. Assume that $B \cup (A_1 \cap \dots \cap A_k) = (B \cup A_1) \cap \dots \cap (B \cup A_k)$. Then $B \cup (A_1 \cap \dots \cap A_{k+1}) = B \cup ((A_1 \cap \dots \cap A_k) \cap A_{k+1})$ by Exercise 89b $= (B \cup (A_1 \cap \dots \cap A_k)) \cap (B \cup A_{k+1})$ by identity 3a $= ((B \cup A_1) \cap \dots \cap (B \cup A_k)) \cap (B \cup A_{k+1})$ by inductive hyp. $= (B \cup A_1) \cap \dots \cap (B \cup A_{k+1})$ by Exercise 89b.
 b. Proof is by induction on n . For $n = 2$, $B \cap (A_1 \cup A_2) = (B \cap A_1) \cup (B \cap A_2)$ by identity 3b. Assume that $B \cap (A_1 \cup \dots \cup A_k) = (B \cap A_1) \cup \dots \cup (B \cap A_k)$. Then $B \cap (A_1 \cup \dots \cup A_{k+1}) = B \cap ((A_1 \cup \dots \cup A_k) \cup A_{k+1})$ by Exercise 87b $= (B \cap (A_1 \cup \dots \cup A_k)) \cup (B \cap A_{k+1})$ by identity 3b $= ((B \cap A_1) \cup \dots \cup (B \cap A_k)) \cup (B \cap A_{k+1})$ by inductive hyp. $= (B \cap A_1) \cup \dots \cup (B \cap A_{k+1})$ by Exercise 87b.
93. a. $\bigcup_{i \in I} A_i = \{x|x \in (-1,1)\}$; $\bigcap_{i \in I} A_i = \{0\}$
 b. $\bigcup_{i \in I} A_i = \{x|x \in [-1,1]\}$; $\bigcap_{i \in I} A_i = \{0\}$
95. $P(1)$ is true—every member of T is greater than 1; otherwise 1 would be the smallest member of T . Assume that $P(k)$ is true, i.e., every member of T is greater than k . Consider $P(k + 1)$, that every member of T is greater than $k + 1$. If $P(k + 1)$ is not true, then there is some member of $T \leq k + 1$. By the inductive hypothesis, every member of T is greater than k ; therefore some member of T equals $k + 1$, and this is the smallest member of T . This is a contradiction, because we assumed that T has no smallest member. Therefore $P(k + 1)$ is true. By the first principle of induction, $P(n)$ is true for all n , and T must be empty. This contradicts the fact that T is a non-empty set.
97. An enumeration of the set is 1, 3, 5, 7, 9, 11, \dots
99. An enumeration of the set is $a, aa, aaa, aaaa, \dots$
101. An enumeration of the set is shown by the arrow through the array



103. Assume that the set has an enumeration

$$\begin{aligned}
 & z_{11}, z_{12}, z_{13}, z_{14}, \dots \\
 & z_{21}, z_{22}, z_{23}, z_{24}, \dots \\
 & z_{31}, z_{32}, z_{33}, z_{34}, \dots \\
 & \vdots
 \end{aligned}$$

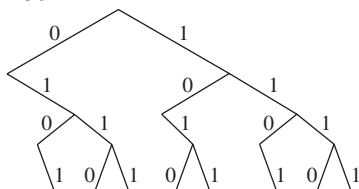
Now construct an infinite sequence Z of positive integers with $Z = z_1, z_2, z_3, \dots$ such that $z_i \neq z_{ii}$ for all i . Then Z differs from every sequence in the enumeration, yet it is a member of the set. This is a contradiction, so the set is uncountable.

- 105.** Let A and B be denumerable sets with enumerations $A = a_1, a_2, a_3, \dots$ and $B = b_1, b_2, b_3, \dots$. Then use the list $a_1, b_1, a_2, b_2, a_3, b_3, \dots$ and eliminate any duplicates. This will be an enumeration of $A \cup B$, which is therefore denumerable.
- 107.** $B = \{S \mid S \text{ is a set and } S \notin S\}$. Then either $B \in B$ or $B \notin B$. If $B \in B$, then B has the property of all members of B , namely $B \notin B$. Hence both $B \in B$ and $B \notin B$ are true. If $B \notin B$, then B has the property characterizing members of B , hence $B \in B$. Therefore both $B \notin B$ and $B \in B$ are true.

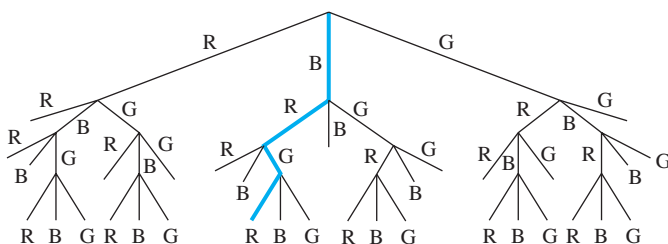
EXERCISES 4.2

- | | | | |
|----------------------|----------------|---------|----------------|
| 1. 30 | 19. 1680 | 37. 32 | 55. 60,466,176 |
| 3. 92 | 21. 25,974,000 | 39. 160 | 57. 3,515,625 |
| 5. $26^3 \cdot 10^2$ | 23. 917 | 41. 8 | 59. 2704 |
| 7. 585 | 25. 15 | 43. 36 | 61. 144 |
| 9. 10^9 | 27. 180 | 45. 6 | 63. 208 |
| 11. 17,576 | 29. 720 | 47. 25 | 65. 96 |
| 13. 16 | 31. 45 | 49. 648 | 67. 1104 |
| 15. 286 | 33. 256 | 51. 72 | |
| 17. 180 | 35. 192 | 53. 36 | |

69.

8 outcomes, which equals $F(6)$

71.



33 ways; the outcome that is highlighted is BRGR.

- 73.** For $m = 2$, the result follows from the multiplication principle. Assume that for $m = k$, there are $n_1 \cdot \dots \cdot n_k$ possible outcomes for the sequence of events 1 to k . Let $m = k + 1$. Then the sequence of events 1 to $k + 1$ consists of the sequence of events 1 to k followed by event $k + 1$. The sequence of events 1 to k has $n_1 \cdot \dots \cdot n_k$ possible outcomes by the inductive hypothesis. The sequence 1 to k followed by event $k + 1$ then has $((n_1 \cdot \dots \cdot n_k)n_{k+1})$ outcomes by the multiplication principle, which equals $n_1 \cdot \dots \cdot n_{k+1}$.
- 75 a.** $P(1) = 1$ (trivial case), $P(2) = 1$ (only one way to multiply 2 terms). For $n > 2$, let the last multiplication occur at position k , $1 \leq k \leq n - 1$. The product is then split into two products of k and $(n - k)$ factors, respectively, which can be parenthesized in $P(k)$ and $P(n - k)$ ways, respectively. By the multiplication principle, there are $P(k)P(n - k)$ ways to parenthesize for a fixed k . Each value for k gives a different set of parentheses, so by the addition principle, $P(n) = P(1)P(n - 1) + P(2)P(n - 2) + \dots + P(n - 1)P(1) = \sum_{k=1}^{n-1} P(k)P(n - k)$.
- b.** The proof will use the Second Principle of Induction. $P(1) = 1 = C(0)$, $P(2) = 1 = C(1)$. Assume that $P(r) = C(r - 1)$ for $1 \leq r \leq m$. Then $P(m + 1) = \sum_{k=1}^m P(k)P(m + 1 - k) = \sum_{k=1}^m C(k - 1)C(m - k) = C(m)$.

EXERCISES 4.3

1. 19
3. 60
5. 5 parts had all three types of defect.
7. a. 2 b. 6
15. a. $|A| + |B| + |C| + |D| - |A \cap B| - |A \cap C| - |A \cap D| - |B \cap C| - |B \cap D| - |C \cap D| + |A \cap B \cap C| + |A \cap B \cap D| + |A \cap C \cap D| + |B \cap C \cap D| - |A \cap B \cap C \cap D|$
b. $2^n - 1$
17. 5 19. No 21. 51 23. 367
25. There are 3 pairs—1 and 6, 2 and 5, 3 and 4—that add up to 7. Each element in the set belongs to one of these pairs. Apply the pigeonhole principle, where the pairs are the bins, and the numbers are the items.
27. This follows from the pigeonhole principle, where the n possible remainders (the numbers 0 through $n - 1$) are the bins.

EXERCISES 4.4

1. a. 42 b. 6720
3. 362,880
5. 3,628,800
7. 40,320; 15,120
9. 2730
11. 19!
13. $(2!)(11!)(8!) = 2(39,916,800)(40,320)$
15. 18!
17. a. 120 b. 36 c. 28
19. $C(300, 25)$
21. $C(17, 5)C(23, 7)$
23. 11,760
25. 427,518
27. 48
29. 22308
31. 792
33. 4
35. 624
37. 5,108
39. 54,912
41. 1,098,240
43. 482,320,623,240
45. 11,662,691,040
47. 902,720
49. 495
51. 40
53. 1770
55. 341,149,446
57. 220
59. 115
61. 14,307,150
63. 4,412,826
65. 8,586,820
67. 3003
69. 2508
71. 19,481
73. 36
75. 20
77. a. 3360 b. 420
79. 27,720
81. 21
83. $C(81, 48) =$ very big number
85. a. 28 b. 7 c. 10
87. a. 8008 b. 84
89. 286
91. $P(n, n) = \frac{n!}{(n-n)!} = \frac{n!}{0!} = n!$ and $P(n, n-1) = \frac{n!}{(n-(n-1))!} = \frac{n!}{1!} = n!$
93. $C(n, r) = \frac{n!}{r!(n-r)!} = \frac{n!}{(n-r)!(n-(n-r))!} = C(n, n-r)$

Whenever r objects are chosen from n , $n - r$ objects are not chosen. Therefore the number of ways to choose r objects out of n is the same as the number of ways to choose $n - r$ objects out of n .

95. Consider selecting r elements from a set of n and putting those in bucket A , then selecting k of those r to put in bucket B . The left side multiplies the number of outcomes from those two sequential tasks. Alternatively, we can select k elements from n and place them in bucket B , then $r - k$ elements from the remaining $n - k$ and place them in bucket A . The right side multiplies the number of outcomes from these two sequential tasks.

$$97. C(2) = \frac{1}{3}C(4,2) = \frac{1}{3} \frac{4!}{2! \cdot 2!} = \frac{4 \cdot 3}{3 \cdot 2} = 2$$

$$C(3) = \frac{1}{4}C(6,3) = \frac{1}{4} \frac{6!}{3! \cdot 3!} = \frac{6 \cdot 5 \cdot 4}{4 \cdot 2 \cdot 3} = 5$$

$$C(4) = \frac{1}{5}C(8,4) = \frac{1}{5} \frac{8!}{4! \cdot 4!} = \frac{8 \cdot 7 \cdot 6 \cdot 5}{5 \cdot 2 \cdot 3 \cdot 4} = 14$$

These results agree with the recurrence relation results.

99. 163452, 163542, 345621, 356421, 634521, 643125
101. 7431652; reading right to left, the first non-increasing value is 1. Again reading right to left, the first value greater than 1 is 2, so swap 1 and 2, giving 7432651. From the right of the 2 value, the numbers decrease; swap 6 and 1, giving 7432156.
103. 3675421; reading right to left, the first non-increasing value is 6. Again reading right to left, the first value greater than 6 is 7, so swap 6 and 7, giving 3765421. From the right of the 7 value, the numbers decrease; swap 6 and 1, swap 5 and 2, giving 3712456.
105. 24589, 24678, 24679, 24689, 24789
107. Make the initial permutation the largest permutation, $n \dots 321$. Then just reverse all inequalities in the body of algorithm permutation generator.

EXERCISES 4.5

1. a. $a^5 + 5a^4b + 10a^3b^2 + 10a^2b^3 + 5ab^4 + b^5$
 b. $x^6 + 6x^5y + 15x^4y^2 + 20x^3y^3 + 15x^2y^4 + 6xy^5 + y^6$
 c. $a^5 + 10a^4 + 40a^3 + 80a^2 + 80a + 32$
 d. $a^4 - 16a^3 + 96a^2 - 256a + 256$
3. $120a^7b^3$
5. $-489,888x^4$
7. $6561y^8$
9. $2560x^3y^2$
11. Think of $(a + b + c)^3$ as $((a + b) + c)^3$. Then $((a + b) + c)^3 = C(3, 0)(a + b)^3 + C(3, 1)(a + b)^2c + C(3, 2)(a + b)c^2 + C(3, 3)c^3 = a^3 + 3a^2b + 3ab^2 + b^3 + 3a^2c + 6abc + 3b^2c + 3ac^2 + 3bc^2 + c^3$
13. 11,200
15. $C(n + 2, r) = C(n + 1, r - 1) + C(n + 1, r)$ (Pascal's formula) $= C(n, r - 2) + C(n, r - 1) + C(n, r - 1) + C(n, r)$ (Pascal's formula again) $= C(n, r) + 2C(n, r - 1) + C(n, r - 2)$
17. From the binomial theorem with $a = 1$, $b = (-1)$: $C(n, 0) - C(n, 1) + C(n, 2) - \dots + (-1)^n C(n, n) = (1 + (-1))^n = 0^n = 0$
19. From the binomial theorem with $a = 1$, $b = 2^{-1}$:

$$C(n, 0) + C(n, 1)2^{-1} + C(n, 2)2^{-2} + \dots + C(n, n)2^{-n} = (1 + 2^{-1})^n$$

so, multiplying by 2^n ,

$$C(n, 0)2^n + C(n, 1)2^{n-1} + C(n, 2)2^{n-2} + \dots + C(n, n)2^{n-n} = 2^n(1 + 2^{-1})^n$$

or

$$C(n, 0)2^n + C(n, 1)2^{n-1} + C(n, 2)2^{n-2} + \cdots + C(n, n) =$$

$$2^n \left(1 + \frac{1}{2}\right)^n = 2^n \left(\frac{2+1}{2}\right)^n = 2^n \left(\frac{3^n}{2^n}\right) = 3^n$$

- 21. a.** $(1+x)^n = C(n, 0) + C(n, 1)x + C(n, 2)x^2 + C(n, 3)x^3 + \cdots + C(n, n)x^n$
b. Differentiating both sides of the equation in (a) gives $n(1+x)^{n-1} = C(n, 1) + 2C(n, 2)x + 3C(n, 3)x^2 + \cdots + nC(n, n)x^{n-1}$ **c.** follows from (b) with $x = 1$ **d.** follows from (b) with $x = -1$
- 23. a.** Out of all the intersections of m sets, $1 \leq m \leq k$, we want the ones that pick all m sets from the k sets in B . There are $C(k, m)$ ways to do this.
b. For any value of m , $1 \leq m \leq k$, an intersection of m sets has $C(k, m)$ that include only sets in B . Counting the intersections in the right side of (1) that contain x (intersections of single sets, intersections of 2 sets, and so forth) we get $C(k, 1) - C(k, 2) + C(k, 3) - \cdots + (-1)^{k+1}C(k, k)$
c. From Exercise 17, $C(k, 0) - C(k, 1) + C(k, 2) - \cdots + (-1)^k C(k, k) = 0$ or $C(k, 1) - C(k, 2) + \cdots + (-1)^{k+1}C(k, k) = C(k, 0)$, but $C(k, 0) = 1$. Therefore x is counted only once in the right side of (1).

EXERCISES 4.6

Some decimal answers in this section are approximations.

- | | | |
|----------------|-----------------|---------------------------|
| 1. 8 | 13. 1/12 | 25. 1326 |
| 3. 1/8 | 15. 8 | 27. $\cong 0.5588$ |
| 5. 1/4 | 17. 1/8 | 29. $\cong 0.3824$ |
| 7. 36 | 19. 1/4 | 31. $\cong 0.0498$ |
| 9. 1/6 | 21. 1/52 | 33. $\cong 0.0023$ |
| 11. 1/6 | 23. 1/2 | |
- 35.** The answer to Exercise 30 should be the sum of the answers to Exercises 28 and 29 because “at least one spade” is either exactly one spade (Exercise 29) or two spades (Exercise 28). Using the probabilities obtained, this arithmetic is correct.
- | | | | |
|------------------------------|-----------------------------|----------------------------|--------------------------|
| 37. a. 1000 | b. 0.001 | c. 0.006 | d. 0.003 |
| 39. a. 69,090,840 | b. $\cong 0.0000017$ | c. $\cong 0.000295$ | d. $\cong 0.0097$ |
| 41. $\cong 0.0000015$ | | 47. $\cong 0.021$ | |
| 43. $\cong 0.0002$ | | 49. $\cong 0.423$ | |
| 45. $\cong 0.002$ | | 51. 365^n | |
- 53.** $B = E'$, so from Exercise 52, $P(B) = 1 - P(E) = 1 - P(365, n)/365^n$.
- 55.** B has a higher probability because B consists of exactly two + exactly three + \cdots + exactly n persons sharing the same birthday, whereas C consists of exactly two.
- | | | | |
|-------------------------------|----------------------------|----------------|----------------|
| 57. 38 | 67. a. 0.55 | b. 0.68 | c. 0.32 |
| 59. $\cong 0.026$ | 69. a. 0.6 | b. 0.25 | c. 0.65 |
| 61. $\cong 0.105$ | d. 0.15 | e. 0.95 | |
| 63. $\cong 0.00000751$ | 71. $\cong 0.93$ | | |
| 65. 6.29908E-12 | 73. a. $\cong 0.24$ | b. 0.43 | c. 0.57 |

75. 0.25

77. 0.3125

79. 0.5

81. 0.5

$$89. \text{ a. } P(E_i | F) = \frac{P(E_i \cap F)}{P(F)} \quad (1)$$

$$P(F | E_i) = \frac{P(F \cap E_i)}{P(E_i)} \text{ or } P(F \cap E_i) = P(F | E_i)P(E_i) \quad (2)$$

Because $P(F \cap E_i) = P(E_i \cap F)$, substitute from Equation (2) into Equation (1), giving

$$P(E_i | F) = \frac{P(F | E_i) P(E_i)}{P(F)}$$

b. The events $E_i, 1 \leq i \leq n$, are all disjoint; therefore the events $F \cap E_i, 1 \leq i \leq n$ are all disjoint. $F = F \cap S = F \cap (E_1 \cup E_2 \cup \dots \cup E_n) = (F \cap E_1) \cup (F \cap E_2) \cup \dots \cup (F \cap E_n)$. Because the probability of the union of disjoint events is the sum of the probabilities of each event,

$$P(F) = \sum_{k=1}^n P(F \cap E_k).$$

c. From Equation (2) of part (a), $P(F \cap E_i) = P(F | E_i)P(E_i)$. Substituting into the result of part (b) gives

$$P(F) = \sum_{k=1}^n P(F | E_k)P(E_k)$$

d. Substituting the result of part (c) into the result of part (a) gives

$$P(E_i | F) = \frac{P(F | E_i)P(E_i)}{\sum_{k=1}^n P(F | E_k)P(E_k)}$$

91. $\cong 0.40$

93. a. 3.5

b. $\cong 3.29$

c. Less than (which turns out to be true, $3.29 < 3.5$). The reason is that the die is now weighted toward a smaller value than the previous expected value, so this smaller value is more likely to occur and will drag down the weighted average.

95. 4.75

97. a. $\cong 0.904$ b. $\cong 0.999$ c. $\cong 0.096$ 99. $\cong 0.547$

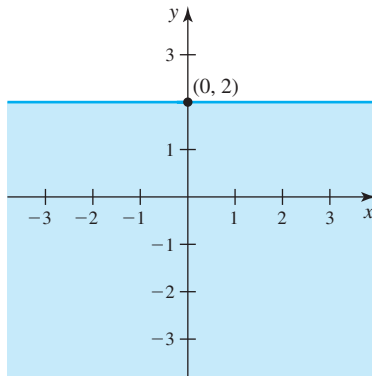
$$101. \frac{n^2 + 3n}{2(n + 1)}$$

CHAPTER 5

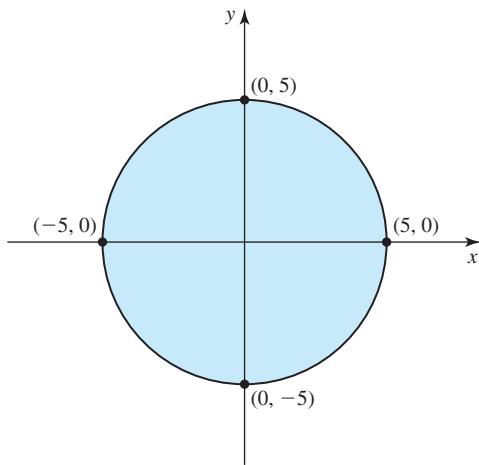
EXERCISES 5.1

- 1. a. $(1, 3), (3, 3)$
- b. $(4, 2), (5, 3)$
- c. $(5, 0), (2, 2)$
- d. $(1, 1), (3, 9)$

5. a.



c.



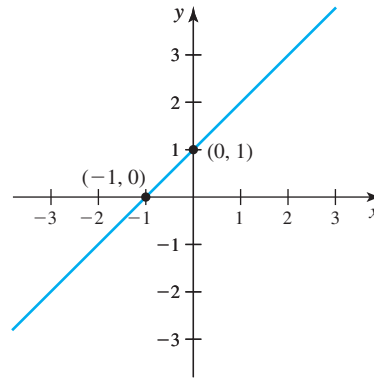
- 7. a. many-to-many
- b. many-to-one
- c. one-to-one
- d. one-to-many

- 11. a. reflexive
- b. reflexive, symmetric, antisymmetric, transitive
- c. none
- d. antisymmetric, transitive

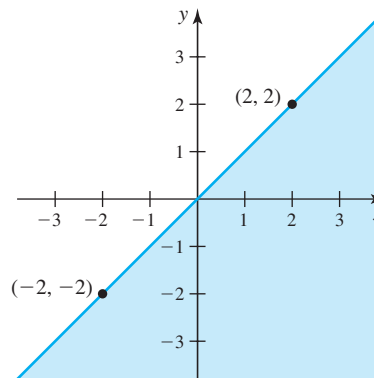
- 15. (b); the equivalence classes are $[0] = \{\dots, -9, -6, -3, 0, 3, 6, 9, \dots\}$, $[1] = \{\dots, -8, -5, -2, 1, 4, 7, 10, \dots\}$, and $[2] = \{\dots, -7, -4, -1, 2, 5, 8, 11, \dots\}$.
- (e); the equivalence classes are sets consisting of squares with equal length sides.

- 3. a. $(1, -1), (-3, 3)$
- b. $(19, 7), (41, 16)$
- c. $(-3, -5), (-4, 1/2), (1/2, 1/3)$
- d. $((1, 2), (3, 2))$

b.

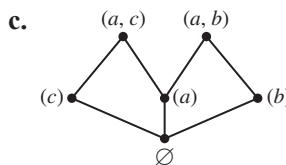
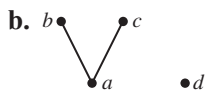
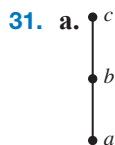


d.



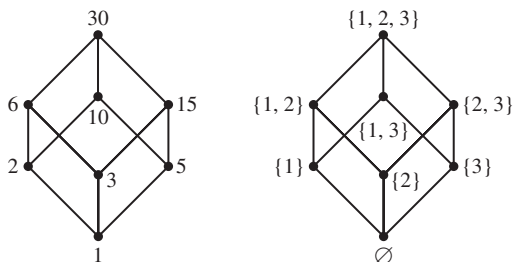
- 9. a. $(2, 6), (3, 17), (0, 0)$
- b. $(2, 12)$
- c. none
- d. $(1, 1), (4, 8)$
- 13. a. reflexive, transitive
- b. reflexive, symmetric, transitive
- c. symmetric
- d. transitive
- e. reflexive, symmetric, transitive

17. a. symmetric
 b. antisymmetric, transitive
 c. reflexive, antisymmetric, transitive
 d. reflexive, symmetric, transitive
19. a. reflexive, transitive
 b. antisymmetric, transitive
 c. reflexive, symmetric, transitive
 d. antisymmetric
21. For example:
 a. $S =$ set of all lines in the plane, $x \rho y \leftrightarrow x$ coincides with y or x is perpendicular to y .
 b. $S =$ set of integers, $x \rho y \leftrightarrow x^2 \leq y^2$
 c. $S =$ set of nonnegative integers, $x \rho y \leftrightarrow x < y$
 d. $S =$ set of integers, $x \rho y \leftrightarrow x \leq |y|$.
23. a. reflexive closure = ρ itself, symmetric closure—add $(2,1),(3,2)$, transitive closure—add $(2,1),(3,2)$
 b. reflexive closure = symmetric closure = transitive closure = ρ itself
 c. reflexive closure—add $(2,2),(3,3)$, symmetric closure—add $(2,1),(3,2)$, transitive closure—add $(1,1), (2,1), (2,2), (3,3)$
 d. reflexive closure—add $(2,2), (3,3)$, symmetric closure—add $(2,1), (3,2), (3,1)$, transitive closure = ρ itself
25. $x \rho^* y \leftrightarrow$ one can fly from x to y (perhaps by multiple hops) on Take-Your-Chance Airlines
27. No—if the relation is irreflexive, it is its own irreflexive closure. If the relation is not irreflexive, there must be some $x \in S$ with (x, x) in the relation; extending the relation will not remove this pair, so no extension can be irreflexive.

29. 2^{n^2} 

33. Reflexivity: If $x \in A$, then $x \in S$, so $(x \leq x)$ because \leq is a reflexive relation on S . Symmetry: if $x, y \in A$ and $x \leq y$, then $x, y \in S$ and $x \leq y$, so $y \leq x$ because \leq is symmetric on S . Transitivity: if $x, y, z \in A$ and $x \leq y$ and $y \leq z$, then $x, y, z \in S$, $x \leq y$, and $y \leq z$, so $x \leq z$ because \leq is transitive on S .

35.

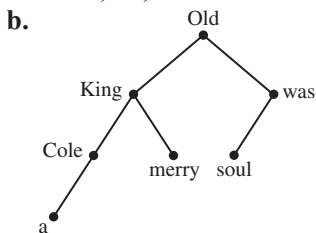


The two graphs are identical in structure.

37. Reflexive: $(s_1, t_1) \mu (s_1, t_1)$ because both $s_1 \rho s_1$ and $t_1 \sigma t_1$ due to reflexivity of ρ and σ . Antisymmetric: $(s_1, t_1) \mu (s_2, t_2)$ and $(s_2, t_2) \mu (s_1, t_1) \rightarrow s_1 \rho s_2$ and $s_2 \rho s_1, t_1 \sigma t_2$ and $t_2 \sigma t_1 \rightarrow s_1 = s_2$ and $t_1 = t_2$ due to antisymmetry of ρ and $\sigma \rightarrow (s_1, t_1) = (s_2, t_2)$. Transitive: $(s_1, t_1) \mu (s_2, t_2)$ and $(s_2, t_2) \mu (s_3, t_3) \rightarrow s_1 \rho s_2$ and $s_2 \rho s_3, t_1 \sigma t_2$ and $t_2 \sigma t_3 \rightarrow s_1 \rho s_3$ and $t_1 \sigma t_3$ due to transitivity of ρ and $\sigma \rightarrow (s_1, t_1) \mu (s_3, t_3)$.
39. Assume that ρ is reflexive and transitive on S . Then for all $x \in S$, $(x, x) \in \rho$, which means $(x, x) \in \rho^{-1}$, so $(x, x) \in \rho \cap \rho^{-1}$ and $\rho \cap \rho^{-1}$ is reflexive. Let $(x, y) \in \rho \cap \rho^{-1}$. Then $(x, y) \in \rho$ and $(x, y) \in \rho^{-1}$, which means $(x, y) \in \rho$ and $(y, x) \in \rho$. This implies $(y, x) \in \rho^{-1}$ and $(y, x) \in \rho$, so $(y, x) \in \rho \cap \rho^{-1}$ and $\rho \cap \rho^{-1}$ is symmetric. Let $(x, y) \in \rho \cap \rho^{-1}$ and $(y, z) \in \rho \cap \rho^{-1}$. Then $(x, y) \in \rho$ and $(x, y) \in \rho^{-1}$ and $(y, z) \in \rho$ and $(y, z) \in \rho^{-1}$, so $(x, y) \in \rho$ and $(y, x) \in \rho$ and $(y, z) \in \rho$ and $(z, y) \in \rho$. Because ρ is transitive, $(x, z) \in \rho$ and $(z, x) \in \rho$ or $(x, z) \in \rho$ and $(x, z) \in \rho^{-1}$, so $(x, z) \in \rho \cap \rho^{-1}$ and $\rho \cap \rho^{-1}$ is transitive.

41. Reflexive: $X \leq X$ because $x_i = x_i$, $1 \leq i \leq k$. Antisymmetric: Let $X \leq Y$ and $Y \leq X$. If $X \neq Y$, let $m + 1$ be the first index where $x_{m+1} \neq y_{m+1}$. Then $x_{m+1} \leq y_{m+1}$ and $y_{m+1} \leq x_{m+1} \rightarrow x_{m+1} = y_{m+1}$, a contradiction. Transitive: Let $X \leq Y$ and $Y \leq Z$. Then $x_p \leq y_p$ for some $p \leq k$ and $y_q \leq z_q$ for some $q \leq k$. Let $m = \min(p, q)$. Then $x_m \leq z_m$ and $X \leq Z$. Therefore \leq is a partial ordering. It is a total ordering by “otherwise”.

43. a. when; no; all but the last



Maximal elements: a, merry, soul

45. a. $[a] = \{a, c\} = [c]$
 b. $[3] = \{1, 2, 3\}$, $[4] = \{4, 5\}$

47. $[1] = \{\dots, -5, -3, -1, 1, 3, 5, \dots\}$

49. If $x \equiv y \pmod{n}$ then $x - y = k_1n$ for some integer k_1 , or $x = k_1n + y$. If $z \equiv w \pmod{n}$ then $z - w = k_2n$ for some integer k_2 , or $z = k_2n + w$.

a. $x + z = (k_1n + y) + (k_2n + w) = y + w + (k_1 + k_2)n$, so $x + z - (y + w) = (k_1 + k_2)n$ where $k_1 + k_2$ is an integer, and $x + z \equiv y + w \pmod{n}$.

b. $x - z = (k_1n + y) - (k_2n + w) = y - w + (k_1 - k_2)n$, so $x - z - (y - w) = (k_1 - k_2)n$ where $k_1 - k_2$ is an integer, and $x - z \equiv y - w \pmod{n}$.

c. $xz = (k_1n + y)(k_2n + w) = k_1k_2n^2 + yk_2n + wk_1n + yw = (k_1k_2n + yk_2 + wk_1)n + yw$ so $xz - yw = (k_1k_2n + yk_2 + wk_1)n$ where $k_1k_2n + yk_2 + wk_1$ is an integer, and $xz \equiv yw \pmod{n}$.

d.
$$x^s - y^s = (k_1n + y)^s - y^s = \left[\sum_{k=0}^s C(s, k)(k_1n)^{s-k}y^k \right] - y^s$$

$$= \left[\sum_{k=0}^{s-1} C(s, k)(k_1n)^{s-k}y^k \right] + y^s - y^s = n \sum_{k=0}^{s-1} C(s, k)k_1^{s-k}n^{s-k-1}y^k = nk_2$$

where k_2 is an integer and $x^s \equiv y^s \pmod{n}$.

51. a. $\{(1, 1), (2, 2), (3, 3), (4, 4), (1, 2), (2, 1), (3, 4), (4, 3)\}$

b. $\{(a, a), (b, b), (c, c), (d, d), (e, e), (a, b), (b, a), (a, c), (c, a), (b, c), (c, b), (d, e), (e, d)\}$

53. Reflexive: $x^2 - x^2 = 0$, which is even. Symmetric: If $x^2 - y^2 = 2n$ then $y^2 - x^2 = -2n$, which is even. Transitive: if $x^2 - y^2 = 2n$ and $y^2 - z^2 = 2m$, then $x^2 - z^2 = x^2 - y^2 + y^2 - z^2 = 2n + 2m = 2(n + m)$, which is even. The equivalence classes are the set of even integers and the set of odd integers.

55. Reflexive: $(x, y) \rho (x, y)$ because $y = y$. Symmetric: If $(x, y) \rho (z, w)$ then $y = w$ so $w = y$ and $(z, w) \rho (x, y)$. Transitive: If $(x, y) \rho (z, w)$ and $(z, w) \rho (s, t)$ then $y = w$ and $w = t$, so $y = t$ and $(x, y) \rho (s, t)$. The equivalence classes are sets of ordered pairs with the same second components.

