Discrete mathematical structures dr dsc pdf free download

The return timelines for seller-fulfilled items sold on Amazon.in are equivalent to the return timelines mentioned above for items fulfilled by Amazon. If you've received a seller-fulfilled product in a condition that is damaged, defective or different from its description on the product detail page on Amazon.in, returns are subject to the seller's approval of the return. If you do not receive a response from the seller for your return request within two business days, you can submit an A-to-Z Guarantee claim. Learn more about returning seller fulfilled items. Note: For seller fulfilled items from Books, Movies & TV Shows categories, the sellers need to be informed of the damage/ defect within 14 days of delivery. For seller-fulfilled items from Fine Art category, the sellers need to be informed of the damage / defect within 10 days of delivery. These items are not eligible for self-return. The seller will arrange the return pick up for these items. For seller-fulfilled items from Sports collectibles and Entertainment collectibles categories, the sellers need to be informed of the damage / defect within 10 days of delivery. • Provide theoretical foundations of computer science to perceive other courses in the programme. • Illustrate applications of discrete structures: logic, relations, functions, set theory and counting. • Describe different mathematical proof techniques, • Illustrate the use of graph theory in computer science Fundamentals of Logic: Basic Connectives and Truth Tables, Logic Equivalence – The Laws of Logic, Logical Implication – Rules of Inference, Fundamentals of Logic contd.: The Use of Quantifiers, Quantifiers, Definitions and the Proofs of Theorems. Text book 1: Chapter2Properties of the Integers: The Well Ordering Principle – Mathematical Induction,Fundamental Principles of Counting: The Rules of Sum and Product, Permutations, Combinations – The Binomial Theorem, Combinations with Repetition.Text book 1: Chapter4 – 4.1, Chapter1 Relations and Functions: Cartesian Products and Relations, Functions – Plain and One-to-One, Onto Functions. The Pigeon-hole Principle, Function Composition and Inverse Functions. Relations: Properties of Relations, Computer Recognition – Zero-One Matrices and Directed Graphs, Partial Orders – Hasse Diagrams, Equivalence Relations and Partitions. Text book 1: Chapter5, Chapter7 – 7.1 to 7.4The Principle of Inclusion and Exclusion: The Principle of Inclusion and Exclusion, Generalizations of the Principle, Derangements – Nothing is in its Right Place, Rook Polynomials.Recurrence Relations: First Order Linear Recurrence Relation, The Second Order Linear Homogeneous Recurrence Relation with Constant Coefficients.Text book 1: Chapter8 – 8.1 to 8.4, Chapter10 – 10.1, 10.2 Introduction to Graph Theory: Definitions and Examples, Sub graphs, Complements, and Graph Isomorphism,Trees: Definitions, Properties, and Examples, Routed Trees, Trees and Sorting, Weighted Trees and Prefix CodesText book 1: Chapter11 – 11.1 to 11.2 Chapter12 – 12.1 to 12.4 • Use propositional and predicate logic in knowledge representation and truth verification. • Demonstrate the application of discrete structures in different fields of computer science. • Solve problems using recurrence relations and generating functions. • Application of different mathematical proofs techniques in proving theorems in the courses. • Compare graphs, trees and their applications. • The question paper will have ten questions. • Each full Question consisting of 20 marks • There will be 2 full questions (with a maximum of four sub questions) from each module. • Each full question will have sub questions covering all the topics under a module. • The students will have to answer 5 full questions, selecting one full question from each module. 1. Ralph P. Grimaldi: Discrete and Combinatorial Mathematics, 5th Edition, Pearson Education. 2004. 1. Basavaraj S Anami and Venakanna S Madalli: Discrete Mathematics – A Concept based approach, Universities Press, 2016 2. Kenneth H. Rosen: Discrete Mathematics and its Applications, 6th Edition, McGraw Hill, 2007. 3. Jayant Ganguly: A Treatise on Discrete Mathematical Structures, Sanguine-Pearson, 2010. 4. D.S. Malik and M.K. Sen: Discrete Mathematical Structures: Theory and Applications, Thomson, 2004. 5. Thomas Koshy: Discrete Mathematics with Applications, Elsevier, 2005, Reprint 2008. Discrete Mathematical Structures Dr. DSC by Dr. DSC exam reference 0 recommended No reviews yet Discrete Mathematical Structures by Dr. DSC is suggested as exam reference for studying the following subjects: Discrete Mathematical Structures in Computer Science Semester 3 (Visveswaraya Technological University) ADDED BY Stupidsid WRITE BOOK REVIEW Share your opinion about this book... Discrete Mathematical Structures This page intentionally left blank Discrete Mathematical Structures Sixth Edition Bernard Kolman Drexel University Robert C. Busby Drexel University Sharon Cutler Ross Georgia Perimeter College Upper Saddle River, NJ 07458 Library of Congress Cataloging-in-Publication Data Kolman, Bernard Discrete mathematical structures / Bernard Kolman, Robert C. Busby, Sharon Cutler Ross. — 6th ed. p. cm. Includes index. 1. Computer Science—Mathematics. I. Busby, Robert C. II. Ross, Sharon Cutler III. Title. QA76.9.M35K64 2009 511.6—dc22 2008010264 President: Greg Tobin Editor-in-Chief: Deirdre Lynch Senior Acquisitions Editor: William Hoffman Associate Editor: Caroline Celano Project Manager: Kristy S. Mosch Senior Managing Editor: Linda Mihatov Behrens Operations Specialist: Lisa McDowell Senior Operations Supervisor: Alan Fischer Marketing Manager: Katie Winter Marketing Assistant: Jon Connelly Art Director/Cover Designer: Heather Scott Interior Design/Illustrations/Composition: Dennis Kletzing AV Project Manager: Thomas Benfatti c Cover Photo: Swinging (Schaukeln), 1925, Kandinsky, Wassily (1866-1944) ARS NY Manager, Cover Visual Research and Permissions: Karen Sanatar Director, Image Resource Center: Melinda Patelli Manager, Rights and Permissions: Zina Arabia Manager, Visual Research: Beth Brenzel Image Permissions Coordinator: Debbie Hewitson Photo Researcher: Beth Anderson Art Studio: Laserwords c 2009, 2004, 2000, 1996, 1987, 1984 Pearson Prentice Hall Pearson Education, Inc. Upper Saddle River, NJ 07458 All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher. Pearson Prentice HallTM is a trademark of Pearson Education, Inc. Printed in the United States of America 10 9 8 7 6 5 4 3 2 1 ISBN-13: 978-0-13-229751-6 ISBN-10: 0-13-229751-5 Pearson Education Ltd., London Pearson Education Singapore, Pte. Ltd. Pearson Education Canada, Inc. Pearson Education—Japan Pearson Education Australia PTY, Limited Pearson Education North Asia, Ltd., Hong Kong Pearson Educación de Mexico, S.A. de C.V. Pearson Education Malaysia, Pte. Ltd. Pearson Education Upper Saddle River, New Jersey To the memory of Lillie and to Judith B.K. To my wife, Patricia, and our sons, Robert and Scott R.C.B. To Bill and bill S.C.R. This page intentionally left blank Contents Preface xi A Word to Students