57. a. Reflexive: $x \rho x$ because x and x start and end with the same bit values. Symmetric: If $x \rho y$ then y starts and ends with the same bits as x , so x starts and ends with the same bits as y and therefore $y \rho x$. Transitive: If $x \rho y$ and $y \rho z$, then y starts and ends with the same bits as x and z starts and ends with the same bits as y , so z starts and ends with the same bits as x and therefore $x \rho z$.

b. 256

c. 4; there are 4 start bit-end bit combinations, 0...0, 1...0, 0...1, and 1...1.

d. 2^6

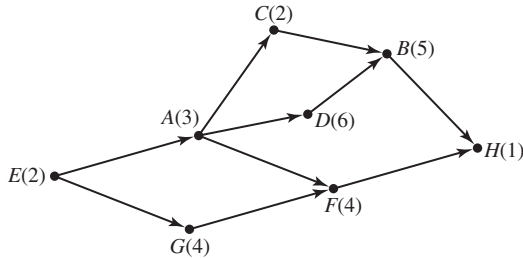
59. Clearly $P \leftrightarrow P$ is a tautology. If $P \leftrightarrow Q$ is a tautology, then P and Q have the same truth values everywhere, so $Q \leftrightarrow P$ is a tautology. If $P \leftrightarrow Q$ and $Q \leftrightarrow R$ are tautologies, then $P, Q,$ and R have the same

truth values everywhere, and $P \leftrightarrow R$ is a tautology. The equivalence classes are sets consisting of wffs with the same truth values everywhere.

61. a. 1 b. 2 c. 5 d. 15
63. Answers agree with Exercise 61.
65. a. 3 b. 7
67. Answers agree with Exercise 65.
69. The number of blocks in a partition can range from 1 (the whole set) to n (a single element in each block). The result follows by the definition of $S(n, k)$ and the addition principle.
71. 6
73. a. 25, 49 b. (3, 4, 5), (0, 5, 5), (8, 6, 10) c. (-4, 4, 2, 0), (-6, 6, 0, -2)
75. 6

EXERCISES 5.2

1. Yes; for example: 1, 2, 3, 8, 4, 5, 6, 7, 9
- 3.



5. Minimum time-to-completion is 17 time units. Critical path: E, A, D, B, H
7. Minimum time to completion is 13 time units. Critical path: E, A, C, B, H
9. For example: G, H, F, D, E, C, A, B
11. For example: E, A, C, D, G, F, B, H
13. For example, 6, 9, 1, 7, 8, 11, 2, 3, 5, 10, 4
15. For example, 5, 6, 4, 1, 2, 3, 7, 10, 8, 11, 9

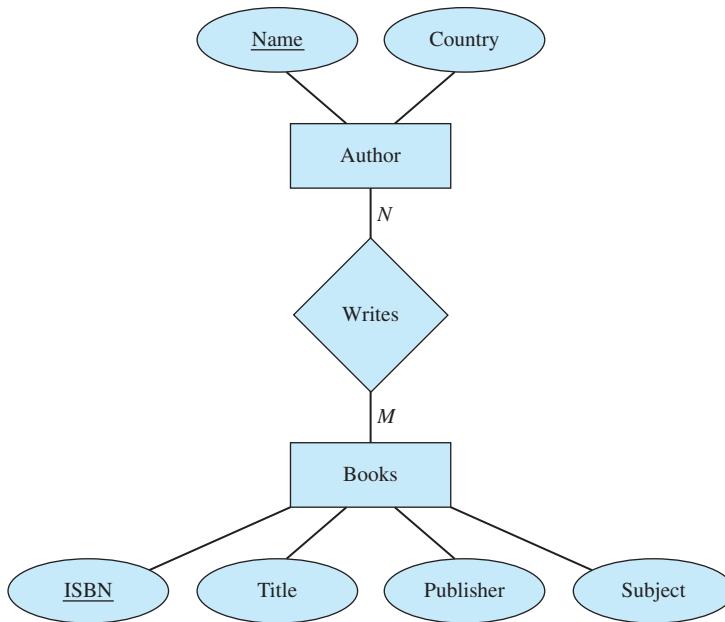
EXERCISES 5.3

1. a. Give the name, type, and breed of all pets that are cats.
b. 2
c. `SELECT PetName, PetType, Breed FROM Pet WHERE PetType = "Cat";`

3. **WhatAml**

<i>PetName</i>	<i>Breed</i>
Spot	Hound
Twinkles	Siamese
Lad	Collie
Lassie	Collie
Mohawk	Moorish idol
Tweetie	Canary
Tiger	Shorthair

5.



The writes relation is many-to-many; that is, one author can write many books, and one book can have more than one author.

7. **Results7**

<i>Name</i>	<i>Country</i>
East, Jane	U.S.
Kovalsco, Bert	U.S.

9. **Results9**

<i>ISBN</i>	<i>Title</i>	<i>Publisher</i>	<i>Subject</i>
0-364-87547-X	Early Tang Paintings	Bellman	Art
0-56-000142-8	Springtime Gardening	Swift-Key	Nature

11. **Results11**

<i>Name</i>
Chan, Jimmy
East, Jane
King, Dorothy
Kovalsco, Bert
Lau, Won
Nkoma, Jon
Quercos, Tom

13. **Results13**

<i>Publisher</i>	<i>Subject</i>
Bellman	Art
Harding	Art
Harding	Nature
Lorraine	Nature
Swift-Key	Nature

In some database systems you have to specify, via something like a “DISTINCT” clause or a “Unique Values” property, that duplicate tuples should be eliminated. Otherwise in this example, (Harding, Nature) would be listed twice.

15. **Results15**

<i>ISBN</i>	<i>Name</i>	<i>Title</i>	<i>Publisher</i>	<i>Subject</i>
0-115-01214-1	Nkoma, Jon	Birds of Africa	Lorraine	Nature
0-364-87547-X	Lau, Won	Early Tang Paintings	Bellman	Art
0-364-87547-X	Chan, Jimmy	Early Tang Paintings	Bellman	Art
0-56-000142-8	East, Jane	Springtime Gardening	Swift-Key	Nature
0-816-35421-9	King, Dorothy	Springtime Gardening	Harding	Nature
0-816-53705-4	Kovalsco, Bert	Baskets for Today	Harding	Art
0-816-88506-0	King, Dorothy	Autumn Annuals	Harding	Nature

17. a. **project(restrict Book where Subject = “Art”) over Title giving Results 17**
 b. **SELECT Title FROM Book WHERE Subject = “Art”;**
 c. Range of x is Book, $\{x.Title|x.Subject = “Art”\}$
 d. **Results17**
- | Title |
|----------------------|
| Baskets for Today |
| Early Tang Paintings |
19. a. **project(join(restrict Book where Publisher = “Harding”) and Writes over ISBN) over Name giving Results19**
 b. **SELECT Name FROM Writes, Book WHERE Book.ISBN = Writes.ISBN AND Publisher = “Harding”;**
 c. Range of x is Writes, Range of y is Book, $\{x.Name|exists y(y.Publisher = “Harding” and y.ISBN = x.ISBN)\}$
 d. **Results19**
- | Name |
|----------------|
| King, Dorothy |
| Kovalsco, Bert |
21. a. **project (join(join(restrict Author where Country = “U.S.”) and Writes over Name) and Book over ISBN) over Title giving Results21**
 b. **SELECT Title FROM Author, Book,Writes WHERE Author.Name = Writes.Name AND Writes.ISBN = Book.ISBN AND Country = “U.S.”;**
 c. Range of x is Book, Range of y is Author, Range of z is Writes, $\{x.Title | exists y, z(y.Country = “U.S.” and y.Name = z.Name and z.ISBN = x.ISBN)\}$
 d. **Results21**
- | Title |
|----------------------|
| Baskets for Today |
| Springtime Gardening |
23. a. **project(join(join(restrict Author where Country = “England”) and Writes over Name) and (restrict Book where Subject = “Art”) over ISBN) over Name, Title giving Results23**
 b. **SELECT Author.Name, Title FROM Book, Author, Writes WHERE Author.Name = Writes.Name AND Writes.ISBN = Book.ISBN AND Country = “England” AND Subject = “Art”;**
 c. Range of x is Author, Range of y is Book, Range of z is Writes, $\{x.Name and y.Title|x.Country = “England” and y.Subject = “Art” and exists z|z.Name = x.Name and y.ISBN = z.ISBN\}$
 d. The empty set; there are no results that match this query.
25. a. $p * q$
 b. If the common attribute is sorted in each table, then the join can be performed by doing something similar to a merge sort (see Exercise 19 in Section 3.3) on the common attribute, which means at most $(p + q)$ rows would need to be examined.
 c. 14
 d. 42
27. a. **SELECT Author.Name, Title, Book.ISBN, RoyaltyPercent FROM Author, Book, Writes WHERE Author.Name = Writes.Name AND Writes.ISBN = Book.ISBN AND RoyaltyPercent <100;**
 b. **ROYALTY LESS THAN 100**

Name	Title	ISBN	RoyaltyPercent
Chan, Jimmy	Early Tang Paintings	0-364-87547-X	20
Lau, Won	Early Tang Paintings	0-364-87547-X	80

29. a. Yes—every attribute described for Employee, Contribution, and Payment is listed and no attributes not mentioned have been included.
- b. The primary key of Employees is *EmployeeID*, presumably a unique identifier for each employee. Likewise, the primary key of Contribution is *ContributionID*. The primary key of Payment is the composite key *ContributionID / PaymentDate*. None of the three attributes alone uniquely identifies a payment, nor does *ContributionID / PaymentAmount* [a given contribution may result in the same payment amount on several different dates] or *PaymentDate / PaymentAmount* [multiple contributions can pay the same amount on the same payment date]. But a given contribution does not have different payment amounts on the same date.
31. a. The three relation tables are

EMPLOYEE

<i>EmployeeID</i>	<i>FirstName</i>	<i>LastName</i>	<i>Department</i>
1	Mary	Black	Accounting
2	June	Brown	Payroll
3	Kevin	White	Accounting
4	Kelly	Chen	Payroll
6	Conner	Smith	Sales

CONTRIBUTION

<i>ContributionID</i>	<i>EmployeeID</i>	<i>ContributionDate</i>	<i>TotalAmount</i>	<i>NumberOfPayments</i>
101	1	1/1/2013	\$300.00	3
102	3	1/1/2013	\$500.00	2
103	6	1/1/2013	\$150.00	2
104	4	4/15/2013	\$100.00	1
105	1	6/1/2013	\$210.00	3
107	2	6/1/2013	\$300.00	2
108	2	1/1/2014	\$600.00	12
109	3	1/1/2014	\$500.00	2

PAYMENT

<i>ContributionID</i>	<i>PaymentDate</i>	<i>PaymentAmount</i>
101	1/15/2013	\$100.00
101	1/31/2013	\$100.00
101	2/15/2013	\$100.00
102	1/15/2013	\$250.00
102	1/31/2013	\$250.00
103	1/15/2013	\$75.00
103	1/31/2013	\$75.00
104	4/30/2013	\$100.00
105	6/15/2013	\$70.00
105	6/30/2013	\$70.00
105	7/15/2013	\$70.00
107	6/15/2013	\$150.00
107	6/30/2013	\$150.00
108	1/15/2014	\$50.00
109	1/15/2014	\$250.00

- b. *EmployeeID* in the Contribution table is a foreign key into the Employee table. *ContributionID* in the Payment table is a foreign key into the Contribution table.
- c. Because the *EmployeeID* is just a sequence of sequential integer values that are likely to have no meaning outside the database, it is probably a blind key generated automatically by the database system.
33. **SELECT** Employee.*EmployeeID*, *PaymentDate*, *PaymentAmount* **FROM** Employee, Contribution, Payment **WHERE** Employee.*EmployeeID* = Contribution.*EmployeeID* **AND** Contribution.*ContributionID* = Payment.*ContributionID* **AND** Payment.*PaymentAmount* > 100; The result is

<i>EmployeeID</i>	<i>PaymentDate</i>	<i>PaymentAmount</i>
2	6/15/2013	\$150.00
2	6/30/2013	\$150.00
3	1/15/2013	\$250.00
3	1/31/2013	\$250.00
3	1/15/2014	\$250.00

35. **SELECT** *FirstName*, *LastName*, *PaymentAmount*, *PaymentDate* **FROM** Employee, Contribution, Payment **WHERE** Employee.*EmployeeID* = Contribution.*EmployeeID* **AND** Contribution.*ContributionID* = Payment.*ContributionID* **AND** Payment.*PaymentDate* = "1/15/2013"; (The last equality is likely to need a system-dependent additional function to convert the 1/15/2013 to a true date type so the equality test will work.) The result is

<i>FirstName</i>	<i>LastName</i>	<i>PaymentAmount</i>	<i>PaymentDate</i>
Mary	Black	\$100.00	1/15/2013
Kevin	White	\$250.00	1/15/2013
Conner	Smith	\$75.00	1/15/2013

EXERCISES 5.4

1. a. Domain = {4, 5, 6, 7, 8} codomain = {8, 9, 10, 11} range = {8, 9, 10}
 b. 8, 10
 c. 6, 7
 d. no, no
3. a. {(0, -1), (1, 1), (2, 3)}
 b. {(1, 1), (2, 3), (4, 7), (5, 9)}
 c. $\{(\sqrt{7}, 2\sqrt{7}-1), (1.5, 2)\}$
5. a. $f(A) = \{3, 9, 15\}$ b. $f(A) =$ all integral multiples of 6
7. a. $f(S) = \{3, 7\}$ b. $f(S) = \{1, 3\}$ c. $f(S) = \{2, 1, 0\}$
9. a. F b. F c. T d. F
11. a. not a function b. function c. function; one-to-one and onto
 d. not a function e. not a function
13. a. not a function b. function, onto, not one-to-one c. function, one-to-one, not onto
15. a. function
 b. not a function
 c. function; onto
 d. bijection; $f^{-1}: \{p, q, r\} \rightarrow \{1, 2, 3\}$ where $f^{-1} = \{(q, 1), (r, 2), (p, 3)\}$
 e. function; one-to-one
 f. bijection; $h^{-1}: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ where $h^{-1}(x, y) = (y - 1, x - 1)$

17. Any odd value of n will produce a bijection because the function will look similar to Figure 5.12. For even values of n the function is not one-to-one because positive and negative values map to the same result.
19. f is neither one-to-one nor onto. For example, $f(xxy) = f(yyy) = 3$, so f is not one-to-one. For any string s , $f(s) \geq 0$; there are no strings in A^* that map to negative values, so f is not onto.
21. f is both one-to-one and onto. If $f(s_1) = f(s_2)$ then $s_1 = s_2$ (just reverse the strings again and you get back where you started), so f is one-to-one. Given any string s in A^* , let y be its reverse. Then y is in A^* and $f(y) = s$, so f is onto.
23. f is neither one-to-one nor onto. For example, $f(\{a, b\}) = f(\{b, c\}) = 2$, so f is not one-to-one. The range of f is the set of sizes of all subsets of $\{a, b, c\}$, which is $\{0, 1, 2, 3\}$, so this is clearly not \mathbb{Z} and f is not onto.
25. For example, $f(x) = 1/x$. For $x \geq 1$, the value $1/x$ is greater than 0 but less than or equal to 1, so $f: S \rightarrow T$. If $f(x_1) = f(x_2)$, then $1/x_1 = 1/x_2$ and $x_1 = x_2$, so f is one-to-one. Given any value y in T , that is, $0 < y \leq 1$, the value $1/y$ is in S and $f(1/y) = 1/(1/y) = y$, so f is onto.
27. a. 3 b. 0 c. 0
29. The greatest integer $\leq x$ is the same as the smallest integer $\geq x$; therefore x is an integer.
31. Let $k \leq x < k + 1$ where k is an integer. Then k is the greatest integer that is less than or equal to x , so $\lfloor x \rfloor = k$. Also, multiplying the entire inequality by -1 , which reverses the direction of the inequality signs, gives $-k \geq -x > -k - 1$ which means that $-k$ is the smallest integer greater than or equal to $-x$, so $\lceil -x \rceil = -k$. Multiplying both sides of this equation by -1 gives $-\lceil -x \rceil = k$.
33. a. False. Let $x = 3.6$. Then $\lceil \lfloor x \rfloor \rceil = \lceil 3 \rceil = 3 \neq x$.
b. False. Let $x = 4.8$. Then $\lfloor 2x \rfloor = \lfloor 9.6 \rfloor = 9$ but $2\lfloor x \rfloor = 2(4) = 8$.
35. If $2^k < n < 2^{k+1}$ then $\log(2^k) < \log n < \log(2^{k+1})$ or $k < \log n < k + 1$ and $\lfloor \log n \rfloor = k$, $\lceil \log n \rceil = k + 1$.
37. a. 9 b. 0 c. 4 d. 2
39. False. For example, let $x = 7$ and $y = 9$. Then $x \bmod 10 + y \bmod 10 = 7 + 9 = 16$ but $(x + y) \bmod 10 = 16 \bmod 10 = 6$.
41. a. $(1, 1), (2, 0), (3, 1), (4, 0), (5, 1)$
b. $c_{A \cap B}(x) = 1 \leftrightarrow x \in A$ and $x \in B \leftrightarrow c_A(x) = 1$ and $c_B(x) = 1 \leftrightarrow c_A(x) \cdot c_B(x) = 1$
c. If $c_{A'}(x) = 1$, then $x \in A'$ and $x \notin A$, so $c_A(x) = 0 = 1 - c_{A'}(x)$. If $c_{A'}(x) = 0$, then $x \notin A'$ and $x \in A$ so $c_A(x) = 1 = 1 - c_{A'}(x)$
d. No. Let $S = \{1, 2, 3\}$, $A = \{1, 2\}$, $B = \{2, 3\}$. Then $c_{A \cup B}(2) = 1$ but $c_A(2) + c_B(2) = 1 + 1$.
43. a. $S(0, n) > A(0, n)$ for $n > 1$ because $n^n > n + 1$.
b. $S(1, n) = S(0, S(0, n)) = S(0, n^n) = n^{n^n}$
c. The googolplex is $10^{10^{100}} = 10^{10^{100}} = S(1, 10)$
45. $g \circ f = \{(1, 6), (2, 7), (3, 9), (4, 9)\}$
47. a. 18 b. 16 c. $3x + 3$
d. $3x + 1$ e. $x + 2$ f. $9x$
49. a. If $f(s_1) = f(s_2)$ then $g(f(s_1)) = g(f(s_2))$ so $(g \circ f)(s_1) = (g \circ f)(s_2)$. Because $g \circ f$ is one-to-one, $s_1 = s_2$ and therefore f is one-to-one.
b. For $u \in U$, there exists $s \in S$ such that $(g \circ f)(s) = u$, because $g \circ f$ is onto. Thus $g(f(s)) = u$ and $f(s)$ is a member of T that is a preimage of u under g , and g is onto.
c. Let $S = \{1, 2, 3\}$, $T = \{1, 2, 3, 4\}$, $U = \{1, 2, 3\}$, $f = \{(1, 1), (2, 2), (3, 3)\}$, $g = \{(1, 1), (2, 2), (3, 3), (4, 3)\}$. Then $f: S \rightarrow T$, $g: T \rightarrow U$, g is not one-to-one but $g \circ f = \{(1, 1), (2, 2), (3, 3)\}$ is one-to-one.
d. same example as for (c)
51. a. $f^{-1}(x) = x/2$ b. $f^{-1}(x) = \sqrt[3]{x}$ c. $f^{-1}(x) = 3x - 4$

53. a. (1, 3, 5, 2) b. (1, 4, 3, 2, 5)
55. Both $h \circ (g \circ f)$ and $(h \circ g) \circ f$ have domain and codomain A . For $x \in A$, $(h \circ (g \circ f))(x) = h((g \circ f)(x)) = h(g(f(x))) = (h \circ g)(f(x)) = ((h \circ g) \circ f)(x)$.
57. a. (1, 2, 5, 3, 4)
 b. $(1, 7, 8) \circ (2, 4, 6)$
 c. $(1, 5, 2, 4) \circ (3, 6)$
 d. $(2, 3) \circ (4, 8) \circ (5, 7)$
59. a. (a, d, e, b) b. (d, e) c. $(a, d) \circ (c, e)$
61. $f^{-1} = (2, 4, 3, 8)$
63. a. 3^4 b. 36
65. a. For $|S| = 2$, $2!/0! = 2$ and $2^2 - C(2, 1)(1)^2 = 4 - 2 = 2$. For $|S| = 3$, $3!/0! = 6$ and $3^3 - C(3, 1)(2)^3 + C(3, 2)(1)^3 = 27 - 3 \cdot 8 + 3 = 6$. For $|S| = 4$, $4!/0! = 24$ and $4^4 - C(4, 1)(3)^4 + C(4, 2)(2)^4 - C(4, 3)(1)^4 = 256 - 4 \cdot 81 + 6 \cdot 16 - 4 = 24$.
 b. Assume f is onto. If two distinct elements of S map to one element of S , then $n - 2$ elements are left to map onto $n - 1$ elements, which cannot be done. Therefore f is one-to-one. Now assume f is one-to-one. Then the n elements of S map to n distinct elements of S ; thus every element of S is in the range of f , and f is onto.
 c. For example, $S = \mathbb{N}$, $f: \mathbb{N} \rightarrow \mathbb{N}$ given by $f(x) = 2x$.
 d. For example, $S = \mathbb{N}$, $f: \mathbb{N} \rightarrow \mathbb{N}$ given by $f(0) = 0, f(x) = x - 1$ for $x \geq 1$.
67. a. n^n
 b. $n!$
 c. $n!$
 d. $n!$
 e. $n! \left[\frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!} \right]$
 f. The number of derangements (answer d) is $< n! \left[\frac{1}{2!} \right] = n! \cdot \frac{1}{2} < n!$ and $n! < n^n$. The total number of functions, with no restrictions, is the maximum. Only some of these functions are one-to-one and onto, but this is the definition of a permutation as well. Not all permutations are derangements, so the number of derangements is smaller still.
69. 1854
71. a. For $x \in S, f(x) = f(x)$, so $x \rho x$ and ρ is reflexive. For $x, y \in S$, if $x \rho y$ then $f(x) = f(y)$ and $f(y) = f(x)$, so $y \rho x$ and ρ is symmetric. For $x, y, z \in S$, if $x \rho y$ and $y \rho z$, then $f(x) = f(y)$ and $f(y) = f(z)$, so $f(x) = f(z)$ and $x \rho z$, so ρ is transitive.
 b. If f is a one-to-one function, then no two elements of S map to the same value, so the equivalence classes each consist of a single element.
 c. $[4] = \{4, -4\}$ because $f(4) = f(-4)$.
73. a. $\{m, n, o, p\}$ b. $\{n, o, p\}; \{o\}$
75. Reflexive: $S \rho S$ by the identify function. Symmetric: If $S \rho T$ and f is a bijection from S to T , then $f^{-1}: T \rightarrow S$ and f^{-1} is a bijection, so $T \rho S$. Transitive: If $S \rho T$ and $T \rho U, f: S \rightarrow T, g: T \rightarrow U, f$ and g bijections, then $g \circ f: S \rightarrow U$ and $g \circ f$ is a bijection, so $S \rho U$.
77. a. (define (square x)
 (* xx)) b. 18

EXERCISES 5.5

1. For example, $n_0 = 1, c_1 = 1/34, c_2 = 1$. For $x \geq 1$, $(1/34)(17x + 1) \leq x \leq 1(17x + 1)$
3. For example, $n_0 = 2, c_1 = 1, c_2 = 2$. For $x \geq 2$, $1(15x^2 + x) \leq 29x^2 - 4x - 15 \leq 2(15x^2 + x)$
5. For example, $n_0 = 1, c_1 = 1, c_2 = 2$. For $x \geq 1$, $1(x^3) \leq x^3 + \log x \leq 2x^3$.

7. Yes. For example, in Exercise 1, we could use the constants $n_0 = 1$, $c_1 = 1/34$, $c_2 = 1/10$. Then the envelope would be entirely below $g(x)$, but it still follows the general “shape” of $g(x)$.

9. 2 11. $\lim_{x \rightarrow \infty} \frac{x}{17x + 1} = \lim_{x \rightarrow \infty} \frac{1}{17} = \frac{1}{17}$

13. $\lim_{x \rightarrow \infty} \frac{x}{x^2} = \lim_{x \rightarrow \infty} \frac{1}{2x} = 0$ 15. $\lim_{x \rightarrow \infty} \frac{\log x}{x} = \lim_{x \rightarrow \infty} \frac{\frac{1}{x} \log e}{1} = 0$

17. $[200 \log x] = [41 \ln x^2] < [\sqrt[4]{x}] < [420x] < [17x \log x] < [10x^2 - 3x + 5] < [2^x x^2]$

19. Knowing that the algorithm is $O(n^3)$ tells you only that the growth rate is less than or equal to n^3 ; it could be n^3 , n^2 , $n \log n$, n , etc. Knowing that it is $o(n^3)$ tell you that the growth rate is less than n^3 , but again it could be n^2 , $n \log n$, etc. The most useful information is that the growth rate is $\Theta(n^2)$, essentially growing at a constant times a parabolic

21. $S(n) = \Theta(n^2)$ 23. $S(n) = \Theta(n \log n)$

25. $S(n) = \Theta(n^{\log_3 3}) = \Theta(n)$

27. a. $C'(n) = \Theta(n \log n)$.

b. The exact solution for $C(n)$ is $C(n) = n(\log n) - n + 1$, which is also $\Theta(n \log n)$.

EXERCISES 5.6

1. $25 \bmod 6 = 5$, $11 \bmod 6 = 5$, $14 \bmod 6 = 2$, and $(5 + 2) \bmod 6 = 1$

3. $262 \bmod 13 = 2$, $74 \bmod 13 = 9$, $188 \bmod 13 = 6$, and $(9 + 6) \bmod 13 = 2$

5. $486 \bmod 5 = 1$, $18 \bmod 5 = 3$, $27 \bmod 5 = 2$, and $(3 \cdot 2) \bmod 5 = 1$

7. Let $x = q_1n + r_1$, $0 \leq r_1 < n$ and $y = q_2n + r_2$, $0 \leq r_2 < n$, so $x \bmod n = r_1$ and $y \bmod n = r_2$. Then $x \cdot y = (q_1q_2n + q_2r_1 + q_1r_2)n + (r_1 \cdot r_2)$, $0 \leq r_1 \cdot r_2 < n^2$. Let $r_1 \cdot r_2 = kn + r$ with $0 \leq r < n$. Then $x \cdot y = (q_1q_2n + q_2r_1 + q_1r_2 + k)n + r$ with $0 \leq r < n$, so $(x \cdot y) \bmod n = r$. Also $x \bmod n \cdot y \bmod n = r_1 \cdot r_2 = kn + r$ with $0 \leq r < n$, so $(x \bmod n \cdot y \bmod n) \bmod n = r$.

9. Answer c is correct.

11.

0	33
1	1
2	13
3	12
4	34
5	38
6	27
7	22
8	
9	
10	

13. a. The values are stored in locations 6, 14, 1, 7, 8, 2, 16, 9, 0.

0	50
1	52
2	18
3	
4	
5	
6	23
7	40
8	24
9	58
10	
11	
12	
13	
14	14
15	
16	33

b. 58 hashes to location 7, which contains another element (40), so, following the collision resolution scheme under which 58 would have been stored, search the next table position, 8, which contains 24, then search the next table position, 9, which contains 58. 41 also hashes to location 7 in the table; proceeding as before, locations 8 and 9 are also checked, and do not contain 41. The next location to check is 10, which is empty. Therefore 41 is not in the table.

15. a. $1/t$ b. $2/t$ c. $3/t$
17. a. ALLS WELL THAT ENDS WELL
 b. TWAS BRILLIG AND THE SLITHY TOVES DID GYRE AND GIMBLE IN THE WABE
 c. COL MUSTARD WITH THE KNIFE IN THE LIBRARY
19. $k = 9$, SLEEP NOW WE MARCH ON ROME TOMORROW
21. a. $x = 10011$ (5-bit string), $p = x \bmod 2^4 = 00011$, $q = p \cdot 2 = 00110$, $s = x \oplus p = 10000$,
 $t = s \cdot 2^{-4} = 00001$, $y = q + t = 00111$
 b. $x = 0011$ (4-bit string), $p = x \bmod 2^3 = 0011$, $q = p \cdot 2 = 0110$, $s = x \oplus p = 0000$,
 $t = s \cdot 2^{-4} = 0000$, $y = q + t = 0110$
23. 1010001111101010
25. a. $d = 3$
 b. $8^3 \bmod 15 = 8^2 \cdot 8 \bmod 15 = 64 \cdot 8 \bmod 15 = 4 \cdot 8 \bmod 15 = 32 \bmod 15 = 2$
 c. $2^3 \bmod 15 = 8$
27. a. $d = 23$ b. $12^7 \bmod 55 = 23$ c. $23^{23} \bmod 55 = 12$
29. a. $n(n - 1)/2$ b. n
31. a. X b. 7
33. check digit = 8
35. a. temp = number
 ones = temp mod 10
 temp = (temp - one)/10
 tens = temp mod 10
 temp = (temp - tens)/10
 hundreds = temp mod 10
 temp = (temp - hundreds)/10
 thousands = temp
 b. temp = 7426
 ones = 6
 temp = 742
 tens = 2
 temp = 74
 hundreds = 4
 temp = 7
 thousands = 7
37. If $x \equiv y \pmod{n}$ then $x - y = kn$ where k is an integer. Therefore $xc - yc = kcn$ where kc is an integer, so $xc \equiv yc \pmod{n}$.
39. a. $f(ka) = (ka) \bmod p$ maps each element of S to a unique value in T . To show that f is one-to-one, suppose $f(k_1a) = f(k_2a)$ for some k_1 and k_2 , $0 \leq k_1, k_2 \leq p - 1$. Then $k_1a \bmod p = k_2a \bmod p$. By Practice 43b, $k_1a \equiv k_2a \pmod{p}$. Because p is a prime number and a is not divisible by p , $\gcd(a, p) = 1$, so by Exercise 38 (cancellation under congruence modulo n), $k_1 \equiv k_2 \pmod{p}$, or $k_1 - k_2 = mp$ for some integer m . But $-p < k_1 - k_2 < p$ so $m = 0$ and $k_1 = k_2$, so $k_1a = k_2a$ and f is one-to-one.
 b. $|S| = |T| = p$ and f is a one-to-one function. Therefore each of the p distinct elements of S map to distinct elements of T , making each of the p elements of T the image of an element of S .
 c. $[a \cdot 2a \cdots (p - 1)a] \bmod p = [(a \bmod p) \cdot (2a \bmod p) \cdots ((p - 1)a \bmod p)] \bmod p$ by Equation (1) of this section. Because f is an onto function, the set T is the set of all residues modulo p of the elements of S . Because $0 \bmod p = 0$, the set $\{1, 2, \dots, (p - 1)\}$ is the set of all residues modulo p of the elements $\{a, 2a, \dots, (p - 1)a\}$. Therefore $[(a \bmod p) \cdot (2a \bmod p) \cdots ((p - 1)a \bmod p)] \bmod p = [1 \cdot 2 \cdots (p - 1)] \bmod p = (p - 1)! \bmod p$.
 d. From part (c), $[a^{p-1}(p - 1)!] \bmod p = (p - 1)! \bmod p$ or, by Practice 43b, $a^{p-1}(p - 1)! \equiv (p - 1)! \pmod{p}$. Because p is a prime, $\gcd((p - 1)!, p) = 1$, so by Exercise 38 (cancellation under congruence modulo n), $a^{p-1} \equiv 1 \pmod{p}$.
 e. $\{1, 2, 3, 4, 5, 6\}$
 f. $4^6 \bmod 7 = (4^3 \cdot 4^3) \bmod 7 = 64 \cdot 64 \bmod 7 = 1 \cdot 1 \bmod 7 = 1$
41. a. $d \cdot e \equiv 1 \pmod{\varphi(n)}$ means $d \cdot e \equiv 1 \pmod{(p - 1)(q - 1)}$, so that $d \cdot e = 1 + k(p - 1)(q - 1)$ for some integer k . Then $T^{ed} = T^{1 + k(p-1)(q-1)} = T(T^{k(p-1)(q-1)}) = T(T^{p-1})^{k(q-1)}$ or $T(T^{q-1})^{k(p-1)}$.

- b. If T is not divisible by p , then $T^{p-1} \equiv 1 \pmod{p}$ by Fermat's little theorem (Exercise 39). $T^{ed} \pmod{p} = T(T^{p-1})^{k(q-1)} \pmod{p} = [T(T^{p-1})(T^{p-1}) \cdots (T^{p-1})] \pmod{p} = [(T \pmod{p}) \cdot 1 \cdot 1 \cdots 1] \pmod{p}$ by Equation (1) and Practice 43b $= T \pmod{p}$ so that $T^{ed} \equiv T \pmod{p}$ by Practice 43b. Similarly if T is not divisible by q , then $T^{q-1} \equiv 1 \pmod{q}$. $T^{ed} \pmod{q} = T(T^{q-1})^{k(p-1)} \pmod{q}$ and $T^{ed} \equiv T \pmod{q}$.
- c. If $p|T$ then $T = kp$ for some integer k and $T^{ed} - T = (kp)^{ed} - kp = p(p^{ed-1}k^{ed} - k)$ where $p^{ed-1}k^{ed} - k$ is an integer, so $T^{ed} \equiv T \pmod{p}$. If $p|T$ and $q|T$, then T is a multiple of p and T is a multiple of q . Because p and q are primes, $T = cpq$ for some integer c , which is a contradiction because $T < pq = n$.
- d. The proof is very similar to part c.
- e. $T^{ed} \equiv T \pmod{p}$, $T^{ed} \equiv T \pmod{q}$ and p and q are relatively prime. This matches the pattern of the Chinese remainder theorem where $a_1 = a_2 = T$ and $x = T^{ed}$. But also, $T \equiv T \pmod{p}$ and $T \equiv T \pmod{q}$. By the Chinese remainder theorem, $T^{ed} \equiv T \pmod{pq}$, or $T^{ed} \equiv T \pmod{n}$. By Practice 43b, $T^{ed} \pmod{n} = T \pmod{n} = T$ because $T < n$.

EXERCISES 5.7

1. 2, -4

3. $x = 1, y = 3, z = -2, w = 4$

5. a. $\begin{bmatrix} 6 & -5 \\ 0 & 3 \\ 5 & 3 \end{bmatrix}$

d. $\begin{bmatrix} -4 & -8 \\ -12 & 2 \end{bmatrix}$

b. $\begin{bmatrix} -2 & 7 \\ -2 & -3 \\ 1 & 5 \end{bmatrix}$

e. $\begin{bmatrix} 14 & -17 \\ 2 & 9 \\ 9 & 1 \end{bmatrix}$

c. $\begin{bmatrix} 12 & 3 & 6 \\ 18 & -3 & 15 \\ 3 & 9 & 6 \end{bmatrix}$

7. a. $\begin{bmatrix} 10 & 7 \\ -2 & -4 \\ 30 & 8 \end{bmatrix}$

c. $\begin{bmatrix} 21 & -23 \\ 33 & -44 \\ 11 & 1 \end{bmatrix}$

b. not possible

d. $\begin{bmatrix} 28 & 4 \\ 6 & 25 \end{bmatrix}$

9. a. $\mathbf{A} \cdot \mathbf{B} = \begin{bmatrix} 10 & 4 \\ 18 & -3 \end{bmatrix}$ $\mathbf{B} \cdot \mathbf{A} = \begin{bmatrix} 14 & 1 \\ 4 & -7 \end{bmatrix}$

b. $\mathbf{A}(\mathbf{B} \cdot \mathbf{C}) = (\mathbf{A} \cdot \mathbf{B})\mathbf{C} = \begin{bmatrix} 68 & -58 \\ 102 & -84 \end{bmatrix}$

c. $\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{A} \cdot \mathbf{B} + \mathbf{A} \cdot \mathbf{C} = \begin{bmatrix} 26 & -9 \\ 40 & -23 \end{bmatrix}$

d. $(\mathbf{A} + \mathbf{B})\mathbf{C} = \mathbf{A} \cdot \mathbf{C} + \mathbf{B} \cdot \mathbf{C} = \begin{bmatrix} 42 & -35 \\ 32 & -28 \end{bmatrix}$

11. Both $\mathbf{A} \cdot (\mathbf{B} \cdot \mathbf{C})$ and $(\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{C}$ can be computed and will result in an $n \times m$ matrix. The i, j element of $\mathbf{A} \cdot (\mathbf{B} \cdot \mathbf{C})$ looks like

$$\begin{aligned} \sum_{s=1}^p a_{is}(\mathbf{B} \cdot \mathbf{C})_{sj} &= \sum_{s=1}^p a_{is} \left(\sum_{k=1}^r b_{sk}c_{kj} \right) = a_{i1}(b_{11}c_{1j} + b_{12}c_{2j} + \cdots + b_{1r}c_{rj}) + \cdots + \\ a_{ip}(b_{p1}c_{1j} + b_{p2}c_{2j} + \cdots + b_{pr}c_{rj}) &= (a_{i1}b_{11} + a_{i2}b_{21} + \cdots + a_{ip}b_{p1})c_{1j} + \cdots + \\ (a_{i1}b_{1r} + a_{i2}b_{2r} + \cdots + a_{ip}b_{pr})c_{rj} &= \sum_{k=1}^r \left(\sum_{s=1}^p a_{is}b_{sk} \right) c_{kj} = \sum_{k=1}^r (\mathbf{A} \cdot \mathbf{B})_{ik}c_{kj} \end{aligned}$$

which is the i, j element of $(\mathbf{A} \cdot \mathbf{B}) \cdot \mathbf{C}$.

13. a. Assume that row i of \mathbf{A} is all 0s. Then for any j , the element in row i , column j of $\mathbf{A} \cdot \mathbf{B}$ is given by

$$\sum_{k=1}^n a_{ik}b_{kj}. \text{ This sum is 0 because } a_{ik} = 0 \text{ for all values of } k.$$

- b. Assume that column j of \mathbf{B} is all 0s. Then for any i , the element in row i , column j of $\mathbf{A} \cdot \mathbf{B}$ is given by

$$\sum_{k=1}^n a_{ik}b_{kj}. \text{ This sum is 0 because } b_{kj} = 0 \text{ for all values of } k.$$

15. a. $\mathbf{A}^T = \begin{bmatrix} 1 & 6 \\ 3 & -2 \\ 4 & 1 \end{bmatrix}$

- b. If \mathbf{A} is symmetric then $a_{ij} = a_{ji}$ and $\mathbf{A}^T(i, j) = \mathbf{A}(j, i) = \mathbf{A}(i, j)$. Therefore $\mathbf{A}^T = \mathbf{A}$. If $\mathbf{A}^T = \mathbf{A}$, then $\mathbf{A}(i, j) = \mathbf{A}^T(i, j) = \mathbf{A}(j, i)$ and \mathbf{A} is symmetric.

- c. $(\mathbf{A}^T)^T = \mathbf{A}$ follows from the definition - two interchanges of row and column gets back to the original.

- d. Let $\mathbf{A} + \mathbf{B} = \mathbf{C}$. Then $\mathbf{C}^T(i, j) = \mathbf{C}(j, i) = \mathbf{A}(j, i) + \mathbf{B}(j, i) = \mathbf{A}^T(i, j) + \mathbf{B}^T(i, j)$ and $\mathbf{C}^T = \mathbf{A}^T + \mathbf{B}^T$.

- e. Let \mathbf{A} be an $n \times m$ matrix and \mathbf{B} be an $m \times p$ matrix; then \mathbf{A}^T is $m \times n$ and \mathbf{B}^T is $p \times m$. Let $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$.

$$\text{Then } \mathbf{C}^T(i, j) = \mathbf{C}(j, i) = \sum_{k=1}^m a_{jk}b_{ki} = \sum_{k=1}^m \mathbf{A}^T(k, j)\mathbf{B}^T(i, k) = \sum_{k=1}^m \mathbf{B}^T(i, k)\mathbf{A}^T(k, j) = (\mathbf{B}^T \cdot \mathbf{A}^T)(i, j) \text{ and } \mathbf{C}^T = \mathbf{B}^T \cdot \mathbf{A}^T.$$

17. For example, $\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

19. This is not always true (for example, use the \mathbf{A} and \mathbf{B} of Practice 52). It is true if $\mathbf{A} = \mathbf{B} = \mathbf{I}$, for example.

21. The i, j entry of \mathbf{A}^2 is $\sum_{k=1}^n a_{ik}a_{kj}$. The j, i entry of \mathbf{A}^2 is $\sum_{k=1}^n a_{jk}a_{ki}$. But these are the same because $a_{ik} = a_{ki}$ and $a_{kj} = a_{jk}$ (\mathbf{A} is symmetric).

23. For $n = 1$, $\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} F(2) & F(1) \\ F(1) & F(0) \end{bmatrix}$. Assume that $\mathbf{A}^k = \begin{bmatrix} F(k+1) & F(k) \\ F(k) & F(k-1) \end{bmatrix}$. Then

$$\begin{aligned} \mathbf{A}^{k+1} = \mathbf{A}^k \cdot \mathbf{A} &= \begin{bmatrix} F(k+1) & F(k) \\ F(k) & F(k-1) \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} F(k+1) + F(k) & F(k+1) \\ F(k) + F(k-1) & F(k) \end{bmatrix} = \\ \begin{bmatrix} F(k+2) & F(k+1) \\ F(k+1) & F(k) \end{bmatrix}. \end{aligned}$$

25. $(r\mathbf{A})(1/r)\mathbf{A}^{-1} = r(1/r)(\mathbf{A} \cdot \mathbf{A}^{-1}) = \mathbf{1I} = \mathbf{I}$ and $(1/r)\mathbf{A}^{-1}(r\mathbf{A}) = (1/r)(r)(\mathbf{A}^{-1} \cdot \mathbf{A}) = \mathbf{1I} = \mathbf{I}$.

27. $x = 6, y = -1$

29. $x = 11, y = 6, z = -2$

31. $x = 9/14, y = 2/7, z = 31/14$

33. $x = 5, y = -3, z = 4, w = 2.$

35. For example,
 $x + 2y = 3$
 $4x + y = 19$
 $3x - y = 16$

The augmented matrix is $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 1 & 19 \\ 3 & -1 & 16 \end{bmatrix}$. The following elementary row operations

-3 $\begin{matrix} \swarrow \\ \searrow \end{matrix}$ $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 1 & 19 \\ 3 & -1 & 16 \end{bmatrix}$

result in

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -7 & 7 \\ 0 & -7 & 7 \end{bmatrix}$$

Because row 2 of the augmented matrix is the sum of rows 1 and 3, the new rows 2 and 3 represent the same equation, so this actually is a system of 2 equations in 2 unknowns. Solving $-7y = 7$ gives $y = -1$, and solving $x + 2y = 3$ or $x + 2(-1) = 3$ gives $x = 5$.

37. Let $g =$ the amount of gold in cubic centimeters, and $c =$ the amount of copper in cubic centimeters. Then $c + g = 52$ and $9c + 19.3g = 859.4$. The solution is $g = 38$ cc, $c = 14$ cc.

The percentage of copper by volume is $14/52 = 26.9\%$.

39. $\mathbf{A}^{-1} = \begin{bmatrix} -1/2 & 3/4 \\ 1/2 & -1/4 \end{bmatrix}$

41. $\mathbf{A}^{-1} = \begin{bmatrix} -14/10 & 1/10 \\ 24/10 & -1/10 \end{bmatrix}$ $\mathbf{A}^{-1} \cdot \mathbf{B} = \begin{bmatrix} 20 \\ 50 \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$

so $x = 20, y = 50$.

43. $x = 6, y = -1$

45. $x = 11, y = 6, z = -2$

47. $\mathbf{A} \wedge \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$ $\mathbf{A} \vee \mathbf{B} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ $\mathbf{A} \times \mathbf{B} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ $\mathbf{B} \times \mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

49. $\mathbf{A} \wedge \mathbf{B} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$ $\mathbf{A} \vee \mathbf{B} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$ $\mathbf{A} \times \mathbf{B} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}$ $\mathbf{B} \times \mathbf{A} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

51. In order for $\mathbf{A} \vee \mathbf{B} = \mathbf{A} \wedge \mathbf{B}$, it must be the case that $a_{ij} \vee b_{ij} = a_{ij} \wedge b_{ij}$ for all i, j . This is true if $a_{ij} = b_{ij} = 1$ or $a_{ij} = b_{ij} = 0$, therefore when $\mathbf{A} = \mathbf{B}$.

53. $2^{\frac{n(n+1)}{2}}$

55. a. The rows in the augmented matrix as it is being transformed are of length $n + 1, n, n - 1, \dots, 3$. (The augmented matrix is size $n \times (n + 1)$ and the next to last row is the last for which such a multiplication is required in order to zero out position $n, n - 1$ in the last row.) Each non-zero element in each row must be multiplied by the scalar value, requiring a total of $(n + 1) + n + (n - 1) + \dots + 3$ which is $(n + 1) + n + (n - 1) + \dots + 3 + 2 + 1 - (2 + 1)$ multiplications, which is $\frac{(n + 1)(n + 2)}{2} - 3$.

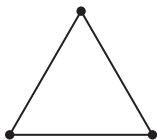
- b. To zero out the first column, a multiple of the first row (length $n + 1$) must be added to each row below it, requiring $(n + 1)$ multiplications and $(n + 1)$ additions for $n - 1$ rows. To zero out the second column, a multiple of the second row (length n) must be added to each row below it, requiring n multiplications and n additions for $n - 2$ rows. The last transformation requires a multiple of the next to last row (length 3) to be added to the 1 row below it, requiring 3 multiplications and 3 additions for 1 row. The totals are $(n + 1)(n - 1) + (n)(n - 2) + \cdots + (3)(1)$ for both multiplications and additions, and this expression (from Exercise 11 in Section 2.2) equals $\frac{(n - 1)(n)(2(n - 1) + 7)}{6} = \frac{2n^3 + 3n^2 - 5n}{6}$
- c. To solve the equations from bottom to top requires
- Row n : $c_n x_n = d_n$ 1 multiplication
 Row $n - 1$: $c_{(n-1)(n-1)} x_{n-1} + c_{(n-1)n} x_n = d_{n-1}$ 1 multiplication, 1 addition, 1 multiplication
 Row $n - 2$: $c_{(n-2)(n-2)} x_{n-2} + c_{(n-2)(n-1)} x_{n-1} + c_{(n-2)n} x_n = d_{n-2}$ 2 multiplications, 2 additions, 1 multiplication
 \vdots
 Row 1: $c_{11} x_1 + c_{12} x_2 + \cdots + c_{1n} x_n = d_1$ $(n - 1)$ multiplications, $(n - 1)$ additions, 1 multiplication
 for a total of $1 + 2 + 3 + \cdots + n = \frac{n(n + 1)}{2}$ multiplications and $1 + 2 + \cdots + (n - 1) = \frac{(n - 1)n}{2}$ additions.
- d. The transformation requires $\Theta(n^2) + 2\Theta(n^3)$ operations, and the resulting equation solving requires $2\Theta(n^2)$ operations, so the overall order of magnitude is $\Theta(n^3)$.

CHAPTER 6

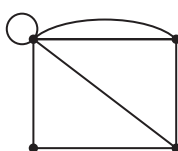
EXERCISES 6.1

1. $g(a) = (1,2), g(b) = (1,3), g(c) = (2,3), g(d) = (2,2)$

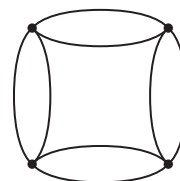
3. a.



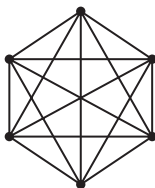
b. For example,



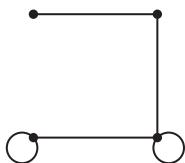
c.



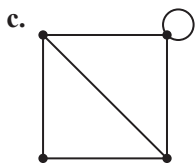
5.



7. a. For example,



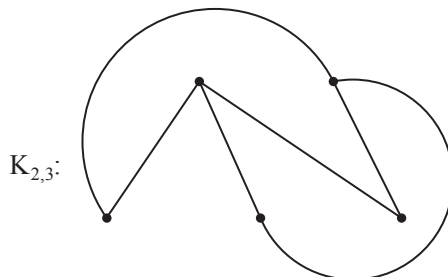
- b. Does not exist; the node of degree 4 would have to have arcs going to 4 distinct other nodes because no loops or parallel arcs are allowed, but there are not 4 distinct other nodes.



- d. Does not exist; in such a graph, the sum of all the degrees would be 11, but the sum of all the degrees is the total number of arc ends, which must be twice the number of arcs, i.e., an even number.
9. a. Because everyone in a department presumably knows someone in the same department, this would mean that no one in the IT department knows anyone in the marketing department (and vice versa).
 b. Carl and Fletcher are not acquainted. SiuYin is acquainted only with Carl.
 c. 2
11. For example: a. star \rightarrow idol \rightarrow statue \rightarrow sculpture
 b. burden \rightarrow load \rightarrow weight \rightarrow influence
 c. piano \rightarrow upright \rightarrow moral \rightarrow significance
13. (b), because there is no node of degree 0.
15. $f_1: 1 \rightarrow a, 2 \rightarrow b, 3 \rightarrow c, 4 \rightarrow d, f_2: a_1 \rightarrow e_2, a_2 \rightarrow e_7, a_3 \rightarrow e_6, a_4 \rightarrow e_1, a_5 \rightarrow e_3, a_6 \rightarrow e_4, a_7 \rightarrow e_5$
17. $f: 1 \rightarrow a, 2 \rightarrow d, 3 \rightarrow b, 4 \rightarrow e, 5 \rightarrow c$
19. Not isomorphic; graph in (b) has a node of degree 5, graph in (a) does not.
21. a. There cannot be a bijection between the two node sets if they are not the same size.
 b. For isomorphic graphs there is a bijection from one arc set to the other, either explicitly or, in the case of simple graphs, implicitly by means of the endpoints; this cannot happen if the arc sets are not the same size.
 c. If the graphs are isomorphic and arcs a_1 and a_2 in one graph both have endpoints $x-y$, then their image arcs in the second graph must have the same endpoints, which cannot happen if the second graph has no parallel arcs.
 d. If the graphs are isomorphic and an arc in one graph has endpoints $x-x$, then its image arc in the second graph must have endpoints $f(x)-f(x)$, which is not possible if the second graph has no loops.
 e. If the graphs are isomorphic and a node of degree k in one graph serves as an endpoint to k arcs, its image in the second graph must serve as an endpoint to the images of those k arcs, which implies it will have degree k also.
 f. If the graphs are isomorphic and if there is a path $n_1, a_1, n_2, a_2, \dots, n_k$ between two nodes in one graph, then $f(n_1), f(a_1), f(n_2), f(a_2), \dots, f(n_k)$ is a path in the second graph. Two nodes in the second graph are the images of nodes in the first graph; if the first graph is connected, there is a path between these nodes and hence there is a path between the two nodes in the second graph.
 g. By the answer to part f, in isomorphic graphs paths map to paths, so cycles map to cycles.

23. 4 graphs:  25. $\frac{n(n-1)}{2} = C(n,2)$

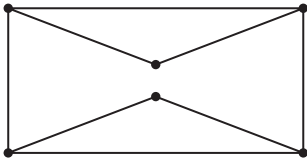
27. If we can draw the graph with arcs that intersect only at nodes, then it is a planar graph.



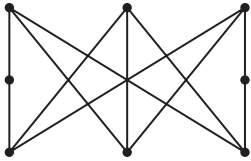
29. 5

31. The proof for Euler's formula does not depend on the graph being simple, so the result still holds for non-simple graphs, but this is not true for inequalities (2) and (3).

33. Planar



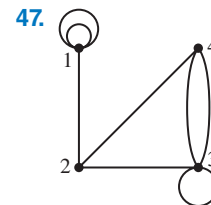
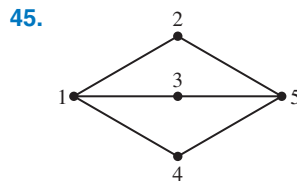
35. Nonplanar—subgraph here can be obtained from $K_{3,3}$ by elementary subdivisions



37.
$$\begin{bmatrix} 1 & 1 & 0 & 0 & 2 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 \end{bmatrix}$$

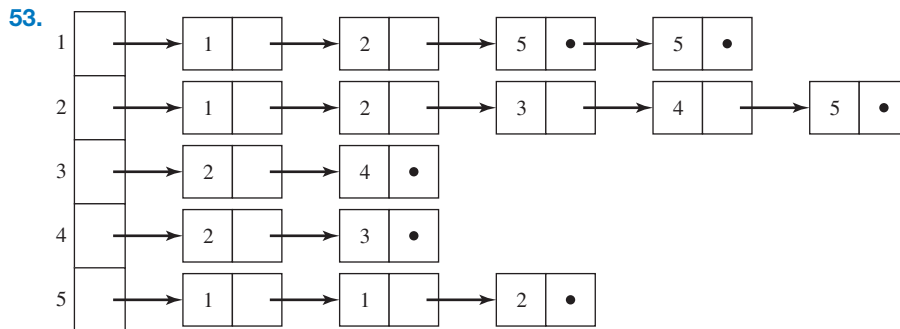
39.
$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

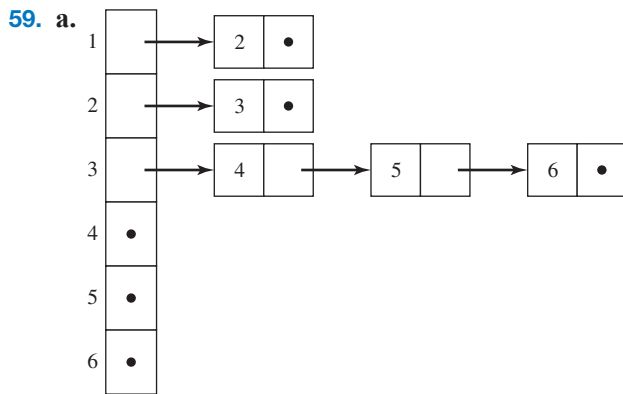
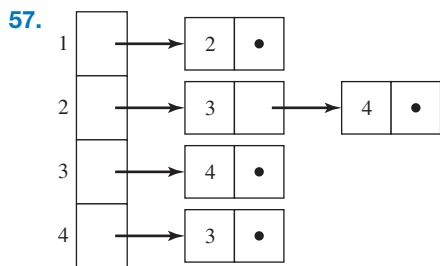
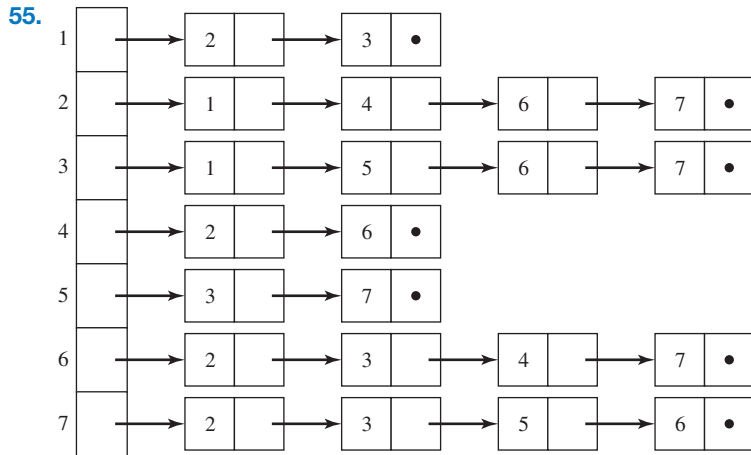
41.
$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



49. The graph consists of n disconnected nodes with a loop at each node.

51. The $n \times n$ matrix with 0s down the main diagonal and 1s elsewhere.

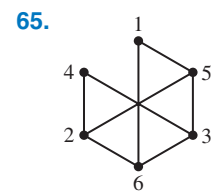
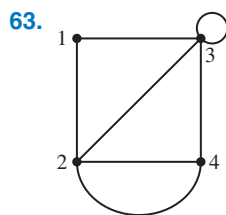




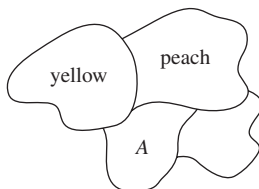
b. 16 c. 36

61.

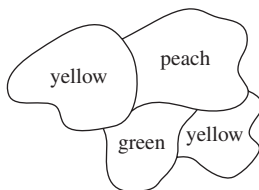
	Node	Pointer
1		5
2		7
3		11
4		0
5	2	6
6	3	0
7	1	8
8	2	9
9	3	10
10	4	0
11	4	0



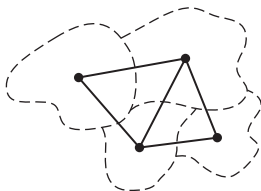
67. By the definition of isomorphic graphs, nodes $x-y$ are adjacent in G_1 if and only if their images are adjacent in G_2 . Therefore nodes are not adjacent in G_1 (and therefore are adjacent in G'_1) if and only if their images are not adjacent in G_2 (and therefore are adjacent in G'_2). Thus the same function f makes the complement graphs isomorphic.
69. If G is not connected then G consists of two or more connected subgraphs that have no paths between them. Let x and y be distinct nodes. If x and y are in different subgraphs, there is no $x-y$ arc in G ; hence there is an $x-y$ arc in G' , and a path exists from x to y in G' . If x and y are in the same subgraph, then pick a node z in a different subgraph. There is an arc $x-z$ in G' and an arc $z-y$ in G' , hence there is a path from x to y in G' .
71. The matrix for G' will have 1s where A had 0s and 0s where A had 1s except for diagonal elements, which remain 0s.
73. The maximum number of arcs occurs in a complete graph; the maximum is $C(n,2) = n(n-1)/2$, therefore $a \leq n(n-1)/2$ or $2a \leq n^2 - n$.
75. Let G be a simple graph with n nodes, $n \geq 2$, and m arcs, $m > C(n-1,2) = (n-1)(n-2)/2$, and suppose that G is not connected. By Exercise 69, G' is connected. By Exercise 74, the number of arcs in G' is at least $n-1$. Therefore the number m of arcs in G is (the number of arcs in a complete graph) - (the number of arcs in G') $= n(n-1)/2 - (\text{the number of arcs in } G') \leq n(n-1)/2 - (n-1) = (n-1)(n/2 - 1) = (n-1)(n-2)/2$ which is a contradiction.
77. At least three colors are required because of the overlapping boundaries. Once the following assignment has been made, the country marked A must be a third color:



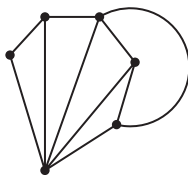
Three colors are sufficient:



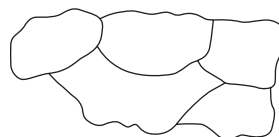
79. a.



b.

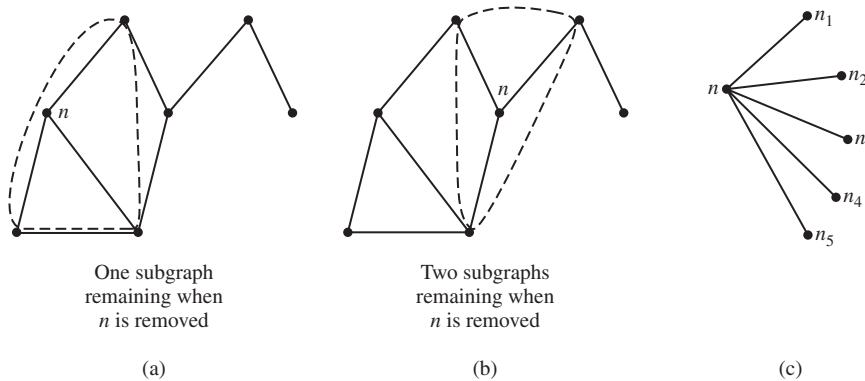


c.



81. The four-color conjecture is equivalent to the statement that the chromatic number for any simple, connected, planar graph is at most 4.
83. The proof is by mathematical induction on the number of nodes in the graph. For the basis step of the induction process, it is clear that five colors are sufficient if the number of nodes is less than or equal to 5. Now assume that any simple, connected, planar graph with $\leq k$ nodes can be colored with five colors, and consider such a graph with $k+1$ nodes. We can assume that $k+1$ is at least 6 because 5 or fewer nodes are taken care of. By Exercise 82, at least one node n of the graph has degree less than or equal

to 5; temporarily removing n (and its adjoining arcs) from the graph will leave a collection of one or more simple, connected, planar subgraphs, each with no more than k nodes (Figures a and b). By the inductive hypothesis, each subgraph has a coloring with no more than five colors (use the same palette of five colors for each subgraph). Now look at the original graph again. If n has degree less than 5 or if the 5 nodes adjacent to n do not use five different colors, there is a fifth color left to use for n . Thus, we assume that n is adjacent to 5 nodes, $n_1, n_2, n_3, n_4,$ and n_5 , arranged clockwise around n and colored, respectively, colors 1, 2, 3, 4, and 5 (Figure c).



Now pick out all the nodes in the graph colored 1 or 3. Suppose there is no path, using just these nodes, between n_1 and n_3 . Then, as far as nodes colored 1 and 3 are concerned, there are two separate sections of graph, one section containing n_1 and one containing n_3 . In the section containing n_1 interchange colors 1 and 3 on all the nodes. Doing this does not violate the (proper) coloring of the subgraphs, it colors n_1 with 3, and it leaves color 1 for n . Now suppose there is a path between n_1 and n_3 using only nodes colored 1 or 3. In this case we pick out all nodes in the original graph colored 2 or 4. Is there a path, using just these nodes, between n_2 and n_4 ? No, there is not. Because of the arrangement of nodes $n_1, n_2, n_3, n_4,$ and n_5 , such a path would have to cross the path connecting n_1 and n_3 . Because the graph is planar, these two paths would have to meet at a node, which would then be colored 1 or 3 from the n_1 - n_3 path and 2 or 4 from the n_2 - n_4 path, an impossibility. Thus, there is no path using only nodes colored 2 or 4 between n_2 and n_4 , and we can rearrange colors as in the previous case. This completes the proof.

85. four; three

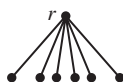
EXERCISES 6.2

1. a. Yes, it is a tree. Put the root at the top.

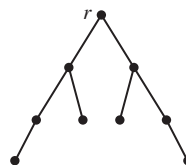


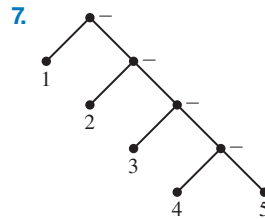
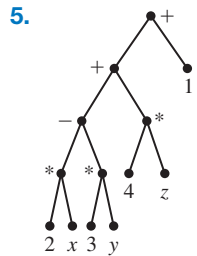
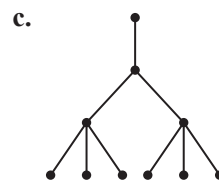
b. Not a tree because there is a cycle.

c. Yes, it is a tree. Put the root at the top and drop down all the branches.



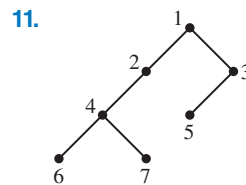
d. Yes, it is a tree. "Shake down" the lower branches.





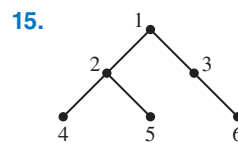
9.

	Left child	Right child
1	2	3
2	0	4
3	5	6
4	7	0
5	0	0
6	0	0
7	0	0



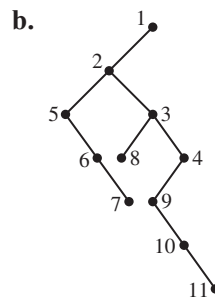
13.

	Name	Left child	Right child
1	All	0	2
2	Gaul	3	4
3	divided	0	0
4	is	5	6
5	into	0	0
6	three	7	0
7	parts	0	0

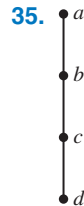
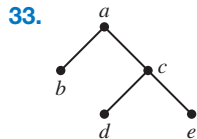


17. a.