... and B, then A and B must (we assume) be sets of real numbers, not matrices, electronic circuits, or rhesus monkeys. In most problems, a universal set will be apparent from the setting of the problem. In Venn diagrams, the universal set U will be denoted by a rectangle, while sets within U will be denoted by circles as shown in Figure 1.2. 4 Chapter 1 Fundamentals A set A is called finite if it has n distinct elements, where n ∈ N. In this case, n is called the cardinality of A and is denoted by |A|. Thus, the sets of Examples 1, 2, 4, 5, and 6 are finite. A set that is not finite is called infinite. The sets introduced in Example 3 (except ∅) are infinite sets. If A is a set, then the set of all subsets of A is called the power set of A and is denoted by P(A). (Be sure to distinguish between P(A), a statement about A, and P(A), the power set of A.) Example 11 Let A = {1, 2, 3}. Then P(A) consists of the following subsets of A: { }, {1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, and {1, 2, 3} (or A). In a later section, we will count the number of subsets that a set can have. ◆ 1.1 Exercises 1. Let A = {1, 2, 4, a, b, c}. Identify each of the following as true or false. (b) 3 ∈ A (c) c ∈ / A (a) 2 ∈ A (e) { } ∈ / A (f) A ∈ A (d) ∅ ∈ A 2. Let A = {x | x is a real number and x < 6}. Identify each of the following as true or false. (b) 6 ∈ A (c) 5 ∈ / A (a) 3 ∈ A (d) −8 ∈ A (f) 3.4 ∈ / A (d) 8 ∈ 3. In each part, give the set of letters in each word by listing the elements of the set. (b) BOOK (a) AARDVARK (c) MISSISSIPPI 4. Give the set by listing its elements. (a) The set of all positive integers that are less than ten. (b) {x | x ∈ Z and x 2 < 12} 5. Let A = {1, {2, 3}, 4}. Identify each of the following as true or false. (b) {1, 4} ⊆ A (c) {2, 3} ⊆ A (a) 3 ∈ A (e) {4} ∈ A (f) {1, 2, 3} ⊆ A (d) {2, 3} ∈ A In Exercises 6 through 9, write the set in the form {x | P(x)}, where P(x) is a property that describes the elements of the set. 6. {2, 4, 6, 8, 10} 7. {a, e, i, o, u} 8. {1, 8, 27, 64, 125} 9. {−2, −1, 0, 1, 2} 10. Let A = {1, 2, 3, 4, 5}. Which of the following sets are equal to A? (a) {4, 1, 2, 3, 5} (b) {2, 3, 4} (c) {1, 2, 3, 4, 5, 6} (d) {x | x is an integer and x 2 ≤ 25} (e) {x | x is a positive integer and x ≤ 5} (f) {x | x is a positive rational number and x ≤ 5} 11. Which of the following sets are the empty set? (a) {x | x is a real number and x 2 − 1 = 0} (b) {x | x is a real number and x 2 + 1 = 0} (c) {x | x is a real number and x 2 = −9} (d) {x | x is a real number and x = 2x + 1} (e) {x | x is a real number and x = x + 1} 12. List all the subsets of {a, b}. 13. List all the subsets of {JAVA, PASCAL, C++}. 14. List all the subsets of { }. 15. Let A = {1, 2, 5, 8, 11}. Identify each of the following as true or false. (b) {8, 1} ∈ A (a) {5, 1} ⊆ A (c) {1, 8, 2, 11, 5} A (d) ∅ ⊆ A (e) {1, 6} A (f) {2} ⊆ A (g) {3} ∈ / A (h) A ⊆ {11, 2, 5, 1, 8, 4} 16. Let A = {x | x is an integer and x 2 < 16}. Identify each of the following as true or false. (b) {−3, −2, −1} ⊆ A (a) {0, 1, 2, 3} ⊆ A (c) { } ⊆ A (d) {x | x is an integer and |x| < 4} ⊆ A (e) A ⊆ {−3, −2, −1, 0, 1, 2, 3} 17. Let A = {1}, B = {1, a, 2, b, c}, C = {b, c}, D = {a, b}, and E = {1, a, 2, b, c, d}. For each part, replace the symbol 2 with either ⊆ or to give a true statement. (b) ∅ 2 A (c) B 2 C (a) A 2 B (d) C 2 E (e) D 2 C (f) B 2 E In Exercises 18 through 20, find the set of smallest cardinality that contains the given sets as subsets. 18. {a, b, c}, {a, d, e, f }, {b, c, e, g} 19. {1, 2}, {1, 3}, ∅ 20. {2, 4, 6, . . . , 20}, {3, 6, 9, . . . , 21} 21. Is it possible to have two different (appropriate) universal sets for a collection of sets? Would having different universal sets create any problems? Explain. 22. Use the Venn diagram in Figure 1.3 to identify each of the following as true or false. (b) B ⊆ A (c) C ⊆ B (a) A ⊆ B (d) x ∈ B (e) x ∈ A (f) y ∈ B 1.2 Operations on Sets U y A B x 5 25. Complete the following statement. A generic Venn diagram for three sets has regions. Describe them in words. 26. (a) If A = {3, 7}, find P(A). (c) What is |P(A)|? (b) What is |A|? C 27. If P(B) = {{ }, {m}, {n}, {m, n}}, then find B. 28. (a) If A = {3, 7, 2}, find P(A). (c) What is |P(A)|? (b) What is |A|? Figure 1.3 23. Use the Venn diagram in Figure 1.4 to identify each of the following as true or false. (b) A ⊆ C (c) C ⊆ B (a) B ⊆ A (e) t ∈ A (f) w ∈ B (d) w ∈ A U A B t In Exercises 30 through 32, draw a Venn diagram that represents these relationships. 30. A ⊆ B, A ⊆ C, B C, and C B 31. x ∈ A, x ∈ B, x ∈ / C, y ∈ B, y ∈ C, and y ∈ / A 32. A ⊆ B, x ∈ / A, x ∈ B, A C, y ∈ B, y ∈ C 33. Describe all the subset relationships that hold for the sets given in Example 3. w 34. Show that if A ⊆ B and B ⊆ C, then A ⊆ C. 35. The statement about sets in Exercise 34 can be restated as “Any subset of is also a subset of any set that contains .” C Figure 1.4 24. (a) Complete the following statement. A generic Venn diagram for a single set has regions. Describe them in words. (b) Complete the following statement. A generic Venn diagram for two sets has regions. Describe them in words. 1.2 29. If P(B) = {{a}, { }, {c}, {b, c}, {a, b}, . . . } and |P(B)| = 8, then B = . 36. Suppose we know that set A has n subsets, S1 , S2 , . . . , Sn . If set B consists of the elements of A and one more element so |B| = |A| + 1, show that B must have 2n subsets. 37. Compare the results of Exercises 12, 13, 26, and 28 and complete the following: Any set with two elements has subsets. Any set with three elements has subsets. Operations on Sets In this section we will discuss several operations that will combine given sets to yield new sets. These operations, which are analogous to the familiar operations on the real numbers, will play a key role in the many applications and ideas that follow. If A and B are sets, we define their union as the set consisting of all elements that belong to A or B and denote it by A ∪ B. Thus A ∪ B = {x | x ∈ A or x ∈ B}. Observe that x ∈ A ∪ B if x ∈ A or x ∈ B or x belongs to both A and B. Example 1 Let A = {a, b, c, e, f } and B = {b, d, r, s}. Find A ∪ B. Solution Since A ∪ B consists of all the elements that belong to either A or B, A ∪ B = ◆ {a, b, c, d, e, f, r, s}. We can illustrate the union of two sets with a Venn diagram as follows. If A and B are the sets in Figure 1.5(a), then A ∪ B is the set represented by the shaded region in Figure 1.5(b). 6 Chapter 1 Fundamentals A A B B U U (b) A ∪ B (a) Figure 1.5 If A and B are sets, we define their intersection as the set consisting of all elements that belong to both A and B and denote it by A ∩ B. Thus A ∩ B = {x | x ∈ A and x ∈ B}. Let A = {a, b, c, e, f }, B = {b, e, f, r, s}, and C = {a, t, u, v}. Find A ∩ B, A ∩ C, and B ∩ C. Example 2 Solution The elements b, e, and f are the only ones that belong to both A and B, so A∩ B = {b, e, f }. Similarly, A ∩ C = {a}. There are no elements that belong to both B ◆ and C, so B ∩ C = { }. Two sets that have no common elements, such as B and C in Example 2, are called disjoint sets. We can illustrate the intersection of two sets by a Venn diagram as follows. If A and B are the sets given in Figure 1.6(a), then A ∩ B is the set represented by the shaded region in Figure 1.6(b). Figure 1.7 illustrates a Venn diagram for two disjoint sets. A A U B B A U (a) Figure 1.6 B U (b) A ∩ B Figure 1.7 The operations of union and intersection can be defined for three or more sets in an obvious manner: A ∪ B ∪ C = {x | x ∈ A or x ∈ B or x ∈ C} and A ∩ B ∩ C = {x | x ∈ A and x ∈ B and x ∈ C}. The shaded region in Figure 1.8(b) is the union of the sets A, B, and C shown in Figure 1.8(a), and the shaded region in Figure 1.8(c) is the intersection of the sets A, B, and C. Note that Figure 1.8(a) says nothing about possible relationships between the sets, but allows for all possible relationships. In general, if A1 , A2 , . . . , An are subsets of U , then A1 ∪ A2 ∪ · · · ∪ An will be denoted by 1.2 Operations on Sets U U U B B A B A A C C C (c) A ∩ B ∩ C (b) A ∪ B ∪ C (a) 7 Figure 1.8 n Ak , read as, “the union from 1 to n of A sub k,” and A1 ∩ A2 ∩ · · · ∩ An will be n Ak , read as, “the intersection from 1 to n of A sub k.” denoted by k=1 k=1 Example 3 Let A = {1, 2, 3, 4, 5, 7}, B = {1, 3, 8, 9}, and C = {1, 3, 6, 8}. Then A ∩ B ∩ C ◆ is the set of elements that belong to A, B, and C. Thus A ∩ B ∩ C = {1, 3}. If A and B are two sets, we define the complement of B with respect to A as the set of all elements that belong to A but not to B, and we denote it by A − B. Thus A − B = {x | x ∈ A and x ∈ / B}. Example 4 Let A = {a, b, c} and B = {b, c, d, e}. Then A − B = {a} and B − A = {d, e}. ◆ If A and B are the sets in Figure 1.9(a), then A − B and B − A are represented by the shaded regions in Figures 1.9(b) and 1.9(c), respectively. U A B (a) U A B (b) A − B U A B (c) B − A Figure 1.9 If U is a universal set containing A, then U − A is called the complement of A and is denoted by A. Thus A = {x | x ∈ / A}. Example 5 Let A = {x | x is an integer and x ≤ 4} and U = Z. Then A = {x | x is an integer ◆ and x > 4}. If A is the set in Figure 1.10, its complement is the shaded region in that figure. If A and B are two sets, we define their symmetric difference as the set of all elements that belong to A or to B, but not to both A and B, and we denote it by A ⊕ B. Thus A ⊕ B = {x | (x ∈ A and x ∈ / B) or (x ∈ B and x ∈ / A)}. Example 6 Let A = {a, b, c, d} and B = {a, c, e, f, g}. Then A ⊕ B = {b, d, e, f, g}. ◆ 8 Chapter 1 Fundamentals U A Figure 1.10 If A and B are as indicated in Figure 1.11(a), their symmetric difference is the shaded region shown in Figure 1.11(b). It is easy to see that A ⊕ B = (A − B) ∪ (B − A). U A B (a) U A B (b) A ∩ B Figure 1.11 Algebraic Properties of Set Operations The operations on sets that we have just defined satisfy many algebraic properties, some of which resemble the algebraic properties satisfied by the real numbers and their operations. All the principal properties listed here can be proved using the definitions given and the rules of logic. We shall prove only one of the properties and leave proofs of the remaining ones as exercises for the reader. Proofs are fundamental to mathematics. We discuss proof techniques in Chapter 2, but in this chapter some proofs are given as examples for later work. Some simple proofs are required in the exercises. Venn diagrams are often useful to suggest or justify the method of proof. THEOREM 1 The operations defined on sets satisfy the following properties: Commutative Properties 1. A ∪ B = B ∪ A 2. A ∩ B = B ∩ A Associative Properties 3. A ∪ (B ∪ C) = (A ∪ B) ∪ C 4. A ∩ (B ∩ C) = (A ∩ B) ∩ C Distributive Properties 5. A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C) 6. A ∪ (B ∩ C) = (A ∪ B) ∩ (A ∪ C) 1.2 Operations on Sets 9 Idempotent Properties 7. A ∪ A = A 8. A ∩ A = A Properties of the Complement 9. 10. 11. 12. 13. 14. 15. (A) = A A∪ A=U A∩ A=∅ ∅=U U =∅ A∪B = A∩B A∩B = A∪B Properties 14 and 15 are known as De Morgan’s laws. Properties of a Universal Set 16. A ∪ U = U 17. A ∩ U = A Properties of the Empty Set 18. A ∪ ∅ = A or A ∪ { } = A 19. A ∩ ∅ = ∅ or A ∩ { } = { } Proof We prove Property 14 here and leave proofs of the remaining properties as exercises for the reader. A common style of proof for statements about sets is to choose an element in one of the sets and see what we know about it. Suppose that / A ∪ B, so x ∈ / A and x ∈ / B. (Why?) This x ∈ A ∪ B. Then we know that x ∈ means x ∈ A ∩ B (why?), so each element of A ∪ B belongs to A ∩ B. Thus A ∪ B ⊆ A ∩ B. Conversely, suppose that x ∈ A ∩ B. Then x ∈ / A and x ∈ / B (why?), so x ∈ / A ∪ B, which means that x ∈ A ∪ B. Thus each element of A ∩ B also belongs to A ∪ B, and A ∩ B ⊆ A ∪ B. Now we see that A ∪ B = A ∩ B. ■ A∩B A The Addition Principle B Figure 1.12 THEOREM 2 Example 7 Suppose now that A and B are finite subsets of a universal set U . It is frequently useful to have a formula for |A ∪ B|, the cardinality of the union. If A and B are disjoint sets, that is, if A ∩ B = ∅, then each element of A ∪ B appears in either A or B, but not in both; therefore, |A ∪ B| = |A| + |B|. If A and B overlap, as shown in Figure 1.12, then elements in A ∩ B belong to both sets, and the sum |A| + |B| counts these elements twice. To correct for this double counting, we subtract |A ∩ B|. Thus we have the following theorem, sometimes called the addition principle. Because of Figure 1.12, this is also called the inclusion-exclusion principle. If A and B are finite sets, then |A ∪ B| = |A| + |B| − |A ∩ B|. ■ Let A = {a, b, c, d, e} and B = {c, e, f, h, k, m}. Verify Theorem 2. Solution We have A ∪ B = {a, b, c, d, e, f, h, k, m} and A ∩ B = {c, e}. Also, |A| = 5, |B| = 6, |A ∪ B| = 9, and |A ∩ B| = 2. Then |A| + |B| − |A ∩ B| = 5 + 6 − 2 ◆ or 9 and Theorem 2 is verified. Chapter 1 Fundamentals 10 A∩B B A A∩B∩C A∩C If A and B are disjoint sets, A ∩ B = ∅ and |A ∩ B| = 0, so the formula in Theorem 2 now becomes |A ∪ B| = |A| + |B|. This special case can be stated in a way that is useful in a variety of counting situations. If a task T1 can be performed in exactly n ways, and a different task T2 can be performed in exactly m ways, then the number of ways of performing task T1 or task T2 is n + m. C B∩C Figure 1.13 THEOREM 3 The situation for three sets is shown in Figure 1.13. We state the three-set addition principle without discussion. Let A, B, and C be finite sets. Then |A ∪ B ∪ C| = |A| + |B| + |C| − |A ∩ B| − |B ∩ C| − |A ∩ C| + |A ∩ B ∩ C|. ■ Theorem 3 can be generalized for more than three sets. This is done in Exercises 50 and 51. Example 8 Let A = {a, b, c, d, e}, B = {a, b, e, g, h}, and C = {b, d, e, g, h, k, m, n}. Verify Theorem 3. Solution We have A ∪ B ∪ C = {a, b, c, d, e, g, h, k, m, n}, A ∩ B = {a, b, e}, A ∩ C = {b, d, e}, B ∩C = {b, e, g, h}, and A ∩ B ∩C = {b, e}, so |A| = 5, |B| = 5, |C| = 8, |A ∪ B ∪ C| = 10, |A ∩ B| = 3, |A ∩ C| = 3, |B ∩ C| = 4, and |A ∩ B ∩ C| = 2. Thus |A|+|B|+|C|−|A∩B|−|B∩C|−|A∩C|+|A∩B∩C| = 5+5+8−3−3−4+2 or 10, and Theorem 3 is verified. ◆ Example 9 A computer company wants to hire 25 programmers to handle systems programming jobs and 40 programmers for applications programming. Of those hired, ten will be expected to perform jobs of both types. How many programmers must be hired? Solution Let A be the set of systems programmers hired and B be the set of applications programmers hired. The company must have |A| = 25 and |B| = 40, and |A ∩ B| = 10. The number of programmers that must be hired is |A ∪ B|, but |A ∪ B| = |A| + |B| − |A ∩ B|. So the company must hire 25 + 40 − 10 or 55 programmers. ◆ Example 10 A survey has been taken on methods of commuter travel. Each respondent was asked to check BUS, TRAIN, or AUTOMOBILE as a major method of traveling to work. More than one answer was permitted. The results reported were as follows: BUS, 30 people; TRAIN, 35 people; AUTOMOBILE, 100 people; BUS and TRAIN, 15 people; BUS and AUTOMOBILE, 15 people; TRAIN and AUTOMOBILE, 20 people; and all three methods, 5 people. How many people completed a survey form? Solution Let B, T , and A be the sets of people who checked BUS, TRAIN, and AUTOMOBILE, respectively. We know |B| = 30, |T | = 35, |A| = 100, |B ∩ T | = 15, |B ∩ A| = 15, |T ∩ A| = 20, and |B ∩ T ∩ A| = 5. So |B| + |T | + |A| − |B ∩ T | − |B ∩ A| − |T ∩ A| + |B ∩ T ∩ A| = 30 + 35 + 100 − 15 − 15 − 20 + 5 or ◆ 120 is |A ∪ B ∪ C|, the number of people who responded. 1.2 Operations on Sets 11 1.2 Exercises In Exercises 1 through 4, let U = {a, b, c, d, e, f, g, h, k}, A = {a, b, c, g}, B = {d, e, f, g}, C = {a, c, f }, and D = { f, h, k}. 1. Compute (b) B ∪ C (c) A ∩ C (a) A ∪ B (e) (A ∪ B) − C (f) A − B (d) B ∩ D (g) A (h) A ⊕ B (i) A ⊕ C (j) (A ∩ B) − C 2. Compute (b) B ∪ D (c) C ∩ D (a) A ∪ D (e) (A ∪ B) − (C ∪ B) (d) A ∩ D (h) C − B (g) B (f) B − C (j) (A ∩ B) − (B ∩ D) (i) C ⊕ D 3. Compute (b) A ∩ B ∩ C (a) A ∪ B ∪ C (d) (A ∪ B) ∩ C (c) A ∩ (B ∪ C) (f) A ∩ B (e) A ∪ B 4. Compute (b) A ∪ U (c) B ∪ B (a) A ∪ ∅ (f) C ∩ D (e) C ∪ D (d) C ∩ { } In Exercises 5 through 8, let U = {1, 2, 3, 4, 5, 6, 7, 8, 9}, A = {1, 2, 4, 6, 8}, B = {2, 4, 5, 9}, C = {x | x is a positive integer and x 2 ≤ 16}, and D = {7, 8}. 5. Compute (b) A ∪ B (e) (d) B ∪ C (g) B ∩ C (h) 6. Compute (b) (a) A − B (e) (d) C (g) C ⊕ D (h) 7. Compute (a) A ∪ B ∪ C (c) A ∩ (B ∪ C) (e) A ∪ B 8. Compute (a) B ∪ C ∪ D (c) A ∪ A (e) A ∪ A A∪C A∩C C∩D (c) A ∪ D (f) A ∩ D B−A A B ⊕C (c) C − D (f) A ⊕ B (b) B (e) U (b) B ∪ C (c) A ∪ A (e) A ⊕ B (f) B ⊕ C (d) C ∩ C 11. Let U be the set of real numbers, A = {x | x is a solution of x 2 − 1 = 0}, and B = {−1, 4}. Compute (b) B (c) A ∪ B (d) A ∩ B (a) A In Exercises 12 and 13, refer to Figure 1.14. U A B w z y x u v C Figure 1.14 12. Identify the following as true or false. (b) x ∈ B ∪ C (a) y ∈ A ∩ B (d) u ∈ /C (c) w ∈ B ∩ C 13. Identify the following as true or false. (b) y ∈ A ∪ B ∪ C (a) x ∈ A ∩ B ∩ C (d) v ∈ B ∩ C (c) z ∈ A ∩ C 14. Describe the shaded region shown in Figure 1.15 using unions and intersections of the sets A, B, and C. (Several descriptions are possible.) A B C Figure 1.15 (b) A ∩ B ∩ C (d) (A ∪ B) ∩ D (f) A ∩ B (b) B ∩ C ∩ D (d) A ∩ A (f) A ∩ (C ∪ D) In Exercises 9 and 10, let U = {a, b, c, d, e, f, g, h}, A = {a, c, f, g}, B = {a, e}, and C = {b, h}. 9. Compute (a) A (d) A ∩ B 10. Compute (a) A ∩ B (c) A ∪ B (f) A − B 15. Let A, B, and C be finite sets with |A| = 6, |B| = 8, |C| = 6, |A ∪ B ∪ C| = 11, |A ∩ B| = 3, |A ∩ C| = 2, and |B ∩ C| = 5. Find |A ∩ B ∩ C|. In Exercises 16 through 18, verify Theorem 2 for the given sets. 16. (a) A = {1, 2, 3, 4}, B = {2, 3, 5, 6, 8} (b) A = {1, 2, 3, 4}, B = {5, 6, 7, 8, 9} 17. (a) A = {a, b, c, d, e, f }, B = {a, c, f, g, h, i, r } (b) A = {a, b, c, d, e}, B = { f, g, r, s, t, u} 18. (a) A = {x | x is a positive integer < 8}, B = {x | x is an integer such that 2 ≤ x ≤ 5} (b) A = {x | x is a positive integer and x 2 ≤ 16}, B = {x | x is a negative integer and x 2 ≤ 25} 12 Chapter 1 Fundamentals 19. If A and B are disjoint sets such that |A ∪ B| = |A|, what must be true about B? 20. Write Property 14 of Theorem 1 in ordinary English. 21. Write Property 15 of Theorem 1 in ordinary English. In Exercises 22 through 24, verify Theorem 3 for the given sets. 22. A = {a, b, c, d, e}, B = {d, e, f, g, h, i, k}, C = {a, c, d, e, k, r, s, t} 23. A = {1, 2, 3, 4, 5, 6}, B = {2, 4, 7, 8, 9}, C = {1, 2, 4, 7, 10, 12} 24. A = {x | x is a positive integer < 8}, B = {x | x is an integer such that 2 ≤ x ≤ 4}, C = {x | x is an integer such that x 2 < 16} 25. In a survey of 260 college students, the following data were obtained: 64 had taken a mathematics course, 100 people questioned, 37 say they eat fruits, 33 say they eat vegetables, 9 say they eat cheese and fruits, 12 eat cheese and vegetables, 10 eat fruits and vegetables, 12 eat only cheese, and 3 report they eat all three offerings. How many people surveyed eat cheese? How many do not eat any of the offerings? 29. In a psychology experiment, the subjects under study were classified according to body type and gender as follows: ENDOMORPH ECTOMORPH MESOMORPH 72 62 54 64 36 38 Male Female (a) How many male subjects were there? 94 had taken a computer science course, (b) How many subjects were ectomorphs? 58 had taken a business course, (c) How many subjects were either female or endomorphs? 28 had taken both a mathematics and a business course, 26 had taken both a mathematics and a computer science course, 22 had taken both a computer science and a business course, and 14 had taken all three types of courses. (a) How many students were surveyed who had taken none of the three types of courses? (b) Of the students surveyed, how many had taken only a computer science course? 26. A survey of 500 television watchers produced the following information: 285 watch football games, 195 watch hockey games, 115 watch basketball games, 45 watch football and basketball games, 70 watch football and hockey games, 50 watch hockey and basketball games, and 50 do not watch any of the three kinds of games. (a) How many people in the survey watch all three kinds of games? (b) How many people watch exactly one of the sports? 27. The Journalism 101 class recently took a survey to determine where the city’s people obtained their news. Unfortunately, some of the reports were damaged. What we know is that 88 people said they obtained their news from television, 73 from the local paper, and 46 from a news magazine. Thirty-four people reported that they obtained news from television and the local paper, 16 said they obtained their news from television and a news magazine, and 12 obtained theirs from the local paper and a news magazine. A total of five people reported that they used all three media. If 166 people were surveyed, how many use none of the three media to obtain their news? How many obtain their news from a news magazine exclusively? 28. The college catering service must decide if the mix of food that is supplied for receptions is appropriate. Of (d) How many subjects were not male mesomorphs? (e) How many subjects were either male, ectomorph, or mesomorph? 30. The following table displays information about the sophomore, junior, and senior classes at Old U. Class Sophomore (S) Junior (J) Senior (R) Major Declared (D) Major Undeclared (U) 143 245 392 289 158 36 For each of the following tell how many students are in the set and describe those students in words. (b) U ∪ R (c) (D ∪ S) ∩ R (a) D ∩ J 31. Create a Venn diagram that displays the information in the table in Exercise 30. 32. Complete the following proof that A ⊆ A ∪ B. Suppose x ∈ A. Then x ∈ A ∪ B, because . Thus by the definition of subset A ⊆ A ∪ B. In Exercises 33 through 38, classify each statement as true, false, or not possible to identify as true or false. 33. Choose x ∈ A ∩ B. (b) x ∈ B (c) x ∈ / A (d) x ∈ / B (a) x ∈ A 34. Choose y ∈ A ∪ B. (b) y ∈ B (c) y ∈ / A (a) y ∈ A (e) y ∈ A ∩ B (d) y ∈ 35. Choose z ∈ A ∪ (B ∩ C). (b) z ∈ B (a) z ∈ A (d) z ∈ B ∩ C (e) z ∈ A (f) y ∈ / A∩B (c) z ∈ C (f) z ∈ /C 1.3 Sequences 36. Choose w ∈ D ∩ (E ∪ F). (b) w ∈ E (c) w ∈ F (a) w ∈ D (e) w ∈ F ∪ E (d) w ∈ D (f) w ∈ (D ∩ E) ∪ (D ∩ F) 37. Choose t ∈ D ∩ E. (b) t ∈ E (c) t ∈ D (a) t ∈ D (e) t ∈ D ∪ E (d) t ∈ E 38. Choose x ∈ A ∪ (B ∩ C). (b) x ∈ B (c) x ∈ C (a) x ∈ A (e) x ∈ (A ∪ B) ∩ (A ∪ C) (d) x ∈ A ∪ B 39. Complete the following proof that A ∩ B ⊆ A. Suppose x ∈ A ∩ B. Then x belongs to . Thus A ∩ B ⊆ A. 40. (a) Draw a Venn diagram to represent the situation C ⊆ A and C ⊆ B. (b) To prove C ⊆ A ∪ B, we should choose an element from which set? (c) Prove that if C ⊆ A and C ⊆ B, then C ⊆ A ∪ B. 41. (a) Draw a Venn diagram to represent the situation A ⊆ C and B ⊆ C. 1.3 13 (b) To prove A ∪ B ⊆ C, we should choose an element from which set? (c) Prove that if A ⊆ C and B ⊆ C, then A ∪ B ⊆ C. 42. Prove that A − (A − B) ⊆ B. 43. Suppose that A ⊕ B = A ⊕ C. Does this guarantee that B = C? Justify your conclusion. 44. Prove that A − B = A ∩ B. 45. If A ∪ B = A ∪ C, must B = C? Explain. 46. If A ∩ B = A ∩ C, must B = C? Explain. 47. Prove that if A ⊆ B and C ⊆ D, then A ∪ C ⊆ B ∪ D and A ∩ C ⊆ B ∩ D. 48. When is A − B = B − A? Explain. 49. Explain the last term in the sum in Theorem 3. Why is |A ∩ B ∩ C| added and |B ∩ C| subtracted? 50. Write the four-set version of Theorem 3; that is, |A ∪ B ∪ C ∪ D| = · · · . 51. Describe in words the n-set version of Theorem 3. Sequences Some of the most important sets arise in connection with sequences. A sequence is simply a list of objects arranged in a definite order; a first element, second element, third element, and so on. The list may stop after n steps, n ∈ N, or it may go on forever. In the first case we say that the sequence is finite, and in the second case we say that it is infinite. The elements may all be different, or some may be repeated. Example 1 The sequence 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1 is a finite sequence with repeated items. The digit zero, for example, occurs as the second, third, fifth, seventh, and eighth ◆ elements of the sequence. Example 2 The list 3, 8, 13, 18, 23, . . . is an infinite sequence. The three dots in the expression mean “and so on,” that is, continue the pattern established by the first few elements. ◆ Example 3 Another infinite sequence is 1, 4, 9, 16, 25, . . . , the list of the squares of all positive integers. ◆ It may happen that how a sequence is to continue is not clear from the first few terms. Also, it may be useful to have a compact notation to describe a sequence. Two kinds of formulas are commonly used to describe sequences. In Example 2, a natural description of the sequence is that successive terms are produced by adding 5 to the previous term. If we use a subscript to indicate a term’s position in the sequence, we can describe the sequence in Example 2 as a1 = 3, an = an−1 + 5, 2 ≤ n. A formula, like this one, that refers to previous terms to define the next term is called recursive. Every recursive formula must include a starting place. On the other hand, in Example 3 it is easy to describe a term using only its position number. In the nth position is the square of n; bn = n 2 , 1 ≤ n. This type of formula is called explicit, because it tells us exactly what value any particular term has. 14 Chapter 1 Fundamentals Example 4 The recursive formula c1 = 5, cn = 2cn−1 , 2 ≤ n ≤ 6, defines the finite sequence ◆ 5, 10, 20, 40, 80, 160. Example 5 The infinite sequence 3, 7, 11, 15, 19, 23, . . . can be defined by the recursive formula d1 = 3, dn = dn−1 + 4. ◆ Example 6 The explicit formula sn = (−4)n , 1 ≤ n, describes the infinite sequence ◆ −4, 16, −64, 256, ... Example 7 The finite sequence 87, 82, 77, 72, 67 can be defined by the explicit formula ◆ tn = 92 − 5n, 1 ≤ n ≤ 5. Example 8 An ordinary English word such as “sturdy” can be viewed as the finite sequence s, t, u, r, d, y composed of letters from the ordinary English alphabet. ◆ In examples such as Example 8, it is common to omit the commas and write the word in the usual way, if no confusion results. Similarly, even a meaningless word such as “abacabcd” may be regarded as a finite sequence of length 8. Sequences of letters or other symbols, written without the commas, are also referred to as strings. Example 9 An infinite string such as abababab. . . may be regarded as the infinite sequence ◆ a, b, a, b, a, b, ... Example 10 The sentence “now is the time for the test” can be regarded as a finite sequence of English words: now, is, the, time, for, the, test. Here the elements of the sequence are themselves words of varying length, so we would not be able simply to omit the commas. The custom is to use spaces instead of commas in this case. ◆ The set corresponding to a sequence is simply the set of all distinct elements in the sequence. Note that an essential feature of a sequence is the order in which the elements are listed. However, the order in which the elements of a set are listed is of no significance at all. Example 11 (a) The set corresponding to the sequence in Example 3 is {1,4,9,16,25, . . .}. ◆ (b) The set corresponding to the sequence in Example 9 is simply {a, b}. The idea of a sequence is important in computer science, where a sequence is sometimes called a linear array or list. We will make a slight but useful distinction between a sequence and an array, and use a slightly different notation. If we have a sequence S : s1 , s2 , s3 , . . . , we think of all the elements of S as completely determined. The element s4 , for example, is some fixed element of S, located in position four. Moreover, if we change any of the elements, we have a new sequence and will probably name it something other than S. Thus if we begin with the finite sequence S : 0, 1, 2, 3, 2, 1, 1 and we change the 3 to a 4, getting 0, 1, 2, 4, 2, 1, 1, we would think of this as a different sequence, say S . 1.3 Sequences S[1] S[2] 15 S[3] ··· Array S: 1 2 3 Figure 1.16 An array, on the other hand, may be viewed as a sequence of positions, which we represent in Figure 1.16 as boxes. The positions form a finite or infinite list, depending on the desired size of the array. Elements from some set may be assigned to the positions of the array S. The element assigned to position n will be denoted by S[n], and the sequence S[1], S[2], S[3], . . . will be called the sequence of values of the array S. The point is that S is considered to be a well-defined object, even if some of the positions have not been assigned values, or if some values are changed during the discussion. The following shows one use of arrays. Characteristic Functions A very useful concept for sets is the characteristic function. We discuss functions in Section 5.1, but for now we can proceed intuitively, and think of a function on a set as a rule that assigns some “value” to each element of the set. If A is a subset of a universal set U , the characteristic function f A of A is defined for each x ∈ U as follows: 1 if x ∈ A f A (x) = 0 if x ∈ / A. We may add and multiply characteristic functions, since their values are numbers, and these operations sometimes help us prove theorems about properties of subsets. THEOREM 1 Characteristic functions of subsets satisfy the following properties: (a) f A∩B = f A f B ; that is, f A∩B (x) = f A (x) f B (x) for all x. (b) f A∪B = f A + f B − f A f B ; that is, f A∪B (x) = f A (x)+ f B (x)− f A (x) f B (x) for all x. (c) f A⊕B = f A + f B −2 f A f B ; that is, f A⊕B (x) = f A (x)+ f B (x)−2 f A (x) f B (x) for all x. Proof (a) f A (x) f B (x) equals 1 if and only if both f A (x) and f B (x) are equal to 1, and this happens if and only if x is in A and x is in B, that is, x is in A ∩ B. Since f A f B is 1 on A ∩ B and 0 otherwise, it must be f A∩B . (b) If x ∈ A, then f A (x) = 1, so f A (x)+ f B (x)− f A (x) f B (x) = 1+ f B (x)− f B (x) = 1. Similarly, when x ∈ B, f A (x) + f B (x) − f A (x) f B (x) = 1. If x is not in A or B, then f A (x) and f B (x) are 0, so f A (x) + f B (x) − f A (x) f B (x) = 0. Thus f A + f B − f A f B is 1 on A ∪ B and 0 otherwise, so it must be f A∪B . ■ (c) We leave the proof of (c) as an exercise. Note that the proof of Theorem 1 proceeds by direct application of the definition of the characteristic function. 16 Chapter 1 Fundamentals Computer Representation of Sets and Subsets Another use of characteristic functions is in representing sets in a computer. To represent a set in a computer, the elements of the set must be arranged in a sequence. The particular sequence selected is of no importance. When we list the set A = {a, b, c, . . . , r } we normally assume no particular ordering of the elements in A. Let us identify for now the set A with the sequence a, b, c, . . . , r . When a universal set U is finite, say U = {x1 , x2 , . . . , xn }, and A is a subset of U , then the characteristic function assigns 1 to an element that belongs to A and 0 to an element that does not belong to A. Thus f A can be represented by a sequence of 0’s and 1’s of length n. Example 12 Let U = {1, 2, 3, 4, 5, 6}, A = {1, 2}, B = {2, 4, 6}, and C = {4, 5, 6}. Then f A (x) has value 1 when x is 1 or 2, and otherwise is 0. Hence f A corresponds to the sequence 1, 1, 0, 0, 0, 0. In a similar way, the finite sequence 0, 1, 0, 1, 0, 1 ◆ represents f B and 0, 0, 0, 1, 1, 1 represents f C . Any set with n elements can be arranged in a sequence of length n, so each of its subsets corresponds to a sequence of zeros and ones of length n, representing the characteristic function of that subset. This fact allows us to represent a universal set in a computer as an array A of length n. Assignment of a zero or one to each location A[k] of the array specifies a unique subset of U . Example 13 Let U = {a, b, e, g, h, r, s, w}. The array of length 8 shown in Figure 1.17 represents U , since A[k] = 1 for 1 ≤ k ≤ 8. 1 1 1 1 1 1 1 1 Figure 1.17 If S = {a, e, r, w}, then f S (x) = 1 0 if x = a, e, r, w if x = b, g, h, s. Hence the array in Figure 1.18 represents the subset S. 1 0 1 Figure 1.18 0 0 1 0 1 ◆ A set is called countable if it is the set corresponding to some sequence. Informally, this means that the members of the set can be arranged in a list, with a first, second, third, . . . , element, and the set can therefore be “counted.” We shall show in Section 2.4 that all finite sets are countable. However, not all infinite sets are countable. A set that is not countable is called uncountable. The most accessible example of an uncountable set is the set of all real numbers that can be represented by an infinite decimal of the form 0.a1 a2 a3 . . . , where ai is an integer and 0 ≤ ai ≤ 9. We shall now show that this set is uncountable. We will prove this result by contradiction; that is, we will show the countability of this set implies an impossible situation. (We will look more closely at proof by contradiction in Chapter 2.) 1.3 Sequences 17 Assume that the set of all decimals 0.a1 a2 a3 . . . is countable. Then we could form the following list (sequence), containing all such decimals: d1 = 0.a1 a2 a3 . . . d2 = 0.b1 b2 b3 . . . d3 = 0.c1 c2 c3 . . . .. . Each of our infinite decimals must appear somewhere on this list. We shall establish a contradiction by constructing an infinite decimal of this type that is not on the list. Now construct a number x as follows: x = 0.x1 x2 x3 . . . , where x1 is 1 if a1 = 2, otherwise x1 is 2; x2 = 1 if b2 = 2, otherwise x2 = 2; x3 = 1 if c3 = 2, otherwise x3 = 2. This process can clearly be continued indefinitely. The resulting number is an infinite decimal consisting of 1’s and 2’s, but by its construction x differs from each number in the list at some position. Thus x is not on the list, a contradiction to our assumption. Hence no matter how the list is constructed, there is some real number of the form 0.x1 x2 x3 . . . that is not in the list. On the other hand, it can be shown that the set of rational numbers is countable. Strings and Regular Expressions Given a set A, we can construct the set A∗ consisting of all finite sequences of elements of A. Often, the set A is not a set of numbers, but some set of symbols. In this case, A is called an alphabet, and the finite sequences in A∗ are called words from A, or sometimes strings from A. For this case in particular, the sequences in A∗ are not written with commas. We assume that A∗ contains the empty sequence or empty string, containing no symbols, and we denote this string by . This string will be useful in Chapters 9 and 10. Example 14 Let A = {a, b, c, . . . , z}, the usual English alphabet. Then A∗ consists of ordinary words, such as ape, sequence, antidisestablishmentarianism, and so on, as well as “words” such as yxaloble, zigadongdong, esy, and pqrst. All finite sequences from A are in A∗ , whether they have meaning or not. ◆ If w1 = s1 s2 s3 . . . sn and w2 = t1 t2 t3 . . . tk are elements of A∗ for some set A, we define the catenation of w1 and w2 as the sequence s1 s2 s3 . . . sn t1 t2 t3 . . . tk . The catenation of w1 with w2 is written as w1 · w2 or w1 w2 , and is another element of A∗ . Note that if w belongs to A∗ , then w · = w and · w = w. This property is convenient and is one of the main reasons for defining the empty string. Example 15 Let A = {John, Sam, Jane, swims, runs, well, quickly, slowly}. Then A∗ contains real sentences such as "Jane swims quickly" and "Sam runs well," as well as nonsense sentences such as "Well swims Jane slowly John." Here we separate the elements in each sequence with spaces. This is often done when the elements of ◆ A are words. The idea of a recursive formula for a sequence is useful in more general settings as well. In the formal languages and the finite state machines we discuss in Chapter 10, the concept of regular expression plays an important role, and regular expressions are defined recursively. A regular expression over A is a string constructed from the elements of A and the symbols (,), ∨, ∗, according to the following definition. RE1. The symbol Λ is a regular expression. RE2. If x ∈ A, the symbol x is a regular expression. RE3. If α and β are regular expressions, then the expression αβ is regular. RE4. If α and β are regular expressions, then the expression (α∨β) is regular. RE5. If α is a regular expression, then the expression (α)∗ is regular. Note here that RE1 and RE2 provide initial regular expressions. The other parts of the definition are used repetitively to define successively larger sets of regular expressions from those already defined. Thus the definition is recursive. By convention, if the regular expression α consists of a single symbol x, where x ∈ A, or if α begins and ends with parentheses, then we write (α)∗ simply as α∗. When no confusion results, we will refer to a regular expression over A simply as a regular expression (omitting reference to A). Example 16 Let A = {0, 1}. Show that the following expressions are all regular expressions over A. (a) 0∗(0 ∨ 1)∗ (b) 00∗(0 ∨ 1)∗1 (c) (01)∗(01 ∨ 1∗) Solution (a) By RE2, 0 and 1 are regular expressions. Thus (0 ∨ 1) is regular by RE4, and so 0∗ and (0 ∨ 1)∗ are regular by RE5 (and the convention mentioned previously). Finally, we see that 0∗(0 ∨ 1)∗ is regular by RE3. (b) We know that 0, 1, and 0∗ are regular. Thus, using RE3 twice, 00∗ is regular. Similarly, since (0 ∨ 1)∗ is regular, ... (c) By RE3, 01 is a regular expression. Since 1∗ is regular, (01∨1∗) is regular by RE4, and using RE5, (01)∗ is regular. Then using RE3, we see that (01)∗(01 ∨ 1∗) is regular. ◆ Associated with each regular expression over A, there is a corresponding subset of A∗. Such sets are called regular subsets of A∗ or just regular sets if no reference to A is needed. To compute the regular set corresponding to a regular expression, we use the following correspondence rules. 1. The expression Λ corresponds to the set {Λ}, where Λ is the empty string in A∗. 2. If x ∈ A, then the regular expression x corresponds to the set {x}. 3. If α and β are regular expressions corresponding to the subsets M and N of A∗, then αβ corresponds to M · N = {s · t | s ∈ M and t ∈ N}. Thus M · N is the set of all catenations of strings in M with strings in N. 4. If the regular expressions α and β correspond to the subsets M and N of A∗, then (α ∨ β) corresponds to M ∪ N. 5. If the regular expression α corresponds to the subset M of A∗, then (α)∗ corresponds to the set M∗. Note that M is a set of strings from A. Elements from M∗ are finite sequences of such strings, and thus may themselves be interpreted as strings from A. Note also that we always have Λ ∈ M∗. Example 17 Let A = {a, b, c}. Then the regular expression a∗ corresponds to the set of all finite sequences of a's, such as aaa, aaaaaaa, and so on. The regular expression a(b ∨ c) corresponds to the set {ab, ac} ⊆ A∗. Finally, the regular expression ab(bc)∗ corresponds to the set of all strings that begin with ab, and then repeat the symbols bc n times, where n ≥ 0. This set includes the strings ab, abbcbc, abbcbcbcbc, and so on. ◆ Example 18 Let A = {0, 1}. Find regular sets corresponding to the three regular expressions in Example 16. Solution (a) The set corresponding to 0∗(0 ∨ 1)∗ consists of all sequences of 0's and 1's. Thus, the set is A∗. (b) The expression 00∗(0 ∨ 1)∗1 corresponds to the set of all sequences of 0's and 1's that begin with at least one 0 and end with at least one 1. (c) The expression (01)∗(01 ∨ 1∗) corresponds to the set of all sequences of 0's and 1's that either repeat the string 01 a total of n ≥ 1 times, or begin with a total of n ≥ 0 repetitions of 01 and end with some number k ≥ 0 of 1's. This set includes, for example, the strings 1111, 01, 010101, 0101010111111, and 011. ◆ 1.3 Exercises In Exercises 1 through 4, give the set corresponding to the sequence. 1. 1, 2, 1, 2, 1, 2, 1, 2, 1 2. 0, 2, 4, 6, 8, 10, . . . 3. aabbccddee . . . zz 4. abbcccdddd 5. Give three different sequences that have {x, y, z} as a corresponding set. 6. Give three different sequences that have {1, 2, 3, . . . } as a corresponding set. In Exercises 7 through 14, write out the first four terms (begin with n = 1) of the sequence whose general term is given. 7. an = 5 n 8. bn = 3n^2 + 2n − 6 9. cn = 1 · 2 · 3 · · · n 10. ... 11. ... 12. ... 13. ... 14. ... 15. Is the sequence defined by ... In Exercises 16 through 23, describe ... Let A = {x | x is an English word whose length is 3}, ... C = {x | x = 4m, m ∈ Z}, D = {x | x ∈ Z and x^2 = 1} ... Identify each set as finite, countable, or uncountable. 25. Let A = {ab, bc, ba}. In each part, tell whether the string belongs to A∗. (a) ababab (b) abc (c) abba (d) abbcbaba (e) bcabbab (f) abbbcba 26. Let U = {arithmetic, algebra, calculus, geometry, trigonometry, analysis, topology, statistics}, B = {analysis, topology, calculus}, C = {algebra, calculus, trigonometry, analysis}, D = {arithmetic, algebra, calculus, topology, statistics}, E = {algebra, calculus, geometry, trigonometry, analysis}. In each of the following, represent the given set by an array of zeros and ones. (a) B ∪ C (b) C ∩ D (c) B ∩ (D ∩ E) (d) B ∪ E (e) ... In Exercises 15 through 20, write ... 21. Write an explicit formula for the sequence 2, 5, 7, 12, 19, 31, . . . (d) B ⊕ C 27. Let U = {b, d, g, h, k, m, n}, B = {b}, C = {d, g, m, n}, and D = {d, k, n}. (a) What is fB(b)? (b) What is fC(b)? (c) Find the sequences of length 8 that correspond to fB, fC, and fD. (d) Represent B ∪ C, C ∪ D, and C ∩ D by arrays of zeros and ones. 28. Complete the proof that fA⊕B = fA + fB − 2fA fB 29. Use characteristic functions to prove that ... Chapter 1 Fundamentals We now prove that fA ... Thus ... 32. Let ... (a) fA∩B = fA fB ... The remaining case to check is x ∈ A ∩ B. If x ∈ A ∩ B, then x ∈ A and x ∈ B, so fA(x) + fB(x) − 2 fA(x) fB(x) = 0. Explain how these steps prove Theorem 1(c). 29. Using characteristic functions, prove that (A ⊕ B) ⊕ C = A ⊕ (B ⊕ C). 30. Let A = {+, ×, a, b}. Show that the following expressions are regular over A. (a) a + b(ab)∗(a × b ∨ a) (b) a(+ ∨ ×)(ab ∨ b)∗ (c) a∗b(a ∨ b)∗ ... b in Exercises 31 and 32, let A = {a, b, c}. In each exercise a string in A∗ is listed and a regular expression over A is given. In each case, tell whether or not the string on the left belongs to the regular set corresponding to the regular expression on the right. 31. (a) ac (b) abcc (c) aaacc (a ∨ b) c∗ (a ∨ b ∨ c)∗ 32. (a) cc (b) aacbbb (c) ... 33. Give three expressions that are not regular over the given A. (a) A = {a, b} ... 34. Let A = ... Describe each of the following ... 35. Give a recursive formula for the given sequence. (a) ... 36. Let ... 1.4 Division in the Integers We shall now discuss some results needed later about division and factoring in the integers. If m is an integer and n is a positive integer, we can plot the integer multiples of n on a line, and locate m as in Figure 1.19. If m is a multiple of n, say m = qn, then we can write m = qn + r, where r is 0. On the other hand, as is shown in Figure 1.19, if m is not a multiple of n, we let qn be the first multiple of n lying to the left of m and let r be m − qn. Then r is the distance from qn to m, so clearly 0 < r < n, and again we have m = qn + r. We state these observations as a theorem. m ...−2n −n 0 n 2n ... qn m (q + 1)n ... Figure 1.19 THEOREM 1 If n and m are integers and n > 0, we can write m = qn + r for integers q and r with 0 ≤ r < n. Moreover, there is just one way to do this. 1.4 Properties of Integers Example 1 (a) If n is 3 and m is 16, then 16 = 5(3) + 1 so q = 5 and r = 1. (b) If n is 10 and m is 3, then 3 = 0(10) + 3 so q = 0 and r = 3. (c) If n is 5 and m is −11, then −11 = −3(5) + 4 so q = −3 and r = 4. ◆ If the r in Theorem 1 is zero, so that m is a multiple of n, we write n | m, which is read "n divides m." If m is not a multiple of n, we write n ∤ m, which is read "n does not divide m." We now prove some simple properties of divisibility. THEOREM 2 Let a, b, and c be integers. (a) If a | b and a | c, then a | (b + c). (b) If a | b and a | c, where b > c, then a | (b − c). (c) If a | b or a | c, then a | bc. (d) If a | b and b | c, then a | c. Proof (a) If a | b and a | c, then b = k1 a and c = k2 a for integers k1 and k2. So b + c = (k1 + k2) a and a | (b + c). (b) This can be proved in exactly the same way as (a). (c) As in (a), we have b = k1 a or c = k2 a. Then either bc = k1 ac or bc = k2 ab, so in either case bc is a multiple of a and a | bc. (d) If a | b and b | c, we have b = k1 a and c = k2 b, so c = k2 b = k2 (k1 a) = (k2 k1) a and hence a | c. ■ Note that again we have a proof that proceeds directly from the original conditions by restating the original conditions. As a consequence of Theorem 2, we have that if a | b and a | c, then a | (mb + nc), for any integers m and n. (See Exercises 23 and 24.) A number p > 1 in Z+ is called prime if the only positive integers that divide p are p and 1. Example 2 The numbers 2, 3, 5, 7, 11, and 13 are prime, while 4, 10, 16, and 21 are not prime. ◆ It is easy to write a set of steps, or an algorithm, to determine if a positive integer n > 1 is a prime number. First we check to see if n is 2. If n > 2, we could divide by every integer from 2 to n − 1, and if none of these is a divisor of n, then n is prime. To make the process more efficient, we note that if mk = n, then either m or k is less than or equal to √n. This means that if n is not prime, it has a divisor k satisfying the inequality 1 < k ≤ √n, so we need only test for divisors in this range. Also, if n has any even number as a divisor, it must have 2 as a divisor. Thus after checking for divisibility by 2, we may skip division by even integers. Algorithm To test whether an integer N > 1 is prime: Step 1 Check whether N is 2. If so, N is prime. If not, proceed to Step 2 Check whether 2 | N. If so, N is not prime; otherwise, proceed to Step 3 Compute the largest integer K ≤ √N. Then 4 Check whether D | N, where D is any odd number such that 1 < D ≤ K. If D | N, then N is not prime; otherwise, N is prime. ● Testing whether an integer is prime is a common task for computers. The algorithm given here is too inefficient for testing very large numbers, but there are many other algorithms for testing whether an integer is prime. THEOREM 3 Every positive integer n > 1 can be written uniquely as p1^{k1} p2^{k2} · · · ps^{ks}, where p1 < p2 < · · · < ps are distinct primes that divide n and the k's are positive integers giving the number of times each prime occurs as a factor of n. ■ We leave the proof of Theorem 3 to Section 2.4, but we give several illustrations. Example 3 (a) 9 = 3 · 3 = 3^2 (b) 24 = 8 · 3 = 2 · 2 · 2 · 3 = 2^3 · 3 (c) 30 = 2 · 3 · 5 ◆ Greatest Common Divisor If a, b, and k are in Z+, and k | a and k | b, we say that k is a common divisor of a and b. If d is the largest such k, d is called the greatest common divisor, or GCD, of a and b, and we write d = GCD(a, b). This number has some interesting properties. It can be written as a combination of a and b, and it is not only larger than all the other common divisors, it is a multiple of each of them. THEOREM 4 If d is GCD(a, b), then (a) d = sa + tb for some integers s and t. (These are not necessarily positive.) (b) If c is any other common divisor of a and b, then c | d. Proof Let x be the smallest positive integer that can be written as sa + tb for some integers s and t, and let c be a common divisor of a and b. Since c | a and c | b, it follows from Theorem 2 that c | x, so c ≤ x. If we can show that x is a common divisor of a and b, it will then be the greatest common divisor of a and b and both parts of the theorem will have been proved. By Theorem 1, a = qx + r with 0 ≤ r < x. Solving for r, we have r = a − qx = a − q(sa + tb) = a − qsa − qtb = (1 − qs)a + (−qt)b. If r is not zero, then since r < x and r is the sum of a multiple of a and a multiple of b, we will have a contradiction to the fact that x is the smallest positive number that is a sum of multiples of a and b. Thus r must be 0 and x | a. In the same way we can show that x | b, and this completes the proof. ■ This proof is more complex than the earlier ones. At this stage you should focus on understanding the details of each step. We will discuss the structure of this proof later. From the definition of greatest common divisor and Theorem 4(b), we have the following result: Let a, b, and d be in Z+. The integer d is the greatest common divisor of a and b if and only if (a) d | a and d | b. (b) Whenever c | a and c | b, then c | d. Example 4 (a) The common divisors of 12 and 30 are 1, 2, 3, and 6, so that GCD(12, 30) = 6 and 6 = 1 · 30 + (−2) · 12. (b) It is clear that GCD(17, 95) = 1 since 17 is prime and 17 ∤ 95. (c) If c is GCD(a, b), then a = ka and b = mc ... Example 5 ... If GCD(a, b) = 1, as in Example 4(b), then a and b are relatively prime. One remaining question is that of how to compute the GCD conveniently in general. Repeated application of Theorem 1 provides the key to doing this. We now present a procedure, called the Euclidean algorithm, for finding GCD(a, b). Suppose that a > b > 0 (otherwise interchange a and b). Then by Theorem 1, we may write a = k1 b + r1, where k1 is in Z+ and 0 ≤ r1 < b. (1) Now Theorem 2 tells us that if n divides a and b, then it must divide r1, since r1 = a − k1 b. Similarly, if n divides b and r1, then it must divide a. We see that the common divisors of a and b are the same as the common divisors of b and r1, so GCD(a, b) = GCD(b, r1). We now continue using Theorem 1 as follows: divide b by r1 : divide r1 by r2 : divide r2 by r3 : ... divide rn−2 by rn−1 : divide rn−1 by rn : b = k2 r1 + r2 r1 = k3 r2 + r3 r2 = k4 r3 + r4 ... rn−2 = kn rn−1 + rn rn−1 = kn+1 rn + rn+1 0 ≤ r2 < r1 0 ≤ r3 < r2 0 ≤ r4 < r3 ... 0 ≤ rn < rn−1 0 ≤ rn+1 < rn. (2) Since a > b > r1 > r2 > r3 > r4 > · · · , the remainder will eventually become zero, so at some point we have rn+1 = 0. We now show that rn = GCD(a, b). We saw previously that GCD(a, b) = GCD(b, r1). Repeating this argument with b and r1, we see that GCD(b, r1) = GCD(r1, r2). Upon continuing, we have GCD(a, b) = GCD(b, r1) = GCD(r1, r2) = · · · = GCD(rn−1, rn). Since rn−1 = kn+1 rn, we see that rn | rn−1. Hence GCD(rn−1, rn) = rn. Thus rn = GCD(a, b). ◆ 24 Chapter 1 Fundamentals Example 6 Compute GCD(273, 98). Since a > b, let a = 273 and b = 98. Using the Euclidean algorithm, divide 273 by 98: divide 98 by 77: divide 77 by 21: divide 21 by 14: divide 14 by 7: 273 = 2 · 98 + 77 98 = 1 · 77 + 21 77 = 3 · 21 + 14 21 = 1 · 14 + 7 14 = 2 · 7 + 0 so GCD(273, 98) = 7, the last nonzero remainder. ◆ In Theorem 4(a), we observed that if d = GCD(a, b), we can find integers s and t such that d = sa + tb. The integers s and t can be found as follows. Solve the next-to-last equation in (2) for rn : rn = rn−2 − kn rn−1. (3) Now solve the second-to-last equation in (2), rn−3 = kn−1 rn−2 + rn−1, for rn−1 : rn−1 = rn−3 − kn−1 rn−2 and substitute this expression in (3): rn = rn−2 − kn [rn−3 − kn−1 rn−2]. Continue to work up through the equations in (2) and (1), replacing ri by an expression involving ri+1 and ri+2, until we arrive at an expression involving only a and b. Example 7 (a) Let a = 273 and b = 98. Then GCD(273, 98) = 7 = 21 − 1 · 14 = 21 − 1 · [77 − 3 · 21] = 4 · 21 − 1 · 77 = 4 · [98 − 77] − 1 · 77 = 4 · 98 − 5 · 77 = 4 · 98 − 5 · [273 − 2 · 98] = 14 · 98 − 5 · 273 Hence s = −5 and t = 14. Note that the key is to carry out the arithmetic only partially. (b) Let a = 108 and b = 60. Then GCD(108, 60) = 12 = 108 − 1 · 96 = 108 − 1 · [60 + 48 = 108 − 1 · 60 − 1 · 48 ...] THEOREM 5 If a and b are in Z+, and a | bc and GCD(a, b) = 1, then a | c. Proof Since a | bc and ... Theorem 4 ... Since a and b have the same common divisors ... ■ We must have the same greatest common divisor. Note that this is another direct proof, but that uses a previous theorem. We state this corollary without proof. Corollary 1 If p is a prime and p | a1 a2 · · · ak, then p | ai for some i. ■ Least Common Multiple If a, b, and k are in Z+, and a | k, b | k, we say k is a common multiple of a and b. The smallest such k, call it c, is called the least common multiple, or LCM, of a and b, and we write c = LCM(a, b). The following result shows that we can obtain the least common multiple from the greatest common divisor, so we do not need a separate procedure for finding the least common multiple. THEOREM 6 If a and b are two positive integers, then GCD(a, b) · LCM(a, b) = ab. Proof Let p1, p2, . . . , pk be all the prime factors of either a or b. Then we can write a = p1^{a1} p2^{a2} · · · pk^{ak} and b = p1^{b1} p2^{b2} · · · pk^{bk} where some of the ai and bi may be zero. It then follows that min(a1, b1) min(a2, b2) ... GCD(a, b) = p1^{min(a1,b1)} ... and LCM(a, b) = p1^{max(a1,b1)} ... Hence GCD(a, b) · LCM(a, b) = ... = ab. ■ Example 8 Let a = 540 and b = 504. Factoring a and b into primes, we obtain a = 540 = 2^2 · 3^3 · 5 and b = 504 = 2^3 · 3^2 · 7. Thus we can write a = 2^2 · 3^3 · 5^1 · 7^0 and b = 2^3 · 3^2 · 5^0 · 7^1. Then GCD(540, 504) = 2^{min(2,3)} · 3^{min(3,2)} · 5^{min(1,0)} · 7^{min(0,1)} = 2^2 · 3^2 · 5^0 · 7^0 = 36. Also, LCM(540, 504) = 2^{max(2,3)} · 3^{max(3,2)} · 5^{max(1,0)} · 7^{max(0,1)} = 2^3 · 3^3 · 5^1 · 7^1 = 7560. Then GCD(540, 504) · LCM(540, 504) = 36 · 7560 = 272,160 = 540 · 504. As a verification, we can also compute GCD(540, 504) by the Euclidean algorithm and obtain the same result. ◆ If n and m are integers and n > 1, Theorem 1 tells us we can write m = qn + r, 0 ≤ r < n. Sometimes the remainder r is more important than the quotient q. Example 9 If the time is now 4 o'clock, what time will it be 101 hours from now? Solution Let n = 24 and m = 4 + 101 or 105. Then we have 105 = 4 · 24 + 9. The remainder 9 answers the question. In 101 hours it will be 9 o'clock. ◆ For each n ∈ Z+, we define a function fn, the mod-n function, as follows: If z is a nonnegative integer, fn(z) = r, the remainder when z is divided by n. (Again, functions are formally defined in Section 5.1, but as in Section 1.3, we need only think of a function as a rule that assigns some "value" to each member of a set.) The naming of these functions is made clear in Section 4.5. Example 9 (a) f3(14) = 2, because 14 = 4 · 3 + 2. (b) f7(153) = 6. ◆ Pseudocode Versions An alternative to expressing an algorithm in ordinary English as we did in this section is to express it in something like a computer language. Throughout the book we use a pseudocode language, which is described in detail in Appendix A. Here we give pseudocode versions for an algorithm that determines if an integer is prime and for an algorithm that calculates the greatest common divisor of two integers. In the pseudocode for the algorithm to determine if an integer is prime, we assume the existence of functions SQR and INT, where SQR(N) returns the greatest √ integer not exceeding N, and INT(X) returns the greatest integer not exceeding X. For example, SQR(10) = 3, SQR(25) = 5, INT(7.124) = 7, and INT(8) = 8. SUBROUTINE PRIME(N) 1. IF (N = 2) THEN a. PRINT ('PRIME') b. RETURN 2. ELSE 1. IF (N/2 = INT(N/2)) THEN 1. PRINT ('NOT PRIME') 2. RETURN 3. ELSE 1. FOR D = 3 THRU SQR(N) BY 2 a. IF (N/D = INT(N/D)) THEN 1. PRINT ('NOT PRIME') 2. RETURN 2. PRINT ('PRIME') 3. RETURN END OF SUBROUTINE PRIME The following gives a pseudocode program for finding the greatest common divisor of two positive integers. This procedure is different from the Euclidean algorithm, but in Chapter 2, we will see how to prove that this algorithm does find the greatest common divisor. FUNCTION GCD(X,Y) 1. WHILE (X ≠ Y) a. IF (X > Y) THEN 1. X ← X − Y b. ELSE 1. Y ← Y − X 2. RETURN (X) END OF FUNCTION GCD Example 10 Use the pseudocode for GCD to calculate the greatest common divisor of 190 and 34. Solution The following table gives the values of X, Y, X − Y, or Y − X as we go through the program. X Y X−Y Y−X 190 156 122 88 54 20 20 6 14 6 8 2 4 2 34 34 34 34 34 14 14 8 2 2 2 Since the last value of X is 2, the greatest common divisor of 190 and 34 is 2. ◆ Representations of Integers The decimal representation of an integer is so familiar that we sometimes regard it as the symbol, or name for that integer. For example, when we write the integer 3264, we are saying that the number is the result of adding 3 times 103, 2 times 102, 6 times 101, and 4 times 100. We say 3264 is the base 10 expansion of n or the decimal expansion of n; 10 is called the base of this expansion. There is nothing special about using the number 10 as a base, and it is likely the result of our having 10 fingers on which to count. Any positive integer b > 1 can be used in a similar way, and these expansions are often of much greater use than the standard decimal expansion. The bases 2, 8, and 16 are frequently used in computer science, and the base 26 is sometimes used in cryptology (the science of producing and deciphering secret codes). THEOREM 7 If b > 1 is an integer, then every positive integer n can be uniquely expressed in the form n = dk b^k + dk−1 b^{k−1} + · · · + d1 b + d0, (4) where 0 ≤ di < b, i = 0, 1, . . . , k, and dk ≠ 0. The sequence dk . . . d1 d0 is called the base b expansion of n. If we need to explicitly indicate the base b, we will write the above sequence as (dk dk−1 . . . d1 d0)b. Proof We prove only the existence of the expansion. First, divide n by b. Theorem 1 ... we obtain n = q0 b + r0 ... Example 11 Find the base 4 representation of 158. Solution We repeatedly divide by 4 and save the remainders: 4 158 39 4 9 4 2 0 Thus 158 = (2132)4. 2 3 1 2 ◆ The following pseudocode algorithm returns the digits of the base b expansion of an integer. We use the expression m mod n to denote the mod-n function value for m, that is, the remainder after dividing m by n. These are the functions defined after Example 8. The mod-n functions are found in most programming languages. SUBROUTINE EXPANSION(N, B) 1. Q ← N 2. K ← 0 3. WHILE (Q ≠ 0) a. Dk ← Q mod B b. Q ← INT(Q/B) c. K ← K + 1 4. RETURN END OF SUBROUTINE EXPANSION When this subroutine ends, the base B expansion of N consists of the integers Dk, which can then be further processed. 1.4 Properties of Integers 29 We are familiar with the addition of integers in base 10 ... Example 12 Let n = (3134) and m = (3223) ... 2 2 1 ... Find the base 4 expansion of m + n. Solution Adding the digits in the last column, we have 4 + 3 = 2 + 4 (11)4, so we record a 1 and carry a 1 to the next column. 1 3 1 3 + 3 2 2 1 Adding the digits in the second column, we have 1 + 1 + 2 = (10)4, so we record a 0 and carry a 1. Continuing, we find the answer is (13003)4 + ... ◆ Cryptology The processes of encoding and decoding ... Example 13 (a) The binary expansion of 39 is (100111)2. (b) (110101)2 = 32 + 16 + 4 + 1 = 53. ◆ Binary addition and multiplication are usually implemented in computer circuitry rather than by software. Another common base in computer science is base 16, or hexadecimal (or hex) representation. This requires us to use additional symbols to write numbers in base 16. ... represent 10, 11, 12, 13, 14, 15 as the letters A, B, C, D, E, and F. Example 14 ... we will use the integers 1, 2, . . . , 25 in this way we can interpret any text as the base 26 representation of an integer. We can interpret "TWO" as the base 26 representation of (T×262)+(W×26)+O = (19×676)+(22×26)+14 = 13430. The ability to "add" words can be made the basis of cryptographic encoding. We ◆ will explore this further in Chapter 11. Example 15 As an example of cryptology, we consider a remarkable code due to Sir Francis Bacon, the English writer and philosopher. Suppose that we wish to encode a message, say FLEE NOW. We first write the base 2 representation of the position of each letter of the message in the English alphabet, starting with A in position 0, and ending with Z in position 25. Thus we have the following table. F 00101 L 01011 E 00100 E 00100 N 01101 O 01110 W 10110 Here F is in position 5, its binary representation is (00101)2, and so on. Now choose an unrelated "dummy" message exactly five times as long (padded with a few extra letters if necessary). Place the dummy message in a third row. The letters of the dummy message correspond exactly to the string of 0's and 1's in the second row of the above table. We agree to write each letter in one font (say Times Roman) if it corresponds to a 0 (Bacon used fonts that were even more similar). Thus if the dummy message is ONCE UPON A TIME IN THE WEST THERE WAS A TOWN, we would write that message as ONCE UPON A TIME IN THE WEST THERE WAS A TOWN. Note that when the letters in this message are arranged in a third row of the table, the pattern of nonitalic for 0 and italic for 1 allow us to decode the message. F 00101 ONCE L 01011 UPONA E 00100 TIMEI E 00100 NTHEW N 01101 ESTTH O 01110 EREWA W 10110 STOWN ... Example 16 Suppose that we wish to decode the following message, using the Bacon code NOW IS THE TIME FOR ALL GOOD MEN TO AID THE COUNTRY Since there are 40 letters, the true message must have 8 letters. Arrange the dummy message, 5 letters at a time in the following table, and then list the corresponding binary digits, using 1 for italic and 0 for plain text. NOWIS THETI MEFOR ALLGO ODMEN TOAID THECO UNTRY 01000 10011 10001 01011 00001 10010 10011 00011 ... The corresponding letters ... STARTJOB ... The decoded message is START JOB. ◆ If the fonts used are quite similar, it will not be obvious that the text is a coded message. Thus this example also illustrates steganography, the science of concealment of coded information. Modern versions include hiding information in stray bits of a digital photograph. One use of this is to watermark copyrighted artistic material. 1.4 Exercises In Exercises 1 through 4, for the given integers m and n, write m as qn + r, with 0 ≤ r < n. 1. m = 20, n = 3 2. m = 64, n = 37 3. m = 3, n = 22 4. m = 48, n = 12 In Exercises 5 through 8, write each integer as a product of powers of primes (as in Theorem 3). 5. 828 6. 1666 7. 1781 8. 1125 9. 107 In Exercises 10 through 13, find the least common multiple of the integers. 10. 72, 108 11. 150, 70 12. 175, 245 13. 32, 27 14 Properties of Integers 14. If f is the mod-7 function, compute each of the following. (a) f (48) (b) f (129) (c) f (3) (d) f (1961) (e) f (1362) (f) f (1960) 15. If f is the mod-11 function, compute each of the following. (a) f (386) (b) f (1232) (c) f (39) (d) f (2722) (e) f (6175) (f) f (1759) 16. Find the base 2 expansion of each of the following ... 17. If f is the mod-7 function, compute each of the following. (a) f (12) (b) f (−12) (c) f (1) (d) f (6273) (e) f (8) (f) f (0) ... Find the base 8 expansion of ... 18. What is the connection between ... Exercises 16 and 17, explain why f (a) = f (b) ... 20. Let f be the mod-n function for a fixed n. If a ... 21. Show that if p is prime ... 22. ... 23. Show that GCD(a, b) ... 24. Prove that if a | b and a | c ... 25. Show that if a | b and b | a, ... 26. ... 27. Show that if a and b are positive integers ... 28. ... 29. Show that GCD(a, b) = ... 30. Show that ... (c) Let a, b ... (d) ... 31. Let f be ... Find the base 10 expansion of each of the following. (i) (1111)2 ... (ii) (3205) (i) (1445) (iv) (1123115) (iii) (124215) ... 42. (a) Write the expansion in base 7 ... In Exercises ... (b) ... (c) ... Let A be ... 43. Find the base 10 expansion ... 44. ... 45. ... 46. Explain how ... 47. ... Use Bacon's code ... to create a dummy message for COME BACK ... 48. Decode the message ... WHEN THE MOON COMES OVER THE MOUNTAIN, THE OWLS FLY HIGH. ... 1.5 Matrices A matrix is a rectangular array of numbers arranged in m horizontal rows and n vertical columns: ⎡ a11 a12 · · · a1n ⎤ ... ⎣ am1 am2 · · · amn ⎦ A = The ith row of A is [ai1 ai2 · · · ain], 1 ≤ i ≤ m, and the jth column of A is ⎡ a1j ⎤ ... ⎣ amj ⎦, 1 ≤ j ≤ n. We also write the matrix A as [aij]. We say that A is m by n, written m × n. If m = n, then A is a square matrix of order n and the numbers a11, a22, . . . , ann form the main diagonal of A. We refer to the number aij, which is in the ith row and jth column of A, as the i, jth element of A or as the (i, j) entry of A, and we often write (1) as A = [aij]. Note that first the row is named and then the column. Example 1 Let A = ⎡ 2 3 5 ⎤ ⎣ −1 6 2 ⎦, B = ⎡ 2 3 ⎤ ⎣ 5 −1 ⎦, C = ... D = ... and E = ⎡ −1 0 2 ⎤ ... Then A is 2 × 3 with a12 = 3 and a23 = 2, B is 2 × 2 with b21 = 4, C is 1 × 4, D is 3 × 1, and E is 3 × 3. ◆ A square matrix A = [aij] for which every entry off the main diagonal is zero, that is, aij = 0 for i ≠ j, is called a diagonal matrix. Example 2 Each of the following is a diagonal matrix. ⎡ 2 0 0 4 0 ⎤, G = ⎡ 3 0 0 −3 1 0 5 0 0 ⎤ and H = ... ◆ Matrices are used in many applications in computer science, and we shall see them in our study of relations and graphs. At this point we present the following simple application showing how matrices can be used to display data in a tabular form. 1.5 Matrices Example 3 33 The following matrix gives the airline distances between the cities indicated. London ⎡ London 0 Madrid ⎢ 785 ⎢ 3469 New York Tokyo 5959 Madrid New York 785 0 3593 6706 3469 3593 0 6757 Tokyo ⎥ 5959 6706 ⎥ 6757 ⎦ 0 ◆ Two m × n matrices A = [aij] and B = [bij] are said to be equal if aij = bij , 1 ≤ i ≤ m, 1 ≤ j ≤ n; that is, if corresponding elements are the same. Notice how easy it is to state the definition using generic elements aij , bij . Example 4 Example 5 If ⎡ 2 −3 −1 5 2 ⎦ A=[0 4 −4 6 ⎤ 2 x −1 5 2 ⎦, B=[y 4 −4 z ⎦ and then A = B if and only if x = −3, y = 0, and z = 6. ◆ If A = [aij] and B = [bij] are m × n matrices, then the sum of A and B is the matrix C = [cij] defined by cij = aij + bij , 1 ≤ i ≤ m, 1 ≤ j ≤ n. That is, C is obtained by adding the corresponding elements of A and B. Once again the use of generic elements makes it easy to state the definition. 4 5 3 3 4 −1 . Then and B = Let A = 0 −3 2 5 0 −2 7 9 2 3+4 4+5 −1 + 3 . = A+B= 5 −3 0 5 + 0 0 + (−3) −2 + 2 ◆ Observe that the sum of the matrices A and B is defined only when A and B have the same number of rows and the same number of columns. We agree to write A + B only when the sum is defined. A matrix all of whose entries are zero is called a zero matrix and is denoted by 0. Example 6 Each of the following is a zero matrix. 0 0 0 0 0 0 0 0 0 ⎡ 0 0 0 ⎦ 0 0 0 ⎦ 0 0 0 ◆ The following theorem gives some basic properties of matrix addition; the proofs are omitted. THEOREM 1 (a) A + B = B + A (b) (A + B) + C = A + (B + C) (c) A + 0 = 0 + A = A ■ If A = [aij] is an m × p matrix and B = [bij] is a p × n matrix, then the product of A and B, denoted AB, is the m × n matrix C = [cij] defined by cij = ai1 b1j + ai2 b2j + · · · + aip bpj 1 ≤ i ≤ m, 1 ≤ j ≤ n. (2) 34 Chapter 1 Fundamentals ⎡ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎣ a11 a12 ... a1p a21 .. . ai1 .. . a22 .. . ai2 .. . ... a2p .. . aip .. . am1 am2 . . . amp ... ⎤ ⎥ ⎥ ⎥ ⎥ ⎥ ⎥ ⎥ ⎥ ⎥ ⎥ ⎥ ⎥ ⎦ b11 ⎥ ⎥ ⎥ b21 ⎥ ⎢ .. ⎥ ⎣ . bp1 ⎤ b12 ... b1j ... b1n b22 ... ... b2j .. ... b2n .. bp2 ... bpj ... bpn ai1 b1j ai2 ... b2j ... aip bpj ⎡ c11 ⎥ ⎥ ⎥ ⎢ c21 ⎢ = ⎢ .. ⎥ ⎢ . ⎥ ⎦ ⎣ cm1 ⎤ c12 ... c1n c22 ... ... c2n ... cm2 ... cmn cij ⎥ ⎥ ⎥ ⎥ ⎥ ⎦ Multiply corresponding ⎥ elements together and ⎥ add the results to form cij. Figure 1.20 Let us explain (2) in more detail. The elements ai1 , ai2 , . . . , ai p form the ith row of A, and the elements b1j , b2j , . . . , b pj form the jth column of B. Then (2) states that for any i and j, the element cij of C = AB can be computed in the following way, illustrated in Figure 1.20. 1. Select row i of A and column j of B, and place them side by side. 2. Multiply corresponding entries and add all the products. Example 7 Let A = 2 1 3 −4 2 3 ⎡ 3 1 2 ⎦. Then and B = ⎣ −2 5 −3 (2)(3) + (3)(−2) + (−4)(5) (2)(1) + (3)(2) + (−4)(−3) (1)(3) + (2)(−2) + (3)(5) (1)(1) + (2)(2) + (3)(−3) −20 20 . = 14 −4 AB = ◆ An array of dimension two is a modification of the idea of a matrix, in the same way that a linear array is a modification of the idea of a sequence. By an m × n array A we will mean an m × n matrix A of mn positions. We may assign numbers to these positions later, make further changes in these assignments, and still refer to the array as A. This is a model for two-dimensional storage of information in a computer. The number assigned to row i and column j of an array A will be denoted A[i, j]. As we have seen, the properties of matrix addition resemble the familiar properties for the addition of real numbers. However, some of the properties of matrix multiplication do not resemble those of real number multiplication. First, observe that if A is an m × p matrix and B is a p × n matrix, then AB can be computed and is an m × n matrix. As for BA, we have the following four possibilities: 1. BA may not be defined; we may have n ≠ m. 2. BA may be defined and then BA is p × p, while AB is m × m and p ≠ m. Thus AB and BA are not equal. 3. AB and BA may both be the same size, but not be equal as matrices. 4. AB = BA. 1.5 Matrices Example 8 35 We agree as before to write AB only when the product is defined. 4 −5 1 −1 2 1 and . Then AB = and B = Let A = −1 3 2 −3 −2 −1 3 . BA = ◆ −5 8 The basic properties of matrix multiplication are given by the following theorem. THEOREM 2 (a) A(BC) = (AB)C (b) A(B + C) = AB + AC (c) (A + B)C = AC + BC ■ The n × n diagonal matrix ⎡ 1 0 · · · 0 ⎤ 0 1 · · · 0 ⎥ In = ⎢ , .. ⎥ ⎣ . . . ... ⎦ 0 0 · · · 1 all of whose diagonal elements are 1, is called the identity matrix of order n. If A is an m × n matrix, it is easy to verify that Im A = AIn = A. If A is an n × n matrix and p is a positive integer, we define Ap = A · A· · · A and A0 = I n . p factors If p and q are nonnegative integers, we can prove the following laws of exponents for matrices: A p Aq = A p+q and (A p)q = A pq . Observe that the rule (AB) p = A p B p does not hold for all square matrices. Howp p p ever, if AB = BA, then (AB) = A B . If A = [aij] is an m × n matrix, then the n × m matrix AT = [aiTj], where aiTj = a ji , 1 ≤ i ≤ m, 1 ≤ j ≤ n, is called the transpose of A. Thus the transpose Example 9 of A is obtained by interchanging the rows and columns of A. ⎡ ⎤ 3 4 5 2 −3 5 0 ⎦. Then and B = ⎣ 2 −1 Let A = 6 1 3 1 6 −2 ⎤ ⎡ ⎡ 2 6 3 2 1 1 ⎦ and BT = ⎣ 4 −1 6 ⎦. AT = ⎣ −3 5 3 5 0 −2 ◆ The following theorem summarizes the basic properties of the transpose operation. THEOREM 3 If A and B are matrices, then (a) (AT)T = A (b) (A + B)T = AT + BT (c) (AB)T = BT AT ■ 36 Chapter 1 Fundamentals Example 10 Example 11 A matrix A = [aij] is called symmetric if AT = A. Thus, if A is symmetric, it must be a square matrix. It is easy to show that A is symmetric if and only if aij = a ji . That is, A is symmetric if and only if the entries of A are symmetric with respect to the main diagonal of A. ⎤ ⎡ ⎤ ⎡ 1 2 −3 1 2 −3 4 5 ⎦ and B = ⎣ 2 4 0 ⎦, then A is symmetric and B If A = ⎣ 2 3 2 1 −3 5 6 is not symmetric. ◆ If x is a nonzero number, there is a number y such that x y = 1. The number y is called the multiplicative inverse of x. When we have a multiplication for objects other than numbers, it is reasonable to ask if a multiplicative inverse exists. One example is matrix multiplication, described in this section. If A and B are n ×n matrices, then we say that B is an inverse of A if AB = In and BA = In , where In is the n × n identity matrix defined earlier. The identity matrix behaves like the number 1, in that In A = AIn = A for any n × n matrix A, so the matrix inverse is analogous to the reciprocal of a nonzero number. However, it is not clear how to construct inverses, or even when they exist. ⎤ ⎡ ⎤ ⎡ −2 3 −3 1 3 0 1 . This can An inverse of the matrix ⎣ 2 2 1 ⎦ is the matrix ⎣ 1 −1 2 −3 4 1 0 1 be verified by checking that ⎤ ⎡ ⎤⎡ ⎤ ⎡ 1 0 0 1 3 0 −2 3 −3 ⎣ 2 2 1 ⎦ ⎣ 1 −1 1 ⎦=⎣ 0 1 0 ⎦ 2 −3 4 0 0 1 1 0 1 and ⎤ ⎡ ⎤⎡ ⎤ ⎡ 1 3 0 −2 3 −3 1 0 0 ⎣ 1 −1 1 ⎦ ⎣ 2 2 1 ⎦ = ⎣ 0 1 0 ⎦ . 2 −3 4 1 0 1 0 0 1 ⎡ ◆ There are tests to determine if an n × n matrix has an inverse, and formulas for computing an inverse if it exists. Many of these are programmed into calculators and computer algebra systems. We will be content to give the results for the case of 2 × 2 matrices. e f a b . has an inverse B = Suppose that the 2 × 2 matrix A = g h c d Then 1 0 e f a b = 0 1 g h c d and we have two pairs of equations ae + bg = 1 ce + dg = 0 and a f + bh = 0 c f + dh = 1. When we solve the first pair for e and g, we find that we must divide by ad − bc. This can only be done if ad − bc ≠ 0. The same condition is needed when solving the second pair for f and h. The results of solving for e, f, g, and h are summarized in Theorem 4. 1.5 Matrices THEOREM 4 A matrix A = a b c d 37 has a unique inverse if and only if ad − bc ≠ 0. In this case we have ⎡ −1 A ⎢ ⎢ =⎢ ⎣ d ad − bc − c ad − bc b − ad − bc ⎤ ⎥ ⎥ ⎥. ⎦ a ad − bc ■ Boolean Matrix Operations A Boolean matrix (also called a bit matrix) is an m × n matrix whose entries are either zero or one. We shall now define three operations on Boolean matrices that have useful applications in Chapter 4. Let A = [aij] and B = [bij] be m × n Boolean matrices. We define A ∨ B = C = [cij], the join of A and B, by 1 if aij = 1 or bij = 1 cij = 0 if aij and bij are both 0 and A ∧ B = D = [dij], the meet of A and B, by 1 if aij = 1 and bij are both 1 dij = 0 if aij = 0 or bij = 0. Example 12 Note that these operations are only possible when A and B have the same size, just as in the case of matrix addition. Instead of adding corresponding elements in A and B, to compute the entries of the result, we simply examine the corresponding elements for particular patterns. ⎡ ⎤⎡ ⎤ 1 0 1 1 1 0 ⎢ 0 1 1 ⎥⎢ 1 0 1 ⎥ Let A = ⎣ and B = ⎣ . 1 1 0 ⎦ 0 0 1 ⎦ 0 0 0 1 1 0 (a) Compute A ∨ B. (b) Compute A ∧ B. Solution (a) Let A ∨ B = [cij]. Then, since a43 and b43 are both 0, we see that c43 = 0. In all other cases, either aij or bij is 1, so cij is also 1. Thus ⎤ ⎡ 1 1 1 ⎢ 1 1 1 ⎥ . A∨B=⎣ 1 1 1 ⎦ 1 1 0 (b) Let A ∧ B = [dij]. Then, since a11 and b11 are both 1, d11 = 1, and since a23 and b23 are both 1, d23 = 1. In all other cases, either aij or bij is 0, so dij = 0. Thus ⎡ ⎤ 1 0 0 ⎢ 0 0 1 ⎥ A∧B=⎣ . 0 0 0 ⎦ 0 0 0 ◆ 38 Chapter 1 Fundamentals Finally, suppose that A = [aij] is an m × p Boolean matrix and B = [bij] is a p×n Boolean matrix. Notice that the condition on the sizes of A and B is exactly the condition needed to form the matrix product AB. We now define another kind of product. The product of A and B, denoted A ⊙ B, is the m ×n Boolean matrix Boolean C = [cij] defined by cij = 1 if aik = 1 and bkj = 1 for some k, 1 ≤ k ≤ p; 0 otherwise. This multiplication is similar to ordinary matrix multiplication. The preceding formula states that for any i and j the element cij of C = A ⊙ B can be computed in the following way, as illustrated in Figure 1.21. (Compare this with Figure 1.20.) 1. Select row i of A and column j of B, and arrange them side by side. 2. Compare corresponding entries. If even a single pair of corresponding entries consists of two 1's, then cij = 1. If this is not the case, then cij = 0. ⎡ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎣ a11 a12 ... a1p a21 .. . ai1 .. . a22 .. . ai2 .. . ... a2p .. . aip .. . am1 am2 . . . amp ... ⎤ ⎡ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎢ ⎦ b11 ⎢ b21 ⎢ .. ⎣ . bp1 ⎤ b12 . . . b1j ... b1n b22 .. . . . . b2j .. . ... b2n .. bp2 . . . bpj . . . bpn ai1 b1j ai2 . . b2j .. aip bpj ⎡ c11 ⎥ ⎥ ⎥ ⎢ c21 ⎢ = ⎢ .. ⎥ ⎣ . ⎦ cm1 ⎤ c12 ... c1n c22 .. ... c2n .. cm2 .. cmn cij ⎥ ⎥ ⎥ ⎥ ⎦ If any corresponding pair of entries are both equal to 1, then cij = 1; otherwise cij = 0. Figure 1.21 We can easily perform the indicated comparisons and checks for each position of the Boolean product. Thus, at least for human beings, the computation of elements in A ⊙ B is considerably easier than the computation of elements in AB. ⎡ Example 13 1 ⎢0 Let A = ⎣ 1 0 Solution 1 1 1 0 ⎤ ⎡ ⎤ 0 1 0 0 0 0 ⎢ and B = ⎣ 0 1 1 0 ⎦. Compute A ⊙ B. 0⎦ 1 0 1 1 1 Let A ⊙ B = [eij]. Then e11 = 1, since row 1 of A and column 1 of B each have a 1 as the first entry. Similarly, e12 = 1, since a12 = 1 and b22 = 1; that is, the first row of A and the second column of B have a 1 in the second position. In a similar way we see that e13 = 1. On the other hand, e14 = 0, since row 1 of A and column 1.5 Matrices 39 4 of B do not have common 1's in any position. Proceeding in this way, we obtain ⎡ ⎤ 1 1 1 0 ⎢ 0 1 1 0 ⎥ A ⊙ B=⎣ . 1 1 1 0 ⎦ 1 0 1 1 ◆ The following theorem, whose proof is left as an exercise, summarizes the basic properties of the Boolean matrix operations just defined. THEOREM 5 1.5 Exercises 1. Let A = ⎡ −2 1 3 4 2 C=[5 2 3 6 0 If A, B, and C are Boolean matrices of compatible sizes, then 1. (a) A ∨ B = B ∨ A (b) A ∧ B = B ∧ A 2. (a) (A ∨ B) ∨ C = A ∨ (B ∨ C) (b) (A ∧ B) ∧ C = A ∧ (B ∧ C) 3. (a) A ∧ (B ∨ C) = (A ∧ B) ∨ (A ∧ C) (b) A ∨ (B ∧ C) = (A ∨ B) ∧ (A ∨ C) 4. (A ⊙ B) ⊙ C = A ⊙ (B ⊙ C) ⎤ ⎡ 3 5 , B = ⎣ −2 ⎦, and 2 4 ⎦ 4 −1 . 8 (b) What is b11 , b31 ? (c) What is c13 , c23 , c33 ? (d) List the elements on the main diagonal of C. 2. Which of the following are diagonal matrices? ⎡ ⎤ 3 0 0 2 3 0 ⎤ (b) B = ⎣ 0 −2 (a) A = 0 0 0 0 5 ⎡ ⎤ 0 0 0 (c) C = ⎣ 0 0 0 ⎦ 0 0 0 ⎤ ⎡ 2 6 −2 0 ⎦ (d) D = ⎣ 0 −1 0 0 3 ⎤ ⎡ 4 0 0 (e) E = ⎣ 0 4 0 ⎦ 0 0 4 a+b c+d 4 6 3. If = , find a, b, c, and d. c−d a−b 10 2 a + 2b 2c + d 2a − b c − 2d = 2 4 1 1 C=[4 3 ⎤ 3 E=[5 0 −2 2 1 A= ⎡ (a) What is a12 , a22 , a23 ? 4. If In Exercises 5 through 10, let 4 4 −2 , find a, b, c, and d. −3 2 4 1 ⎡ ⎤ 0 1 3 , B = ⎣ 1 2 ⎦, −2 2 3 ⎤ 3 −3 2 5 ⎦, D = , 4 1 2 ⎤ −1 −2 3 ⎦ −3 , F = . 4 5 2 5. If possible, compute each of the following. (b) AB (a) C + E (d) AB + DF (c) CB + F 6. If possible, compute each of the following. (a) A(BD) and (AB)D (b) A(C + E) and AC + AE (c) FD + AB 7. If possible, compute each of the following. (a) EB + FA (b) A(B + D) and AB + AD (d) AC + DE (c) (F + D)A 8. If possible, compute each of the following. (a) AT and (AT)T (b) (C + E)T and CT + ET (d) (BT C) + A (c) (AB)T and BT AT 9. If possible, compute each of the following. (b) (BC)T and CT BT (a) AT (D + F) ■ 40 Chapter 1 Fundamentals (c) (BT + A)C 10. Compute D3 . (d) (DT + E)F 11. Let A be an m × n matrix. Show that Im A = AIn = A. (Hint: Choose a generic element of Im A.) 2 1 −1 2 12. Let A = and B = . Show that 3 −2 3 4 AB = BA. ⎤ ⎡ 3 0 0 0 ⎦. 13. Let A = ⎣ 0 −2 0 0 4 (b) What is Ak ? (a) Compute A3 . 14. Show that A0 = 0 for any matrix A. 15. Show that InT = In . 16. (a) Show that if A has a row of zeros, then AB has a corresponding row of zeros. (Hint: Use the generic element definition of AB given in this section.) (b) Show that if B has a column of zeros, then AB has a corresponding column of zeros. 17. Show that the jth column of the matrix product AB is equal to the matrix product ABj , where Bj is the jth column of B. 18. If 0 is the 2 × 2 zero matrix, find two 2 × 2 matrices A and B, with A ≠ 0 and B ≠ 0, such that AB = 0. 0 1 19. If A = , show that A2 = I2 . 1 0 0 a 20. Determine all 2 × 2 matrices A = such that b c A2 = I2 . 21. Let A and B be symmetric matrices. (a) Show that A + B is also symmetric. (b) Is AB also symmetric? 22. Let A be an n × n matrix. (a) Show that AAT and AT A are symmetric. (b) Show that A + AT is symmetric. 23. Prove Theorem 3. [Hint: For part (c), show that the i, jth element of (AB)T equals the i, jth element of BT AT .] 24. Let A be a symmetric 2 × 2 matrix that has an inverse. Must A−1 also be symmetric? Explain your reasoning. 25. Find the inverse of each matrix. 2 1 −3 (b) (a) 5 7 0 6 5 (c) 4 −2 26. Find the inverse of each matrix. −8 3 3 (b) (a) 9 −2 −1 −2 −9 (c) −6 −4 4 9 4 10 For Exercises 27 and 28, let ⎤ ⎡ ⎤ ⎡ 0.2 0.4 0.2 1 0 1 0.2 0.6 ⎦ , A = ⎣ 0 0 1 ⎦ , B = ⎣ −0.4 0.2 −0.6 0.2 1 1 1 ⎡ ⎤ ⎡ ⎤ 1 −1 0 2 −1 1 0 1 ⎦ , and D = ⎣ 1 0 −1 ⎦ . C = ⎣ −1 0 1 0 1 1 1 27. (a) Verify that C is an inverse of A. (b) Verify that D is an inverse of B. 28. Determine whether CD is the inverse of AB. Explain your reasoning. 29. Show that if A and B are n × n matrices and A−1, B−1 both exist, then (AB)−1 = (B−1 A−1). In Exercises 30 and 31, compute A ∨ B, A ∧ B, and A ⊙ B for the given matrices A and B. 1 0 1 1 30. (a) A = , B= 0 1 0 1 1 1 0 0 , B= (b) A = 0 1 1 1 1 1 0 0 , B= (c) A = 1 1 1 0 ⎤ ⎡ ⎤ ⎡ 1 1 1 1 0 0 31. (a) A = ⎣ 0 1 1 ⎦ , B = ⎣ 0 0 1 ⎦ 1 0 1 1 0 0 ⎤ ⎡ ⎤ ⎡ 0 1 1 0 0 1 (b) A = ⎣ 1 1 0 ⎦ , B = ⎣ 1 1 0 ⎦ 1 0 1 1 0 0 ⎤ ⎡ ⎤ ⎡ 1 1 1 1 0 0 (c) A = ⎣ 0 0 1 ⎦ , B = ⎣ 1 1 1 ⎦ 1 0 0 1 0 1 In Exercises 32 and 33, let F = [fij] be a p × q matrix. 32. (a) (b) (c) (d) 33. (a) (b) (c) (d) (e) (f) Give a generic element in row k of F. What is the largest value of j? Give a generic element in column l of F. What is the largest value of i? Give the element in the lower left corner of F. Give the element in the upper left corner of F. Give the element in the lower right corner of F. Give the element in the upper right corner of F. Give a generic element in row r of FT . Give a generic element on the diagonal of F. In Exercises 34 and 35, let M = [mij] , N = [nij] , and P = [pij] be square matrices of the same size. Use the entries of M, N, and P to represent the entry in the ij-position of each of the following. 34. (a) P + M + N (b) PT + M (c) NT (d) NM 35. (a) PM (b) MP (c) P(M + N) (d) MN + PN 1.6 Mathematical Structures In Exercises 36 and 37, let M = [mij] , N = [nij] , and P = [pij] be square Boolean matrices of the same size. Use the entries of M, N, and P to represent the entry in the ijposition of each of the following. 36. (a) M ∧ P (b) P ∨ N (c) (M∧N)∨(M∧P) 37. (a) M ⊙ P (b) P ⊙ M (c) (M ⊙ N) ⊙ P 38. Complete the following proofs. (a) A ∨ A = A. Proof: Let bij be an element of A ∨ A. If aij = 0, then bij = , because . If bij = 1, then aij = because . Hence bij = aij for each i, j pair. (b) A ∧ A = A. Proof: Let bij be an element of A ∧ A. If bij = 0, then . If bij = 1, then . (Explain.) Hence bij = aij for each i, j pair. 39. Show that A ∨ B = B ∨ A. 40. Show that A ∧ B = B ∧ A. 41. Show that A ∨ (B ∨ C) = (A ∨ B) ∨ C. 42. Show that A ∧ (B ∧ C) = (A ∧ B) ∧ C. 43. Show that A ⊙ (B ⊙ C) = (A ⊙ B) ⊙ C. 44. Show that A ∧ (B ∨ C) = (A ∧ B) ∨ (A ∧ C). 45. Show that A ∨ (B ∧ C) = (A ∨ B) ∧ (A ∨ C). 46. What fact does Example 8 illustrate? 47. Let A = [aij] and B = [bij] be two n × n matrices and let C = [cij] represent AB. Prove that if k is an integer and k | aij for all i, j, then k | cij for all i, j. 48. Let p be a prime number with p > 2, and let A and B be matrices all of whose entries are integers. Suppose that p divides all the entries of A+B and all the entries of A−B. 1.6 41 Prove that p divides all the entries of A and all the entries of B. Another operation on matrices is scalar multiplication. Let k be a real number and A = [aij] be an m × n matrix. The result of multiplying A by the scalar k is the matrix kA = [kaij]. For Exercises 49 through 53, use the definition of scalar multiplication and the matrices given. ⎤ ⎡ ⎤ ⎡ 2 4 −6 2 −3 −1 0 9 ⎦, 5 2 ⎦, B = ⎣ 4 A=[0 7 −1 3 4 −4 6 and ⎡ 4 C=[3 −2 ⎤ 0 1 ⎦ 5 49. Compute each of the following. (b) 5B (c) −1C (a) 3A 50. Compute each of the following. (b) 3A + 3B (a) 3(A + B) (d) A(−2C) (c) −2(AC) 51. Show that scalar multiplication has the following property. k(A + B) = kA + kB 52. Show that scalar multiplication has the following property. k(AB) = (kA)B = A(kB) 53. Let A be an m × n matrix. Find a matrix K such that KA = kA, for a fixed k. Mathematical Structures Several times in this chapter, we have defined a new kind of mathematical object; for example, a set or a matrix. Then notation was introduced for representing the new type of object and a way to determine whether two objects are the same was described. Next we classified objects of the new type; for example, finite or infinite for sets, and Boolean or symmetric for matrices. And then operations were defined for the objects and the properties of these operations were examined. Such a collection of objects with operations defined on them and the accompanying properties form a mathematical structure or system. In this book we deal only with discrete mathematical structures. Example 1 The collection of sets with the operations of union, intersection, and complement and their accompanying properties is a (discrete) mathematical structure. We de◆ note this structure by (sets, ∪, ∩, ‾). Example 2 The collection of 3 × 3 matrices with the operations of addition, multiplication, and transpose is a mathematical structure denoted by (3 × 3 matrices, +, ∗, T). ◆ 42 Chapter 1 Fundamentals An important property we have not identified before is closure. A structure is closed with respect to an operation if that operation always produces another member of the collection of objects. Example 3 The structure (5 × 5 matrices, +, ∗, T) is closed with respect to addition because the sum of two 5 × 5 matrices is another 5 × 5 matrix. ◆ Example 4 The structure (odd integers, +, ∗) is not closed with respect to addition. The sum of two odd integers is an even integer. This structure does have the closure property for multiplication, since the product of two odd numbers is an odd number. ◆ An operation that combines two objects is a binary operation. An operation that requires only one object is a unary operation. Binary operations often have similar properties, as we have seen earlier. Example 5 (a) Set intersection is a binary operation since it combines two sets to produce a new set. ◆ (b) Producing the transpose of a matrix is a unary operation. Common properties have been given names. For example, if the order of the objects does not affect the outcome of a binary operation, we say that the operation is commutative. That is, if x □ y = y □ x, where □ is some binary operation, □ is commutative. Example 6 (a) Join and meet for Boolean matrices are commutative operations. A∨B=B∨A and A ∧ B = B ∧ A. (b) Ordinary matrix multiplication is not a commutative operation. AB ≠ BA. ◆ Note that when we say an operation has a property, this means that the statement of the property is true when the operation is used with any objects in the structure. If there is even one case when the statement is not true, the operation does not have that property. If □ is a binary operation, then □ is associative or has the associative property if (x □ y) □ z = x □ (y □ z). Example 7 Set union is an associative operation, since (A ∪ B) ∪ C = A ∪ (B ∪ C) is always true. ◆ If a mathematical structure has two binary operations, say □ and ▽, a distributive property has the following pattern: x □ (y ▽ z) = (x □ y) ▽ (x □ z). We say that "□ distributes over ▽." Example 8 (a) We are familiar with the distributive property for real numbers; if a, b, and c are real numbers, then a · (b + c) = a · b + a · c. Note that because we have an agreement about real number arithmetic to multiply before adding, parentheses are not needed on the right-hand side. (b) The structure (sets, ∪, ∩, ‾) has two distributive properties: A ∪ (B ∩ C) = (A ∪ B) ∩ (A ∪ C) and A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C). ◆ 1.6 Mathematical Structures 43 Several of the structures we have seen have a unary operation and two binary operations. For such structures we can ask whether De Morgan's laws are properties of the system. If the unary operation is ◦ and the binary operations are □ and ▽, then De Morgan's laws are (x □ y)◦ = x◦ ▽ y◦ and (x ▽ y)◦ = x◦ □ y◦. Example 9 (a) As we saw in Section 1.2, sets satisfy De Morgan's laws for union, intersection, and complement: ‾(A ∪ B) = ‾A ∩ ‾B and ‾(A ∩ B) = ‾A ∪ ‾B. (b) The structure (real numbers, +, ∗, √) does not satisfy De Morgan's laws, since √(x + y) ≠ √x ∗ √y. ◆ A structure with a binary operation □ may contain a distinguished object e, with the property x □ e = e □ x = x for all x in the collection. We call e an identity for □. In fact, an identity for an operation must be unique. THEOREM 1 If e is an identity for a binary operation □, then e is unique. Proof Assume another object i also has the identity property, so x □ i = i □ x = x. Then e □ i = e, but since e is an identity for □, i □ e = e □ i = i. Thus, i = e. ■ Therefore there is at most one object with the identity property for □. This is one of our first examples of a proof that does not proceed directly. We assumed that there were two identity elements and showed that they were in fact the same element. Example 10 For (n × n matrices, +, ∗, T), In is the identity for matrix multiplication and the ◆ n × n zero matrix is the identity for matrix addition. If a binary operation □ has an identity e, we say y is a □-inverse of x if x □ y = y □ x = e. THEOREM 2 Example 11 Example 12 If □ is an associative operation and x has a □-inverse y, then y is unique. Proof Assume there is another □-inverse for x, say z. Then (z □ x) □ y = e □ y = y and z □ (x □ y) = z □ e = z. Since □ is associative, (z □ x) □ y = z □ (x □ y) and so y = z. ■ (a) In the structure (3 × 3 matrices, +, ∗, T), each matrix A = [aij] has a +-inverse, or additive inverse, −A = [−aij]. (b) In the structure (integers, +, ∗), only the integers 1 and −1 have multiplicative inverses. ◆ Let □, ▽, and ◦ be defined for the set {0, 1} by the following tables. □ 0 1 0 0 1 1 1 0 ▽ 0 1 0 0 0 1 0 1 x◦ 0 1 x 1 0 Thus 1 □ 0 = 1, 0 ▽ 1 = 0, and 1◦ = 0. Determine if each of the following is true for ({0, 1}, □, ▽, ◦). (a) □ is commutative. (b) ▽ is associative. 44 Chapter 1 Fundamentals (c) De Morgan's laws hold. (d) Two distributive properties hold for the structure. Solution (a) The statement x □ y = y □ x must be true for all choices of x and y. Here there is only one case to check: Is 0 □ 1 = 1 □ 0 true? Since both 0 □ 1 and 1 □ 0 are 1, □ is commutative. (b) The eight possible cases to be checked are left as an exercise. See Exercise 6(b). 0◦ ▽ 0◦ = 1 ▽ 1 = 1 (c) (0 □ 0)◦ = 0◦ = 1 ◦ ◦ (0 □ 1)◦ = 1◦ = 0 0◦ ▽ 1◦ = 1 ▽ 0 = 0 ◦ ◦ (1 □ 0)◦ = 1◦ = 0 1◦ ▽ 0◦ = 0 ▽ 1 = 0 (1 □ 1)◦ = 0◦ = 1 1◦ ▽ 1◦ = 0 ▽ 0 = 0 The last pair shows that De Morgan's laws do not hold in this structure. (d) One possible distributive property is x □ (y ▽ z) = (x □ y) ▽ (x □ z). We must check all possible cases. One way to organize this is shown in a table. x y z y▽z x□(y▽z) x□y x□z (x□y)▽(x□z) 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 1 0 1 1 0 0 1 1 0 0 1 0 1 0 1 1 1 1 0 0 0 0 (A) (B) Since columns (A) and (B) are not identical, this possible distributive property does not hold in this structure. The check for the other distribu◆ tive property is Exercise 7. In later sections, we will find it useful to consider mathematical structures themselves as objects and to classify them according to the properties associated with their operations. 1.6 Exercises In Exercises 1 and 2, tell whether the structure has the closure property with respect to the operation. 1. (a) (sets, ∪, ∩, ‾) union (b) (3 × 5 matrices, +, ∗, T) multiplication transpose In Exercises 3 and 4, tell whether the structure has the closure property with respect to the operation. 3. (a) (integers, +, −, ∗, ÷) ∗ (b) (A∗, catenation) addition 5. Show that ⊕ is a commutative operation for sets. 6. Using the definitions in Example 12, (a) show that □ is associative. (b) Show that ▽ is associative. complement 2. (a) (4 × 4 matrices, +, ∗, T) (b) (prime numbers, +, ∗) 7. Using the definitions in Example 12, determine if the other possible distributive property holds. 8. Give the identity element, if one exists, for each binary operation in the given structure. √) (a) (real numbers, +, ∗, (b) (sets, ∪, ∩, ‾) division (c) ({0, 1}, □, ▽, ◦) as defined in Example 12 catenation 4. (a) (n × n Boolean matrices, ∨, ∧, ⊙, T) meet (d) (subsets of a finite set A, ⊕, ‾) 1.6 Mathematical Structures 9. Give the identity element, if one exists, for each binary operation in the structure (5 × 5 Boolean matrices, ∨, ∧, ⊙). In Exercises 10 through 16, use the structure S = (n × n diagonal matrices, +, ∗, T). 10. Show that S is closed with respect to addition. 11. Show that S is closed with respect to multiplication. 12. Show that S is closed with respect to the transpose operation. 13. Does S have an identity for addition? If so, what is it? 14. Does S have an identity for multiplication? If so, what is it? 15. Let A be an n × n diagonal matrix. Describe the additive inverse of A. 16. Let A be an n × n diagonal matrix. Describe the multiplicative inverse of A. In Exercises 17 through 23, use the structure R = (M, +, ∗, T), where M is the set of matrices of the form a 0 0 0, where a is a real number. 17. Show that R is closed with respect to addition. 18. Show that R is closed with respect to multiplication. 19. Show that R is closed with respect to the transpose operation. 20. Does R have an identity for addition? If so, what is it? 21. Does R have an identity for multiplication? If so, what is it? 22. Let A be an element of M. Describe the additive inverse for A. 23. Let A be an element of M. Describe the multiplicative inverse for A. In Exercises 24 through 28, let R = (Q, □), where x □ y = x+y . Determine which of the following properties hold for this structure: 24. 25. 26. 27. 28. 29. Closure Commutative Associative An identity element An inverse for every element Let R = (2 × 1 matrices, ▽), where w x +w x = . y z y+z ▽ 1 45 Determine which of the following properties hold for this structure. (b) Commutative (a) Closure (c) Associative 30. Let R be as in Exercise 29. Determine which of the following properties hold for this structure. (a) An identity element (b) An inverse for every element 31. Let S = (1 × 2 matrices, □), where x y □ y+z w z = x + w . Determine which of the following properties hold for this structure. (b) Commutative (a) Closure (c) Associative 32. Let S be as in Exercise 31. Determine which of the following properties hold for this structure. (a) An identity element (b) An inverse for every element 33. (a) Give a symbolic statement of the distributive property for scalar multiplication over ▽ as defined in Exercise 29. (b) Is the distributive property in part (a) a property of R? 34. (a) Give a symbolic statement of the distributive property for scalar multiplication over □ as defined in Exercise 31. (b) Is the distributive property in part (a) a property of S? 35. For a Boolean matrix B, we define comp B to be the matrix formed by changing each 0 entry of B to 1 and each 1 entry of B to 0. Let R = (5 × 5 Boolean matrices, ∧, ∨, comp). Do De Morgan's laws hold for R? Justify your answer. The properties of a mathematical structure can be used to rewrite expressions just as is done in ordinary algebra. In Exercises 36 through 39, rewrite the given expression to produce the requested result. 36. (A ∪ B) ∩ (A ∪ ‾B) 37. (A ∩ B) ∩ ‾A one set, no operations two sets, two operations 38. (A ∪ B) ∪ (A ∩ B) two sets, two operations 39. (A ∪ B) ∩ (A ∪ B) one set, no operations 46 Chapter 1 Fundamentals Tips for Proofs Many exercises in this chapter ask that you show, prove, or verify a statement. To show or prove a statement means to give a written explanation demonstrating that the statement is always true. To verify a statement, in this book, means to check its truth for a particular case; see, for example, Section 1.2, Exercises 16 and 24. Most proofs required in this chapter proceed directly from the given conditions using definitions and previously proven facts; an example is Section 1.4, Theorem 2. A powerful tool for constructing a proof is to choose a generic object of the type in the statement and to see what you know about this object. Remember that you must explain why the statement is always true, so choosing a specific object will only verify the statement for that object. The most common way to show that two sets are equal is to show each is a subset of the other (Section 1.2, Theorem 1). In proving statements about sets or matrix operations, try to work at the level of object names rather than at the element or entry-level. For example, Section 1.5, Exercise 22 is more easily proved by using the facts that if A is symmetric, then AT = A and Theorem 3 rather than by using the fact that if A = [aij] is symmetric, then aij = aji for each i and j. One other style of direct proof is seen in Section 1.6, Example 12. Sometimes we show the statement is always true by examining all possible cases. Key Ideas for Review • ∅ (empty set): the set with no elements 1 if x ∈ A • Characteristic function of a set A: fA (x) = 0 if x ∉ A • Equal sets: sets with the same elements • Countable set: a set that corresponds to a sequence • A ⊆ B (A is a subset of B): Every element of A is an element of B. • Word: finite sequence of elements of A • Set: a well-defined collection of objects • • • • • • • |A| (cardinality of A): the number of elements of A Infinite set: see page 4 P(A) (power set of A): the set of all subsets of A A ∪ B (union of A and B): {x | x ∈ A or x ∈ B} A ∩ B (intersection of A and B):{x | x ∈ A and x ∈ B} Disjoint sets: two sets with no elements in common A − B (complement of B with respect to A): {x | x ∈ A and x ∉ B} • ‾A (complement of A): {x | x ∉ A} • Algebraic properties of set operations: see pages 8–9 • Regular expression: see page 17 • Theorem: If n and m are integers and n > 0, we can write • • • • • Theorem (the addition principle): If A and B are finite sets, • • Theorem (the three-set addition principle): If A, B, and C • then |A ∪ B| = |A| + |B| − |A ∩ B|. are finite sets, then |A ∪ B ∪ C| = |A| + |B| + |C| − |A ∩ B| − |A ∩ C| − |B ∩ C| + |A ∩ B ∩ C|. • Inclusion-exclusion principle: see page 9 • • • Sequence: list of objects arranged in a definite order • • Recursive formula: formula that uses previously defined • terms • Explicit formula: formula that does not use previously defined terms • Linear array: see page 14 • • m = qn + r for integers q and r with 0 ≤ r < n. Moreover, there is just one way to do this. GCD(a, b): d = GCD(a, b) if d | a, d | b, and d is the largest common divisor of a and b. Theorem: If d is GCD(a, b), then (a) d = sa + tb for some integers s and t. (b) If c | a and c | b, then c | d. Relatively prime: two integers a and b with GCD(a, b) = 1 Euclidean algorithm: method used to find GCD(a, b); see pages 22–23 LCM(a, b): c = LCM(a, b) if a | c, b | c, and c is the smallest common multiple of a and b Theorem: GCD(a, b) · LCM(a, b) = ab Base b expansion of a number: see page 27 Cryptology: the science of producing and deciphering secret codes Bacon's code: see page 29 Steganography: the science of concealment of coded information mod-n function: fn (z) = r , where r is the remainder when z is divided by n Matrix: rectangular array of numbers Chapter 1 Self-Test • Size of a matrix: A is m × n if it has m rows and n columns • Diagonal matrix: a square matrix with zero entries off the main diagonal • Equal matrices: matrices of the same size whose corre• • • • • • • • • sponding entries are equal A + B: the matrix obtained by adding corresponding entries of A and B Zero matrix: a matrix all of whose entries are zero AB: see page 33 In (identity matrix): a square matrix with ones on the diagonal and zeros elsewhere Array of dimension 2: see page 34 AT : the matrix obtained from A by interchanging the rows and columns of A Symmetric matrix: AT = A Inverse of a matrix: see page 36 Boolean matrix: a matrix whose entries are either one or zero 47 • A ∨ B: see page 37 • A ∧ B: see page 37 • A ⊙ B: see page 37 • Properties of Boolean matrix operations: see page 39 • Mathematical structure: a collection of objects with opera- tions defined on them and the accompanying properties • Binary operation: an operation that combines two objects • Unary operation: an operation that requires only one object • Closure property: each application of the operation pro- duces another object in the collection • Associative property: (x □ y) □ z = x □ (y □ z) • Distributive property: x □ (y ▽ z) = (x □ y) ▽ (x □ z) • De Morgan's laws: (x □ y)◦ = x◦ ▽ y◦ and (x ▽ y)◦ = x◦ □ y◦ • Identity for □: an element e such that x □ e = e □ x = x for all x in the structure • □-inverse for x: an element y such that x □ y = y □ x = e, where e is the identity for □ Chapter 1 Self-Test 1. What kind of mathematical object is P(A)? 2. What kind of mathematical object is |P(A)|? 3. What kind of mathematical object is LCM(a, b)? 4. What kind of mathematical object is kA? 5. What are the components of a mathematical structure? 6. Let A = {x | x is a real number and 0 < x < 1}, B = {x | x is a real number and x^2 + 1 = 0}, C = {x | x = 4m, m ∈ Z}, D = {0, 2, 4, 6, . . . }, and E = {x | x ∈ Z and x^2 ≤ 100}. (a) Identify the following as true or false. (i) C ⊆ D (iii) {4, 16} ⊆ E (v) B ⊆ ∅ (ii) {4, 16} ⊆ C (iv) D ⊆ D (b) Identify the following as true or false. (i) (ii) (iv) (v) C ∩ E (C ∪ E) ∅ ⊆ (A ∪ B) (iii) C ∩ D = D C∪E⊆D A∩D ⊆ A∩C 7. Let A = {x | x = 2n, n ∈ Z+}, B = {x | x = 2n + 1, n ∈ Z+}, C = {x | x = 4n, n ∈ Z+}, and D = {x | x(x^2 − 6x + 8) = 0, x ∈ Z}. Use Z as the universal set and find (b) A (a) A ∪ B (c) (A ∩ D) ⊕ (A ∩ B) (d) A ∪ C (e) A − C 8. Draw a Venn diagram to represent (a) A ∩ B and (b) A ∩ B. 9. Under what conditions will A ∩ B = A ∪ B? 10. Suppose that 109 of the 150 mathematics students at Verysmall College take at least one of the following computer languages: PASCAL, BASIC, C++ . Suppose 45 study BASIC, 61 study PASCAL, 53 study C++ , 18 study BASIC and PASCAL, 15 study BASIC and C++ , and 23 study PASCAL and C++ . (a) How many students study all three languages? (b) How many students study only BASIC? (c) How many students do not study any of the languages? 11. Define a sequence as follows: a0 = 0, a1 = 0, an = 1 − 3an−1 + 2an−2 . Compute the first six terms of this sequence. 12. Let U = {a, b, c, d, e, f, g, h, i, j}, A = {a, b, d, f }, B = {a, b, c, h, j}, C = {b, c, f, h, i}, and D = {g, h}. Represent each of the following sets by an array of zeros and ones. (b) A ∩ B (a) A ∪ B (d) (A ∩ B) ∪ D (c) A ∩ (B ∪ C) 13. Let I = {a, b, c}. In each part that follows is listed a string in I∗ and a regular expression over I . For each, state whether the string belongs to the regular set corresponding to the expression. (b) acbb ((acb) ∨ b)∗ (a) ab a ∗ bc∗ (d) abaca (ab)∗ ac (c) bc ((ab∗) ∨ c) 14. Use the Euclidean algorithm to compute GCD(4389, 7293) and write it as s(7293) + t (4389). 48 Chapter 1 Fundamentals 2 6 4 2 0 and B = . Compute, −1 3 2 −3 1 if possible, each of the following. (b) BA (c) BT (a) AB 15. Let A = (d) A + B (g) B−1 A (e) AT B (f) B−1 ⎡ ⎤ ⎡ 1 0 1 1 1 0 16. Let C = ⎣ 1 1 0 ⎦ and D = ⎣ 0 1 0 ⎦. 0 1 1 1 1 0 Compute each of the following. (b) C ∨ D (c) C ∧ D (a) C ⊙ D 17. Let S = (2 × 2 Boolean matrices, ∧, ∨, ‾) and A be a 2 × 2 Boolean matrix. Describe the ∧-inverse of A in S. Experiment 1 In many voting procedures the rules are one person, one vote, and a simple majority is required to elect a candidate or to pass a motion. But it is not unusual to have a procedure where individual voters have more than one vote or where something other than a simple majority is required. An example of such a situation is when not all shareholders in a company own the same number of shares, and each shareholder has as many votes as shares. Does a shareholder with twice as many shares as another have twice as much control, or power, over the company?