	Left child	Right sibling
1	2	0
2	5	3
3	8	4
4	9	0
5	0	6
6	0	7
7	0	0
8	0	0
9	0	10
10	0	11
11	0	0

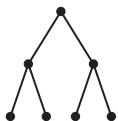


- 19. preorder: $abdehfcg$, inorder: $dbhefagc$, postorder: $dhefbgca$
- 21. preorder: $abecfjgdghi$, inorder: $ebajfcghdi$, postorder: $ebjfgchida$
- 23. preorder: $abcefdgh$, inorder: $ecfbgdha$, postorder: $efcg hdba$
- 25. prefix: $+ / 3 4 - 2 y$, postfix: $3 4 / 2 y - +$
- 27. infix: $((2 + 3) * (6 * x)) - 7$, postfix: $2 3 + 6 x * * 7 -$
- 29. prefix: $+ * 4 - 7 x z$, infix: $(4 * (7 - x)) + z$
- 31. 10

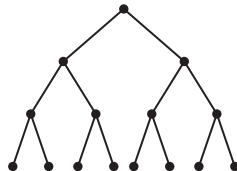


- 37. If the root has no left child and no right child, return 0 as the height, else invoke the algorithm on the left child if it exists, invoke the algorithm on the right child if it exists, return the maximum of those two values plus 1.
- 39. Consider a simple graph that is a nonrooted tree. A tree is an acyclic and connected graph, so for any two nodes x and y , a path from x to y exists. If the path is not unique, then the two paths diverge at some node n_1 and converge at some node n_2 , and there is a cycle from n_1 through n_2 and back to n_1 , which is a contradiction. Now consider a simple graph that has a unique path between any two nodes. The graph is clearly connected. Also, there are no cycles because the presence of a cycle produces a nonunique path between two nodes on the cycle. The graph is thus acyclic and connected and is a nonrooted tree.
- 41. If G is a nonrooted tree, then G is connected. Suppose we remove an arc a between n_1 and n_2 and G remains connected. Then there is a path from n_1 to n_2 . Adding a to this path results in a cycle from n_1 to n_1 , which contradicts the definition of a tree. On the other hand, suppose G is connected and removing any single arc makes G unconnected. If G is not a tree, then it contains a cycle. If a single arc is removed from the cycle, the graph is still connected because any path that made use of the removed arc can use the rest of the cycle instead. This is a contradiction, so G is a nonrooted tree.
- 43. Proof is by induction on d . For $d = 0$, the only node is the root, and $2^0 = 1$. Assume that there are at most 2^d nodes at depth d , and consider depth $d + 1$. There are at most two children for each node at depth d , so the maximum number of nodes at depth $d + 1$ is $2 \cdot 2^d = 2^{d+1}$.

45. a. 7 nodes



b. 15 nodes



c. $2^{h+1} - 1$

- 47. a. In a full binary tree, all internal nodes have two children, so the total number of “children nodes” is $2x$; the only “non-child” node is the root, so there is a total of $2x + 1$ nodes.
- b. From part a, there are $2x + 1$ total nodes, x of which are internal, leaving $2x + 1 - x = x + 1$ leaves.
- c. Consider a full binary tree with n nodes; let x be the number of internal nodes. From part a, $n = 2x + 1$. Therefore, $x = (n - 1)/2$. From part b, the number of leaves = $x + 1 = (n - 1)/2 + 1 = (n + 1)/2$.
- 49. By Exercise 45, a full binary tree of height $h - 1$ has $2^h - 1$ nodes. When $n = 2^h$, this is the beginning of level h . The height h remains the same until $n = 2^{h+1}$, when it increases by 1. Therefore for $2^h \leq n < 2^{h+1}$, the height of the tree remains the same and is given by $h = \lfloor \log n \rfloor$.

51. 2

53. a. There is only one binary tree with one node, so $B(1) = 1$. For a binary tree with n nodes, $n > 1$, the “shape” of the tree is determined by the “shape” of the left and right subtrees; the two subtrees have a total of $n - 1$ nodes. Let the left subtree have k nodes; the right subtree then has $n - 1 - k$ nodes; k can range from 0 to $n - 1$. For each value of k , there are $B(k)$ ways to form the left subtree, then $B(n - 1 - k)$ ways to form the right subtree, so by the multiplication principle, there are $B(k)B(n - 1 - k)$ different trees.

b. $B(0) = 1, B(1) = 1, B(n) = \sum_{k=0}^{n-1} B(k)B(n - 1 - k), = \sum_{k=1}^n B(k - 1)B(n - k)$

which is the same as the Catalan sequence, so by Exercise 97 of Section 4.4,

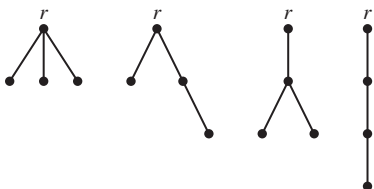
$$B(n) = \frac{1}{n + 1} C(2n, n)$$

c. $B(3) = 5$. The 5 distinct binary trees are

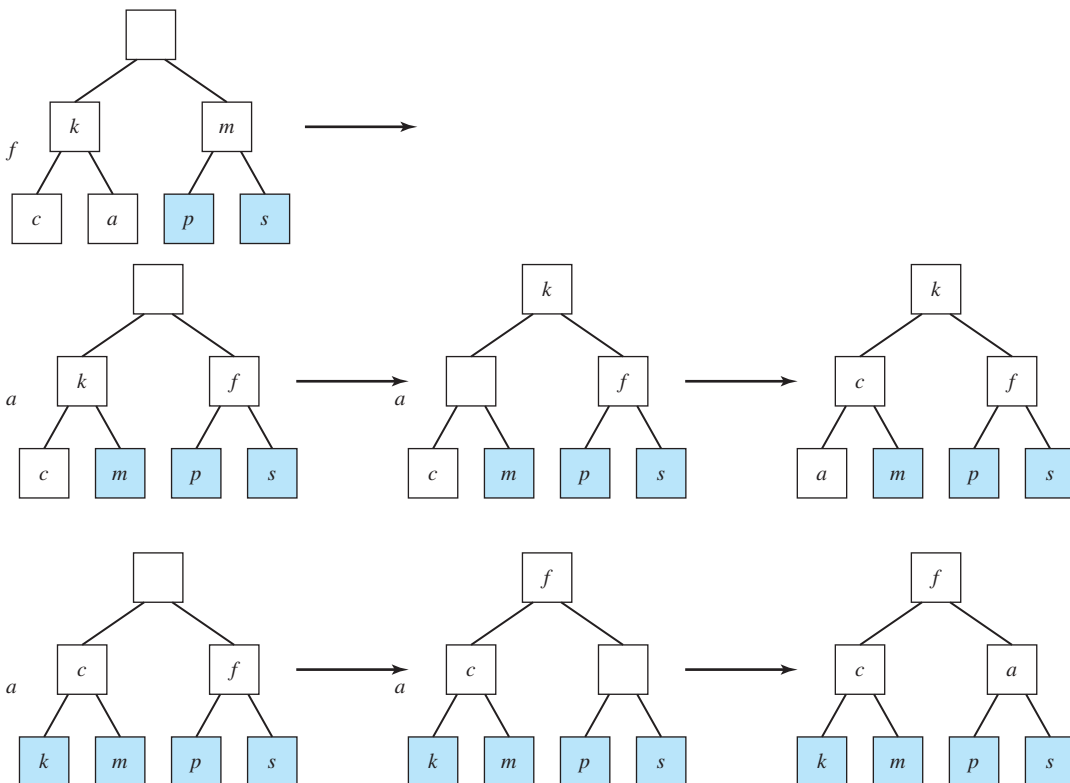


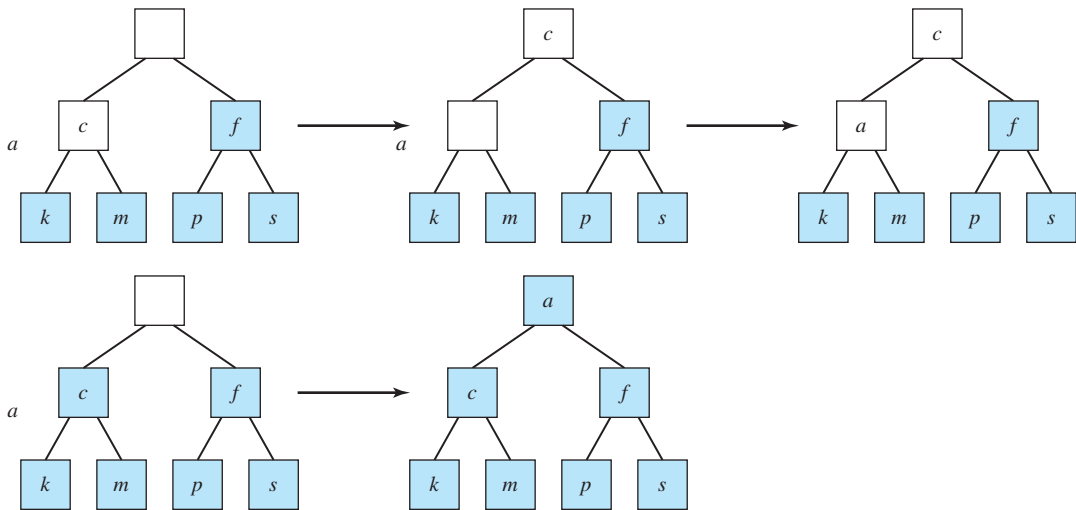
d. $B(6) = 132$

55.

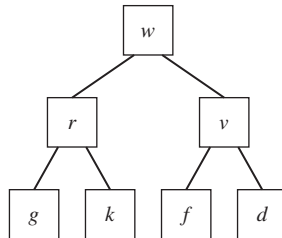


57. a.

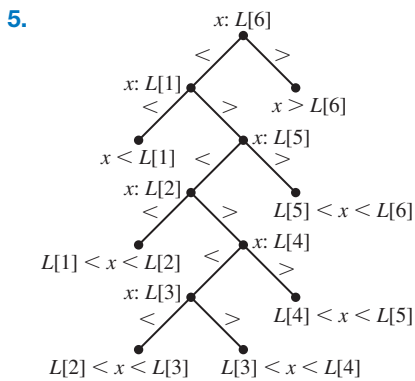
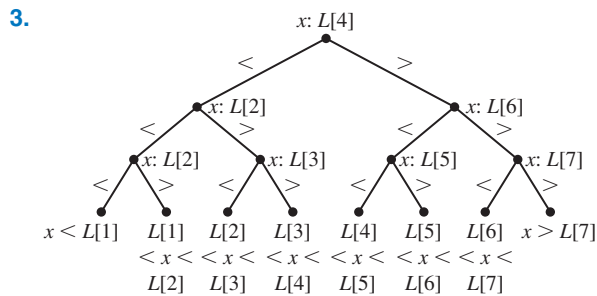
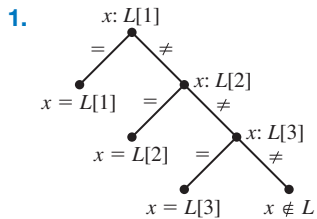




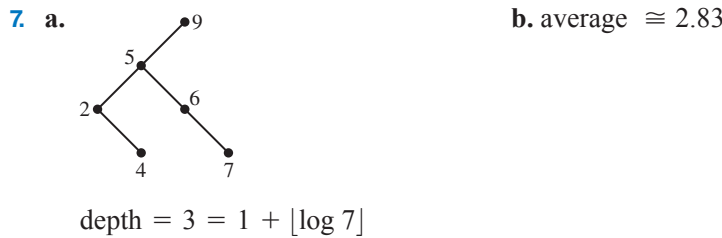
b. The array, when rearranged into a heap (but not yet sorted) would be



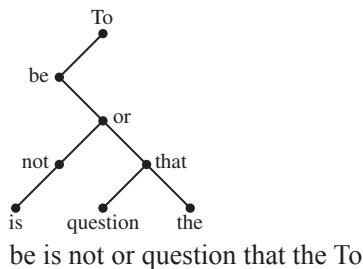
EXERCISES 6.3



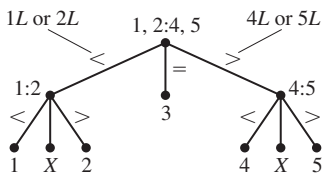
depth = 6; algorithm is not optimal because $6 > 1 + \log_2[6] = 3$



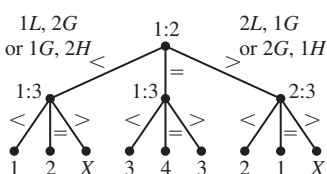
9. a. 3 b. For example: *g, d, a, k, i, s* 11.



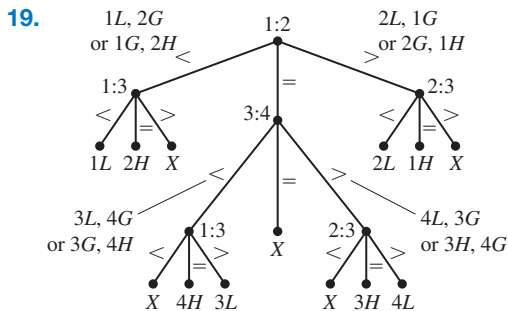
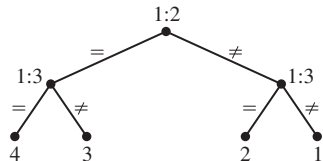
13. a. 5 b. 16 c. 45
 15. a. 5 b. 2 c. 1L or 2L 1, 2:4, 5 4L or 5L



17. a. 4 b. 2 c. 1L, 2G or 1G, 2H 1:2 2L, 1G or 2G, 1H



This problem (because we do not have to decide whether the counterfeit coin is heavy or light) can also be done with a binary tree of depth 2:



21. $2 * (1 + \lceil \log n \rceil)$.

23. a. $\log n! = \log[(n)(n-1)(n-2) \cdots (2)(1)] = \log n + \log(n-1) + \log(n-2) + \cdots + \log 2 + \log 1 \leq \log n + \log n + \log n + \cdots + \log n \text{ for } n \geq 1 = n \log n$
 b. $\log n! = \log[(n)(n-1)(n-2) \cdots (2)(1)] = \log n + \log(n-1) + \log(n-2) + \cdots + \log 2 + \log 1 \geq \log n + \log(n-1) + \cdots + \log \lceil n/2 \rceil \geq \log \lceil n/2 \rceil + \log \lceil n/2 \rceil + \cdots + \log \lceil n/2 \rceil \geq \lceil n/2 \rceil$
 $\log \lceil n/2 \rceil \geq \left(\frac{n}{2}\right) \log \left(\frac{n}{2}\right) = \left(\frac{n}{2}\right) (\log n - \log 2) = \left(\frac{n}{2}\right) (\log n - 1) = \left(\frac{n}{2}\right) \log n - \left(\frac{n}{2}\right) =$
 $\left(\frac{n}{4}\right) \log n + \left(\frac{n}{4}\right) \log n - \left(\frac{n}{2}\right) = \left(\frac{n}{4}\right) \log n + \left(\frac{n}{4}\right) (\log n - 2) \geq \left(\frac{n}{4}\right) \log n$ because $\log n \geq 2$ for $n \geq 4$

EXERCISES 6.4

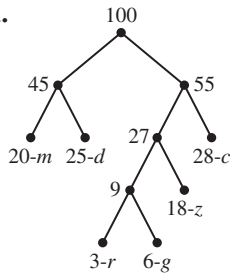
1. No, because the code for *m*, 01, is a prefix of the code for *d*, 011.

3. a. oooue b. iaou c. eee

5. a. (pw)a b. paw c. ((a))

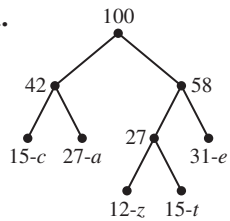
7. a-0101, b-011, c-10, d-11

9. a.

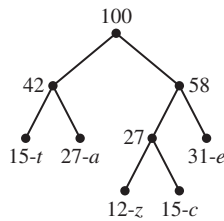


b. c-11, d-01, g-1001, m-00, r-1000, z-101

11. a.

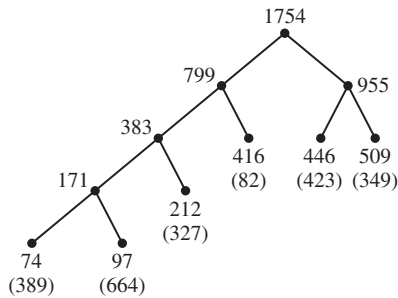


or



b. a-01 a-01
 z-100 z-100
 t-101 or t-00
 e-11 e-11
 c-00 c-101

13. a.



b. 82-01
 664-0001
 327-001
 349-11
 423-10
 389-0000

15. a. 85,000 bytes b. 34000 bytes

17. One of several possibilities: s-000, h-001, a-01, t-100, c-101, e-11

19. a. One possible Huffman code is

B-110100	0-01	5-1000
C-11101	1-101	6-010
G-1101100	2-1001	7-1100
R-110101	3-011	8-110111
S-1101101	4-1111	9-11100

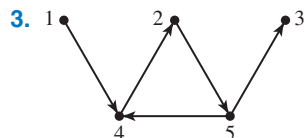
b. The new file takes about 44 percent of the space of the original file.

CHAPTER 7

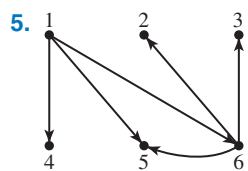
EXERCISES 7.1

1. $A = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

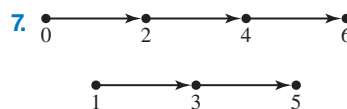
$\rho = \{(1, 1), (2, 1), (2, 3), (3, 2)\}$



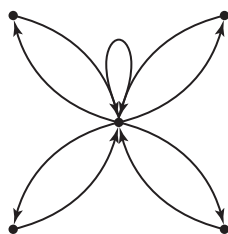
$\rho = \{(1, 4), (2, 5), (4, 2), (5, 3), (5, 4)\}$



$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

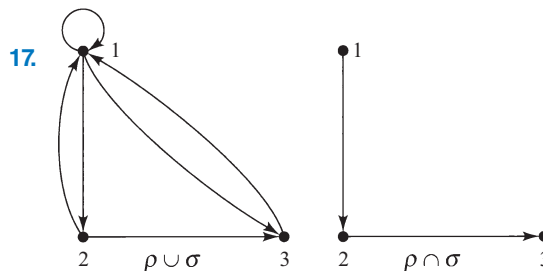


- 9. For every pair of nodes a and b , if there is an arc from a to b , then there is also an arc from b to a .
- 11. The graph can be written as a “star” with node 1 at the center; i.e., 1 is adjacent to every node and every node is adjacent to 1, but no other nodes are adjacent. For example, with $n = 5$:



- 13. No node has a loop.

15. $\rho \cup \sigma: \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$ $\rho \cap \sigma: \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$



19. $A^2 = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 0 & 0 & 1 \end{bmatrix}$ $A^{(2)} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

- 21. R will have all 1 entries.

$$23. \mathbf{R} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$25. \mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$27. \mathbf{R} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$29. \mathbf{R} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$31. \mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$33. \mathbf{R} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$35. \mathbf{R} = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

37. Transitive closure = $\{(1, 2), (1, 3), (2, 2), (2, 3), (3, 2), (3, 3)\}$

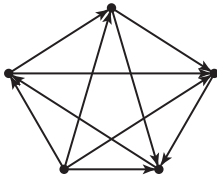
39. a. Add $(2, 1)$ and $(3, 2)$ to ρ to get the transitive closure.

b. ρ is its own transitive closure.

c. Add $(1, 1), (2, 1), (2, 2), (3, 3)$ to ρ to get the transitive closure.

d. ρ is its own transitive closure.

41.



43. $\mathbf{A}^2[i, j] = \sum_{k=1}^n a_{ik}a_{kj}$. If a term such as $a_{i2}a_{2j}$ in this sum is 0, then either $a_{i2} = 0$ or $a_{2j} = 0$ (or both) and

there is either no path of length 1 from n_i to n_2 or no path of length 1 from n_2 to n_j (or both). Thus there are no paths of length 2 from n_i to n_j passing through n_2 . If $a_{i2}a_{2j} \neq 0$, then $a_{i2} = p$ and $a_{2j} = q$, where p and q are positive integers. Then there are p paths of length 1 from n_i to n_2 and q paths of length 1 from n_2 to n_j . By the multiplication principle, there are pq possible paths of length 2 from n_i to n_j through n_2 . By the addition principle, the sum of all such terms gives all possible paths of length 2 from n_i to n_j .

$$45. 3; \mathbf{A}^2 = \begin{bmatrix} 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

EXERCISES 7.2

1. No, four nodes of degree 3
3. No odd nodes, so yes; such a path can start at any node and will end there. For example, 1-2-6-3-1-4-6-5-1
5. No, four nodes of odd degree
7. Two odd nodes, 1 and 3, so yes; such a path must begin at one odd node and end at the other. For example, 1-4-5-1-2-5-6-2-3-6-7-3
9. No, six nodes of odd degree
11. No, four nodes of odd degree

$$13. \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

$$15. \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

total after row 2 is 0

$i = 8$

17. A connected directed graph will have an Euler path if and only if (a) All nodes have out-degree equal to in-degree or (b) One node, n_p , has out-degree one greater than in-degree and another node, n_q , has in-degree one greater than out-degree.
19. Every node has equal in-degree and out-degree, so a path exists from an arbitrary node back to that node. For example, 2-4-2-3-1-2.
21. No
23. yes; for example, 1-4-2-6-3-5-1
25. yes; for example, 1-2-3-7-6-5-4-1
27. yes; for example, 6-5-8-7-3-4-1-2-6
29. Any two nodes must be part of the Hamiltonian circuit; therefore there is a path between them, namely, that part of the circuit that is between them.
31. a. $(n - 1)^n$ b. $(n - 1)(n - 2)^{n-1}$ c. $(n - 1)!$ d. 14! seconds or about 24.22 hours.
33. a. $n = 2$ or $n =$ any odd number. b. $n > 2$
35. Such a graph is a chain, so just pick a node and then walk around the chain back to the starting node.
37. a. Consider each node of G in turn. At each node add as many new arcs as possible without creating a circuit. This process terminates because the complete graph with n nodes would contain a circuit.
 b. H is not a complete graph or it would contain a Hamiltonian circuit. Therefore there must be two nodes p and q that are not adjacent in H , but adding arc $p-q$ would complete a Hamiltonian circuit. Therefore there is a Hamiltonian path beginning at node p and ending at node q .
 c. If both these arcs exist in H , then H has a Hamiltonian circuit, as follows: $x_i, p, x_2, x_3, \dots, x_{i-1}, q, x_{n-1}, x_{n-2}, \dots, x_i$
 d. Nodes p and q are not adjacent (otherwise there would be a Hamiltonian circuit), so the only nodes that can be adjacent to p or q (with at most one arc) are the $n - 2$ nodes $x_i, 2 \leq i \leq n - 1$. From part (c), for each such node x_i , if p is adjacent to x_i then q is not adjacent to x_{i-1} , so the total of the two degrees cannot exceed $n - 2$.

- e. H was constructed from G by adding additional arcs, so for any node its degree in G is \leq its degree in H . Using this fact together with the result of part (d), $\text{degree}(p) + \text{degree}(q) < n$ in graph G .
- f. Nodes p and q are not adjacent, yet $\text{degree}(p) + \text{degree}(q) < n$. This contradicts condition (2), so the assumption that G does not have a Hamiltonian circuit is wrong.

EXERCISES 7.3

1. $IN = \{2\}$

	1	2	3	4	5	6	7	8
d	3	0	2	∞	∞	∞	1	∞
s	2	-	2	2	2	2	2	2

$p = 7, IN = \{2, 7\}$

	1	2	3	4	5	6	7	8
d	3	0	2	∞	∞	6	1	2
s	2	-	2	2	2	7	2	7

$p = 3, IN = \{2, 7, 3\}$

	1	2	3	4	5	6	7	8
d	3	0	2	3	∞	6	1	2
s	2	-	2	3	2	7	2	7

3. $IN = \{1\}$

	1	2	3	4	5	6	7	8
d	0	3	5	∞	8	1	∞	∞
s	-	1	1	1	1	1	1	1

$p = 6, IN = \{1, 6\}$

	1	2	3	4	5	6	7	8
d	0	3	5	∞	7	1	6	∞
s	-	1	1	1	6	1	6	1

$p = 2, IN = \{1, 6, 2\}$

	1	2	3	4	5	6	7	8
d	0	3	5	∞	7	1	4	∞
s	-	1	1	1	6	1	2	1

$p = 7, IN = \{1, 6, 2, 7\}$

	1	2	3	4	5	6	7	8
d	0	3	5	∞	7	1	4	5
s	-	1	1	1	6	1	2	7

5. $IN = \{a\}$

	a	b	c	d	e	f
d	0	1	3	∞	∞	∞
s	-	a	a	a	a	a

$p = 8, IN = \{2, 7, 3, 8\}$

	1	2	3	4	5	6	7	8
d	3	0	2	3	3	6	1	2
s	2	-	2	3	8	7	2	7

$p = 5, IN = \{2, 7, 3, 8, 5\}$

	1	2	3	4	5	6	7	8
d	3	0	2	3	3	6	1	2
s	2	-	2	3	8	7	2	7

path: 2, 7, 8, 5 distance = 3

$p = 3, IN = \{1, 6, 2, 7, 3\}$

	1	2	3	4	5	6	7	8
d	0	3	5	6	7	1	4	5
s	-	1	1	3	6	1	2	7

$p = 8, IN = \{1, 6, 2, 7, 3, 8\}$

	1	2	3	4	5	6	7	8
d	0	3	5	6	6	1	4	5
s	-	1	1	3	8	1	2	7

$p = 5, IN = \{1, 6, 2, 7, 3, 8, 5\}$

	1	2	3	4	5	6	7	8
d	0	3	5	6	6	1	4	5
s	-	1	1	3	8	1	2	7

path: 1, 2, 7, 8, 5 distance = 6

$p = b, IN = \{a, b\}$

	a	b	c	d	e	f
d	0	1	2	∞	∞	2
s	-	a	b	a	a	b

$$p = c, IN = \{a, b, c\}$$

	a	b	c	d	e	f
d	0	1	2	4	6	2
s	-	a	b	c	c	b

$$p = f, IN = \{a, b, c, f\}$$

	a	b	c	d	e	f
d	0	1	2	4	3	2
s	-	a	b	c	f	b

$$p = e, IN = \{a, b, c, f, e\}$$

	a	b	c	d	e	f
d	0	1	2	4	3	2
s	-	a	b	c	f	b

path: a, b, f, e distance = 3

7. $IN = \{1\}$

	1	2	3	4	5	6	7
d	0	2	∞	∞	3	2	∞
s	-	1	1	1	1	1	1

$$p = 2, IN = \{1, 2\}$$

	1	2	3	4	5	6	7
d	0	2	3	∞	3	2	∞
s	-	1	2	1	1	1	1

$$p = 6, IN = \{1, 2, 6\}$$

	1	2	3	4	5	6	7
d	0	2	3	∞	3	2	5
s	-	1	2	1	1	1	6

$$p = 3, IN = \{1, 2, 6, 3\}$$

	1	2	3	4	5	6	7
d	0	2	3	4	3	2	5
s	-	1	2	3	1	1	6

$$p = 5, IN = \{1, 2, 6, 3, 5\}$$

	1	2	3	4	5	6	7
d	0	2	3	4	3	2	5
s	-	1	2	3	1	1	6

$$p = 4, IN = \{1, 2, 6, 3, 5, 4\}$$

	1	2	3	4	5	6	7
d	0	2	3	4	3	2	5
s	-	1	2	3	1	1	6

$$p = 7, IN = \{1, 2, 6, 3, 5, 4, 7\}$$

	1	2	3	4	5	6	7
d	0	2	3	4	3	2	5
s	-	1	2	3	1	1	6

path: 1,6,7 distance = 5

9. a. Change the condition on the while loop to continue until all nodes are in IN. Also, rather than writing out a particular shortest path, make d and s output parameters that carry the information about shortest paths and their distances.

b. No

11.

	1	2	3	4	5	6	7	8
d	3	0	2	∞	∞	∞	1	∞
s	2	-	2	2	2	2	2	2

(1)

	1	2	3	4	5	6	7	8
d	3	0	2	3	11	4	1	2
s	2	-	2	3	1	1	2	7

(2)

	1	2	3	4	5	6	7	8
d	3	0	2	3	3	4	1	2
s	2	-	2	3	8	1	2	7

(3)

No further changes in d or s . Agrees with Exercise 1 for 2 to 5

13.

	1	2	3	4	5	6	7
<i>d</i>	0	2	∞	∞	3	2	∞
<i>s</i>	-	1	1	1	1	1	1

(1)

	1	2	3	4	5	6	7
<i>d</i>	0	2	3	4	3	2	5
<i>s</i>	-	1	2	3	1	1	6

(3)

	1	2	3	4	5	6	7
<i>d</i>	0	2	3	∞	3	2	5
<i>s</i>	-	1	2	1	1	1	6

(2)

No further changes in *d* or *s*.
Agrees with Exercise 7 for 1 to 7

15. Initial **A** and after $k = x$:

	<i>x</i>	1	2	3	<i>y</i>
<i>x</i>	0	1	∞	4	∞
1	1	0	3	1	5
2	∞	3	0	2	2
3	4	1	2	0	3
<i>y</i>	∞	5	2	3	0

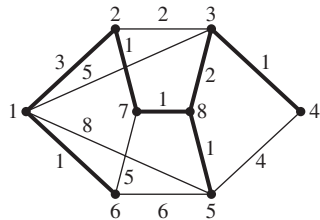
after $k = 1$ and $k = 2$:

	<i>x</i>	1	2	3	<i>y</i>
<i>x</i>	0	1	4	2	6
1	1	0	3	1	5
2	4	3	0	2	2
3	2	1	2	0	3
<i>y</i>	6	5	2	3	0

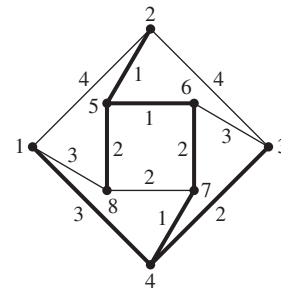
after $k = 3$ and $k = y$:

	<i>x</i>	1	2	3	<i>y</i>
<i>x</i>	0	1	4	2	5
1	1	0	3	1	4
2	4	3	0	2	2
3	2	1	2	0	3
<i>y</i>	5	4	2	3	0

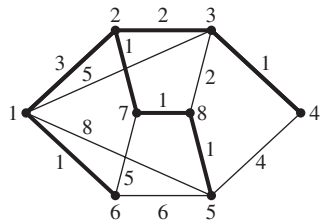
17. $IN = \{1, 6, 2, 7, 8, 5, 3, 4\}$



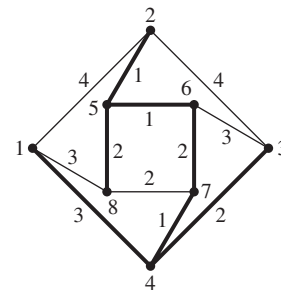
19. $IN = \{1, 4, 7, 3, 6, 5, 2, 8\}$

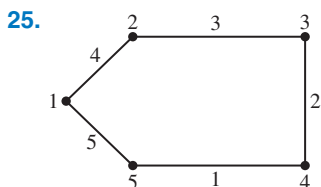


21. For example,

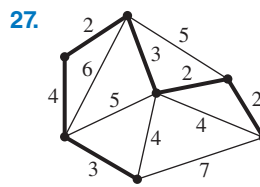


23. For example,





The shortest path from 1 to 5 is 1-5 with distance 5. If the algorithm added the node closest to IN at each step, it would choose the path 1-2-3-4-5 with distance 10.

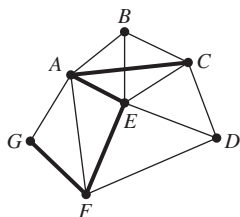


The solution is to find a minimal spanning tree for the graph, as shown here.

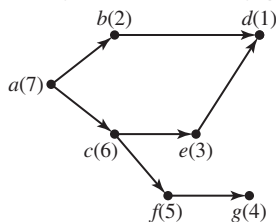
29. Dijkstra's algorithm is $\Theta(n^2)$ in the worst case, which is when all nodes are brought into IN. This is the situation to find the distance from the start node to any other node. Repeating this process with all n nodes, in turn, as the start node would result in an algorithm of order $n\Theta(n^2) = \Theta(n^3)$. Floyd's algorithm is clearly $\Theta(n^3)$ because of the nested **for** loops. Therefore the algorithms are the same order of magnitude. Although Floyd's algorithm has the advantage of simplicity to code, that is more than balanced out by the fact that Floyd's algorithm does not give the actual shortest paths.