In this experiment you will investigate this question and some related ones. First, we begin with some definitions. The number of votes that a voter has is called the voter's weight. Here only counting numbers can be weights. The total number of votes needed to elect a candidate or to pass a motion is the quota. The collection of the quota and the individual weights for all voters is called a weighted voting system. If the voters are designated v1 , v2 , . . . , vk with corresponding weights w1 , w2 , . . . , wk and q is the quota, then the weighted voting system may be conveniently represented by [q : w1 , w2 , . . . , wk]. For ease of computations, the weights are usually listed from largest to smallest. 1. For the weighted voting system [9 : 9, 4, 2, 1], what is the quota? How many voters are there? What is the total number of votes available? 2. In a weighted voting system [q : w1 , w2 , . . . , wk], what are the restrictions on the possible values of q? Explain each restriction. 3. For the weighted voting system [9 : 9, 4, 2, 1], describe how much power voter v1 has. Such a voter is called a dictator. Why is this appropriate? Could a system have two dictators? Explain why or why not. 4. For [8 : 5, 3, 2, 1], is v1 a dictator? Describe v1 's power relative to the other voters. More interesting cases arise when the power of each voter is not so obvious as in these first examples. One way to measure a voter's power was developed by John Banzhaf in 1965. A coalition is a subset of the voters in a weighted voting system. If the total number of votes controlled by the members of the coalition equals or exceeds the quota, we call the coalition a winning coalition. If not, this is a losing coalition. 5. (a) List all the coalitions for [9 : 9, 4, 2, 1]. Which of these are winning coalitions? (b) List all the winning coalitions for [8 : 5, 3, 2, 1]. Banzhaf's idea is to measure a voter's power by examining how many times removal of that voter from a coalition would change the coalition from winning to losing. Consider the system [7 : 5, 3, 2]. The winning coalitions are {v1 , v2 }, {v1 , v3 }, {v2 , v3 }, and {v1 , v2 , v3 }. Each member of the first three coalitions has the power to change it from winning to losing, but no member has this power in the last Experiment 1 49 coalition. All together there are six opportunities for change. Each of v1 , v2 , v3 has two of these opportunities. The Banzhaf power distribution for this system is v1 : 26 , v2 : 26 , v3 : 26 . According to this analysis, all three voters have the same amount of power despite having different weights. The fraction of power assigned to a voter is the voter's Banzhaf power index. 6. Here is a test for Banzhaf's method of measuring power. Calculate the Banzhaf power distribution for [9 : 9, 4, 2, 1]. Explain how the results are consistent with the designation of v1 as a dictator. 7. Calculate the Banzhaf power distribution for [8 : 5, 3, 2, 1]. A voter like v1 that must belong to every winning coalition has veto power in the system. 8. Let [q : 6, 3, 1] be a weighted voting system. (a) Give values for q for which at least one voter has no power. A voter who has no power is called a dummy. (b) Give values for q for which exactly one voter has veto power and identify those voters. (c) Give values for q for which at least two voters have veto power and identify that voter. (d) Give values for q for which at least two voters have veto power and identify the voters. 9. Suppose v1 , v2 , v3 , v4 form a weighted voting system. Is it possible ... Banzhaf power distribution for this system. Would the distribution change if the tie-breaking vote were changed to the coalition containing the senior partner is the winning coalition. Give the Banzhaf power distribution for this system. Would the distribution change if the coalition containing the most junior member is the losing coalition? Explain. 10. Suppose you are the voter with weight one in [8 : 5, 3, 2, 1]. (a) What is your Banzhaf power index? (b) Unhappy with this situation, you offer to buy a vote from one of the other voters. If each is willing to sell and each asks the same price, from whom should you buy a vote and why? Give the Banzhaf power distribution for this system for the resulting weighted voting system. 11. Here is another feature of Banzhaf's way of measuring power. Let [q : w1 , w2 , . . . , wk] be a weighted voting system and n be a positive integer. Prove that the Banzhaf power distributions for [q : w1 , w2 , . . . , wk] and [nq : nw1 , nw2 , . . . , nwk] are the same. 12. We now return to the original question about power. Suppose we have a weighted voting system [q : w1 , w2] with v1 has weight w1 , v2 has weight w2 , and w1 = 2w2 . Construct such a system where the Banzhaf power index of v1 is (a) the same as that of v2 (b) twice that of v2 (c) more than twice that of v2 . CHAPTER 2 Logic Prerequisites: Chapter 1 Logic is the discipline that deals with the methods of reasoning. On an elementary level, logic provides rules and techniques for determining whether a given argument is valid. Logical reasoning is used in mathematics to prove theorems, in computer science to verify the correctness of programs and to prove theorems, in the natural and physical sciences to draw conclusions from experiments, in the social sciences, and in our everyday lives to solve a multitude of problems. Indeed, we are constantly using logical reasoning. In this chapter we discuss a few of the basic ideas. Looking Back In the 1840s Augustus De Morgan, a British mathematician, set out to extend the logic developed by the early Greeks and others and to correct some of the weaknesses in these ideas. De Morgan (1806–1871) was born in India but was educated in England. He taught at London University for many years and was the first to use the word "induction" for a method of proof that had been used in a rather informal manner but put it on a firm rigorous foundation. In 1847, a few years after De Morgan's work on an extended system of logic had appeared, his countryman George Boole published the book entitled The Mathematical Analysis of Logic and followed it up a few years later by the book An Investigation of the Laws of Thought. Boole's objective in these books was at Queen's College in Ireland for many years. Thus, De Morgan started and Boole completed the task of folding a large part of the study of logic into mathematics. We shall study in this chapter and Chapter 8 we shall further examine important applications of the work of Boole to many areas in mathematics and computer science: to investigate the fundamental laws of those operations of the mind by which reasoning is performed; to give expression to them in the symbolical language of a Calculus; and upon this foundation to establish the science of Logic and construct its method.• Boole's work in this area firmly established the point of view that logic should use symbols and that algebraic properties should be studied in logic. George Boole (1815–1864) taught math in Ireland. Augustus De Morgan (1806–1871) Quoted 50 Augustus De Morgan in Victor J. Katz, A History of Mathematics, An Introduction, New York: HarperCollins, 1993, p. 619. George Boole 2.1 Propositions and Logical Operations 2.1 51 Propositions and Logical Operations A statement or proposition is a declarative sentence that is either true or false, but not both. Example 1 Which of the following are statements? (a) (b) (c) (d) (e) (f) (g) The earth is round. 2+3=5 Do you speak English? 3−x =5 Take two aspirins. The temperature on the surface of the planet Venus is 800 ◦ F. The sun will come out tomorrow. Solution (a) and (b) are statements that happen to be true. (c) is a question, so it is not a statement. (d) is a declarative sentence, but not a statement, since it is true or false depending on the value of x. (e) is not a statement; it is a command. (f) is a declarative sentence whose truth or falsity we do not know at this time; however, we can in principle determine if it is true or false, so it is a statement. (g) is a statement since it is either true or false, but not both, although we would have to wait until tomorrow to find out if it is true or false. ◆ Logical Connectives and Compound Statements TABLE 2.1 p ∼p T F F T Example 2 In mathematics, the letters x, y, z, . . . often denote variables that can be replaced by real numbers, and these variables can be combined with the familiar operations +, ×, −, and ÷. In logic, the letters p, q, r, . . . denote propositional variables; that is, variables that can be replaced by statements. Thus we can write p: The sun is shining today. q: It is cold. Statements or propositional variables can be combined by logical connectives to obtain compound statements. For example, we may combine the preceding statements by the connective and to form the compound statement p and q: The sun is shining today and it is cold. The truth value of a compound statement depends only on the truth values of the statements being combined and on the types of connectives being used. We shall look at the most important connectives. If p is a statement, the negation of p is the statement not p, denoted by ∼ p. Thus ∼ p is the statement "it is not the case that p." From this definition, it follows that if p is true, then ∼ p is false, and if p is false, then ∼ p is true. The truth value of ∼ p relative to p is given in Table 2.1. Such a table, giving the truth values of a compound statement in terms of its component parts, is called a truth table. Strictly speaking, not is not a connective, since it does not join two statements, and ∼ p is not really a compound statement. However, not is a unary operation for the collection of statements and ∼ p is a statement if p is. Example 3 Give the negation of the following statements: (a) p: 2 + 3 > 1 (b) q: It is cold. 52 Chapter 2 Logic Solution (a) ∼ p: 2 + 3 is not greater than 1. That is, ∼ p: 2 + 3 ≤ 1. Since p is true in this case, ∼ p is false. (b) ∼q: It is not the case that it is cold. More simply, ∼q: It is not cold. ◆ TABLE 2.2 p q p∧q T T F F T F T F T F F F Example 3 If p and q are statements, the conjunction of p and q is the compound statement "p and q," denoted by p ∧ q. The connective and is denoted by the symbol ∧. In the language of Section 1.6, and is a binary operation on the set of statements. The compound statement p ∧ q is true when both p and q are true; otherwise, it is false. The truth values of p ∧ q in terms of the truth values of p and q are given in the truth table shown in Table 2.2. Observe that in giving the truth table of p ∧ q we need to look at four possible cases. This follows from the fact that each of p and q can be true or false. Form the conjunction of p and q for each of the following. (a) p: It is snowing. (b) p: 2 < 3 (c) p: It is snowing. q: I am cold. q: −5 > −8 q: 3 < 5 Solution (a) p ∧ q: It is snowing and I am cold. (b) p ∧ q: 2 < 3 and −5 > −8 (c) p ∧ q: It is snowing and 3 < 5. ◆ Example 3(c) shows that in logic, unlike in everyday English, we may join two totally unrelated statements by the connective and. If p and q are statements, the disjunction of p and q is the compound statement "p or q," denoted by p ∨ q. The connective or is denoted by the symbol ∨. The compound statement p ∨ q is true if at least one of p or q is true; it is false when both p and q are false. The truth values of p ∨ q are given in the truth table shown in Table 2.3. Example 4 TABLE 2.3 p q p∨q T T F F T F T F T T T F Form the disjunction of p and q for each of the following. (a) p: 2 is a positive integer q: √2 is a rational number. (b) p: 2 + 3 = 5 q: London is the capital of France. Solution (a) p ∨ q: 2 is a positive integer or √2 is a rational number. Since p is true, the disjunction p ∨ q is true, even though q is false. (b) p ∨ q: 2 + 3 = 5 or London is the capital of France. Since both p and q are false, p ∨ q is false. ◆ Example 4(b) shows that in logic, unlike in ordinary English, we may join two totally unrelated statements by the connective or. The connective or is more complicated than the connective and because it is used in two different ways in English. Suppose that we say "I left for Spain on Monday or I left for Spain on Friday." In this compound statement we have the disjunction of the statements p: I left for Spain on Monday and q: I left for Spain on Friday. Of course, exactly one of the two possibilities could have occurred. Both could not have occurred, so the connective or is being used in an exclusive sense. On the other hand, consider the disjunction "I passed mathematics or I failed French." In this case, at least one of the two possibilities occurred. However, both 2.1 Propositions and Logical Operations 53 could have occurred, so the connective or is being used in an inclusive sense. In mathematics and computer science we agree to use the connective or always in the inclusive manner. In general, a compound statement may have many component parts, each of which is itself a statement, represented by some propositional variable. The statement s: p ∨ (q ∧ (p ∨ r)) involves three propositions, p, q, and r, each of which may independently be true or false. There are altogether 23 or 8 possible combinations of truth values for p, q, and r , and a truth table for s must give the truth or falsity of s in all these cases. If a compound statement s contains n component statements, there will need to be 2n rows in the truth table for s. (In Section 3.1 we look at how to count the possibilities in such cases.) Such a truth table may be systematically constructed in the following way. Step 1 The first n columns of the table are labeled by the component propositional variables. Further columns are included for all intermediate combinations of the variables, culminating in a column for the full statement. Step 2 Under each of the first n headings, we list the 2n possible n-tuples of truth values for the n component statements. Step 3 For each of the remaining columns, we compute, in sequence, the remaining truth values. Example 5 p q p∧q ∼p ∼(p∧q) TTFFTFTFTTFFFTFTTTTT ... (1) (2) (3) (2) Make a truth table for the statement (p ∧ q) ∨ (∼ p). Solution Because two propositions are involved, the truth table will have 22 or 4 rows. In the first two columns we list all possible pairs of truth values for p and q. The numbers below the remaining columns show the order in which the columns were filled. ◆ Quantifiers In Section 1.1, we defined sets by specifying a property P(x) that elements of the set have in common. Thus, an element of {x | P(x)} is an object t for which the statement P(t) is true. Such a sentence P(x) is called a predicate, because in English the property is grammatically a predicate. P(x) is also called a propositional function, because each choice of x produces a proposition P(x) that is either true or false. Another use of predicates is in programming. Two common constructions are "if P(x), then execute certain steps" and "while Q(x), do specified actions." The predicates P(x) and Q(x) are called the guards for the block of programming code. Often the guard for a block is a conjunction or disjunction. Example 6 Let A = {x | x is an integer less than 8}. Here P(x) is the sentence "x is an integer less than 8." The common property is "is an integer less than 8." P(1) is the statement "1 is an integer < 8." Since P(1) is true, 1 ∈ A. Example 7 (a) Consider the following program fragment, 1. IF N < 10 THEN a. Replace N with N + 1 b. RETURN Here the statement N < 10 is the guard. The second statement N is: Consider the following program fragment, 1. WHILE t ∈ T and s ∈ S a. PRINT t + s b. RETURN Here the guard is t ∈ T and s ∈ S. ◆ The universal quantification of a predicate P(x) is the statement "For all values of x, P(x) is true." We assume here that only values of x that make sense in P(x) are considered. The universal quantification of P(x) is denoted ∀x P(x). The symbol ∀ is called the universal quantifier. Example 8 (a) The sentence P(x): −(−x) = x is a predicate that makes sense for real numbers x. The universal quantification of P(x), ∀x P(x), is a true statement, because for all real numbers, −(−x) = x. (b) Let Q(x): x + 1 < 4. ∀x Q(x) is a false statement, because Q(5) is not true. ◆ Universal quantification can also be stated in English as "for every x," "every x," or "for any x." A predicate may contain several variables. Universal quantification may be applied to each of the variables. For example, a commutative property can be expressed as ∀x ∀y x ∗ y = y ∗ x. The order in which the universal quantifiers are considered does not change the truth value. Often mathematical statements contain implied universal quantifications (for example in Theorem 1, Section 1.2). In some situations we only require that there be at least one value for which the predicate is true. The existential quantification of a predicate P(x) is the statement "There exists a value of x for which P(x) is true." The existential quantification of P(x) is denoted ∃x P(x). The symbol ∃ is called the existential quantifier. Example 9 (a) Let Q(x): x + 1 < 4. The existential quantification of Q(x), ∃x Q(x), is a true statement, because Q(2) is a true statement. (b) The statement ∃y y + 2 = y is false. There is no value of y for which the propositional function y + 2 = y produces a true statement. ◆ In English ∃x can also be read "there is an x," "there is some x," "there exists an x," or "there is at least one x." Occasionally a variation of a universal or existential quantifier is useful. The possible values of the variable may be restricted as in the following examples. The statement ∀x ∈ A P(x) represents "for all x in the set A, P(x) is true," and the statement ∃k ≥ n Q(k) represents "there is a k greater than or equal to n such that Q(k) is true." Example 10 (a) Let D be the set of n × n diagonal matrices. Consider the statement ∃M ∈ D such that M−1 does not exist. To determine if this statement is true or false, we must decide if there is an n × n diagonal matrix that does not have an inverse. (b) For the statement ∀n ≥ 2n > 2n−1 , ... Existential quantification will be defined. If n is an integer greater than or equal to 2 need to be considered. ◆ Existential quantification may be applied to several variables in a predicate and the order in which the quantifications are considered may affect the truth value. We give a mathematical example here. 2.1 Propositions and Logical Operations 55 Example 11 Let P(x, y) be ... ∃y ∀x x + y = x. ... (b) The statement ∀x ∃y x + y = x ... If P(x) is a statement "for every x there is a y such that x + y = x" ... (b) The statement ∃y ∀x x + y = x is true, because the number 0 has this property. ... (c) Let P: I am ... (d) ... The negation of the statement ∃x P(x) is the statement ∀x ∼P(x). Example 12 ...

... 58 Chapter 2 Logic ... 2.2 Conditional Statements ... If p and q are statements, the compound statement "if p then q," denoted p ⇒ q, is called a conditional statement, or implication. The statement p is called the antecedent or hypothesis, and the statement q is called the consequent or conclusion. The connective if . . . then is denoted by the symbol ⇒. ... Example 1 Form the implication p ⇒ q for each of the following. (a) p: I am hungry. q: I will eat. (b) p: It is snowing. q: 3 + 5 = 8. Solution (a) If I am hungry, then I will eat. (b) If it is snowing, then 3 + 5 = 8. ◆ ... If p ⇒ q is an implication, then the converse of p ⇒ q is the implication q ⇒ p, and the contrapositive of p ⇒ q is the implication ∼q ⇒ ∼p. ... Example 2 Give the converse and the contrapositive of the implication "If it is raining, then I get wet." Solution We have p: It is raining; and q: I get wet. The converse is q ⇒ p: If I get wet, then it is raining. The contrapositive is ∼q ⇒ ∼p: If I do not get wet, then it is not raining. ◆ If p and q are statements, the compound statement p if and only if q, denoted by p ⇔ q, is called an equivalence or biconditional. The connective if and only if is denoted by the symbol ⇔. The truth values of p ⇔ q are given in Table 2.7. Observe that p ⇔ q is true only when both p and q are true or when both p and q are false. ... A statement that is true for all possible values of its propositional variables is called a tautology. A statement that is always false is called a contradiction or an absurdity, and a statement that can be either true or false, depending on the truth values of its propositional variables, is called a contingency. ... Example 5 (a) The statement p ∨ ∼p is a tautology. (b) The statement p ∧ ∼p is an absurdity. (Verify this.) (c) The statement (p ⇒ q) ∧ (p ∨ q) is a contingency. ◆ We have now defined a new mathematical structure with two binary operations and one unary operation, (propositions, ∧, ∨, ∼). It makes no sense to say two propositions are equal; instead we say p and q are logically equivalent, or simply equivalent, if p ⇔ q is a tautology. ... THEOREM 1 The operations for propositions have the following properties. Commutative Properties 1. p ∨ q ≡ q ∨ p 2. p ∧ q ≡ q ∧ p Associative Properties 3. p ∨ (q ∨ r) ≡ (p ∨ q) ∨ r 4. p ∧ (q ∧ r) ≡ (p ∧ q) ∧ r Distributive Properties 5. p ∨ (q ∧ r) ≡ (p ∨ q) ∧ (p ∨ r) 6. p ∧ (q ∨ r) ≡ (p ∧ q) ∨ (p ∧ r) Idempotent Properties 7. p ∨ p ≡ p 8. p ∧ p ≡ p Properties of Negation 9. ∼(∼p) ≡ p 10. ∼(p ∨ q) ≡ (∼p) ∧ (∼q) 11. ∼(p ∧ q) ≡ (∼p) ∨ (∼q) ... THEOREM 2 (a) (p ⇒ q) ≡ ((∼p) ∨ q) (b) (p ⇒ q) ≡ (∼q ⇒ ∼p) (c) (p ⇔ q) ≡ ((p ⇒ q) ∧ (q ⇒ p)) (d) ∼(p ⇒ q) ≡ (p ∧ ∼q) (e) ∼(p ⇔ q) ≡ ((p ∧ ∼q) ∨ (q ∧ ∼p)) ... THEOREM 3 (a) ∼(∀x P(x)) ≡ ∃x ∼P(x) (b) ∼(∃x P(x)) ≡ ∀x (∼P(x)) (c) ∃x (P(x) ⇒ Q(x)) ≡ ∀x P(x) ⇒ ∃x Q(x) (d) ∃x (P(x) ∨ Q(x)) ≡ ∃x P(x) ∨ ∃x Q(x) (e) ∀x (P(x) ∧ Q(x)) ≡ ∀x P(x) ∧ ∀x Q(x) ... THEOREM 4 Each of the following is a tautology. (a) (p ∧ q) ⇒ p (b) (p ∧ q) ⇒ q (c) p ⇒ (p ∨ q) (d) q ⇒ (p ∨ q) (e) ∼p ⇒ (p ⇒ q) (f) ∼(p ⇒ q) ⇒ p (g) (p ∧ (p ⇒ q)) ⇒ q (h) (∼p ∧ (p ∨ q)) ⇒ q (i) (∼q ∧ (p ⇒ q)) ⇒ ∼p (j) ((p ⇒ q) ∧ (q ⇒ r)) ⇒ (p ⇒ r) ... 2.2 Exercises ... If the flood destroys my house or the fire destroys my house, then my insurance company will pay me. 9 3 (a) Which of the following is the converse of p? (b) Which of the following is the contrapositive of p? 4 5 2 9 5 5 (a) Jack did not eat fat, but he did eat broccoli. (b) Mary lost her lamb or the wolf ate the lamb. (c) If Tom stole a pie and ran away, then the three pigs do not have any supper. 26. Consider the following conditional statement: 2 1 (i) If my insurance company pays me, then the flood destroys my house or the fire destroys my house. (ii) If my insurance company pays me, then the flood destroys my house and the fire destroys my house. (iii) If my insurance company does not pay me, then the flood does not destroy my house or the fire does not destroy my house. (iv) If my insurance company does not pay me, then the flood does not destroy my house and the fire does not destroy my house. 5 7 2 6 In Exercises 21 and 22, find the truth value of each statement if p and q are true and r , s, and t are false. 21. (a) ∼ (p ⇒ q) (b) (∼ p) ⇒ r (c) (p ⇒ s) ∧ (s ⇒ t) (d) t ⇒ ∼q 22. (a) (∼q) ⇒ (r ⇒ (r ⇒ (p ∨ s))) (b) p ⇒ (r ⇒ q) (c) (q ⇒ (r ⇒ s)) ∧ ((p ⇒ s) ⇒ (∼t)) (d) (r ∧ s ∧ t) ⇒ (p ∨ q) 23. Use the definition of p ↓ q given for Exercise 31 in Section 2.1 and show that ((p ↓ p) ↓ (q ↓ q)) is equivalent to p ∧ q. 24. Write the negation of each of the following in good English. (a) The weather is bad and I will not go to work. (b) If Carol is not sick, then if she goes to the picnic, she will have a good time. (c) I will not win the game or I will not enter the contest. 25. Write the negation of each of the following in good English. 2.3 27. 28. 29. 30. 31. 32. 33. 34. 35. 36. Prove Theorem 1 part 6. Prove Theorem 1 part 11. Prove Theorem 2 part (e). Prove Theorem 3 part (d). Prove Theorem 3 part (e). Prove Theorem 4 part (a). Prove Theorem 4 part (d). Prove Theorem 4 part (g). Prove Theorem 4 part (j). Explain why proving part (e) of Theorem 4 provides a one-line proof of part (f) of Theorem 4. 37. Explain why proving part (a) of Theorem 4 provides a one-line proof of part (b) of Theorem 4. Methods of Proof Some methods of proof we have already used are direct proofs using generic elements, definitions, and previously proven facts, and proofs by cases, such as examining all possible truth value situations in a truth table. Here we look at proofs in more detail. If an implication p ⇒ q is a tautology, where p and q may be compound statements involving any number of propositional variables, we say that q logically follows from p. Suppose that an implication of the form (p1 ∧ p2 ∧ · · · ∧ pn) ⇒ q is a tautology. Then this implication is true regardless of the truth values of any of its components. In this case, we say that q logically follows from p1 , p2 , . . . , pn . 2.3 Methods of Proof 63 When q logically follows from p1 , p2 , . . . , pn , we write p1 p2 .. . ∴ pn q where the symbol ∴ means therefore. This means if we know that p1 is true, p2 is true, . . . , and pn is true, then we know q is true. Virtually all mathematical theorems are composed of implications of the type (p1 ∧ p2 ∧ · · · ∧ pn) ⇒ q. The pi 's are called the hypotheses or premises, and q is called the conclusion. To "prove the theorem" means to show that the implication is a tautology. Note that we are not trying to show that q (the conclusion) is true, but only that q will be true if all the pi are true. For this reason, mathematical proofs often begin with the statement "suppose that p1 , p2 , . . . , and pn are true" and conclude with the statement "therefore, q is true." The proof does not show that q is true, but simply shows if the pi are all true, then q has to be true. Arguments based on tautologies represent universally correct methods of reasoning. Their validity depends only on the form of the statements involved and not on the truth values of the variables they contain. Such arguments are called rules of inference. The various steps in a mathematical proof of a theorem must follow from the use of various rules of inference, and a mathematical proof of a theorem must begin with the hypotheses, proceed through various steps, each justified by some rule of inference, and arrive at the conclusion. Example 1 According to Theorem 4(j) of the last section, ((p ⇒ q) ∧ (q ⇒ r)) ⇒ (p ⇒ r) is a tautology. Thus the argument p⇒q q ⇒r ∴ p⇒r ◆ is universally valid, and so is a rule of inference. Example 2 Is the following argument valid? If you invest in the stock market, then you will get rich. If you get rich, then you will be happy. ∴ If you invest in the stock market, then you will be happy. Solution The argument is of the form given in Example 1, hence the argument is valid, ◆ although the conclusion may be false. Example 3 The tautology (p ⇔ q) ⇔ ((p ⇒ q) ∧ (q ⇒ p)) is Theorem 2(c), Section 2.2. Thus both of the following arguments are valid. p⇔q ∴ (p ⇒ q) ∧ (q ⇒ p) p⇒q q⇒p ∴ p⇔q ◆ 64 Chapter 2 Logic Some mathematical theorems are equivalences; that is, they are of the form p ⇔ q. They are usually stated p if and only if q. By Example 3, the proof of such a theorem is logically equivalent with proving both p ⇒ q and q ⇒ p, and this is almost always the way in which equivalences are proved. We wish to show that both p ⇒ q and q ⇒ p are true. To do so, we first assume that p is true, and show that q must then be true; next we assume that q is true and show that p must then be true. A very important rule of inference is p p⇒q ∴ q. That is, p is true, and p ⇒ q is true, so q is true. This follows from Theorem 4(g), Section 2.2. Some rules of inference were given Latin names by classical scholars. Theorem 4(g) is referred to as modus ponens, or loosely, the method of asserting. Example 4 Is the following argument valid? Smoking is healthy. If smoking is healthy, then cigarettes are prescribed by physicians. ∴ Cigarettes are prescribed by physicians. Solution The argument is valid since it is of the form modus ponens. However, the conclusion is false. Observe that the first premise p: smoking is healthy is false. The second premise p ⇒ q is then true and (p ∧ (p ⇒ q)), the conjunction of the two premises, is false. ◆ Example 5 Is the following argument valid? If taxes are lowered, then income rises. Income rises. ∴ Taxes are lowered. Solution Let p: taxes are lowered and q: income rises. Then the argument is of the form p⇒q q ∴ p. Assume that p ⇒ q and q are both true. Now p ⇒ q may be true with p being false. Then the conclusion p is false. Hence the argument is not valid. Another approach to answering this question is to verify whether the statement ((p ⇒ q) ∧ q) logically implies the statement p. A truth table shows this is not the case. (Verify this.) ◆ An important proof technique, which is an example of an indirect method of proof, follows from the tautology (p ⇒ q) ⇔ ((∼q) ⇒ (∼ p)). This states, as we previously mentioned, that an implication is equivalent to its contrapositive. Thus to prove p ⇒ q indirectly, we assume q is false (the statement ∼q) and show that p is then false (the statement ∼ p). Example 6 Let n be an integer. Prove that if n 2 is odd, then n is odd. 2.3 Methods of Proof 65 Solution Let p: n 2 is odd and q: n is odd. We have to prove that p ⇒ q is true. Instead, we prove the contrapositive ∼q ⇒ ∼ p. Thus suppose that n is not odd, so that n is even. Then n = 2k, where k is an integer. We have n 2 = (2k)2 = 4k 2 = 2(2k 2), so n 2 is even. We thus show that if n is even, then n 2 is even, which is the contrapositive of the given statement. Hence the given statement has been proved. ◆ Another important indirect proof technique is proof by contradiction. This method is based on the tautology ((p ⇒ q) ∧ (∼q)) ⇒ (∼ p). Thus the rule of inference p⇒q ∼q ∴ ∼p is valid. Informally, this states that if a statement p implies a false statement q, then p must be false. This is often applied to the case where q is an absurdity or contradiction, that is, a statement that is always false. An example is given by taking q as the contradiction r ∧ (∼r). Thus any statement that implies a contradiction must be false. In order to use proof by contradiction, suppose we wish to show that a statement q logically follows from statements p1 , p2 , . . . , pn . Assume that ∼q is true (that is, q is false) as an extra hypothesis, and that p1 , p2 , . . . , pn are also true. If this enlarged hypothesis p1 ∧ p2 ∧ · · · ∧ pn ∧ (∼q) implies a contradiction, then at least one of the statements p1 , p2 , . . . , pn , ∼q must be false. This means that if all the pi 's are true, then ∼q must be false, so q must be true. Thus q follows from p1 , p2 , . . . , pn . This is proof by contradiction. Example 7 Prove there is no rational number p/q whose square is 2. In other words, show √ 2 is irrational. Solution This statement is a good candidate for proof by contradiction, because we could not check all possible rational numbers to demonstrate that none had a square equal to 2. Assume (p/q)2 = 2 for some integers p and q, which have no common factors. If the original choice of p/q is not in lowest terms, we can replace it with its equivalent lowest-term form. Then p 2 = 2q 2 , so p 2 is even. This implies p is even, since the square of an odd number is odd. Thus, p = 2n for some integer n. We see that 2q 2 = p 2 = (2n)2 = 4n 2 , so q 2 = 2n 2 . Thus q 2 is even, and so q is even. We now have that both p and q are even, and therefore have a common factor 2. This is a contradiction to the assumption. Thus the assumption must be false. ◆ We have presented several rules of inference and logical equivalences that correspond to valid proof techniques. In order to prove a theorem of the (typical) form (p1 ∧ p2 ∧ · · · ∧ pn) ⇒ q, we begin with the hypothesis p1 , p2 , . . . , pn and show that some result r1 logically follows. Then, using p1 , p2 , . . . , pn , r1 , we show that some other statement r2 logically follows. We continue this process, producing intermediate statements r1 , r2 , . . . , rk , called steps in the proof, until we can finally show that the conclusion q logically follows from p1 , p2 , . . . , pn , r1 , r2 , . . . , rk . Each logical step must be justified by some valid proof technique, based on the rules of inference we have developed, or on some other rules that come from tautological implications we have not discussed. At any stage, we can replace a statement that needs to be derived by its contrapositive statement, or any other equivalent form. 66 Chapter 2 Logic In practice, the construction of proofs is an art and must be learned in part from observation and experience. The choice of intermediate steps and methods of deriving them is a creative activity that cannot be precisely described. But a few simple techniques are applicable to a wide variety of settings. We will focus on these techniques throughout the book. The "Tips for Proofs" notes at the end of each chapter highlight the methods most useful for that chapter's material. Example 8 Let m and n be integers. Prove that n 2 = m 2 if and only if n is m or n is −m. Solution Let us analyze the proof as we present it. Suppose p is the statement n 2 = m 2 , q is the statement n is m, and r is the statement n is −m. Then we wish to prove the theorem p ⇔ (q ∨ r). We know from previous discussion that we may instead prove s: p ⇒ (q ∨ r) and t: (q ∨ r) ⇒ p are true. First, we assume that either q: n is m or r : n is −m is true. If q is true, then n 2 = m 2 , and if r is true, then n 2 = (−m)2 = m 2 , so in either case p is true. We have therefore shown that the implication t: (q ∨ r) ⇒ p is true. Now we must prove that s: p ⇒ (q ∨ r) is true; that is, we assume p and try to prove either q or r . If p is true, then n 2 = m 2 , so n 2 − m 2 = 0. But n 2 −m 2 = (n −m)(n +m). If r1 is the intermediate statement (n −m)(n +m) = 0, we have shown p ⇒ r1 is true. We now show that r1 ⇒ (q ∨r) is true, by showing that the contrapositive ∼(q ∨ r) ⇒ (∼r1) is true. Now ∼(q ∨ r) is equivalent to (∼q) ∧ (∼r), so we show that (∼q) ∧ (∼r) ⇒ (∼r1). Thus, if (∼q): n is not m and (∼r): n is not −m are true, then (n − m) ≠ 0 and (n + m) ≠ 0, so (n − m)(n + m) ≠ 0 and r1 is false. We have therefore shown that r1 ⇒ (q ∨ r) is true. Finally, from the truth of p ⇒ r1 and r1 ⇒ (q ∨ r), we can conclude that p ⇒ (q ∨ r) is true, and we are done. ◆ We do not usually analyze proofs in this detailed manner. We have done so only to illustrate that proofs are devised by piecing together equivalences and valid steps resulting from rules of inference. The amount of detail given in a proof depends on who the reader is likely to be. As a final remark, we remind the reader that many mathematical theorems actually mean that the statement is true for all objects of a certain type. Sometimes this is not evident. Thus the theorem in Example 8 really states that for all integers m and n, n 2 = m 2 if and only if n is m or n is −m. Similarly, the statement "If x and y are real numbers, and x ≠ y, then x < y or y < x" is a statement about all real numbers x and y. To prove such a theorem, we must make sure that the steps in the proof are valid for every √ real number. We could not assume, for example, that x is 2, or that y is π or 3. This is why proofs often begin by selecting a generic element, denoted by a variable. On the other hand, we know from Section 2.2 that the negation of a statement of the form ∀x P(x) is ∃x ∼P(x), so we need only find a single example where the statement is false to disprove it. Example 9 Prove or disprove the statement that if x and y are real numbers, (x 2 = y 2) ⇔ (x = y). Solution The statement can be restated in the form ∀x ∀y R(x, y). Thus, to prove this result, we would need to provide steps, each of which would be true for all x and y. To disprove the result, we need only find one example for which the implication is false. Since (−3)2 = 32 , but −3 ≠ 3, the result is false. Our example is called a ◆ counterexample, and any other counterexample would do just as well. 2.3 Methods of Proof 67 In summary, if a statement claims that a property holds for all objects of a certain type, then to prove it, we must use steps that are valid for all objects of that type and that do not make references to any particular object. To disprove such a statement, we need only show one counterexample, that is, one particular object or set of objects for which the claim fails. 2.3 Exercises In Exercises 1 through 11, state whether the argument given is valid or not. If it is valid, identify the tautology or tautologies on which it is based. 1. 2. 3. 4. 5. 6. If I drive to work, then I will arrive tired. I am not tired when I arrive at work. ∴ I do not drive to work. If I drive to work, then I will arrive tired. I arrive at work tired. ∴ I drive to work. If I drive to work, then I will arrive tired. I do not drive to work. ∴ I will not arrive tired. If I drive to work, then I will arrive tired. I drive to work. ∴ I will arrive tired. I will become famous or I will not become a writer. I will become a writer. ∴ I will become famous. I will become famous or I will be a writer. I will not be a writer. ∴ I will become famous. 7. If I try hard and I have talent, then I will become a musician. If I become a musician, then I will be happy. ∴ If I will not be happy, then I did not try hard or I do not have talent. 8. If I graduate this semester, then I will have passed the physics course. If I do not study physics for 10 hours a week, then I will not pass physics. If I study physics for 10 hours a week, then I cannot play volleyball. ∴ If I play volleyball, I will not graduate this semester. 9. If my plumbing plans do not meet the construction code, then I cannot build my house. If I hire a licensed contractor, then my plumbing plans will meet the construction code. I hire a licensed contractor. ∴ I can build my house. 10. (a) p∨q ∼q ∴ p (b) p⇒q ∼p ∴ ∼q 11. Write each argument in Exercise 10 as a single compound statement. 12. (a) (p ⇒ q) ∧ (q ⇒ r) (∼q) ∧ r ∴ p (b) ∼(p ⇒ q) p ∴ ∼q 13. Write each argument in Exercise 12 as a single compound statement. 14. Prove that the sum of two even numbers is even. 15. Prove that the sum of two odd numbers is even. 16. Prove that the structure (even integers, +, ∗) is closed with respect to ∗ . 17. Prove that the structure (odd integers, +, ∗) is closed with respect to ∗ . 18. Prove that n 2 is even if and only if n is even. 19. Prove that A = B if and only if A ⊆ B and B ⊆ A. 20. Let A and B be subsets of a universal set U . Prove that A ⊆ B if and only if B ⊆ A. 21. Show that (a) A ⊆ B is a necessary and sufficient condition for A ∪ B = B. (b) A ⊆ B is a necessary and sufficient condition for A ∩ B = A. 22. Show that k is odd is a necessary and sufficient condition for k 3 to be odd. 23. Prove or disprove: n 2 + 41n + 41 is a prime number for every integer n. 24. Prove or disprove: The sum of any five consecutive integers is divisible by 5. 25. Prove or disprove that 3 | (n 3 − n) for every positive integer n. 26. Prove or disprove: 1 + 2n > 3n , for all n ∈ Z + . 27. Determine if the following is a valid argument. Explain your conclusion. Prove: ∀x x 3 > x 2 . Proof: ∀x x 2 > 0 so ∀x x 2 (x − 1) > 0(x − 1) and ∀x x 3 − x 2 > 0. Hence ∀x x 3 > x 2 . 28. Determine if the following is a valid argument. Explain your conclusion. 68 Chapter 2 Logic Prove: If A and B are matrices such that AB = 0, then either A = 0 or B = 0. Proof: There are two cases to consider: A = 0 or A ≠ 0. If A = 0, then we are done. If A ≠ 0, then A−1 (AB) = A−1 0 and (A−1 A)B = 0 and B = 0. 29. Determine if the following is a valid argument. Explain your conclusion. Let m and n be two relatively prime integers. Prove that if mn is a cube, then m and n are each cubes. Proof: We first note that in the factorization of any cube into prime factors, each prime must have an exponent that is a multiple of 3. Write m and n each as a product of b a a a b b primes; m = p1 1 p2 2 · · · pk k and n = q1 1 q2 2 · · · qj j . Suppose m is not a cube. Then at least one ai is not a multiple of 3. Since each prime factor of mn must have an expob nent that is a multiple of 3, n must have a factor pi i such that bi ≠ 0 and ai + bi is a multiple of 3. But this means that m and n share a factor, pi . This contradicts the fact that m and n are relatively prime. 2.4 30. Determine if the following is a valid argument. Explain your conclusion. Prove: If x is an irrational number, then 1 − x is also an irrational number. Proof: Suppose 1 − x is rational. Then we can write a a 1 − x as , with a, b ∈ Z . Now we have 1 − = x and b b b−a x = , a rational number. This is a contradiction. b Hence, if x is irrational, so is 1 − x. 31. Prove that the sum of two prime numbers, each larger than 2, is not a prime number. 32. Prove that if two lines are each perpendicular to a third line in the plane, then the two lines are parallel. 33. Prove that if x is a rational number and y is an irrational number, then x + y is an irrational number. 34. Prove that if 2y is an irrational number, then y is an irrational number. Mathematical Induction Here we discuss another proof technique. Suppose the statement to be proved can be put in the form ∀n ≥ n 0 P(n), where n 0 is some fixed integer. That is, suppose we wish to show that P(n) is true for all integers n ≥ n 0 . The following result shows how this can be done. Suppose that (a) P(n 0) is true and (b) If P(k) is true for some k ≥ n 0 , then P(k + 1) must also be true. Then P(n) is true for all n ≥ n 0 . This result is called the principle of mathematical induction. Thus to prove the truth of a statement ∀n ≥ n 0 P(n), using the principle of mathematical induction, we must begin by proving directly that the first proposition P(n 0) is true. This is called the basis step of the induction and is generally very easy. Then we must prove that P(k) ⇒ P(k + 1) is a tautology for any choice of k ≥ n 0 . Since the only case where an implication is false is if the antecedent is true and the consequent is false, this step is usually done by showing that if P(k) were true, then P(k + 1) would also have to be true. Note that this is not the same as assuming that P(k) is true for some value of k. This step is called the induction step, and some work will usually be required to show that the implication is always true. Example 1 Show, by mathematical induction, that for all n ≥ 1, 1 + 2 + 3 + ··· + n = n(n + 1) . 2 Solution Let P(n) be the predicate 1 + 2 + 3 + · · · + n = n(n + 1) . In this example, n 0 = 1. 2 Basis Step We must first show that P(1) is true. P(1) is the statement 1= which is clearly true. 1(1 + 1) , 2 2.4 Mathematical Induction 69 Induction Step We must now show that for k ≥ 1, if P(k) is true, then P(k + 1) must also be true. We assume that for some fixed k ≥ 1, 1 + 2 + 3 + ··· + k = k(k + 1) . 2 (1) We now wish to show the truth of P(k + 1): 1 + 2 + 3 + · · · + (k + 1) = (k + 1)((k + 1) + 1) . 2 The left-hand side of P(k + 1) can be written as 1 + 2 + 3 + · · · + k + (k + 1) and we have (1 + 2 + 3 + · · · + k) + (k + 1) k(k + 1) + (k + 1) 2 k +1 = (k + 1) 2 using (1) to replace 1 + 2 + · · · + k = factoring = (k + 1)(k + 2) 2 = (k + 1)((k + 1) + 1) 2 the right-hand side of P(k + 1) Thus, we have shown the left-hand side of P(k + 1) equals the right-hand side of P(k + 1). By the principle of mathematical induction, it follows that P(n) is true for all n ≥ 1. ◆ Example 2 Let A1 , A2 , A3 , . . . , An be any n sets. We show by mathematical induction that n i=1 Ai = n Ai . i=1 (This is an extended version of one of De Morgan's laws.) Let P(n) be the predicate that the equality holds for any n sets. We prove by mathematical induction that for all n ≥ 1, P(n) is true. Basis Step P(1) is the statement A1 = A1 , which is obviously true. 70 Chapter 2 Logic Induction Step We use P(k) to show P(k + 1). The left-hand side of P(k + 1) is k+1 Ai = A1 ∪ A2 ∪ · · · ∪ Ak ∪ Ak+1 i=1 = (A1 ∪ A2 ∪ · · · ∪ Ak) ∪ Ak+1 associative property of ∪ = (A1 ∪ A2 ∪ · · · ∪ Ak) ∩ Ak+1 k Ai ∩ Ak+1 = by De Morgan's law for two sets using P(k) i=1 = k+1 Ai right-hand side of P(k + 1) i=1 Thus, the implication P(k) ⇒ P(k + 1) is a tautology, and by the principle of mathematical induction P(n) is true for all n ≥ 1. ◆ Example 3 We show by mathematical induction that any finite, nonempty set is countable; that is, it can be arranged in a list. Let P(n) be the predicate that if A is any set with |A| = n ≥ 1, then A is countable. (See Chapter 1 for definitions.) Basis Step Here n 0 is 1, so we let A be any set with one element, say A = {x}. In this case x forms a sequence all by itself whose set is A, so P(1) is true. Induction Step We want to use the statement P(k) that if A is any set with k elements, then A is countable. Now choose any set B with k + 1 elements and pick any element x in B. Since B − {x} is a set with k elements, the induction hypothesis P(k) tells us there is a sequence x1 , x2 , . . . , xk with B − {x} as its corresponding set. The sequence x1 , x2 , . . . , xk , x then has B as the corresponding set so B is countable. Since B can be any set with k + 1 elements, P(k + 1) is true if P(k) is. Thus, by ◆ the principle of mathematical induction, P(n) is true for all n ≥ 1. In proving results by induction, you should not start by assuming that P(k + 1) is true and attempting to manipulate this result until you arrive at a true statement. This common mistake is always an incorrect use of the principle of mathematical induction. A natural connection exists between recursion and induction, because objects that are recursively defined often use a natural sequence in their definition. Induction is frequently the best, maybe the only, way to prove results about recursively defined objects. Example 4 Consider the following recursive definition of the factorial function: 1! = 1, n! = n(n − 1)!, n > 1. Suppose we wish to prove for all n ≥ 1, n! ≥ 2n−1 . We proceed by mathematical induction. Let P(n): n! ≥ 2n−1 . Here n 0 is 1. Basis Step P(1) is the statement 1! ≥ 20 . Since 1! is 1, this statement is true. 2.4 Mathematical Induction 71 Induction Step We want to show P(k) ⇒ P(k + 1) is a tautology. It will be a tautology if P(k) true guarantees P(k + 1) is true. Suppose k! ≥ 2k−1 for some k ≥ 1. Then by the recursive definition, the left side of P(k + 1) is (k + 1)! = (k + 1)k! ≥ (k + 1)2k−1 using P(k) ≥2×2 k + 1 ≥ 2, since k ≥ 1 = 2k right-hand side of P(k + 1) k−1 Thus, P(k + 1) is true. By the principle of mathematical induction, it follows that P(n) is true for all n ≥ 1. ◆ The following example shows one way in which induction can be useful in computer programming. The pseudocode used in this and following examples is described in Appendix A. Example 5 Consider the following function given in pseudocode. FUNCTION SQ(A) 1. C ← 0 2. D ← 0 3. WHILE (D ≠ A) a. C ← C + A b. D ← D + 1 4. RETURN (C) END OF FUNCTION SQ The name of the function, SQ, suggests that it computes the square of A. Step 3b shows A must be a positive integer if the looping is to end. A few trials with particular values of A will provide evidence that the function does carry out this task. However, suppose we now want to prove that SQ always computes the square of the positive integer A, no matter how large A might be. We shall give a proof by mathematical induction. For each integer n ≥ 0, let Cn and Dn be the values of the variables C and D, respectively, after passing through the WHILE loop n times. In particular, C0 and D0 represent the values of the variables before looping starts. Let P(n) be the predicate Cn = A × Dn . We shall prove by induction that ∀n ≥ 0 P(n) is true. Here n 0 is 0. Basis Step P(0) is the statement C0 = A × D0 , which is true since the value of both C and D is zero "after" zero passes through the WHILE loop. Induction Step We must now use P(k) : Ck = A × Dk (2) to show that P(k + 1): Ck+1 = A × Dk+1 . After a pass through the loop, C is 72 Chapter 2 Logic increased by A, and D is increased by 1, so Ck+1 = Ck + A and Dk+1 = Dk + 1. left-hand side of P(k + 1): Ck+1 = Ck + A using (2) to replace Ck = A × Dk + A = A × (Dk + 1) factoring right-hand side of = A × Dk+1 P(k + 1) By the principle of mathematical induction, it follows that as long as looping occurs, Cn = A × Dn . The loop must terminate. (Why?) When the loop terminates, D = A, so C = A × A, or A2 , and this is the value returned by the function SQ. ◆ Example 5 illustrates the use of a loop invariant, a relationship between variables that persists through all iterations of the loop. This technique for proving that loops and programs do what is claimed they do is an important part of the theory of algorithm verification. In Example 5 it is clear that the looping stops if A is a positive integer, but for more complex cases, this may also be proved by induction. Example 6 Use the technique of Example 5 to prove that the pseudocode program given in Section 1.4 does compute the greatest common divisor of two positive integers. Solution Here is the pseudocode given earlier. FUNCTION GCD(X,Y) 1. WHILE (X ≠ Y) a. IF (X > Y) THEN 1. X ← X − Y b. ELSE 1. Y ← Y − X 2. RETURN (X) END OF FUNCTION GCD We claim that if X and Y are positive integers, then GCD returns GCD(X, Y). To prove this, let X n and Yn be the values of X and Y after n ≥ 0 passes through the WHILE loop. We claim that P(n): GCD(X n , Yn) = GCD(X, Y) is true for all n ≥ 0, and we prove this by mathematical induction. Here n 0 is 0. Basis Step X 0 = X , Y0 = Y , since these are the values of the variables before looping begins; thus P(0) is the statement GCD(X 0 , Y0) = GCD(X, Y), which is true. Induction Step Consider the left-hand side of P(k + 1), that is, GCD(X k+1 , Yk+1). After the k + 1 pass through the loop, either X k+1 = X k and Yk+1 = Yk − X k or X k+1 = X k − Yk and Yk+1 = Yk . Then if P(k): GCD(X k , Yk) = GCD(X, Y) is true, we have, by Theorem 5, Section 1.4, that GCD(X k+1 , Yk+1) = GCD(X k , Yk) = GCD(X, Y). Thus, by the principle of mathematical induction, P(n) is true for all n ≥ 0. The exit condition for the loop is X n = Yn and we have GCD(X n , Yn) = X n . Hence ◆ the function always returns the value GCD(X, Y). 2.4 Mathematical Induction 73 A discussion of how to construct a loop invariant for a given loop would take us too far from the main ideas of this section. In writing programs, loop invariant construction usually precedes decisions about the steps to be executed in the loop. In the exercises, loop invariants will be provided where needed. Strong Induction A slightly different form of mathematical induction is easier to use in some proofs. In the strong form of mathematical induction, or strong induction, the induction step is to show that P(n 0) ∧ P(n 0 + 1) ∧ P(n 0 + 2) ∧ · · · ∧ P(k) ⇒ P(k + 1) is a tautology. As before, the only case we need to check is that if each P(j), j = n 0 , . . . , k, is true, then P(k + 1) is true. The strong form of induction is equivalent to the form we first presented, so it is a matter of convenience which one we use in a proof. a a Prove that every positive integer n > 1 can be written uniquely as p11 p22 · · · psas , where the pi are primes and p1 < p2 < · · · < ps (Theorem 3, Section 1.4). Example 7 Proof (by strong induction) Basis Step Here n 0 is 2. P(2) is clearly true, since 2 is prime. Induction Step We use P(2), P(3), . . . , P(k) to show P(k + 1): k + 1 can be written uniquely as a a p11 p22 · · · psas , where the pi are primes and p1 < p2 < · · · < ps . There are two cases to consider. If k + 1 is a prime, then P(k + 1) is true. If k + 1 is not prime, then k + 1 = lm, 2 ≤ l ≤ k, 2 ≤ m ≤ k. Using P(l) and P(m), we have b b c c a a k = lm = q1 1 q2 2 · · · qtbt r1 1 r2 2 · · · rucu = p11 p22 · · · psas , where each pi = q j or rk , p1 < p2 < · · · < ps , and if q j = rk = pi , then ai = b j + ck , otherwise pi = q j and ai = b j or pi = rk and ai = ck . Since the factorization of l and m are unique, ◆ so is the factorization of k + 1. 2.4 Exercises In Exercises 1 through 7, prove the statement is true by using mathematical induction. 1. 2 + 4 + 6 + · · · + 2n = n(n + 1) n(2n + 1)(2n − 1) 2. 1 + 3 + 5 + · · · + (2n − 1) = 3 2 2 2 2 3. 1 + 21 + 22 + · · · + 2n = 2n+1 − 1 5n(n + 1) 4. 5 + 10 + 15 + · · · + 5n = 2 n(n + 1)(2n + 1) 5. 12 + 22 + 32 + · · · + n 2 = 6 6. 1 + a + a 2 + · · · + a n−1 = an − 1 a−1 7. a + ar + ar 2 + · · · + ar n−1 = a(1 − r n) for r ≠ 1 1−r 8. Let P(n): 13 + 23 + 33 + · · · + n 3 = (a) Use P(k) to show P(k + 1). n 2 (n + 1)2 + 4 . 4 (b) Is P(n) true for all n ≥ 1? 9. Let P(n): 1 + 5 + 9 + · · · + (4n − 3) = (2n + 1)(n − 1). (a) Use P(k) to show P(k + 1). (b) Is P(n) true for all n ≥ 1? 10. Prove 1 + 2n < 3n for n ≥ 2. 11. Prove n < 2n for n > 1. (2n + 1)2 8 13. Find the least n for which the statement is true and then prove that (1 + n 2) < 2n . 12. Prove 1 + 2 + 3 + · · · + n < 1, then p | a. 30. Prove that if GCD(a, b) = 1, then GCD(a n , bn) = 1 for all n ≥ 1. (Hint: Use Exercise 29.) 31. (a) Find the smallest positive integer n 0 such that 2n0 > n 20 . (b) Prove 2n > n 2 for all n ≥ n 0 . 32. Prove or disprove: 2 + 8 + 18 + · · · + 2n 2 = n 2 + n. 33. Prove or disprove: x − y divides x n − y n for n ≥ 1. In Exercises 34 through 39, show that the given algorithm, correctly used, produces the output stated, by using mathematical induction to prove the relationship indicated is a loop invariant and checking values when the looping stops. All variables represent nonnegative integers. 34. SUBROUTINE COMP(X,Y;Z) 1. Z ← X 2. W ← Y 3. WHILE (W > 0) a. Z ← Z + Y b. W ← W − 1 4. RETURN END OF SUBROUTINE COMP COMPUTES: Z = X + Y 2 LOOP INVARIANT: (Y × W) + Z = X + Y 2 35. SUBROUTINE DIFF (X,Y;Z) 1. Z ← X 2. W ← Y 3. WHILE (W > 0) a. Z ← Z − 1 b. W ← W − 1 4. RETURN END OF SUBROUTINE DIFF COMPUTES: Z = X − Y LOOP INVARIANT: X − Z + W = Y 36. SUBROUTINE EXP2 (N,M;R) 1. R ← 1 2. K ← 2M 3. WHILE (K > 0) a. R ← R × N b. K ← K − 1 4. RETURN END OF SUBROUTINE EXP2 COMPUTES: R = N 2M LOOP INVARIANT: R × N K = N 2M 2.5 Mathematical Statements 37. SUBROUTINE POWER (X,Y;Z) 1. Z ← 0 2. W ← Y 3. WHILE (W > 0) a. Z ← Z + X b. W ← W − 1 4. W ← Y − 1 5. U ← Z 6. WHILE (W > 0) a. Z ← Z + U b. W ← W − 1 7. RETURN END OF SUBROUTINE POWER COMPUTES: Z = X × Y 2 LOOP INVARIANT (first loop): X + (X × W) = X × Y 2 LOOP INVARIANT (second loop): X + (X × Y × W) = X × Y 2 (Hint: Use the value of Z at the end of the first loop in loop 2.) 38. SUBROUTINE DIV(X,Y) 1. IF (Y = 0) THEN a. PRINT ('error Y = 0') 2. ELSE a. R ← X b. K ← 0 c. WHILE (K ≥ Y) 2.5 75 1. R ← R − Y 2. K ← K + 1 d. IF (R = 0) THEN 1. PRINT ('true') e. ELSE 1. PRINT ('false') 3. RETURN END OF SUBROUTINE DIV COMPUTES: TRUTH VALUE OF Y | X. LOOP INVARIANT: R + K × Y = X 39. SUBROUTINE SQS(X,Y;Z) 1. Z ← Y 2. W ← X 3. WHILE (W > 0) a. Z ← Z + X b. W ← W − 1 4. W ← Y − 1 5. WHILE (W > 0) a. Z ← Z + X b. W ← W − 1 6. RETURN END OF SUBROUTINE SQS COMPUTES: Z = X 2 × Y 2 LOOP INVARIANT (first loop): Z + (X × W) = Y + X 2 LOOP INVARIANT (second loop): Z + (Y × W) = X 2 + Y 2 Mathematical Statements The previous sections in this chapter lay the groundwork for learning to write proofs. The examples and exercises give many mathematical statements for which you are asked to write proofs. How are statements like this produced? Some of the most common ways to create a statement that may be mathematically true are from experience, by applying old ideas in new settings, by extending known facts to more general cases, and by experimentation. A mathematical statement that has not been proved or disproved yet is called a conjecture. Example 1 (a) Young students soon notice that adding two odd numbers yields an even number. A conjecture that captures this experience is "The sum of two odd numbers is an even number." (b) Suppose S is a set of objects and a binary operation has been defined for these objects. Then the truth of s t = t s, s, t ∈ S can be proved or disproved to determine whether is commutative. (c) If A, B, and C are n × n matrices, the fact that (A + B)T = AT + BT suggests the conjecture that (A + B + C)T = AT + BT + CT . ◆ Example 2 Produce a reasonable conjecture about the sum of the powers of 2 beginning with 20 . 76 Chapter 2 Logic Solution We begin by experimenting with small cases and recording the results. 20 2 0 + 21 2 +2 +2 0 1 2 2 +2 +2 +2 0 1 2 3 2 +2 +2 +2 +2 0 1 2 3 4 1 1 1+2 3 1+2+4 7 1+2+4+8 15 1 + 2 + 4 + 8 + 16 31 Note that each sum is one less than the next term to be added. (Recognizing a pattern may take more experimentation than shown here.) Thus, a reasonable conjecture is "The sum of the powers of 2 beginning with 20 is one less than the next power of 2 to be added." This would be clearer in symbolic form. The sum of an arbitrary number of powers of 2 beginning with 20 can be represented by 20 + 21 + · · · + 2n . The result is then 2n+1 − 1, one less than the next power of 2 to be added. 20 + 21 + · · · + 2n = 2n+1 − 1 is given to be proved in Exercise 3, Section 2.4. ◆ Even when a mathematical statement is written in ordinary language, the proof often needs a way to represent a general case. For example, a proof of the conjecture "The sum of two odd numbers is an even number" requires a way to represent a generic odd number. One way to do this is with 2n + 1. (What is another one?) We need to identify what n can represent. If n is a whole number, then the original conjecture becomes "The sum of two odd whole numbers is an even whole number." If n is an integer, the statement becomes "The sum of two odd integers is an even integer." Let us examine a proof of this last version. The conjecture refers to two odd integers, so we must represent another odd integer. This requires another variable, say k. Now the two odd integers are represented as 2n + 1 and 2k + 1, where both n and k are integers. Note that this description includes the case that an odd integer is added to itself. Any even integer can be written as 2t, t ∈ Z. Example 3 The sum of two odd integers is an even integer. Proof Let 2n + 1 and 2k + 1 be two odd integers. Their sum is (2n + 1) + (2k + 1) or 2n + 2k + 2. 2n + 2k + 2 = 2(n + k + 1). Since n + k + 1 is an integer, (2n + 1) + (2k + 1) is an even integer. ◆ This proof assumes that the reader knows the distributive property and that the sum of two integers is an integer. Depending on the audience for the proof, these facts might have to be proved first or spelled out. Remember that here a proof is a written communication, and the intended audience must be considered when writing a proof. Example 4 Let A, B, C, and D be n × n Boolean matrices. The fact that A ∧ (B ∨ C) = (A ∧ B) ∨ (A ∧ C) suggests that the extended case A ∧ (B ∨ C ∨ D) = (A ∧ B) ∨ (A ∧ C) ∨ (A ∧ D) may be true. 2.5 Mathematical Statements 77 Often the key to proving an extension is to apply the original statement. Here A ∧ (B ∨ C ∨ D) = A ∧ ((B ∨ C) ∨ D), because ∨ is associative. The original statement for three matrices implies that A∧((B∨C)∨D) = (A∧(B∨C))∨(A∧D) when we view (B ∧ C) as a single matrix. Now using the original statement again, we have A ∧ ((B ∨ C) ∨ D) = (A ∧ (B ∨ C)) ∨ (A ∧ D) = ((A ∧ B) ∨ (A ∧ C)) ∨ (A ∧ D) = (A ∧ B) ∨ (A ∧ C) ∨ (A ∧ D). The last equality follows again from the fact that ∨ is associative. ◆ Notice that in Example 4 we used more than the original statement to complete the proof. Thus, we cannot expect that extension statements will always be true. A variation of applying an old idea in a new setting is to identify a special situation where a statement that is false in general will now be true. For example, in general, matrix multiplication is not commutative, but for some special cases it may be true that AB = BA. Example 5 Let S be the set of matrices of the form ⎡ ⎤ a 0 0 ⎣ 0 0 0 ⎦, 0 0 b where a, b ∈ R. Prove that for the mathematical structure (S, matrix multiplication) the commutative property holds. Solution We need to represent two generic elements of S, and because we need to examine the results of matrix multiplication, the entries of these generic elements need to be shown. Let ⎡ ⎤ c1 0 0 C=⎣ 0 0 0 ⎦ 0 0 c2 ⎡ and ⎤ d1 0 0 D = ⎣ 0 0 0 ⎦. 0 0 d2 (How do we know C and D are in S?) Then ⎤ 0 c1 d1 0 0 0 ⎦ CD = ⎣ 0 0 0 c2 d2 ⎡ ⎡ and ⎤ 0 d1 c1 0 0 0 ⎦. DC = ⎣ 0 0 0 d2 c2 Hence, CD = DC, because multiplication in R is commutative; c1 d1 = d1 c1 and c2 d2 = d2 c2 . As noted after Example 3, the intended audience for the proof ◆ determines how much detail about R is included. There are many other ways to create conjectures than have been discussed thus far. More of these are covered when we discuss more proof techniques in the following chapters. 78 Chapter 2 Logic 2.5 Exercises 1. Prove or disprove that the cube of an even number is even. 2. Prove or disprove that the cube of an odd number is odd. 3. (a) Prove or disprove that the sum of three consecutive integers is divisible by 3. (b) Prove or disprove that the sum of four consecutive integers is divisible by 4. 4. Prove that the product of a nonzero rational number and an irrational number is a(n) number. 5. Prove that the quotient of a nonzero rational number and an irrational number is a(n) number. 6. State and prove the extension of A ∪ B = A ∩ B for three sets. 7. State and prove the extension of A ∩ B = A ∪ B for three sets. 8. Let A, B, C, and D be n × n Boolean matrices. State and prove the extension of A ∨ (B ∧ C) = (A ∨ B) ∧ (A ∨ C) to four matrices. 9. Modify the conjecture and proof in Example 4 to give the general case for m + 1 Boolean matrices. 10. Let p and q be propositions. State and prove an extension of ∼(p ∨ q) ≡ ∼ p ∧ ∼q to the case of (b) n propositions. (a) 3 propositions 11. Let p and q be propositions. State and prove an extension of ∼(p ∧ q) ≡ ∼ p ∨ ∼q to the case of (b) n propositions. (a) 3 propositions 12. Prove that the commutative property holds for the mathematical structure (3 × 3 diagonal matrices, matrix multiplication). 13. Prove that the commutative property holds for the mathematical structure (r × r diagonal matrices, matrix multiplication). 14. State and prove a conjecture about the sum of the first n positive odd integers. 2.6 For Exercises 15 through 17, use the sequence 3, 9, 15, 21, 27, 33, ... 15. Give both a recursive and an explicit formula for this sequence. 16. Experiment with the sums of the terms of the sequence to produce a reasonable conjecture. 17. Prove the conjecture from Exercise 16. 18. State and prove a reasonable conjecture about the sum of the first n terms of the sequence 6, 10, 14, 18, 22, ... For Exercises 19 through 22, use the recursively defined sequence g1 = 1, g2 = 3, gn = gn−1 + gn−2 . 19. Write the first ten terms of the sequence. 20. Experiment with the sums of terms in the sequence that occupy even-numbered positions to produce a reasonable conjecture. 21. Experiment with the sums of terms of the sequence to produce a reasonable conjecture about the sum of the first n terms of the sequence. 22. Experiment with the sums of terms in the sequence that occupy odd-numbered positions to produce a reasonable conjecture. 23. Prove the conjecture from Exercise 20. 24. Prove the conjecture from Exercise 21. 25. Prove the conjecture from Exercise 22. 26. Produce and prove a conjecture about the sum of the powers of 3 beginning with 30 . 27. Build on the work done in Exercise 3 to produce a conjecture about when the sum of k consecutive integers is divisible by k. 28. Prove the conjecture of Exercise 27. Logic and Problem Solving In previous sections, we investigated the use of logic to prove mathematical theorems and to verify the correctness of computational algorithms. However, logic is also valuable in less formal settings. Logic is used every day to decide between alternatives and investigate consequences. It is, in many situations, essentially the same as precise and careful thinking, particularly in cases where the details of the situation are complex. One of the most important uses of logic is to develop correct and efficient algorithms to solve problems. Part of this process may be to express the solution in terms of a computation, even if the problem does not at first seem computational. In Section 2.5, we noted that one common way to create mathematical conjectures is to apply old ideas in new settings. This approach is also a powerful problem-solving technique. To demonstrate these two ideas, we present a variety of seemingly unrelated problems that can be solved by a single method. 2.6 Logic and Problem Solving 79 We begin with an easily stated mathematical question. Given a set A and a finite number of subsets of A, say A1 , A2 , . . . , Ak , is it possible to find a collection S of the Ai 's with the following two properties: 1. Any two sets in S are disjoint. 2. The union of all sets in S is A? The collection S is called an exact cover of A with respect to the Ai 's, because together the Ai 's "cover" A with no overlaps. Example 1 Let A = {a, b, c, d, e, f, g, h, i, j} and A1 = {a, c, d}, A2 = {a, b, e, f }, A3 = {b, f, g}, A4 = {d, h, i}, A5 = {a, h, j}, A6 = {e, h}, A7 = {c, i, j}, A8 = {i, j}. Is there an exact cover of A with respect to A1 , A2 , A3 , A4 , A5 , A6 , A7 , A8 ? Solution We start with A1 and consider each of the other sets in turn. We reject A2 , since A1 ∩ A2 = ∅, but A3 is suitable. A1 and A3 together contain the elements a, b, c, d, f, and g of A. The sets A4 and A5 contain d and a respectively, and so they are rejected. At this point we can see that A6 and A8 will complete the search, and ◆ A1 , A3 , A6 , A8 form an exact cover of A. If we did not find an exact cover that contained A1 , we would have started the search over with A2 as the first choice. This method would be continued until either we found an exact cover or failed for each choice of a starting set. A systematic way to try all possibilities, such as we just described, is a key step in writing algorithms and computer programs to solve many types of problems involving only a finite number of possible solutions. The technique is called backtracking, and is an essential part of many algorithms that search all possible outcomes of a process. This method is a way of implementing a brute force search of possibilities and will always produce a solution if one exists. Example 2 Let U = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} and let A1 = {2, 4, 6, 7}, A2 = {1, 2, 5}, A3 = {9, 10}, A4 = {5, 8, 1}, A5 = {1, 3, 5}. Find an exact cover of U with respect to the given subsets. Solution A1 is not disjoint from A2 , so A2 cannot be used. A1 , A3 , and A4 are a collection of disjoint sets, but A4 has elements in common with A5 , so A5 is eliminated. However, A1 ∪ A3 ∪ A4 = A and we begin the search again with A2 . A2 is only disjoint from A3 . Again, this is not enough for a cover; A2 ∪ A3 = A. Continuing to check, starting with A3 , A4 , and A5 in turn, we find there is no exact cover of A ◆ with respect to these subsets. The exact cover problem can be restated in a way that leads to an algorithm that can be implemented by a computer program. In Section 1.3, we show that if U is a set with n elements, then each subset of U can be represented by a unique sequence of 0's and 1's of length n. We can use this representation to restate the exact cover problem. If U is a set with n elements, and if S1 , S2 , . . . , Sk are a finite number of subsets of U , we form a matrix M whose rows correspond, in order, to the representations of S1 , S2 , . . . , Sk as sequences of 0's and 1's. Example 3 Represent the subsets A1 , A2 , . . . , A8 of Example 1 by a matrix M. 80 Chapter 2 Logic Solution The set A of Example 1 has 10 elements. The eight subsets can be represented in order by the rows of the following matrix ⎡ ⎤ 1 0 1 1 0 0 0 0 0 0 ⎢ 1 1 0 0 1 1 0 0 0 0 ⎥ ⎢ 0 1 0 0 0 1 1 0 0 0 ⎥ ⎢ 0 0 0 1 0 0 0 1 1 0 ⎥ M= ⎢ ⎥. ⎢ 1 0 0 0 0 0 0 1 0 1 ⎥ ⎢ 0 0 0 0 1 0 0 1 0 0 ⎥ ⎢ ⎥ ⎣ 0 0 1 0 0 0 0 0 1 1 ⎦ 0 0 0 0 0 0 0 0 1 1 ◆ We can now use M to find an exact cover of A with respect to A1 , A2 , A3 , A4 , A5 , A6 , A7 , and A8 . Let 0= 0 0 0 0 0 0 0 0 0 0 and 1= 1 1 1 1 1 1 1 1 1 1 . In matrix terms the problem becomes: Find a collection r1 , r2 , . . . , rk of rows of M having the following properties: (a) ri ∧ r j = 0 for 1 ≤ i, j ≤ k and (b) r1 ∨ r2 ∨ · · · ∨ rk = 1 It is easy to see that one solution is given by rows 1, 3, 6, and 8. This is the solution found before. Once it is reduced to searching a Boolean matrix, the exact cover problem can be more easily tackled with computer-implemented algorithms. The best-known algorithm for solving the exact cover problem is Donald Knuth's Algorithm X. Algorithm ALGORITHM X Label the rows of the original matrix A and retain these labels throughout the application of the algorithm. Begin with M = A and L = { }. Step 1 If M has a column of 0's, then there is no solution, since by definition the rows representing an exact cover must together have 1's in every position. Step 2 Otherwise: (a) Choose the column c of M with the fewest 1's. (b) Choose a row r with a 1 in column c, and place the number r in L. (c) Eliminate any row r1 having the property that r ∧ r1 = 0 (where 0 represents a row with all 0 entries). (d) Eliminate all columns in which r has a 1. (e) Eliminate row r. ● (f) Let M be the resulting matrix and repeat steps 1 and 2. The logic behind (c) is that any such row r1 represents a subset that is not disjoint from the subset of row r. The logic of (d) is that the deleted columns record elements that are already in the subset of row r, and so cannot play any further role in additional rows chosen. 2.6 Logic and Problem Solving 81 We repeat Steps 1 and 2 producing ever smaller matrices, choosing a row each time and accumulating corresponding row labels from A, until the process terminates. Let us call such a succession of choices and computations a path. In general, there may be many paths to follow, since we may have several rows available at each successive stage in the algorithm. A path is successful if the final matrix produced is empty (has no rows or columns), in which case L will contain the row labels of an exact cover. A path fails if the final matrix produced has a column of 0's. Note that if all paths that start with row r fail, there may still be a solution. In this case

[illegible]

of the function defined in Section 1.3, Exercise 38. What is the running time for computing F(n)? 23. (a) Write a recurrence relation to count the number of ways a 3 × 3 square can be placed on an n × n square with the edges of the squares parallel. (b) What is the running time of an algorithm that uses the recurrence relation in (a) to count the number of placements? 24. Prove Rule 3. 25. Prove Rule 4. 26. Prove Rule 6. 27. Prove Rule 7. 5.4 Permutation Functions 28. Prove that if (f) = (g) = (h), then f + g is O(h). 5.4 205 29. Prove that if (f) = (g) and c = 0, then (c f) = (g). Permutation Functions In this section we discuss bijections from a set A to itself. Of special importance is the case when A is finite. Bijections on a finite set occur in a wide variety of applications in mathematics, computer science, and physics. A bijection from a set A to itself is called a permutation of A. Example 1 Let A = R and let f : A → A be defined by f (a) = 2a + 1. Since f is one to one and onto (verify), it follows that f is a permutation of A. ◆ If A = {a1 , a2 , . . . , an } is a finite set and p is a bijection on A, we list the elements of A and the corresponding function values p(a1), p(a2), . . . , p(an) in the following form: a1 a2 ··· an . (1) p(a1) p(a2) · · · p(an) Observe that (1) completely describes p since it gives the value of p for every element of A. We often write a1 a2 ··· an p= . p(a1) p(a2) · · · p(an) Thus, if p is a permutation of a finite set A = {a1 , a2 , . . . , an }, then the sequence p(a1), p(a2), . . . , p(an) is just a rearrangement of the elements of A and so corresponds exactly to a permutation of A in the sense of Section 3.1. Example 2 Example 3 Let A = {1, 2, 3}. 1 1A = 1 1 p3 = 2 Then all the permutations of A are 2 3 1 2 3 1 2 , p1 = , p2 = 2 3 1 3 2 2 1 2 3 1 2 3 1 2 , p4 = , p5 = 3 1 3 1 2 3 2 3 3 3 1 , . ◆ Using the permutations of Example 2, compute (a) p4−1 ; (b) p3 ◦ p2 . Solution (a) Viewing p4 as a function, we have p4 = {(1, 3), (2, 1), (3, 2)}. Then p4−1 = {(3, 1), (1, 2), (2, 3)} or, when written in increasing order of the first component of each ordered pair, we have p4−1 = {(1, 2), (2, 3), (3, 1)}. Thus p4−1 = 1 2 3 2 3 1 = p3 . 206 Chapter 5 Functions (b) The function p2 takes 1 to 2 and p3 takes 2 to 3, so p3 ◦ p2 takes 1 to 3. Also, p2 takes 2 to 1 and p3 takes 1 to 2, so p3 ◦ p2 takes 2 to 2. Finally, p2 takes 3 to 3 and p3 takes 3 to 1, so p3 ◦ p2 takes 3 to 1. Thus 1 2 3. p3 ◦ p2 = 3 2 1 We may view the process of forming p3 ◦ p2 as shown in Figure 5.11. Observe that p3 ◦ p2 = p5 . ◆ ⎛ p3 ◦ p2 = ⎝ 1 2 ? ⎞ ⎛ 2 3 1 2 ⎞ ◦ ⎛ ? ? 2 1 3 1 3 3 ⎞ ⎛ ⎞ = ⎝ 1 2 3 3 2 1 ⎠ ⎠ Figure 5.11 The composition of two permutations is another permutation, usually referred to as the product of these permutations. In the remainder of this chapter, we will follow this convention. THEOREM 1 If A = {a1 , a2 , . . . , an } is a set containing n elements, then there are n! = n · (n − 1) · · · 2 · 1 permutations of A. Proof This result follows from Theorem 4 of Section 3.1 by letting r = n. (2) ■ Let b1 , b2 , . . . , br be r distinct elements of the set A = {a1 , a2 , . . . , an }. The permutation p : A → A defined by p(b1) = b2 p(b2) = b3 .. . p(br −1) = br p(br) = b1 p(x) = x, if x ∈ A, x ∈ / {b1 , b2 , . . . , br }, is called a cyclic permutation of length r , or simply a cycle of length r , and will be denoted by (b1 , b2 , . . . , br). Do not confuse this terminology with that used for cycles in a digraph (Section 4.3). The two concepts are different and we use slightly different notations. If the elements b1 , b2 , . . . , br are arranged uniformly on a circle, as shown in Figure 5.12, then a cycle p of length r moves these elements in a clockwise direction so that b1 is sent to b2 , b2 to b3 , . . . , br −1 to br , and br to b1 . All the other elements of A are left fixed by p. Example 4 Let A = {1, 2, 3, 4, 5}. The cycle (1, 3, 5) denotes the permutation 1 2 3 4 5 . 3 2 5 4 1 ◆ 5.4 Permutation Functions br 207 Observe that if p = (b1 , b2 , . . . , br) is a cycle of length r , then we can also write p by starting with any bi , 1 ≤ i ≤ r , and moving in a clockwise direction, as shown in Figure 5.12. Thus, as cycles, b1 b2 br–1 (3, 5, 8, 2) = (5, 8, 2, 3) = (8, 2, 3, 5) = (2, 3, 5, 8). b3 Figure 5.12 Example 5 Note also that the notation for a cycle does not include the number of elements in the set A. Thus the cycle (3, 2, 1, 4) could be a permutation of the set {1, 2, 3, 4} or of {1, 2, 3, 4, 5, 6, 7, 8}. We need to be told explicitly the set on which a cycle is defined. It follows from the definition that a cycle on a set A is of length 1 if and only if it is the identity permutation, 1 A . Since cycles are permutations, we can form their product. However, as we show in the following example, the product of two cycles need not be a cycle. Let A = {1, 2, 3, 4, 5, 6}. Compute (4, 1, 3, 5)◦(5, 6, 3) and (5, 6, 3)◦(4, 1, 3, 5). Solution We have (4, 1, 3, 5) = and (5, 6, 3) = Then (4, 1, 3, 5) ◦ (5, 6, 3) = = and 1 2 3 4 5 6 3 2 5 1 4 6 1 2 3 4 5 6 1 2 5 4 6 3 1 2 3 4 5 6 1 2 3 4 5 6 3 2 4 1 6 5 . 1 2 3 4 5 6 ◦ 1 2 5 4 6 3 1 2 3 4 5 6 ◦ (5, 6, 3) ◦ (4, 1, 3, 5) = 3 2 5 1 4 6 1 2 3 4 5 6 . = 5 2 6 1 4 3 Observe that 1 2 3 4 5 6 1 2 5 4 6 3 (4, 1, 3, 5) ◦ (5, 6, 3) = (5, 6, 3) ◦ (4, 1, 3, 5) and that neither product is a cycle. ◆ Two cycles of a set A are said to be disjoint if no element of A appears in both cycles. Example 6 Let A = {1, 2, 3, 4, 5, 6}. Then the cycles (1, 2, 5) and (3, 4, 6) are disjoint, ◆ whereas the cycles (1, 2, 5) and (2, 4, 6) are not. It is not difficult to show that if p1 = (a1 , a2 , . . . , ar) and p2 = (b1 , b2 , . . . , bs) are disjoint cycles of A, then p1 ◦ p2 = p2 ◦ p1 . This can be seen by observing that p1 affects only the a's, while p2 affects only the b's. We shall now present a fundamental theorem and, instead of giving its proof, we shall give an example that imitates the proof. 208 Chapter 5 Functions THEOREM 2 Example 7 A permutation of a finite set that is not the identity or a cycle can be written as a ■ product of disjoint cycles of length ≥ 2. Write the permutation p= 1 2 3 4 5 6 7 8 3 4 6 5 2 1 8 7 of the set A = {1, 2, 3, 4, 5, 6, 7, 8} as a product of disjoint cycles. Solution We start with 1 and find that p(1) = 3, p(3) = 6, and p(6) = 1, so we have the cycle (1, 3, 6). Next we choose the first element of A that has not appeared in a previous cycle. We choose 2, and we have p(2) = 4, p(4) = 5, and p(5) = 2, so we obtain the cycle (2, 4, 5). We now choose 7, the first element of A that has not appeared in a previous cycle. Since p(7) = 8 and p(8) = 7, we obtain the cycle (7, 8). We can then write p as a product of disjoint cycles as p = (7, 8) ◦ (2, 4, 5) ◦ (1, 3, 6). ◆ It is not difficult to show that in Theorem 2, when a permutation is written as a product of disjoint cycles, the product is unique except for the order of the cycles. We saw in Section 5.1 how a permutation of the alphabet produces a substitution code. Permutations are also used to produce transposition codes. Unlike a substitution code in which each letter is replaced by a substitute, the letters in transposition coded messages are not changed, but are rearranged. Thus if a message TEST THE WATERS is subjected to the permutation (1, 2, 3) ◦ (4, 7) ◦ (5, 10, 11) ◦ (6, 8, 12, 13, 9), where the numbers refer to the positions of the letters in the message, the message becomes STEEEATHSTTWR. If the permutation is known to both parties, then the receiver of the message has only to apply the inverse permutation to decode. Example 8 One commonly used transposition code is the keyword columnar transposition. For this it is only necessary to remember a keyword, say JONES. The message to be encoded is written under the keyword in successive rows, padding at the end if necessary. For example, the message THE FIFTH GOBLET CONTAINS THE GOLD would be arranged as shown: J T F B O N G O H T L N S O N E H E T T L E F G T A H D S I O C I E X Note that the message has been padded with an X to fill out the row. Then the coded message is constructed by writing the columns in succession, beginning with column 4 (since E is the first keyword letter to appear in the alphabet) and following with the letters in columns 1, 3, 2, 5. The encoded message is thus FGTAHDTFBOGCEHETTLHTLNSOIOCIEX. The recipient of the message divides the number of letters by 5 to find that there are 6 rows. She writes the coded message, six letters at a time, in columns 4, 1, 3, 2, 5, then reads the original message from the rows. ◆ 5.4 Permutation Functions 209 Notice that although the encoded message is a permutation of the original message string, this permutation depends on the length of the message. In Example 8, the permutation of the 30 positions begins 1 2 3 ··· . 7 19 13 · · · But using the keyword JONES to encode MAKE ME AN OFFER produces a permutation of 15 positions that begin 1 2 3 ··· . 4 10 7 · · · A common variation of this idea, used in the U.S. Civil War, is to transpose words rather than letters and add some superfluous known words for confusion. These extra words were called arbitraries during the Civil War. Even and Odd Permutations A cycle of length 2 is called a transposition. That is, a transposition is a cycle p = (ai , a j), where p(ai) = a j and p(a j) = ai . Observe that if p = (ai , a j) is a transposition of A, then p ◦ p = 1 A , the identity permutation of A. Every cycle can be written as a product of transpositions. In fact, (b1 , b2 , . . . , br) = (b1 , br) ◦ (b1 , br −1) ◦ · · · ◦ (b1 , b3) ◦ (b1 , b2). This case can be verified by induction on r , as follows: Basis Step If r = 2, then the cycle is just (b1 , b2), which already has the proper form. Induction Step We use P(k) to show P(k + 1). Let (b1 , b2 , . . . , bk , bk+1) be a cycle of length k + 1. Then (b1 , b2 , . . . , bk , bk+1) = (b1 , bk+1) ◦ (b1 , b2 , . . . , bk), as may be verified by computing the composition. Using P(k), (b1 , b2 , . . . , bk) = (b1 , bk) ◦ (b1 , bk−1) ◦ · · · ◦ (b1 , b2). Thus, by substitution, (b1 , b2 , . . . , bk+1) = (b1 , bk+1) ◦ (b1 , bk) ◦ · · · ◦ (b1 , b3)(b1 , b2). This completes the induction step. Thus, by the principle of mathematical induction, the result holds for every cycle. For example, (1, 2, 3, 4, 5) = (1, 5) ◦ (1, 4) ◦ (1, 3) ◦ (1, 2). We now obtain the following corollary of Theorem 2. Corollary 1 Every permutation of a finite set with at least two elements can be written as a ■ product of transpositions. Observe that the transpositions in Corollary 1 need not be disjoint. Example 9 Write the permutation p of Example 7 as a product of transpositions. 210 Chapter 5 Functions Solution We have p = (7, 8) ◦ (2, 4, 5) ◦ (1, 3, 6). Since we can write (1, 3, 6) = (1, 6) ◦ (1, 3) (2, 4, 5) = (2, 5) ◦ (2, 4), we have p = (7, 8) ◦ (2, 5) ◦ (2, 4) ◦ (1, 6) ◦ (1, 3). ◆ We have observed that every cycle can be written as a product of transpositions. However, this can be done in many different ways. For example, (1, 2, 3) = (1, 3) ◦ (1, 2) = (2, 1) ◦ (2, 3) = (1, 3) ◦ (3, 1) ◦ (1, 3) ◦ (1, 2) ◦ (3, 2) ◦ (2, 3). It then follows that every permutation on a set of two or more elements can be written as a product of transpositions in many ways. However, the following theorem, whose proof we omit, brings some order to the situation. THEOREM 3 If a permutation of a finite set can be written as a product of an even number of transpositions, then it can never be written as a product of an odd number of ■ transpositions, and conversely. A permutation of a finite set is called even if it can be written as a product of an even number of transpositions, and it is called odd if it can be written as a product of an odd number of transpositions. Example 10 Is the permutation p= 1 2 3 4 5 6 7 2 4 5 7 6 3 1 even or odd? Solution We first write p as a product of disjoint cycles, obtaining p = (3, 5, 6) ◦ (1, 2, 4, 7). (Verify this.) Next we write each of the cycles as a product of transpositions: (1, 2, 4, 7) = (1, 7) ◦ (1, 4) ◦ (1, 2) (3, 5, 6) = (3, 6) ◦ (3, 5). Then p = (3, 6) ◦ (3, 5) ◦ (1, 7) ◦ (1, 4) ◦ (1, 2). Since p is a product of an odd number of transpositions, it is an odd permutation. ◆ From the definition of even and odd permutations, it follows (see Exercises 22 through 24) that (a) the product of two even permutations is even. (b) the product of two odd permutations is even. (c) the product of an even and an odd permutation is odd. THEOREM 4 Let A = {a1 , a2 , . . . , an } be a finite set with n elements, n ≥ 2. There are n!/2 even permutations and n!/2 odd permutations. 5.4 Permutation Functions 211 Proof Let An be the set of all even permutations of A, and let Bn be the set of all odd permutations. We shall define a function f : An → Bn , which we show is one to one and onto, and this will show that An and Bn have the same number of elements. Since n ≥ 2, we can choose a particular transposition q0 of A. Say that q0 = (an−1 , an). We now define the function f : An → Bn by f (p) = q0 ◦ p, p ∈ An . Observe that if p ∈ An , then p is an even permutation, so q0 ◦ p is an odd permutation and thus f (p) ∈ Bn . Suppose now that p1 and p2 are in An and f (p1) = f (p2). Then q 0 ◦ p1 = q 0 ◦ p2 . (3) We now compose each side of equation (3) with q0 : q0 ◦ (q0 ◦ p1) = q0 ◦ (q0 ◦ p2); so, by the associative property, (q0 ◦ q0) ◦ p1 = (q0 ◦ q0) ◦ p2 or, since q0 ◦ q0 = 1 A , 1 A ◦ p1 = 1 A ◦ p2 p1 = p2 . Thus f is one to one. Now let q ∈ Bn . Then q0 ◦ q ∈ An , and f (q0 ◦ q) = q0 ◦ (q0 ◦ q) = (q0 ◦ q0) ◦ q = 1 A ◦ q = q, which means that f is an onto function. Since f : An → Bn is one to one and onto, we conclude that An and Bn have the same number of elements. Note that An ∩ Bn = ∅ since no permutation can be both even and odd. Also, by Theorem 1, |An ∪ Bn | = n!. Thus, by Theorem 2 of Section 1.2, n! = |An ∪ Bn | = |An | + |Bn | − |An ∩ Bn | = 2|An |. We then have |An | = |Bn | = n! . 2 ■ 5.4 Exercises 1. Which of the following functions f : R → R are permutations of R? (a) f is defined by f (a) = a − 1. (b) f is defined by f (a) = a 2 . 2. Which of the following functions f : R → R are permutations of R? (a) f is defined by f (a) = a 3 . (b) f is defined by f (a) = ea . 3. Which of the following functions f : Z → Z are permutations of Z? (a) f is defined by f (a) = a + 1. (b) f is defined by f (a) = (a − 1)2 . 4. Which of the following functions f : Z → Z are permutations of Z? (a) f is defined by f (a) = a 2 + 1. (b) f is defined by f (a) = a 3 − 3. In Exercises 5 through 8, let A = {1, 2, 3, 4, 5, 6} and p1 = p2 = p3 = 1 3 2 4 3 1 4 2 5 6 6 5 1 2 2 3 3 1 4 5 5 4 6 6 1 6 2 3 3 2 4 5 5 4 6 1 , , . 212 Chapter 5 Functions 5. Compute (b) p3 ◦ p1 (a) p1−1 6. Compute (b) p1 ◦ (p3 ◦ p2−1) (a) (p2 ◦ p1) ◦ p2 7. Compute (b) p1−1 ◦ p2−1 (a) p3−1 8. Compute (b) p3 ◦ (p2 ◦ p1)−1 (a) (p3 ◦ p2) ◦ p1 In Exercises 9 and 10, let A = {1, 2, 3, 4, 5, 6, 7, 8}. Compute the products. 9. (a) (b) 10. (a) (b) 11. Let (a) (b) (3, 5, 7, 8) ◦ (1, 3, 2) (2, 6) ◦ (3, 5, 7, 8) ◦ (2, 5, 3, 4) (1, 4) ◦ (2, 4, 5, 6) ◦ (1, 4, 6, 7) (5, 8) ◦ (1, 2, 3, 4) ◦ (3, 5, 6, 7) A = {a, b, c, d, e, f, g}. Compute the products. (a, f, g) ◦ (b, c, d, e) (f, g) ◦ (b, c, f) ◦ (a, b, c) In Exercises 12 and 13, let A = {1, 2, 3, 4, 5, 6, 7, 8}. Write each permutation as the product of disjoint cycles. 1 2 3 4 5 6 7 8 12. (a) 4 3 2 5 1 8 7 6 1 2 3 4 5 6 7 8 (b) 2 3 4 1 7 5 8 6 1 2 3 4 5 6 7 8 13. (a) 6 5 7 8 4 3 2 1 1 2 3 4 5 6 7 8 (b) 2 3 1 4 6 7 8 5 14. Let A = {a, b, c, d, e, f, g}. Write each permutation as the product of disjoint cycles. a b c d e f g (a) g d b a c f e a b c d e f g (b) d e a b g f c 15. Let A = {1, 2, 3, 4, 5, 6, 7, 8}. Write each permutation as a product of transpositions. (b) (3, 1, 6) ◦ (4, 8, 2, 5) (a) (2, 1, 4, 5, 8, 6) 16. Code the message WHERE ARE YOU by applying the permutation (1, 7, 3, 5, 11) ◦ (2, 6, 9) ◦ (4, 8, 10). 17. Decode the message ATEHAOMOMNTI, which was encoded using the permutation (3, 7, 1, 12) ◦ (2, 5, 8) ◦ (4, 10, 6, 11, 9). 18. (a) Give the complete permutation of the positions for the message in Example 8. (b) Write the permutation found in part (a) as the product of disjoint cycles. 19. (a) Encode the message MAKE ME AN OFFER using the keyword JONES and the method of Example 8. (b) Write the permutation of the positions for the message in part (a). In Exercises 20 and 21, let A = {1, 2, 3, 4, 5, 6, 7, 8}. Determine whether the permutation is even or odd. 1 2 3 4 5 6 7 8 20. (a) 4 2 1 6 5 8 7 3 1 2 3 4 5 6 7 8 (b) 7 3 4 2 1 8 6 5 21. (a) (6, 4, 2, 1, 5) (b) (4, 8) ◦ (3, 5, 2, 1) ◦ (2, 4, 7, 1) 22. Prove that the product of two even permutations is even. 23. Prove that the product of two odd permutations is even. 24. Prove that the product of an even and an odd permutation is odd. 25. Let A = {1, 2, 3, 4, 5}. Let f = (5, 2, 3) and g = (3, 4, 1) be permutations of A. Compute each of the following and write the result as the product of disjoint cycles. (b) f −1 ◦ g −1 (a) f ◦ g 26. Show that if p is a permutation of a finite set A, then p 2 = p ◦ p is a permutation of A. 27. Let A = {1, 2, 3, 4, 5, 6} and 1 2 3 p= 2 4 3 4 1 5 5 6 6 be a permutation of A. (a) Write p as a product of disjoint cycles. (b) Compute p −1 . (c) Compute p 2 . (d) Find the period of p, that is, the smallest positive integer k such that p k = 1 A . 28. Let A = {1, 2, 3, 4, 5, 6} and 1 2 3 4 5 6 p= 4 3 5 1 2 6 be a permutation of A. (a) Write p as a product of disjoint cycles. (b) Compute p −1 . (c) Compute p 2 . (d) Find the period of p, that is, the smallest positive integer k such that p k = 1 A . 29. (a) Use mathematical induction to show that if p is a permutation of a finite set A, then p n = p ◦ p ◦ · · · ◦ p (n factors) is a permutation of A for n ∈ Z+ . (b) If A is a finite set and p is a permutation of A, show that p m = 1 A for some m ∈ Z+ . 30. Let p be a permutation of a set A. Define the following relation R on A: a R b if and only if p n (a) = b for some n ∈ Z. [p 0 is defined as the identity permutation and p −n is defined as (p −1)n .] Show that R is an equivalence relation and describe the equivalence classes. Key Ideas for Review 213 31. Build a composition table for the permutations of A = {1, 2, 3} given in Example 2. length 3 can be formed using the elements of A and such that a1 < a2 < a3 ? 32. Describe how to use the composition table in Exercise 31 to identify p −1 for any permutation p of A. 37. We call a permutation p = (a1 , a2 , . . . , an) up-down if the elements in the odd positions form an increasing sequence and the elements in the even positions form a decreasing sequence. (a) Let A = {1, 2, 3}. How many up-down permutations of A are there? 33. Find all subsets of {1 A , p1 , p2 , p3 , p4 , p5 }, the permutations in Example 2, that satisfy the closure property for composition. 34. For each permutation, p, of A in Example 2, determine its period. How does this relate to the subset in Exercise 33 to which p belongs? 35. Let A = {1, 2, 3, . . . , n}. How many permutations of A, p = (a1 , a2 , . . . , an), are there for which ai < ai+1 , 1 ≤ i ≤ n − 1? How many permutations of A, p = (a1 , a2 , . . . , an), are there for which ai > ai+1 , 1 ≤ i ≤ n − 1? 36. Let A = {1, 2, 3, 4, 5}. How many different sequences of (b) Let A = {1, 2, 3, 4}. How many up-down permutations of A are there? 38. Let A = {1, 2, 3, 4, 5}. How many up-down permutations of A are there? 39. Prove that the number of up-down permutations for A = {1, 2, 3, . . . , n} is the as the number of increasing 0 same 1 sequences of length n2 that can be formed from elements of A. Tips for Proofs Before beginning a proof, you might find it helpful to consider what the statement does not say. This can help clarify your thinking about what facts and tools are available for the proof. Consider Theorem 4, Section 5.1. It does not say that if f is one to one, then f is onto. The additional facts that |A| = |B| and that f is everywhere defined will need to be used in the proof. To show that a function is one to one or onto, we need to use generic elements. See Example 11, Section 5.1. Either the definition of one-to-oneness or its contrapositive may be used to prove this property. We also have the fact that if f : A → B is everywhere defined and |A| = |B| = n, then f is one to one if and only if f is onto. In addition, if we wish to show f is one to one and onto, we may do this by constructing the inverse function f −1 . Establishing a one-to-one correspondence is a powerful counting strategy, because it allows us to count a different set than the original one. For example, see Theorem 4, Section 5.4, and Exercises 32 through 34, Section 5.2. To prove that f and g have the same order or one is of lower order than the other, the principal tools are the rules for -classes or manipulation of inequalities (Section 5.3, Examples 2 and 3). Key Ideas for Review • Function: see page 181 • Identity function, 1 A : 1 A (a) = a • One-to-one function f from A to B: a = a implies • • • • • • f (a) = f (a) Onto function f from A to B: Ran(f) = B Bijection: one-to-one and onto function One-to-one correspondence: onto, one-to-one, everywhere defined function If f is a function from A to B, 1 B ◦ f = f ; f ◦ 1 A = f If f is an invertible function from A to B, f −1 ◦ f = 1 A ; f ◦ f −1 = 1 B (g ◦ f)−1 = f −1 ◦ g −1 • Substitution code: see page 187 • Boolean function f : Ran(f) ⊆ {true, false} • Hashing function: see page 192 • Fuzzy set: see page 193 • Fuzzy predicate: see page 196 • O(g) (big Oh of g): see page 200 • f and g of the same order: f is O(g) and g is O(f) • Theorem: The relation , f g if and only if f and g have the same order, is an equivalence relation. • Lower -class: see page 201 • Rules for determining -class of a function: see page 203 214 Chapter 5 Functions • Running time of an algorithm: -class of a function that • • • • • • • • describes the number of steps performed by the algorithm Permutation function: a bijection from a set A to itself Theorem: If A is a set that has n elements, then there are n! permutations of A. Cycle of length r : (b1 , b2 , . . . , br); see page 206 Theorem: A permutation of a finite set that is not the identity or a cycle can be written as a product of disjoint cycles. Transposition: a cycle of length 2 Corollary: Every permutation of a finite set with at least two elements can be written as a product of transpositions. Transposition code: see page 208 Keyword columnar transposition code: see page 208 • Even (odd) permutation: one that can be written as a prod- uct of an even (odd) number of transpositions • Theorem: If a permutation of a finite set can be written as a product of an even number of transpositions, then it can never be written as a product of an odd number of transpositions, and conversely. • The product of (a) Two even permutations is even. (b) Two odd permutations is even. (c) An even and an odd permutation is odd. • Theorem: If A is a set that has n elements, then there are n!/2 even permutations and n!/2 odd permutations of A. Chapter 5 Self-Test 1. How does a function differ from a general relation? 2. What is a common strategy to prove that a function is one to one? 3. What is a common strategy to prove that a function is onto? 4. Why are mod functions often used in constructing hashing functions? 5. What does the -class of a function represent? 6. Let A = {a, b, c, d}, B = {1, 2, 3}, and R = {(a, 2), (b, 1), (c, 2), (d, 1)}. Is R a function? Is R −1 a function? Explain your answers. 7. Let A = B = R. Let f : A → B be the function defined by f (x) = −5x + 8. Show that f is one to one and onto. 8. Compute (b) −1.6 (a) 16.29 9. Compute (b) −1.6" (a) 16.29" 10. Compute (b) lg(64) (a) lg(1) 11. Let Q be the propositional function defined by 1 0 Q(x) : ∃y x y = . 0 1 2 1 2 3 Evaluate Q and Q . 0 3 4 6 12. Assume that 9500 account records need to be stored using the hashing function h, which takes the first two digits of the account number as one number and the last four digits as another number, adds them, and then applies the mod-89 function. (a) How many linked lists will be needed? (b) If an approximately even distribution of records is achieved, roughly how many records will be stored in each linked list? (c) Compute h(473810), h(125332), and h(308691). 13. Show that f (n) = 2n 2 + 9n + 5 is O(n 2). 14. Determine the -class of f (n) = lg(n) + n 2 + 2n . 15. Consider the following pseudocode. 1. X ← 10 2. I ← 0 3. UNTIL (I > N) a. X ← X + I b. I ← I + 2 Write a function of N that describes the number of steps required and give the -class of the function. 16. Let A = {1, 2, 3, 4, 5, 6} and let p1 = (3, 6, 2) and p2 = (5, 1, 4) be permutations of A. (a) Compute p1 ◦ p2 and write the result as a product of cycles and as the product of transpositions. (b) Compute p1−1 ◦ p2−1 . 1 2 3 4 5 6 7 and 17. Let p1 = 7 3 2 1 4 5 6 1 2 3 4 5 6 7 . p2 = 6 3 2 1 5 4 7 (a) Compute p1 ◦ p2 . (b) Compute p1−1 . (c) Is p1 an even or odd permutation? Explain. Experiment 5 215 Experiment 5 The -class of a function that describes the number of steps performed by an algorithm is referred to as the running time of the algorithm. In this experiment you will analyze several algorithms, presented in pseudocode, to determine their running times. Part I. The first algorithm is one method for computing the product of two n × n matrices. Assume that the matrices are each stored in an array of dimension 2 and that A[i, j] holds the element of A in row i and column j. Algorithm MATMUL(A, B; C) 1. FOR I = a. FOR 1. 2. 1 THRU N J = 1 THRU N C[I,J] ← 0 FOR K = 1 THRU N a. C[I,J] ← C[I,J] + A[I,K] × B[K,J] ● Assume that each assignment of a value, each addition, and each element multiplication take the same fixed amount of time. 1. How many assignments will be done in the second FOR loop? 2. How many element multiplications are done in the third FOR loop? 3. What is the running time of MATMUL? Justify your answer. Part II. The following recursive algorithm will compute n! for any positive integer n. Algorithm FAC(N) 1. IF (N = 1) THEN a. A ← 1 2. ELSE a. A ← N × FAC(N 3. RETURN (A) − 1) ● 1. Let Sn be the number of steps needed to calculate n! using FAC. Write a recurrence relation for Sn in terms of Sn−1 . 2. Solve the recurrence relation in question 1 and use the result to determine the running time of FAC. Part III. The function SEEK will give the cell in which a specified value is stored in cells i through i + n − 1 (inclusive) of an array A. Assume that i ≥ 1. FUNCTION SEEK(ITEM, I, I + N 1. CELL ← 0 2. FOR J = I THRU I + N − 1 a. IF (A[J] = ITEM) THEN b. CELL ← J 3. RETURN (CELL) END OF FUNCTION SEEK − 1) 1. How many cells are there from A[i] to A[i + n − 1] (inclusive)? 2. Give a verbal description of how SEEK operates. ● 216 Chapter 5 Functions 3. What is the running time of SEEK? Justify your answer. Part IV. The algorithm HUNT will give the cell in which a specified value is stored in cells i through i + n − 1 (inclusive) of an array A. Assume that i ≥ 1. To simplify the analysis of this algorithm, assume that n, the number of cells to be inspected, is a power of 2. Algorithm HUNT (ITEM , I , I + N − 1) 1. CELL ← 0 2. IF (N = 1 AND A[I] = ITEM) THEN a. CELL ← I 3. ELSE a. CELL1 ← HUNT(ITEM, I, I + N/2 − 1) b. CELL2 ← HUNT(ITEM, I + N/2, I + N − 1) 4. IF (CELL1 = 0) THEN a. CELL ← CELL1 5. ELSE a. CELL ← CELL2 6. RETURN (CELL) ● 1. Give a verbal description of how HUNT operates. 2. What is the running time of HUNT? Justify your answer. 3. Under what circumstances would it be better to use SEEK (Part III) rather than HUNT? When would it be better to use HUNT rather than SEEK? CHAPTER 6 Order Relations and Structures Prerequisites: Chapter 4 In this chapter we study partially ordered sets, including lattices and Boolean algebras. These structures are useful in set theory, algebra, sorting and searching, and, especially in the case of Boolean algebras, in the construction of logical representations for computer circuits. Looking Back We have already met George Boole in Chapter 2 and noted the importance of his development of symbolic logic. In 1938, Claude E. Shannon, an American mathematician (whom we will meet in Chapter 11) wrote a Master's thesis at MIT entitled A Symbolic Analysis of Relay and Switching Circuits, in which he showed how to use the work of Boole in the design of circuits. The heart of a computer consists of circuits that perform the many different operations and tasks of the computer. Every electronic circuit can be written as a combination of the connectors and, or, and not, and by using a device that can be turned on or off, we can construct each type of connector. Since this combination is not unique, it is desirable to write it in as simple a manner as possible (a problem that will be examined briefly in this chapter). The benefits of doing so include greater speed, smaller circuits, and lower power consumption. The field in computer science that deals with this problem is logic design, and Boolean algebra plays a major role in this field. As electronic circuits have grown enormously in size and complexity, the field of computer-aided design (CAD) has been developed, whereby very sophisticated computer programs carry out the logic design. circuit 217 218 Chapter 6 Order Relations and Structures 6.1 Partially Ordered Sets A relation R on a set A is called a partial order if R is reflexive, antisymmetric, and transitive. The set A together with the partial order R is called a partially ordered set, or simply a poset, and we will denote this poset by (A, R). If there is no possibility of confusion about the partial order, we may refer to the poset simply as A, rather than (A, R). Example 1 Let A be a collection of subsets of a set S. The relation ⊆ of set inclusion is a partial order on A, so (A, ⊆) is a poset. ◆ Example 2 Let Z+ be the set of positive integers. The usual relation ≤ (less than or equal to) is a partial order on Z+ , as is ≥ (greater than or equal to). ◆ Example 3 The relation of divisibility (a R b if and only if a | b) is a partial order on Z+ . ◆ Example 4 Let R be the set of all equivalence relations on a set A. Since R consists of subsets of A × A, R is a partially ordered set under the partial order of set containment. If R and S are equivalence relations on A, the same property may be expressed in relational notation as follows. R ⊆ S if and only if x R y implies x S y for all x, y in A. Then (R, ⊆) is a poset. ◆ Example 5 The relation < on Z+ is not a partial order, since it is not reflexive. ◆ Example 6 Let R be a partial order on a set A, and let R −1 be the inverse relation of R. Then R −1 is also a partial order. To see this, we recall the characterization of reflexive, antisymmetric, and transitive given in Section 4.4. If R has these three properties, then ⊆ R, R ∩ R −1 ⊆ , and R 2 ⊆ R. By taking inverses, we have = −1 ⊆ R −1 , R −1 ∩ (R −1)−1 = R −1 ∩ R ⊆ , and (R −1)2 ⊆ R −1 , so, by Section 4.4, R −1 is reflexive, antisymmetric, and transitive. Thus R −1 is also a partial order. The poset (A, R −1) is called the dual of the poset (A, R), and the partial order R −1 is called the dual of the partial order R. ◆ The most familiar partial orders are the relations ≤ and ≥ on Z and R. For this reason, when speaking in general of a partial order R on a set A, we shall often use the symbols ≤ or ≥ for R. This makes the properties of R more familiar and easier to remember. Thus the reader may see the symbol ≤ used for many different partial orders on different sets. Do not mistake this to mean that these relations are all the same or that they are the familiar relation ≤ on Z or R. If it becomes absolutely necessary to distinguish partial orders from one another, we may also use symbols such as ≤1 , ≤ , ≥1 , ≥ , and so on, to denote partial orders. We will observe the following convention. Whenever (A, ≤) is a poset, we will always use the symbol ≥ for the partial order ≤−1 , and thus (A, ≥) will be the dual poset. Similarly, the dual of poset (A, ≤1) will be denoted by (A, ≥1), and the dual of the poset (B, ≤) will be denoted by (B, ≥). Again, this convention is to remind us of the familiar dual posets (Z, ≤) and (Z, ≥), as well as the posets (R, ≤) and (R, ≥). If (A, ≤) is a poset, the elements a and b of A are said to be comparable if a≤b or b ≤ a. 6.1 Partially Ordered Sets 219 Observe that in a partially ordered set every pair of elements need not be comparable. For example, consider the poset in Example 3. The elements 2 and 7 are not comparable, since 2 7 and 7 2. Thus the word "partial" in partially ordered set means that some elements may not be comparable. If every pair of elements in a poset A is comparable, we say that A is a linearly ordered set, and the partial order is called a linear order. We also say that A is a chain. Example 7 ◆ The poset of Example 2 is linearly ordered. The following theorem is sometimes useful since it shows how to construct a new poset from given posets. THEOREM 1 If (A, ≤) and (B, ≤) are posets, then (A × B, ≤) is a poset, with partial order ≤ defined by (a, b) ≤ (a , b) if a ≤ a in A and b ≤ b in B. Note that the symbol ≤ is being used to denote three distinct partial orders. The reader should find it easy to determine which of the three is meant at any time. Proof If (a, b) ∈ A × B, then (a, b) ≤ (a, b) since a ≤ a in A and b ≤ b in B, so ≤ satisfies the reflexive property in A × B. Now suppose that (a, b) ≤ (a , b) and (a , b) ≤ (a, b), where a and a ∈ A and b and b ∈ B. Then a ≤ a and a ≤ a in A b ≤ b and b ≤ b in B. Since A and B are posets, the antisymmetry of the partial orders on A and B implies that a = a and b = b . Hence ≤ satisfies the antisymmetry property in A × B. Finally, suppose that (a, b) ≤ (a , b) and (a , b) ≤ (a , b), where a, a , a ∈ A, and b, b , b ∈ B. Then a ≤ a and a ≤ a , so a ≤ a , by the transitive property of the partial order on A. Similarly, b ≤ b and b ≤ b , so b ≤ b , by the transitive property of the partial order on B. Hence (a, b) ≤ (a , b). Consequently, the transitive property holds for the partial order on A × B, and we ■ conclude that A × B is a poset. The partial order ≤ defined on the Cartesian product A × B is called the product partial order. If (A, ≤) is a poset, we say that a < b if a ≤ b but a = b. Suppose now that (A, ≤) and (B, ≤) are posets. In Theorem 1 we have defined the product partial 220 Chapter 6 Order Relations and Structures order on A × B. Another useful partial order on A × B, denoted by ≺, is defined as follows: (a, b) ≺ (a , b) if a < a or if a = a and b ≤ b . This ordering is called lexicographic, or "dictionary" order. The ordering of the elements in the first coordinate dominates, except in case of "ties," when attention passes to the second coordinate. If (A, ≤) and (B, ≤) are linearly ordered sets, then the lexicographic order ≺ on A × B is also a linear order. Example 8 p3 Let A = R, with the usual ordering ≤. Then the plane R2 = R × R may be given lexicographic order. This is illustrated in Figure 6.1. We see that the plane is linearly ordered by lexicographic order. Each vertical line has the usual order, and points on one line are less than points on a line farther to the right. Thus, in Figure 6.1, p1 ≺ p2 , p1 ≺ p3 , and p2 ≺ p3 . ◆ p1 p2 x x = x1 x = x2 Figure 6.1 Lexicographic ordering is A1 × A2 × · · · × An as follows: easily extended to Cartesian products (a1 , a2 , . . . , an) ≺ (a1 , a2 , . . . , an) if and only if a1 < a1 or a1 = a1 and a2 < a2 or a1 = a1 , a2 = a2 , and a3 < a3 or . . . a1 = a1 , a2 = a2 , . . . , an−1 = an−1 and an ≤ an . Thus the first coordinate dominates except for equality, in which case we consider the second coordinate. If equality holds again, we pass to the next coordinate, and so on. Example 9 Let S = {a, b, . . . , z} be the ordinary alphabet, linearly ordered in the usual way (a ≤ b, b ≤ c, . . . , y ≤ z). Then S n = S × S × · · · × S (n factors) can be identified with the set of all words having length n. Lexicographic order on S n has the property that if w1 ≺ w2 (w1 , w2 ∈ S n), then w1 would precede w2 in a dictionary listing. This fact accounts for the name of the ordering. Thus park ≺ part, help ≺ hind, jump ≺ mump. The third is true since j ≺ m; the second, since h = h, e ≺ i; and the first is true since p = p, a = a, r = r , ◆ k ≺ t. If S is a poset, we can extend lexicographic order to S ∗ (see Section 1.3) in the following way. If x = a1 a2 · · · an and y = b1 b2 · · · bk are in S ∗ with n ≤ k, we say that x ≺ y if (a1 , . . . , an) ≺ (b1 , . . . , bn) in S n under lexicographic ordering of S n . In the previous paragraph, we use the fact that the n-tuple (a1 , a2 , . . . , an) ∈ S n , and the string a1 a2 · · · an ∈ S ∗ are really the same sequence of length n, written in two different notations. The notations differ for historical reasons, and we will use them interchangeably depending on context. 6.1 Partially Ordered Sets Example 10 221 Let S be {a, b, . . . , z}, ordered as usual. Then S ∗ is the set of all