EXERCISES 7.4

- 1. *abc ef dh gji*
- 3. *d abc ef h gji*
- 5. *e bac f dh gji*
- 7. *abc fj g de h ki*
- 9. *fc ab de h ki gj*
- 11. *abc de g f h ji*
- 13. *d af b ce h g ij*
- 15. *e b c f g a d h ji*
- 17. *abc de f g h i j k*
- 19. *fc j a b g de h i k*
- 21. *abc e g d f h*
- 23. *fb*
- 25. *abc de g h f*
- 27. *fb*
- 29.



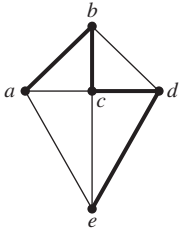
31. Begin a *dfs* at node a: *a c f g e b d*



- 33. Because of the recursion, a stack would be a good data structure.
- 35. Suppose the depth-first search has visited node x and has moved on to visit node y . The algorithm looks for nodes adjacent to y that are unvisited. If a visited node (other than y 's "parent" x) is on y 's adjacency list, then the graph contains a cycle. For example, in Figure 7.13, when the recursive algorithm is invoked on node g , node d is a previously visited node that is adjacent to g but not the parent node of g (which is f). This situation detects the cycle $d-g-f-d$.

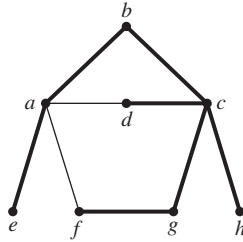
EXERCISES 7.5

1.



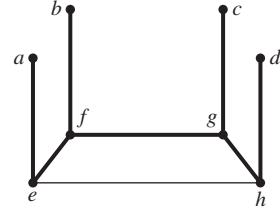
back arcs: $a-c, a-e, b-d, c-e$

3.



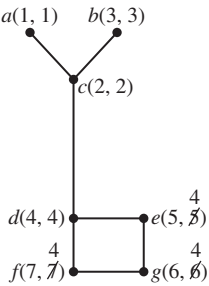
back arcs: $a-d, a-f$

5.



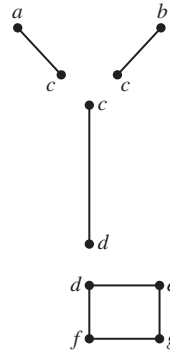
back arcs: $e-h$

7.

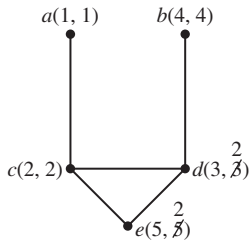


articulation points: c, d

biconnected components

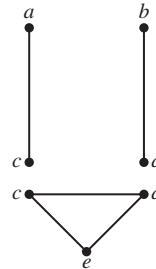


9.

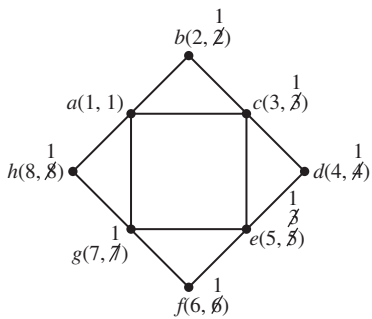


articulation points: c, d

biconnected components

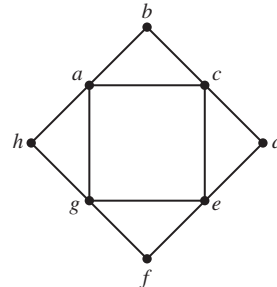


11.



articulation points: none

biconnected components



CHAPTER 8

EXERCISES 8.1

1.	+	0	1	a	a'
	0	0	1	a	a'
	1	1	1	1	1
	a	a	1	a	1
	a'	a'	1	1	a'

	·	0	1	a	a'
	0	0	0	0	0
	1	0	1	a	a'
	a	0	a	a	0
	a'	0	a'	0	a'

3. a. $\max(x, y) = \max(y, x)$, $\min(x, y) = \min(y, x)$, $\max(\max(x, y), z) = \max(x, \max(y, z)) = \max(x, y, z)$, $\min(\min(x, y), z) = \min(x, \min(y, z)) = \min(x, y, z)$, $\max(x, \min(y, z)) = \min(\max(x, y), \max(x, z))$, $\min(x, \max(y, z)) = \max(\min(x, y), \min(x, z))$. The last two can be shown by taking cases: $x < y < z$, $y < x < z$, etc.

- b. Let m be the 0 element. Then we must have $\max(x, m) = x$ for all $x \in Z$. But for $x = m - 1$, $\max(m - 1, m) = m$.

5. a. 16

$$\begin{array}{lll} \text{b. } (f_1 + f_2)(0, 0) = 1 & (f_1 \cdot f_2)(0, 0) = 1 & f_1'(0, 0) = 0 \\ (f_1 + f_2)(0, 1) = 1 & (f_1 \cdot f_2)(0, 1) = 0 & f_1'(0, 1) = 1 \\ (f_1 + f_2)(1, 0) = 1 & (f_1 \cdot f_2)(1, 0) = 0 & f_1'(1, 0) = 0 \\ (f_1 + f_2)(1, 1) = 0 & (f_1 \cdot f_2)(1, 1) = 0 & f_1'(1, 1) = 1 \end{array}$$

- c. $+$ and \cdot are binary operations on B , $'$ is a unary operation on B . Max and min are commutative and associative operations; the distributive laws follow by considering different cases for the values of $f_1(x, y)$, $f_2(x, y)$, and $f_3(x, y)$ for a fixed $(x, y) \in S^2$. For any f in B and (x, y) in S^2 , $(f + 0)(x, y) = \max(f(x, y), 0(x, y)) = \max(f(x, y), 0) = f(x, y)$, and $(f \cdot 1)(x, y) = \min(f(x, y), 1(x, y)) = \min(f(x, y), 1) = f(x, y)$. Also $(f + f')(x, y) = \max(f(x, y), f'(x, y)) = 1$ and $(f \cdot f')(x, y) = \min(f(x, y), f'(x, y)) = 0$ because one value of the pair $(f(x, y), f'(x, y))$ is 1 and the other is 0.

7. Show that x acts like the complement of x' , that is, that it satisfies properties 5a and 5b with respect to x' .

$$\begin{aligned} x' + x &= x + x' & (1a) \\ &= 1 & (5a) \end{aligned}$$

and

$$\begin{aligned} x' \cdot x &= x \cdot x' & (1b) \\ &= 0 & (5b) \end{aligned}$$

Therefore $x = (x)'$ by the theorem on the uniqueness of complements.

9. a. $x + (x \cdot y)$

$$= x \cdot 1 + x \cdot y \quad (4b)$$

$$= x(1 + y) \quad (3b)$$

$$= x(y + 1) \quad (1a)$$

$$= x \cdot 1 \quad (\text{universal bound})$$

$$= x \quad (4b)$$

$$x \cdot (x + y) = x \text{ by duality}$$

- b. $x \cdot [y + (x \cdot z)]$

$$= x \cdot y + x \cdot (x \cdot z) \quad (3b)$$

$$= x \cdot y + (x \cdot x) \cdot z \quad (2b)$$

$$= x \cdot y + x \cdot z \quad (\text{dual of idempotent})$$

$$x + [y \cdot (x + z)] = (x + y) \cdot (x + z) \text{ by duality}$$

- c. $(x + y) \cdot (x' + y)$
 $= (y + x) \cdot (y + x')$ (1a)
 $= y + (x \cdot x')$ (3a)
 $= y + 0$ (5b)
 $= y$ (4a)
- $(x \cdot y) + (x' \cdot y) = y$ by duality
- d. $(x + (y \cdot z))'$
 $= x' \cdot (y \cdot z)'$ (De Morgan's laws)
 $= x' \cdot (y' + z')$ (De Morgan's laws)
 $= x' \cdot y' + x' \cdot z'$ (3b)
- $(x \cdot (y + z))' = (x' + y') \cdot (x' + z')$ by duality
- e. $(x + y) \cdot (x + 1)$
 $= (x + y) \cdot x + (x + y) \cdot 1$ (3b)
 $= x \cdot (x + y) + (x + y) \cdot 1$ (1b)
 $= (x \cdot x) + (x \cdot y) + (x + y) \cdot 1$ (3b)
 $= x + (x \cdot y) + (x + y) \cdot 1$ (dual of idempotent)
 $= x + (x \cdot y) + (x + y)$ (4b)
 $= (x \cdot y) + x + (x + y)$ (1a)
 $= (x \cdot y) + (x + x) + y$ (2a)
 $= (x \cdot y) + x + y$ (idempotent)
 $= x + (x \cdot y) + y$ (1a)
- $(x \cdot y) + (x \cdot 0) = x \cdot (x + y) \cdot y$ by duality
11. a. $x + (x' \cdot y + x \cdot y)'$
 $= x + (y \cdot x' + y \cdot x)'$ (1b)
 $= x + (y \cdot (x' + x))'$ (3b)
 $= x + (y \cdot (x + x'))'$ (1a)
 $= x + (y \cdot 1)'$ (5a)
 $= x + y'$ (4b)
- b. $((x \cdot y) \cdot z) + (y \cdot z)$
 $= (x \cdot (y \cdot z)) + (y \cdot z)$ (2b)
 $= ((y \cdot z) \cdot x) + y \cdot z$ (1b)
 $= ((y \cdot z) \cdot x) + (y \cdot z) \cdot 1$ (4b)
 $= (y \cdot z) \cdot (x + 1)$ (3b)
 $= (y \cdot z) \cdot 1$ (universal bound)
 $= y \cdot z$ (4b)
- c. $x \cdot y + y \cdot x'$
 $= y \cdot x + y \cdot x'$ (1b)
 $= y \cdot (x + x')$ (3b)
 $= y \cdot 1$ (5a)
 $= y \cdot (x + 1)$ (universal bound)
 $= y \cdot x + y \cdot 1$ (3b)
 $= y \cdot x + y$ (4b)
 $= x \cdot y + y$ (1b)
- d. $(x + y)' \cdot z + x' \cdot z \cdot y$
 $= x' \cdot y' \cdot z + x' \cdot z \cdot y$ (De Morgan's laws)
 $= x' \cdot z \cdot y' + x' \cdot z \cdot y$ (1b)
 $= x' \cdot z \cdot (y' + y)$ (3b)
 $= x' \cdot z \cdot (y + y')$ (1a)
 $= x' \cdot z \cdot 1$ (5a)
 $= x' \cdot z$ (4b)

$$\begin{aligned}
\text{e. } (x \cdot y') + (y \cdot z') + (x' \cdot z) &= (x \cdot y') \cdot 1 + (y \cdot z') \cdot 1 + (x' \cdot z) \cdot 1 && (4b) \\
&= (x \cdot y') \cdot (z + z') + (y \cdot z') \cdot (x + x') + (x' \cdot z) \cdot (y + y') && (5a) \\
&= x \cdot y' \cdot z + x \cdot y' \cdot z' + y \cdot z' \cdot x + y \cdot z' \cdot x' + x' \cdot z \cdot y + x' \cdot z \cdot y' && (3b) \\
&= x \cdot y' \cdot z + x' \cdot z \cdot y' + y \cdot z' \cdot x + x \cdot y' \cdot z' + x' \cdot z \cdot y + y \cdot z' \cdot x' && (1a) \\
&= y' \cdot z \cdot x + y' \cdot z \cdot x' + x \cdot z' \cdot y + x \cdot z' \cdot y' + x' \cdot y \cdot z + x' \cdot y \cdot z' && (1b) \\
&= (y' \cdot z) \cdot (x + x') + (x \cdot z') \cdot (y + y') + (x' \cdot y) \cdot (z + z') && (3b) \\
&= (y' \cdot z) \cdot 1 + (x \cdot z') \cdot 1 + (x' \cdot y) \cdot 1 && (5a) \\
&= (y' \cdot z) + (x \cdot z') + (x' \cdot y) && (4b) \\
&= (x' \cdot y) + (y' \cdot z) + (x \cdot z') && (1a)
\end{aligned}$$

13. This is an “if and only if” problem, so there are two things to prove.

a. Let $x = 0$. Then

$$\begin{aligned}
x \cdot y' + x' \cdot y &= 0 \cdot y' + x' \cdot y && (x = 0) \\
&= y' \cdot 0 + x' \cdot y && (1b) \\
&= 0 + x' \cdot y && (\text{dual of universal bound}) \\
&= x' \cdot y + 0 && (1a) \\
&= x' \cdot y && (4a) \\
&= 1 \cdot y && (\text{Practice 4}) \\
&= y \cdot 1 && (1b) \\
&= y && (4b)
\end{aligned}$$

b. Let $x \cdot y' + x' \cdot y = y$. Then

$$\begin{aligned}
x \cdot x' + x' \cdot x &= x && (\text{letting } y \text{ in the hypothesis have the value } x) \\
x \cdot x' + x \cdot x' &= x && (1b) \\
0 + 0 &= x && (5b) \\
0 &= x && (4a)
\end{aligned}$$

15. a. $x \oplus y$

$$\begin{aligned}
&= x \cdot y' + y \cdot x' && (\text{definition of } \oplus) \\
&= y \cdot x' + x \cdot y' && (1a) \\
&= y \oplus x && (\text{definition of } \oplus)
\end{aligned}$$

b. $x \oplus x$

$$\begin{aligned}
&= x \cdot x' + x \cdot x' && (\text{definition of } \oplus) \\
&= 0 + 0 && (5b) \\
&= 0 && (4a)
\end{aligned}$$

c. $0 \oplus x$

$$\begin{aligned}
&= 0 \cdot x' + x \cdot 0' && (\text{definition of } \oplus) \\
&= x' \cdot 0 + x \cdot 0' && (1b) \\
&= 0 + x \cdot 0' && (\text{dual of universal bound}) \\
&= 0 + x \cdot 1 && (\text{Practice 4}) \\
&= 0 + x && (4b) \\
&= x + 0 && (1a) \\
&= x && (4a)
\end{aligned}$$

d. $1 \oplus x$

$$\begin{aligned}
&= 1 \cdot x' + x \cdot 1' && (\text{definition of } \oplus) \\
&= x' \cdot 1 + x \cdot 1' && (1b) \\
&= x' + x \cdot 1' && (4b) \\
&= x' + x \cdot 0 && (\text{Practice 4}) \\
&= x' + 0 && (\text{dual of universal bound}) \\
&= x' && (4a)
\end{aligned}$$

17. Suppose $x + 0_1 = x$ for all $x \in B$. Then $0 + 0_1 = 0$ and $0_1 + 0 = 0_1$ so $0_1 = 0_1 + 0 = 0 + 0_1 = 0$ and $0_1 = 0$. Then $1 = 0'$, so 1 is unique by the theorem on uniqueness of complements.



21. a. (i) bijection (ii) for $x, y \in S, f(x \cdot y) = f(x) + f(y)$
 b. Let $f(0) = 5, f(1) = 7$. Then $f(0 \cdot 0) = f(1) = 7 = 5 + 5 = f(0) + f(0), f(0 \cdot 1) = f(0) = 5 = 5 + 7 = f(0) + f(1), f(1 \cdot 0) = f(0) = 5 = 7 + 5 = f(1) + f(0), f(1 \cdot 1) = f(1) = 7 = 7 + 7 = f(1) + f(1)$

23. a. $f: \mathbb{R} \rightarrow \mathbb{R}^+$. f is onto: given $y \in \mathbb{R}^+$, let $x = \log y$; then $x \in \mathbb{R}$ and $f(x) = 2^x = 2^{\log y} = y$. f is one-to-one: if $f(x) = f(w)$ then $2^x = 2^w$ and (taking the log of both sides) $x = w$.

b. for $x, y \in \mathbb{R}, g(x + y) = g(x) \cdot g(y)$

c. f is a bijection from \mathbb{R} to \mathbb{R}^+ and for $x, y \in \mathbb{R}, f(x + y) = 2^{x+y} = 2^x \cdot 2^y = f(x) \cdot f(y)$

d. $f^{-1}(y) = \log y$

e. f^{-1} is a bijection from \mathbb{R}^+ to \mathbb{R} and for any

$$x, y \in \mathbb{R}^+, f^{-1}(x \cdot y) = \log(x \cdot y) = \log x + \log y = f^{-1}(x) + f^{-1}(y)$$

25. $f_1 \rightarrow \{1, 3\}, f_2 \rightarrow \{1, 2\}, f_3 \rightarrow \{1, 2, 3\}, f_5 \rightarrow \{1, 2, 4\}, f_6 \rightarrow \{1, 3, 4\}, f_7 \rightarrow \{2, 3, 4\}, f_8 \rightarrow \{2, 3\}, f_9 \rightarrow \{2, 4\}, f_{10} \rightarrow \{1, 4\}, f_{11} \rightarrow \{3, 4\}$

27. a. For any $y \in B, y = f(x)$ for some $x \in B$. Then $y \& f(0) = f(x) \& f(0) = f(x + 0) = f(x) = y$, and $f(0) = \phi$ because the zero element in any Boolean algebra is unique (see Exercise 17).

b. $f(1) = f(0') = [f(0)]'' = \phi'' = \top$

29. a. i. If $x \leq y$, then $x \leq y$ and $x \leq x$, so x is a lower bound of x and y . If $w^* \leq x$ and $w^* \leq y$, then $w^* \leq x$, so x is a greatest lower bound and $x = x \cdot y$. If $x = x \cdot y$, then x is a greatest lower bound of x and y , so $x \leq y$.
 ii. Similar to i.

b. i. Let $x + y = z$. Then z is a least upper bound of x and y , which is a least upper bound of y and x , so $z = y + x$. ii. Similar to i. iii. Let $(x + y) + z = p$ and $x + (y + z) = q$. Then $y \leq x + y \leq p$ and $z \leq p$ so p is an upper bound for y and z ; because $y + z$ is the least upper bound for y and $z, y + z \leq p$. Also $x \leq x + y \leq p$. Therefore p is an upper bound for x and $y + z$, and $q \leq p$ because q is the least upper bound for x and $y + z$. Similarly $p \leq q$, so $p = q$. iv. Similar to iii.

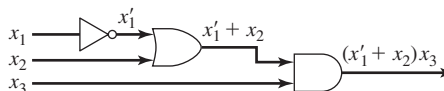
c. $x + 0 = x \leftrightarrow 0 \leq x$, which is true because 0 is a least element. $x \cdot 1 = x \leftrightarrow x \leq 1$, which is true because 1 is a greatest element.

d. (a) no—no least element, (b) yes, (c) yes, (d) no—not distributive: $2 + (3 \cdot 4) = 2 + 1 = 2$ and $(2 + 3) \cdot (2 + 4) = 5 \cdot 5 = 5$. Also, both 3 and 4 are complements of 2, so complements are not unique.

EXERCISES 8.2

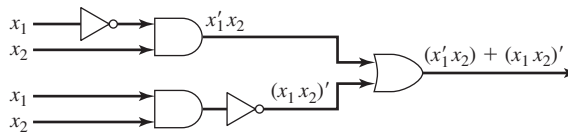
1.

x_1	x_2	x_3	$(x'_1 + x_2)x_3$
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0



3.

x_1	x_2	$x_1'x_2 + (x_1x_2)'$
1	1	0
1	0	1
0	1	1
0	0	1



5. $x_1x_2 + x_2'$

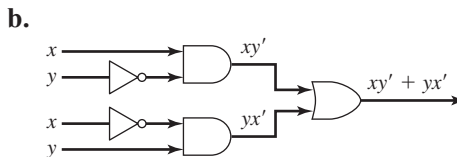
x_1	x_2	$f(x_1, x_2)$
1	1	1
1	0	1
0	1	0
0	0	1

7. $(x_1x_2)'(x_2 + x_3')$

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	1
0	0	1	0
0	0	0	1

9. a.

x	y	$f(x, y)$
1	1	0
1	0	1
0	1	1
0	0	0



c. The truth function for the network is the same as part (a). The network illustrates “ x OR y ” and “NOT both x AND y ”

11. $x_1'x_2'$

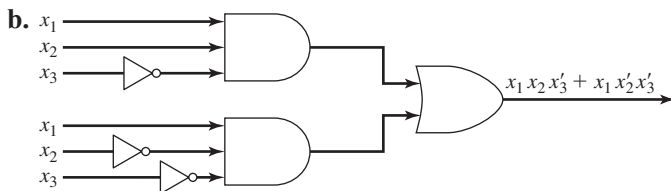
13. $x_1x_2x_3 + x_1'x_2x_3$

15. $x_1x_2'x_3 + x_1x_2x_3' + x_1'x_2x_3'$

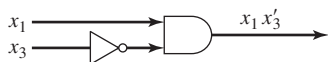
17. $x_1x_2x_3x_4 + x_1x_2x_3'x_4 + x_1x_2'x_3x_4 + x_1x_2'x_3'x_4 + x_1'x_2x_3x_4 + x_1'x_2'x_3x_4$

19. $x_1x_2'x_3x_4' + x_1'x_2x_3x_4 + x_1'x_2'x_3x_4 + x_1'x_2x_3'x_4 + x_1'x_2'x_3'x_4 + x_1'x_2'x_3x_4'$

21. a. $x_1x_2x_3' + x_1x_2'x_3'$



c. $x_1x_2x_3' + x_1x_2'x_3' = x_1x_3'(x_2 + x_2') = x_1x_3'(x_2 + x_2') = x_1x_3' \cdot 1 = x_1x_3'$



23. a.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
1	1	1	1
1	1	0	0
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	0
0	0	0	0

b. $x_1x_2x_3 + x_1x_2'x_3 + x_1'x_2x_3 + x_1'x_2'x_3$

c. $x_1x_3 + x_1'x_2 = (x_1x_3 + x_1')(x_1x_3 + x_2) = (x_1' + x_1x_3)(x_2 + x_1x_3) = (x_1' + x_1)(x_1' + x_3)(x_2 + x_1)(x_2 + x_3) = (x_1 + x_1')(x_1' + x_3)(x_1 + x_2)(x_2 + x_3) = (x_1' + x_3)(x_1 + x_2)(x_2 + x_3) = (x_1 + x_2)(x_1' + x_3)(x_2 + x_3)$

25. a. $(x_1' + x_2')(x_1' + x_2)(x_1 + x_2')$

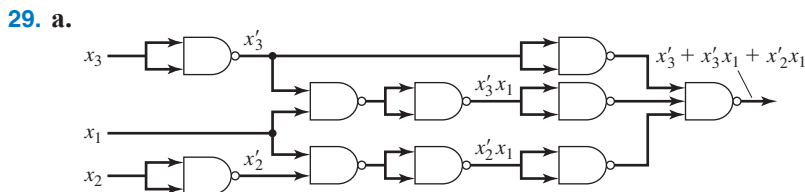
b. $(x_1' + x_2)(x_1 + x_2)$

c. $(x_1' + x_2' + x_3)(x_1' + x_2 + x_3')(x_1' + x_2 + x_3)(x_1 + x_2' + x_3)(x_1 + x_2 + x_3')(x_1 + x_2 + x_3)$

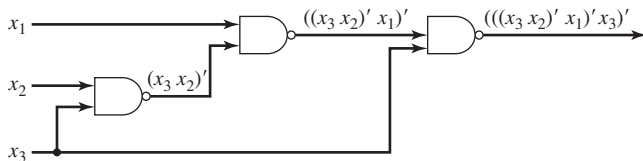
d. $(x_1' + x_2' + x_3')(x_1' + x_2 + x_3)(x_1 + x_2' + x_3)(x_1 + x_2 + x_3')$

e. $(x_1' + x_2' + x_3')(x_1' + x_2' + x_3)(x_1 + x_2' + x_3')(x_1 + x_2 + x_3')(x_1 + x_2 + x_3)$

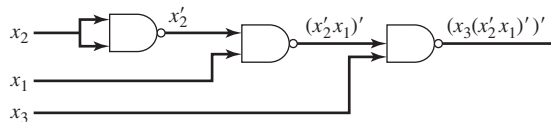
27. a. $\begin{array}{r} 1100 \\ 0100 \\ (1)0000 \end{array}$ **b.** $\begin{array}{r} 1001 \\ 0111 \\ (1)0000 \end{array}$ **c.** $\begin{array}{r} 001 \\ 111 \\ (1)000 \end{array}$



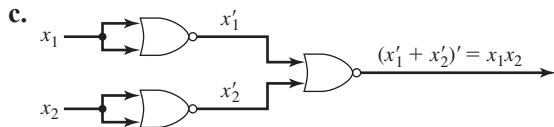
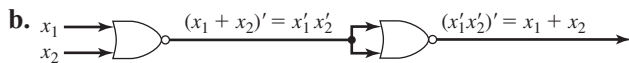
b. $x_3'x_1 + x_2'x_1 + x_3' = x_1x_3' + x_1x_2' + x_3' = x_1(x_3' + x_2') + x_3' = (x_3x_2)'x_1 + x_3' = (((x_3x_2)'x_1)')x_3'$



or, alternatively, $x_3'x_1 + x_2'x_1 + x_3' = x_3'x_1 + x_3' + x_2'x_1 = x_3'(x_1 + 1) + x_2'x_1 = x_3' \cdot 1 + x_2'x_1 = x_3' + x_2'x_1 = (x_3(x_2'x_1)')'$



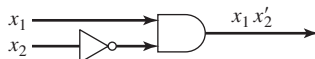
31. a. $(x_1 + x_1)' = x_1'x_1' = x_1'$



33.

X_1	X_2	$f(x_1, x_2)$
1	1	0
1	0	1
0	1	0
0	0	0

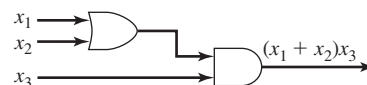
x_1x_2'



35. $x_1 = \text{neutral}, x_2 = \text{park}, x_3 = \text{seatbelt}$

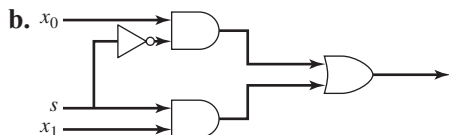
X_1	X_2	X_3	$f(x_1, x_2, x_3)$
1	1	1	-
1	1	0	-
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

$(x_1 + x_2)x_3$



37. a. Let the two input lines be x_0 and x_1 , and the selector line be s . The truth function is

X_0	X_1	s	$f(x_0, x_1, s)$
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0



39. $x_1 = \text{pressure}$ (1 when pressure > 50 psi, otherwise 0)
 $x_2 = \text{salinity}$ (1 when salinity > 45 g/L, otherwise 0)
 $x_3 = \text{temperature}$ (1 when temperature > 53°C, otherwise 0)
 $x_4 = \text{acidity}$ (1 when acidity < 7.0 pH, otherwise 0)

The output for each valve should be 1 when the valve is to open, 0 otherwise. The canonical sum-of-products forms are

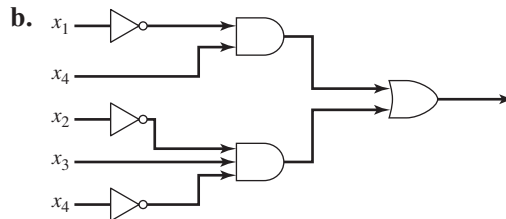
$$A = x_1x_2x_3x_4 + x_1x_2x_3x'_4 + x_1x_2x'_3x_4 + x_1x_2x'_3x'_4$$

$$B = x_1x'_2x_3x_4 + x'_1x_2x_3x_4 + x'_1x'_2x_3x_4$$

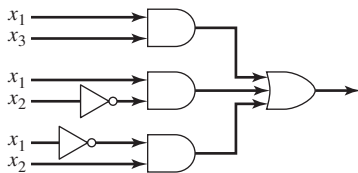
Using these forms, the circuit for A would require 2 inverters (one for x_3 and one for x_4 , assuming we split the output from an inverter into more than one gate), 4 AND gates, and 1 OR gate; B would require 2 inverters, 3 AND gates, and 1 OR gate. It is possible to write simpler equivalent expressions.

EXERCISES 8.3

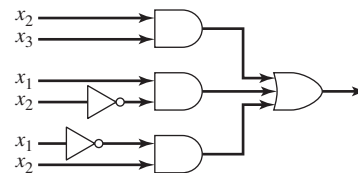
- 1. $x'_1x_3 + x_1x'_3 + x'_1x_2$ or $x'_1x_3 + x_1x'_3 + x_2x'_3$
- 3. $x_3 + x_2$
- 5. $x_1x'_3x'_4 + x'_1x_3x'_4 + x'_2x'_4 + x_1x'_2$
- 7. $x_1x'_2x_4 + x'_1x_3x'_4 + x'_2x_3x'_4$ or $x_1x'_2x_4 + x'_1x_3x'_4 + x_1x'_2x_3$
- 9. $x_1x_2 + x_2x_3$
- 11. $x_1x_4 + x'_1x'_2x_3$
- 13. a. $x'_1x_4 + x'_2x_3x'_4$



- 15. $x_2x'_3x_4 + x'_2x_3x'_4 + x'_1x_4$
- 17. $x_1x_3 + x_1x'_2 + x'_1x_2$ or $x_2x_3 + x_1x'_2 + x'_1x_2$



or



- 19. $x_2x'_4 + x'_1x'_2x_4$ Here the don't care at $x_1x_2x_3x'_4$ has been treated as a 1, as has the don't care at $x'_1x'_2x_3x_4$; the don't care at $x_1x'_2x_3x_4$ has been ignored.
- 21. $x_3 + x_2$
- 23. $x_1x_3x'_4 + x_1x'_2x'_3 + x'_1x'_2x_3 + x'_1x'_3x'_4$
- 25. $x_1x'_2x_3x'_4 + x'_1x_3x_4 + x'_1x'_3x'_4 + x'_1x'_2x'_4$
or
 $x_1x'_2x_3x'_4 + x'_1x_3x_4 + x'_1x'_3x'_4 + x'_1x'_2x'_3$
- 27. $x_1x_2 + x_2x'_4 + x_1x'_3x_4 + x'_1x'_2x'_3$
or
 $x_1x_2 + x_2x'_4 + x'_2x'_3x_4 + x'_1x'_3x'_4$
or
 $x_1x_2 + x_2x'_4 + x'_2x'_3x_4 + x'_1x'_2x'_3$
- 29. $x_1x_3 + x'_1x_2x'_3 + x'_2x'_3x'_4$

CHAPTER 9

EXERCISES 9.1

1. a. not commutative, not associative
b. The completed table is

\cdot	p	q	r	s	
p	p	q	r	s	
q	q	r	s	p	commutative
r	r	s	p	q	
s	s	p	q	r	

3. a. associative, not commutative b. commutative, not associative c. neither
d. both e. commutative, not associative
5. a. semigroup b. none c. none
d. monoid; $i = 1 + 0\sqrt{2}$ e. group; $i = 1 + 0\sqrt{2}$ f. group; $i = 1$
7. a. group; $i =$ zero polynomial d. group; $i = 1$
b. none e. group; $i = 0$
c. group; $i = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ f. monoid; $i =$ function mapping every x to 0

9.

\circ	R_1	R_2	R_3	F_1	F_2	F_3
R_1	R_2	R_3	R_1	F_3	F_1	F_2
R_2	R_3	R_1	R_2	F_2	F_3	F_1
R_3	R_1	R_2	R_3	F_1	F_2	F_3
F_1	F_2	F_3	F_1	R_3	R_1	R_2
F_2	F_3	F_1	F_2	R_2	R_3	R_1
F_3	F_1	F_2	F_3	R_1	R_2	R_3

Identity element is R_3 ; inverse for F_1 is F_1 ; inverse for R_2 is R_1 .