possible "words" of any length, whether such words are meaningful or not. Thus we have help < helping in S • since help < help 4 in S . Similarly, we have since helper < helping helper < helpin in S 6 . As the example help < helping shows, this order includes prefix order; that is, any word is greater than all of its prefixes (beginning parts). This is also the way that words occur in the dictionary. Thus we have dictionary ordering again, but this time for words of any finite length. ● Since a partial order is a relation, we can look at the digraph of any partial order on a finite set. We shall find that the digraphs of partial orders can be represented in a simpler manner than those of general relations. The following theorem provides the first result in this direction. THEOREM 2 The digraph of a partial order has no cycle of length greater than 1. Proof Suppose that the digraph of the partial order ≤ on the set A contains a cycle of length n ≥ 2. Then there exist distinct elements a1 , a2 , . . . , an ∈ A such that a1 ≤ a2, a2 ≤ a3 , . . . , an−1 ≤ an, an ≤ a1 . By the transitivity of the partial order, used n − 1 times, a1 ≤ an . By antisymmetry, an ≤ a1 and a1 ≤ an imply that an = a1 , a contradiction to the assumption that a1 , a2 , . . . , an are distinct. ■ Hasse Diagrams Let A be a finite set. Theorem 2 has shown that the digraph of a partial order on A has only cycles of length 1. Indeed, since a partial order is reflexive, every vertex in the digraph of the partial order is contained in a cycle of length 1. To simplify matters, we shall delete all such cycles from the digraph. Thus the digraph shown in Figure 6.2(a) would be drawn as shown in Figure 6.2(b). We shall also eliminate all edges that are implied by the transitive property. Thus, if a ≤ b and b ≤ c, it follows that a ≤ c. In this case, we omit the edge from a to c; however, we do draw the edges from a to b and from b to c. For example, the digraph shown in Figure 6.3(a) would be drawn as shown in Figure 6.3(b). We also agree to draw the digraph of a partial order with all edges pointing upward, so that arrows may be omitted from the edges. Finally, we replace the circles representing the vertices by dots. Thus the diagram shown in Figure 6.4 gives the final form of the digraph shown in Figure 6.2(a). The resulting diagram of a partial order, much simpler than its digraph, is called the Hasse diagram of the partial order of the poset. Since the Hasse diagram completely describes the associated partial order, we shall find it to be a very useful tool. 222 Chapter 6 Order Relations and Structures c c b b a a (a) (b) Figure 6.2 c c b b a a a (a) (b) c c (a) Figure 6.3 (b) Figure 6.4 Example 1 Let A = {1, 2, 3, 4, 12}. Consider the partial order of divisibility on A. That is, if a and b ∈ A, a ≤ b if and only if a | b. Draw the Hasse diagram of the poset (A, ≤). Solution The Hasse diagram is shown in Figure 6.5. To emphasize the simplicity of the Hasse diagram, we show in Figure 6.6 the digraph of the poset in Figure 6.5. 12 12 4 4 3 3 2 2 1 1 Figure 6.5 Example 12 Figure 6.6 Let S = {a, b, c} and A = P(S). Draw the Hasse diagram of the poset A with the partial order ⊆ (set inclusion). Solution We first determine A, obtaining A = {∅, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c}}. The Hasse diagram can then be drawn as shown in Figure 6.7. ● {a, b, c} {a, b} {a, c} {b, c} {b} {a} {c} ∅ Figure 6.7 Figure 6.8 Observe that the Hasse diagram of a finite linearly ordered set is always of the form shown in Figure 6.8. It is easily seen that if (A, ≤) is a poset and (A, ≥) is the dual poset, then the Hasse diagram of (A, ≥) is just the Hasse diagram of (A, ≤) turned upside down. Example 13 Figure 6.9(a) shows the Hasse diagram of a poset (A, ≤), where A = {a, b, c, d, e, f}. Figure 6.9(b) shows the Hasse diagram of the dual poset (A, ≥). Notice that, as stated, each of these diagrams can be constructed by turning the other upside down. ● a f b c e d a f b c e d (a) (b) Figure 6.9 Topological Sorting If A is a poset with partial order ≤, we sometimes need to find a linear order < for the set A that will merely be an extension of the given partial order in the sense that if a ≤ b, then a < b. The process of constructing a linear order such as < is called topological sorting. This problem might arise when we have to enter a finite poset A into a computer. The elements of A must be entered in some order, and we might want them entered so that the partial order is preserved. That is, if a ≤ b, then a is entered before b. A topological sorting < will give an order of entry of the elements that meets this condition. Example 14 Give a topological sorting for the poset whose Hasse diagram is shown in Figure 6.10. Solution The partial order < whose Hasse diagram is shown in Figure 6.11(a) is clearly a linear order. It is easy to see that every pair in ≤ is also in the order <, so < is a topological sorting of the partial order ≤. Figures 6.11(b) and (c) show two other solutions to this problem. ● f g f f f e d c e g b d e b c g e b d c a (a) (b) (c) Figure 6.10 a (b) (c) Figure 6.11 Isomorphism Let (A, ≤) and (A′, ≤′) be posets and let f : A → A′ be a one-to-one correspondence between A and A′. The function f is called an isomorphism from (A, ≤) to (A′, ≤′) if, for any a and b in A, a ≤ b if and only if f (a) ≤′ f (b). If f : A → A is an isomorphism, we say that (A, ≤) and (A′, ≤′) are isomorphic posets. Example 15 Let A be the set Z+ of positive integers, and let ≤ be the usual partial order on A (see Example 2). Let A′ be the set of positive even integers, and let ≤′ be the usual partial order on A′. The function f : A → A′ given by f (a) = 2a is an isomorphism from (A, ≤) to (A′, ≤′). First, f is one to one since, if f (a) = f (b), then 2a = 2b, so a = b. Next, Dom(f) = A, so f is everywhere defined. Finally, if c ∈ A′, then c = 2a for some a ∈ Z+; therefore, c = f (a). This shows that f is onto, so we see that f is a one-to-one correspondence. Finally, if a and b are elements of A, then it is clear that a ≤ b if and only if 2a ≤ 2b. Thus f is an isomorphism. ● Suppose that f : A → A′ is an isomorphism from a poset (A, ≤) to a poset (A′, ≤′). Suppose also that B is a subset of A, and B′ = f (B) is the corresponding subset of A′. Then we see from the definition of isomorphism that the following principle must hold. THEOREM 3 Principle of Correspondence If the elements of B have any property relating to one another or to other elements of A, and if this property can be defined entirely in terms of the relation ≤, then the elements of B′ must possess exactly the same property, defined in terms of ≤′. 6.1 Partially Ordered Sets 225 ■ Example 16 a b c d Figure 6.12 Let (A, ≤) be the poset whose Hasse diagram is shown in Figure 6.12, and suppose that f is an isomorphism from (A, ≤) to some other poset (A′, ≤′). Note first that d ≤ x for any x in A (later we will call an element such as d a "least element" of A). Then the corresponding element f (d) in A′ must satisfy the property f (d) ≤ y for all y in A′. As another example, note that a and b are incomparable in A. Such a pair is called incomparable in A. It then follows from the principle of correspondence that f (a) and f (b) must be incomparable in A′. ● For a finite poset, one of the objects that is defined entirely in terms of the partial order is its Hasse diagram. It follows from the principle of correspondence that two finite isomorphic posets must have the same Hasse diagrams. To be precise, let (A, ≤) and (A′, ≤′) be finite posets, let f : A → A′ be a one-to-one correspondence, and let H be any Hasse diagram of (A, ≤). Then 1. If f is an isomorphism and each label a of H is replaced by f (a), then H will become a Hasse diagram for (A′, ≤′). Conversely, 2. If H becomes a Hasse diagram for (A′, ≤′), whenever each label a is replaced by f (a), then f is an isomorphism. This justifies the name "isomorphism," since isomorphic posets have the same (iso-) "shape" (morph) as described by their Hasse diagrams. Example 17 Let A = {1, 2, 3, 6} and let ≤ be the relation | (divides). Figure 6.13(a) shows the Hasse diagram for (A, ≤). Let A′ = P({a, b}) = {∅, {a}, {b}, {a, b}}, and let ≤′ be set containment, ⊆. If f : A → A′ is defined by f (1) = ∅, f (2) = {a}, f (3) = {b}, f (6) = {a, b}, then it is easily seen that f is a one-to-one correspondence. If each label a ∈ A of the Hasse diagram is replaced by f (a), the result is as shown in Figure 6.13(b). Since this is clearly a Hasse diagram for (A′, ≤′), the function f is an ● isomorphism. {a, b} 6 3 2 1 {a} (a) Figure 6.13 {b} ∅ (b) Chapter 6 Order Relations and Structures 226 6.1 Exercises 1. Determine whether the relation R is a partial order on the set A. (a) A = Z, and a R b if and only if a = 2b. (b) A = Z, and a R b if and only if b² | a. 2. Determine whether the relation R is a partial order on the set A. (a) A = Z, and a R b if and only if a = bk for some k ∈ Z+. Note that k depends on a and b. (b) A = R, and a R b if and only if a ≤ b. 3. Determine whether the relation R is a partial order on the set A. (a) A = P(S), where S is a set. The relation R is set inclusion. (b) A = R × R, and (a, b) R (a′, b′) if and only if a ≤ a′ and b ≤ b′, where ≤ is the usual partial order on R. 5. On the set A = {a, b, c}, find all partial orders ≤ in which a ≤ b. 6. What can you say about the relation R on a set A if R is a partial order and an equivalence relation? 5 2 1 7. Outline the structure of the proof given for Theorem 1. 8. Outline the structure of the proof given for Theorem 2. In Exercises 9 and 10, determine the Hasse diagram of the relation R. 9. A = {1, 2, 3, 4}, R = {(1, 1), (1, 2), (2, 2), (2, 4), (1, 3), (3, 3), (3, 4), (1, 4), (4, 4)}. 10. A = {a, b, c, d, e}, R = {(a, a), (b, b), (c, c), (a, c), (c, d), (c, e), (a, d), (d, d), (a, e), (b, c), (b, d), (b, e), (e, e)}. In Exercises 11 and 12, describe the ordered pairs in the relation determined by the Hasse diagram on the set A = {1, 2, 3, 4} (Figures 6.14 and 6.15). 11. 12. 4 Figure 6.17 15. Determine the Hasse diagram of the relation on A = {1, 2, 3, 4, 5} whose matrix is shown. [1 0 [0] 1 [0 [0 0] 0 1 0 0 0 0 1 1 0 1 1 1 0 0 1 1 1 1] 16. Determine the Hasse diagram of the relation on A = {1, 2, 3, 4, 5} whose matrix is shown. 2 2 1 Figure 6.14 4 Figure 6.15 In Exercises 13 and 14, determine the Hasse diagram of the partial order having the given digraph (Figures 6.16 and 6.17). 13. [1 0 0 0 1 0 0 0 1 1 0 0 1 1 1 0 0 1 1 1 1] [1] [ ] [1] 6.1 Partially Ordered Sets 227 In Exercises 17 and 18, determine the matrix of the partial order whose Hasse diagram is given (Figures 6.18 and 6.19). In Exercises 29 and 30, draw the Hasse diagram of a topological sorting of the given poset (Figures 6.20 and 6.21). 17. 2 29 3 4 5 18 4 5 3 0 8 2 7 2 3 1 Figure 6.18 6 4 1 19. Let A = { A, B, C, E, O, M, P, S} have the usual alphabetical order, where represents a "blank" character and ≤ x for all x ∈ A. Arrange the following in lexicographic order (as elements of A × A × A × A). (b) MOPE (c) CAP (a) MOP (d) BASE (f) ACE (d) MAP (g) MACE (h) CAPE 20. Let A = Z+ × Z+ have lexicographic order. Mark each of the following as true or false. (a) (2, 12) ≺ (5, 3) (d) (15, 92) ≺ (12, 3) (c) (4, 6) ≺ (4, 2) In Exercises 21 through 24, consider the partial order of divisibility on the set A. Draw the Hasse diagram of the poset and determine which posets are linearly ordered. 21. A = {1, 2, 3, 5, 6, 10, 15, 30} 22. A = {2, 4, 8, 16, 32} 23. A = {3, 6, 12, 36, 72} 24. A = {1, 2, 3, 4, 5, 6, 10, 12, 15, 30, 60} Describe how to use M R to determine if R is a partial order. 26. A partial order may or may not be a linear order, but any poset can be partitioned into subsets that each are a linear order. Give a partition into linearly ordered subsets with as few subsets as possible for each of the following. The poset whose Hasse diagram is given in (b) Figure 6.9(a) (a) Figure 6.5 (d) Figure 6.20 (c) Figure 6.15 (e) Figure 6.21 27. For each of the posets whose Hasse diagrams are given in Exercise 26, find the smallest number of subsets needed to form a partition (b) Figure 6.5 (d) Figure 6.21 6.28. What is the relationship between the smallest number of subsets needed to partition a partial order into linearly ordered subsets and the cardinality of the largest set of incomparable elements in the partial order? Justify your response. 8 7 5 Figure 6.19 21. 22. 23. 24. 25. 9 2 6 1 3 1 5 4 Figure 6.21 Determine all topological sortings for the poset whose Hasse diagram is shown in Figure 6.20. 32. Show that every finite poset can be partitioned into a finite number of linearly ordered subsets. 33. (Finding a subset (A, ≤)) If (A, ≤) is a poset and A is a subset of A, show that (B, ≤) is also a poset, where ≤ is the restriction of ≤ to B. 34. Show that if R is a linear order on the set A, then R−1 is also a linear order on A. 35. A relation R on a set A is called a quasiorder if it is transitive and irreflexive. Let A = P(S) be the power set of a set S, and consider the following relation R on A: U R T if and only if U ⊂ T (proper containment). Show that R is a quasiorder. 34. Let A = {x | x is a real number and −5 ≤ x ≤ 20}. Show that the usual relation < is a quasiorder (see Exercise 33) on A. 35. If R is a quasiorder on A (see Exercise 33), prove that R−1 is also a quasiorder on A. 36. Modify the relation in Example 3 to produce a quasiorder on Z+. 37. Let B = {2, 3, 6, 9, 12, 18, 24} and let A = B × B. Define the following relation on A: (a, b) ≺ (a′, b′) if and only if a | a′ and b ≤ b′, where ≤ is the usual partial order. Show that ≺ is a partial order. 38. Let S be the set of 2 × 2 Boolean matrices with M ≤ N if and only if mij ≤ nij, 1 ≤ i ≤ 2, 1 ≤ j ≤ 2. Prove that R is a partial order on S. 39. Let A = {1, 2, 3, 5, 6, 10, 15, 30}. Let B = {∅, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c}}. Show that (A, |) and (B, ⊆) are isomorphic. 40. Let A = {1, 2, 3, 5, 6, 10, 15, 30}, that is, the set of all positive divisors of 30. Show that (A, |) and (B, ⊆) are isomorphic. 41. Prove that if two finite posets (A, ≤) and (B, ≤) are isomorphic, ... Let R be a relation on A. Let A = {0, 1, 2, 3} and let ≤ be the relation R on A. 42. Show that the poset of nonnegative real numbers with the usual partial order "less than or equal to" on integers. Show that (A, ≤) and (A, ≥) are isomorphic posets. 6.2 Extremal Elements of Partially Ordered Sets Certain elements in a poset are of special importance for many of the properties and applications of posets. In this section we discuss these elements, and in later sections we shall see the important role played by them. In this section we consider a poset (A, ≤). An element a ∈ A is called a maximal element of A if there is no element c in A such that a < c (see Section 6.1). An element b ∈ A is called a minimal element of A if there is no element c in A such that c < b. It follows immediately that, if (A, ≤) is a poset and (A, ≥) is its dual poset, an element a ∈ A is a maximal element of (A, ≥) if and only if a is a minimal element of (A, ≤). Also, a is a minimal element of (A, ≥) if and only if it is a maximal element of (A, ≤). Example 1 Consider the poset A whose Hasse diagram is shown in Figure 6.22. The elements a1, a2, and a3 are maximal elements of A, and the elements b1, b2 , and b3 are minimal elements. Observe that, since there is no line between b2 and b3, we can conclude neither that b3 ≤ b2 nor that b2 ≤ b3. ● Example 2 Let A be the poset of nonnegative real numbers with the usual partial order ≤. Then 0 is a minimal element of A. There are no maximal elements of A. ● Example 3 The poset Z with the usual partial order ≤ has no maximal elements and has no minimal elements. ● THEOREM 1 Let A be a finite nonempty poset with partial order ≤. Then A has at least one maximal element and at least one minimal element. Proof Let a be any element of A. If a is not maximal, we can find an element a1 ∈ A such that a < a1. If a1 is not maximal, we can find an element a2 ∈ A such that a1 < a2 . This argument cannot be continued indefinitely, since A is a finite set. Thus we eventually obtain the finite chain a < a1 < a2 < · · · < ak−1 < ak , which cannot be extended. Hence we cannot have ak < b for any b ∈ A, so ak is a maximal element of (A, ≤). This same argument says that the dual poset (A, ≥) has a maximal element, so (A, ≤) has a minimal element. ■ By using the concept of a minimal element, we can give an algorithm for finding a topological sorting of a given finite poset (A, ≤). We remark first that if a ∈ A and B = A − {a}, then B is also a poset under the restriction of ≤ to B × B (see Section 6.1). We then have the following algorithm, which produces a linear array named SORT. We assume that SORT is ordered by increasing index, that is, SORT[1] < SORT[2] < · · · . The relation < on A defined in this way is a topological sorting of (A, ≤). 6.2 Extremal Elements of Partially Ordered Sets 229 Algorithm For finding a topological sorting of a finite poset (A, ≤). Step 1 Choose a minimal element a of A. Step 2 Make a the next entry of SORT and replace A with A − {a}. Step 3 Repeat steps 1 and 2 until A = { }. Example 4 ■ Let A = {a, b, c, d, e}, and let the Hasse diagram of a partial order ≤ on A be as shown in Figure 6.23(a). A minimal element of this poset is the vertex labeled d (we could also have chosen e). We put d in SORT[1] and in Figure 6.23(b) we show the Hasse diagram of A − {d}. A minimal element of the new A is e, so e becomes SORT[2], and A − {e} is shown in Figure 6.23(c). This process continues until we have exhausted A and filled SORT. Figure 6.23(f) shows the completed array SORT and the Hasse diagram of the poset corresponding to SORT. This is a topological sorting of (A, ≤). ● a b c d e a b c e a b c a b (a) (b) (c) (d) (e) SORT d e c b a (f) Figure 6.23 An element a ∈ A is called a greatest element of A if x ≤ a for all x ∈ A. An element a ∈ A is called a least element of A if a ≤ x for all x ∈ A. As before, an element a of (A, ≤) is a greatest (or least) element if and only if it is a least (or greatest) element of (A, ≥). Example 5 Consider the poset defined in Example 2. Then 0 is a least element; there is no greatest element. ● Example 6 Let S = {a, b, c} and consider the poset A = P(S) defined in Example 12 of Section 6.1. The empty set is a least element of A, and the set S is a greatest element of A. ● Example 7 The poset Z with the usual partial order has neither a least nor a greatest element. ● THEOREM 2 A poset has at most one greatest element and at most one least element. Proof Suppose that a and b are greatest elements of a poset A. Then, since b is a greatest element, we have a ≤ b. Similarly, since a is a greatest element, we have b ≤ a. Hence a = b by the antisymmetry property. Thus, if the poset has a greatest element, it only has one such element. Since this fact is true for all posets, the dual poset (A, ≥) has at most one greatest element, so (A, ≤) also has at most one least element. ■ The greatest element of a poset, if it exists, is denoted by I and is often called the unit element. Similarly, the least element of a poset, if it exists, is denoted by 0 and is often called the zero element. Consider a poset A and a subset B of A. An element a ∈ A is called an upper bound of B if b ≤ a for all b ∈ B. An element a ∈ A is called a lower bound of B if a ≤ b for all b ∈ B. Example 8 b f g d c e a b Consider the poset A = {a, b, c, d, e, f, g, h}, whose Hasse diagram is shown in Figure 6.24. Find all upper and lower bounds of the following subsets of A: (a) B1 = {a, b}; (b) B2 = {c, d, e}. Solution (a) B1 has no lower bounds; its upper bounds are c, d, e, f, g, and h. (b) The upper bounds of B2 are f, g, and h; its lower bounds are c, a, and b. ● As Example 8 shows, a subset B of a poset may or may not have upper or lower bounds (in A). Moreover, an upper or lower bound of B may or may not belong to B itself. Let A be a poset and B a subset of A. An element a ∈ A is called a least upper bound of B, (LUB(B)), if a is an upper bound of B and a ≤ a′, whenever a′ is an upper bound of B. Thus a = LUB(B) if b ≤ a for all b ∈ B, and if whenever a′ ∈ A is also an upper bound of B, then a ≤ a′. Similarly, an element a ∈ A is called a greatest lower bound of B, (GLB(B)), if a is a lower bound of B and a′ ≤ a, whenever a′ is a lower bound of B. Thus a = GLB(B) if a ≤ b for all b ∈ B, and if whenever a′ ∈ A is also a lower bound of B, then a′ ≤ a. As usual, upper bounds in (A, ≤) correspond to lower bounds in (A, ≥) (for the same set of elements), and lower bounds in (A, ≤) correspond to upper bounds in (A, ≥). Similar statements hold for greatest lower bounds and least upper bounds. Example 9 Let A be the poset considered in Example 8 with subsets B1 and B2 as defined in that example. Find all least upper bounds and all greatest lower bounds of (a) B1 ; (b) B2. Solution (a) Since B1 has no lower bounds, it has no greatest lower bounds. However, LUB(B1) = c. (b) Since the lower bounds of B2 are c, a, and b, we find that GLB(B2) = c. The upper bounds of B2 are f, g, and h. Since f and g are not comparable, we conclude that B2 has no least upper bound. THEOREM 3 231 The proof is similar to the proof of Theorem 2. 10 9 6 8 7 5 3 2 1 Let (A, ≤) be a poset. Then a subset B of A has at most one LUB and at most one GLB. ■ We conclude this section with some remarks about LUB and GLB in a finite poset A, as viewed from the Hasse diagram of A. Let B = {b1 , b2 , . . . , br }. If a = LUB(B), then a is the first vertex that can be reached from b1, b2 , . . . , br by upward paths. 4 Figure 6.25 Example 10 Let A = {1, 2, 3, 4, 5, . . . , 11} be the poset whose Hasse diagram is shown in Figure 6.25. Find the LUB and GLB of B = {6, 7, 10}, if they exist. Solution Exploring all upward paths from vertices 6, 7, and 10, we find that LUB(B) = 10. Similarly, by examining all downward paths from 6, 7, and 10, we find that GLB(B) = 4. ● The next result follows immediately from the principle of correspondence (see Section 6.1). THEOREM 4 Suppose that (A, ≤) and (A′, ≤′) are isomorphic posets under the isomorphism f : A → A′. (a) If a is a maximal (minimal) element of (A, ≤), then f (a) is a maximal (minimal) element of (A′, ≤′). (b) If a is the greatest (least) element of (A, ≤), then f (a) is the greatest (least) element of (A′, ≤′). (c) If a is an upper bound (lower bound, least upper bound, greatest lower bound) of a subset B of A, then f (a) is an upper bound (lower bound, least upper bound, greatest lower bound) for the subset f (B) of A′. (d) If every subset of (A, ≤) has a LUB (GLB), then every subset of (A′, ≤′) has a LUB (GLB). ■ Example 11 Show that the posets (A, ≤) and (A′, ≤′), whose Hasse diagrams are shown in Figures 6.26(a) and (b), respectively, are not isomorphic. Solution The two posets are not isomorphic because (A, ≤) has a greatest element a, while (A′, ≤′) does not have a greatest element. We could also argue that they are not isomorphic because (A, ≤) does not have a least element, while (A′, ≤′) does have ◆ a least element. Chapter 6 Order Relations and Structures 232 6.2 Exercises In Exercises 1 through 8, determine all maximal and minimal elements of the poset. 1. 3 2 5 f 4 e 1 Figure 6.27 In Exercises 9 through 12, determine the greatest and least elements, if they exist, of the poset. 9. Figure 6.31 10. Figure 6.32 11. A = {2, 4, 6, 8, 12, 18, 24, 36, 72} with the partial order of divisibility. 6. A = {x | x is a real number and 0 ≤ x < 1} with the usual partial order ≤. 8. Figure 6.28 In Exercises 17 and 18, determine if the statements are equivalent. Justify your conclusion. 47. 12. If a ∈ A is a maximal element, then there is no c ∈ A such that a < c. 7 6 3 5 8 2 c 1 b f Figure 6.30 a Figure 6.29 6. A = {x | x is a real number and 0 < x < 1} with the usual partial order ≤. 7. A = {x | x is a real number and 0 < x ≤ 1} with the usual partial order ≤. 8. A = {2, 3, 4, 6, 8, 24, 48} with the partial order of divisibility. In Exercises 9 through 16, determine the greatest and least elements, if they exist, of the poset. 10 f e d (a) If a ∈ A is a minimal element, then there is no c ∈ A such that c < a. (b) If a ∈ A is a minimal element, then for all b ∈ A, a ≤ b. 18. (a) If a ∈ A is a minimal element, then there is no c ∈ A such that c ≤ a. (b) If a ∈ A is a minimal element, then for all b ∈ A, a ≤ b. 5. A = R with the usual partial order ≤. 13. A = {x | x is a real number and 0 < x < 1} with the usual partial order ≤. 14. A = {x | x is a real number and 0 ≤ x ≤ 1} with the usual partial order ≤. 15. A = {2, 4, 6, 8, 12, 18, 24, 36, 72} with the partial order of divisibility. 16. A = {2, 3, 4, 6, 12, 18, 24, 36} with the partial order of divisibility. In Exercises 19 through 26, find, if they exist, (a) all upper bounds of B; (b) all lower bounds of B; (c) the least upper bound of B; (d) the greatest lower bound of B. 19. Determine if the given statement is true or false. Explain your reasoning. (a) A nonempty finite poset has a maximal element. (b) A nonempty finite poset has a greatest element. ... 20. Prove that if (A, ≤) has a least element, then (A, ≤) has a unique least element. 22. Prove that if (A, ≤) has a greatest element, then A has a unique greatest element. 23. Find, if they exist, (a) all minimal elements of A; (b) all maximal elements. 6.33 23 1 Figure 6.34 B = {a, b, c} ... 6.35 Figure 6.36. A = R and B = [1, 2]. 6.3 Lattices A lattice is a poset (L, ≤) in which every subset {a, b} consisting of two elements has a least upper bound and a greatest lower bound. We denote LUB({a, b}) by a ∨ b and call it the join of a and b. Similarly, we denote GLB({a, b}) by a ∧ b and call it the meet of a and b. Lattice structures often appear in computing and mathematical applications. Observe that a lattice is a mathematical structure as described in Section 1.6, with two binary operations, join and meet. Example 1 Let S be a set and let L = P(S). As we have seen, ⊆, containment, is a partial order relation on L. Let A and B belong to the poset (L, ⊆). Then A ∨ B is the set A ∪ B. To see this, note that A ⊆ A ∪ B, B ⊆ A ∪ B, and, if A ⊆ C and B ⊆ C, then it follows that A ∪ B ⊆ C. Similarly, we can show that the element A ∧ B in (L, ⊆) is the set A ∩ B. Thus, L is a lattice. ● Example 2 Consider the poset (Z+, ≤), where for a and b in Z+, a ≤ b if and only if a | b. Then L is a lattice in which the join and meet of a and b are their least common multiple and greatest common divisor, respectively (see Section 1.4). That is, a ∨ b = LCM(a, b) and a ∧ b = GCD(a, b). ● Example 3 ● Let n be a positive integer and let Dn be the set of all positive divisors of n. Then Dn is a lattice under the relation of divisibility as considered in Example 2. Thus, if n = 20, we have D20 = {1, 2, 4, 5, 10, 20}. The Hasse diagram of D20 is shown in Figure 6.39(a). If n = 30, we have D30 = {1, 2, 3, 5, 6, 10, 15, 30}. The Hasse diagram of D30 is shown in Figure 6.39(b). 234 Chapter 6 Order Relations and Structures 20 30 4 6 10 2 5 15 10 2 5 3 1 1 (a) (b) Figure 6.39 Example 4 Which of the Hasse diagrams in Figure 6.40 represent lattices? Solution Hasse diagrams (a), (b), (d), and (e) represent lattices. Diagram (c) does not represent a lattice because f ∨ g does not exist. Diagram (f) does not represent a lattice because neither d ∧ e nor b ∨ c exist. Diagram (g) does not represent a lattice because c ∧ d does not exist. ● f g e d c b e c b h e d b c e d c b e f d c b d c b a a a a a a a (a) (b) (c) (d) (e) (f) (g) Figure 6.40 Example 5 We have already observed in Example 4 of Section 6.1 that the set of all equivalence relations on a set A is a poset under the partial order of set containment. We can now conclude that this poset is a lattice where the meet of the equivalence relations R and S is their intersection R ∩ S and their join is (R ∪ S)∞, the transitive closure of their union (see Section 4.8). ● Let (L, ≤) be a poset and let (L, ≥) be the dual poset. If (L, ≤) is a lattice, we can show that (L, ≥) is also a lattice. In fact, for any a and b in L, the least upper bound of a and b in (L, ≤) is equal to the greatest lower bound of a and b in (L, ≥). Similarly, the greatest lower bound of a and b in (L, ≤) is equal to the least upper bound of a and b in (L, ≥). 6.3 Lattices 235 upper bound of a and b in (L, ≤), is equal to the least upper bound of a and b in (L, ≥). Thus, if L is a finite set, this property can easily be seen by examining the Hasse diagrams of the poset and its dual. Example 6 Let S be a set and L = P(S). Then (L, ⊆) is a lattice, and its dual lattice is (L, ⊇), where ⊆ is "contained in" and ⊇ is "contains." The discussion preceding this example then shows that in the poset (L, ⊇) the join A ∨ B is the set A ∩ B, and the meet A ∧ B is the set A ∪ B. ● THEOREM 1 If (L1, ≤) and (L2, ≤) are lattices, then (L, ≤) is a lattice, where L = L1 × L2, and the partial order ≤ of L is the product partial order. Proof We denote the join and meet in L1 by ∨1 and ∧1 , respectively, and the join and meet in L2 by ∨2 and ∧2, respectively. We already know from Theorem 1 of Section 6.1 that L is a poset. We now need to show that if (a1, b1) and (a2 , b2) ∈ L, then (a1 , b1) ∨ (a2 , b2) and (a1 , b1) ∧ (a2 , b2) exist in L. We leave it as an exercise to verify that (a1, b1) ∨ (a2, b2) = (a1 ∨1 a2 , b1 ∨2 b2) (a1, b1) ∧ (a2, b2) = (a1 ∧1 a2 , b1 ∧2 b2). Thus L is a lattice. Example 7 Let L1 and L2 be the lattices shown in Figures 6.41(a) and (b), respectively. Then L = L1 × L2 is the lattice shown in Figure 6.41(c). ● (I1, I2) (I1, I2) (a, b) (a, b) (I1, b) (I1, 01) (a, I2) (01, 02) (a, 02) (01, 02) (01, I2) (01, b) (I1, 02) (01, 02) (a) (b) (c) Figure 6.41 Let (L, ≤) be a lattice. A nonempty subset S of L is called a sublattice of L if a ∨ b ∈ S and a ∧ b ∈ S whenever a ∈ S and b ∈ S. Example 8 The lattice Dn of all positive divisors of n (see Example 3) is a sublattice of the lattice Z+ under the relation of divisibility (see Example 2). ● Example 9 Consider the lattice L shown in Figure 6.42(a). The partially ordered subset Sb shown in Figure 6.42(b) is not a sublattice of L since a ∧ b ∉ Sb. However, the partially ordered subset Sc in Figure 6.42(c) is a lattice and it can be readily seen. ...
Thus, as stated, Sc is a lattice with respect to the partial order restricted from L... Isomorphic Lattices If f : L1 → L2 is an isomorphism from the poset (L1, ≤1) to the poset (L2, ≤2), then Theorem 4 of Section 6.2 tells us that L1 is a lattice if and only if L2 is a lattice. In fact, if a and b are elements of L1, then f (a ∧ b) = f (a) ∧ f (b) and f (a ∨ b) = f (a) ∨ f (b). If two lattices are isomorphic, as posets, we say they are isomorphic lattices. Example 10 Let L be the lattice D6, and let L′ be the lattice P(S) under the relation of containment, where S = {a, b}. These posets were discussed in Example 16 of Section 6.1, where they were shown to be isomorphic. Thus, since both are lattices, they are isomorphic lattices. ● Properties of Lattices Before proving a number of the properties of lattices, we recall the meaning of a ∨ b and a ∧ b. 1. a ≤ a ∨ b and b ≤ a ∨ b; a ∨ b is an upper bound of a and b. 2. If a ≤ c and b ≤ c, then a ∨ b ≤ c; a ∨ b is the least upper bound of a and b. 1′. a ∧ b ≤ a and a ∧ b ≤ b; a ∧ b is a lower bound of a and b. 2′. If c ≤ a and c ≤ b, then c ≤ a ∧ b; a ∧ b is the greatest lower bound of a and b. THEOREM 2 Let L be a lattice. Then for every a and b in L, (a) a ∨ b = b if and only if a ≤ b. (b) a ∧ b = a if and only if a ≤ b. (c) a ∧ b = a if and only if a ∨ b = b. Proof (a) Suppose that a ∨ b = b. Since a ≤ a ∨ b = b, we have a ≤ b. Conversely, if a ≤ b, then, since b ≤ b, b is an upper bound of a and b; so by definition of least upper bound we have a ∨ b ≤ b. Since a ∨ b is an upper bound, b ≤ a ∨ b, so a ∨ b = b. (b) The proof is analogous to the proof of part (a), and we leave it as an exercise for the reader. (c) The proof follows from parts (a) and (b). ■ Example 11 Let L be a linearly ordered set. If a and b ∈ L, then either a ≤ b or b ≤ a. Thus, by Theorem 2, ... It follows from this that L is a lattice, since every pair of elements has a least upper bound and a greatest lower bound. ● THEOREM 3 Let L be a lattice. Then 1. Idempotent Properties (a) a ∨ a = a (b) a ∧ a = a 2. Commutative Properties (a) a ∨ b = b ∨ a (b) a ∧ b = b ∧ a 3. Associative Properties (a) a ∨ (b ∨ c) = (a ∨ b) ∨ c (b) a ∧ (b ∧ c) = (a ∧ b) ∧ c 4. Absorption Properties (a) a ∨ (a ∧ b) = a (b) a ∧ (a ∨ b) = a 238 Chapter 6 Order Relations and Structures Proof 1. The statements follow from the definition of LUB and GLB. 2. The definition of LUB and GLB treat a and b symmetrically, so the results follow. 3. (a) From the definition of LUB, we have a ≤ a ∨ (b ∨ c) and b ∨ c ≤ a ∨ (b ∨ c). Moreover, b ≤ b ∨ c and c ≤ b ∨ c, so, by transitivity, b ≤ a ∨ (b ∨ c) and c ≤ a ∨ (b ∨ c). Thus a ∨ (b ∨ c) is an upper bound of a and b, so by definition of least upper bound we have a ∨ b ≤ a ∨ (b ∨ c). Since c ≤ a ∨ (b ∨ c), we obtain (a ∨ b) ∨ c ≤ a ∨ (b ∨ c). A similar argument shows that a ∨ (b ∨ c) ≤ (a ∨ b) ∨ c. By antisymmetry, property 3(a) follows. (b) The proof is analogous to the proof of part (a) and we omit it. 4. (a) Since a ∧ b ≤ a and a ≤ a, we see that a is an upper bound of a ∧ b and a; so a ∨ (a ∧ b) ≤ a. On the other hand, by the definition of LUB, we have a ≤ a ∨ (a ∧ b), so a ∨ (a ∧ b) = a. (b) The proof is analogous to the proof of part (a) and we omit it. ■ It follows from property 3 that we can write a ∨ (b ∨ c) and (a ∨ b) ∨ c merely as a ∨ b ∨ c, and similarly for a ∧ b ∧ c. Moreover, we can write LUB({a1 , a2 , . . . , an }) as a1 ∨ a2 ∨ · · · ∨ an GLB({a1 , a2 , . . . , an }) as a1 ∧ a2 ∧ · · · ∧ an since we can show by induction that these joins and meets are independent of the grouping of the terms. THEOREM 4 Let L be a lattice. Then, for every a, b, and c in L, 1. If a ≤ b, then (a) a ∨ c ≤ b ∨ c. (b) a ∧ c ≤ b ∧ c. 2. a ≤ c and b ≤ c if and only if a ∨ b ≤ c. 3. c ≤ a and c ≤ b if and only if c ≤ a ∧ b. 4. If a ≤ b and c ≤ d, then (a) a ∨ c ≤ b ∨ d. (b) a ∧ c ≤ b ∧ d. Proof The proof is left as an exercise. ■ Special Types of Lattices A lattice L is said to be bounded if it has a greatest element I and a least element 0 (see Section 6.2). Example 13 The lattice Z+ under the partial order of divisibility, as defined in Example 2, is not a bounded lattice since it has a least element, the number 1, but no greatest element. ● 6.3 Lattices 239 Example 14 The lattice Z under the partial order ≤ is not bounded since it has neither a greatest nor a least element. ● Example 15 The lattice P(S) of all subsets of a set S, as defined in Example 1, is bounded. Its ◆ greatest element is S and its least element is ∅. If L is a bounded lattice, then for all a ∈ L, 0 ≤ a ≤ I a ∨ 0 = a, a ∧ 0 = 0 a ∨ I = I, a ∧ I = a. THEOREM 5 Let L = {a1 , a2 , . . . , an } be a finite lattice. Then L is bounded. Proof The greatest element of L is a1 ∨ a2 ∨ · · · ∨ an , and its least element is a1 ∧ a2 ∧ · · · ∧ an. ■ Note that the proof of Theorem 5 is a constructive proof. We show that L is bounded by constructing the greatest and the least elements. A lattice L is called distributive if for any elements a, b, and c in L we have the following distributive properties: 1. a ∧ (b ∨ c) = (a ∧ b) ∨ (a ∧ c) 2. a ∨ (b ∧ c) = (a ∨ b) ∧ (a ∨ c) If L is not distributive, we say that L is nondistributive. We leave it as an exercise to show that the distributive property holds when any two of the elements a, b, or c are equal or when any one of the elements is 0 or I. This observation reduces the number of cases that must be checked in verifying that a distributive property holds. However, verification of a distributive property is generally a tedious task. Example 16 For a set S, the lattice P(S) is distributive, since union and intersection (the join and meet, respectively) each satisfy the distributive property shown in Section 1.2. ● Example 17 The lattice shown in Figure 6.43 is distributive, as can be seen by verifying the distributive properties for all ordered triples chosen from the elements a, b, c, and d. ● Example 18 Show that the lattices pictured in Figure 6.44 are nondistributive. Solution (a) We have I I a while b ∧ 0 (a) Figure 6.44 a ∧ (b ∨ c) = a ∧ I = a (a ∧ b) ∨ (a ∧ c) = b ∨ 0 = b. (b) Observe that 0 (b) a ∧ (b ∨ c) = a ∧ I = a (a ∧ b) ∨ (a ∧ c) = 0 ∨ 0 = 0. ◆ 240 Chapter 6 Order Relations and Structures The nondistributive lattices discussed in Example 18 are useful for showing that a given lattice is nondistributive, as the following theorem, whose proof we omit, asserts. THEOREM 6 A lattice L is nondistributive if and only if it contains a sublattice that is isomorphic to one of the two lattices of Example 18. ■ Theorem 6 can be used quite efficiently by inspecting the Hasse diagram of L. Let L be a bounded lattice with greatest element I and least element 0, and let a ∈ L. An element a′ ∈ L is called a complement of a if a ∨ a′ = I and a ∧ a′ = 0. Observe that 0′ = I and I′ = 0. Example 19 The lattice L = P(S) is such that every element has a complement, since if A ∈ L, then its set complement Ā has the properties A ∪ Ā = S and A ∩ Ā = ∅. That is, the set complement is also the complement in the lattice L. ● Example 20 The lattices in Figure 6.44 each have the property that every element has a complement. The element c in both cases has two complements, a and b. ● Example 21 Consider the lattices D20 and D30 discussed in Example 3 and shown in Figure 6.39. Observe that every element in D30 has a complement. For example, if a = 5, then a′ = 6. However, the elements 2 and 10 in D20 have no complements. ● Examples 20 and 21 show that an element a in a lattice need not have a complement, and it may have more than one complement. However, for a bounded distributive lattice, the situation is more restrictive, as shown by the following theorem. THEOREM 7 Let L be a bounded distributive lattice. If a complement exists, it is unique. Proof Let a′ and a″ be complements of the element a ∈ L. Then a ∨ a′ = I, a ∨ a″ = I, a ∧ a′ = 0, a ∧ a″ = 0. Using the distributive laws, we obtain a′ = a′ ∨ 0 = a′ ∨ (a ∧ a″) = (a′ ∨ a) ∧ (a′ ∨ a″) = I ∧ (a′ ∨ a″) = a′ ∨ a″. Also, a″ = a″ ∨ 0 = a″ ∨ (a ∧ a′) = (a″ ∨ a) ∧ (a″ ∨ a′) = I ∧ (a′ ∨ a″) = a′ ∨ a″. Hence a′ = a″. ■ 6.3 Lattices 241 The proof of Theorem 7 is a direct proof, but it is not obvious how the representations of a′ and a″ were chosen. There is some trial and error involved in this sort of proof, but we expect to use the hypothesis that L is bounded and that L is distributive. An alternative proof is outlined in Exercise 36. A lattice L is called complemented if it is bounded and if every element in L has a complement. Example 22 The lattice L = P(S) is complemented. Observe that in this case each element of L has a unique complement, which can be seen directly or is implied by Theorem 7. ● Example 23 The lattices discussed in Example 20 and shown in Figure 6.44 are complemented. In this case, the complements are not unique. ● 6.3 Exercises In Exercises 1 through 6 (Figures 6.45 through 6.50), determine whether the Hasse diagram represents a lattice. 1. 2e d c b a 3. 4 ... 7. Is the poset A = {2, 3, 6, 12, 24, 36, 72} under the relation of divisibility a lattice? 8. Let A = {1, 2, 3, 6}. Is (A, R) a lattice? If it is, find the LUB and GLB. 10. Amplify the explanations in the solution of Example 4 by explaining why the specified object does not exist. 11. If L1 and L2 are the lattices shown in Figure 6.51, draw the Hasse diagram of L1 × L2 with the product partial order. ... 12. Complete the proof of Theorem 1 by verifying that (a1, b1) ∨ (a2, b2) = (a1 ∨1 a2 , b1 ∨2 b2) and (a1, b1) ∧ (a2, b2) = (a1 ∧1 a2 , b1 ∧2 b2). 13. Let L = P(S) be the lattice of all subsets of a set S under the relation of containment. Let T be a subset of S. Show that P(T) is a sublattice of L. 15. Show that a subset of a linearly ordered poset is a sublattice. 16. Find all sublattices of D24 that contain at least five elements. 17. Give the Hasse diagrams of all nonisomorphic lattices that have one, two, three, four, or five elements. 18. Show that if a bounded lattice has two or more elements, then 0 ≠ I. 19. Prove Theorem 2(b). 20. Show that the lattice Dn is distributive for any n. 21. (a) In Example 17 (Figure 6.43), how many ordered triples must be checked to see if the lattice is distributive? (b) In Example 18 (Figure 6.44), what is the maximum number of ordered triples that would need to be checked to show that the lattice is not distributive? 22. Show that a sublattice of a distributive lattice is distributive. 23. Show that if L1 and L2 are distributive lattices, then L = L1 × L2 is also distributive, where the order of L is the product of the orders in L1 and L2. 24. Prove that if a and b are elements in a bounded, distributive lattice and if a has a complement a′, then a ∨ (a′ ∧ b) = a ∨ b a ∧ (a′ ∨ b) = a ∧ b. 30. e 31. g d e f c b d a Figure 6.56 32. Prove Theorem 4. 33. Let L be a bounded lattice with at least two elements. Show that no element of L is its own complement. 34. Consider the complemented lattice shown in Figure 6.57. Give the complements of each element. 25. Let L be a distributive lattice. Show that if there exists an a with a ∧ x = a ∧ y and a ∨ x = a ∨ y, then x = y. 26. A lattice is said to be modular if, for all a, b, c, a ≤ c implies that a ∨ (b ∧ c) = (a ∨ b) ∧ c. (a) Show that a distributive lattice is modular. (b) Show that the lattice shown in Figure 6.52 is a nondistributive lattice that is modular. e d b a c 0 Figure 6.57 e a b c d I 0 Figure 6.52 27. Find the complement of each element in D42. 28. Prove that if a lattice has a complement, then it is bounded. 31. Show that P(S) is a complemented lattice. ...
36. Prove that if L is a bounded lattice and f : L → L′ is an isomorphism, ... 6.4 Finite Boolean Algebras In this section we discuss a certain type of lattice that has a great many applications in computer science. We have seen in Example 6 of Section 6.3 that if S is a set, L = P(S), and ⊆ is the usual relation of containment, then the poset (L, ⊆) is a lattice. These lattices have many properties that are not shared by lattices in general. For this reason they are easier to work with, and they play a more important role in various applications. We will restrict our attention to the lattices (P(S), ⊆), where S is a finite set, and we begin by finding all essentially different examples. THEOREM 1 If S1 = {x1 , x2 , . . . , xn } and S2 = {y1 , y2 , . . . , yn } are any two finite sets with n elements, then the lattices (P(S1), ⊆) and (P(S2), ⊆) are isomorphic. Consequently, the Hasse diagrams of these lattices may be drawn identically. Proof Arrange the sets as shown in Figure 6.58 so that each element of S1 is directly over the correspondingly numbered element in S2. For each subset A of S1, let f (A) be the subset of S2 consisting of all elements that correspond to the elements of A. Figure 6.59 shows a typical subset A of S1 and the corresponding subset f (A) of S2. It is easily seen that the function f is a one-to-one correspondence from subsets of S1 to subsets of S2. Equally clear is the fact that if A and B are any subsets of S1, then A ⊆ B if and only if f (A) ⊆ f (B). We omit the details. Thus the lattices (P(S1), ⊆) and (P(S2), ⊆) are isomorphic. ■ S1 : x1 x2 x3 x4 · · · xn S2 : y1 y2 y3 y4 · · · yn xn yn Figure 6.58 Figure 6.59 The essential point of this theorem is that the lattice (P(S), ⊆) is completely determined as a poset by the number |S| and does not depend in any way on the nature of the elements in S. Example 1 Figures 6.60(a) and (b) show Hasse diagrams for the lattices (P(S), ⊆) and (P(T), ⊆), respectively, where S = {a, b, c} and T = {2, 3, 5}. It is clear from this figure that the two lattices are isomorphic. In fact, we see that one possible isomorphism f : S → T is given by f ({a}) = {2}, f ({a, b}) = {2, 3}, f ({b}) = {3}, f ({b, c}) = {3, 5}, f ({c}) = {5}, f ({a, c}) = {2, 5}, f (S) = T, f (∅) = ∅. ● Thus, for each n = 0, 1, 2, . . . , there is only one type of lattice having the form (P(S), ⊆). This lattice depends only on n, not on S, and it has 2ⁿ elements, as was shown in Example 2 of Section 3.1. Recall from Section 1.3 that if a set S has n elements, then all subsets of S can be represented by sequences of 0's and 1's of length n. We can therefore label the Hasse diagram of a lattice (P(S), ⊆) by such sequences. In doing so, we free the diagram from dependence on a particular set S and emphasize the fact that it depends only on n. ● Example 2 Figure 6.60(c) shows how the diagrams that appear in Figure 6.60(a) and (b) can be labeled by sequences of 0's and 1's. This labeling serves equally well to describe the lattice of Figure 6.60(a) or (b), or for that matter the lattice (P(S), ⊆) that arises from any set S having three elements. ● If the Hasse diagram of the lattice corresponding to a set with n elements is labeled by sequences of 0's and 1's of length n, as described previously, then the resulting lattice is named Bn . The properties of the partial order on Bn can be described directly as follows. If x = a1 a2 · · · an and y = b1 b2 · · · bn are any two elements of Bn, then 1. x ≤ y if and only if ak ≤ bk for k = 1, 2, . . . , n, where ak and bk are 0 or 1. 2. x ∧ y = c1 c2 · · · cn , where ck = min{ak , bk }. 3. x ∨ y = d1 d2 · · · dn , where dk = max{ak , bk }. 4. x has a complement x′ = z1 z2 · · · zn , where zk = 1 if xk = 0, and zk = 0 if xk = 1. The truth of these statements can be seen by noting that (Bn, ≤) is isomorphic with (P(S), ⊆), so each x and y in Bn correspond to subsets A and B of S. Then x ≤ y, x ∧ y, x ∨ y, and x′ correspond to A ⊆ B, A ∩ B, A ∪ B, and Ā (set complement), respectively (verify). Figure 6.61 shows the Hasse diagrams of the lattices Bn for n = 0, 1, 2, 3. We have seen that each lattice (P(S), ⊆) is isomorphic with Bn, where n = |S|. Other lattices may also be isomorphic with one of the Bn and thus possess all the special properties that the Bn possess. Example 3 In Example 17 of Section 6.1, we considered the lattice D6 consisting of all positive integer divisors of 6 under the partial order of divisibility. The Hasse diagram of D6 is shown in Figure 6.13(a). We now see that D6 is isomorphic with B2. In fact, f : D6 → B2 is an isomorphism, where f (1) = 00, f (2) = 10, f (3) = 01, f (6) = 11. ● We are therefore led to make the following definition. A finite lattice is called a Boolean algebra if it is isomorphic with Bn for some nonnegative integer n. 6.4 Finite Boolean Algebras 245 111 110 101 011 100 010 001 000 n = 3 11 10 01 00 n = 2 1 0 n = 1 0 n = 0 Figure 6.61 Thus each Bn is a Boolean algebra and so is each lattice (P(S), ⊆), where S is a finite set. Example 3 shows that D6 is also a Boolean algebra. We will work only with finite posets in this section. For the curious, however, we note that there are infinite posets that share all the relevant properties of the lattices (P(S), ⊆) (for infinite sets S, of course), but that are not isomorphic with one of these lattices. This necessitates the restriction of our definition of Boolean algebra to the finite case, which is sufficient for the applications that we present. Example 4 Consider the lattices D20 and D30 of all positive integer divisors of 20 and 30, respectively, under the partial order of divisibility. These posets were introduced in Example 3 of Section 6.3, and their Hasse diagrams were shown in Figure 6.39. Since D20 has six elements and 6 ≠ 2ⁿ for any integer n ≥ 0, we conclude that D20 is not a Boolean algebra. The poset D30 has eight elements, and since 8 = 2³, it could be a Boolean algebra. By comparing Figure 6.39(b) and Figure 6.61, we see that D30 is isomorphic with B3 . In fact, we see that the one-to-one correspondence f : D30 → B3 defined by f (1) = 000, f (2) = 100, f (3) = 010, f (5) = 001, f (6) = 110, f (10) = 101, f (15) = 011, f (30) = 111 is an isomorphism. Thus D30 is a Boolean algebra. ● If a finite lattice L does not contain 2ⁿ elements for some nonnegative integer n, we know that L cannot be a Boolean algebra. If |L| = 2ⁿ, then L may or may not be a Boolean algebra. If L is relatively small, we may be able to compare its Hasse diagram with the Hasse diagram of Bn. In this way we saw in Example 4 that D30 is a Boolean algebra. However, this technique may not be practical if L is large. In that case, we may be able to show that L is a Boolean algebra by directly constructing an isomorphism with some Bn or, equivalently, with (P(S), ⊆) for some finite set S. Suppose, for example, that we want to know whether a lattice Dn is a Boolean algebra, and we want a method that works no matter how large n is. The following theorem gives a partial answer. 246 Chapter 6 Order Relations and Structures THEOREM 2 Let n = p1 p2 · · · pk , where the pi are distinct primes. Then Dn is a Boolean algebra. Proof Let S = {p1 , p2 , . . . , pk }. If T ⊆ S and aT is the product of the primes in T , then aT | n. Any divisor of n must be of the form aT for some subset T of S (where we let a∅ = 1). The reader may verify that if V and T are subsets of S, V ⊆ T if and only if aV | aT . Also, it follows from the proof of Theorem 6 of Section 1.4 that aV∩T = aV ∧ aT = GCD(aV , aT) and aV∪T = aV ∨ aT = LCM(aV , aT). Thus the function f : P(S) → Dn given by f (T) = aT is an isomorphism from P(S) to Dn. Since P(S) is a Boolean algebra, so is Dn. ■ Example 5 Since 210 = 2 · 3 · 5 · 7, 66 = 2 · 3 · 11, and 646 = 2 · 17 · 19, we see from Theorem 2 that D210 , D66 , and D646 are all Boolean algebras. ● In other cases of large lattices L, we may be able to show that L is not a Boolean algebra by showing that the partial order of L does not have the necessary properties. A Boolean algebra is a lattice that is isomorphic with some Bn. If L is any Boolean algebra, then each element x of L has a complement x′. According to Example 16, Section 6.3, L must also be distributive. The principle of correspondence (see Section 6.1) then tells us that the following rule holds. THEOREM 3 Substitution Rule for Boolean Algebras Any formula involving ∪ or ∩ that holds for arbitrary subsets of a set S will continue to hold for arbitrary elements of a Boolean algebra L if ∧ is substituted for ∩ and ∨ for ∪. ■ Example 6 If L is any Boolean algebra and x, y, and z are in L, then the following three properties hold. 1. (x′)′ = x Involution Property 2. (x ∧ y)′ = x′ ∨ y′ De Morgan's Laws 3. (x ∨ y)′ = x′ ∧ y′ This is true by the substitution rule for Boolean algebras, since we know that the corresponding formulas 1′. (Ā)‾ = A 2′. (A ∩ B)‾ = Ā ∪ B̄ 3′. (A ∪ B)‾ = Ā ∩ B̄ hold for arbitrary subsets A and B of a set S. ● In a similar way, we can list other properties that must hold in any Boolean algebra by the substitution rule. Next we summarize all the basic properties of a Boolean algebra (L, ≤) and, next to each one, we list the corresponding property for subsets of a set S. We suppose that x, y, and z are arbitrary elements in L, and A, B, and C are arbitrary subsets of S. Also, we denote the greatest and least elements of L by I and 0, respectively. 6.4 Finite Boolean Algebras 247 1. x ≤ y if and only if x ∨ y = y. 1′. A ⊆ B if and only if A ∪ B = B. 2. x ≤ y if and only if x ∧ y = x. 2′. A ⊆ B if and only if A ∩ B = A. 3. (a) x ∨ x = x. (b) x ∧ x = x. 3′. (a) A ∪ A = A. (b) A ∩ A = A. 4. (a) x ∨ y = y ∨ x. (b) x ∧ y = y ∧ x. 4′. (a) A ∪ B = B ∪ A. (b) A ∩ B = B ∩ A. 5. (a) x ∨ (y ∨ z) = (x ∨ y) ∨ z. (b) x ∧ (y ∧ z) = (x ∧ y) ∧ z. 5′. (a) A ∪ (B ∪ C) = (A ∪ B) ∪ C. (b) A ∩ (B ∩ C) = (A ∩ B) ∩ C. 6. (a) x ∨ (x ∧ y) = x. (b) x ∧ (x ∨ y) = x. 6′. (a) A ∪ (A ∩ B) = A. (b) A ∩ (A ∪ B) = A. 7. 0 ≤ x ≤ I for all x in L. 7′. ∅ ⊆ A ⊆ S for all A in P(S). 8. (a) x ∨ 0 = x. (b) x ∧ 0 = 0. 8′. (a) A ∪ ∅ = A. (b) A ∩ ∅ = ∅. 9. (a) x ∨ I = I. (b) x ∧ I = x. 9′. (a) A ∪ S = S. (b) A ∩ S = A. 10. (a) x ∧ (y ∨ z) = (x ∧ y) ∨ (x ∧ z). (b) x ∨ (y ∧ z) = (x ∨ y) ∧ (x ∨ z). 10′. (a) A ∩ (B ∪ C) = (A ∩ B) ∪ (A ∩ C). (b) A ∪ (B ∩ C) = (A ∪ B) ∩ (A ∪ C). 11. Every element x has a unique complement x′ satisfying (a) x ∨ x′ = I. (b) x ∧ x′ = 0. 11′. Every element A has a unique complement Ā satisfying (a) A ∪ Ā = S. (b) A ∩ Ā = ∅. 12. (a) 0′ = I. (b) I′ = 0. 12′. (a) ∅‾ = S. (b) S̄ = ∅. 13. (x′)′ = x. 13′. (Ā)‾ = A. 14. (a) (x ∧ y)′ = x′ ∨ y′. (b) (x ∨ y)′ = x′ ∧ y′. 14′. (a) (A ∩ B)‾ = Ā ∪ B̄. (b) (A ∪ B)‾ = Ā ∩ B̄. Thus we may be able to show that a lattice L is not a Boolean algebra by showing that it does not possess one or more of these properties. Example 7 Show that the lattice whose Hasse diagram is shown in Figure 6.62 is not a Boolean algebra. Solution Elements a and e are both complements of c; that is, they both satisfy properties 11(a) and 11(b) with respect to the element c. But property 11 says that such an element is unique in any Boolean algebra. Thus the given lattice cannot be a ◆ Boolean algebra. Example 8 Show that if n is a positive integer and p² | n, where p is a prime number, then Dn is not a Boolean algebra. Solution Suppose that p² | n so that n = p²q for some positive integer q. Since p is also a divisor of n, p is an element of Dn. Thus, by the remarks given previously, if Dn is a Boolean algebra, then p must have a complement p′. Then GCD(p, p′) = 1 and LCM(p, p′) = n. By Theorem 6 of Section 1.4, pp′ = n, so p′ = n/p = pq. This shows that GCD(p, pq) = 1, which is impossible, since p and pq have p as a common divisor. Hence Dn cannot be a Boolean algebra. ● 248 Chapter 6 Order Relations and Structures If we combine Example 8 and Theorem 2, we see that Dn is a Boolean algebra if and only if n is the product of distinct primes, that is, if and only if no prime divides n more than once. If n = 40, then n = 2³ · 5, so 2 divides 40 more than once. Hence D40 is not a Boolean algebra. ● Example 9 If n = 75, then n = 3 · 5² · 5, so 5 divides 75 more than once. Hence D75 is not a Boolean algebra. ● Example 9 If n = 30, we know that D30 is a Boolean algebra. ... 10 If we ... Example 10 If n = 385, then n = 5 · 7 · 11, so every prime divides n only once. Hence D385 is a Boolean algebra. ● Example 9 let us summarize what we have shown about Boolean algebras. We may now say that a finite lattice L is a Boolean algebra if it is isomorphic with Bn for some n. In this case, we may use any of the properties listed previously. Simply, and if Boolean algebras can be used to represent switching circuits, the simplest way to simplify or rewrite expressions involving elements of L. Because of this, most of the work in this section can be done in Bn. ... Boolean polynomial ... Chapter 6 Order Relations and Structures ... 6.4 Exercises 1. Which of the following are Boolean algebras? Justify your answer. (a) D385 ... 6.5 Functions on Boolean Algebras Tables listing the values of a function f for all elements of Bn, such as shown in Figure 6.71(a), are often called truth tables for f. This is because they are analogous to tables that arise in logic (see Section 2.1). Suppose that the xk represent propositions, and f (x1, x2, . . . , xn) represents a compound sentence constructed from the xk's. If we think of the value 0 for a sentence as meaning that the sentence is false, and 1 as meaning that the sentence is true, then tables such as Figure 6.71(a) show us how truth or falsity of f (x1 , x2 , . . . , xn) depends on the truth or falsity of its component sentences xk. Thus such tables are called truth tables for f. A function f : Bn → B is called a Boolean function. In the next section we present an application of Boolean functions... Boolean polynomials... A Boolean polynomial in the n variables x1, x2, . . . , xn is defined recursively as follows: 1. x1 , x2 , . . . , xn are all Boolean polynomials. 2. The symbols 0 and 1 are Boolean polynomials. 3. If p(x1 , x2 , . . . , xn) and q(x1 , x2 , . . . , xn) are two Boolean polynomials, then so are p(x1 , x2 , . . . , xn) ∨ q(x1 , x2 , . . . , xn) and p(x1 , x2 , . . . , xn) ∧ q(x1 , x2 , . . . , xn). 4. If p(x1 , x2 , . . . , xn) is a Boolean polynomial, then so is (p(x1 , x2 , . . . , xn))′. By tradition, (0)′ is denoted 0′, (1)′ is denoted 1′, and (xk)′ is denoted x′k. 5. There are no Boolean polynomials in the variables xk other than those that can be obtained by repeated use of rules 1, 2, 3, and 4. Boolean polynomials are also called Boolean expressions. Example 1 The following are Boolean polynomials in the variables x, y, and z. p1 (x, y, z) = (x ∨ y) ∧ z p2 (x, y, z) = (x ∨ y′) ∨ (y ∧ 1) p3 (x, y, z) = (x ∨ (y′ ∧ z)) ∨ (x ∧ (y ∧ 1)) p4 (x, y, z) = (x ∨ (y ∨ z′)) ∧ ((x′ ∧ z)′ ∧ (y′ ∨ 0)) ● Ordinary polynomials in several variables, such as x²y + z⁴, xy + yz, x²y² − xy², and so on, are generally interpreted as expressions representing algebraic computations with unspecified numbers. As such, they are subject to the usual rules of arithmetic. Thus the polynomials x² + 2x + 1 and (x + 1)(x + 1) are considered equivalent, and so are x(xy + yz)(x + z) and x³y + 2x²yz + xyz², since in each case we can turn one into the other with algebraic manipulation. Similarly, Boolean polynomials may be interpreted as representing Boolean computations with unspecified elements of B, that is, with 0's and 1's. As such, these polynomials are subject to the rules of Boolean arithmetic, that is, to the rules obeyed by ∧, ∨, and ′ in Boolean algebras. As with ordinary polynomials, two Boolean polynomials are considered equivalent if we can turn one into the other with Boolean manipulations. In Section 5.1 we showed how ordinary polynomials could produce functions by substitution. This process works whether the polynomials involve one or several variables. Thus the polynomial xy + yz³ produces a function f : R³ → R by letting f (x, y, z) be the value of xy + yz³ for any vector (x, y, z) of real numbers. For example, f (3, 4, 2) = (3)(4) + (4)(2)³ or 44. In a similar way, Boolean polynomials involving n variables produce functions from Bn to B. These Boolean functions are a natural generalization of those introduced in Section 5.2. 3 3 Example 2 Consider the Boolean polynomial p(x1 , x2 , x3) = (x1 ∧ x2) ∨ (x1 ∨ (x′2 ∧ x3)). Construct the truth table for the Boolean function f : B3 → B determined by this Boolean polynomial. Solution The Boolean function f : B3 → B is described by substituting all the 2³ ordered triples of values from B for x1 , x2 , and x3 . The truth table for the resulting function f is shown in Figure 6.73. ● Boolean polynomials can also be written in a graphical or schematic way. If x and y are variables, then the basic polynomials x ∨ y, x ∧ y, and x′ are shown schematically in Figure 6.74. Each symbol has lines for the variables on the left and a line on the right representing the polynomial as a whole. The symbol for x ∨ y is called an or gate, that for x ∧ y is called an and gate, and the symbol for x′ is called an inverter. The logical names arise because the truth tables showing the functions represented by x ∨ y and x ∧ y are exact analogs of the truth table for the connectives "or" and "and," respectively. Recall that functions from Bn to B can be used to describe the desired behavior of circuits with n 0-or-1 inputs and one 0-or-1 output. In the case of the 252 Chapter 6 Order Relations and Structures x1 x2 x3 f (x1 , x2 , x3) 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1 1 Figure 6.72 x x ∨ y y x y x ∧ y x x′ (b) Figure 6.73 functions corresponding to the Boolean polynomials x ∨ y, x ∧ y, and x′, the desired circuits can be implemented, and the schematic forms of Figure 6.73 are also used to represent these circuits. By repeatedly using these schematic forms for ∨, ∧, and ′, we can make a schematic form to represent any Boolean polynomial. For the reasons given previously, such diagrams are called logic diagrams for the polynomial. Example 3 Let p(x, y, z) = (x ∧ y) ∨ (y ∧ z′). Figure 6.74(a) shows the truth table for the corresponding function f : B3 → B. Figure 6.74(b) shows the logic diagram for p. ● x y z f (x, y, z) 0 0 0 0 0 0 1 0 0 1 0 1 0 1 1 0 1 0 0 0 1 0 1 0 1 1 0 1 1 1 1 1 (a) x y z (b) Figure 6.74 Suppose that p is a Boolean polynomial in n variables, and f is the corresponding function from Bn to B. We know that f may be viewed as a description of the behavior of a circuit having n inputs and one output. In the same way, the logic diagram of p can be viewed as a description of the construction of such a circuit, at least in terms of and gates, or gates, and inverters. Thus if the function f , describing the desired behavior of a circuit, can be produced by a Boolean polynomial p, then the logic diagram for p will give one way to construct a circuit having that behavior. In general, many different polynomials will produce the same function. The design and use of these polynomials is an important consideration in the design of circuits. It is almost impossible to overemphasize the importance of these facts for the study of computer circuitry. Nearly all circuit design is done by software packages, but it is important to understand the basic principles of circuit design outlined in this section and the next. 6.5 Exercises 1. Consider the Boolean polynomial p(x, y, z) = x ∧ (y ∨ z′). If B = {0, 1}, compute the truth table of the function f : B3 → B defined by p. 2. Consider the Boolean polynomial p(x, y, z) = (x ∨ y) ∧ (z ∨ x′). If B = {0, 1}, compute the truth table of the function f : B3 → B defined by p. 3. Consider the Boolean polynomial p(x, y, z) = (x ∧ y′) ∨ (y ∧ (x′ ∨ y)). If B = {0, 1}, compute the truth table of the function f : B3 → B defined by p. 4. Consider the Boolean polynomial p(x, y, z) = (x ∧ y) ∨ (x′ ∧ (y ∧ z′)). If B = {0, 1}, compute the truth table of the function f : B3 → B defined by p. In Exercises 5 through 10, apply the rules of Boolean arithmetic to show that the given Boolean polynomials are equivalent. 5. (x ∨ y) ∧ (x′ ∨ y) ≡ y 6. x ∧ (y ∨ (y′ ∧ (y ∨ y′))) ≡ x 7. (z′ ∨ x) ∧ ((x ∧ y) ∨ z) ∧ (z′ ∨ y) ≡ x ∧ y 8. (x ∧ z) ∨ (y′ ∨ (y ∧ z)) ∨ ((y ∧ z) ∧ x) ≡ (x ∧ z) ∨ y′ 9. (x ∧ y) ∨ (x′ ∧ z) ∨ (y ∧ z) ≡ (x ∧ y) ∨ (x′ ∧ z) 10. In Exercises 11 through 16, construct a logic diagram implementing the function f of (a) Exercise 1; (b) Exercise 2; ... 15. Construct a logic diagram implementing the function f of (a) Exercise 3; (b) Exercise 4. 16. Give the Boolean function described by the logic diagram in Figure 6.75. 17. Give the Boolean function described by the logic diagram in Figure 6.76. 18. Give the Boolean function described by the logic diagram in Figure 6.77. 19. Give the Boolean function described by the logic diagram in Figure 6.77. 21. Use the properties of a Boolean algebra to refine the function in Exercise 18 to use the minimal number of variables and operations. Draw the logic diagram for the new function. 22. Use the properties of a Boolean algebra to refine the function in Exercise 19 to use the minimal number of variables and operations. Draw the logic diagram for the new function. 23. Use the properties of a Boolean algebra to refine the function in Exercise 20 to use the minimal number of variables and operations. Draw the logic diagram for the new function. 6.6 Circuit Design In Section 6.5 we considered functions from Bn to B, where Bn is the Boolean algebra B × B × · · · × B (n factors). We noted that such functions can represent input-output requirements for models of many practical computer circuits. We also pointed out that if the function is given by some Boolean expression, then we can construct a logic diagram for it and thus model the implementation of the function. In this section we show that all functions from Bn to B are given by Boolean expressions, and thus logic diagrams can be constructed for any such function. Our discussion illustrates a method for finding a Boolean expression that produces a given function. If f : Bn → B, we will let S(f) = {b ∈ Bn | f (b) = 1}. We then have the following result. THEOREM 1 Let f, f1 and f2 be three functions from Bn to B. (a) If S(f) = S(f1) ∪ S(f2), then f (b) = f1 (b) ∨ f2 (b) for all b in B. (b) If S(f) = S(f1) ∩ S(f2), then f (b) = f1 (b) ∧ f2 (b) for all b in B. (∨ and ∧ are LUB and GLB, respectively, in B.) Proof (a) Let b ∈ Bn. If b ∈ S(f), then, by the definition of S(f), f (b) = 1. Since S(f) = S(f1) ∪ S(f2), either b ∈ S(f1) or b ∈ S(f2), or both. In any case, f1 (b) ∨ f2 (b) = 1. Now, if b ∉ S(f), then f (b) = 0. Also, we must have b ∉ S(f1) and b ∉ S(f2), so f1 (b) = 0 and f2 (b) = 0. This means that f1 (b) ∨ f2 (b) = 0. Thus, for all b ∈ Bn, f (b) = f1 (b) ∨ f2 (b). (b) This part is proved in a manner completely analogous to that used in part (a). ■ Recall that a function f : Bn → B can be viewed as a function f (x1 , x2 , . . . , xn) of n variables, each of which may assume the values 0 or 1. If E(x1 , x2 , . . . , xn) is a Boolean expression, then the function that it produces is generated by substituting all combinations of 0's and 1's for the xi's in the expression. Example 1 Let f : B2 → B be the function whose truth table is shown in Figure 6.78(a). Then S(f) = {(1, 0), (0, 1)}, and S(f1) = {(1, 0)}, S(f2) = {(0, 1)}. Let f1 and f2 be the functions from B2 to B whose truth tables are shown in Figure 6.78(b) and (c) ... (a) Figure 6.78 It is not hard to show that any function f : Bn → B, f = f1 ∨ f2 ... Table 6.1 shows the correspondence ... may be expressed. ... ● Thus f : B2 → B, f (x, y) = x ∧ y′ ... ● If S(f) contains only one element, the function f is easily described by a Boolean expression.

between functions of two variables that are 1 at just one element and the Boolean expression that produces these functions. TABLE 6.1 Example 2 f( ) Expression Producing f {(0, 0)} x ∧ y {(0, 1)} x ∧ y {(1, 0)} x ∧ y {(1, 1)} x∧y Let f : B2 → B be the function whose truth table is shown in Figure 6.79(a). This function is equal to 1 only at the element (0, 1) of B2 ; that is, S(f) = {(0, 1)}. Thus f(x, y) = 1 only when x = 0 and y = 1. This is also true for the expression E(x, y) = x ∧ y, so f is produced by this expression. The function f : B3 → B whose truth table is shown in Figure 6.79(b) has S(f) = {(0, 1, 1)}; that is, f equals 1 only when x = 0, y = 1, and z = 1. This is also true for the Boolean expression x ∧ y ∧ z, which must therefore produce f. ◆ If b ∈ Bn, then b is a sequence (c1, c2, ..., cn) of length n, where each ck is 0 or 1. Let E b be the Boolean expression x̄1 ∧ x̄2 ∧ · · · ∧ x̄n, where x̄k = xk when ck = 1 and x̄k = x̄k when ck = 0. Such an expression is called a minterm. Example 2 illustrates the fact that any function f : Bn → B for which S(f) is a single element of Bn is produced by a minterm expression. In fact, if S(f) = {b}, it is easily seen that the minterm expression E b produces f. We leave the proof of this fact to the reader.

Figure 6.79 THEOREM 2 Any function f : Bn → B is produced by a Boolean expression. Proof Let S(f) = {b1, b2, ..., bk}, and for each i, let fi : Bn → B be the function defined by fi(bi) = 1 and fi(b) = 0 if b ≠ bi. Then S(fi) = {bi}, so S(f) = S(f1) ∪ · · · ∪ S(fn) and by Theorem 1, f = f1 ∨ f2 ∨ · · · ∨ fn. By the preceding discussion, each fi is produced by the minterm E bi. Thus f is produced by the Boolean expression E b1 ∨ E b2 ∨ · · · ∨ E bk and this completes the proof. Example 3 Consider the function f : B3 → B whose truth table is shown in Figure 6.80. Since S(f) = {(0, 1, 1), (1, 1, 1)}, Theorem 2 shows that f is produced by the Boolean expression E(x, y, z) = E(0,1,1) ∨ E(1,1,1) = (x ∧ y ∧ z) ∨ (x ∧ y ∧ z). This expression, however, is not the simplest Boolean expression that produces f. Using properties of Boolean algebras, we have (x ∧ y ∧ z) ∨ (x ∧ y ∧ z) = (x ∨ x) ∧ (y ∧ z) = 1 ∧ (y ∧ z) = y ∧ z. Thus f is also produced by the simple expression y ∧ z. ◆ The process of writing a function as an "or" combination of minterms and simplifying the resulting expression can be systematized in various ways. We will demonstrate a graphical procedure utilizing what is known as a Karnaugh map. This procedure is easy to use for functions f : Bn → B if n is not too large. We will illustrate the method for n = 2, 3, and 4. If n is large or if a programmable algorithm is desired, other techniques may be preferable. The exercises explore a nongraphical method. We consider first the case where n = 2 so that f is a function of two variables, say x and y. In Figure 6.81(a), we show a 2 × 2 matrix of squares with each square containing one possible input b from B2. In Figure 6.81(b), we have placed the corresponding minterm E b inside each square.

Karnaugh Maps

The labeling of the squares in Figure 6.81 can be simplified if we omit references to x and y and display the inputs only in the row and column labels. Figure 6.82 shows the squares of Figure 6.81 with such labels. In the Karnaugh map, we note that x appears everywhere in the first row and x̄ appears everywhere in the second row. We label these rows accordingly, and we perform a similar labeling of the columns. A function f : B2 → B is shown in Figure 6.83(a). The function is 1 at (0, 0), (0, 1), and (1, 1). In Figure 6.83(b), we show the Karnaugh map of f. Since S(f) = {(0, 0), (0, 1), (1, 1)}, the corresponding expression for f is (x ∧ y) ∨ (x ∧ y) ∨ (x ∧ y). The resulting 2 × 2 array of 0's and 1's is called the Karnaugh map of f. Figure 6.82 The outcome of Example 4 is typical. When the 1-values of a function f : B2 → B exactly fill one row or one column, the label of that row or column gives the Boolean expression for f. Of course, we already know that if the 1-values of f fill just one square, then f is produced by the corresponding minterm. It can be shown that the larger the rectangle of 1-values of f, the smaller the expression for f will be. Finally, if the 1-values of f do not lie in a rectangle, we can decompose these values into the union (of possibly overlapping) rectangles. Then, by Theorem 1, the Boolean expression for f can be found by computing the expressions corresponding to each rectangle and combining them with ∨.

Example 5 Consider the function f : B2 → B whose truth table is shown in Figure 6.83(a). The Karnaugh map of f is shown in Figure 6.83(b). The 1-values of f are shown as the union of two rectangles. The rectangle having the 1's in the horizontal rectangle corresponds to the expression x (verify). The rectangle having all its 1's in the vertical rectangle corresponds to the expression y (verify). Thus f corresponds to the expression x ∨ y. In Figure 6.83(c), we show a different decomposition of the 1-values of f into rectangles. This decomposition is also correct, but it leads to the more complicated expression y ∨ (x ∧ y). We see that the decomposition into rectangles is not unique and that we should try to use the largest possible ◆ rectangles. We now turn to the case of a function f : B3 → B, which we consider to be a function of x, y, and z. We could proceed as in the case of two variables and construct a cube of side 2 to contain the values of f. This would work, but three-dimensional figures are awkward to draw and use, and the idea would not generalize. Instead, we use a rectangle of size 2 × 4. In Figures 6.84(a) and (b), respectively, we show the inputs (from B3) and corresponding minterms for each square of such a rectangle.

Consider the Karnaugh map shown in Figure 6.85. The function f : B3 → B exactly fill one row of the corresponding squares of this rectangle. This function is produced by the expression x. Similarly, the second row ... produces x̄. Expressions x ∧ y, x ∧ y ∧ z, x ∨ y, x ∧ z, and so on ... Situations are shown in Figure 6.85(a). Consider the situation shown in Figure 6.85(a). Theorem 1(a) shows that f can be computed by joining all the minterms corresponding to squares of the region with the symbol ∨. Thus f is produced by (x ∧ y ∧ z) ∨ (x ∧ y ∧ z) ∨ (x ∧ y ∧ z) ∨ (x ∧ y ∧ z) = ((x ∨ x) ∧ (y ∧ z)) ∨ ((x ∨ x) ∧ (y ∧ z)) = (1 ∧ (y ∧ z)) ∨ (1 ∧ (y ∧ z)) = (y ∧ z) ∨ (y ∧ z) = y ∧ (z ∨ z) = y. A similar computation shows that the other five regions are correctly labeled. If we think of the left and right edges of our basic rectangle as glued together to make a cylinder, as we show in Figure 6.86, we can say that the six large regions of Figure 6.85 consist of any two adjacent columns of the cylinder, or of the top or bottom half cylinder. The six basic regions shown in Figure 6.85 are the only ones whose corresponding Boolean expressions need be considered. That is why we used them to label Figure 6.84(b), and we keep them as labels for all Karnaugh maps of functions from B3 to B.

Theorem 1(b) tells us that, if f : B3 → B is the intersection of two or three of the basic six regions, then a Boolean expression for f can be computed by combining the expressions for these basic regions with ∧ symbols. Thus, if the 1-values of the function f are as shown in Figure 6.87(a), then we get them by intersecting the regions shown in Figures 6.85(a) and (d). The Boolean expression for f is therefore y ∧ z. Similar derivations can be given for the other three columns. If the 1-values of f are as shown in Figure 6.87(b), we get them by intersecting the regions of Figures 6.85(c) and (e), so a Boolean expression for f is x ∧ z. In a similar fashion, we can compute the expression for any function whose 1-values fill two horizontally adjacent squares. There are eight such functions if we again consider the rectangle to be formed into a cylinder. Thus we include the case where the 1-values of f are as shown in Figure 6.87(c). The resulting Boolean expression is x ∧ z. If we intersect three of the basic regions and the intersection is not empty, the intersection will be a single square, and the resulting Boolean expression is a minterm. In Figure 6.87(d), the 1-values of f form the intersection of the three regions shown in Figures 6.85(a), (c), and (f). The corresponding minterm is x ∧ z ∧ x. Thus we need not remember the placement of minterms in Figure 6.84(b), but instead may reconstruct it. We have seen how to compute a Boolean expression for any function f : B3 → B whose 1-values form a rectangle of adjacent squares (in the cylinder) of size 2n × 2m, n = 0, 1; m = 0, 1, 2. In general, if the set of 1-values of f do not form such a rectangle, we may write this set as the union of such rectangles. Then a Boolean expression for f is computed by combining the expressions associated with each rectangle with ∨ symbols. This is true by Theorem 1(a). 260 Chapter 6 Order Relations and Structures The preceding discussion shows that the larger the rectangles that are chosen, the simpler will be the resulting Boolean expression.

Example 6 Consider the function f whose truth table and corresponding Karnaugh map are shown in Figure 6.88. The placement of the 1's can be derived by locating the corresponding inputs in Figure 6.84(a). One way to decompose the 1-values of f is shown in Figure 6.89(a). The resulting Boolean expression is ... Example 7 Figure 6.89 shows the truth table and corresponding Karnaugh map for a function f. The decomposition into rectangles shown in Figure 6.89(b) uses the idea that the first and last columns are considered adjacent (by wraparound of the cylinder). Thus the symbols are left open ended to signify that they join in one 2 × 2 rectangle corresponding to z. The resulting Boolean expression is z ∨ (x ∧ y) (verify). 6.6 Circuit Designs 261 ... Finally, without additional comment, we present in Figure 6.90 the distribution of inputs and corresponding labeling of rectangles for the case of a function f : B4 → B, considered as a function of x, y, z, and w. Here again, we consider the first and last columns to be adjacent, and the first and last rows to be adjacent, both by wraparound, and we look for rectangles with sides of length some power of 2, so the length is 1, 2, or 4. The expression corresponding to such rectangles is given by intersecting the large labeled rectangles of Figure 6.91. z 00 01 11 10 0000 0001 0011 0010 01 0100 0101 0111 0110 11 1100 1101 1111 1110 10 1000 1001 1011 1010 (a) z x y z w (b) Figure 6.90 Example 8 Figure 6.92 shows the Karnaugh map of a function f : B4 → B. The 1-values are placed by considering the location of inputs in Figure 6.90(a). Thus f(0101) = 1, f(0001) = 0, and so on. The center 2 × 2 square represents the Boolean expression w ∧ y (verify). The four corners also form a 2 × 2 square, since the right and left edges and the top and bottom edges are considered adjacent. From a geometric point of view, we can see that if we wrap the rectangle around horizontally, getting a cylinder, then when we further wrap around vertically, we will get a torus or inner tube. On this inner tube, the four corners form a 2 × 2 square, which represents the Boolean expression w ∧ y (verify). It then follows that the decomposition leads to the Boolean expression (w ∧ y) ∨ (w ∧ y) for f. ◆ 262 Chapter 6 Order Relations and Structures Shaded region is Shaded region is z Shaded region is x (a) (b) Shaded region is z (c) (d) Shaded region is z Shaded region is x (e) (f) Shaded region is w Shaded region is y (g) (h) Figure 6.91 Example 9 Figure 6.93 shows the Karnaugh map of a function f : B4 → B. The decomposition of 1-values into rectangles of sides 1, 2, or 4 is shown in Figure 6.93. Note that the 1-values in the top row... (w ∧ x ∧ y) ∨ (w ∧ y) ∨ (x ∧ y ∧ z) ◆ Figure 6.92 Figure 6.93 ... This procedure can be reversed. ... The first term comes from the 2 × 2 square formed by joining the 1 × 2 rectangle in the upper-left corner and the 1 × 2 rectangle in the lower-left corner. The second comes from the rectangle of size 1 × 2 in the upper-right corner, and the last is a minterm corresponding to the isolated square. ◆ 6.6 Circuit Designs 263 6.6 Exercises In Exercises 1 through 6, construct Karnaugh maps for the functions whose truth tables are given. ... 7. Construct a Karnaugh map for the function f for which S(f) = {(0, 0, 0, 1), (0, 0, 1, 1), (0, 1, 0, 1), (0, 1, 1, 1), (1, 0, 0, 0), (1, 0, 1, 0)}. 8. Construct a Karnaugh map for the function f for which S(f) = {(0, 0, 0, 0), (0, 0, 1, 1), (0, 1, 1, 0), (1, 0, 0, 0), (1, 1, 1, 1)}. In Exercises 9 through 16 (Figures 6.94 through 6.101), Karnaugh maps are shown of functions, and a decomposition of 1-values into rectangles is given. Write the Boolean expression for these functions, which arise from the maps and rectangular decompositions. Chapter 6 Order Relations and Structures 264 ... Figure 6.98 Figure 6.99 Figure 6.100 Figure 6.101 ... In Exercises 17 through 24 (Figures 6.102 through 6.109), use the Karnaugh map method to find a Boolean expression for the function f. ... 25. Let f be the function of Exercise 1. Let f be the function of Exercise 4. 26. Let f be the function of Exercise 5. Key Ideas for Review 22. Let f be the function of Exercise 6. 23. Let f be the function of Exercise 7. 24. Let f be the function of Exercise 8. The following exercises develop another way to produce an expression for a Boolean function. For Exercises 25 and 26, let f : B2 → B with S(f) = {(0, 0), (0, 1), (1, 0)}. 25. Give the corresponding minterms for each element in S(f). 26. If two elements of S(f) are adjacent, or differ in exactly one position, the variable xi in that position is not needed in the sum of the two corresponding minterms, as follows. Suppose the corresponding minterms at a1 ∧ a2 ∧ · · · ∧ an and b1 ∧ b2 ∧ · · · ∧ bn differ only at position i. Replace this pair with the reduced minterm a1 ∧ a2 ∧ · · · ∧ ai−1 ∧ ai+1 ∧ · · · ∧ an, where ai is not present. ... 265 (a) Give the reduced minterms for each pair of elements in S(f) that differ in exactly one position. (b) Verify that f is produced by the join of the reduced minterms in part (a). The process in Exercise 26 can be repeated. For Exercises 27 through 29, let f : B3 → B with S(f) = {(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 1)}. 27. Give the reduced minterms for each pair of elements in S(f) that differ in exactly one position. 28. Using the reduced minterms produced for Exercise 27, again compare each pair of reduced minterms that differ in exactly one position. 29. (a) Form the join of the expressions produced by the reduced minterms of Exercise 28 and any (reduced) minterms that were not used. (b) Verify that f is produced by the expression formed in part (a) produces f. Tips for Proofs Statements of the form ∀x P(x) or ∃x Q(x) are candidates for proof by contradiction, since it is natural to explore what would happen if there were an object with property P or if there were an object with property not Q. Figure 6.102 ... Many examples and exercises in this chapter ask that you check whether a specified lattice is a Boolean algebra or prove that it is not. The fact that the number of elements is a power of 2 is a necessary condition, but not a sufficient condition. This means that if the number of elements is not a power of 2, then the lattice is not a Boolean algebra, but there are lattices with 2n elements that are not Boolean algebras. The check on a distributivity lattice DN is an easy one and worth memorizing. If the Hasse diagram is "small," then comparing it with those of the Bn is an efficient way to carry out the check. Attempting to construct an isomorphism between the given lattice and a known Boolean algebra is the next most efficient method. As a last resort, try to verify that the properties of a Boolean algebra are satisfied by the lattice. But be sensible about this. Use the "single" cases first—there is a unique greatest element 1; 0 ∨ 1; and so on—and those that require working through lots of cases, such as the associative property for ∧.