11. a. No—not the same operation
b. No—zero polynomial (identity) does not belong to P ; also, closure does not hold.
c. No—not every element of \mathbb{Z}^* has an inverse in \mathbb{Z}^*
d. Yes
13. $[\{0\}, +_{12}], [Z_{12}, +_{12}], [\{0, 2, 4, 6, 8, 10\}, +_{12}], [\{0, 4, 8\}, +_{12}], [\{0, 3, 6, 9\}, +_{12}], [\{0, 6\}, +_{12}]$
15. $\alpha_1 = i, \alpha_2 = (1, 2) \circ (3, 4), \alpha_3 = (1, 3) \circ (2, 4), \alpha_4 = (1, 4) \circ (2, 3), \alpha_5 = (1, 3) \circ (1, 2),$
 $\alpha_6 = (1, 2) \circ (1, 3), \alpha_7 = (1, 3) \circ (1, 4), \alpha_8 = (1, 4) \circ (1, 2), \alpha_9 = (1, 4) \circ (1, 3), \alpha_{10} = (1, 2) \circ (1, 4),$
 $\alpha_{11} = (2, 4) \circ (2, 3), \alpha_{12} = (2, 3) \circ (2, 4)$
17. a. No b. No c. Yes, but not an isomorphism
19. a. Yes; $f: \mathbb{Z} \rightarrow 12\mathbb{Z}, f(x) = 12x$
b. No; \mathbb{Z}_5 is finite, $5\mathbb{Z}$ is infinite
c. Yes; $f: 5\mathbb{Z} \rightarrow 12\mathbb{Z}, f(x) = \frac{12}{5}x$
d. No; both sets have 6 elements, but $[S_3, \circ]$ is noncommutative, $[\mathbb{Z}_6, +_6]$ is commutative.

- 21. a.** Closure: $\begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & w \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & w+z \\ 0 & 1 \end{bmatrix} \in M_2^0(\mathbb{Z})$. Matrix multiplication is associative. $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \in M_2^0(\mathbb{Z})$. The inverse of $\begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix}$ is $\begin{bmatrix} 1 & -z \\ 0 & 1 \end{bmatrix}$, which belongs to $M_2^0(\mathbb{Z})$.
- b.** f is a bijection and $f\left(\begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & w \\ 0 & 1 \end{bmatrix}\right) = f\left(\begin{bmatrix} 1 & w+z \\ 0 & 1 \end{bmatrix}\right) = w+z = z+w = f\left(\begin{bmatrix} 1 & z \\ 0 & 1 \end{bmatrix}\right) + f\left(\begin{bmatrix} 1 & w \\ 0 & 1 \end{bmatrix}\right)$
- c.** $f\left(\begin{bmatrix} 1 & 7 \\ 0 & 1 \end{bmatrix}\right) = 7$ and $f\left(\begin{bmatrix} 1 & -3 \\ 0 & 1 \end{bmatrix}\right) = -3$, $7 + (-3) = 4$, $f^{-1}(4) = \begin{bmatrix} 1 & 4 \\ 0 & 1 \end{bmatrix}$
- d.** $f^{-1}(2) = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$ and $f^{-1}(3) \begin{bmatrix} 1 & 3 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 3 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 5 \\ 0 & 1 \end{bmatrix}$ and $f\left(\begin{bmatrix} 1 & 5 \\ 0 & 1 \end{bmatrix}\right) = 5$
- 23. a.** $i \cdot i = i$ so $i = i^{-1}$ **b.** $x^{-1} \cdot x = x \cdot x^{-1} = i$, so $x = (x^{-1})^{-1}$
- 25. a.** $x \rho x$ because $i \cdot x \cdot i^{-1} = x \cdot i^{-1} = x \cdot i = x$. If $x \rho y$ then for some $g \in G$, $g \cdot x \cdot g^{-1} = y$ or $g \cdot x = y \cdot g$ or $x = g^{-1} \cdot y \cdot g = (g^{-1}) \cdot y \cdot (g^{-1})^{-1}$ so $y \rho x$. If $x \rho y$ and $y \rho z$ then for some $g_1, g_2 \in G$, $g_1 \cdot x \cdot g_1^{-1} = y$ and $g_2 \cdot y \cdot g_2^{-1} = z$ so $g_2 \cdot g_1 \cdot x \cdot g_1^{-1} \cdot g_2^{-1} = z$ or $(g_2 \cdot g_1) \cdot x \cdot (g_2 \cdot g_1)^{-1} = z$ and $x \rho z$.
- b.** Suppose G is commutative and $y \in [x]$. Then for some $g \in G$, $y = g \cdot x \cdot g^{-1} = x \cdot g \cdot g^{-1} = x \cdot i = x$. Thus $[x] = \{x\}$. Conversely, suppose $[x] = \{x\}$ for each $x \in G$. Let $x, y \in G$, and denote the element $y \cdot x \cdot y^{-1}$ by z . Then $x \rho z$, so $z = x$ and $y \cdot x \cdot y^{-1} = x$ or $y \cdot x = x \cdot y$.
- 27. a.** $i_L = i_L \cdot i_R = i_R$ so $i_L = i_R$ and this element is an identity in $[S, \cdot]$.
- b.** For example, $\begin{array}{c|cc} \cdot & a & b \\ \hline a & a & b \\ b & a & b \end{array}$
- c.** For example, $\begin{array}{c|cc} \cdot & a & b \\ \hline a & a & a \\ b & b & b \end{array}$
- d.** For example, $[\mathbb{R}^+, +]$
- 29.** Let $x \in S$ with left inverse y . Then $y \in S$, so let z be the left inverse of y . Then $x \cdot y = i_L \cdot (x \cdot y) = (z \cdot y) \cdot (x \cdot y) = z \cdot (y \cdot x) \cdot y = z \cdot i_L \cdot y = z \cdot y = i_L$, so y is also a right inverse of x . Also, $x \cdot i_L = x \cdot (y \cdot x) = (x \cdot y) \cdot x = i_L \cdot x = x$, so i_L is also a right identity in S and therefore an identity.
- 31.** For some fixed $a \in S$, let x_1 be the solution to $x \cdot a = a$. Let b be any element of S . Then $a \cdot x = b$ for some $x \in S$ and $x_1 \cdot b = x_1 \cdot (a \cdot x) = (x_1 \cdot a) \cdot x = a \cdot x = b$. Therefore x_1 is a left identity in S . Also, for any $b \in S$, there is an x such that $x \cdot b = x_1$; hence every element of S has a left inverse. Result follows from Exercise 29.
- 33.** If G is commutative, then $(x \cdot y)^2 = (x \cdot y) \cdot (x \cdot y) = x \cdot (y \cdot x) \cdot y = x \cdot (x \cdot y) \cdot y = (x \cdot x) \cdot (y \cdot y) = x^2 \cdot y^2$. For the converse, let $x, y \in G$; then $x \cdot y \cdot x \cdot y = x \cdot x \cdot y \cdot y$, and by left and right cancellation, $y \cdot x = x \cdot y$, so G is commutative.
- 35.** Closure: let $x, y \in B_k$. Then $(x \cdot y)^k = x^k \cdot y^k$ (because of commutativity) $= i \cdot i = i$, so $x \cdot y \in B_k$. Identity: $i^k = i$, so $i \in B_k$. Inverses: for $x \in B_k$, $(x^{-1})^k = (x^k)^{-1} = i^{-1} = i$, so $x^{-1} \in B_k$.
- 37. a.** $S \cap T \subseteq G$. Closure: for $x, y \in S \cap T$, $x \cdot y \in S$ because of closure in S , $x \cdot y \in T$ because of closure in T , so $x \cdot y \in S \cap T$. Identity: $i \in S$ and $i \in T$ so $i \in S \cap T$. Inverses: for $x \in S \cap T$, $x^{-1} \in S$ and $x^{-1} \in T$ so $x^{-1} \in S \cap T$.
- b.** No. For example, $\{0, 4, 8\}, +_{12}$ and $\{0, 6\}, +_{12}$ are subgroups of $[Z_{12}, +_{12}]$ but $\{0, 4, 6, 8\}, +_{12}$ is not a subgroup of $[Z_{12}, +_{12}]$ (not closed).

- 39. a.** Closure: let $f, g \in H_a$. Then $(f \circ g)(a) = f(g(a)) = f(a) = a$, so $f \circ g \in H_a$. Identity: the identity mapping on A maps a to a . Inverses: let $f \in H_a$. Then $f(a) = a$ so $f^{-1}(a) = a$, and $f^{-1} \in H_a$.
- b.** $(n-1)!$
- 41. a.** Let $x = a^{\bar{1}}, y = a^{\bar{2}} \in A$. Then $x \cdot y^{-1} = a^{\bar{1}} \cdot (a^{\bar{2}})^{-1} = a^{\bar{1}} \cdot (a^{-1})^{\bar{2}} = a^{\bar{1}-\bar{2}} \in A$. By Exercise 40, A is a subgroup.
- b.** $2^0 = 0, 2^1 = 2, 2^2 = 2 +_7 2 = 4, 2^3 = 6, 2^4 = 1, 2^5 = 3, 2^6 = 5$
- c.** $5^0 = 0, 5^1 = 5, 5^2 = 5 +_7 5 = 3, 5^3 = 1, 5^4 = 6, 5^5 = 4, 5^6 = 2$
- d.** $3^0 = 0, 3^1 = 3, 3^2 = 3 +_4 3 = 2, 3^3 = 1$
- 43. a.** $[\text{Aut}(S), \circ]$ is closed because composition of isomorphisms is an isomorphism (Practice 31). Associativity always holds for function composition. The identity function i_S is an automorphism on S . Finally, if f is an automorphism on S , so is f^{-1} .
- b.**
- | | | | | |
|-------------------|-------------------|---------|-----|-----|
| $0 \rightarrow 0$ | $0 \rightarrow 0$ | \circ | i | f |
| $1 \rightarrow 1$ | $1 \rightarrow 3$ | i | i | f |
| $2 \rightarrow 2$ | $2 \rightarrow 2$ | f | f | i |
| $3 \rightarrow 3$ | $3 \rightarrow 1$ | | | |
- $i:$ $f:$
- 45.** Let i_G and i_H denote the identity elements of G and H , respectively. Let f be an isomorphism, $f: G \rightarrow H$. Then $f(i_G) = i_H$ and because f is one-to-one, i_G is the only such element. Now let f be a homomorphism from G onto H ; then $f(i_G) = i_H$. Suppose i_G is the only such element, and let $f(g_1) = f(g_2)$ for $g_1, g_2 \in G$. Then $f(g_1 \cdot g_2^{-1}) = f(g_1) \cdot f(g_2^{-1}) = f(g_1) \cdot (f(g_2))^{-1} = f(g_1) \cdot (f(g_1))^{-1} = i_H$. Therefore $g_1 \cdot g_2^{-1} = i_G$ and $g_1 = i_G \cdot g_2 = g_2$. Thus f is one-to-one; f is already an onto homomorphism, so it is an isomorphism.
- 47. a.** The $+$ operation is a binary operation on E (well-defined and closed). Associativity holds because $([x] + [y]) + [z] = [x + y] + [z] = [(x + y) + z] = [x + (y + z)] = [x] + [y + z] = [x] + ([y] + [z])$. $[0]$ is the identity because $[x] + [0] = [x + 0] = [x]$ and $[0] + [x] = [0 + x] = [x]$. Each element has an inverse: $[x] + [-x] = [x + (-x)] = [0] = [-x] + [x]$. Commutativity holds because $[x] + [y] = [x + y] = [y + x] = [y] + [x]$.
- b.** The function $f: \mathbb{Z}_5 \rightarrow E_5$ given by $f(0) = [0], f(1) = [1], f(2) = [2], f(3) = [3], f(4) = [4]$ is a bijection. It is also a homomorphism. For x, y elements of \mathbb{Z}_5 , $f(x + y) = [x + y] = [x] + [y] = f(x) + f(y)$.
- c.** The inverse of $[10]$ is $[-10] = [4]$. The preimage of $[21]$ is 7.

EXERCISES 9.2

- 1.** $f(G)$ is closed: Let $f(x)$ and $f(y)$ be elements in $f(G)$. Then because f is a homomorphism, $f(x) + f(y) = f(x \cdot y)$. Because x and y belong to G , $x \cdot y$ belongs to G and $f(x \cdot y)$ is an element of $f(G)$. i_H belongs to $f(G)$: Let $f(x)$ be an element in $f(G)$. Then $f(x) + f(i_G) = f(x \cdot i_G) = f(x)$ and $f(i_G) + f(x) = f(i_G \cdot x) = f(x)$, so $f(i_G)$ is an identity for the subset $f(G)$ and therefore $f(i_G) = i_H$. Elements in $f(G)$ have inverses in $f(G)$: Let $f(x)$ be an element in $f(G)$. Then x belongs to G and x^{-1} exists in G . Then $f(x^{-1}) + f(x) = f(x^{-1} \cdot x) = f(i_G) = i_H$. Similarly $f(x) + f(x^{-1}) = i_H$, so $f(x^{-1}) = -f(x)$ and $f(x)$ has an inverse in $f(G)$.
- 3.** $K = 4\mathbb{Z}$
- 5. a.** $f((x, y) + (r, s)) = f(x + r, y + s) = (x + r) + (y + s) = (x + y) + (r + s) = f(x, y) + f(r, s)$.
- b.** $K = \{(x, -x) | x \in \mathbb{Z}\}$.
- 7.** $7 +_{12} S = \{7, 11, 3\}$
- 9. a.** \mathbf{H} has no row of all 0s and no two rows alike, so the minimum distance = 3 and the code is single-error correcting.

- b. \mathbf{H} can encode all of \mathbb{Z}_2^3 . $000 \rightarrow 000000$, $001 \rightarrow 001101$, $010 \rightarrow 010011$, $011 \rightarrow 011110$,
 $100 \rightarrow 100111$, $101 \rightarrow 101010$, $110 \rightarrow 110100$, $111 \rightarrow 111001$

11. For example, $\mathbf{H} =$

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

13. Let $X = (x_1, \dots, x_n)$ and $Y = (y_1, \dots, y_n)$ be elements of \mathbb{Z}_2^n . The i th component of $(X +_2 Y) \cdot \mathbf{H}$ is given by $(x_1 +_2 y_1) \cdot \mathbf{H}_{1i} +_2 (x_2 +_2 y_2) \cdot \mathbf{H}_{2i} +_2 \dots +_2 (x_n +_2 y_n) \cdot \mathbf{H}_{ni}$. Because the distributive law holds and addition modulo 2 is commutative, this expression equals $(x_1 \mathbf{H}_{1i} +_2 x_2 \mathbf{H}_{2i} +_2 \dots +_2 x_n \mathbf{H}_{ni}) +_2 (y_1 \mathbf{H}_{1i} +_2 y_2 \mathbf{H}_{2i} +_2 \dots +_2 y_n \mathbf{H}_{ni})$, which is the i th component of $X \cdot \mathbf{H} +_2 Y \cdot \mathbf{H}$.

15. Coset leaders Syndromes
 0000000 000
 0000001 001
 0000010 010
 0010000 011
 0000100 100
 0100000 101
 1000000 110
 0001000 111

17. The decoded word is 011000010101001.

EXERCISES 9.3

1. a. 0001111110 b. aaacaaaa c. 00100110

3.

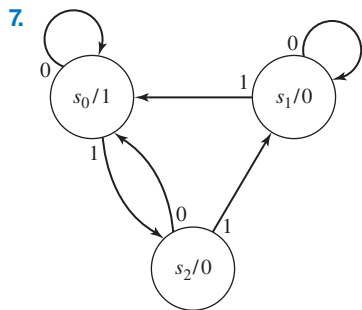
Present state	Next state		Output
	Present input		
	0	1	
s_0	s_1	s_1	0
s_1	s_2	s_1	1
s_2	s_2	s_0	0

Output is 010010

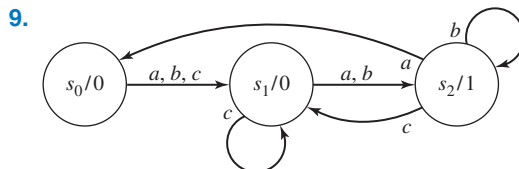
5.

Present state	Next state		Output
	Present input		
	0	1	
s_0	s_1	s_2	a
s_1	s_2	s_3	b
s_2	s_2	s_1	c
s_3	s_2	s_3	b

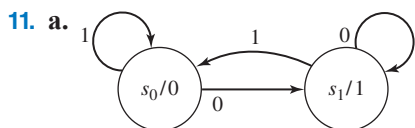
Output is abbcbb



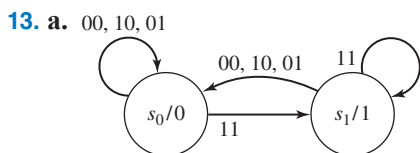
Output is 101110



Output is 0001101

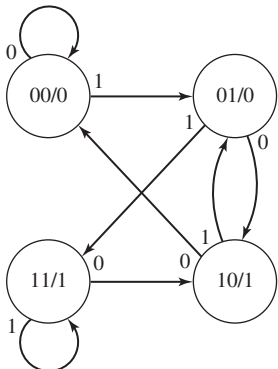


b. 010100

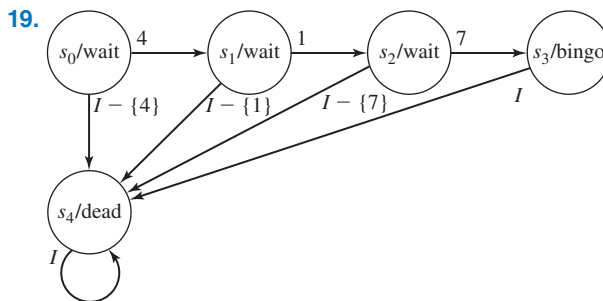
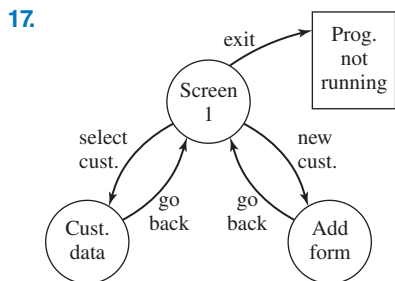


b. 010010

15. a. Name each state with the sequence of the last two input bits read.



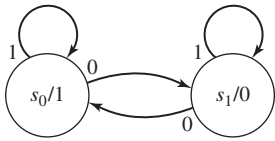
b. The length of time required to remember a given input grows without bound and eventually would exceed the number of states.



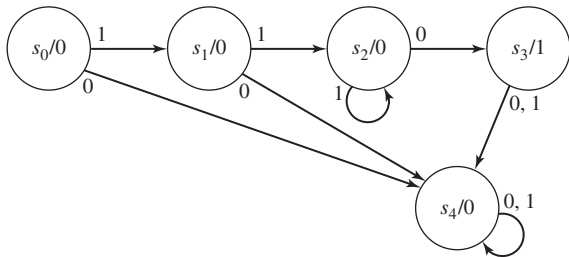
21. No

23. Yes

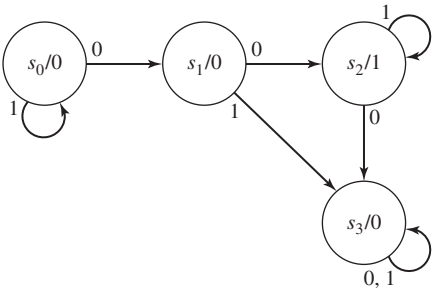
25. a.



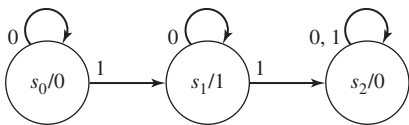
b.



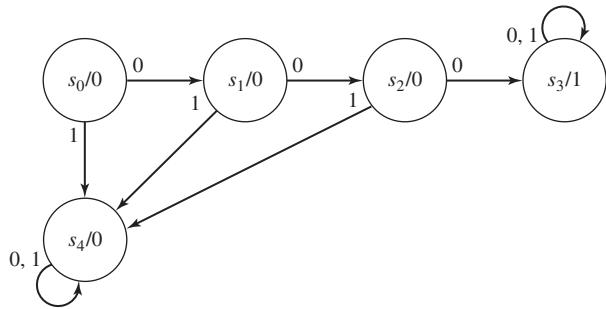
c.



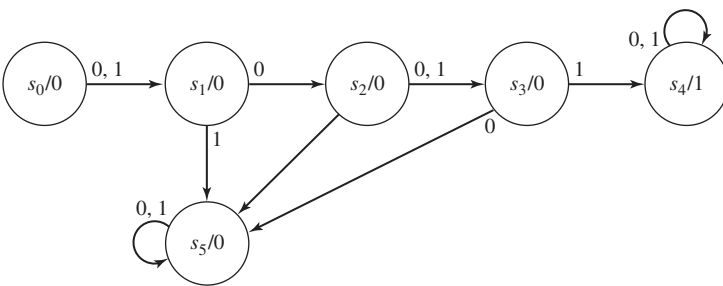
27. a.



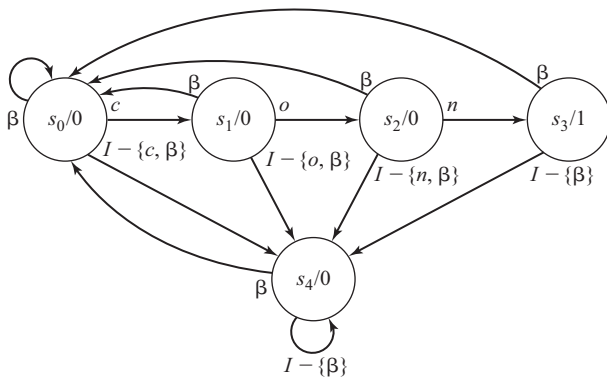
b.



c.



29. The object is to recognize the substring β con.



31. Once a state is revisited, behavior will be periodic because the input is always 0 and there is no choice of paths from a state. The maximum number of inputs that can occur before this happens is $n - 1$ (visiting all n states before repeating). The maximum length of a period is n (output from all n states, with the last state returning to s_0).
33. 0^*
35. $01^* \vee (110)^*$
37. $(1 \vee 01)(01)^*$
39. 10^*1
41. $1^* \vee (010)^*$
43. a. $0(0 \vee 1)^*1$ b. $1^*01^*(01^*0)^*1^*$ c. 100^*1
45. a. Yes b. No c. No
47. $dd^*(+ \vee -)dd^*$ where d stands for any digit
49. a. $(00)^*$ b. 111^*0 c. 1^*001^*
51. a. 0^*10^* b. $000(1 \vee 0)^*$ c. $(1 \vee 0)0(1 \vee 0)1(1 \vee 0)^*$
53. a. Proof is by induction on the length of the regular expression. For the base step, if $A = \emptyset, \lambda,$ or $i,$ then $A^R = \emptyset, \lambda,$ or $i.$ Assume that for all expressions of length $\leq k, A$ regular $\rightarrow A^R$ regular. Let A be a regular expression of length $k + 1.$ If $A = BC,$ where B and C are regular, then B^R and C^R are regular by inductive hypothesis and $A^R = C^R B^R,$ so A^R is regular. Similarly, if $A = B \vee C,$ then $A^R = B^R \vee C^R$ (regular), and if $A = B^*,$ then $A^R = (B^R)^*$ (regular).
- b. No—no regular expression describes this set.
55. beer, beter
57. beter, better, bettter
59. bit, but, beet
61. $b t$
63. s_1
65. $A = \{0\}, B = \{1, 2, 5\}, C = \{3, 4\}, D = \{6\}$

Present state	Next state		Output
	Present input		
	0	1	
A	C	D	1
B	C	B	0
C	B	A	1
D	C	B	1

67. $A = \{0\}, B = \{5\}, C = \{2\}, D = \{7, 8\}, E = \{1, 3\}, F = \{4, 6\}$

Present state	Next state		Output
	Present input		
	0	1	
A	E	C	0
B	F	D	0
C	E	F	0
D	D	E	0
E	C	E	1
F	B	F	1

69. $A = \{0\}, B = \{2\}, C = \{1, 4\}, D = \{3\}, E = \{5\}$

Present state	Next state		Output
	Present input		
	0	1	
A	C	D	0
B	E	C	0
C	B	C	1
D	C	B	2
E	C	A	2

71. $A = \{0, 2\}, B = \{1, 3\}, C = \{4\}$

Present state	Next state			Output
	Present input			
	a	b	c	
A	B	C	A	1
B	C	A	B	0
C	B	A	A	0

73. $A = \{0\}, B = \{2, 4\}, C = \{1, 5\}, D = \{3\}$

Present state	Next state		Output
	Present input		
	0	1	
A	D	A	0
B	C	B	0
C	B	D	1
D	A	B	1

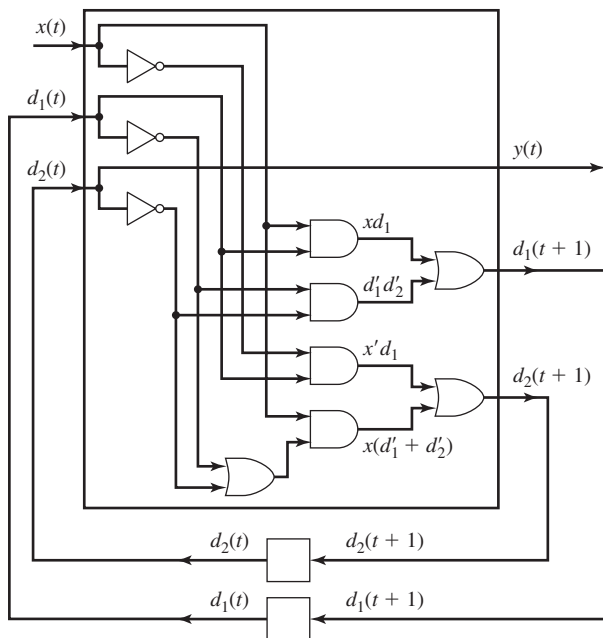
75. Possible answer:

	d_1	d_2	$x(t)$	$d_1(t)$	$d_2(t)$	$y(t)$	$d_1(t+1)$	$d_2(t+1)$
s_0	0	0	0	0	0	0	1	0
s_1	0	1	1	0	0	0	1	1
s_2	1	0	0	0	1	1	0	0
s_3	1	1	1	0	1	1	0	1
			0	1	0	0	0	1
			1	1	0	0	1	1
			0	1	1	1	0	1
			1	1	1	1	1	0

$$y(t) = d_1' d_2 + d_1 d_2' = d_2$$

$$d_1(t+1) = x' d_1' d_2' + x d_1' d_2' + x d_1 d_2' + x d_1 d_2 = d_1' d_2' + x d_1$$

$$d_2(t+1) = x d_1' d_2' + x d_1' d_2 + x' d_1 d_2' + x d_1 d_2 + x' d_1 d_2 = x(d_1' + d_2') + x' d_1$$



EXERCISES 9.4

1. a. halts with final tape $\cdots b 0 0 0 0 0 b \cdots$
 b. does not change the tape and moves forever to the left
3. One answer: State 2 is a final state.
 $(0, b, b, 2, R)$ blank tape or no more 1's, go to final state
 $(0, 1, 1, 1, R)$ has read odd number of 1's
 $(1, 1, 1, 0, R)$ has read even number of 1's
5. One answer: State 3 is a final state.
 $(0, 0, 0, 0, R)$ } pass over 0s to first 1
 $(0, 1, 1, 1, R)$ }
 $(1, 0, 0, 1, R)$ } pass over 0s to second 1
 $(1, 1, 1, 2, R)$ }
 $(2, b, b, 3, R)$ end of string, halt and accept
7. One answer: State 9 is a final state
 $(0, b, b, 9, R)$ accepts blank tape
 $(0, 0, 0, 0, R)$ } finds first 1, marks with X
 $(0, 1, X, 1, R)$ }
 $(1, 1, 1, 1, R)$ } searches right for 2s
 $(1, Y, Y, 1, R)$ }
 $(1, 2, Y, 3, R)$ } pair of 2s, marks with Y's
 $(3, 2, Y, 4, L)$ }
 $(4, Y, Y, 4, L)$ } searches left for 0s
 $(4, X, X, 4, L)$ }
 $(4, 1, 1, 4, L)$ }
 $(4, Z, Z, 4, L)$ }

- (4, 0, Z, 5, L) } pair of 0s, marks with Z's
- (5, 0, Z, 6, R) }
- (6, Z, Z, 6, R) } passes right to next 1
- (6, X, X, 6, R) }
- (6, 1, X, 1, R) }
- (6, Y, Y, 7, R) } no more 1s
- (7, Y, Y, 7, R) } no more 2s
- (7, b, b, 8, L) }
- (8, Y, Y, 8, L) }
- (8, X, X, 8, L) } no more 0s, halts and accepts
- (8, Z, Z, 8, L) }
- (8, b, b, 9, L) }

9. One answer: State 8 is a final state

- (0, 0, b, 1, R) } 0 read on left of w_1
- (1, 0, 0, 1, R) }
- (1, 1, 1, 1, R) } moves right to *
- (1, *, *, 2, R) }
- (2, X, X, 2, R) } passes over X's
- (2, 1, 1, 8, R) } nonzero on left of w_2 , halts and accepts
- (2, b, b, 8, R) }
- (2, 0, X, 3, L) } left symbols match
- (3, X, X, 3, L) } moves left to *
- (3, *, *, 4, L) }
- (4, 1, 1, 4, L) }
- (4, 0, 0, 4, L) } finds leftmost symbol
- (4, b, b, 0, R) }
- (0, 1, b, 5, R) } 1 read on left of w_1
- (5, 0, 0, 5, R) }
- (5, 1, 1, 5, R) } moves right to *
- (5, *, *, 6, R) }
- (6, X, X, 6, R) } passes over X's
- (6, 0, 0, 8, R) } non-one on left of w_2 , halts and accepts
- (6, b, b, 8, R) }
- (6, 1, X, 3, L) } left symbols match
- (0, *, *, 7, R) } word left of * is empty
- (7, X, X, 7, R) }
- (7, 0, 0, 8, R) } word right of * nonempty, halts and accepts
- (7, 1, 1, 8, R) }
- (0, b, b, 0, R) } w_1 initially empty

11. (0, 0, 1, 0, R) } changes 0 to 1
 (0, 1, 0, 0, R) } changes 1 to 0

13. (0, 1, 1, 1, R) } passes center point
 (1, 0, 0, 1, R) } adds 0 on right end
 (1, b, 0, 2, L) }
- (2, 0, 0, 2, L) }
- (2, 1, 1, 2, L) } adds 0 on left end
 (2, b, 0, 0, R) }
- (0, 0, 0, 0, R) } returns to center point

15. The general idea is to decrement the binary number by 1; with each decrement, add a 1 to the string of 1s being built.¹

(0, 1, 1, 0, R)	}	finds least significant digit
(0, 0, 0, 0, R)		
(0, b, b, 1, L)		
(1, 1, 0, 2, R)	}	if least significant digit = 1, then decrement
(2, 0, 0, 2, R)		
(2, 1, 1, 2, R)	}	finds end of original string
(2, b, b, 3, R)		
(3, 1, 1, 3, R)		
(3, b, 1, 4, L)	}	finds right end of new string writes 1 at end of new string
(4, 1, 1, 4, L)		
(4, b, b, 1, L)	}	goes back to original string
(1, 0, 1, 5, L)		
(5, 0, 1, 5, L)	}	if least significant digit = 0, change to 1 and look left for another 1 to decrement
(5, b, b, 8, R)		
(5, 1, 0, 6, L)	}	increment 0s to the left just incremented the final remaining 0, clean up found a 1 to decrement
(6, 1, 1, 2, R)		
(6, 0, 0, 2, R)	}	prepare to move to end of original string
(6, b, b, 7, R)		
(7, 0, b, 2, R)	}	the 1 just decremented was the most significant digit blank out the leading 0
(8, 1, b, 9, L)		
		clean up and halt

17. $f(n_1, n_2, n_3) = \begin{cases} n_2 + 1 & \text{if } n_2 > 0 \\ \text{undefined} & \text{if } n_2 = 0 \end{cases}$

19. (0, 1, 1, 1, R) } $n = 0$, add 1 and halt
 (1, b, 1, 4, R) }
 (1, 1, 1, 2, R) } $n = 1$, add additional 1 and halt
 (2, b, 1, 4, R) }
 (2, 1, 1, 3, R) }
 (3, 1, b, 3, R) } $n > 2$, erase extra 1s and halt
 (3, b, b, 4, R) }

21. One answer:

(0, 1, 1, 1, R)	}	$n = 0, 2 \cdot 0 = 0$
(1, b, b, 8, R)		
(1, 1, 1, 2, R)	}	$n > 0$, finds end of \bar{n}
(2, 1, 1, 2, R)		
(2, b, b, 3, L)		
(3, 1, X, 4, R)	}	changes 1 to X, adds 1 at right end of string
(4, X, X, 4, R)		
(4, 1, 1, 4, R)		
(4, b, 1, 5, L)		
(5, 1, 1, 5, L)	}	goes left to next 1 of \bar{n}
(5, X, X, 6, L)		
(6, X, X, 6, L)		
(6, 1, X, 4, R)		

¹My thanks to Alicia Kime of Fairmont State College and her student Tim Holmes for this solution.