Key Ideas for Review • Partial order on a set: relation that is reflexive, antisymmetric, and transitive • Partially ordered set or poset: set together with a partial order • Linearly ordered set: partially ordered set in which every pair of elements is comparable • Theorem: If A and B are posets, then A × B is a poset with the product partial order. • Dual of a poset (A, ≤): the poset (A, ≥), where ≥ denotes the inverse of ≤ • Hasse diagram: see page 222 • Topological sorting: see page 223 • Isomorphism of posets: see page 228 • Theorem: A finite nonempty poset has at least one maximal element and at least one minimal element. • Greatest (least) element of a poset A: see page 229 • Theorem: A poset has at most one greatest element and at most one least element. 266 Chapter 6 Order Relations and Structures • Upper (lower) bound of subset B of poset A: element a ∈ A such that b ≤ a (a ≤ b) for all b ∈ B • Least upper bound of subset B of poset A: element a of A such that a is an upper (lower) bound of B and a ≤ a' (a' ≤ a), where a' is any upper (lower) bound of B • Lattice: a poset in which every subset consisting of two elements has a LUB and a GLB • Theorem: If L1 and L2 are lattices, then L = L1 × L2 is a lattice. • Theorem: Let L be a lattice, and a, b ∈ L. Then (a) a ∨ b = b if and only if a ≤ b. (b) a ∧ b = a if and only if a ≤ b. (c) a ∧ b = a if and only if a ∨ b = b. • Theorem: Let L be a lattice. Then 1. (a) a ∨ a = a (b) a ∧ a = a 2. (a) a ∨ b = b ∨ a (b) a ∧ b = b ∧ a 3. (a) a ∨ (b ∨ c) = (a ∨ b) ∨ c (b) a ∧ (b ∧ c) = (a ∧ b) ∧ c 4. (a) a ∨ (a ∧ b) = a (b) a ∧ (a ∨ b) = a • Theorem: Let L be a lattice, and a, b, c ∈ L. 1. If a ≤ b, then (a) a ∨ c ≤ b ∨ c (b) a ∧ c ≤ b ∧ c 2. a ≤ c and b ≤ c if and only if a ∨ b ≤ c 3. c ≤ a and c ≤ b if and only if c ≤ a ∧ b 4. If a ≤ b and c ≤ d, then (a) a ∨ c ≤ b ∨ d (b) a ∧ c ≤ b ∧ d • Isomorphic lattices: see page 240 • Bounded lattice: lattice that has a greatest element I and a least element 0 • Theorem: A finite lattice is bounded. Distributive lattice: lattice that satisfies the distributive laws: a ∧ (b ∨ c) = (a ∧ b) ∨ (a ∧ c), a ∨ (b ∧ c) = (a ∨ b) ∧ (a ∨ c) • Complement of a: element a' ∈ L (bounded lattice) such that a ∨ a' = I and a ∧ a' = 0. • Theorem: Let L be a bounded distributive lattice. If a complement exists, it is unique. • Complemented lattice: bounded lattice in which every ele- ment has a complement • Boolean algebra: a lattice isomorphic with (P(S), ⊆) for some finite set S • Properties of a Boolean algebra: see page 247 • Truth tables: see page 250 • Boolean expression: see page 251 • Minterm: a Boolean expression of the form x̄1 ∧ x̄2 ∧ · · · ∧ x̄n, where each x̄k is either xk or xk • Theorem: Any function f : Bn → B is produced by a Boolean expression. • Karnaugh map: see page 257 Chapter 6 Self-Test 1. Upon what common relation is the concept of partial order based? 2. What is a model for any Boolean algebra? 3. What kind of mathematical object is a Karnaugh map and what is it used for? 5. What is the relationship between Boolean functions and Boolean expressions? 6. Determine whether the given relation is a partial order. Explain your answer. (a) A is any set; aR b if and only if a ≠ b. (b) A is the set of parallel lines in the plane; l1 R l2 if and only if l1 coincides with l2 or l1 is parallel to l2. x1 Figure 6.102 to 6.104 Figure 6.105 Figure 6.107 Figure 6.108 Figure 6.109 (A, ≤) Figure 6.106 ... 7. Find the greatest element, least element, and all maximal and minimal elements of A. (a) A = {2, 3, 4, 6, 8, 12, 24, 48} and ≤ is divisibility. (b) A = {2, 3, 4, 6, 12, 18, 24, 36} and ≤ is divisibility ... 8. Given the Hasse diagram in Figure 6.102, (a) Draw the Hasse diagram of the partial order... (b) Is (A, ≤) a lattice? 9. Determine whether the poset is a lattice. ... 10. Show that a linearly ordered poset is a distributive lattice. 11. Find the complement of each element in D105. Experiment 6 Let A = {a, b, c, d} and R be a relation on A whose matrix is ... ... (c) Which of these posets are not lattices? Explain. (b) Which of these posets are not Boolean algebras? Explain. ... Let (D63, ≤) is a Boolean algebra with ... Prove or disprove the statement: (D63, ≤), is a distributive lattice. 14. Find a Boolean expression that produces the function f whose truth table is ... 15. Use the Karnaugh map method to find a Boolean expression for the function f whose truth table is as follows. ... Experiment 6 Many real-life and computer processes can be modeled by a partial order that shows the sequence of actions to complete the process. Section 6.1 contains a brief description of how to convert such a partial order into a linear order that could be followed by a person or a computer program. In many cases, a strict linear order is not necessary, for example, if several actions can take place concurrently. In this experiment you will learn the basic ideas of another modeling technique for concurrent actions. Part I. To obtain a degree in computer science at Old U, the following courses are required: Computer Science I, Computer Science II, Discrete Mathematics, Data Structures, Compiler Design, Algorithm Analysis, Assembly Architecture, Formal Languages, Operating Systems, and Complexity Theory. Computer Science I has no prerequisites. Computer Science II requires Computer Science I. Computer Science II is a prerequisite for Data Structures, Assembly Architecture, and Formal Languages. Data Structures is a prerequisite for Algorithm Analysis and Operating Systems. Algorithm Analysis is a prerequisite for Complexity Theory; Compiler Design requires both Assembly Architecture and Formal Languages. Discrete Mathematics is a prerequisite for Data Structures and Algorithm Analysis. 1. Give a Hasse diagram that shows the prerequisite relationships among these courses. 268 Chapter 6 Order Relations and Structures 2. Suppose you were able to take only one course each term. Give a topological sorting of the partial order in Question 1. p1 p2 t3 p4 p5 t1 t2 Figure 2 Figure 1 p3 p6 p5 t3 Part II. Progressing through the required courses in a linear fashion is not necessary and certainly not optimal in terms of how long it will take to obtain your degree. Here we present a different type of digraph to model conditions (prerequisites) and possibly concurrent actions. A Petri net is a digraph whose set of vertices can be partitioned into P, the set of places (or conditions), and T, the set of transitions (or actions), along with a function f that associates a nonnegative number with each element of P. Any edge in the digraph is between an element of P and an element of T. That is, the digraph is bipartite. Elements of P are denoted by circles and those of T by vertical bars. As a way to indicate that a condition pi ∈ P is satisfied, one or more dots, called tokens, are marked inside the circle for pi. Allowing more than one token at a place permits Petri nets to model a broad range of situations. The function f is defined by f(pi) is the number of tokens at pi; hence, f is sometimes called a marking of the Petri net. Let P = {p1, p2, ..., pn}, and T = {t1, t2, ..., tk}. The Petri net depicted by Figure 1, given P, T, R, and f. Completing the action represented by t ∈ T is called firing the transition. A transition can fire only when the conditions for that action are met. Formally, a transition t can be fired only when every pi such that (pi, t) ∈ R contains at least one token. When this is true, we say that t is enabled. Firing a transition t creates a new Petri net in which a token is removed from each pi such that (pi, t) ∈ R and a token is placed in each pj such that (t, pj) ∈ R. In Figure 1, firing t produces the net in Figure 2. Only t2 is enabled in Figure 2. The result of firing t2 is shown in Figure 4. If several transitions are enabled, then any one of them may fire at any time. This models the sometimes random nature of choices for concurrent actions. 1. Create a Petri net to represent the course prerequisite situation of Part I. Identify the set of places and the set of transitions. Based on your academic record, mark the net appropriately. Which transitions are enabled in this digraph? 2. (a) Give the new marking function for the net represented by Figure 3. (b) Give the new marking function for the net represented by Figure 4. Part III. One simple example of concurrent processes lacks a common feature of concurrent processes, repeatability. Consider the following Petri net, N, shown in Figure 5. N Figure 5 Experiment 6 269 We see that each transition in T can be enabled and fired repeatedly. A marking of a Petri net is live if every transition can be fired by some sequence of transition firings. Not every marking is live. In the net shown in Figure 6, once t1 or t2 is fired, either t3 or t4 is enabled (but not both). After that transition is fired, no further action is possible. t3 t4 t1 t2 Figure 6 If no transitions can be fired, we say there is a deadlock. 1. Give a marking for N different from that in Figure 5 that is live. 2. Is it possible to mark Figure 6 using some tokens so that a deadlock never occurs? If so, draw this marked Petri net. If not, explain why. Part IV. Figure 7 gives a model of a simple computer network where two workers share a high-speed Internet connection and a database. Only one user of a resource is allowed at a time, but both resources are needed for some tasks the workers perform. Internet available B releases Internet Internet in use by B A releases Internet A uses Internet B acquires Internet A acquires Internet B does task A does task Database in use by A Database in use by B B releases database A releases database Database available B acquires database A acquires database Figure 7 1. Give a sequence of transition firings that returns the net to its original marking. Explain in words what the workers have done. 2. Give a sequence of transition firings that produces deadlock in this net. Explain in words what the workers have done. 3. If they were in one room, the workers might choose to communicate to avoid deadlock, but this is not always possible. Redesign the Petri net in Figure 7 so that deadlock cannot occur. Consider using conditions such as p: The system is available; q: A is in control of the system; and r: B is in control of the system. Clearly identify all conditions and actions. CHAPTER 7 Trees Prerequisites: Chapter 4 In this chapter we study a special type of relation that is exceptionally useful in a variety of biology and computer science applications and is usually represented by its digraph. These relations are essential for the construction of databases and language compilers, to name just two important computer science areas. They are called trees or sometimes rooted trees, because of the appearance of their digraphs. Looking Back Trees were first used in 1847 by the German mathematician Karl Georg Christian von Staudt in his work on projective geometry, and in the same year by the German physicist Gustav Robert Kirchhoff in a paper on electrical networks. However, it is Arthur Cayley (who we met in Chapter 1) who first used the word tree in a mathematics paper on differential transformations. Trees are used in a wide variety of applications, such as family trees, organizational charts, sports competitions, biology, and computer file structures. The great German composer Johann Sebastian Bach was very proud of his long family musical heritage, dating back to the late 1500s. In 1735, he prepared a genealogy of his family entitled, Ursprung der musicalisch-Bachischen Familie ("Origin of the Musical Bach Family"). A small part of this family is shown in the tree below. During their annual family reunions they performed together, and on some occasions there were more than 100 family members present. Veit Johannes Christoph Heinrich Johann Egidius Georg Christoph Johann Christoph Johann Michael Johann Bernhard Johann Ernst Johann Christoph Johann Nikolaus Johann Sebastian Johann Christoph Wilhelm Friedemann Carl Philipp Emanuel Johann Christoph Friedrich Family tree of Johann Sebastian Bach 270 7.1 Trees 7.1 Trees Let A be a set, and let T be a relation on A. We say that T is a tree if there is a vertex v0 in A with the property that there exists a unique path in T from v0 to every other vertex in A, but no path from v0 to v0. We show below that the vertex v0, described in the definition of a tree, is unique. It is often called the root of the tree T, and T is then referred to as a rooted tree. We write (T, v0) to denote a rooted tree T with root v0. If (T, v0) is a rooted tree on the set A, an element v of A will often be referred to as a vertex in T. This terminology simplifies the discussion, since it often happens that the underlying set A of T is of no importance. To help us see the nature of trees, we will prove some simple properties satisfied by trees. THEOREM 1 Let (T, v0) be a rooted tree. Then (a) There are no cycles in T. (b) v0 is the only root of T. (c) Each vertex in T, other than v0, has in-degree one, and v0 has in-degree zero. Proof (a) Suppose that there is a cycle q in T, beginning and ending at vertex v. By definition of a tree, we know that v ≠ v0, and there must be a path p from v0 to v. Then q ◦ p (see Section 4.3) is a path from v0 to v that is different from p, and this contradicts the definition of a tree. (b) If v0' is another root of T, there is a path p from v0 to v0' and a path q from v0' to v0 (since v0' is a root). Then q ◦ p is a cycle from v0 to v0, and this is impossible by definition. Hence the vertex v0 is the unique root. (c) Let w1 be a vertex in T other than v0. Then there is a unique path v0, . . . , vk, w1 from v0 to w1 in T. This means that (vk, w1) ∈ T, so w1 has in-degree at least one. If the in-degree of w1 is more than one, there must be distinct vertices w2 and w3 such that (w2, w1) and (w3, w1) are both in T. If w2 ≠ v0 and w3 ≠ v0, there are paths p2 from v0 to w2 and p3 from v0 to w3, by definition. Then (w2, w1) ◦ p2 and (w3, w1) ◦ p3 are two different paths from v0 to w1, and this contradicts the definition of a tree with root v0. Hence, the in-degree of w1 is one. We leave it as an exercise to complete the proof if w2 = v0 or w3 = v0 and to show that v0 has in-degree zero. ■ Theorem 1 summarizes the geometric properties of a tree. With these properties in mind, we can see how the digraph of a typical tree must look. Let us first draw the root v0. No edges enter v0, but several may leave, and we draw these edges downward. The terminal vertices of the edges beginning at v0 will be called the level 1 vertices, while v0 will be said to be at level 0. Also, v0 is sometimes called the parent of these level 1 vertices, and the level 1 vertices are called the offspring of v0. This is shown in Figure 7.1(a). Each vertex at level 1 has no other edges entering it, by part (c) of Theorem 1, but each of these vertices may have edges leaving the vertex. The edges leaving a vertex of level 1 are drawn downward and terminate at various vertices, which are said to be at level 2. Figure 7.1(b) shows the situation at this point. A parent-offspring relationship holds also for these levels (and at every consecutive pair of levels). For example, v3 would 272 Chapter 7 Trees v0 Level 0 v1 v2 v3 Level 1 v4 (a) v5 v6 v7 v8 v9 Level 2 (b) Figure 7.1 be called the parent of the three offspring v7, v8, and v9. The offspring of any one vertex are sometimes called siblings. The preceding process continues for as many levels as are required to complete the digraph. If we view the digraph upside down, we will see why these relations are called trees. The largest level number of a tree is called the height of the tree. We should note that a tree may have infinitely many levels and that any level other than level 0 may contain an infinite number of vertices. In fact, any vertex could have infinitely many offspring. However, in all our future discussions, trees will always be assumed to have a finite number of vertices. Thus the trees will always have a bottom (highest-numbered) level consisting of vertices with no offspring. The vertices of the tree that have no offspring are called the leaves of the tree. The vertices of a tree that lie at any one level simply form a subset of A. Often, however, it is useful to suppose that the offspring of each vertex of the tree are linearly ordered. Thus, if a vertex v has four offspring, we may assume that they are ordered, so we may refer to them as the first, second, third, or fourth offspring of v. Whenever we draw the digraph of a tree, we automatically assume some ordering of the offspring of each vertex by arranging them from left to right. Such a tree will be called an ordered tree. Generally, ordering of offspring in a tree is not explicitly mentioned. If ordering is needed, it is usually introduced at the time when the need arises, and it often is specified by the way the digraph of the tree is drawn. The following relational properties of trees are easily verified. THEOREM 2 Let (T, v0) be a rooted tree on a set A. Then (a) T is irreflexive. (b) T is asymmetric. (c) If (a, b) ∈ T and (b, c) ∈ T, then (a, c) ∉ T, for all a, b, and c in A. Proof. The proof is left as an exercise. ■ Example 1 Let A be the set consisting of a given woman v0 and all of her female descendants. We now define the relation T on A: If v1 and v2 are elements of A, then v1 T v2 if and only if v1 is the mother of v2. The relation T on A is a rooted tree with root v0. ◆ Example 2 Let A = {v1, v2, v3, v4, v5, v6, v7, v8, v9, v10} and T = {(v2, v3), (v2, v1), (v4, v5), (v4, v6), (v5, v8), (v6, v7), (v4, v2), (v7, v9), (v7, v10)}. Show that T is a rooted tree and identify the root. Solution Since no paths begin at vertices v1, v3, v8, v9, and v10, these vertices cannot be roots of a tree. There are no paths from vertices v6, v7, v2, and v5 to vertex v4, so we must eliminate these vertices as possible roots. Thus, if T is a rooted tree, its root must be vertex v4. It is easy to show that there is a path from v4 to every other vertex. For example, the path v4, v6, v7, v9 leads from v4 to v9, since (v4, v6), (v6, v7), and (v7, v9) are all in T. We draw the digraph of T, beginning with vertex v4, and with edges shown downward. The result is shown in Figure 7.2. A quick inspection of this digraph shows that paths from vertex v4 to every other vertex are unique, and there are no paths from v4 to v4. Thus T is a tree with root v4. ◆ 7.1 Trees 273 v4 v2 v5 v6 v3 v1 v8 v7 v9 v10 Figure 7.2 If n is a positive integer, we say that a tree T is an n-tree if every vertex has at most n offspring. If all vertices of T, other than the leaves, have exactly n offspring, we say that T is a complete n-tree. In particular, a 2-tree is often called a binary tree, and a complete 2-tree is often called a complete binary tree. Binary trees are extremely important, since there are efficient methods of implementing them and searching through them on computers. We will see some of these methods in Section 7.3, and we will also see that any tree can be represented as a binary tree. Let T be a rooted tree on A and let v be a vertex of T. The subtree of T beginning at v is the restriction of T to the set B consisting of v and all its descendants, that is, all vertices of T that can be reached by a path beginning at v. Observe that B ⊆ A. Let T(v) be the restriction of the relation T to B; that is, T ∩ (B × B) (see Section 4.2). In other words, T(v) is the relation that results from T in the following way. Delete all vertices that are not descendants of v and all edges that do not begin and end at any such vertex. Then we have the following result. v0 v5 273 Proof By the definition of T(v), we see that there is a path from v to every other vertex in T(v). If there is a vertex w in T(v) such that there are two distinct paths q and q' from v to w, and if p is the path in T from v0 to v, then q ◦ p and q' ◦ p would be two distinct paths in T from v0 to w. This is impossible, since T is a tree with root v0. Thus each path from v to another vertex w in T(v) must be unique. Also, if q is a cycle at v in T(v), then q is also a cycle in T. This contradicts Theorem ■ 1(a); therefore, q cannot exist. It follows that T(v) is a tree with root v. v6 v3 v8 v9 T(v) is a rooted tree with root v. We will say that T(v) is the subtree of T beginning at v. Figure 7.3 Example 3 Subtrees and substructures (Section 6.3) are examples of substructures of a mathematical structure. In general, if a set A and a collection of operations and their properties form a mathematical structure (Section 1.6), then a substructure of the same type is a subset of A with the same operations that satisfies all the properties that define this type of structure. This concept is used again in Chapter 9. Consider the tree T of Example 2. This tree has root v4 and is shown in Figure 7.2. Figure 7.3 shows the subtrees of T(v5), T(v2), and T(v6) of T. ◆ 274 Chapter 7 Trees 7.1 Exercises In Exercises 1 through 8, determine if the relation R is a tree. If it is a tree, find the root. 1. A = {a, b, c, d, e} R = {(a, d), (b, c), (c, a), (d, e)} 3. A = {a, b, c, d, e, f} R = {(a, b), (c, e), (f, a), (f, c), (f, d)} 5. A = {1, 2, 3, 4, 5, 6} R = {(2, 1), (3, 4), (5, 2), (6, 5), (6, 3)} 7. A = {t, u, v, w, x, y, z} R = {(t, u), (u, w), (u, x), (u, v), (v, z), (v, y)} 2. A = {a, b, c, d, e} R = {(a, b), (b, e), (c, d), (d, b), (c, a)} 4. A = {1, 2, 3, 4, 5, 6} R = {(1, 1), (2, 1), (2, 3), (3, 4), (4, 5), (4, 6)} 6. A = {1, 2, 3, 4, 5, 6} R = {(1, 2), (1, 3), (4, 2), (5, 4), (5, 6)} 8. A = {u, v, w, x, y, z} R = {(u, x), (u, v), (w, v), (x, z), (x, y)} In Exercises 9 through 13, consider the rooted tree (T, v0) shown in Figure 7.4. 9. (a) What are the levels of v3, v5, and v7? (b) List all level 2 vertices. 10. What are the siblings of v8? 11. What are the descendants of v8? 12. (a) Compute the tree T(v2). (b) Compute the tree T(v3). 13. (a) What is the height of (T, v0)? (b) What is the height of T(v5)? v0 v1 v2 v3 v4 v5 v6 v7 v8 v9 v10 v11 v12 v13 v14 v15 v16 Figure 7.4 In Exercises 14 through 18, consider the rooted tree (T, v0) shown in Figure 7.5. 14. (a) List all level 4 vertices. (b) List all leaves. 15. What are the siblings of v2? 16. What are the descendants of v2? 17. (a) Compute the tree T(v4). (b) Compute the tree T(v2). 18. (a) What is the height of (T, v0)? (b) What is the height of T(v4)? 19. Show that the maximum number of vertices in a binary tree of height n is 2n+1 − 1. 20. Prove that the largest possible number of leaves in an n-tree of height k is nk. 21. If T is a complete n-tree with exactly three levels, prove that the number of vertices of T must be 1 + kn, where 2 ≤ k ≤ n + 1. 22. Let T be a complete n-tree with m leaves. Prove that the number of vertices of T that are not leaves is (m − 1)/(n − 1), in which k are nonleaves and l are leaves. (That is, m = k + l.) Prove that m = nk + 1 and l = (n − 1)k + 1. 24. Prove Theorem 2(a). 25. Prove Theorem 2(b). 26. Prove Theorem 2(c). 27. Let T be a tree. Suppose that T has r vertices and s edges. Find a formula relating r to s. 28. Draw all possible unordered trees on the set S = {a, b, c}. 29. What is the maximum height for a tree on S = {a, b, c, d, e}? Explain. 30. What is the maximum height for a complete binary tree on S = {a, b, c, d, e}? 31. Show that if (T, v0) is a rooted tree, then v0 has in-degree zero. 7.2 275 32. An n-tree of height k is balanced if all the leaves are at level k or k − 1. Which of the following are balanced trees? Figure 7.5 Tree shown in Figure 7.5 (a) T of Figure 7.4 ... (b) T ... 33. Let T be a balanced n-tree with 125 leaves. (a) What are the possible values of n? (b) What are the possible values for the height of T? 34. Let T be a balanced n-tree of height 4 with 125 leaves. What are the possible values of n? Labeled Trees It is often useful to label the vertices or edges of a digraph to indicate that the digraph is being used for a particular purpose. This is especially true for many uses of trees in computer science and biology. We will now give a series of examples in which the sets of vertices of the trees are not important, but rather the utility of the tree is best emphasized by the labels on these vertices. Thus we will represent the vertices simply as dots and show the label of each vertex next to the dot representing that vertex. Consider the fully parenthesized, algebraic expression (3 − (2 × x)) + ((x − 2) − (3 + x)). We assume, in such an expression, that no operation such as −, +, ×, or ÷ can be performed until both of its arguments have been evaluated, that is, until all computations inside both the left and right arguments have been performed. Therefore, we cannot perform the central addition until we have evaluated (3 − (2 × x)) and ((x − 2) − (3 + x)). We cannot perform the central subtraction in ((x − 2) − (3 + x)) until we evaluate (x − 2) and (3 + x), and so on. It is easy to see that each such expression has a central operator, corresponding to the last computation that can be performed. Thus + is central to the main expression, − is central to (3 − (2 × x)), and so on. An important graphical representation of such an expression is as a labeled binary tree. In this tree the root is labeled with the central operator of the main expression. The two offspring of the root are labeled with the central operator of the expressions for the left and right arguments, respectively. If either argument is a constant or variable, instead of an expression, this constant or variable is used to label the corresponding offspring vertex. This process continues until the expression is exhausted. Figure 7.6 shows the tree for the original expression of this example. To illustrate the technique further, we have shown in Figure 7.7 the tree corresponding to the full parenthesized expression ((3 × (1 − x)) ÷ ((4 + (7 − (y + 2))) × (7 + (x ÷ y)))). Our construction of the computer implementation of a tree also starts with a binary tree. We start with an n-tree (T, v0). Each vertex in T has at most n offspring. We imagine that each vertex potentially has exactly n offspring, which would be ordered from 1 to n, but that some of the offspring in the sequence may be missing. The remaining offspring are labeled with the position that they occupy in the hypothetical sequence. Thus the offspring of any vertex are labeled with distinct numbers from the set {1, 2, . . . , n}. 276 Chapter 7 Trees − 3 + × 2 − x x − 2 3 + x Figure 7.6 Such a labeled digraph is sometimes called positional, and we will also use this term. Note that positional trees are also ordered trees. When drawing the digraphs of a positional tree, we will imagine that the n offspring positions for each vertex are arranged symmetrically below the vertex, and we place in its appropriate position each offspring that actually exists. Figure 7.8 shows the digraph of a positional 3-tree, with all actually occurring positions labeled. If offspring 1 of any vertex v actually exists, the edge from v to that offspring is drawn sloping to the left. Offspring 2 of any vertex v is drawn vertically downward from v, whenever it occurs. Similarly, offspring labeled 3 will be drawn to the right. Naturally, the root is not labeled, since it is not an offspring. L 2 1 1 2 2 3 3 1 3 R L 3 3 1 3 R Figure 7.8 Figure 7.7 The positional binary tree is of special importance. In this case, for obvious reasons, the positions for potential offspring are often labeled left and right, instead of 1 and 2. Figure 7.9 shows the digraph of a positional binary tree, with offspring labeled L for left and R for right. Labeled trees may have several sets of labels, all in force simultaneously. We will usually omit the left-right labels on a positional binary tree in order to emphasize other useful labels. The positions of the offspring will then be indicated by the direction of the edges, as we have drawn them in Figure 7.9. 7.2 Labeled Trees 277 Binary Positional Trees as Data Structures In Section 4.6 we discussed an idealized information storage unit called a cell. A cell contains two items. One is data of some sort and the other is a pointer to the next cell, that is, an address where the next cell is located. A collection of such cells, linked together by their pointers, is called a linked list. The discussion in Section 4.6 included both a graphical representation of linked lists, and an implementation of them that used arrays. We need here an extended version of this concept, called a doubly linked list, in which each cell contains two pointers and a data item. We use the pictorial symbol to represent these new cells. The center space represents data storage and the two pointers, called the left pointer and the right pointer, are represented as before by dots and arrows. Once again we use the symbol for a pointer signifying no additional data. Sometimes a doubly linked list is arranged so that each cell points to both the next cell and the previous cell. This is useful if we want to search through a set of data items in either direction. Our use of doubly linked lists here is very different. We will use them to represent binary positional labeled trees. Each cell will correspond to a vertex, and the data part can contain a label for the vertex or a pointer to such a label. The left and right pointers will direct us to the left and right offspring vertices, if they exist. If either offspring fails to exist, the corresponding pointer will be We implement this representation by using three arrays: LEFT holds pointers to the left offspring, RIGHT holds the pointers to the right offspring, and DATA holds information or labels related to each vertex, or pointers to such information. The value 0, used as a pointer, will signify that the corresponding offspring does not exist. To the linked list and the arrays we add a starting entry that points to the root of the tree. Example 1 We consider again the positional binary tree shown in Figure 7.6. In Figure 7.10(a), we represent this tree as a doubly linked list, in symbolic form. In Figure 7.10(b), we show the implementation of this list as a sequence of three arrays (see also Section 4.6). The first row of these arrays is just a starting point whose left pointer Start + − × 3 2 x − − x 2 3 x (a) INDEX LEFT 2 1 3 2 4 0 3 0 5 6 0 7 8 0 0 9 0 10 11 0 12 13 0 0 14 0 0 DATA RIGHT 0 + − 3 × 2 x − − x 2 + 3 x 0 8 − 5 3 0 7 0 0 0 0 − 12 0 0 0 11 0 0 13 0 0 ... (b) Figure 7.10 278 Chapter 7 Trees points to the root of the tree. As an example of how to interpret the three arrays, consider the fifth entry in the array DATA, which is ×. The fifth entry in LEFT is 6, which means that the left offspring of × is the sixth entry in the array DATA, or 2. Similarly, the fifth entry in RIGHT is 7, so the right offspring of × is the seventh entry in DATA, or x. ◆ Example 2 Now consider the tree of Figure 7.7. We represent this tree as a doubly linked list. As before, Figure 7.11(b) shows the implementation of this linked list in three arrays. Again, the first entry is a starting point whose left pointer points to the root of the tree. We have listed the vertices in a somewhat unnatural order to show that, if the pointers are correctly determined, any ordering of vertices can be used. ◆ INDEX LEFT ... Figure 7.11 Huffman Code Trees The simple ASCII code for the letters of the English alphabet represents each letter by a string of 0's and 1's of length 7. A Huffman code also uses strings of 0's and 1's, but the strings are of variable length. The more frequently used letters are assigned shorter strings. In this way messages may be encoded using fewer bits than with the ASCII code. A labeled positional binary tree is an efficient way to represent a Huffman code. The string assigned to a letter gives a path from the root of the tree to the leaf labeled with that letter; a 0 indicates that we should go to the left, a 1 that we should go to the right. To decode a message, we follow the path from the root and continue with the remainder of the string. We then decode the string 0101100. 7.2 Labeled Trees 279 Solution We begin at the root and move down to the left (using the first 0). This leaf is labeled E. We return to the root and use the string 101100. We go to the right, then to the left and record A. Repeating the process with 1100 produces S and finally another E. The string 0101100 represents EASE. Example 3 One disadvantage of a Huffman code is the potential for errors in transmission. If an error was made in transmitting string 0101100 and string 0101101 is received, then the word read is EAR, not EASE. ◆ 7.2 Exercises In Exercises 1 through 10, construct the tree of the algebraic expression. 1. (7 + (6 − 2)) − (x − (y − 4)) 2. (x + (y − (x + y))) × ((3 ÷ (2 × 7)) × 4) 3. 3 − (x + (6 × (4 ÷ (2 − 3)))) 4. (((2 × 7) + x) ÷ y) ÷ (3 − 11) 5. ((2 + x) − (x × 3)) − (x − 2) 6. (11 − (11 × (11 + 11))) + (11 + (11 × 11)) 7. (3 − (2 − (11 − (9 − 4)))) ÷ (2 + (3 + (4 + 7))) 8. (x ÷ y) ÷ ((x + 3) − (z ÷ 4)) 9. ((2 × x) + (3 − (4 × x))) + (x − (3 × 11)) 10. ((1 + 1) + (1 − 2)) ÷ ((2 − x) + 1) 11. The annual NCAA Basketball Tournament begins with 64 teams. One loss eliminates a team. Suppose the results of the tournament are represented in a binary tree whose leaves are labeled with the original teams and whose interior vertices are labeled with the winner of the game between the children of the vertex. What is the height of the tree created? 12. Make a "family" tree that shows the descendants of one of your great grandparents. 13. Give an example of a situation whose representation would be a tree with height 29. 14. Prove that a binary tree of height n has at most 2n+1 − 1 vertices. 15. How many distinct positional 3-trees are there with height 2? 16. Construct the digraphs of all distinct binary positional trees having three or fewer edges and height 2. 17. The following is the doubly linked list representation of a binary positional labeled tree. Construct the digraph of this tree with each vertex labeled as indicated. INDEX LEFT DATA RIGHT ... 18. The following is the doubly linked list representation of a binary positional labeled tree. Construct the digraph of this tree with each vertex labeled as indicated. INDEX LEFT DATA RIGHT ... In Exercises 19 through 22, give arrays LEFT, DATA, and RIGHT describing the tree given as a doubly linked list. 19. Exercise 1 20. Exercise 2 21. Exercise 5 22. Exercise 8 ... In Exercises 25 and 26, give a representation as a doubly linked list of the tree and construct the digraph ... 23. Exercise 12 24. Exercise 23 25. Exercise 7 26. Exercise 9 In Exercises 27 and 28, use the Huffman code tree in Figure 7.12 to decode the message. 27. (a) 1100101011 (b) 0101101011 ... 28. (a) 0110101 (b) 0111101110 ... In Exercises 29 and 30, use the Huffman code tree in Figure 7.13 to decode the following messages. 30. Use the Huffman code tree in Figure 7.12 to find the string that represents the given word. (b) SEA (a) CARE (d) CASE (c) ACE 27. Construct the labeled tree that represents the Huffman code. A 000 B 01 C 001 D 1100 E 1101 Tree Searching There are many occasions when it is useful to consider each vertex of a tree T exactly once in some specific order. As each successive vertex is encountered, we may wish to take some action or perform some computation appropriate to the application being represented by the tree. For example, if the tree T is labeled, the label on each vertex may be displayed. If T is the tree of an algebraic expression, then at each vertex we may want to perform the computation indicated by the operator that labels that vertex. Performing appropriate tasks at a vertex will be called visiting the vertex. This is a convenient, nonspecific term that allows us to write algorithms without giving the details of what constitutes a "visit" in each particular case. The process of visiting each vertex of a tree in some specific order will be called searching the tree or performing a tree search. In some texts, this process is called walking or traversing the tree. Let us consider tree searches on binary positional trees. Recall that in a binary positional tree each vertex has two potential offspring. We denote these potential offspring by vL (the left offspring) and vR (the right offspring), and either or both may be missing. If a binary tree T is not empty, its root will be denoted by v0. If vL exists, the subtree T(vL) will be called the left subtree of T, and T(vR) is the right subtree of T, if vR exists. Note that T(v L), if it exists, is a positional binary tree with root vL, and similarly T(vR) is a positional binary tree with root vR. This notation allows us to specify searching algorithms in a natural and powerful recursive form. Recall that recursive algorithms are those that refer to themselves. We first describe a method of searching called a preorder search. For the moment, we leave the details of visiting a vertex of a tree unspecified. Consider the following algorithm for searching a positional binary tree T with root v. 7.3 Tree Searching 281 Algorithm PREORDER Step 1 Visit v. Step 2 If vL exists, then apply this algorithm to (T(vL), vL). Step 3 If vR exists, then apply this algorithm to (T(vR), vR). ● Informally, we see that a preorder search of a tree consists of the following three steps: 1. Visit the root. 2. Search the left subtree if it exists. 3. Search the right subtree if it exists. Example 1 Let T be the labeled, positional binary tree whose digraph is shown in Figure 7.14(a). The root of this tree is the vertex labeled A. Suppose that, for any vertex v of T, visiting v prints out the label of v. Let us now apply the preorder search algorithm to this tree. Note first that if a tree consists only of one vertex, its root, then a search of this tree simply prints out the label of the root. In Figure 7.14(b), we have placed boxes around the subtrees of T and numbered these subtrees (in the corner of the boxes) for convenient reference. A B H E C D F I K G J L (a) A H B E C D G K F J I L (b) Figure 7.14 According to PREORDER, applying it to T, we will visit the root and print A, then search subtree 1, and then subtree 7. Applying PREORDER to subtree 1 results in visiting the root of subtree 1 and printing B, then searching subtree 2, and finally searching subtree 4. The search of subtree 2 first prints the symbol C and 282 Chapter 7 Trees then searches subtree 3. Subtree 3 has just one vertex, and so, as previously mentioned, a search of this tree yields just the symbol D. Up to this point, the search has yielded the string ABCD. Note that we have had to interrupt the search of each tree (except subtree 3, which is a leaf of T) in order to apply the search procedure to a subtree. Thus we cannot finish the search of T by searching subtree 7 until we apply the search procedures to subtrees 2 and 4. We could not complete the search of subtree 2 until we search subtree 3, and so on. The bookkeeping brought about by these interruptions produces the labels in the desired order, and recursion is a simple way to specify this bookkeeping. Returning to the search, we have completed searching subtree 2, and we now must search subtree 4, since this is the right subtree of tree 1. Thus we print E and search 5 and 6 in order. These searches produce F and G. The search of subtree 1 is now complete, and we go to subtree 7. Applying the same procedure, we can see that the search of subtree 7 will ultimately produce the string HIJKL. The result, ◆ then, of the complete search of T is to print the string ABCDEFGHIJKL. Example 2 Consider the completely parenthesized expression (a − b) × (c + (d ÷ e)). Figure 7.15(a) shows the digraph of the labeled, positional binary tree representation of this expression. We apply the search procedure PREORDER to this tree, as we did to the tree in Example 1. Figure 7.15(b) shows the various subtrees encountered in the search. Proceeding as in Example 1 and supposing again that visiting v simply prints out the label of v, we see that the string × − a b + c ÷ d e is the result of the search. This is the prefix or Polish form of the given algebraic expression. Once again, the numbering of the boxes in Figure 7.15(b) shows the order in which the ◆ algorithm PREORDER is applied to subtrees. × − + a b c ÷ d e (a) × − + a b c ÷ d e (b) Figure 7.15 The Polish form of an algebraic expression is interesting because it represents the expression unambiguously, without the need for parentheses. To evaluate an expression in Polish form, proceed as follows. Move from left to right until we find a string of the form Fxy, where F is the symbol for a binary operation (+, −, ×, and so on) and x and y are numbers. Evaluate x F y and substitute the answer for the string Fxy. Continue this procedure until only one number remains. For example, in the preceding expression, suppose that a = 6, b = 4, c = 5, d = 2, and e = 2. Then we are to evaluate × − 6 4 + 5 ÷ 2 2. This is done in the following sequence of steps. 7.3 Tree Searching 283 1. × − 6 4 + 5 ÷ 2 2 2. × 2 + 5 ÷ 2 2 since the first string of the correct type is − 6 4 and 6 − 4 = 2 3. × 2 + 5 1 replacing ÷ 2 2 by 2 ÷ 2 or 1 4. × 2 6 replacing + 5 1 by 5 + 1 or 6 5. 12 replacing × 2 6 by 2 × 6 This example is one of the primary reasons for calling this type of search the preorder search, because here the operation symbol precedes the arguments. Consider now the following informal descriptions of two other procedures for searching a positional binary tree T with root v. Algorithm INORDER Step 1 Search the left subtree (T(vL), vL), if it exists. Step 2 Visit the root, v. Step 3 Search the right subtree (T(vR), vR), if it exists. ● Algorithm POSTORDER Step 1 Search the left subtree (T(vL), vL), if it exists. Step 2 Search the right subtree (T(vR), vR), if it exists. Step 3 Visit the root, v. ● As indicated by the naming of the algorithms, these searches are called, respectively, the inorder and postorder searches. The names indicate when the root of the (sub)tree is visited relative to when the left and right subtrees are searched. Informally, in a preorder search, the order is root, left, right; for an inorder search, it is left, root, right; and for a postorder search, it is left, right, root. Example 3 Consider the tree of Figure 7.14(b) and apply the algorithm INORDER to search it. First we must search subtree 1. This requires us to first search subtree 2, and this in turn requires us to search subtree 3. As before, a search of a tree with only one vertex simply prints the label of the vertex. Thus D is the first symbol printed. The search of subtree 2 continues by printing C and then stops, since there is no right subtree at C. We then visit the root of subtree 1 and print B, and then proceed to the search of subtree 4, which yields F, E, and G, in that order. We then visit the root of T and print A and proceed to search subtree 7. The reader may complete the analysis of the search of subtree 7 to show that the subtree yields the string IJHKL. Thus the complete search yields the string DCBFEGAIJHKL. Suppose now that we apply algorithm POSTORDER to search the same tree. Again, the search of a tree with just one vertex will yield the label of that vertex. In general, we must search both the left and the right subtrees of a tree with root v before we print out the label at v. Referring again to Figure 7.14(b), we see that both subtree 1 and subtree 7 must be searched before A is printed. Subtrees 2 and 4 must be searched before B is printed, and so on. The search of subtree 2 requires us to search subtree 3, and D is the first symbol printed. The search of subtree 2 continues by printing C. We now search subtree 4, yielding F, G, and E. We next visit the root of subtree 1 and print B. Then we proceed with the search of subtree 7 and print the symbols J, I, L, K, and H. Finally, we visit the root of T and print A. Thus we print out the string DCFGEBJILKHA. ◆ 284 Chapter 7 Trees Example 4 Let us now apply the inorder and postorder searches to the algebraic expression tree of Example 2 [see Figure 7.15(a)]. The use of INORDER produces the string a − b × c + d ÷ e. Notice that this is exactly the expression that we began with in Example 2, with all parentheses removed. Since the algebraic symbols lie between their arguments, this is often called the infix notation, and this explains the name INORDER. The expression given is ambiguous without parentheses. It could have come from the expression a − (b × ((c + d) ÷ e)), which would have produced a different tree. Thus the tree cannot be recovered from the output of search procedure INORDER, while it can be shown that the tree is recoverable from the Polish form produced by PREORDER. For this reason, Polish notation is often better for computer applications, although infix form is more familiar to human beings. The use of search procedure POSTORDER on this tree produces the string a b − c d e ÷ + ×. This is the postfix or reverse Polish form of the expression. It is evaluated in a manner similar to that used for Polish form, except that the operator symbol is after its arguments rather than before them. If a = 2, b = 1, c = 3, d = 4, and e = 2, the preceding expression is evaluated in the following sequence of steps. 1. 2 1 − 3 4 2 ÷ + × 2. 1 3 4 2 ÷ + × replacing 2 1 − by 2 − 1 or 1 3. 1 3 2 + × replacing 4 2 ÷ by 4 ÷ 2 or 2 4. 1 5 × replacing 3 2 + by 3 + 2 or 5 5. 5 replacing 1 5 × by 1 × 5 or 5 ◆ Reverse Polish form is also parentheses free, and from it we can recover the tree of the expression. It is used even more frequently than the Polish form and is the method of evaluating expressions in some calculators. Searching General Trees Until now, we have only shown how to search binary positional trees. We now show that any (unordered) tree T may be represented as a binary positional tree, which then can be searched by the preceding methods. If T is a tree with n vertices, we construct the binary positional tree B(T) that implements T. The vertices of B(T) are the same as the vertices of T, and we now define a binary positional tree B(T). Define a binary positional tree B(T) whose vertices will be the same as the vertices of T. If v is a vertex in B(T), then the left offspring of v is the first offspring of v in T (if it exists), and the right offspring of v in B(T) is the next sibling of v in T (if it exists) in the given order of siblings in T. Example 5 Figure 7.16(a) shows the digraph of a labeled tree T. We assume that each set of siblings is ordered from left to right, as they are drawn. Thus the offspring of vertex 1, that is, vertices 2, 3, and 4, are ordered with vertex 2 first, 3 second, and 4 third. Similarly, the first offspring of vertex 5 is vertex 11, the second is vertex 12, and the third is vertex 13. In Figure 7.16(b), we show the digraph of the corresponding binary positional tree, B(T). To obtain Figure 7.16(b), we simply draw a left edge from each vertex v to its first offspring (if v has offspring). Then we draw a right edge from each vertex v to its next sibling (in the order given), if v has a next sibling. Thus the left edge from vertex 2, in Figure 7.16(b), goes to vertex 5, because vertex 5 is the first offspring of vertex 2 in the tree T. Also, the right edge from vertex 2, in Figure 7.16(b), goes to vertex 3, since vertex 3 is the next sibling in line (among all offspring of vertex 1). A doubly-linked-list representation of B(T) is sometimes simply referred to as a linked-list representation of T. ◆ Example 6 Figure 7.17(a) shows the digraph of another labeled tree, with siblings ordered from left to right, as indicated. Figure 7.17(b) shows the digraph of the corresponding tree B(T), and Figure 7.17(c) gives an array representation of B(T). As mentioned, the data in Figure 7.17(c) would be called a linked-list representation of T. ◆ a b c d e f g h i j k l (a) ... Figure 7.17 Pseudocode Versions The three search algorithms in this section have straightforward pseudocode versions, which we present here. In each, we assume that the subroutine VISIT has been previously defined. SUBROUTINE PREORDER(T, v) 1. CALL VISIT(v) 2. IF (vL exists) THEN a. CALL PREORDER(T(vL), vL) 3. IF (vR exists) THEN a. CALL PREORDER(T(vR), vR) 4. RETURN END OF SUBROUTINE PREORDER SUBROUTINE INORDER(T, v) 1. IF (vL exists) THEN a. CALL INORDER(T(vL), vL) 2. CALL VISIT(v) 3. IF (vR exists) THEN a. CALL INORDER(T(vR), vR) 4. RETURN END OF SUBROUTINE INORDER SUBROUTINE POSTORDER(T, v) 1. IF (vL exists) THEN a. CALL POSTORDER(T(vL), vL) 2. IF (vR exists) THEN a. CALL POSTORDER(T(vR), vR) 3. CALL VISIT(v) 4. RETURN END OF SUBROUTINE POSTORDER 7.3 Exercises In Exercises 1 through 5 (Figures 7.18 through 7.22), the digraphs of labeled, positional binary trees are shown. In each case we suppose that visiting a node results in printing out the label of that node. For each exercise, show the result of performing a preorder search of the tree whose digraph is shown. 6 through 10 ... For Exercises 6 through 10, show the result of performing an inorder search of the tree shown in Figure 7.18 ... For Exercises 11 through 15, show the result of performing a postorder search of the tree shown in the specified figure. 16. Show the result of performing a preorder search of the tree shown in Figure 7.23. 17. (a) Every binary tree whose preorder search produces the string JBACDIHEFG must have 10 vertices. What else do the trees have in common? (b) Every binary tree whose postorder search produces the string SEARCHING must have 9 vertices. What else do the trees have in common? 20. Show the result of performing a postorder search of the tree shown in Figure 7.23. In Exercises 18 and 19, evaluate the expression, which is given in Polish, or prefix, notation. 18. × − + 3 4 − 7 2 ÷ 12 × 3 − 6 4 19. ÷ − × 3 x × 4 y + 15 × 2 − 6 y, where x is 2 and y is 3. In Exercises 20 through 24, evaluate the expression, which is given in reverse Polish, or postfix, notation. 20. 3 x y + × ... 21. ... 24. The string of a postorder search of a binary tree is SEARCHING, and inorder search is ... 25. Draw a binary tree whose preorder search produces the string CATSANDDOGS. 26. Draw a binary tree whose inorder search produces the string SEARCHING. 27. Draw a binary tree whose postorder search produces the string TREEHOUSE. INDEX 1 2 3 4 5 6 7 8 9 10 11 12 LEFT 12 0 3 4 6 0 0 0 10 ... DATA H T W O B R A N C H E S RIGHT ... Figure 7.25 For Exercises 28 and 29, use the tree of Exercise 26 ... 28. Draw the digraph of the labeled binary tree ... 33. Let A be the binary tree whose digraph is shown in Figure 7.23. Construct the digraph of the positional binary tree B(T) ... 35. In Exercises 35 and 36 (Figures 7.26 and 7.27), consider the digraph of the labeled binary positional tree shown. If this tree is the binary form B(T) of some tree T, construct the digraph of that tree T. 37. Write pseudocode for an algorithm that evaluates... 38. ... Exercise 327 39. Write a recurrence relation for AVLn, the minimum number of vertices needed for an AVL tree of height n. Justify your recurrence relation by giving a procedure for forming an AVL tree of height k with a minimum number of vertices from smaller AVL trees with minimum numbers of vertices. t h a Figure 7.26 7.4 Undirected Trees An undirected tree is simply the symmetric closure of a tree (see Section 4.7); that is, it is the relation that results from a tree when all the edges are made bidirectional. As is the custom with symmetric relations, we represent an undirected tree by its graph, rather than by its digraph. The graph of an undirected tree T will have a single line without arrows connecting vertices a and b whenever (a, b) and (b, a) belong to T. The set {a, b}, where (a, b) and (b, a) are in T, is called an undirected edge of T (see Section 4.4). In this case, the vertices a and b are called adjacent vertices. Thus each undirected edge {a, b} corresponds to two ordinary edges, (a, b) and (b, a). The lines in the graph of an undirected tree T correspond to the undirected edges in T. Example 1 Figure 7.28(a) shows the graph of an undirected tree T. In Figures 7.28(b) and (c), we show digraphs of ordinary trees T1 and T2, respectively, which have T as symmetric closure. This shows that an undirected tree will, in general, correspond to many directed trees. Labels are included to show the correspondence of underlying vertices in the three relations. Note that the graph of T in Figure 7.28(a) has six lines (undirected edges), although the relation T contains 12 pairs. We want to present some useful alternative definitions of an undirected tree, and to do so we must make a few remarks about symmetric relations. 7.4 Undirected Trees 289 b b d a c c d e f e f d a a g (a) c b e f g (b) (c) Figure 7.28 Let R be a symmetric relation, and let p : v1, v2, . . . , vn be a path in R. We will say that p is simple if no two edges of p correspond to the same undirected edge. If, in addition, v1 equals vn (so that p is a cycle), we will call p a simple cycle. Example 2 a f e b c d Figure 7.29 THEOREM 1 v4 v3 v2 v5 v1 v6 v7 vi vn (a) v0 p g-p-1 a b (c) Figure 7.30 Figure 7.29 shows the graph of a symmetric relation R. The path a, b, c, e, d is simple, but the path f, e, d, c, d, a is not simple, since d, c and c, d correspond to the same undirected edge. Also, f, e, a, b, a, f and d, a, b, a are simple cycles, but f, e, d, c, e, f is not a simple cycle, since f, e and e, f correspond to the same undirected edge. ◆ We will say that a symmetric relation R is acyclic if it contains no simple cycles. Recall (see Section 4.4) that a symmetric relation R is connected if there is a path in R from any vertex to any other vertex. The following theorem provides a useful statement equivalent to the previous definition of an undirected tree. Let R be a symmetric relation on a set A. Then the following statements are equivalent. (a) R is an undirected tree. (b) R is connected and acyclic. Proof We will prove that part (a) implies part (b), and we will omit the proof that part (b) implies part (a). We suppose that R is an undirected tree, which means that R is the symmetric closure of some tree T on A. Note first that if (a, b) ∈ R, we must have either (a, b) ∈ T or (b, a) ∈ T. In geometric terms, this means that every undirected edge in the graph of R, drawn without arrows, appears in the digraph of T, directed one way or the other. We will show by contradiction that R has no simple cycles. Suppose that R has a simple cycle p : v1, v2, . . . , vn, v1. For each edge (vi, vj) in p, choose whichever pair (vi, vj) or (vj, vi) is in T. The result is a closed figure with edges in T, where each edge may be pointing in either direction. Now there are three possibilities. Either all arrows point clockwise, as in Figure 7.30(a), all point counterclockwise, or some pair must be as in Figure 7.30(b). Figure 7.30(b) is impossible, since in a tree T every vertex (except the root) has in-degree 1 (see Theorem 1 of Section 7.1). But either of the other two cases would mean that T contains a cycle, which is also impossible. Thus the existence of the cycle p in R leads to a contradiction and so is impossible. We must also show that R is connected. Let v0 be the root of the tree T. Then, (d) if a and b are any vertices in A, there must be paths p from v0 to a and q from v0 to b, as shown in Figure 7.30(c). Now all paths in T are reversible in R, so the path 290 Chapter 7 Trees q · p⁻¹, shown in Figure 7.30(d), connects a with b in R, where p⁻¹ is the reverse path of p. Since a and b are arbitrary, R is connected, and part (b) is proved. ■ There are other useful characterizations of undirected trees. We state two of these without proof in the following theorem. THEOREM 2 Let R be a symmetric relation on a set A. Then R is an undirected tree if and only if either of the following statements is true. (a) R is acyclic, and if any undirected edge is added to R, the new relation will not be acyclic. (b) R is connected, and if any undirected edge is removed from R, the new relation will not be connected. ■ Note that Theorem 2 and 1 tell us that an undirected tree must have exactly the "right" number of edges; one too many and a cycle will be created; one too few and the tree will become disconnected. The following theorem will be useful in finding certain types of trees. THEOREM 3 A tree with n vertices has n − 1 edges. Proof Because a tree is connected, there must be at least n − 1 edges to connect the n vertices. Suppose that there are more than n − 1 edges. Then either the root has indegree 1 or some other vertex has in-degree at least 2. But by Theorem 1, Section 7.1, this is impossible. Thus there are exactly n − 1 edges. ■ Spanning Trees of Connected Relations If R is a symmetric, connected relation on a set A, we say that a tree T on A is a spanning tree for R if T is a tree with exactly the same vertices as R and which can be obtained from R by deleting some edges of R. Example 3 The symmetric relation R whose graph is shown in Figure 7.31(a) has the tree T′, whose digraph is shown in Figure 7.31(b), as a spanning tree. Also, the tree T″, whose digraph is shown in Figure 7.31(c), is a spanning tree for R. Since R, T′, and T″ are all relations on the same set A, we have labeled the vertices to show the correspondence of elements. As this example illustrates, spanning trees are not unique. ◆ a b c d e a b c d e a b c d e (a) (b) (c) Figure 7.31 An undirected spanning tree for a symmetric connected relation R is useful in some applications. This is just the symmetric closure of a spanning tree. Figure 7.31(d) shows an undirected spanning tree for R that is derived from the spanning tree of Figure 7.31(c). If R is a complicated relation that is symmetric and connected, it might be difficult to devise a scheme for searching R, that is, for visiting each of its vertices once in some systematic manner. If R is reduced to a spanning tree, the searching algorithms discussed in Section 7.3 can be used. Theorem 2(b) suggests an algorithm for finding an undirected spanning tree for a relation R. Simply remove undirected edges from R until we reach a point where removal of one more undirected edge will result in a relation that is not connected. The result will be an undirected spanning tree. Example 4 In Figure 7.32(a), we repeat the graph of Figure 7.31(a). We then show the result of successive removal of undirected edges, culminating in Figure 7.32(f), the undirected spanning tree, which agrees with Figure 7.31(d). ◆ a a b c d e b c a d e b c d e f f f (a) (b) (c) a a b c d e b c d e (d) b c d f (e) e b c d e f (f) Figure 7.32 This algorithm is fine for small relations whose graphs are easily drawn. For large relations, perhaps stored in a computer, it is inefficient because at each stage we must check for connectedness, and this in itself requires a complicated algorithm. We now introduce a more efficient method, which also yields a spanning tree, rather than an undirected spanning tree. Let R be a relation on a set A, and let a, b ∈ A. Let A0 = A − {a, b}, and A′ = A0 ∪ {a′}, where a′ is some new element not in A. Define a relation R on A as follows. Suppose u, v ∈ A, u ≠ a, v ≠ a. Let (a′, u) ∈ R if and only if (a, u) ∈ R or (b, u) ∈ R. Let (u, a′) ∈ R if and only if (u, a) ∈ R or (u, b) ∈ R. Finally, let (u, v) ∈ R if and only if (u, v) ∈ R. We say that R is a result of merging the vertices a and b. Imagine, in the digraph of R, that the vertices are pins, and the edges are elastic bands that can be shrunk to zero length. Now physically move pins a and b together, shrinking the edge between them, if there is one, to zero length. The resulting digraph is the digraph of R. If R is symmetric, we may perform this operation on the graph of R. The result is the graph of the symmetric relation R. Example 5 Figure 7.33(a) shows the graph of a symmetric relation R. In Figure 7.33(b), we show the result of merging vertices v0 and v1 into a new vertex v0. In Figure 7.33(c), we show the result of merging vertices v0 and v2 into a new vertex v0. Notice in Figure 7.33(b) that the undirected edges that were previously present between v0 and v5 and between v1 and v5 have been combined into one undirected edge. 292 Chapter 7 Trees v0 v0 v1 v2 v3 v4 v5 v0 v3 v4 v5 v6 v3 v4 v5 v6 (a) v6 (c) Figure 7.33 The algebraic form of this merging process is also very important. Let us restrict our attention to symmetric relations and their graphs. We know from Section 4.2 how to construct the matrix of a relation R. If R is a relation on A, we will temporarily refer to elements of A as vertices of R. This will facilitate the discussion. Suppose now that vertices a and b of a relation R are merged into a new vertex a′ that replaces a and b to obtain the relation R. To determine the matrix of R, we proceed as follows. Step 1 Let row i represent vertex a and row j represent vertex b. Replace row i by the join of rows i and j. The join of two n-tuples of 0's and 1's has a 1 in some position exactly when either of those two n-tuples has a 1 in that position. Step 2 Replace column i by the join of columns i and j. Step 3 Restore the main diagonal to its original values in R. Step 4 Delete row j and column j. We make the following observation regarding Step 3. If e = (a, b) or (b, a), then after merging a and b, e becomes a cycle of length 1 at a. We do not want to create this situation, since it does not correspond to "shrinking (a, b) to zero." Step 3 corrects for this occurrence. Example 6 v0 v1 v2 v3 v4 v5 v6 v0 [1 1] [0 0] [1 1] [0 0] [1 1] [0 0] [1 1] [0 0] [1 1] [0 0] [1 1] [0 0] [1 1] [1 1] [1 1] 0 1 0 0 0 1 0 v1 v2 v3 v4 v5 v6 v1 v2 v3 v4 v5 v6 ... Figure 7.34 gives the matrices for the corresponding symmetric relations whose graphs are given in Figure 7.33. In Figure 7.34(b), we have merged vertices v0 and v1 into v0. Note that this is done by taking the join of the first two rows and entering the result in row 1, doing the same for the columns, then restoring the diagonal, and removing row 2 and column 2. If vertices v0 and v2 in the graph whose matrix is given by Figure 7.34(b) are merged, the resulting matrix has the ◆ matrix given by Figure 7.34(c). Figure 7.34 ... We can now give an algorithm for finding an undirected spanning tree for a symmetric, connected relation R on the set A = {v1, v2, . . . , vn}. The method is equivalent to a special case of an algorithm called Prim's algorithm. The steps are as follows: Step 1 Choose a vertex v1 of R, and arrange the matrix of R so that the first row corresponds to v1. Step 2 Choose a vertex v2 of R adjacent to v1, merge v1 and v2 into a new vertex v1′, representing {v1, v2}, and replace v1 by v1′. Compute the matrix of the resulting relation R. Call the vertex v1′ a merged vertex. Step 3 Repeat Steps 1 and 2 on R and on all subsequent relations until a relation with a single vertex is obtained. At each stage, keep a record of the set of original vertices that is represented by each merged vertex. Step 4 Construct the spanning tree as follows. At each stage, when merging vertices a and b, select an edge in R from one of the original vertices represented by a to one of the original vertices represented by b. Example 7 We apply Prim's algorithm to the symmetric relation whose graph is shown in Figure 7.35. In Table 7.1, we show the matrices that are obtained when the original set of vertices is reduced by merging until a single vertex is obtained, and at each stage we keep track of the set of original vertices represented by each merged vertex, as well as of the new vertex that is about to be merged. TABLE 7.1 Matrix a b a [a b 0 [b c] 0 [d] 1 a b [d] 1 d c Figure 7.35 a d a d a b c Figure 7.36 [b 0 0 1 1 0 0] [1 1 0] [0 1 0] 1 [1 1 0 0] 0 0 b 1 0 1 d 0 1 1 0 0 d 1 1 0] 1 0 Original Vertices Represented by Merged Vertices New Vertex to Be Merged (with First Row) — a ← {a, c} — b {a, c, b} d a ← c b, c, b d — The first unknown chosen is a, and we choose c as the vertex to be merged with a, since there is a 1 at vertex c in row 1. We also select the edge (a, c) from the original graph. At the second stage, there is a 1 at vertex b in row 1, so we merge b with vertex a. Again, we need an edge in R from a vertex of {a, c} to b, say (a, b). Finally, we must merge d with {a, c, b}. We select (a, d) say. The selected edges (a, c), (c, b), and (a, d) form the spanning tree for R, which is shown in Figure 7.36. Note that the first vertex selected becomes the root of the spanning tree that is constructed. 294 Chapter 7 Trees 7.4 Exercises In Exercises 1 through 6, construct an undirected spanning tree for the connected symmetric relation whose graph is given. 1. a b 2. a b c d f g 3. a b c d e f g 4. a b c d e e f e g Exercises 7 through 12. Figures 7.37 through 7.42. Figure 7.37 (a) f g Figure 7.38 f e d g h Figure 7.39 d e f g h Figure 7.40 c d e 5. a b c d e f g 6. Exercises ... use Prim's algorithm to construct a spanning tree for the connected graph shown. Use the indicated vertex as the root of the tree and draw the digraph of the spanning tree produced. 7. Figure 7.37; a 8. Figure 7.38; b 9. Figure 7.39; c 10. Figure 7.40; use e as the root. Figure 7.41; use a as the root. 12. Figure 7.42; use d as the root. Each connected symmetric relation has a unique undirected spanning tree. 3. Let G be the graph shown in Figure 7.39. a a f a d g c f ... Figure 7.41 Figure 7.42 14. For the connected graphs shown in Figure 7.44, show the graphs of all nonisomorphic undirected spanning trees. 13. Figure 7.39 14. Figure 7.40. 1 2 5 3 1 4 2 6 1 2 3 3 7 4 Figure 7.40 5. Let G be the graph shown in Figure 7.41. a b d e Figure 7.41 c 4 5 Figure 7.44 Figure 7.45 15. For the undirected tree shown in Figure 7.45, show the digraphs of all spanning trees. How many are there? Two trees, T1 and T2, are isomorphic if they are isomorphic posets; that is, there is a one-to-one correspondence, f, between the vertices of T1 and T2 and (a, b) is an edge in T1 if and only if (f(a), f(b)) is an edge in T2. 16. For the graph shown in Figure 7.44, show the graphs of all nonisomorphic undirected spanning trees. 18. For the graph in Figure 7.46, give all spanning trees. A1 A1 A5 19. 20. 21. 22. A2 A3 A2 A3 A3 A5 Figure 7.47 Figure 7.48 A4 A4 Figure 7.49 23. How many different spanning trees are there? Figure 7.46 Figure 7.47 For the graph in Figure 7.47, show the graphs of all nonisomorphic spanning trees. For the graph in Figure 7.48, how many different spanning trees are there? For the graph shown in Figure 7.48, show the graphs of all nonisomorphic spanning trees. 7.5 A1 A2 A4 295 24. State your conclusion for Figure 7.49 as a theorem and prove it. 25. For the graph shown in Figure 7.49, how many nonisomorphic spanning trees are there? Justify your answer. 26. Prove that a symmetric connected relation has an undirected spanning tree. Minimal Spanning Trees In many applications of symmetric connected relations, the (undirected) graph of the relation models a situation in which the edges as well as the vertices carry information. A weighted graph is a graph for which each edge is labeled with a numerical value called its weight. Example 1 The small town of Social Circle maintains a system of walking trails between the recreational areas in town. The system is modeled by the weighted graph in Figure 7.50, where the weights represent the distances in kilometers between sites. ◆ Example 2 A communications company is investigating the costs of upgrading links between the relay stations it owns. The weighted graph in Figure 7.51 shows the stations and the cost in millions of dollars for upgrading each link. ◆ The weight of an edge (vi, vj) is sometimes referred to as the distance between vertices vi and vj. A vertex u is a nearest neighbor of vertex v if u and B 2 6 2.7 3 3 4 2 5 C F 6 3 4 2 9 H 1.8 I D 2.8 J 1.7 3 2 G F Figure 7.51 2.5 5.5 3 E 2.2 3.3 3 H 4 5 Figure 7.50 2 4 2 4 3 4 D 6 C 3 6 B 2 1 4 A 2 I F 296 Chapter 7 Trees v are adjacent and no other vertex is joined to v by an edge of lesser weight than (u, v). Notice that, ungrammatically, v may have more than one nearest neighbor. Example 3 In the graph shown in Figure 7.50, vertex C is a nearest neighbor of vertex A. Vertices E and G are both nearest neighbors of vertex F. ◆ A vertex v is a nearest neighbor of a set of vertices V = {v1, v2, . . . , vk} in a graph if v is adjacent to some member vi of V and no other vertex adjacent to a member of V is joined by an edge of lesser weight than (v, vi). This vertex v may belong to V. Example 4 Referring to the graph given in Figure 7.51, let V = {C, E, J}. Then vertex D is a nearest neighbor of V, because (D, E) has weight 2.2 and no other vertex adjacent to C, E, or J is linked by an edge of lesser weight to one of these vertices. ◆ With applications of weighted graphs, it is often necessary to find an undirected spanning tree for which the total weight of the edges in the tree is as small as possible. Such a spanning tree is called a minimal spanning tree. Prim's algorithm (Section 7.4) can easily be adapted to produce a minimal spanning tree for a weighted graph. We restate Prim's algorithm as it would be applied to a symmetric, connected relation given by its undirected weighted graph. Algorithm PRIM'S ALGORITHM Let R be a symmetric, connected relation with n vertices. Step 1 Choose a vertex v1 of R. Let V = {v1} and E = { }. Step 2 Choose a nearest neighbor vi of V that is adjacent to vj, vj ∈ V, and for which the edge (vi, vj) does not form a cycle with members of E. Add vi to V and add (vi, vj) to E. Step 3 Repeat Step 2 until |E| = n − 1. Then V contains all n vertices of R, and E contains the edges of a minimal spanning tree for R. ● In this version of Prim's algorithm, we begin at any vertex of R and construct a minimal spanning tree by adding an edge to a nearest neighbor of the set of vertices already linked, as long as adding this edge does not complete a cycle. This is an example of a greedy algorithm. At each stage we chose what is "best" based on local conditions, rather than looking at the global situation. Greedy algorithms do not always produce optimal solutions, but we can show that for Prim's algorithm the solution is optimal. THEOREM 1 Prim's algorithm, as given above, produces a minimal spanning tree for the relation. Proof Let R have n vertices. Let T be the spanning tree for R produced by Prim's algorithm. Suppose that the edges of T, in the order in which they were selected, are t1, t2, . . . , tn−1. For each i from 1 to n − 1, we define Ti to be the tree with edges t1, t2, . . . , ti and T0 = { }. Then T0 ⊂ T1 ⊂ · · · ⊂ Tn−1 = T. We now prove, by mathematical induction, that each Ti is contained in a minimal spanning tree for R. Basis Step Clearly P(0): T0 = { } is contained in every minimal spanning tree for R. 7.5 Minimal Spanning Trees 297 Induction Step Let P(k): Tk is contained in a minimal spanning tree T for R. We use P(k) to show P(k + 1): Tk+1 is contained in a minimal spanning tree for R. By definition we have {t1, t2, . . . , tk} ⊆ T. If tk+1 also belongs to T, then Tk+1 ⊆ T and we have P(k + 1) is true. If tk+1 does not belong to T, then T ∪ {tk+1} must contain a cycle. (Why?) This cycle would be as shown in Figure 7.52 for some edges s1, s2, . . . , sr in T. Now the edges of this cycle cannot all be from Tk, or Tk+1 would contain this cycle. Let sl be the edge with smallest index l that is not in Tk. Then sl has one vertex in the tree Tk and one not in Tk. This means that when tk+1 was chosen by Prim's algorithm, sl was also available. Thus the weight of sl is at least as large as that of tk+1. The spanning tree (T − {sl}) ∪ {tk+1} contains Tk+1. The weight of this tree is less than or equal to the weight of T, so it is a minimal spanning tree for R. Thus, P(k + 1) is true. So Tn−1 = T is contained in a minimal spanning tree and must in fact be that minimal spanning tree. (Why?) ■ s2 ti+1 sr Figure 7.52 Example 5 Social Circle, the town in Example 1, plans to pave some of the walking trails to make them bicycle paths as well. As a first stage, the town wants to link all the recreational areas with bicycle paths as cheaply as possible. Assuming that construction costs are the same on all parts of the system, use Prim's algorithm to find a plan for the town's paving. Solution Referring to Figure 7.50, if we choose A as the first vertex, the nearest neighbor is C, 2 kilometers away. So (A, C) is the first edge selected. Considering the set of vertices {A, C}, B is the nearest neighbor, and we may choose either (A, B) or (B, C) as the next edge. Arbitrarily, we choose (B, C). B is a nearest neighbor of {A, B, C}, but the only edge available (A, B) would make a cycle, so we must move to a next nearest neighbor and choose (C, F) or (C, G). Following through (c) shows the beginning steps and Figure 7.53(d) shows a possible final result. Figure 7.53(e) shows a minimal spanning tree using Prim's algorithm beginning with vertex E. In either case, the bicycle paths would cover 21 kilometers. ◆ Example 6 A minimal spanning tree for the communication network in Example 2 may be found by using Prim's algorithm beginning at any vertex. Figure 7.54 shows a minimal spanning tree produced by beginning at I. The total cost of upgrading these links would be $20,200,000. ◆ B A 2 C 2 (a) H 5 F A 3 G (d) B 2 H A 1 5 2 C F 4 G H 3 E 2 6 G C 2 1 2 4 3 C 2 B A 2 Figure 7.53 2 5 E C 3 G (a) A B A B D 2.8 1.7 1.2 2.5 1.7 G Figure 7.54 2.2 F E 298 Chapter 7 Trees How is Prim's algorithm in Section 7.4 related to Prim's algorithm given in this section? The earlier version is a special case where the edges of the graph were not assigned weights; or, put another way, all edges are given weight 1. A matrix version might be more easily programmed. The key to the modification is to consider an entry in the matrix as representing not only the existence of an edge, but also the weight of the edge. Thus, we construct a matrix containing the weights of edges, with a 0 to show there is no edge between a pair of vertices as before. Figure 7.55 shows a connected weighted graph and the associated matrix of weights. A 3 B 3 5 D C A [A 0 B 3 C 3 D 0 B 3 C 3 D 0 5 5 0 2 0 5 2 0 3 0] Figure 7.55 Algorithm PRIM'S ALGORITHM (Matrix Version) Let R be a symmetric, connected relation with n vertices and M be the associated matrix of weights. Step 1 Choose the smallest entry in M, say mij. Let a be the vertex that is represented by column j. Step 2 Merge a with b as follows: Replace row i with ... Call the resulting matrix M. Step 3 Repeat Steps 1 and 2 on M and subsequent matrices until a single vertex is obtained. At each stage, keep a record of the set of original vertices that is represented by each merged vertex. Step 4 Construct the minimal spanning tree as follows. At each stage, when merging vertices a and b, select the edge represented by the minimal weight from one of the original vertices represented by a to one of the original vertices represented by b. ● Example 7 Minimal Spanning Trees 299 Apply the matrix version of Prim's algorithm to the graph and associated matrix in Figure 7.55. Solution Either 2 may be selected as m1 1. We choose m 3 4 and merge C and D. This produces [A A 0 m 3 B [4 C 3 A B 3 2 C Figure 7.56 ... with {C, D} and with edge (C, D). We repeat Steps 1 and 2 on m using m 1 3 = 3. This gives 3 D B A 0 3 A 0 B 3 ... with (A, C, D) and selecting edge (A, C). A final merge yields the ◆ edge (B, D). The minimal spanning tree is shown in Figure 7.56. If a symmetric connected relation R has n vertices, then Prim's algorithm has running time Θ(n²). (This can be improved somewhat.) If R has relatively few edges, a different algorithm may be more efficient. This is similar to the case for determining whether a relation is transitive, as seen in Section 4.6. Kruskal's algorithm is another example of a greedy algorithm that produces an optimal solution. Algorithm KRUSKAL'S ALGORITHM Let R be a symmetric, connected relation with n vertices and let S = {e1, e2, . . . , ek} be the set of weighted edges of R. Step 1 Choose an edge e1 in S of least weight. Let E = {e1}. Replace S with S − {e1}. Step 2 Select an edge ei in S of least weight that will not make a cycle with members of E. Replace E with E ∪ {ei} and S with S − {ei}. Step 3 Repeat Step 2 until |E| = n − 1. ● Since R has n vertices, the n − 1 edges in E will form a spanning tree. The selection process in Step 2 guarantees that this is a minimal spanning tree. (We omit the proof.) Roughly speaking, the running time of Kruskal's algorithm is Θ(k lg(k)), where k is the number of edges in R. Example 8 A minimal spanning tree from Kruskal's algorithm for the walking trails in Example 1 is given in Figure 7.57. One sequence of edge selections is (D, E), (D, H), (A, C), (A, B), (E, G), (B, F), and (C, E) for a total weight of 21 kilometers. Naturally either of the algorithms for minimal spanning trees should produce trees of the same weight. Example 9 Use Kruskal's algorithm to find a minimal spanning tree for the relation given by the graph in Figure 7.58. 300 Chapter 7 Trees 7 D 12 20 12 C F 15 15 20 A 3 2 4 C B 10 14 G E 12 F D B 10 H C 14 E 4 5 3 2 2 G Figure 7.58 Solution Initially, there are two edges of least weight, (B, C) and (E, F). Both of these are selected. Next there are three edges, (A, G), (B, G), and (D, E), of weight 12. All of these may be added without creating any cycles. Edge (F, G) of weight 14 is the remaining edge of least weight. Adding (F, G) gives us six edges for a 7-vertex graph, so a minimal spanning tree has been found. ◆ 7.5 Exercises In Exercises 1 through 6, use Prim's algorithm as given in this section to find a minimal spanning tree for the connected graph indicated. Use the specified vertex as the initial vertex. 1. Let G be the graph shown in Figure 7.61. Let G be the graph shown in Figure 7.60. Begin at E. 3. Let G be the graph shown in Figure 7.58. Begin at G. 4 4 2 5 3 A 5 4 G B 2 3 2 6 C 2 2 4 6 Figure 7.60. Begin at K. A 6 B H 2 2 3 5 G 3 7 3 5 2 F 3 2 2 6 I 2 J 7 C 4 3 2 3 2 D Figure 7.59 2. Figure 7.60 2. N M I 1 7 In Exercises 10 through 12, use Kruskal's algorithm to find a minimal spanning tree for the indicated graph. 10. Let G be the graph shown in Figure 7.51. 2 3 4 9 2 7 10 2 Figure 7.61 9. Let G be the graph shown in Figure 7.59. Begin at F. 5 8 2 9 2 C 4 3 8 11. Let G be the graph shown in Figure 7.59. 12. Let G be the graph shown in Figure 7.60. 13. The distances between eight cities are given in the following table. Use Kruskal's algorithm to find a minimal spanning tree whose vertices are these cities. What is the total distance for the tree? 7.5 Minimal Spanning Trees 301 Albany Annandale Atlanta Augusta Weston 121 120 113 10 135 63 Allen Allendale Allen Allendale Anderson Abbeville Athens Atlanta Augusta 69 121 52 30 97 149 119 172 92 29 97 160 63 155 135 163 206 122 204 66 63 16 59 91 174 101 147 52 97 29 174 101 147 52 97 29 174 101 147 52 97 29 174 101 147 52 97 29 174 101 147 52 97 29 174 101 147 52 97 29 174 101 147 52 97 29 174 101 147 52 97 29 117 161 16 149 222 160 206 59 16. Modify Prim's algorithm to handle the case of finding a maximal spanning tree if it is given that the spanning tree is required to have a maximal total weight. 17. Use the modification of Prim's algorithm developed in Exercise 16 on the graph shown in Figure 7.60 if the edge from D to E must be included in the tree. 17. Use the modification of Prim's algorithm developed in Exercise 16 on the graph shown in Figure 7.61 to find a maximal spanning tree. 18. Modify Kruskal's algorithm so that it will produce a maximal spanning tree, that is, one with the largest possible sum of the weights. 19. Suppose that the graph in Figure 7.60 represents possible flows through a system of pipes. Find a spanning tree that gives the maximum possible flow in this system. 20. Modify Prim's algorithm as developed in Exercise 16 on the graph shown in Figure 7.59. Redo Exercise 13 with the requirement that the route from Atlanta to Augusta must be included. How much longer does this make the tree? 18. Modify Kruskal's algorithm so that it will produce a maximal spanning tree, that is, one with the largest possible sum of the weights. 21. Use the modification of Prim's algorithm developed in Exercise 18 to find a maximal spanning tree for the graph shown in Figure 7.60. 22. Use the modified Prim's algorithm from Exercise 20 to find a maximal spanning tree for the graph in Figure 7.59. 23. Give a simple example of a weighted graph that has a unique maximal spanning tree. 21. Prove that the graph in Figure 7.61 has a unique minimal spanning tree. Give reasons for your answer. 23. Give a simple condition on the weights of a graph that will guarantee that there is a unique maximal spanning tree for the graph. 24. Prove or disprove that an edge of unique least weight in a connected graph must be included in any minimal spanning tree. 25. A greedy algorithm does not always produce an optimal solution. Consider the problem of selecting a specified amount of money from an unlimited supply of coins using as few coins as possible. A greedy algorithm would be to select as many of the largest denomination coins first, then as many of the next largest as possible, and so on until the specified amount has been selected. If the coins in Asia are the 1-sober, the 7-nobee, and the 11-nobee, construct an example to show that the greedy algorithm does not always give a minimal solution. 26. Expand the proof of Theorem 1 by completing the following. (a) Why must T ∪ {tk+1} contain a cycle? (b) Why is Tn−1 the minimal spanning tree, not just contained in a minimal spanning tree? (c) Prove that a tree having n vertices has n − 1 edges. because (T − {sl}) ∪ {tk+1} is a spanning tree for R, then it must be T. Why? Tips for Proofs Statements about trees; Other Theorems used in Section 7.5; use Theorem 1 of Section 7.1 and Theorem 2 of Section 7.4. 7.5 Exercises 303 Many of the proofs of Theorems in this chapter make use of the basis of many indirect proofs for statements about trees. (See Theorem 1(c), Section 7.1.) Counting arguments are also common in proofs about trees; for example, Theorem 3, Section 7.4. Because a tree, like a lattice, is a relation with certain properties, the various facts and representations for relations are available to create proofs for tree theorems. We see this in Theorem 2, Section 7.1. The development of Prim's algorithm in Section 7.4 uses the matrix representation of a relation; this representation can be used in a proof that the running time of the algorithm is Θ(n²). 302 Chapter 7 Trees Key Ideas for Review • Tree: relation on a finite set A such that there exists a vertex v0 ∈ A with the property that there is a unique path from v0 to any other vertex in A and no path from v0 to v0 • Root of tree: vertex v0 in the preceding definition Rooted tree (T, v0): tree T with root v0 Theorem: Let (T, v0) be a rooted tree. Then (a) There are no cycles in T. (b) v0 is the only root of T. (c) Each vertex in T, other than v0, has in-degree one, and v0 has in-degree zero. Level: see page 271 Height of a tree: the largest level number of a tree Leaves: vertices having no offspring Theorem: Let T be a rooted tree on a set A. Then (a) T is irreflexive. (b) T is asymmetric. (c) If (a, b) ∈ T and (b, c) ∈ T, then (a, c) ∉ T, for all a, b, and c in A. n-tree: tree in which every vertex has at most n offspring Complete n-tree: see page 273 Binary tree: 2-tree Theorem: If (T, v0) is a rooted tree and v ∈ T, then T(v) is also a rooted tree with root v. T(v): subtree of T beginning at v • Positional binary tree: see page 275 • Computer representation of trees: see page 276 Computer representation of trees: see page 279 Huffman code tree: see page 278 • Preorder search: see page 280 • Inorder search: see page 281 • Postorder search: see page 283 • Reverse Polish notation: see page 284 • Searching general trees: see page 284 • Linked-list representation of a tree: see page 285 • Undirected tree: symmetric closure of a tree • Simple path: No two edges correspond to the same undi- rected edge. • Connected symmetric relation R: There is a path in R from any vertex to any other vertex. • Theorem: A tree with n vertices has n − 1 edges. • Spanning tree for symmetric, connected relation R: tree reaching all the vertices of R and whose edges are edges of R Undirected spanning tree: symmetric closure of a spanning tree • Prim's algorithm: see page 293 Weighted graph: a graph whose edges are each labeled with a numerical value • Distance between vertices vi and vj : weight of (vi, vj) • Nearest neighbor of v: see page 296 • Minimal spanning tree: undirected spanning tree for which the total weight of the edges is as small as possible Prim's algorithm: see page 296 • Greedy algorithm: see page 296 Kruskal's algorithm: see page 299 Chapter 7 Self-Test v0 1. On this chapter, what kind of mathematical object is a tree? 2. What changes in performing a preorder, inorder, or postorder search? v3 3. What are the other three possible sequences for searching a binary tree? v1 v7 4. What advantages might a complete n-tree have over a general tree as a data structure? 5. What information would help you decide between using Prim's and Kruskal's algorithm to find a minimal spanning tree? Chapter 7 Trees 6. Determine if the relation R = {(1, 7), (2, 3), (4, 1), (2, 4), (2, 5), (5, 3), (6, 2)} is a tree on the set A = {1, 2, 3, 4, 5, 6, 7}. If it is not a tree, what is the root? If it is not a tree, make the least number of changes necessary to make it a tree and give the root. 7. Consider (T, v0), the tree whose digraph is given in Figure 7.62. (a) What is the height of T ? (b) List the leaves of T. v8 v6 v2 v4 v5 v9 v10 Figure 7.62(c) How many subtrees of T contain v4 ? (d) List the siblings of v7 . 8. Let (T, v0) be a rooted tree. Prove that if any edge is removed from T, then the resulting relation cannot be a tree. 9. (a) Construct the labeled tree representing the algebraic expression ((((x + 3)(x + 3) − (x + 2)(x + 2)) ÷ 6(x−3))+(x−8))) (b) Give the infix, prefix, and postfix forms of the expression. 10. The arrays LEFT, DATA, RIGHT give a doubly-linked-list representation of a labeled, positional binary tree. Construct the digraph of this tree. INDEX LEFT 1 2 3 4 5 6 7 8 8 0 2 4 0 0 6 9 0 0 0 0 DATA S T R U C T U R E w z RIGHT 0 5 0 0 7 9 0 3 10 0 11. Here, you wish to visit the vertices in the tree constructed in Problem 10 in the order of the word STRUCTURE. 12. Create a binary tree for which the results of performing a postorder search are the string TRE 1 S2 S3 T2 R and for which the results of performing a preorder search are SNT RE 1 BT S1 . Assume that to visit a vertex means to print the contents of the vertex. 13. Draw a complete 3-tree with height 3 using the smallest possible number of vertices. 14. The digraph of a labeled, binary positional tree is shown in Figure 7.63. Construct the digraph of the labeled ordered tree T such that T = B(T). y x r T Figure 7.63 15. Give an undirected spanning tree for the relation whose graph is given in Figure 1. K G A B E H F C D I J Figure 7.64 16. Use Prim's greedy algorithm to find a minimal spanning tree for the graph in Figure 7.65. Use vertex E as the initial vertex and list the edges in the order in which they are chosen. A 3 2 E F 6 G 5 3 C D 7 3 5 4 F B Figure 7.65 17. Use Kruskal's algorithm to find a minimal spanning tree for the graph in Figure 7.65. List the edges in the order in which they are chosen. Experiment 7 Ways to store and retrieve information in binary trees are presented in Sections 7.2 and 7.3. In this experiment you will investigate another type of tree that is frequently used for data storage. A B-tree of degree k is a tree with the following properties: 1. All leaves are on the same level. 2. If it is not a leaf, the root has at least two children and at most k children. 3 Figure 7.66 4. Any vertex that is not a leaf or the root has at least k/2 children and at most k children. 1 Figure 7.67 Here B-trees is a degree 3 B-tree. 304 Chapter 7 Trees Experiment 7 4. Find all the B-trees of degree 3 with height 2. 5. Give an example of a B-tree of degree 5 (but not less) with height 3. 3. Discuss the features of your examples in Questions 1 through 3 that suggest that a B-tree would be a good storage structure. Part II. The properties that define a B-tree of degree k do not restrict how the tree can look, but also limit the number of leaves for a given height and the number of leaves that T can have? Explain your reasoning. 2. If T is a B-tree of degree k and T has height h, what is the minimum number of leaves that T can have? Explain your reasoning. 3. If T is a B-tree of degree k and T has n leaves, what is the maximum height that T can have? Explain your reasoning. Part III. If T is a B-tree of degree k and T has n leaves, what is the minimum height that T can have? Explain your reasoning. 5. Explain how your results in Part II, Questions 3 and 4, support your conclusions in Part I, Question 4. CHAPTER 8 Topics in Graph Theory Prerequisites: Chapters 3 and 5 Graph theory begins with very simple geometric ideas and has many powerful applications. Some uses of graphs are discussed in Chapters 4, 6, and 7. In those chapters a graph is associated with a digraph. In this chapter, we give an alternate definition of graph that includes the more general multigraphs and is more appropriate for the applications developed in this chapter. Looking Back Graph theory began in 1736 when Leonhard Euler solved a problem that had been puzzling the good citizens of the town of K¨onigsberg in Prussia (now Kaliningrad in Russia). The river Pregel divides the town into four sections, and in Euler's day seven bridges connected these regions. The people wanted to know if it were possible to start at any location in town, cross every bridge exactly once, and return to the starting location. Euler showed that it is impossible to take such a walk. A problem in graph theory that attracted considerably more attention is the four-color map problem. Frank Guthrie, a former student of Augustus De Morgan, observed that he was able to color the map of England so that no two adjacent counties have the same color by using four different colors. He asked his brother, who in 1852 was a student of De Morgan, to ask De Morgan whether his conjecture that four colors will suffice to color every map so that no two adjacent countries have the same color was true. De Morgan publicized the problem, and for years many people worked on the problem. In 1879, Alfred Bray Kempe, a lawyer who had studied mathematics at Cambridge University, published a proof of the four-color conjecture that was highly acclaimed. Unfortunately, his proof had a fatal error. The theorem was finally proved by the American mathematicians Kenneth Appel and Wolfgang Haken at the University of Illinois. Their proof used as a basis the work of Kempe and over 1000 hours of computer time to check 1936 different map configurations. Their proof, the first of a major mathematical result by using a computer, was initially met with considerable skepticism. Some years later, the number of configurations was reduced, but to date there is no proof that does not involve significant use of a computer. (Wolfgang Haken (l) and Kenneth Appel (r) 305 306 Chapter 8 Topics in Graph Theory 8.1 Graphs A graph G consists of a finite set V of objects called vertices, a finite set E of objects called edges, and a function γ that assigns to each edge a subset {v, w}, where v and w are vertices (and may be the same). We will write G = (V, E, γ) when we need to name the parts of G. If e is an edge, and γ(e) = {v, w}, we say that e is an edge between v and w and that e is determined by v and w. The vertices v and w are called the end points of e. If there is only one edge between v and w, we often identify e with the set {v, w}. This should cause no confusion. The restriction that there is only a finite number of vertices may be dropped, but for the discussion here all graphs have a finite number of vertices. Example 1 Let V = {1, 2, 3, 4} and E = {e1, e2, e3, e4, e5}. Let γ be defined by γ(e1) = γ(e5) = {1, 2}, γ(e2) = {4, 3}, γ(e3) = {1, 3}, γ(e4) = {2, 4}. Then G = (V, E, γ) is a graph. 1 2 3 4 Figure 8.1 Graphs are usually represented by pictures, using a point for each vertex and a line for each edge. G in Example 1 is represented in Figure 8.1. We usually omit the names of the edges, since they have no intrinsic meaning. Also, we may want to put other more useful labels on the edges. We sometimes omit the labels on the vertices as well if the graphical information is adequate for the discussion. Graphs are often used to record information about relationships or connections. An edge between vi and vj indicates a connection between the objects vi and vj . In a pictorial representation of a graph, the connections are the most important information, and generally a number of different pictures may represent the same graph. Figures 8.2 and 8.3 also represent the graph G given in Example 1. 1 2 3 4 Figure 8.2 1 2 3 ◆ 4 Figure 8.3 The degree of a vertex is the number of edges having that vertex as an end point. A graph may contain an edge from a vertex to itself; such an edge is referred to as a loop. A loop contributes 2 to the degree of a vertex, since that vertex serves as both end points of the loop. Example 3 (a) In the graph in Figure 8.4, the vertex A has degree 2, vertex B has degree 4, and vertex D has degree 3. (b) In Figure 8.5, vertex a has degree 4, vertex e has degree 0, and vertex b has degree 2. (c) Each vertex of the graph in Figure 8.6 has degree 2. A vertex with degree 0 is called an isolated vertex. A pair of vertices that determine an edge are adjacent vertices. 8.1 Graphs 307 b a a B b C A D Figure 8.4 e c d Figure 8.5 Figure 8.6 In Figure 8.5, a and b are adjacent vertices; vertices a and d are not adjacent. ◆ A path π in a graph G consists of a pair (Vπ, Eπ) of sequences: a vertex sequence Vπ : v1, v2, . . . , vk and an edge sequence Eπ : e1, e2, . . . , ek−1 for which 1. Each successive pair vi, vi+1 of vertices is adjacent in G, and edge ei has vi and vi+1 as end points, for i = 1, . . . , k − 1. 2. No edge occurs more than once in the edge sequence. Thus we may begin at v1 and travel through the edges e1, e2, . . . , ek−1 to vk without using any edge twice. A circuit is a path that begins and ends at the same vertex. In Chapter 4 we called such paths cycles; the word "circuit" is more common in general graph theory. A path is called simple if no vertex appears more than once in the vertex sequence, except possibly if v1 = vk. In this case the path is called a simple circuit. This expanded definition of path is needed to handle paths in graphs that may have multiple edges between vertices. For the graph in Figure 8.4 we define a path π by sequences Vπ1 : A, B, D, C and Eπ1: e1, e3, e4, and the path π2 by the sequences Vπ2 : A, B, A and Eπ2 : e1, e2. ◆ The vertices alone would not be sufficient to define these paths. For π1 we would not know which edge to travel from A to B, e1 or e2, and for π2 we would not know which edge to use first and which to use second. The edges alone would not always be enough either, for if we only knew the edge sequence e1, e2 for π2, we would not know if the vertex sequence was A, B, A or B, A, B. If the vertices of a path have only one edge between each adjacent pair, then the edge sequence is completely determined by the vertex sequence. In this case we specify the path by the vertex sequence alone and write π : v1, v2, . . . , vk. Thus for the graph in Figure 8.2 we can write π3 : 1, 3, 4, 2, since this path contains no adjacent pair with multiple edges between them. In cases such as this, it is not necessary to label the edges. Example 5 (a) Paths π1 and π2 in the graph of Figure 8.4 were defined previously. Path π1 is not simple, since vertex D appears twice, but π2 is a simple circuit, since the only vertex appearing twice occurs at the beginning and at the end. (b) The path π4 : D, E, B, C in the graph of Figure 8.4 is simple. Examples of paths in the graph of Figure 8.5 are π5 : a, b, c, a and π6 : d, c, a, a. Here π5 is a simple circuit, but π6 is not simple. (d) In Figure 8.6 the path π7 : c, a, b, c, d in Figure 8.5 is not simple. 308 Chapter 8 Topics in Graph Theory A graph is called connected if there is a path from any vertex to any other vertex in the graph. Otherwise, the graph is disconnected. If the graph is disconnected, the various connected pieces are called the components of the graph. Example 6 The graphs in Figures 8.1 and 8.4 are connected. Those in Figures 8.5 and 8.6 are disconnected. The graph in Figure 8.6 has two components. ◆ Some important special families of graphs will be useful in our discussions. We present them here. U1 1. For each integer n ≥ 1, we let Un denote the graph with n vertices and no edges. Figure 8.7 shows U2 and U5 . We call Un the discrete graph on n vertices. 2. For each integer n ≥ 1, let Kn denote the graph with vertices {v1, v2, . . . , vn} and with an edge {vi, vj} for every i and j. In other words, every vertex in Kn is connected to every other vertex. In Figure 8.8 we show K3 , K4 , and K5 . The graph Kn is called the complete graph on n vertices. More generally, if each vertex of a graph has the same degree as every other vertex, the graph is called regular. The graphs Un are also regular. 3. For each integer n ≥ 1, we let Ln denote the graph with n vertices {v1, v2, . . . , vn} and with edges {vi, vi+1} for 1 ≤ i < n. We show L2 and L4 in Figure 8.9. We call Ln the linear graph on n vertices. 8.1 Graphs 309 Figure 8.7 K3 K4 K5 Figure 8.8 Example 7 L2 L4 Figure 8.9 Figure 8.11 All the Kn and Ln are connected, while the Un are disconnected for n > 1. In fact, the graph Un has exactly n components. Subgraphs and Quotient Graphs Suppose that G = (V, E, γ) is a graph. Choose a subset E1 of the edges in E and a subset V1 of the vertices in V, so that V1 contains (at least) all the end points of edges in E1. Then H = (V1, E1, γ1) is also a graph where γ1 is γ restricted to edges in E1. Such a graph H is called a subgraph of G. Subgraphs play an important role in analyzing graph properties. Example 8 The graphs shown in Figures 8.11, 8.12, and 8.13 are each a subgraph of the graph G shown in Figure 8.10. ◆ One of the most important subgraphs is the one that arises by deleting one edge and no vertices. If G = (V, E, γ) is a graph and e ∈ E, then we denote by Ge the graph obtained by omitting the edge e from E and keeping all vertices. Example 9 If G is the graph of Figure 8.10, and e is the edge labeled e3 , the graph of Figure 8.14 is Ge. ◆ b h f i h b g d c i f g d Figure 8.10 a c Figure 8.12 Figure 8.11 b i 309 ◆ a h Graphs g e b c Figure 8.13 Our second important construction is defined for graphs without multiple edges between the same vertices. Suppose that G = (V, E, γ) is a graph without multiple edges. The vertices of G are the equivalence classes of vertices in V defined by R (see Section 4.5). If [v] and [w] are the equivalence classes of vertices v and w of G, then there is an edge in G from [v] to [w] if and only if some vertex in [v] is connected to some vertex in [w] in the graph G. Informally, this just says that we get G R by merging all the vertices in each equivalence class into a single vertex and combining any edges that are superimposed by such a process. Example 9 Let G be the graph of Figure 8.15 (which has no multiple edges), and let R be the equivalence relation on V defined by the partition {{a, m, i}, {b, f, j}, {c, g, k}, {d, h, l}}. Then G R is shown in Figure 8.15. (b) Let G be the graph shown in Figure 8.15(a) a b c e f g [a] [b] [c] [h] d d e f g a h k i j [k] [l] h [b] [d] Figure 8.15 h d [d] [f] Figure 8.16 [d] [f] Figure 8.17 ◆ If G(ν) is also an equivalence relation defined by the partition a b c d e f g {a, b} Figure 8.16. ◆ [l] Figure 8.17 G = (V, E, γ) is a graph, there will be some vertices v and w ∈ V that are the end points of an edge e ∈ E, and we may let R be the equivalence relation on V defined by the partition {{v, w}, {x} for all x ∈ V − {v, w}}. In this case we denote the quotient graph G by G[e]. The graph G[e] is obtained by merging two vertices of G. The edge e disappears, and any other edge between v and w becomes a loop. 310 Chapter 8 Topics in Graph Theory Example 11 If G is the graph of Figure 8.10, and e is the edge labeled e3 , then G[e3] is the graph shown in Figure 8.17. We have treated the process in this chapter. ◆ Merging one edge is used in constructing a spanning tree (see Chapter 7). The graph G e is a result of merging vertices of the edge e. 8.1 Exercises In Exercises 1 through 4 (Figures 8.18 through 8.21), give V, the set of vertices, and E, the set of edges, for the graph G = (V, E, γ). 1. a 2. a b c d e Figure 8.18 a e b c d f Figure 8.19 3. b c d e f g Figure 8.20 b c d e f 2 0 2 3 4 6 Figure 8.21 4. 5. Give two paths in the graph of Figure 8.20 that contain only one edge. 6. In Exercises 7 through 8, give two subgraphs with 3 vertices of the graph in Exercise 3. 7. 8. Which graphs in Exercises 1 through 4 are regular? Give the degree of each vertex. 9. Which graphs in Exercises 1 through 4 are connected? 10. Give the vertices of degree 3 of the graph in Figure 8.21. 11. Give the degree of each vertex in Figure 8.20. 12. Give two subgraphs G of the graph in Figure 8.20 each with 3 vertices. ◆ In Figure 8.22. Draw the subgraph of G with vertex set {a, b, c, e} and as many edges as possible. 13. Give an example of a circuit in the graph of Exercise 4 that is not a simple circuit. In Exercises 15 through 18, use the graph G in Figure 8.22. Draw a picture of the graph G ⊗ e. c d a b b c d e a b c Figure 8.22 Figure 8.23 15. G = (V, E, γ) where V = {a, b, c, d, e}, E = {e1, e2, e3, e4, e5, e6}, and γ(e1) = γ(e5) = {a, c}, γ(e2) = {a, d}, γ(e3) = {e, c}, γ(e4) = {b, c}, γ(e6) = {e, d}. 16. G = (V, E, γ) where V = {a, b, c, d, e, f, g, h}, E = {e1, e2, . . . , e9}, γ(e1) = {a, c}, γ(e2) = {a, b}, γ(e3) = {d, c}, γ(e4) = {b, d}, γ(e5) = {e, a}, γ(e6) = {e, d}, γ(e7) = {f, e}, γ(e8) = {e, g}, and γ(e9) = {f, g}. 19. Give the degree of each vertex in the graph of Figure 8.20. 17. Give the quotient graph G[e] in Figure 8.21. 10 4 2 12 5 6 8 7 13 9 16 14 15 Figure 8.23 21. Let G be the graph shown in Figure 8.23. (a) Give the vertices of a path of length 5 beginning at 1. In Exercises 18 through 21, let R = {{1, 11}, {2, 4}, {3, 7}, {5, 8}, {6, 9}, {10}}. 22. R = {{1, 11}, {2, 3, 4, 8, 12}, {5, 6, 9, 10}}, 23. R = {{1, 4, 7, 10}, {2, 5, 8, 11}, {3, 6, 9, 12}}, 24. R = {{1, 6}, {2, 7}, {3, 8}, {4, 9}, {5, 10}, {11, 12}, {13, 14}, {15, 16}}. 20. R = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12}, {13}, {14}, {15}, {16}}. 25. R = {{1, 2, 3, 4, 5, 6, 7}, {8}, {9, 10, 11, 12, 13, 14, 15, 16}}. Draw the quotient graph G/R. 26. Consider the graph G of Figure 8.22. Draw the quotient graph G R, R the equivalence relation of Exercise 14, in Figure 8.22. 8.1 Graphs 311 8.2 Euler Paths and Circuits 26. If a graph G has no loops or multiple edges, prove that the number of edges is less than or equal to n(n − 1)/2. 27. How many edges are in Kn, the complete graph on n vertices? 28. How many edges are there in the complete graph Kn? 28. Give an example of a regular, connected graph on six vertices that is not complete. 25. Complete the following statement. Every linear graph on edges. Explain your answer. n vertices must have 17. Give an example of a graph on six vertices that is regular of degree 3. 26. Compare the definitions of path in Chapters 4 and 8. 29. What is the total number of edges in Kn, the complete graph on n vertices? Justify your answer. 16. Give an example of a graph with n vertices and exactly n − 1 edges with all vertices of degree 2. 31. Draw the complete graph on seven vertices. 32. Consider K6. What is the degree of each vertex? 8.2 Euler Paths and Circuits a b d ∂ ∂ ∂ ∂ G ∂ ∂ ∂ ∂ ∂ ∂ ∂ E ∂ ∂ ∂ ∂ ∂ ∂ ∂ ∂ ∂ 31. Explain how the definition of digraph in Chapter 4 differs from the definition of graph in Chapter 8. 26. Prove or disprove that the graphs shown in Figures 8.24(a) and (b) are isomorphic. 29. Prove or disprove that the graphs shown in Figures 8.24(a) and (b) are isomorphic. 30. Prove that if a graph G has no loops or multiple edges, then the sum of the degrees of all vertices is twice the number of edges. 32. Prove that the number of vertices of odd degree in a graph must be even. 34. Prove Theorem 2 in Exercise 32 to prove that a graph with n vertices and n − 1 edges, where n is the degree of n vertices are even numbers. Euler Paths and Circuits In this section and the next, we consider broad categories of problems for which graph theory is used. In the first type of problem, the task is to travel a path using each edge of the graph exactly once. It may or may not be necessary to begin and end at the same vertex. A simple example of this is the common puzzle problem that asks the solver to trace a geometric figure without lifting pencil from paper or tracing an edge more than once. A path in a graph G is called an Euler path if it includes every edge exactly once. An Euler circuit is an Euler path that is a circuit. Leonhard Euler (1707– 1783) worked in many areas of mathematics. The names "Euler path" and "Euler circuit" recognize his work with the K¨onigsberg Bridge problem. Example 1 Figure 8.25 shows the street map of a small neighborhood. A recycling ordinance has been passed, and those responsible for picking up the recyclables must start and end each trip by having the truck in the recycling terminal. They would like to plan the truck route so that the entire neighborhood can be covered and each street need be traveled only once. This graph can be constructed having an edge for each street and a vertex for each intersection. In this problem we then look for an Euler circuit in the graph. ◆ Example 2 (a) An Euler path in Figure 8.26 is π : E, D, B, A, C, D. (b) One Euler circuit in the graph of Figure 8.27 is π : 5, 3, 2, 1, 3, 4, 5. ◆ E 2 4 D B C Figure 8.26 Figure 8.27 A little experimentation will show that no Euler circuit is possible for the graph in Figure 8.26. We also see that an Euler path is not possible for the graph in Figure 8.6 in Section 8.1. (Why?) a b A b A c d Room A b c d Room B b (Outside) Figure 8.28 Example 3 f (l) f g Room C B b e j e g j Figure 8.29 Consider the floor plan of a three-room structure that is shown in Figure 8.28. Each room is connected to every room that it shares a wall with and to the outside along each wall. The problem is this: Is it possible to begin in a room or outside and take a walk that goes through each door exactly once? The graph of Figure 8.29 is constructed so that a vertex corresponds to each room and the outside, and an edge corresponds to each door. A possible graph for this structure is shown in Figure 8.29. The translation of the problem is whether or not there exists an Euler path for this graph. We will solve this problem later. ◆ Two questions arise naturally at this point. Is it possible to determine whether an Euler path or Euler circuit exists without actually finding it? If there must be an Euler path or circuit, is there an efficient way to find one? Consider again the graphs in Example 2. In Figure 8.26 the edge {D, E} must be either the first or the last traveled, because there is no other way to travel to or from vertex E. This means that if G has a vertex of degree 1, there cannot be an Euler circuit. The edge to this vertex must be used first or last. Euler Paths and Circuits 313 Let G be a connected graph. If it has a vertex of degree 1, there cannot be an Euler circuit, and if there is an Euler path, it must begin or end at this vertex. A similar argument applies to any vertex v of odd degree, say 2n + 1. We may travel in on one of these edges and out on another one n times, leaving one edge from v untraveled. This last edge may be used for leaving v or arriving at v, but not both, so a circuit cannot be completed. We have just shown the first of the following results. THEOREM 1 (a) If a graph G has a vertex of odd degree, there can be no Euler circuit in G. (b) If G is a connected graph and every vertex has even degree, then there is an Euler circuit in G. Proof (b) Suppose that there are connected graphs where every vertex has even degree, but there is no Euler circuit. Choose such a G with the smallest number of edges. G must have more than one vertex since, if there were only one vertex of even degree, there is an Euler circuit. If there is more than one vertex, each vertex must have at least two edges. We show that G must have a circuit. Let v1 be a fixed vertex of G, then since G is connected and has more than one vertex, there must be an edge between v1 and some other vertex v2. Since v2 has at least degree 2, there must be another edge from v2 to v3. If v3 = v1 we have a circuit. If the other vertex is v1, we have a circuit. If not, the process continues. This is a simple path (all vertices distinct), and since G has only finitely many vertices, we must eventually return to some vertex vi, which completes a circuit. Let π be a simple path in G with the largest possible number of edges. Since G has even degree and the path π must end in a vertex of degree at least two, vi must be a vertex in the path. It is possible to construct a circuit π : vi, vi+1, . . . , vi in G. We will show that G must have an Euler circuit. Since we now know that G has a simple circuit, we have a circuit π in G that uses the largest number of edges. Since π is a circuit, deleting its edges will reduce the degree of every vertex by 0 or 2, so G 1 is also a graph with all vertices of even degree. The graph G(1) may not be connected, but we can choose a largest connected component (piece) and call this graph G 2 (G 2 may be G 1). Now G 2 has fewer edges than G, and so (because of the way G was chosen), G 2 must have an Euler path π 1 . If π passes through all the vertices on G 2 , then π and π 1 clearly have a vertex v in common. It is then possible to piece π and π 1 together to form an Euler circuit. Since π is not in π, there must be a vertex of G that is not in G 2 . If π passes through all the vertices on G, then G 2 has a vertex in common with π, say v. Then we can construct a path by traveling π from v, returning to v, then traveling π 1 from v back to v. If w is a vertex of G not in G 2 , there is a path in G from v to w. Since G is connected, this path must contain a vertex x in common with π. Since G 2 contains all the edges at x in G that are not in π, G 2 contains x. This is a contradiction. Thus G 2 must contain all the vertices on π and we have shown that π and π 1 can be combined to form an Euler circuit. Since G has an Euler circuit, we have a contradiction. So we must conclude that there exists a graph G with every vertex of even degree that has no Euler circuit is false. 314 Chapter 8 Topics in Graph Theory begin and end at any of the vertices of odd degree. Our argument shows that if there are exactly two vertices of odd degree, then we can construct an Euler path between them. COROLLARY 1 (a) If a graph G has more than two vertices of odd degree, then there can be no Euler path in G. (b) If G is connected and has exactly two vertices of odd degree, there is an Euler path in G. Any Euler path in G must begin at one vertex of odd degree and end at the other. Proof (a) Let v1, v2, v3 be vertices of odd degree. Any possible Euler path must leave (or arrive at) each of v1, v2, v3 with no way to return (or leave) since each of these vertices has odd degree. One vertex of these three vertices may be the beginning of the Euler path and another the end, but this leaves the third vertex at one end of an untraveled edge. Thus there is no Euler path. (b) Let u and v be the two vertices of odd degree. Adding the edge {u, v} to G produces a connected graph G′ all of whose vertices have even degree. By Theorem 1(b), there is an Euler circuit π in G′. Omitting {u, v} from π produces an Euler path that begins at u (or v) and ends at v (or u). ■ Example 4 Which of the graphs in Figures 8.30, 8.31, and 8.32 have an Euler circuit, an Euler path but not an Euler circuit, or neither? Figure 8.30 Figure 8.31 Figure 8.32 Solution (a) In Figure 8.30, each of the four vertices has degree 3; thus, by Theorems 1 and 2, there is neither an Euler circuit nor an Euler path. (b) The graph in Figure 8.31 has exactly two vertices of odd degree. There is no Euler circuit, but there must be an Euler path. (c) In Figure 8.32, every vertex has even degree; thus the graph must have an ◆ Euler circuit. Example 5 Let us return to Example 3. We see that the four vertices have degrees 4, 4, 5, and 7, respectively. Thus the problem can be solved by Theorem 2; that is, there is an Euler path. One is shown in Figure 8.33. Using the labels of Figure 8.29, this path is specified by Vπ : C, D, C, A, D, A, B, D, B and Eπ : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13. ◆ Theorems 1 and 2 are examples of existence theorems. They guarantee the existence of an object of a certain type, but they give no information on how to 8.2 Euler Paths and Circuits A 315 Room B Room C D (Outside) Figure 8.33 Room A B C D Figure 8.34 produce the object. There is one hint in Theorem 2(b) about how to proceed. In Figure 8.34, a path from B to B is shown in Figure 8.34. We can extend this path to ... B, A, D is shown in Figure 8.34. Since there are two vertices of odd degree in the graph of Figure 8.34, namely A and B, we must begin at one and end at the other. One possible Euler path for this graph is specified by Vπ : B, A, D, C, A, B, C, D and Eπ : 1, 2, 3, 4, 5, 6, 7. We next give an algorithm that produces an Euler circuit for a connected graph with no vertices of odd degree. We need an additional definition before stating the algorithm. An edge is a bridge in a connected graph G if deleting it would create a disconnected graph. For example, in the graph of Figure 8.4, Section 8.1, {B, E} is a bridge. Algorithm FLEURY'S ALGORITHM Let G = (V, E, γ) be a connected graph with each vertex of even degree. Step 1 Select an edge e1 that is not a bridge in G. Let its vertices be v1, v2 . Let π be specified by Vπ : v1, v2 and Eπ : e1. Remove e1 from E and let G1 be the resulting subgraph of G. Step 2 Suppose that Vπ : v1, v2, . . . , vk and Eπ : e1, e2, . . . , ek−1 have been constructed so far, and that all of these edges and any resulting isolated vertices have been removed from V and E to form Gk−1. Since vk has even degree, and ek−1 ends there, there must be an edge ek in Gk−1 that also has vk as a vertex. If there is more than one such edge, select one that is not a bridge for Gk−1. Denote the vertex of ek other than vk by vk+1, and extend Vπ and Eπ to Vπ : v1, v2, . . . , vk, vk+1 and Eπ : e1, e2, . . . , ek−1, ek. Then delete ek and any isolated vertices from Gk−1 to form Gk. Step 3 Repeat Step 2 until no edges remain in E. ● Example 6 Use Fleury's algorithm to construct an Euler circuit for the graph in Figure 8.35. E G H Figure 8.35 C A B Chapter 8 Topics in Graph Theory 316 Solution According to Step 1, we may begin anywhere. Arbitrarily choose vertex A. We summarize the results of applying Step 2 repeatedly in Table 8.1. TABLE 8.1 Current Path π : A π : A, B π : A, B, E π : A, B, E, D π : A, B, E, D, C π : A, B, E, D, C, A ... Next Edge Reasoning {A, B} (B, E) {E, D} (D, C) (C, A) ... No edges from A is a bridge. Choose any one. Only one edge from B remains. No edges from E is a bridge. Choose any one. {A, G} Only one edge from D remains. Only one edge from C remains. No edges from A is a bridge. Only one edge from G remains. Only one edge from H remains. Only one edge from G remains. Only one edge from E remains. Only one edge from F remains. The graph in Figure 8.36 has been obtained from G by deleting edges... The entire path is π : A, B, E, D, C, A, G, F, E, H, G, F, E, H, G, A, C, B, H, H, G, F, A, A, G, B, F, E, F, E, H, A, B, C, D, E, F, G, H, A. ◆ Figure 8.36 The graph in Figure 8.36 has been obtained from G by deleting edges; this makes sure that the remaining graph can still be traversed. 8.2 Exercises In Exercises 1 through 8 (Figures 8.37 through 8.44), tell whether the graph has an Euler circuit, an Euler path but no Euler circuit, or neither. Give reasons for your choice. 1. Figure 8.37 2. Figure 8.38 3. Figure 8.39 4. Figure 8.40 5. Figure 8.41 6. Figure 8.42 7. Figure 8.43 8. Figure 8.44 Figure 8.37 Figure 8.38 Figure 8.39 Figure 8.40 Figure 8.41 In Exercises 9 and 10, determine if it is possible to make the indicated map. 9. Figure 8.45 10. Figure 8.46. 11. An art museum arranged its current exhibit in the five rooms shown in Figure 8.48. Is there a way to tour the exhibit so that you pass through each door exactly once? If so, give a sketch of your tour. 8.2 Euler Paths and Circuits 317 12. Use Fleury's algorithm to produce an Euler circuit for the graph in Figure 8.49. 13. Use Fleury's algorithm to produce an Euler circuit for the graph in Figure 8.49. 14. Is it possible to visit every room in the house by passing through each door exactly once? Explain your reasoning. Figure 8.43 8. Figure 8.44 In Exercises 9 and 10, (Figures 8.45 and 8.46), tell if it is possible to trace the figure without lifting the pencil. Explain why or why not. 9. Figure 8.45 Figure 8.46 11. Figure 8.47 In Exercises 15 through 18, use Fleury's algorithm to find an Euler circuit or Euler path for the graph. 15. Figure 8.47 16. Figure 8.47 Figure 8.48 Figure 8.47 Figure 8.49 Figure 8.50 15. Figure 8.50 16. Figure 8.52 Figure 8.51 17. Figure 8.52 Figure 8.53 Figure 8.52 8 Topics in Graph Theory 318 Chapter 8 Topics in Graph Theory. Begin at a. 19. 20. Consider the graph of the Koenigsberg bridges problem. Use Fleury's algorithm ... 21. Modify Fleury's algorithm so that it will find an Euler path in a graph with exactly two vertices of odd degree. 22. (a) Give an example of a graph with an Euler circuit that also has a bridge. 18. (a) Give an example of an Euler path in the graph shown in Figure 8.47. 19. Let G be a graph with n vertices. Prove that if G has an Euler circuit, then no vertex of G has odd degree. 23. In Example 8, find an Euler circuit ... 24. How many edges must be deleted from K5 to produce a graph with an Euler circuit? 25. (a) Give an example of a connected graph with n vertices of which exactly two have odd degree. 26. Consider the graph of Exercise 20 and Figure 8.47. Have an Euler circuit? Justify your answer. 18. 21. A 22. 23. Figure 8.53 19. Modify the graph in Figure 8.52 by adding the minimum number of duplicate edges needed to make all vertices have even degree. Use Fleury's algorithm to find an Euler circuit for the modified version of the graph. Begin at the upper-left corner. 24. 20. Modify the graph in Figure 8.53 by adding the minimum number of duplicate edges needed to make all vertices have even degree. Use Fleury's algorithm to find an Euler circuit for the modified version of the graph. Begin at E. 8.3 Hamiltonian Paths and Circuits We now turn to the second category of graph problems in which the task is to visit each vertex exactly once, with the exception of the beginning vertex if it must also be the last vertex. For example, such a path would be useful to someone who must service vending machines at a number of locations. Each vertex would represent one location. A Hamiltonian path is a path that contains each vertex exactly once. A Hamiltonian circuit is a circuit that contains each vertex exactly once except for the first vertex, which is also the last. This sort of path is named for the mathematician Sir William Hamilton, who developed and marketed a game consisting of a wooden graph in the shape of a regular dodecahedron and instructions to find what we have called a Hamiltonian circuit. A planar version of this solid is shown in Figure 8.54(a), with a Hamiltonian circuit (one of many) shown in Figure 8.54(b) by the consecutively numbered vertices. It is clear that loops and multiple edges are of no use in finding Hamiltonian circuits, since loops could not be used, and only one edge can be used between any two vertices. Thus we will suppose that any graph we mention has no loops or multiple edges. 1 20 13 12 11 14 19 18 17 15 16 10 3 (a) 2 9 4 5 6 7 8 (b) Figure 8.54 Example 1 Consider the graph in Figure 8.55. The path a, b, c, d, e is a Hamiltonian path because it contains each vertex exactly once. It is not hard to see, however, that there is no Hamiltonian circuit for this graph. For the graph shown in Figure 8.56, the path A, D, C, B, A is a Hamiltonian circuit. In Figures 8.57 and 8.58, no Hamiltonian path is possible. (Verify this.) Figure 8.55 Figure 8.56 Figure 8.57 Figure 8.58 ◆ Any complete graph Kn has Hamiltonian circuits. In fact, starting at any vertex, you can visit the other vertices sequentially in any desired order. What questions can be asked about Hamiltonian paths and circuits? The first question: is it possible to find a Hamiltonian circuit in a graph? It is not possible to tell by looking at the graph. Questions analogous to those about Euler paths and circuits can be asked about Hamiltonian paths and circuits. Is it possible to determine whether a Hamiltonian path or circuit exists? If there must be a Hamiltonian circuit, is there an efficient way to find it? Surprisingly, considering Theorems 1 and 2 of Section 8.2, the first question about Hamiltonian paths and circuits has not been completely answered and the second is almost entirely open. However, we can make several observations based on the examples. If a graph G on n vertices has a Hamiltonian circuit, then G must have at least n edges. We now state some partial answers that say if a graph G has "enough" edges, a Hamiltonian circuit can be found. These are again existence theorems; they say nothing about how to find Hamiltonian circuits. THEOREM 1 Let G be a connected graph with n vertices, n > 2, and no loops or multiple edges. G has a Hamiltonian circuit if for any two vertices u and v of G that are not adjacent, the degree of u plus the degree of v is greater than or equal to n. ■ We omit the proof of this result, but from it we can prove the following: Corollary 1 G has a Hamiltonian circuit if each vertex has degree greater than or equal to n/2. Proof The sum of the degrees of any two vertices is at least n/2 + n/2 = n, so the hypotheses of Theorem 1 hold. ■ THEOREM 2 Let the number of edges of G be m. Then G has a Hamiltonian circuit if m ≥ ½(n² − 3n + 6) (recall that n is the number of vertices). ■ Proof Suppose that u and v are any two vertices of G that are not adjacent. We write deg(u) for the degree of u. Let H be the graph produced by eliminating u and v from G along with any edges that have u or v as end points. Then H has n − 2 vertices and m − deg(u) − deg(v) edges (one fewer edge would have been removed if u and v had been adjacent). The maximum number of edges that H could possibly have is n−2C2 (see Section 3.2). This happens when there is an edge connecting every distinct pair of vertices. Thus the number of edges of H is at most n − 2 C 2 = ½(n − 2)(n − 3) or ½(n² − 5n + 6). We then have m − deg(u) − deg(v) ≤ ½(n² − 5n + 6). Therefore, deg(u) + deg(v) ≥ m − ½(n² − 5n + 6) ≥ ½(n² − 3n + 6) − ½(n² − 5n + 6) = n. Thus the result follows from Theorem 1. ■ Example 3 A H B D C E F G H Figure 8.59 The converses of Theorems 1 and 2 given previously are not true; that is, the conditions given are sufficient, but not necessary, for the conclusion. Consider the graph represented by Figure 8.59. Here n, the number of vertices, is 8, each vertex has degree 2, and deg(u) + deg(v) = 4 for every pair of nonadjacent vertices u and v. The total number of edges is also 8. Thus the premises of Theorems 1 and 2 fail to be satisfied, but there are certainly Hamiltonian circuits for this graph. ◆ The problem we have been considering has a number of important variations. In one case, the edges may have weights representing distance, cost, and the like. The problem is then to find a Hamiltonian circuit (or path) for which the total sum of weights in the path is a minimum. For example, the vertices might represent cities; the edges, lines of transportation; and the weight of an edge, the cost of traveling along that edge. This version of the problem is often called the traveling salesperson problem. Another important category of problems involving Hamiltonian paths will be discussed in Section 8.5. 8.3 Exercises In Exercises 1 through 8 (Figures 8.60 through 8.65), determine whether the graph has a Hamiltonian circuit, a Hamiltonian path but no Hamiltonian circuit, or neither. If the graph has a Hamiltonian circuit, give the circuit. 1. A 2. B 5. 6. Figure 8.64 Figure 8.65 a C D c d E Figure 8.61 Figure 8.60 7. Give two Hamiltonian circuits in K5 that have no edges in common. In Exercises 9 through 11 (Figures 8.66 through 8.69), find a Hamiltonian circuit for the graph given. 8.