- (6, b , b , 7, R)
 - (7, X , 1, 7, R)
 - (7, 1, 1, 7, R)
 - (7, b , b , 8, L)
 - (8, 1, b , 9, L)
- } \bar{n} is doubled, changes X 's to 1's
- } finds right end, erases extra 1, halts

23. One answer:

- (0, 1, b , 1, R)
 - (1, *, b , 3, R)
 - (1, 1, b , 2, R)
 - (2, 1, 1, 2, R)
 - (2, *, 1, 3, R)
- erases one extra 1
- $n_1 = 0$
- } $n_1 > 0$, replaces * with leftmost 1 of \bar{n}_1 , halts

25. One answer:

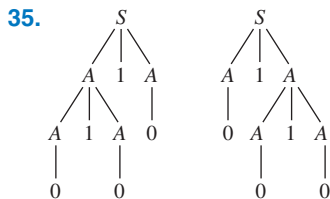
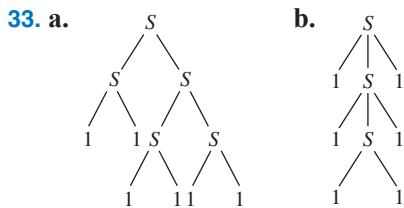
- (0, 1, 1, 0, R)
 - (0, *, *, 0, R)
 - (0, b , b , 1, L)
 - (0, X , X , 1, L)
 - (1, 1, X , 2, L)
 - (2, 1, 1, 2, L)
 - (2, *, *, 2, L)
 - (2, b , b , 3, R)
 - (2, X , X , 3, R)
 - (3, 1, X , 0, R)
 - (3, *, X , 4, L)
 - (4, X , X , 4, L)
 - (4, b , 1, 5, R)
 - (5, X , b , 5, R)
 - (5, 1, b , 5, R)
 - (5, b , b , 9, R)
 - (1, *, *, 6, R)
 - (6, X , X , 6, R)
 - (6, b , b , 7, L)
 - (7, X , b , 7, L)
 - (7, *, 1, 8, L)
 - (8, 1, 1, 8, L)
 - (8, X , b , 8, L)
 - (8, b , b , 9, R)
- } move to right end of 1s for n_2
- } X 's rightmost 1 of n_2
- } move to left end of 1s for n_1 , X leftmost 1
- $n_1 < n_2$
- } write 0 on tape and halt
- } all of n_2 used, now write $n_1 - n_2$ on tape
- } erase n_2
- } clean up $n_1 - n_2$ and halt

27. invoke T_1 , invoke T

EXERCISES 9.5

1. $L(G) = \{\lambda, a\}$; the grammar is type 0.
3. $L(G) = 0(10)^*$; the grammar is regular.
5. $G = (V, V_T, S, P)$ where $V = \{a, S\}$, $V_T = \{a\}$, and $P = \{S \rightarrow \lambda, S \rightarrow a\}$
7. The grammar of Exercise 3 is already regular.
9. $L(G) = (ab)^*$
11. $L(G) = aa^*bb^*$. G is context-sensitive. An example of a regular grammar that generates $L(G)$ is $G' = (V, V_T, S, P)$ where $V = \{a, b, A, B, S\}$, $V_T = \{a, b\}$ and $P = \{S \rightarrow aA, S \rightarrow aB, A \rightarrow aA, A \rightarrow aB, B \rightarrow bB, B \rightarrow b\}$

13. a. $\langle S \rangle ::= 0\langle A \rangle \mid 1\langle A \rangle$, $\langle A \rangle ::= 1\langle B \rangle\langle B \rangle$, $\langle B \rangle ::= 01 \mid 11$
 b. $\langle S \rangle ::= 0\mid 0\langle A \rangle$, $\langle A \rangle ::= 1\langle B \rangle$, $\langle B \rangle ::= 0\langle A \rangle \mid 0$
 c. $\langle S \rangle ::= 0\langle S \rangle \mid 11\langle A \rangle$, $\langle A \rangle ::= 1\langle A \rangle \mid 1$
15. a. $1^*1(00)^*$
 b. $S \rightarrow 1, S \rightarrow 1S, S \rightarrow 0A, A \rightarrow 0, A \rightarrow 0B, B \rightarrow 0A$
17. For example, $G = (V, V_T, S, P)$ where $V = \{(\ ,)\}$, $V_T = \{(\ ,)\}$, and $P = \{S \rightarrow \lambda, S \rightarrow (S)S\}$
19. For example, $G = (V, V_T, S, P)$ where $V = \{a, b, A, B, S\}$, $V_T = \{a, b\}$, and $P = \{S \rightarrow \lambda, S \rightarrow aSa, S \rightarrow bSb, S \rightarrow a, S \rightarrow b\}$
21. For example, $G = (V, V_T, S, P)$ where $V = \{0, 1, A, B, S\}$, $V_T = \{0, 1\}$, and $P = \{S \rightarrow 1A, A \rightarrow 1, A \rightarrow 1B, B \rightarrow 0, B \rightarrow 1, B \rightarrow 0B, B \rightarrow 1B\}$
23. For example, $G = (V, V_T, S, P)$ where $V = \{1, S\}$, $V_T = \{1\}$, and $P = \{S \rightarrow \lambda, S \rightarrow 11S\}$
25. For example, $G = (V, V_T, S, P)$ where $V = \{0, 1, S\}$, $V_T = \{0, 1\}$, and $P = \{S \rightarrow \lambda, S \rightarrow A, A \rightarrow 01, A \rightarrow 0S1\}$
27. For example, $G = (V, V_T, S, P)$ where $V = \{0, 1, S\}$, $V_T = \{0, 1\}$, and $P = \{S \rightarrow SS, S \rightarrow 01, S \rightarrow 10, S \rightarrow 0S1, S \rightarrow 1S0\}$
29. For example, $G = (V, V_T, S, P)$ where $V = \{0, S, A, B, X\}$, $V_T = \{0\}$, and $P = \{S \rightarrow A0B, A0 \rightarrow A00X, X0 \rightarrow 00X, XB \rightarrow B, A \rightarrow \lambda, B \rightarrow \lambda\}$
31. For example, $G = (V, V_T, S, P)$ where $V = \{0, 1, S, S_1, A, B, M\}$, $V_T = \{0, 1\}$, and $P = \{S \rightarrow \lambda, S \rightarrow 0S_1A, S \rightarrow 1S_1B, S_1 \rightarrow 0S_1A, S_1 \rightarrow 1S_1B, S_1 \rightarrow M, MA \rightarrow M0, MB \rightarrow M1, M \rightarrow \lambda, 0A \rightarrow A0, 0B \rightarrow B0, 1A \rightarrow A1, 1B \rightarrow B1\}$



37. The set of productions for G' is formed from the set of productions for G as follows: For A and B nonterminals, whenever $A \xRightarrow{*} B$ in G and $B \rightarrow \alpha$ is a production in G with $|\alpha| \geq 2$, add the production $A \rightarrow \alpha$ to the set, then eliminate all productions of the form $A \rightarrow B$. For any productions of the form $A \rightarrow a$, $a \in V_T$, add to the set of productions those obtained by replacing any A on the right of an existing production by a , then eliminate all productions of the form $A \rightarrow \lambda$. The remaining productions all have right sides with length ≥ 2 and $L(G') \subseteq L(G)$. Only λ and a finite number of one-length words may have been eliminated, so $L(G) - L(G')$ is a finite set.

Answers to Self-Tests

CHAPTER 1

SECTION 1.1

1. F If a statement is not a tautology, it does not have values that are all true, but that does not make them all false.
2. T Because of the truth table for disjunction, $() \vee T$ is T .
3. F The statement must have an implication as its main connective.
4. T
5. F The negation of a disjunction is the *conjunction* of the negations.

SECTION 1.2

1. T
2. F
3. T
4. F It is one in which hypothesis \rightarrow conclusion is always true.
5. T

SECTION 1.3

1. F In fact, $(\forall x)(P(x) \wedge [P(x)]')$ would be false in all interpretations.
2. T
3. T
4. F There is no one predicate wff defined on an interpretation, nor is the domain at all determined by truth values.
5. T

SECTION 1.4

1. T
2. F Existential instantiation should be used early.
3. F Universal instantiation would only strip off the leading universal quantifier.
4. F Wffs in propositional logic are not even wffs in predicate logic.
5. T Predicate logic is correct—only valid wffs are provable.

SECTION 1.5

1. T
2. F A single negated predicate is only one kind of Horn clause.
3. T
4. F A Prolog recursive rule is not a rule of inference.
5. T

SECTION 1.6

1. F It guarantees only that the output satisfies certain conditions, given that the input satisfies certain conditions.
2. F Nothing much can be said about the precondition without knowing the assignment, but at any rate the strict inequality will not go away.
3. F Program testing involves test data sets.
4. T
5. T

CHAPTER 2**SECTION 2.1**

1. F A conjecture that only asserts something about a finite number of cases can be proved by proving all cases.
2. T
3. F A universal quantifier is understood, because the formal statement of the theorem is $(\forall x)(x \text{ odd} \rightarrow 2 * x \text{ is even})$.
4. F The second statement is the converse of the first, not the contrapositive.
5. T

SECTION 2.2

1. T
2. F The basis step need not be $n = 1$.
3. T
4. T
5. F This omits the first $k - 1$ terms of the series.

SECTION 2.3

1. F A loop invariant remains true after loop termination.
2. F It means that correctness has been proved only given that the loop terminates.

3. F The first principle of induction is used because the values at pass $k + 1$ through the loop depend only on the values at pass k .
4. F But Q should give the desired result when the condition B' is true.
5. T (12 is the remainder when 42 is divided by 30).

SECTION 2.4

1. T
2. F The linear combination must equal 1.
3. F For example, $36 = 2^2 \cdot 3^2$, so the prime factors are 2 and 3, neither of which is $> 6 = \sqrt{36}$.
4. F For example, $\varphi(5) = 4$, which is not a prime.
5. T

CHAPTER 3

SECTION 3.1

1. T
2. F 18, 20, 22, 24 can be generated, but not, for example, 26.
3. F They are valuable because they represent natural ways of thinking about certain problems, but they typically use more storage and perform more operations than a corresponding iterative program.
4. T
5. T

SECTION 3.2

1. F Induction may be used to verify a proposed closed-form solution but not to determine such a solution from the recurrence relation.
2. F It is not linear because of the presence of the $S(n - 2)$ term.
3. F It is a pattern for the general solution, but it is not a closed-form solution because of the summation; in any specific case, there must be a closed-form expression to evaluate the summation.
4. T
5. T

SECTION 3.3

1. T
2. F It requires $m(n - m + 1)$ comparisons.
3. T
4. F The recursive version looks at the first item in the list, and if that is not the target, it searches the rest of the list. The “input size” goes down only by 1 each time the algorithm is invoked.
5. T

CHAPTER 4**SECTION 4.1**

1. F It is not a proper subset of itself.
2. T
3. T
4. F This is the closure property.
5. F It is a way to prove that certain sets are uncountable.

SECTION 4.2

1. T
2. F
3. T
4. T
5. F

SECTION 4.3

1. F
2. F The number of elements in the union plus the number of elements in the intersection is the sum of the number of elements in each set.
3. F All must be finite.
4. F
5. T

SECTION 4.4

1. T
2. T
3. F Use $C(n, r)$.
4. F It is $n!/n_1!n_2!n_3!$.
5. T

SECTION 4.5

1. F Combinations, not arrangements.
2. T
3. F All terms are found in row n .
4. T
5. T

SECTION 4.6

1. T
2. T
3. F This will work only if the events are independent.
4. F A random variable is not a variable; it is a function that assigns numerical values to the members of a sample space, and these values are generally not randomly chosen.
5. T

CHAPTER 5**SECTION 5.1**

1. T
2. F (x, x) can belong.
3. T
4. F The relation of equality is both a partial ordering and an equivalence relation.
5. F An equivalence relation does this.

SECTION 5.2

1. F
2. T
3. F The converse is true.
4. F The maximum value is used.
5. F See Example 18 and Practice 19.

SECTION 5.3

1. T
2. T
3. F
4. T
5. F If the data satisfy data integrity to begin with, then data integrity will still be true after a delete. It is referential integrity that may be lost.

SECTION 5.4

1. F It may not have an image for each member of the domain.
2. F Every element of the range has a preimage; begin with an element of the codomain.
3. T

4. T
5. F The original function must be a bijection.

SECTION 5.5

1. F Other constants may work where these do not.
2. F Either $f = \Theta(g)$ or $f = o(g)$
3. T
4. F
5. T

SECTION 5.6

1. T
2. F If the hash table entry does not match the target value, then the collision resolution algorithm must be followed before the search can be deemed a failure.
3. T
4. F It derives from the difficulty of finding the prime factors of n .
5. T

SECTION 5.7

1. T
2. F The two products are not necessarily equal.
3. T
4. F
5. F See Practice 58.

CHAPTER 6

SECTION 6.1

1. F A *complete* graph has an arc between any two nodes.
2. T
3. F A planar graph could still be drawn with arcs that cross.
4. F It means that nodes 2, 3, and 4 are all adjacent to some one node.
5. F It could be symmetric; it just doesn't have to be.

SECTION 6.2

1. T
2. F A complete (binary) tree has nothing to do with a complete graph.
3. T
4. T
5. T

SECTION 6.3

1. T
2. F This is the worst case; other cases could require fewer comparisons.
3. T
4. F The binary search tree depends on the order in which data elements are inserted.
5. F It must have at least $n!$ leaves.

SECTION 6.4

1. T
2. F In a prefix code, *no* code word is the prefix of another code word.
3. F Characters that occur most frequently have the shortest strings, and frequency does not affect the number of 0s versus the number of 1s.
4. T
5. F The code table file (giving the code for each character) must be stored along with the encoded file.

CHAPTER 7**SECTION 7.1**

1. T
2. T
3. F $\mathbf{R} = \mathbf{A} \vee \mathbf{A}^{(2)} \vee \dots \vee \mathbf{A}^{(n)}$
4. F That's what $\mathbf{R} = \mathbf{A} \vee \mathbf{A}^{(2)} \vee \dots \vee \mathbf{A}^{(n)}$ does; Warshall's algorithm expands the set of nodes available for use on a path.
5. F Warshall's algorithm computes transitive closure.

SECTION 7.2

1. T
2. F The graph can have at most two odd nodes.
3. T
4. F Some arcs may go unused.
5. F No efficient algorithm is known, but trial and error solves the problem.

SECTION 7.3

1. F This is how Prim's algorithm works.
2. F A greedy algorithm does not necessarily divide a problem into smaller subproblems, but it takes the "best" action based on limited knowledge at each step.
3. T
4. T
5. F It will generally not form a tree at all.

SECTION 7.4

1. F The starting node for a depth-first search can be any node in the graph.
2. T
3. T
4. F It is the equivalent of *depth-first* search, assuming that sibling nodes are labeled in order from left to right.
5. F Use a succession of *depth-first* searches.

SECTION 7.5

1. F Any path between some set of two nodes must pass through it.
2. T
3. T
4. T
5. F The root is a special case.

CHAPTER 8**SECTION 8.1**

1. F $x + x' = 1$
2. T
3. T
$$\begin{aligned} x + (y + x \cdot z) &= x + (x \cdot z + y) \\ &= x \cdot 1 + (x \cdot z + y) \\ &= (x \cdot 1 + x \cdot z) + y \\ &= x \cdot (1 + z) + y \\ &= x \cdot (z + 1) + y \\ &= x \cdot 1 + y \\ &= x + y \end{aligned}$$
4. T
5. T

SECTION 8.2

1. T $(x + y)' = x' \cdot y'$
2. F It has as many terms as the function has 1-values (or one term if the function has all 0-values).
3. F Only one half-adder is needed (see Figure 8.14a).
4. T Because $x + x = x$
5. F This usually results in an unnecessarily large number of devices; start again from the truth function.

SECTION 8.3

1. T
2. T
3. F Look for two-square blocks first and work up from there.
4. T
5. F The term is essential if the check for that row is the only one in some column.

CHAPTER 9**SECTION 9.1**

1. F This describes a commutative operation.
2. T Even though this is not the definition of the identity
3. T
4. T By Lagrange's theorem.
5. F f must also be a bijection.

SECTION 9.2

1. T
2. T
3. F \mathbf{H} maps Z_2^n to Z_2^r .
4. T If the syndrome of X equals 0_r , then X is in the kernel of the homomorphism generated by \mathbf{H} and is a code word; it will be assumed that no errors occurred.
5. F No coding scheme can guarantee zero bit errors.

SECTION 9.3

1. T
2. T $(0 \vee 1)*00$
3. F See Figure 9.8a.
4. T
5. F

SECTION 9.4

1. F Entering a final state causes a Turing machine to halt, but it can also halt in a nonfinal state if there is no instruction for the current state–input pair.
2. F There will be $n + 2$ 1s on the tape.
3. F
4. F This is the version with the quantifiers reversed that is trivially true.
5. T

SECTION 9.5

1. T
2. F Depending on the productions selected, nonterminal symbols may never be removed.
3. F There may be an equivalent grammar for the language that is context-free.
4. T
5. T

Index

A

- ABA routing number, 437
- Absolute value, 107
- Absorption property, 633
- Ackermann's function, 407
- Acquaintanceship graph, 499
- Acyclic graph, 482
- Addition inference rule, 29
- Addition modulo n , 343, 438, 692
- Addition of matrices, 449
- Addition principle, 254–55
 - definition of, 254
- Adjacency list, 494
- Adjacency matrix, 492
- Adjacency relation, 555
- Adjacent nodes, 481
- Adleman, Len, 431
- AES (Advanced Encryption Standard), 431
- Algebraic structure, 686–708. *See also* Group
 - associative property, 687
 - commutative property, 687
 - free monoid, 695
 - group of permutations, 693
 - identity element, 687
 - inverse element, 687
 - monoid, 689
 - semigroup, 689
 - semigroup of transformations, 693
- ALGOL, 164, 788
- Algorithm
 - definition of, 12
 - divide-and-conquer, 468
 - graph traversal, 596–604
 - greedy, 586
 - optimal, 533
 - recursive, 166–71
- Algorithms
 - ArtPoint*, 610
 - Bellman-Ford*, 592
 - BinarySearch*, 169, 208
 - BreadthFirst*, 599
 - BubbleSort*, 214
 - CombGenerator*, 285
 - DepthFirst*, 597
 - Dijkstra's*, 582
 - Euclidean*, 133
 - EulerPath*, 574
 - Floyd's*, 593
 - HeapSort*, 527
 - HuffmanTree*, 542
 - Inorder*, 515
 - Kruskal's*, 594
 - MatrixMultiplication*, 452
 - MergeSort*, 215
 - Minimize*, 742
 - PermGenerator*, 282, 283
 - Postorder*, 515
 - Preorder*, 514
 - QuickSort*, 215
 - SelectionSort*, 169, 215
 - SequentialSearch*, 204, 207
 - Sum-of-Products*, 643
 - TautologyTest*, 13
 - TopologicalSort*, 360
 - Warshall*, 563
- Alphabet, 694, 782
- Alternating group, 701
- Ambiguous grammar, 798
- Analysis of algorithms, 203–10, 314, 415
 - average case, 205, 314
 - binary search, 209
 - breadth-first search, 601
 - computational complexity, 416
 - definition of, 203
 - depth-first search, 601
 - Dijkstra's algorithm, 586
 - Euclidean algorithm, 210
 - Euler path, 565
 - lower bound, 511
 - sequential search, 205, 207
 - upper bound, 210
 - using recurrence relations, 206–10
 - Warshall's algorithm, 565
 - worst case, 205
- AND connective, 11, 75
- AND gate, 12, 639
- Antecedent, 3
- Antisymmetric relation, 332
- Appel, Kenneth, 507
- Arc, 477, 478
- Archimedes, 286
- Aristotle, 59, 71
- Arithmetic progression (arithmetic sequence), 124
- Array, 448
- Articulation point., 607
- Assertion, 85–87
 - definition of, 86
- Assignment rule of inference, 87–89
 - definition of, 87
- Associative equivalence rule, 28
- Associative property, 9, 234, 687
- Asymmetric encryption, 431
- Asymmetric relation, 349
- Atias, Nir, 497
- Augmented matrix, 453
- Automorphism, 714
- Axiom, 98, 773

B

- Back arc, 608
- Backus-Naur form (BNF), 163, 787
- Ballot problem, 175
- BASIC, 259
- Basis (basis step), 111
- Bayes' theorem, 308–9
 - statement of, 309
- Behm, Patrick, 136
- Bell numbers, 354

- Bellman-Ford algorithm, 592
 - Benford's law, 325
 - Benoit, Paul, 136
 - Bernoulli experiment, 313
 - Bernoulli trial, 313
 - Bernoulli, Jacob, 313
 - Biconnected components, 608
 - Big oh, 417
 - Big theta, 413
 - Bijection, 390
 - Binary adder, 731–33
 - Binary connective, 3
 - Binary GCD algorithm, 142
 - Binary operation, 228–30
 - associative, 687
 - commutative, 687
 - definition of, 229
 - well-defined, 229, 686
 - Binary predicate, 40
 - Binary relation, 328–36
 - antisymmetric, 332
 - asymmetric, 349
 - closure of, 334–36
 - complement of, 331
 - definition of, 329
 - and directed graph, 555–57
 - equivalence relation, 339–44
 - intersection of, 331
 - inverse of, 350
 - irreflexive, 349
 - many-to-many, 330
 - many-to-one, 330
 - one-to-many, 330
 - one-to-one, 330
 - partial ordering, 336–38
 - properties of, 332–34
 - reflexive, 332
 - reflexive closure of, 334
 - from S to T, 330
 - symmetric, 332
 - symmetric closure of, 334
 - transitive, 332
 - transitive closure of, 334, 561
 - union of, 331
 - Binary search algorithm, 169, 208
 - Binary search analysis, 209
 - Binary search tree, 352, 511, 534
 - Binary string, 163
 - Binary tree, 510
 - Binary tree search, 352, 534
 - Binomial coefficient, 297
 - Binomial distribution, 313–14
 - definition of, 313
 - Binomial theorem, 294–98
 - statement of, 296
 - Bipartite complete graph, 483
 - Birthday problem, 318
 - Blind key, 368
 - Block cipher, 429
 - Blocks of a partition, 339
 - BNF. *See* Backus-Naur form
 - Boole, George, 458, 617
 - Boolean algebra, 618–31
 - absorption property, 633
 - complement of an element, 625
 - definition of, 620
 - De Morgan's laws, 624
 - double negation property, 624
 - dual of a property, 623
 - idempotent property, 622
 - isomorphism, 629
 - modular property, 633
 - universal bound property, 624
 - Boolean AND, 458
 - Boolean expression, 639
 - Boolean matrix, 458–59
 - Boolean AND, 458
 - Boolean OR, 458
 - multiplication, 458
 - and reachability, 558
 - Boolean OR, 458
 - Bottom up parsing, 794
 - Bourg, David M., 749
 - Breadth-first search, 598–601
 - analysis of, 601
 - B-tree, 526
 - Bubble sort algorithm, 214
 - Byte, 257
- C**
- ©, 224
 - Caesar, Julius, 428
 - Caesar cipher, 428
 - Cancellation laws, 696
 - Canonical parity-check matrix, 721
 - Canonical product-of-sums form, 660
 - Canonical sum-of-products form, 643
 - Cantor, Georg, 237, 401
 - Cantor's diagonalization method, 237
 - Cantor's theorem, 401
 - Carbin, Michael, 654
 - Cardinality
 - of a relation, 366
 - of a set, 236, 401
 - Cartesian product (cross product), 233
 - Castile, Tracy, 262
 - Catalan numbers, 175, 263, 293, 526
 - Cayley, Arthur, 708
 - Cayley's theorem, 708
 - Ceiling function, 386
 - Center of a group, 713
 - Chain, 338
 - Chaining, 426
 - Characteristic equation, 190
 - Characteristic function, 406
 - Chinese remainder theorem, 432, 445
 - Chomsky, Noam, 786
 - Chomsky hierarchy, 790
 - Chromatic number of a graph, 507
 - Chung, Fan, 287
 - Church, Alonzo, 770, 771, 773
 - Church-Turing thesis, 770
 - Ciphertext, 428
 - Circular left shift, 430, 439
 - Closed form solution, 180
 - Closure of binary relations, 334–36, 556
 - definition of, 334
 - Coding theory, 714–26
 - canonical parity-check matrix, 721
 - check bits, 721
 - coset leader, 723
 - double-error detecting code, 715
 - group code, 719
 - Hamming distance, 718
 - information bits, 721
 - maximum likelihood decoding, 715
 - minimum distance of a code, 718
 - perfect code, 722
 - single-error correcting code, 715
 - syndrome, 724
 - weight of a code word, 719
 - Codomain, 383
 - Coefficient, 690
 - Collision, 425
 - Collision resolution, 425
 - Combinational network, 638–48, 744
 - definition of, 642
 - Combinations, 274–77
 - definition of, 274
 - eliminating duplicates, 279
 - generating, 284
 - with repetitions, 279
 - Combinatorial proof, 296
 - Combinatorics, 252–87
 - addition principle, 254
 - combinations, 277
 - decision tree, 257–58
 - definition of, 252
 - multiplication principle, 253
 - permutations, 272–74
 - pigeonhole principle, 269
 - Common difference, 124
 - Common ratio, 124

- Commutative diagram, 391, 479, 628
 - Commutative equivalence rule, 28
 - Commutative group, 688
 - Commutative property, 9, 234, 687
 - Complement
 - of a binary relation, 331
 - of a boolean algebra element, 625
 - of a graph, 506
 - of a set, 232
 - Complement property, 9, 234
 - Complete binary tree, 510
 - Complete formal system, 27, 58
 - Complete graph, 482
 - Composite number, 107
 - Composition function, 391
 - Composition of functions, 390–92
 - definition of, 391
 - Compound statement, 2
 - Computational complexity, 416, 776–78
 - Computational procedure, 769
 - Computer security, 143, 427–35
 - ciphertext, 428
 - cryptographic hash function, 434
 - cryptology, 427–33
 - one-way encryption, 434
 - password encryption, 433–35
 - plaintext, 428
 - Concatenation, 163, 694
 - Conclusion of an argument, 25
 - Conditional probability, 306–7
 - definition of, 307
 - independent events, 307
 - Conditional rule of inference, 90–92
 - definition of, 90
 - Conditional statement, 90
 - Congruence modulo n , 343
 - Conjunct, 2
 - Conjunction, 2
 - Conjunction inference rule, 29
 - Conjunctive normal form, 660
 - Connected component of a graph, 603
 - Connected graph, 482
 - Connective. *See* Logical connective
 - Consequent, 3
 - Constant symbol, 41
 - Context free (type 2) grammar, 790, 793–95
 - Context free language, 790
 - Context sensitive (type 1) grammar, 790
 - Context sensitive language, 790
 - Contradiction, 8, 104–6
 - Contraposition, 103–4
 - Contraposition inference rule, 37
 - Contrapositive, 21, 103
 - Converse, 21, 103
 - Correct formal system, 27, 58
 - Correct program, 85
 - Coset, 717
 - Countable set, 236
 - Counterexample, 99
 - Critical path, 359
 - Cryptographic hash function, 434
 - Cryptography, 427–33
 - AES, 431
 - asymmetric encryption, 431
 - block cipher, 429
 - Caesar cipher, 428
 - definition of, 428
 - DES, 429, 656
 - diffusion, 429
 - private key encryption, 431
 - public key encryption, 431
 - RSA public key encryption
 - algorithm, 431
 - simple substitution cipher, 429
 - symmetric encryption, 431
 - Cryptology, 428
 - Cutler, William, 286
 - Cycle in a graph, 482
 - Cycle notation, 395
 - Cyclic group, 713
- D**
- Darwin, Charles, 211
 - Data compaction, 540
 - Data compression, 540
 - Data flow diagram, 480
 - Data integrity, 375
 - Database, 365–76
 - add operation, 375
 - blind key, 368
 - composite primary key, 367
 - conceptual model of, 365
 - data integrity, 375
 - definition of, 365
 - delete operation, 375
 - entity, 365
 - entity attributes, 365
 - entity integrity, 367, 375
 - entity-relationship diagram, 365
 - entity-relationship model, 365–66
 - foreign key, 368
 - integrity, 375–76
 - join operation, 370
 - metadata, 366
 - modify operation, 375
 - operations, 369–73
 - outer join, 369
 - primary key, 367
 - project operation, 369
 - in Prolog, 73
 - referential integrity, 375
 - relational, 366
 - relational algebra, 372
 - relational calculus, 372
 - relational model, 366–69
 - restrict operation, 369
 - SQL, 372
 - surrogate key, 368
 - tuple, 366
 - Decision problem, 771–76
 - definition of, 772
 - negative solution for, 772
 - positive solution for, 772
 - uncomputable, 772
 - undecidable, 772
 - unsolvable, 772
 - Decision tree, 257–58, 511, 529–36
 - definition of, 529
 - for searching, 529–35
 - for sorting, 535–36
 - Declarative language, 73
 - Deduction method, 32
 - Deductive reasoning, 99
 - Degree of a node, 481
 - Degree of a polynomial, 691
 - Degree of a relation, 366
 - Degree of separation, 499
 - Delay element, 744
 - DeMoivre’s theorem, 126
 - De Morgan, Augustus, 9, 507
 - De Morgan’s laws, 10, 28, 178, 248, 250, 624, 633
 - Denumerable set, 236, 401
 - Depth of a node, 510
 - Depth-first search, 81, 596–98
 - analysis of, 601
 - Dequeue, 599
 - Derangement, 396
 - Derivation in a grammar, 784
 - Derivation rules, 803–04
 - definition of, 27
 - for predicate logic, 58–62
 - for propositional logic, 28–33
 - DES (Data Encryption Standard), 429, 656
 - Descriptive language, 73
 - Determinant of a matrix, 473
 - Diagonal matrix, 463
 - Difference of sets, 233
 - Diffusion, 429
 - Digraph. *See* Directed graph

- Dijkstra, Edsger W., 581
 Dijkstra's algorithm, 581
 analysis of, 586
 Dimensions of a matrix, 446
 Direct proof, 101–3
 Directed graph
 connected, 568
 definition of, 478
 initial point, 478
 path in, 483
 reachability in, 557–62, 602
 reachability matrix, 559
 reachable node in, 483, 557
 terminal point, 478
 Warshall's algorithm, 562–65
 Disjoint cycles, 395
 Disjoint sets, 232
 Disjunct, 3
 Disjunction, 3
 Disjunctive normal form, 643
 Disjunctive syllogism, 37
 Distributive inference rule, 37
 Distributive property, 9, 234, 250
 Divide-and-conquer algorithm, 208, 468
 Divide-and-conquer recurrence relation, 193–97
 Divides, 107
 DNA, 205
 Document Type Definition, 788
 Domain
 of a function, 383
 of an interpretation, 41
 Don't-care condition, 653
 Double negation equivalence rule, 28
 Double negation property, 624, 633
 DTD, 788
 Dual
 of Boolean algebra property, 623
 of set identity, 235
 of a tautological equivalence, 9
 Dummy variable, 41
 Dwyer function, 407
- E**
- Edge, 477, 478
 Elementary row operations, 454
 Empty set, 224
 Empty string, 163, 694
 Enqueue, 599
 Entity integrity, 367, 375
 Entity relationship diagram, 365
 Equal functions, 387
 Equal matrices, 447
 Equal sets, 223
- Equivalence class, 339
 Equivalence connective, 3
 Equivalence relation, 339–44
 definition of, 339
 and equivalence class, 339
 and partitions, 340
 Equivalence rules
 associative, 28
 commutative, 28
 definition of, 28
 De Morgan's laws, 28
 double negation, 28
 implication, 28
 Equivalent Boolean expressions, 645
 Equivalent grammars, 791
 Equivalent sets, 401–2
 Cantor's theorem, 401
 definition of, 401
 Equivalent states, 739
 Equivalent wffs, 8
 Erasing convention, 789
 Euclid, 98, 133, 148
 Euclidean algorithm, 133–35, 144, 145, 210, 217, 432
 binary GCD algorithm, 142
 greatest common divisor, 133
 Euler, Leonhard, 488, 571
 Euler path, 572, 573
 Euler phi function, 149–51, 431
 definition of, 150
 Euler's formula, 488
 Even node, 572
 Even number, 101
 Even permutation, 700
 Event, 302
 Exclusive OR, 22, 430, 634, 656
 Exhaustive proof, 100–101
 Existential generalization, 59, 62
 Existential instantiation, 59, 60
 Existential quantifier, 40
 Expected value, 310–12
 definition of, 310
 linearity of, 312
 Expert system, 81
 Exportation inference rule, 37
 Extensible Markup Language, 788
- F**
- Faivre, Alain, 136
 Fermat, Pierre de, 143
 Fermat's last theorem, 143
 Fermat's little theorem, 432, 445
 Fibonacci sequence, 159, 188, 301
 Field-programmable gate array (FPGA), 647
- Figurate numbers, 200
 Final state, 734, 765
 Finite set, 223
 Finite state machine, 479, 728–48
 binary adder, 731–33
 definition of, 729
 equivalent states in, 739
 final state, 734
 input alphabet, 729
 k-equivalent states in, 739
 Kleene's theorem, 737
 minimization, 737–44
 minimization algorithm, 742
 next-state function, 729
 output alphabet, 729
 output function, 729
 recognition by, 734
 and regular sets, 737
 and sequential networks, 744–48
 state graph, 730
 state table, 730
 unreachable state of, 737
 First principle of induction, 110–12, 250
 definition of, 111
 First order recurrence relation, 182
 Five-color theorem, 507
 Flip-flop, 744
 Floating point operations, 460
 Floor function, 386
 Floyd's algorithm, 593
 Foreign key, 368
 Forest, 510
 Formal language, 782–95. *See also*
 Language
 and computational devices, 792–93
 definition of, 785
 generated by a grammar, 784
 hierarchy of, 793
 Formal logic, 1–96
 predicate logic, 58–69, 773
 propositional logic, 25–35
 Formal system
 completeness of, 27, 58
 correctness of, 27, 58
 Four-color problem, 507
 Free monoid, 695
 Free tree, 509
 Free variable, 42
 Full binary tree, 510
 Full-adder, 649
 Function, 381–402
 Ackermann's function, 407
 bijection, 390
 as a binary relation, 383
 ceiling function, 386

- characteristic, 406
 - codomain, 383
 - composition, 391
 - composition of, 390–92
 - definition of, 383
 - domain, 383
 - Dwyer function, 407
 - equality of, 387
 - floor function, 386
 - hash function, 441
 - identity function, 392
 - image under, 383
 - injective, 389
 - inverse, 392–94
 - left inverse, 408
 - modulo, 386
 - of more than one variable, 385
 - next-state, 729
 - nondeterministic, 411
 - number of, 397–400
 - number-theoretic, 767
 - one-to-one, 389
 - onto, 388
 - order of magnitude of, 412–21
 - output, 729
 - partial, 767
 - permutation, 394–96
 - preimage under, 383
 - properties of, 388–90
 - range of, 388
 - right inverse, 408
 - Smorynski function, 407
 - surjective, 388
 - total, 767
 - truth, 640
 - Turing-computable, 768
 - Fundamental Theorem of Arithmetic, 144–48
 - definition of, 144
 - Fuzzy logic, 23, 250
 - Fuzzy set, 250
- G**
- Gates, William, 211
 - Gauss, Karl Friedrich, 110, 454
 - Gaussian elimination, 453–57
 - Generating permutations, 280–85
 - Generator of a cyclic group, 713
 - Geometric progression (geometric sequence), 124
 - Global clustering coefficient, 500
 - Gödel, Kurt, 98, 773
 - Goldbach conjecture, 149, 154
 - Golden ratio, 160, 219
 - Goodman, Jacob, 211
 - Graham, Ronald, 287
 - Grammar
 - ambiguous, 798
 - Backus-Naur form, 787
 - bottom-up parsing, 794
 - Chomsky hierarchy, 790
 - classes of, 789–92
 - context-free (type 2), 790, 793–95
 - context-sensitive (type 1), 790
 - direct generation (direct derivation) in, 784
 - equivalent, 791
 - erasing convention, 789
 - generation (derivation) in, 784
 - language generated by, 784
 - parse tree, 793
 - phrase-structure (type 0) grammar, 783
 - production, 783
 - regular (type 3), 790
 - start symbol, 783
 - terminal, 783
 - top-down parsing, 794
 - Graph, 476–96. *See also* Tree
 - acquaintanceship graph, 499
 - acyclic, 482
 - adjacency list, 494
 - adjacency matrix, 492
 - adjacency relation, 555
 - adjacent nodes in, 481
 - algorithms, 553–604
 - applications, 479–81
 - arc, 477, 478
 - articulation point, 607
 - Bellman-Ford algorithm, 592
 - biconnected, 608
 - biconnected components, 608
 - bipartite complete, 483
 - breadth-first search of, 598–601
 - chromatic number of, 507
 - colorability, 506
 - coloring of, 507
 - complement of, 506
 - complete, 482
 - connected, 482
 - connected component of, 603
 - cycle in, 482
 - definition (formal), 478
 - definition (informal), 477
 - degree of a node, 481
 - degree of separation, 499
 - depth-first search of, 596–98
 - Dijkstra’s algorithm, 581
 - directed. *See* Directed graph
 - dual graph for a map, 507
 - edge, 477, 478
 - endpoints of an arc, 478
 - Euler path in, 572
 - Euler’s formula, 488
 - even node, 572
 - five-color theorem, 507
 - Floyd’s algorithm, 593
 - four-color problem, 507
 - global clustering coefficient, 500
 - Hamiltonian circuit in, 576
 - homeomorphic, 491
 - in-degree of a node, 579
 - isolated node, 481
 - isomorphic, 484–87
 - isomorphism, 486
 - Kruskal’s algorithm, 594
 - Kuratowski theorem, 491
 - labeled, 479
 - length of a path, 482
 - loop in, 481
 - loop-free, 481
 - map-coloring problem, 506
 - minimal spanning tree, 587
 - node, 477, 478
 - odd node, 572
 - out-degree of a node, 579
 - parallel arcs in, 481
 - path in, 482
 - Petersen, 492
 - planar, 487–92
 - Prim’s algorithm, 587
 - self-complementary, 506
 - shortest path problem, 581–87
 - simple, 481
 - six-color theorem, 508
 - spanning tree in, 587
 - sparse adjacency matrix, 494
 - subgraph of, 482
 - theorem on Euler paths, 573
 - traversal, 596
 - traversal algorithms, 596–604
 - trees, 509–21
 - vertex, 477, 478
 - weighted, 479
 - Graph coloring problem, 778
 - Greatest common divisor, 133, 144
 - Greatest element, 338
 - Greatest lower bound, 637
 - Greedy algorithm, 586
 - Group
 - alternating, 701
 - automorphism, 714
 - Cayley’s theorem, 708
 - center of a, 713
 - commutative, 688
 - coset, 717
 - cyclic, 713

- Group (*cont.*)
- definition of, 688
 - generator of, 713
 - homomorphism, 702
 - idempotent element, 713
 - improper subgroup of, 700
 - isomorphic, 702–8
 - isomorphism, 702
 - kernel of a homomorphism, 716
 - Lagrange’s theorem, 701
 - left cancellation law, 696
 - left identity element, 712
 - left inverse element, 713
 - order of, 698
 - permutation group, 700
 - of permutations, 693
 - proper subgroup of, 700
 - right cancellation law, 696
 - right identity element, 712
 - right inverse element, 713
 - subgroup of, 699
 - symmetric group of degree n , 693
 - of symmetries of an equilateral triangle, 710
- Group code, 719
- H**
- Haken, Wolfgang, 507
- Half-adder, 649
- Halting problem, 774
- Hamilton, William Rowan, 576
- Hamiltonian circuit, 576, 777
- Hamming distance, 718
- Hamming, Richard W., 718
- Hash function, 424, 441
- Hash table, 425
- Hashing, 424–27
- chaining, 426
 - collision, 425
 - collision resolution, 425
 - cryptographic hash function, 434
 - hash function, 424
 - hash table, 425
 - linear probing, 425
 - load factor, 427
- Hasse diagram, 336, 479
- Heap, 527
- Heapsort algorithm, 527
- Height of a tree, 510
- Highway inspector problem, 554, 571
- Hilbert, David, 772
- Hilbert’s tenth problem, 772
- Hoare, Anthony, 86
- Hoare triple, 86
- Homeomorphic graphs, 491
- Homogeneous recurrence relation, 182
- Homomorphism, 702
- Hopkins, Brian, 571
- Horn clause, 76
- Horner’s method, 213
- HTML, 788
- Huffman code, 539–46
- encoding algorithm, 542
 - JPEG compression, 547
 - prefix code, 541
- Hypertext Markup Language, 788
- Hypothesis of an argument, 25
- Hypothetical syllogism, 33, 37
- I**
- Idempotent property, 622
- Identity element, 687
- Identity function, 392
- Identity matrix, 451
- Identity permutation, 396
- Identity property, 9, 234
- Image under a function, 383
- Immediate predecessor, 336
- Implication, 3
- Implication equivalence rule, 28
- Improper subgroup, 700
- Inclusive OR, 22
- Inconsistency inference rule, 37
- Independent events, 307
- Index of summation, 182
- Index set, 250
- Induction
- basis step, 111
 - equivalence to well-ordering, 119
 - first principle of, 110–12
 - inductive assumption, 112
 - inductive hypothesis, 112
 - inductive step, 111
 - proofs by, 112–18
 - second principle of, 118–22
 - structural induction, 164, 520
- Inductive assumption, 112
- Inductive definition, 158
- Inductive hypothesis, 112
- Inductive reasoning, 99
- Inductive step, 111
- Inference rules
- addition, 29
 - conjunction, 29
 - contraposition, 37
 - definition of, 29
 - disjunctive syllogism, 37
 - distributive, 37
 - existential generalization, 59, 62
 - existential instantiation, 59, 60
 - exportation, 37
 - hypothetical syllogism, 33, 37
 - inconsistency, 37
 - modus ponens, 29
 - modus tollens, 29
 - resolution, 76
 - self-reference, 37
 - simplification, 29
 - universal generalization, 59, 61
 - universal instantiation, 59
- Infinite sequence. *See* Sequence
- Infix notation, 244, 518
- Ingerman, P.Z., 788
- Initial point, 478
- Injective function, 389
- Inorder tree traversal, 515
- Input alphabet, 729
- Integer overflow, 344
- Intermediate statement, 43
- Internal node in a tree, 510
- Interpretation, 41
- domain of, 41
- Intersection of binary relations, 331
- Intersection of sets, 231
- Intractable problem, 417, 776
- Inverse
- of a binary relation, 350
 - of a function, 392–94
 - of an implication, 21
 - of a matrix, 452
- Inverse element, 687
- Inverse function, 393
- Inverter, 12, 639
- Invertible matrix, 452
- IP address, 257
- Irreflexive relation, 349
- ISBN, 435
- Isolated node, 481
- Isomorphic Boolean algebras, 629
- Isomorphic graphs, 484–87
- Isomorphic groups, 702–8
- Isomorphic partially ordered sets, 634
- Isomorphic structures, 626
- Isomorphic trees, 527
- Isomorphism, 626
- Iterative algorithm, 166
- J**
- Join operation, 370
- JPEG compression, 547
- K**
- Karnaugh map, 665–72
- steps in using, 669
- k -equivalent states, 739

kernel of a homomorphism, 716
 Kim, Deokhwan, 654
 Kleene, Stephen, 737, 771
 Kleene's theorem, 737
 Knight, John, 136
 Knowledge-based system, 81
 Koutis, Ioannis, 460
 Krocker, Kirk, 460
 Kruskal's algorithm, 594
 Kuratowski, Kazimierz, 491
 Kuratowski theorem, 491

L

Labeled graph, 479
 Lagrange, Joseph-Louis, 701
 Lagrange's theorem, 701
 Lamé, Gabriel, 217
 Language. *See also* Formal language
 alphabet, 782
 context-free, 790
 context-sensitive, 790
 generated by a grammar, 784
 over an alphabet, 782
 palindrome, 797
 pumping lemma, 798
 regular, 790
 type 0, 790
 vocabulary, 782
 word, 782

Lattice, 637
 complemented, 637
 distributive, 637

Leaf of a tree, 510
 Least common multiple, 152
 Least element, 338
 Least upper bound, 637
 Left cancellation law, 696
 Left child node, 510
 Left identity element, 712
 Left inverse
 of an element, 713
 of a function, 408

Length of a string, 694
 Level-order tree traversal, 606
 Lexical analyzer, 794
 Lexicographical ordering, 281, 351
 L'Hôpital's rule, 422
 Linear bounded automaton (lba), 792
 Linear combination, 144
 Linear equations
 definition of, 447
 Gaussian elimination, 453–57
 solving systems of, 453–57

Linear first-order recurrence relations,
 180–88

Linear probing, 425
 Linear recurrence relation, 182
 Linear second-order recurrence
 relations, 188–193
 Linked list, 494
 Little oh, 417
 Load factor, 427
 Logarithm function, 809–12
 Logic. *See also* Formal logic
 fuzzy, 23
 many-valued, 23
 three-valued, 23, 374
 two-valued, 23

Logic network, 479, 638–53
 AND gate, 639
 canonical product-of-sums form,
 660
 canonical sum-of-products form,
 643
 combinational network, 638–48,
 744
 conjunctive normal form, 660
 delay element, 744
 disjunctive normal form, 643
 don't-care condition, 653, 672
 field-programmable gate array,
 647
 flip-flop, 744
 full-adder, 649
 half-adder, 649
 inverter, 639
 Karnaugh map, 665–71
 minimization, 663–77
 multiplexor, 663
 NAND gate, 650
 NOR gate, 651
 OR gate, 639
 programmable logic device, 647
 Quine–McCluskey procedure,
 673–77
 sequential, 744–801

Logic programming, 73–82
 Logical connective, 2–8
 AND, 11, 75
 conjunction, 2
 disjunction, 3
 equivalence, 3
 implication, 3
 negation, 4
 NOT, 11, 75
 OR, 11, 75
 order of precedence, 6
 and programming, 11
 truth table, 3
 XOR, 22

Loop in a graph, 481
 Loop invariant, 130
 Loop rule of inference, 129–33
 definition of, 131
 Lossy compression scheme, 547
 Lower bound for sorting, 536
 Lucas sequence, 174, 201
 Lukasiewicz, J., 518

M

Machine. *See* Finite-state machine
 Maclaurin series, 411
 Main connective, 6
 Main diagonal of a matrix, 447
 Many-valued logic, 23
 Map coloring problem, 506
 Mapping. *See* Function
 Markov, A., 771
 Marley, Scott, 24
 Master theorem, 417–21
 proof of, 419–21
 statement of, 418

Mathematical induction. *See* Induction
 Mathematical structure, 619

Matrix, 446–59
 addition, 449
 augmented, 453
 Boolean, 458–59
 of coefficients, 447
 cofactor of an element, 473
 definition of, 446
 determinant of, 473
 diagonal, 463
 dimensions of, 446
 elementary row operations on, 454
 equal, 447
 Gaussian elimination, 453–57
 identity, 451
 inverse of, 452
 invertible, 452
 main diagonal, 447
 minor of an element, 473
 multiplication, 450
 operations on, 448–53
 scalar multiplication, 448
 Strassen's algorithm, 467
 subtraction, 449
 symmetric, 447
 transpose of, 463
 upper triangular, 453
 zero, 449

Maximal element, 338
 Mergesort algorithm, 215
 Metadata, 366

Meynadier, Jean-Marc, 136
 Miller, Gary, 460
 Minimal element, 338
 Minimization
 of Boolean expressions, 645–47
 of combinational network, 663–77
 of finite-state machine, 737–44
 Misailovic, Sasa, 654
 Modular arithmetic, 692
 Modular multiplicative inverse, 356
 Modular property, 633
 Modulo function, 386, 423–40
 ABA routing number, 437
 in cryptographic hashing, 434
 to decompose integers, 438
 to generate integer values, 437
 for hashing, 427
 ISBN, 435
 modular arithmetic designs, 438
 residue of x modulo n , 423
 in RSA, 431
 UPC-A, 436
 Modus ponens, 27, 29
 Modus tollens, 29
 Monoid, 689
 Morse code, 541
 Multiplexor, 663
 Multiplication modulo n , 692
 Multiplication of matrices, 450
 Multiplication principle, 252–53
 definition of, 253

N
 \mathbb{N} , 224
 n factorial, 99, 272
 NAND gate, 22, 650
 n -ary predicate, 40
 n -ary relation, 330
 n -ary relation on a set, 356
 Necessary condition, 4
 Negation connective, 4
 Negation of a statement, 5
 Network. *See* Logic network
 Netz, Reviel, 286
 Next-state function, 729
 Node, 337, 477, 478
 Nondeterministic function, 411
 Nondeterministic Turing machine, 777
 Nonrooted tree, 509
 NOR gate, 23, 651
 NOT connective, 11, 75
 NP, 777
 NP-complete problem, 778
 Null pointer, 495
 Null set, 224

Number theory, 107, 143–51
 Number theoretic function, 767

O

Odd node, 572
 Odd number, 101
 Odd permutation, 700
 One-to-one function, 389
 One-way encryption, 434
 Only, 43
 Only if, 4
 Onto function, 388
 Operation
 binary, 228–30
 unary, 230
 Optimal algorithm, 533
 OR connective, 11, 75
 OR gate, 12, 639
 Order of a group, 698
 Order of magnitude, 412–21
 big oh, 417
 big theta, 413
 definition of, 413
 little oh, 417
 master theorem, 418
 Order of precedence, 6
 Ordered pair, 228
 Ore, Oystein, 580
 Outer join, 369
 Output alphabet, 729
 Output function, 729

P

P
 definition of, 777
 and NP, 77–78
 Page, Larry, 497
 Palindrome, 163, 797
 Palindrome language, 797
 Papadimitriou, Christos, 211
 Paradox, 69, 251
 Parallel arcs, 481
 Parent node in a tree, 509
 Parity bit, 126, 733
 Parse tree, 511, 793
 Parsing
 bottom-up, 794
 top-down, 794
 Partial correctness, 132
 Partial function, 767
 Partial ordering, 336–38
 chain, 338
 definition of, 336
 greatest element, 338
 greatest lower bound, 637
 Hasse diagram, 336
 immediate predecessor in, 336
 lattice, 637
 least element, 338
 least upper bound, 637
 maximal element, 338
 minimal element, 338
 predecessor in, 336
 restriction of, 336
 successor in, 336
 total ordering, 338
 Partially ordered set, 336
 dual of, 350
 Partition, 339
 Partition refinement, 741
 Pascal, Blaise, 294
 Pascal's formula, 295
 Pascal's triangle, 294–96
 Password encryption, 433–35
 Path
 in a directed graph, 483
 in a graph, 482
 Pattern matching, 205
 Peirce arrow, 23
 Peng, Richard, 460
 Pentagonal numbers, 200
 Perfect square, 107
 Perl, 757
 Permutation function, 394–96
 cycle notation, 395
 definition of, 394
 derangement, 396
 disjoint cycles, 395
 identity permutation, 396
 Permutation group, 700
 Permutations, 272–74
 definition of, 272
 eliminating duplicates, 279
 generating, 282
 with repetitions, 279
 PERT chart, 357, 479
 Petersen graph, 492
 Phrase structure (type 0) grammar, 783
 Pig latin, 797
 Pigeonhole principle, 269
 Plaintext, 428
 Planar graph, 487–92
 Plato, 154
 Plaza, Jan, 186
 Polish notation, 518
 Polynomial, 690
 Poset, 336
 Post, E., 771
 Postcondition, 86
 Postfix notation, 244, 518

- Postorder tree traversal, 515
 - Power set, 227
 - Precondition, 86
 - Predecessor, 336
 - Predicate, 39–42
 - binary, 40
 - definition of, 39
 - n -ary, 40
 - ternary, 40
 - unary, 40, 223
 - Predicate logic, 58–69
 - completeness of, 63, 773
 - correctness of, 63, 773
 - definition of, 58
 - derivation rules for, 58–62
 - valid argument in, 58
 - well-formed formula, 41
 - Predicate well-formed formula, 35, 41
 - validity of, 48
 - Prefix code, 541
 - Prefix notation, 518
 - Preimage, 383
 - Prenex normal form, 57
 - Preorder tree traversal, 514
 - Primary key, 367
 - Prime number, 107
 - Mersenne primes, 153
 - Prim's algorithm, 587
 - Principle of inclusion and exclusion,
 - 150, 264–68, 300, 304
 - definition for n sets, 267
 - definition for three sets, 265
 - Principle of well-ordering, 119, 250, 702
 - Private key encryption, 431
 - Probability, 301–14
 - axioms, 304
 - Bayes' theorem, 308–9, 321
 - Bernoulli trial, 313
 - binomial distribution, 313–14
 - birthday problem, 318
 - conditional, 306–7
 - equally likely outcomes, 302
 - event, 302
 - expected value, 310–12
 - independent events, 307
 - linearity of expected value, 312
 - probability distribution, 305
 - random variable, 310
 - sample space, 302
 - weighted average, 310
 - Probability distribution, 305
 - Procedural language, 73
 - Product notation, 807–08
 - Production, 783
 - Program testing, 85
 - Program validation, 85
 - Program verification, 84
 - Programmable logic device (PLD), 647
 - Programming language
 - declarative, 73
 - descriptive, 73
 - procedural, 73
 - Project operation, 369
 - Prolog, 73–81
 - database, 73
 - fact, 73
 - program, 73
 - query, 73
 - recursive rule, 80
 - rule, 75
 - rule of inference, 76
 - Proof by cases, 104
 - Proof of correctness, 84–92, 129–35
 - assertions, 85–87
 - assignment rule, 87–89
 - conditional rule, 90–92
 - definition of, 85
 - loop rule, 129–33
 - Proof sequence, 27
 - Proof techniques, 98–107
 - contradiction, 104–6
 - contraposition, 103–4
 - direct proof, 101–3
 - exhaustive proof, 100–101
 - indirect proof, 104
 - induction, 110–22
 - serendipity, 107
 - Proper subgroup, 700
 - Proper subset, 225
 - Proposition, 2
 - Propositional calculus, 25
 - Propositional logic, 25–35
 - completeness of, 32
 - correctness of, 32
 - definition of, 25
 - derivation rules for, 28–33
 - valid argument in, 25–28
 - well-formed formula, 25
 - Propositional well-formed formula, 25
 - Pseudocode, 12
 - Public key encryption, 431
 - Pumping lemma, 798
 - Pushdown automaton (pda), 792
 - Pythagorean Society, 200
- Q**
- \mathbb{Q} , 224
 - Quantifier, 39–42
 - existential, 40
 - universal, 39
 - Queue, 599
 - dequeue, 599
 - enqueue, 599
 - Quicksort algorithm, 215
 - Quine–McCluskey procedure, 673–77
- R**
- \mathbb{R} , 224
 - Random variable, 310
 - Range of a function, 388
 - Rational number, 105
 - Reachability, 557–62, 602
 - Reachability matrix, 559
 - Reachable node, 483, 557
 - Recaman's sequence, 180
 - Recognition
 - by finite-state machine, 734
 - by Turing machine, 765
 - Recurrence relation, 159, 180–97
 - characteristic equation of, 190
 - closed-form solution, 180
 - constant coefficients in, 182
 - definition of, 159
 - divide-and-conquer, 193–97
 - first-order, 182
 - general solution (divide-and-conquer), 196
 - general solution (first-order), 183
 - general solution (second-order), 193, 196
 - homogeneous, 182
 - linear, 182
 - linear first-order, 182
 - second-order, 188
 - solving, 180
 - Recursion, 79–81, 158–71
 - recursive algorithm, 166–71
 - recursive definition, 79, 158
 - recursive operation, 165–66
 - recursive sequence, 158–62
 - recursive set, 162–64
 - Referential integrity, 375
 - Refinement of a partition, 354, 741
 - Reflexive relation, 332
 - Regular expression, 735
 - Regular (type 3) grammar, 790
 - Regular language, 790
 - Regular set, 736
 - Relation. *See* Binary relation
 - Relational algebra, 372
 - Relational calculus, 372
 - Relational database. *See* Database
 - Relatively prime, 146
 - Residue of x modulo n , 423
 - Resolution, 76

- Restrict operation, 369
 - Restriction of a partial ordering, 336
 - Reverse Polish notation (RPN), 518
 - Right cancellation law, 696
 - Right child node, 510
 - Right identity element, 712
 - Right inverse
 - of an element, 713
 - of a function, 408
 - Rinard, Martin, 654
 - Rivest, Ron, 431
 - Root of a tree, 509
 - RSA public-key encryption
 - algorithm, 431
 - Rule-based system, 81
 - Russell, Bertrand, 251
 - Russell's paradox, 251
- S**
- Sample space, 302
 - Satisfiability problem, 778
 - Scalar, 448
 - Scalar multiplication, 448
 - Scanner, 794
 - Scope of a quantifier, 41
 - Searching
 - binary search, 169
 - binary tree search, 352, 534
 - decision tree for, 529–35
 - by hash function, 425
 - lower bound for, 532–33
 - sequential search, 204, 415
 - Second principle of induction, 118–22, 144, 160, 251
 - definition of, 118
 - Second-order recurrence relation, 188
 - Seemann, Glenn, 749
 - Selection sort algorithm, 169, 215
 - Self-complementary graph, 506
 - Self-reference inference rule, 37
 - Semigroup, 689
 - Semigroup of transformations, 693
 - Sequence
 - definition of, 158
 - Fibonacci, 159
 - recursive, 158–62
 - Sequential network, 744–48
 - definition of, 745
 - Sequential search algorithm, 204
 - Sequential search analysis, 205, 207, 216
 - Serendipity, 106–7
 - Sets, 222–39
 - binary operation on, 229
 - binary relation on, 329
 - Cantor's diagonalization
 - method, 237
 - cardinality of, 236–39, 401
 - Cartesian product (cross product)
 - of, 233
 - characteristic function of, 406
 - closure of, 229, 686
 - complement of, 232
 - countable, 236
 - denumerable, 236, 401
 - difference of, 232
 - disjoint, 232
 - empty, 224
 - equality of, 223
 - equivalent, 401–402
 - finite, 223
 - fuzzy, 250
 - identities, 233–36
 - intersection of, 231
 - membership, 222
 - n -ary relation on, 330
 - notation, 222–24
 - null, 224
 - operations on, 230–33
 - partial ordering on, 336
 - partially ordered, 336, 634
 - partition of, 339
 - permutations of, 394
 - power set of, 227
 - proper subset of, 225
 - recognition by finite-state
 - machine, 734
 - recognition by Turing machine, 765
 - subset of, 224
 - symmetric difference of, 248
 - unary operation on, 230
 - uncountable, 236
 - union, 231
 - universal, 231
- Shamir, Adi, 431
- Shannon, Claude, 638
- Sharan, Roded, 497
- Sheffer stroke, 22
- Sieve of Eratosthenes, 154
- Simple closed polygon, 127, 263
 - triangulation of, 263
- Simple graph, 481
- Simple statement, 2
- Simple substitution cipher, 429
- Simplex method, 417
- Simplification inference rule, 29
- Six-color theorem, 508
- Sloane, Neil, 179
- Smorynski function, 407
- Smullyan, Raymond, 25, 771
- Sorting
 - bubble sort, 214
 - decision tree for, 535–36
 - heapsort, 527
 - lower bound for, 536
 - mergesort, 215
 - quicksort, 215
 - selection sort, 169, 215
- Spanning tree, 587, 601
- Sparse adjacency
 - matrix, 494
- SQL, 372
- Square numbers, 200
- Stack, 167, 409
 - pop instruction, 409
 - push instruction, 409
 - stack overflow, 167
- Standard sets, 224
- Start symbol, 783
- State graph, 730
- State table, 730
- Statement, 2
- Statement letter, 2
- Statement logic, 25
- Stewart, Ian, 174
- Stirling numbers, 355
- Stirling's triangle, 355
- Strassen's algorithm, 467
- String, 694
 - binary, 163
 - concatenation, 163
 - empty, 163
 - palindrome, 163
 - pattern matching, 205
- Structural induction, 164, 520
- Subgraph, 482
- Subgroup, 699
- Subset, 224
- Subtraction of matrices, 449
- Successor, 336
- Sudoku, 154
- Sufficient condition, 4
- Summation notation, 182, 805–07
- Sun-Tsu, 445
- Surrogate key, 368
- Swift, Jonathan, 550
- Symbol, 694
- Symbol table, 441
- Symbolic logic, 68
- Symmetric difference of sets, 248
- Symmetric encryption, 431
- Symmetric matrix, 447
- Symmetric relation, 332
- Syntax analyzer, 794
- Syntax rule, 6

T

Tape alphabet, 762
 Tautological equivalence, 9
 associative property, 9
 commutative property, 9
 complement property, 9
 distributive property, 9
 identity property, 9
 Tautology, 8–10
 definition of, 8
 Temporary hypothesis, 64
 Terminal, 783
 Terminal point, 478
 Ternary predicate, 40
 Ternary tree, 537
 Theorem, 98, 773
 Three-valued logic, 23, 374
 Tiling problem, 117
 Token, 163
 Top down parsing, 794
 Topological sorting, 356–61, 602
 definition of, 359
 Total function, 767
 Total ordering, 338
 Towers of Hanoi, 179, 199
 Transitive relation, 332
 Transpose of a matrix, 463
 Transposition, 700
 Traveling salesman problem, 554, 577
 Tree, 353, 509–21
 applications of, 511–12
 binary, 510
 binary search, 534
 B-tree, 526
 child node, 509
 complete binary, 510
 decision tree, 257–58, 529–36
 definition of, 509
 depth of, 510
 depth of a node, 510
 forest, 510
 free, 509
 full binary, 510
 height of, 510
 inorder traversal, 515
 internal node in, 510
 isomorphic, 527
 leaf of, 510
 left child in, 510
 left child-right child
 representation, 513
 level-order traversal, 606
 nonrooted, 509
 parent node, 509

 postorder traversal, 515
 preorder traversal, 514
 right child in, 510
 root, 509
 ternary, 537
 traversal, 514
 Tree arc, 608
 Tree traversal, 514
 Triangular numbers, 200
 Triangulation of a convex polygon, 263
 Truth function, 640
 Truth table, 3
 Tuple, 366
 Turing, Alan M., 428, 760, 771
 Turing computable function, 768
 Turing machine, 759–78
 acceptance by, 765
 Church–Turing thesis, 770
 decision problem, 771–76
 definition of, 762
 final state, 765
 as function computer, 767–69
 halting problem, 774
 nondeterministic, 777
 NP, 777
 NP-complete problem, 778
 P, 777
 recognition by, 765
 as set recognizer, 764–67
 tape alphabet, 762
 unsolvability of halting problem,
 774
 Two's complement, 660
 Two-valued logic, 23
 Type 0 language, 790

U

Unary connective, 3
 Unary operation, 230
 Unary predicate, 40, 223
 Uncomputability, 771–76
 Uncomputable problem, 772
 Uncountable set, 236
 Undecidable problem, 772
 Union of binary relations, 331
 Union of sets, 231
 Universal bound property, 624
 Universal generalization, 59, 61
 Universal instantiation, 59
 Universal quantifier, 39
 Universal set, 231
 Universe of discourse, 231
 Unreachable state, 777

Unsolvability of the halting
 problem, 774
 Unsolvable problem, 772
 UPC-A (Universal Product Code), 436
 Upper bound, 210
 Upper triangular matrix, 453

V

Valid argument
 in predicate logic, 58
 in propositional logic, 25–28
 Valid predicate wff, 48
 Vandermonde's identity, 293
 Vector, 448
 dot product of, 463
 magnitude of, 463
 Venn, John, 231
 Venn diagram, 231
 Vertex, 337, 477, 478
 Vocabulary, 782

W

Warshall's algorithm, 562–65
 analysis of, 565
 Weighted average, 310
 Weighted graph, 479
 Well-balanced parentheses, 780
 Well-defined binary operation, 229
 Well-formed formula (wff), 6
 comparison of propositional and
 predicate, 48
 equivalent, 8
 predicate, 35, 41
 propositional, 25
 Well-ordering principle, 119, 250, 702
 Wff. *See* Well-formed formula
 Wiles, Andrew, 143
 Wilson, Robin, 571
 Word, 694, 782

X

XML, 788
 XOR, 22

Y

Yin, Xiang, 136

Z

\mathbb{Z} , 224
 Zero matrix, 449
 Zero-degree polynomial, 691