3. 1 4. 2 3 B 3 5 2 D 5 4 3 6 F Figure 8.62 7 E Figure 8.63 A 2 C Figure 8.66 5 4 G 5 2 4 F 6 6 4 H 8.4 Transport Networks 9. B In Exercises 12 through 15, find a Hamiltonian circuit of minimal weight for the graph represented by the given figure. H 2 3 3 A D 6 2 2 5 C E 5 6 4 3 4 F G Figure 8.67 10. B 2 6 2.7 A 2.1 2 4 4.2 1 4 4.2 1 3 3 9 1 2 8 1.8 J 3 2 3.4 H C 3.6 D 2.1 2.5 5.3 1.7 E 2 2 4.4 F G Figure 8.68 11. 15 B 12 10 C 16 10 14 G F Figure 8.69 8 4 4 4 2 1 5 2 Figure 8.70 3 5 4 2 3 12 20 12 6 3 3 15 20 A E 321 12, Figure 8.66 13. Figure 8.67 14. Figure 8.68 15. Figure 8.69 16. Find a Hamiltonian circuit of minimal weight for the graph represented by Figure 8.66 if you must begin and end at D. 17. Find a Hamiltonian circuit of minimal weight for the graph represented by Figure 8.67 if you must begin and end at F. 18. For the complete graph on n vertices, K_n, $n \geq 3$, (a) how many edges must a Hamiltonian circuit have? (b) how many different Hamiltonian circuits, beginning at a fixed vertex, are there? 19. Prove that K_n, the complete graph on n vertices with $n \geq 3$, has $(n-1)!$ Hamiltonian circuits. 20. Give an example of a graph with at least four vertices with a circuit that is both an Euler and a Hamiltonian circuit. 21. Give an example of a graph that has an Euler circuit and a Hamiltonian circuit that are not the same. 22. Let $G = (V, E)$ be a graph with $|V| = n$ that has no multiple edges. The relation R on V defined by G can be represented by a matrix, M_R. Explain how to use M_R to determine if G is connected. 23. Using the usual vertex labels of strings of 0's and 1's, give a Hamiltonian circuit for the Hasse diagram of B_3. 24. Using the usual vertex labels of strings of 0's and 1's, give a Hamiltonian circuit for the Hasse diagram of B_4. 25. The problem of finding a Hamiltonian circuit for the Hasse diagram of B_n is equivalent to a problem about strings of 0's and 1's of length n. State this problem. Transport Networks We have previously examined several types of labeled graphs. In this section we return to the idea of a directed graph (digraph). An important use of labeled digraphs is to model what are commonly called transport networks. Consider the labeled digraph shown in Figure 8.70. This might represent a pipeline that carries water from vertex 1 to vertex 6 as part of a municipal water system. The label on an edge represents the maximum flow that can be passed through that edge and is called the capacity of the edge. Many situations can be modeled in this way. For instance, Figure 8.70 might equally well represent an oil pipeline, a highway system, a communications network, or an electric power grid. The vertices of a network are usually called nodes and may denote pumping stations, shipping depots, relay stations, or highway interchanges. More formally, a transport network, or a network, is a connected digraph N with the following properties: (a) There is a unique node, the source, that has in-degree 0. We generally label the source node 1. (b) There is a unique node, the sink, that has out-degree 0. If N has n nodes, we generally label the sink as node n. (c) The graph N is labeled. The label, C_{ij}, on edge (i, j) is a nonnegative number called the capacity of the edge. For simplicity we also assume that all edges carry material in one direction only; that is, if (i, j) is in N, then (j, i) is not. Flows The purpose of a network is to implement a flow of water, oil, electricity, traffic, or whatever the network is designed to carry. Mathematically, a flow in a network N is a function that assigns to each edge (i, j) of N a nonnegative number F_{ij} that does not exceed C_{ij}. Intuitively, F_{ij} represents the amount of material passing through the edge (i, j) when the flow is F. Informally, we refer to F_{ij} as the flow through edge (i, j). We also require that for each node other than the source and the sink, the sum of the F_{ik} on edges entering node k must be equal to the sum of the F_{kj} on edges leaving node k. This means that material cannot accumulate, be created, dissipate, or be lost at any node other than the source or the sink. This is called conservation of flow. A consequence of this requirement is that the sum of the flows leaving the source must equal the sum of the flows entering the sink. This sum is called the value of the flow, written value(F). We can represent a flow F by labeling each edge (i, j) with the pair (C_{ij}, F_{ij}). A flow F in the network represented by Figure 8.70 is shown in Figure 8.71. Example 1 (4, 2) 1 (4, 3) (2, 1) (5, 3) 2 (4, 3) (3, 2) 6 (3, 3) 5 (2, 2) Figure 8.71 5 Example 2 In Figure 8.71, flow is conserved at node 4 since there are input flows of size 2 and 1, and an output flow of size 3. (Verify that flow is conserved properly at the other 4 nodes.) Here value(F) = 5. Maximum Flows For any network an important problem is to determine the maximum value of a flow through the network and to describe a flow that has the maximum value. For obvious reasons this is commonly referred to as the maximum flow problem. Figure 8.72(a) shows a flow that has value 8. Three of the five edges are carrying their maximum capacity. This seems to be a good flow function, but Figure 8.72(b) shows a flow with value 10 for the same network. ● (6, 4) 1 (4, 4) 2 (6, 6) (2, 2) 3 (4, 6) (4, 2) 1 2 (6, 6) (2, 0) (4, 3) 4 (4, 8) (b) Figure 8.72 Example 2 shows that even for a small network, we need a systematic procedure for solving the maximum flow problem. Examining the flow in Figure 8.72(a) shows that using the edge from node 3 to node 2 as we did was a mistake. We should reduce flow in edge (3, 2) so that we can increase it in other edges. Suppose that in some network N we have an edge (i, j) that is carrying a flow of 5 units. If we want to reduce this flow to 3 units, we can imagine that it is 8.4 Transport Networks 65 (4, 4) 1 2 (2, 2) 3 (4, 2) 3 Figure 8.73 (4, 4) 2 (2, 1) 2 4 (6, 6) 1 2 (6, 2) 4 3 (4, 2) 3 combined with a flow of 2 units in the opposite direction. Although edge (j, i) is not in N, there is no harm in considering such a virtual flow as long as it only has the effect of reducing the existing flow in the edge (i, j). Figure 8.73 displays a portion of the flow shown in Figure 8.72(a). The path π: 1, 2, 3, 4 is not an actual path in this network, since (2, 3) is not an actual edge. However, π is a path in the symmetric closure of the network. (Refer to Section 4.7 for symmetric closures.) Moreover, if we consider a virtual flow of two units through π, the effect on the network is to increase the flows through edges (1, 2) and (3, 4) by two units and to decrease the flow through (3, 2) by two units. Thus, the flow of Figure 8.72(a) becomes the flow of Figure 8.72(b). We now describe this improvement in general terms. Let N be a network and let G be the symmetric closure of N. Choose a path in G and an edge (i, j) in this path. If (i, j) belongs to N, then we say this edge has positive excess capacity if $e_{ij} = C_{ij} - F_{ij} > 0$. If (i, j) is not an edge of N, then we are traveling this edge in the wrong direction. In this case we say (i, j) has excess capacity $e_{ij} = F_{ji}$ if $F_{ji} > 0$. Then increasing flows through edge (i, j) will have the effect of reducing F_{ji}. We now give a procedure for solving a maximum flow problem. The Labeling Algorithm This algorithm is due to Ford and Fulkerson and is often called the labeling algorithm. The labeling referred to is an additional labeling of nodes. We have used integer capacities for simplicity, but Ford and Fulkerson show that this algorithm will stop in a finite number of steps if the capacities are rational numbers. Let N be a network with n nodes and G be the symmetric closure of N. All edges and paths used are in G. Begin with all flows set to 0. As we proceed, it will be convenient to track the excess capacities in the edges and how they change rather than tracking the increasing flows. When the algorithm terminates, it is easy to find the maximum flow from the final excess capacities. ALGORITHM THE LABELING ALGORITHM Step 1 Let N1 be the set of all nodes connected to the source by an edge with positive excess capacity. Label each j in N1 with [E j, 1], where E j is the excess capacity e_{1j} of edge $(1, j)$. The 1 in the label indicates that j is connected to the source, node 1. Step 2 Let node j in N1 be the node with smallest node number and let N2(j) be the set of all unlabeled nodes, other than the source, that are joined to node j and have positive excess capacity. Suppose that node k is in N2 (j) and (j, k) is the edge with positive excess capacity. Label node k with [E k, j], where E k is the minimum of E j and the excess capacity e_{jk} of edge (j, k). In this way, repeat this process for when all the nodes in N2 (j) are labeled, then repeat this process for the other nodes in N1. Let N2 = ∪ N2 (j). Note that after Step 1, we have labeled each node j in N1 with E j, the amount of material that can flow from the source to j through one edge and with the information that this flow came from node 1. In Step 2, previously unlabeled nodes k that can be reached from the source by a path π : 1, j, k are labeled with [E k, j]. Here E k is the amount of flow that can pass through π since it is the smaller of the amount that can reach j and the amount that can then pass on to k. Thus when Step 2 is finished, we have constructed two-step paths to all nodes in N2. The label for each of these nodes records the total flow that can reach the 324 Chapter 8 Topics in Graph Theory node through the path and its immediate predecessor in the path. We attempt to continue this construction increasing the lengths of the paths until we reach the sink (if possible). Then the total flow can be increased and we can retrace the path used for this increase. Step 3 Repeat Step 2, labeling all previously unlabeled nodes N3 that can be reached from a node in N2 by an edge having positive excess capacity. Continue this process forming sets N4 , N5 , . . . until after a finite number of steps either (i) the sink has not been labeled and no other nodes can be labeled. It can happen that no nodes have been labeled; remember that the source is not labeled. Or (ii) the sink has been labeled. Step 4 In case (i), the algorithm terminates and the total flow then is a maximum flow. (We show this later.) Step 5 In case (ii) the sink, node n, has been labeled with [E n, m], where E n is the amount of extra flow that can be made to reach the sink through a path π. We examine π in reverse order. If edge $(i, j) \in N$, then we increase the flow in (i, j) by E n and decrease the excess capacity e_{ij} by the same amount. Simultaneously, we increase the excess capacity of the (virtual) edge (j, i) by E n since there is that much more flow in (i, j) to reverse. If, on the other hand, $(i, j) \notin N$, we decrease the flow in (j, i) by E n and decrease its excess capacity by E n. We simultaneously decrease the excess capacity in (i, j) by the same amount, since there is less flow in (i, j) to reverse. We now have a new flow that is E n units greater than before and we return to Step 1. ● Example 3 Use the labeling algorithm to find a maximum flow for the network in Figure 8.70. Solution Figure 8.74 shows the network with initial capacities of all edges in G. The initial flow in all edges is zero. $e_{14} = 4$ $e_{41} = 0$ $e_{45} = 3$ $e_{24} = 2$ $e_{42} = 0$ $e_{12} = 5$ $e_{21} = 0$ $e_{54} = 0$ $e_{25} = 2$ $e_{52} = 0$ $e_{23} = 3$ $e_{32} = 0$ $e_{56} = 4$ $e_{65} = 0$ $e_{36} = 3$ $e_{63} = 0$ Figure 8.74 Step 1 Starting at the source, we can reach nodes 2 and 4 by edges having excess capacity, so N1 = {2, 4}. We label nodes 2 and 4 with the labels [5, 1] and [4, 1], respectively, as shown in Figure 8.75. Step 2 From node 2 we can reach nodes 5 and 3 using edges with positive excess capacity. Node 5 is labeled with [2, 2] since only two additional units of flow can pass through edge (2, 5). Node 3 is labeled with [3, 2] since 3 8.4 Transport Networks $e_{41} = 0$ $e_{14} = 4$ [4, 1] 4 $e_{45} = 3$ $e_{54} = 0$ [5, 1] ... is the excess capacity of edge (2, 3). Thus N2 (2) = {3, 5}. From node 4 we could go to node 5, but node 5 has already been labeled. Thus N2 = {3, 5}. Step 3 From node 3 we can reach node 6 ... Figure 8.76 Step 5 We work backward through the path 1, 2, 3, 6 and subtract 3 from the excess capacity of each edge, indicating an increased flow through each edge, and adding an equal amount to the excess capacities of the (virtual) edges. We now return to Step 1 with the situation shown in Figure 8.77. Proceeding as before, nodes 2 and 4 are labeled [2, 1] and [4, 1], respectively. Note that E 2 is now only 2 units, the new excess capacity of edge (1, 2). Node 2 can no longer be used to label node 3, since there is no excess capacity in the edge (2, 3). But node 5 now will be labeled [2, 2]. Once again no labeled node can be reached from node 4, so we now move 326 Chapter 8 Topics in Graph Theory to Step 2 and label nodes 3 and 6 (the sink). The final result of Step 3 is shown in Figure 8.78, and we have discovered the flow that 2 units to a total of 5 units. We now return to Step 1 again with the situation shown in Figure 8.78. We move to Step 5 again and work back along the path 1, 2, 5, 6, subtracting 2 from the excess capacities of these edges and adding 2 to the capacities of the corresponding (virtual) edges. We return to Step 1 with Figure 8.79. This time Steps 1 and 2 produce the following results. Only node 4 is labeled from node 1, with [4, 1]. Node 5 is the only node labeled from node 4, with [3, 4]. Step 3 begins with Figure 8.80. At this point, node 5 could label node 2 using the excess capacity of edge (5, 2). (Verify that this would label node 2 with [2, 5].) However, node 5 can also be used to label the sink. The sink is labeled [2, 5] and the total flow is increased to 7 units. In Step 5, we work back along the path 1, 4, 5, 6, adjusting excess capacities. We return to Step 1 with the configuration shown in Figure 8.81. In this case, we can only label node 4 from node 1 with [2, 1]. ... In Step 5, we find the edge (5, 2) and ... and the algorithm terminates. The final flow has value 7. By subtracting the final excess capacity of each edge (i, j) from its capacity ... of all edges. This flow is shown in Figure 8.83. ● [1, 4] (4, 4) 1 (4, 3) (2, 1) (5, 3) 2 (4, 3) (3, 2) 6 (3, 3) 5 (2, 2) Figure 8.83 The Max Flow Min Cut Theorem In a network N, a cut is a set K of edges having the property that every path from the source to the sink contains at least one edge from K. In effect, a cut does "cut" a digraph into two pieces, one containing the source and one containing the sink. If the edges of a cut were removed, nothing could flow from the source to the sink. The capacity of a cut K, c(K), is the sum of the capacities of all edges in K. Figure 8.84 shows two cuts for the network represented by Figure 8.70. Each cut is marked by a jagged line and consists of all edges touched by the jagged line. ● Verify that c(K1) = 10 and c(K2) = 7. Chapter 8 Topics in Graph Theory 328 If F is any flow and K is any cut, then value(F) ≤ c(K). This is true because all parts of F must pass through the edges of K, and c(K) is the maximum amount that can pass through the edges of K. Now suppose for some flow F and some cut K, value(F) = c(K); in other words, the flow F uses the full capacity of all edges in K. Then F would be a flow with maximum value, since no flow can have value bigger than c(K). Similarly, K must be a minimum capacity cut, because every cut must have capacity at least equal to value(F). From this discussion we conclude the following. THEOREM 1 The Max Flow Min Cut Theorem A maximum flow F in a network has value equal to the capacity of a minimum cut of the network. ■ Example 5 The minimum cut corresponding to the maximum flow found in Example 3 is K = {(5, 6), (3, 6)} with c(K) = 7 = value(F). ● We now show that the labeling algorithm results in a maximum flow by finding a minimum cut whose capacity is equal to the value of the flow. Suppose that the algorithm has been run and has stopped at Step 4. Then the sink has not been labeled. Divide the nodes into two sets, M1 and M2, where M1 contains the source and all nodes that have been labeled, and M2 contains all unlabeled nodes, other than the source. Let K consist of all edges of the network N that connect a node in M1 with a node in M2. Any path π in N from the source to the sink begins with a node in M1 and ends with a node in M2. If i is the last node in π that belongs to M1 and j is the node that follows i in the path, then j belongs to M2 and so by definition (i, j) is in K. Therefore, K is a cut. Now suppose that (i, j) is an edge in K, so that i ∈ M1 and j ∈ M2. The final flow F produced by the algorithm must result in (i, j) carrying its full capacity; otherwise, we could use node i and the excess capacity to label j, which by definition is not labeled. Thus the value of the final flow of the algorithm is equal to the capacity of K, and so F is a maximum flow. 8.4 Exercises In Exercises 1 through 4 (Figures 8.85 through 8.88), label the network in the given figure in the manner of Figure 8.71. Let C_{ij} be the first number and F_{ij} the second number. ... In Exercises 5 through 10, find a maximum flow in the given network by using the labeling algorithm. 5. ... 11 Matching Problems The definition of a transport network can be extended, and the concept of a maximal flow in such a network can be used to model situations that, at first glance, do not seem to be network problems. ... Consider for example, water flowing ... A transport network N with some nodes that correspond to ... Example 1 ... We now show how to model a matching problem as a transport network problem. ... Let A = {a1, a2, ..., am} and B = {b1, b2, ..., bn} ... a matching function M is an M-matching ... Example 3 Let A be a set of girls and B a set of boys attending a school dance. Define R by a R b if and only if a knows b. ... THEOREM 1 Hall's Marriage Theorem Let R be a relation from A to B. Then there exists a complete matching M if and only if for each X ⊆ A, |X| ≤ |R(X)|. Proof If a complete matching M exists, then M(X) ⊆ R(X) for every subset X of A. But M is one to one, so |X| = |M(X)| ≤ |R(X)|. Conversely, suppose that for any X ⊆ A, |X| ≤ |R(X)|. ... 8.5 Exercises ... 8.6 Coloring Graphs Suppose that G = (V, E) is a graph with no multiple edges. A coloring of G using n colors is a function $f : V \to C$ where C is a set of n colors such that ... The smallest number of colors needed ... is the chromatic number of G, denoted by χ(G). ... Figure 8.104 ... Figure 8.105 ... A coloring of a map is a way to color each region (country, state, county, province, etc.) so that two distinct regions sharing a common border have the same color. ... THEOREM 1 The Four Color Theorem ... Chromatic Polynomials Closely related to the problem of computing χ(G) is the problem of computing the total number of different proper colorings of a graph G using a set C = {c1, c2, ..., cx} of colors. If G is a graph and x ≥ 0 is an integer, let $P_G(x)$ be the number of ways to color G properly using x or fewer colors. Since $P_G(x)$ is a definite number for each x, we see that P_G is a function. ... Example 4 Consider the linear graph L_4 ... $P_{L_4}(x) = x(x-1)^3$. ... Example 5 ... $P_{K_n}(x) = x(x-1)(x-2)\cdots(x-n+1)$ ... THEOREM 2 ... If G is a disconnected graph with components $G_1, G_2, \ldots, G_m$, then $P_G(x)$ is the product of the chromatic polynomials for each component ... THEOREM 3 ... $P_G(x) = P_{G_e}(x) - P_{G^e}(x)$ ... 8.6 Exercises ... Tips for Proofs ... Key Ideas for Review • Graph: $G = (V, E, \gamma)$, where V is a finite set of objects, called vertices, E is a set of objects, called edges, and γ is a function that assigns to each edge a two-element subset of V. • Degree of a vertex: number of edges at the vertex • Path: see page 307 • Circuit: path that begins and ends at the same vertex • Simple path or circuit: see page 307 • Connected graph: There is a path from any vertex to any other vertex. • Subgraph: see page 308 • Euler path (circuit): path (circuit) that contains every edge ... • Hamiltonian path: path that includes each vertex of the graph exactly once • Hamiltonian circuit ... • Transport network: see page 322 • Capacity of a cut: see page 327 • Max Flow Min Cut Theorem: A maximum flow in a network has value equal to the capacity of a minimum cut of the network. • Matching function: see page 330 • Hall's Marriage Theorem ... • Coloring of a graph using n colors: see page 334 • Proper coloring of a graph: adjacent edges have different colors. • Chromatic number of a graph G, χ(G): smallest number of colors needed for a proper coloring of G • Planar graph: graph that can be drawn in a plane with no crossing edges • Chromatic polynomial P_G: number of proper colorings of G in terms of the number of colors available Chapter 8 Self-Test ... Experiment 8 ... CHAPTER 9 Semigroups and Groups Prerequisites: Chapters 4, 5, and 6 The notion of a mathematical structure was introduced in Section 1.6. In the following chapters, other types of mathematical systems were developed; some, such as (propositions, ∧, ∨, ∼), were not given specific names; but others, such as B_n, the Boolean algebra on n elements, were named. In this chapter, we identify more types of mathematical structures, namely, semigroups, groups, rings, and fields. ... Looking Back The term group was applied by Evariste Galois (1811–1832) in 1830 to a set of permutations of a finite set that satisfies certain properties. Galois was born and died in Paris. He was educated by his mother at home until the age of 12. ... 9.1 Binary Operations Revisited We defined binary operations earlier (see Section 1.6) and noted in Section 5.2 that a binary operation may be used to define a function. Here we turn the process around and define a binary operation as a function with certain properties. A binary operation on a set A is an everywhere defined function $f : A \times A \to A$. Observe the following properties that a binary operation must satisfy: 1. Since Dom(f) = A × A, f assigns an element f(a, b) of A to each ordered pair (a, b) in A × A. That is, the binary operation must be defined for each ordered pair of elements of A. 2. Since a binary operation is a function, only one element of A is assigned to each ordered pair. Thus we can say that a binary operation is a rule that assigns to each ordered pair of elements of A a unique element of A. ... Example 1 Let A = Z. Define a ∗ b as a + b. Then ∗ is a binary operation on Z. ● Example 2 Let A = R. Define a ∗ b as a/b. Then ∗ is not a binary operation, since it is not defined for every ordered pair of elements of A. For example, 3 ∗ 0 is not defined, since we cannot divide by zero. ● Example 3 Let $A = Z^+$. Define a ∗ b as a − b. Then ∗ is not a binary operation since it does not assign an element of A to every ordered pair of elements of A; for example, 2 ∗ 5 ∉ A. ● Example 4 Let A = Z. Define a ∗ b as a number less than both a and b. Then ∗ is not a binary operation, since it does not assign a unique element of A to each ordered pair of elements of A; for example, 8 ∗ 6 could be 5, 4, 3, 1, and so on. Thus, in this case, ∗ would be a relation from A × A to A, but not a function. ● Example 5 Let A = Z. Define a ∗ b as max{a, b}. Then ∗ is a binary operation; for example, 2 ∗ 4 = 4, −3 ∗ (−5) = −3. ● Example 6 Let A = P(S), for some set S. If V and W are subsets of S, define V ∗ W as V ∪ W. Then ∗ is a binary operation on A. Moreover, if we define V ∗′ W as V ∩ W, then ∗′ is another binary operation on A. ● As Example 6 shows, it is possible to define many binary operations on the same set. Example 7 Let M be the set of all n × n Boolean matrices for a fixed n. Define A ∗ B as A ∨ B (see Section 1.5). Then ∗ is a binary operation. This is also true of A ∧ B. ● Example 8 Let L be a lattice. Define a ∗ b as a ∧ b (the greatest lower bound of a and b). Then ∗ is a binary operation on L. This is also true of a ∨ b (the least upper bound of a and b). ● Tables If A = {a1, a2, ..., an} is a finite set, we can define a binary operation on A by means of a table as shown in Figure 9.1. The entry in position i, j denotes the element $a_i \ast a_j$. ● Example 9 Let A = {0, 1}. We define the binary operations ∨ and ∧ by the following tables: ∨ 0 1 0 0 1 1 1 1 ∧ 0 1 0 0 0 1 0 1 ● For A = {a, b}, we shall determine the number of binary operations that can be defined on A. Every binary operation ∗ on A can be described by a table ∗ a b a b Since every blank can be filled in with the element a or b, we conclude that there are 2 · 2 · 2 · 2 = 2^4 or 16 ways to complete the table. Thus, there are 16 binary operations on A. Properties of Binary Operations Several of the properties defined for binary operations in Section 1.6 are of particular importance in this chapter. We repeat them here. A binary operation on a set A is said to be commutative if a ∗ b = b ∗ a for all elements a and b in A. Example 10 The binary operation of addition on Z (as discussed in Example 1) is commutative. ● Example 11 The binary operation of subtraction on Z is not commutative, since 2 − 3 ≠ 3 − 2. ● A binary operation that is described by a table is commutative if and only if the entries in the table are symmetric with respect to the main diagonal. ● Example 12 Which of the following binary operations on A = {a, b, c, d} are commutative? ... Solution The operation in (a) is not commutative, since a ∗ b is c while b ∗ a is b. The operation in (b) is commutative, since the entries in the table are symmetric with respect to the main diagonal. ● A binary operation ∗ on a set A is said to be associative if a ∗ (b ∗ c) = (a ∗ b) ∗ c for all elements a, b, and c in A. Example 13 The binary operation of addition on Z is associative. ● Example 14 The binary operation of subtraction on Z is not associative, since 2 − (3 − 5) ≠ (2 − 3) − 5. ● Example 15 Let L be a lattice. The binary operation defined by a ∗ b = a ∧ b (see Example 8) is commutative and associative. It also satisfies the idempotent property a ∧ a = a. A partial converse of this example is also true, as shown in Example 16. ● Example 16 Let ∗ be a binary operation on a set A, and suppose that ∗ satisfies the following properties for any a, b, and c in A. 1. a = a ∗ a Idempotent property 2. a ∗ b = b ∗ a Commutative property 3. a ∗ (b ∗ c) = (a ∗ b) ∗ c Associative property Define a relation ≤ on A by a ≤ b if and only if a = a ∗ b. Show that (A, ≤) is a poset, and for all a, b in A, GLB(a, b) = a ∗ b. Solution We must show that ≤ is reflexive, antisymmetric, and transitive. Since a = a ∗ a, a ≤ a for all a in A, and ≤ is reflexive. Now suppose that a ≤ b and b ≤ a. Then, by definition and property 2, a = a ∗ b = b ∗ a = b, so a = b. Thus ≤ is antisymmetric. If a ≤ b and b ≤ c, then a = a ∗ b = a ∗ (b ∗ c) = (a ∗ b) ∗ c = a ∗ c, so a ≤ c and ≤ is transitive. Finally, we must show that, for all a and b in A, a ∗ b = a ∧ b (the greatest lower bound of a and b with respect to ≤). We have a ∗ b = a ∗ (b ∗ b) = (a ∗ b) ∗ b, so a ∗ b ≤ b. In a similar way, we can show that a ∗ b ≤ a, so a ∗ b is a lower bound for a and b. Now, if c ≤ a and c ≤ b, then c = c ∗ a and c = c ∗ b by definition. Thus c = (c ∗ a) ∗ b = c ∗ (a ∗ b), so c ≤ a ∗ b. This shows that a ∗ b is the greatest lower bound of a and b. ● 348 Chapter 9 Semigroups and Groups 9.1 Exercises In Exercises 1 through 8, determine whether the description of ∗ is a valid definition of a binary operation on the set. 1. On R, where a ∗ b is ab (ordinary multiplication). 2. On Z^+, where a ∗ b is a/b. 3. On Z, where a ∗ b is a^b. 4. On Z^+, where a ∗ b is a^b. 5. On Z^+, where a ∗ b is a − b. 6. On R, where a ∗ b is $a\sqrt{b}$. 7. On R, where a ∗ b is the largest rational number that is less than ab. 8. On Z, where a ∗ b is 2a + b. In Exercises 9 through 19, determine whether the binary operation ∗ is commutative and whether it is associative on the set. 9. On Z^+, where a ∗ b is a + b + 2. 10. On Z, where a ∗ b is ab. 11. On R, where a ∗ b is a × |b|. 12. On the set of nonzero real numbers, where a ∗ b is a/b. 13. On R, where a ∗ b is the minimum of a and b. 14. On the set of n × n Boolean matrices, where A ∗ B is A ⊙ B (see Section 1.5). 15. On R, where a ∗ b is ab/3. 16. On R, where a ∗ b is ab + 2b. 17. On a lattice A, where a ∗ b is a ∨ b. 18. On the set of rational numbers, where a ∗ b is (a + b)/2. 19. On Z^+, where a ∗ b is 2^{a+b}. 20. Prove or disprove that the binary operation on Z^+ of a ∗ b = GCD(a, b) has the idempotent property. ... 9.2 Semigroups In this section we define a simple mathematical structure, consisting of a set together with a binary operation, that has many important applications. A semigroup is a nonempty set S together with an associative binary operation ∗ defined on S. We shall denote the semigroup by (S, ∗) or, when it is clear what the operation ∗ is, simply by S. We also refer to a ∗ b as the product of a and b. The semigroup (S, ∗) is said to be commutative if ∗ is a commutative operation. Example 1 It follows from Section 1.4 that (Z, +) is a commutative semigroup. ● Example 2 The set P(S), where S is a set, together with the operation of union is a commutative semigroup. ● Example 3 The set Z with the binary operation of subtraction is not a semigroup, since subtraction is not associative. ● Example 4 Let S be a fixed nonempty set, and let S^S be the set of all functions f : S → S. If f and g are elements of S^S, we define f ∗ g as f ◦ g, the composite function. Then ∗ is a binary operation on S^S, and it follows from Section 4.7 that ∗ is associative. Hence (S^S, ∗) is a semigroup. The semigroup S^S is not commutative. ● Example 5 Let (L, ≤) be a lattice. Define a binary operation on L by a ∗ b = a ∨ b. Then L is a semigroup. ● Example 6 Let A = {a1, a2, . . . , an} be a nonempty set. Recall from Section 1.3 that A^* is the set of all finite sequences of elements of A. That is, A^* consists of all words that can be formed from the alphabet A. Let α and β be elements of A^*. Observe that catenation is a binary operation · on A^*. Recall that if α = a1 a2 · · · an and β = b1 b2 · · · bk, then α · β = a1 a2 · · · an b1 b2 · · · bk. It is easy to see that if α, β, and γ are any elements of A^*, then α · (β · γ) = (α · β) · γ so that · is an associative binary operation, and (A^*, ·) is a semigroup. The semigroup (A^*, ·) is called the free semigroup generated by A. ● In a semigroup (S, ∗) we can establish the following generalization of the associative property; we omit the proof. THEOREM 1 If $a_1, a_2, \ldots, a_n$, n ≥ 3, are arbitrary elements of a semigroup, then all products of the elements $a_1, a_2, \ldots, a_n$ that can be formed by inserting meaningful parentheses arbitrarily are equal. ■ Example 7 Theorem 1 shows that the products $((a_1 \ast a_2) \ast a_3) \ast a_4$, $a_1 \ast (a_2 \ast (a_3 \ast a_4))$, $(a_1 \ast (a_2 \ast a_3)) \ast a_4$ are all equal. ● 350 Chapter 9 Semigroups and Groups If $a_1, a_2, \ldots, a_n$ are elements in a semigroup (S, ∗), we shall write their product as $a_1 \ast a_2 \ast \cdots \ast a_n$, omitting the parentheses. An element e in a semigroup (S, ∗) is called an identity element if e ∗ a = a ∗ e = a for all a ∈ S. As shown by Theorem 1, Section 1.6, an identity element must be unique. Example 8 The number 0 is an identity in the semigroup (Z, +). ● Example 9 The semigroup (Z^+, +) has no identity element. ● A monoid is a semigroup (S, ∗) that has an identity. Example 10 The semigroup P(S) defined in Example 2 has the identity ∅, since ∅ ∗ A = ∅ ∪ A = A = A ∪ ∅ = A ∗ ∅ for any element A ∈ P(S). Hence P(S) is a monoid. ● Example 11 The semigroup S^S defined in Example 4 has the identity 1_S, since $1_S \ast f = 1_S \circ f = f \circ 1_S = f \ast 1_S$ for any element f ∈ S^S. Thus, we see that S^S is a monoid. ● Example 12 The semigroup A^* defined in Example 6 is actually a monoid with identity Λ, the empty sequence, since α · Λ = Λ · α = α for all α ∈ A^*. ● Example 13 The set of all relations on a set A is a monoid under the operation of composition. The identity element is the equality relation Δ (see Section 4.7). ● Let (S, ∗) be a semigroup and let T be a subset of S. If T is closed under the operation ∗ (that is, a ∗ b ∈ T whenever a and b are elements of T), then (T, ∗) is called a subsemigroup of (S, ∗). Similarly, let (S, ∗) be a monoid with identity e, and let T be a nonempty subset of S. If T is closed under the operation ∗ and e ∈ T, then (T, ∗) is called a submonoid of (S, ∗). Observe that the associative property holds in any subset of a semigroup so that a subsemigroup (T, ∗) of a semigroup (S, ∗) is itself a semigroup. Similarly, a submonoid of a monoid is itself a monoid. Example 14 If T is the set of all even integers, then (T, ×) is a subsemigroup of the monoid (Z, ×), where × is ordinary multiplication, but it is not a submonoid since the identity of Z, the number 1, does not belong to T. ● Example 15 If (S, ∗) is a semigroup, then (S, ∗) is a subsemigroup of (S, ∗). Similarly, let (S, ∗) be a monoid. Then (S, ∗) is a submonoid of (S, ∗), and if T = {e}, then (T, ∗) is also a submonoid of (S, ∗). ● Suppose that (S, ∗) is a semigroup, and let a ∈ S. For n ∈ Z^+, we define the powers of a^n recursively as follows: $a^1 = a$, $a^n = a^{n-1} \ast a$, n ≥ 2. Moreover, if (S, ∗) is a monoid, we also define $a^0 = e$. It can be shown that if m and n are nonnegative integers, then $a^m \ast a^n = a^{m+n}$. Example 16 (a) If (S, ∗) is a semigroup, a ∈ S, and T = {a^i | i ∈ Z^+}, then (T, ∗) is a subsemigroup of (S, ∗). (b) If (S, ∗) is a monoid, a ∈ S, and T = {a^i | i ∈ Z^+ or i = 0}, then (T, ∗) is a submonoid of (S, ∗). ● Isomorphism and Homomorphism An isomorphism between two posets was defined in Section 6.1 as a one-to-one correspondence that preserved order relations, the distinguishing feature of posets. We now define an isomorphism between two semigroups as a one-to-one correspondence that preserves the binary operations. In general, an isomorphism between

structures of the same type should preserve the distinguishing features of the structures. Let (S, ∗) and (T, ∗) be two semigroups. A function f : S → T is called an isomorphism from (S, ∗) to (T, ∗) if it is a one-to-one correspondence from S to T , and if f (a ∗ b) = f (a) ∗ f (b) for all a and b in S. If f is an isomorphism from (S, ∗) to (T, ∗), then, since f is a one-to-one correspondence, it follows from Theorem 1 of Section 5.1 that f −1 exists and is a one-to-one correspondence from T to S. We now show that f −1 is an isomorphism from (T, ∗) to (S, ∗). Let a and b be any elements of T . Since f is onto, we can find elements a and b in S such that f (a) = a and f (b) = b . Then a = f −1 (a) and b = f −1 (b). Now f −1 (a ∗ b) = f −1 (f (a) ∗ f (b)) = f −1 (f (a ∗ b)) = (f −1 ◦ f)(a ∗ b) = a ∗ b = f −1 (a) ∗ f −1 (b). Hence f −1 is an isomorphism. We now say that the semigroups (S, ∗) and (T, ∗) are isomorphic and we write S ' T . To show that two semigroups (S, ∗) and (T, ∗) are isomorphic, we use the following procedure: Step 1 Step 2 Step 3 Step 4 Define a function f : S → T with Dom(f) = S. Show that f is one-to-one. Show that f is onto. Show that f (a ∗ b) = f (a) ∗ f (b). 352 Chapter 9 Semigroups and Groups Example 17 Let T be the set of all even integers. Show that the semigroups (Z, +) and (T, +) are isomorphic. Solution Step 1 We define the function f : Z → T by f (a) = 2a. Step 2 We now show that f is one to one as follows. Suppose that f (a1) = f (a2). Then 2a1 = 2a2 , so a1 = a2 . Hence f is one to one. Step 3 We next show that f is onto. Suppose that b is any even integer. Then a = b/2 ∈ Z and f (a) = f (b/2) = 2(b/2) = b, so f is onto. Step 4 We have f (a + b) = 2(a + b) = 2a + 2b = f (a) + f (b). Hence (Z, +) and (T, +) are isomorphic semigroups. ◆ In general, it is rather straightforward to verify that a given function f : S → T is or is not an isomorphism. However, it is generally more difficult to show that two semigroups are isomorphic, because one has to create the isomorphism f . As in the case of poset or lattice isomorphisms, when two semigroups (S, ∗) and (T, ∗) are isomorphic, they can differ only in the nature of their elements; their semigroup structures are identical. If S and T are finite semigroups, their respective binary operations can be given by tables. Then S and T are isomorphic if we can rearrange and relabel the elements of S so that its table is identical with that of T . Example 18 Let S = {a, b, c} and T = {x, y, z}. It is easy to verify that the following operation tables give semigroup structures for S and T , respectively. ∗ x y z ∗ a b c a a b c b b c a c c a b x z x y y x y z z y z x Let f (a) = y f (b) = x f (c) = z. Replacing the elements in S by their images and rearranging the table, we obtain ◆ exactly the table for T . Thus S and T are isomorphic. THEOREM 2 Let (S, ∗) and (T, ∗) be monoids with identities e and e , respectively. Let f : S → T be an isomorphism. Then f (e) = e . 9.2 Semigroups 353 Proof Let b be any element of T . Since f is onto, there is an element a in S such that f (a) = b. Then a =a∗e b = f (a) = f (a ∗ e) = f (a) ∗ f (e) = b ∗ f (e). Similarly, since a = e ∗ a, b = f (e) ∗ b. Thus for any b ∈ T , b = b ∗ f (e) = f (e) ∗ b, which means that f (e) is an identity for T . Thus since the identity is unique, it follows that f (e) = e . ■ If (S, ∗) and (T, ∗) are semigroups such that S has an identity and T does not, it then follows from Theorem 2 that (S, ∗) and (T, ∗) cannot be isomorphic. Example 19 Let T be the set of all even integers and let × be ordinary multiplication. Then the semigroups (Z, ×) and (T, ×) are not isomorphic, since Z has an identity and T ◆ does not. By dropping the conditions of one to one and onto in the definition of an isomorphism of two semigroups, we get another important method for comparing the algebraic structures of the two semigroups. Let (S, ∗) and (T, ∗) be two semigroups. An everywhere-defined function f : S → T is called a homomorphism from (S, ∗) to (T, ∗) if f (a ∗ b) = f (a) ∗ f (b) for all a and b in S. If f is also onto, we say that T is a homomorphic image of S. Example 20 Let A = {0, 1} and consider the semigroups (A∗ , ·) and (A, +), where · is the catenation operation and + is defined by the table + 0 1 0 0 1 1 1 0 Define the function f : A∗ → A by 1 if α has an odd number of 1’s f (α) = 0 if α has an even number of 1’s. It is easy to verify that if α and β are any elements of A∗ , then f (α·β) = f (α) + f (β). Thus f is a homomorphism. The function f is onto since f (0) = 0 f (1) = 1 but f is not an isomorphism, since it is not one to one. ◆ 354 Chapter 9 Semigroups and Groups The difference between an isomorphism and a homomorphism is that an isomorphism must be one to one and onto. For both an isomorphism and a homomorphism, the image of a product is the product of the images. The proof of the following theorem, which is left as an exercise for the reader, is completely analogous to the proof of Theorem 2. THEOREM 3 Let (S, ∗) and (T, ∗) be monoids with identities e and e , respectively. Let ■ f : S → T be a homomorphism from (S, ∗) onto (T, ∗). Then f (e) = e . Theorem 3 is a stronger, or more general, statement than Theorem 2, because it requires fewer (weaker) conditions for the conclusion. Theorem 3, together with the following two theorems, shows that, if a semigroup (T, ∗) is a homomorphic image of the semigroup (S, ∗), then (T, ∗) has a strong algebraic resemblance to (S, ∗). THEOREM 4 Let f be a homomorphism from a semigroup (S, ∗) to a semigroup (T, ∗). If S is a subsemigroup of (S, ∗), then f (S) = {t ∈ T | t = f (s) for some s ∈ S }, the image of S under f , is a subsemigroup of (T, ∗). Proof If t1 and t2 are any elements of f (S), then there exist s1 and s2 in S with t1 = f (s1) Then and t2 = f (s2). t1 ∗ t2 = f (s1) ∗ f (s2) = f (s1 ∗ s2) = f (s3), where s3 = s1 ∗ s2 ∈ S. Hence t1 ∗ t2 ∈ f (S). Thus f (S) is closed under the operation ∗ . Since the associative property ■ holds in T , it holds in f (S), so f (S) is a subsemigroup of (T, ∗). THEOREM 5 If f is a homomorphism from a commutative semigroup (S, ∗) onto a semigroup (T, ∗), then (T, ∗) is also commutative. Proof Let t1 and t2 be any elements of T . Then there exist s1 and s2 in S with t1 = f (s1) and t2 = f (s2). Therefore, t1 ∗ t2 = f (s1) ∗ f (s2) = f (s1 ∗ s2) = f (s2 ∗ s1) = f (s2) ∗ f (s1) = t2 ∗ t1 . Hence (T, ∗) is also commutative. ■ 9.2 Exercises 1. Let A = {a, b}. Which of the following tables define a semigroup on A? Which define a monoid on A? (b) ∗ a b (a) ∗ a b a a b a a b b a a b b b 2. Let A = {a, b}. Which of the following tables define a semigroup on A? Which define a monoid on A? (b) ∗ a b (a) ∗ a b a b a a a b b a b b b a 9.2 Semigroups 3. Let A = {a, b}. Which of the following tables define a semigroup on A? Which define a monoid on A? (b) ∗ a b (a) ∗ a b a a a a b b b b b a a In Exercises 4 through 16, determine whether the set together with the binary operation is a semigroup, a monoid, or neither. If it is a monoid, specify the identity. If it is a semigroup or a monoid, determine if it is commutative. 4. Z+ , where ∗ is defined as ordinary multiplication. 5. Z+ , where a ∗ b is defined as max{a, b}. 6. Z+ , where a ∗ b is defined as GCD{a, b}. 7. Z+ , where a ∗ b is defined as a. 8. The nonzero real numbers, where ∗ is ordinary multiplication. 9. P(S), with S a set, where ∗ is defined as intersection. 10. A Boolean algebra B, where a ∗ b is defined as a ∧ b. 11. S = {1, 2, 3, 6, 12}, where a ∗ b is defined as GCD(a, b). 12. S = {1, 2, 3, 6, 9, 18}, where a ∗ b is defined as LCM(a, b). 13. Z, where a ∗ b = a + b − ab. 14. The even integers, where a ∗ b is defined as ab . 2 15. The set of 2 × 1 matrices, where a c a+c ∗ = . b d b+d +1 16. The set of integers of the form 3k + 1, k ∈ Z+ , where ∗ is ordinary multiplication. 17. Does the following table define a semigroup or a monoid? ∗ a b c a c b a b b c b c a b c 18. Does the following table define a semigroup or a monoid? ∗ a b c a a c b b c b a c b a c 19. Complete the following table to obtain a semigroup. ∗ a b c a c a b a b b c b c a 355 20. Complete the following table so that it defines a monoid. ∗ a b c d a c b b d a c a b c d d b a 21. Let S = {a, b}. Write the operation table for the semigroup S S . Is the semigroup commutative? 22. Let S = {a, b}. Write the operation table for the semigroup (P(S), ∪). 23. Let A = {a, b, c} and consider the semigroup (A∗ , ·), where · is the operation of catenation. If α = abac, β = cba, and γ = babc, compute (b) γ ·(α·α) (c) (γ ·β)·α (a) (α·β)·γ 24. What is required for a subset of the elements of a semigroup to be a subsemigroup? 25. What is required for a subset of the elements of a monoid to be a submonoid? 26. Prove or disprove that the intersection of two subsemigroups of a semigroup (S, ∗) is a subsemigroup of (S, ∗). 27. Prove or disprove that the intersection of two submonoids of a monoid (S, ∗) is a submonoid of (S, ∗). 28. Let A = {0, 1}, and consider the semigroup (A∗ , ·), where · is the operation of catenation. Let T be the subset of A∗ consisting of all sequences having an odd number of 1’s. Is (T, ·) a subsemigroup of (A, ·)? 29. Let A = {a, b}. Are there two semigroups (A, ∗) and (A, ∗) that are not isomorphic? 30. An element x in a monoid is called an idempotent if x 2 = x ∗ x = x. Show that the set of all idempotents in a commutative monoid S is a submonoid of S. 31. Let (S1 , ∗1), (S2 , ∗2), and (S3 , ∗3) be semigroups and f : S1 → S2 and g : S2 → S3 be homomorphisms. Prove that g ◦ f is a homomorphism from S1 to S3 . 32. Let (S1 , ∗), (S2 , ∗), and (S3 , ∗) be semigroups, and let f : S1 → S2 and g : S2 → S3 be isomorphisms. Show that g ◦ f : S1 → S3 is an isomorphism. 33. Which properties of f are used in the proof of Theorem 2? 34. Explain why the proof of Theorem 1 can be used as a proof of Theorem 3. 35. Let R+ be the set of all positive real numbers. Show that the function f : R+ → R defined by f (x) = ln x is an isomorphism of the semigroup (R+ , ×) to the semigroup (R, +), where × and + are ordinary multiplication and addition, respectively. 36. Let (S, ∗) be a semigroup and A, a finite subet of S. Define Aˆ to be the set of all finite products of elements in A. (a) Prove that Aˆ is a subsemigroup of (S, ∗). (b) Prove that Aˆ is the smallest subsemigroup of (S, ∗) that contains A. 356 Chapter 9 Semigroups and Groups 9.3 Products and Quotients of Semigroups In this section we shall obtain new semigroups from existing semigroups. THEOREM 1 If (S, ∗) and (T, ∗) are semigroups, then (S × T, ∗) is a semigroup, where ∗ is defined by (s1 , t1) ∗ (s2 , t2) = (s1 ∗ s2 , t1 ∗ t2). Proof The proof is left as an exercise. ■ It follows at once from Theorem 1 that if S and T are monoids with identities e S and eT , respectively, then S × T is a monoid with identity (e S , eT). We now turn to a discussion of equivalence relations on a semigroup (S, ∗). Since a semigroup is not merely a set, we shall find that certain equivalence relations on a semigroup give additional information about the structure of the semigroup. An equivalence relation R on the semigroup (S, ∗) is called a congruence relation if a R a Example 1 and b R b imply (a ∗ b) R (a ∗ b). Consider the semigroup (Z, +) and the equivalence relation R on Z defined by a Rb if and only if a ≡ b (mod 2). Recall that we discussed this equivalence relation in Section 4.5. Remember that if a ≡ b (mod 2), then 2 | (a − b). We now show that this relation is a congruence relation as follows. If a ≡ b (mod 2) and c ≡ d (mod 2), then 2 divides a − b and 2 divides c − d, so a − b = 2m and c − d = 2n, where m and n are in Z. Adding, we have (a − b) + (c − d) = 2m + 2n or (a + c) − (b + d) = 2(m + n), so a + c ≡ b + d (mod 2). Hence the relation is a congruence relation. Example 2 ◆ Let A = {0, 1} and consider the free semigroup (A∗ , ·) generated by A. Define the following relation on A: αRβ if and only if α and β have the same number of 1’s. Show that R is a congruence relation on (A∗ , ·). Solution We first show that R is an equivalence relation. We have 1. α R α for any α ∈ A∗ . 9.3 Products and Quotients of Semigroups 357 2. If α R β, then α and β have the same number of 1’s, so β R α. 3. If α R β and β R γ , then α and β have the same number of 1’s and β and γ have the same number of 1’s, so α and γ have the same number of 1’s. Hence α R γ. We next show that R is a congruence relation. Suppose that α R α and β R β . Then α and α have the same number of 1’s and β and β have the same number of 1’s. Since the number of 1’s in α·β is the sum of the number of 1’s in α and the number of 1’s in β, we conclude that the number of 1’s in α·β is the same as the number of 1’s in α ·β . Hence (α·β) R (α ·β) ◆ and thus R is a congruence relation. Example 3 Consider the semigroup (Z, +), where + is ordinary addition. Let f (x) = x 2 − x − 2. We now define the following relation on Z: a Rb if and only if f (a) = f (b). It is straightforward to verify that R is an equivalence relation on Z. However, R is not a congruence relation since we have and but −1 R 2 since f (−1) = f (2) = 0 −2 R 3 since f (−2) = f (3) = 4 / (2 + 3) (−1 + (−2)) R since f (−3) = 10 and f (5) = 18. ◆ Recall from Section 4.5 that an equivalence relation R on the semigroup (S, ∗) determines a partition of S. We let [a] = R(a) be the equivalence class containing a and S/R denote the set of all equivalence classes. The notation [a] is more traditional in this setting and produces less confusing computations. THEOREM 2 Let R be a congruence relation on the semigroup (S, ∗). Consider the relation from S/R × S/R to S/R in which the ordered pair ([a], [b]) is, for a and b in S, related to [a ∗ b]. (a) is a function from S/R×S/R to S/R, and as usual we denote ([a], [b]) by [a] [b]. Thus [a] [b] = [a ∗ b]. (b) (S/R,) is a semigroup. Proof Suppose that ([a], [b]) = ([a], [b]). Then a R a and b R b , so we must have a ∗ b R a ∗ b , since R is a congruence relation. Thus [a ∗ b] = [a ∗ b]; that is, is a function. This means that is a binary operation on S/R. Next, we must verify that is an associative operation. We have [a] ([b] [c]) = [a] [b ∗ c] = [a ∗ (b ∗ c)] = [(a ∗ b) ∗ c] by the associative property of ∗ in S = [a ∗ b] [c] = ([a] [b]) [c]. 358 Chapter 9 Semigroups and Groups Hence S/R is a semigroup. We call S/R the quotient semigroup or factor semigroup. Observe that is a type of “quotient binary relation” on S/R that is constructed from the semigroup (G/R,) is a group, where the operation is defined in G/R by [a] [b] = [a ∗ b]. • Left coset a H of H in G determined by a: {ah | h ∈ H } • Normal subgroup: subgroup H such that a H = H a for all a in G 383 • Theorem: Let R be a congruence relation on a group G, and let H = [e], the equivalence class containing the identity. Then H is a normal subgroup of G and, for each a ∈ G, [a] = a H = H a. • Theorem: Let N be a normal subgroup of a group G, and let R be the relation defined on G by a R b if and only if a −1 b ∈ N . Then (a) R is a congruence relation on G. (b) N is the equivalence class [e] relative to R, where e is the identity of G. Ring (S, +, ∗): nonempty set S such that (S, +) is an Abelian group, ∗ is associative, and ∗ distributes over +. Commutative ring: ring in which the operation ∗ is commutative Ring with identity: ring with identity for multiplication Field: commutative ring with identity in which every nonzero element has a multiplicative inverse Theorem: The ring Zn is a field when n is a prime. Theorem: (a) If G = {g1 , g2 , . . . , gn } is a finite Abelian group with identity denoted by e, and a is any element of G, then a n = e. (b) (Fermat's Little Theorem) If p is a prime number and a is any integer, then a p ≡ a (mod p). Chapter 9 Self-Test 1. What does it mean to say a set is closed with respect to a binary operation? 2. How does an isomorphism of semigroups differ from an isomorphism of posets? How are an isomorphism of groups and an isomorphism of semigroups alike? 3. What are the properties that define a congruence relation? 4. Why are groups said to have more structure than semigroups? 5. How does a field differ from a ring? 6. For each of the following, determine whether the description of ∗ is a valid definition of a binary operation on the given set. (a) On the set of even integers, where a ∗ b = a + b (b) On Z+ , where a ∗ b = a + b (c) On Z+ , where a ∗ b = 2ab 7. Complete the table so that ∗ is a commutative and idempotent binary operation. 8. Let ∗ be defined on S = {a, b, c, d} by the table. 9. Determine whether the set together with the operation is a semigroup, a monoid, or neither for each of the pairs given in Exercise 1. 10. Let A = {0, 1} and consider the semigroup (A∗, ·). ... Semigroups and Groups 11. Let G be a group and define f : G → G by f (a) = a −1 . Is f a homomorphism? Justify your answer. 12. Let G be the group whose multiplication table is given below and let H be a subgroup of G ... subgroup of G 2 . 14. Let G be a group with identity e. Show that if x 2 = x for some x in G, then x = e. 15. Let G be the group ... Let H1 , H2 , . . . , Hk be subgroups of a group G. Prove k that Hi is also a subgroup of G. f f d c b a e i=1 √ 17. Prove that if n is an irrational number, then ... the group (G 2 , ∗2). If N is a normal sub- Experiment 9 The purpose of this experiment is to investigate relationships among groups, subgroups, and the structure of groups ... operation of composition. It is also the group of symmetries of a triangle. (See Section 9.4.) D is the group of symmetries of a square ... whose multiplication table is given in Table 2. You may find it helpful to write out the multiplication tables for S3 , D, and S4 ... 1 2 3 4 1. Identify the identity element e for each of the five groups. 2. For each of the five groups, do the following. For each element, find its order ... group? (The order of a group is the number of elements.) 4. For each of the five groups, find all subgroups of the group. 5. ... 7. The groups G 1 and D are both of order 8. Are they isomorphic? Explain your reasoning. CHAPTER 10 Languages and Finite-State Machines Prerequisites: Chapters 7 and 9 Whenever we tell a mathematically trained person or a computer to carry out this task. What language do we use to communicate with the computer? In the early days of computers, the language used was machine language, which was labor intensive, clumsy to use, and prone to clerical errors. It was soon realized that the ideal programming language is a natural language such as English, Spanish, or French. Although much research has been done in this area, it is still very difficult to have a computer understand everyday language. Computer scientists and linguists have developed the field of formal language theory to provide models that can be used to develop formal languages to communicate with a computer. In this chapter we introduce the study of formal languages and develop another mathematical structure, phrase structure grammars, a simple device for the construction of useful formal languages. We also examine several popular methods for representing these grammars. Finally, we provide an introduction to the notion of a finite-state machine, an abstract mathematical model of a computer that is able to recognize elements of a formal language. Looking Back The first computer program was written in 1843 by Ada Byron King, Countess of Lovelace (1815–1852), who was the daughter of Lord Byron, the great British poet. This computer program is contained in a translation and extensive supplement of a publication in Italian dealing with the operation of the analytical engine. This was a computer designed by Charles Babbage (1791–1871), a British mathematician who at an early age designed, but never built, a computer that shares many properties with computers being used today. The computer programming language Ada is named in honor of Lady Lovelace. Lady Lovelace 386 10.1 Languages 10.1 387 Languages In Section 1.3, we considered the set S ∗ consisting of all finite strings of elements from the set S. There are many possible interpretations of the elements of S ∗ , depending on the nature of S. If we think of S as a set of "words," then S ∗ may be regarded as the collection of all possible "sentences" formed from words in S. Of course, such "sentences" do not have to be meaningful or even sensibly constructed. We may think of a language as a complete specification, at least in principle, of three things. First, there must be a set S consisting of all "words" that are to be regarded as being part of the language. Second, a subset of S ∗ must be designated as the set of "properly constructed sentences" in the language. The meaning of this term will depend very much on the language being constructed. Finally, it must be determined which of the properly constructed sentences have meaning and what the meaning is. Suppose, for example, that S consists of all English words. The specification of a properly constructed sentence involves the complete rules of English grammar; the meaning of a sentence is determined by this construction and by the meaning of the words. The sentence "Going to the store John George to sing." is a string in S ∗ , but is not a properly constructed sentence. The arrangement of nouns and verbs phrases is illegal. On the other hand, the sentence "Noiseless blue sounds sit cross-legged under the mountaintop." is properly constructed, but completely meaningless. For another example, S may consist of the integers, the symbols +, −, ×, and ÷, and left and right parentheses. We will obtain a language if we designate as proper those strings in S ∗ that represent unambiguously parenthesized algebraic expressions. Thus ((3 − 2) + (4 × 7)) ÷ 9 and (7 − (8 − (9 − 10))) are properly constructed "sentences" in this language. On the other hand, the three strings (2 − 3)) + 4, 4 − 3 − 2, and)2 + (3 −) × 4 are not properly constructed. The first has too many parentheses, the second has too few (we do not know which subtraction to perform first), and the third has parentheses and numbers completely out of place. All properly constructed expressions have meaning except those involving division by zero. The meaning of an expression is the rational number it represents. Thus the meaning of ((2 − 1) ÷ 3) + (4 × 6) is 73/3, while 2 + (3 ÷ 0) and (4 + 2) − (0 ÷ 0) are not meaningful. The specification of the proper construction of sentences is called the syntax of a language. The specification of the meaning of sentences is called the semantics of a language. Among the languages that are of fundamental importance in computer science are the programming languages. These include BASIC, FORTRAN, JAVA, PASCAL, C++ , LISP, ADA, FORTH, and many other general and special-purpose languages. When they are taught to program in some programming language, people are actually taught the syntax of the language. In a compiled language such as FORTRAN, most mistakes in syntax are detected by the compiler, and appropriate error messages are generated. The semantics of a programming language forms a much more difficult and advanced topic of study. The meaning of a line of programming is taken to be the entire sequence of events that takes place inside the computer as a result of executing or interpreting that line. 388 Chapter 10 Languages and Finite-State Machines We will not deal with semantics at all. We will study the syntax of a class of languages called phrase structure grammars. Although these are not nearly complex enough to include natural languages such as English, they are general enough to encompass many languages of importance in computer science. This includes most aspects of programming languages, although the complete specification of some higher-level programming languages exceeds the scope of these grammars. On the other hand, phrase structure grammars are simple enough to be studied precisely, since the syntax is determined by substitution rules. The grammars that will occupy most of our attention lead to interesting examples of labeled trees. Grammars A phrase structure grammar G is defined to be a 4-tuple (V, S, v0 , →), where V is a finite set, S is a subset of V , v0 ∈ V − S, and → is a finite relation on V ∗ . The idea here is that S is, as discussed previously, the set of all allowed "words" in the language, and V consists of S together with some other symbols. The element v0 of V is a starting point for the substitutions, which will be discussed shortly. Finally, the relation → on V ∗ specifies allowable replacements, in the sense that, if w → w′ , then we may replace w by w′ whenever the string w occurs, either alone or as a substring of some other string. Traditionally, the statement w → w′ is called a production of G. Then w and w are termed the left and right sides of the production, respectively. We assume that no production of G has the empty string as its left side. We will call → the production relation of G. With these ingredients, we can introduce a substitution relation, denoted by ⇒, on V ∗ . We let x ⇒ y mean that x = l · w · r , y = l · w′ · r, and w → w′ , where l and r are completely arbitrary strings in V ∗ . In other words, x ⇒ y means that y results from x by using one of the allowed productions to replace part or all of x. The relation ⇒ is usually called direct derivability. Finally, we let ⇒∞ be the transitive closure of ⇒ (see Section 4.3), and we say that a string w in S ∗ is a syntactically correct sentence if and only if v0 ⇒∞ w. In more detail, this says that a string w is a properly constructed sentence if w is in S ∗ , not just in V ∗ , and if we can get from v0 to w by making a finite number of substitutions. This may seem complicated, but it is really a simple idea, as the following examples will show. If G = (V, S, v0 , →) is a phrase structure grammar, we will call S the set of terminal symbols and N = V − S the set of nonterminal symbols. Note that V = S ∪ N. The reader should be warned that other texts have slight variations of the definitions and notations that we have used for phrase structure grammars. Example 1 Let S = {John, Jill, drives, jogs, carelessly, rapidly, frequently}, N = {sentence, noun, verbphrase, verb, adverb}, and let V = S ∪ N . Let v0 = sentence, and suppose that the relation → on V ∗ is described by sentence |→ noun verbphrase noun |→ John noun |→ Jill verbphrase |→ verb adverb verb |→ drives verb |→ jogs adverb |→ carelessly adverb |→ rapidly adverb |→ frequently 10.1 Languages 389 The set S contains all the allowed words in the language; N consists of words that describe parts of sentences but that are not actually contained in the language. We claim that the sentence "Jill drives frequently," which we will denote by w, is an allowable or syntactically correct sentence in this language. To prove this, we consider the following sequence of strings in V ∗ . sentence noun jill jill jill verbphrase verbphrase verb drives drives adverb adverb frequently Now each of these strings follows from the preceding one by using a production to make a partial or complete substitution. In other words, each string is related to the following string by the relation ⇒, so sentence ⇒∞ w. By definition then, w is syntactically correct since, for this example, v0 is sentence. In phrase structure grammars, correct syntax simply refers to the process by which a sentence is formed, nothing else. ◆ It should be noted that the sequence of substitutions that produces a valid sentence, a sequence that will be called a derivation of the sentence, is not unique and is not necessarily obtained in the derivation given there. sentence noun noun noun Jill verbphrase verb adverb drives adverb drives frequently Example 1 is a very simple example. The set of all properly constructed sentences that can be produced using a grammar G is called the language of G and is denoted by L(G). The language of the grammar given in Example 1 is a somewhat simple-minded sublanguage of English, and it contains exactly 12 sentences, which the reader may verify that "John jogs carelessly" is in the language L(G) of this grammar, while "Jill frequently jogs" is not in L(G). It is also true that many different phrase structure grammars may produce the same language; that is, they have exactly the same set of syntactically correct sentences. Thus a grammar cannot be reconstructed from its language. In Section 10.2 we will give examples in which different grammars are used to construct the same language. Example 1 illustrates the process of derivation of a sentence in a phrase structure grammar. Another method that may sometimes be used to illustrate the derivation of a sentence is the construction of a derivation tree for the sentence. The starting symbol, v0 , is taken as the label for the root of this tree. The level-1 vertices correspond to and are labeled in order by the various words involved in the first substitution for v0 . Then the offspring of each vertex, at every succeeding level, are labeled by the various words (if any) that are substituted for that vertex the next time it is subjected to substitution. Consider, for example, the first derivation of sentence w in Example 1. Its derivation tree begins with "sentence," and the next-level vertices correspond to "noun" and "verbphrase" since the first substitution replaces the word "sentence" with the string "noun verbphrase." This part of the tree is 390 Chapter 10 Languages and Finite-State Machines shown in Figure 10.1(a). Next, we substitute "Jill" for "noun," and the tree becomes as shown in Figure 10.1(b). The next two substitutions, "verb adverb" for "verbphrase" and "drives" for "verb," extend the tree as shown in Figures 10.1(c) and (d). Finally, the tree is completed with the substitution of "frequently" for "adverb." The finished derivation tree is shown in Figure 10.1(e). sentence sentence noun verbphrase noun Jill (a) sentence verbphrase (b) sentence noun verbphrase noun verbphrase Jill verb adverb Jill verb adverb drives (c) sentence (d) sentence noun verbphrase Jill verb adverb drives frequently (e) Figure 10.1 The second derivation sequence, following Example 1 for the sentence w, yields a derivation tree in the stages shown in Figure 10.2 on page 391. Notice that the same tree results in both figures. Thus these two derivations yield the same tree, and the differing orders of substitution simply create the tree in different ways. The sentence being derived labels the leaves of the resulting tree. Example 2 Let V = {v0 , w, a, b, c}, S = {a, b, c}, and let |→ be the relation on V ∗ given by 1. v0 |→ aw 2. w |→ bbw 3. w |→ c Consider the phrase structure grammar G = (V, S, v0 , |→). To derive a sentence of L(G), it is necessary to perform successive substitutions, using (1), (2), and (3), until all symbols are eliminated other than the terminal symbols a, b, and c. Since we begin with the symbol v0 , we must first use production (1), or we could never eliminate v0 . This first substitution results in the string aw. We may now use (2) or (3) to substitute for w. If we use production (2), the result will contain a w. Thus one application of (3) to eliminate the string w is necessary (here we mean to use (3) to give the one production) and we can then use either (2) or (3) to substitute for w. Once we use (3), only terminal symbols remain, so the process ends. We may summarize this analysis by saying that L(G) is the subset of S ∗ corresponding to the regular expression a(bb)∗ c (see Section 1.3). Thus the word ab6 c is in the language of G, and its derivation tree is shown in Figure 10.3. Note that, unlike the tree of Example 1, the derivation tree for ab6 c is not a binary tree. 10.1 Languages 391 sentence sentence noun verbphrase noun verbphrase verb adverb (a) sentence (b) sentence noun verbphrase noun verbphrase verb adverb verb adverb frequently (c) drives frequently (d) sentence v0 noun verbphrase a w Jill verb adverb b b w drives frequently (e) Figure 10.2 b b w b b w c Figure 10.3 Example 3 Let V = {v0 , w, a, b, c}, S = {a, b, c}, and let |→ be a relation on V ∗ given by 1. v0 |→ av0 b 2. v0 b |→ bw 3. abw |→ c. Let G = (V, S, v0 , |→) be the corresponding phrase structure grammar. As we did in Example 2, we determine the form of allowable sentences in L(G). Since we must begin with the symbol v0 alone, we must use production (1) first. We may continue to use (1) any number of times, but we must eventually use production (2) to eliminate v0 . Repeated use of (1) will result in a string of the form a n v0 b n ; that is, there are equal numbers of a's and b's. When (2) is used, the result is a string of the form a m (abw)bm with m ≥ 0. At this point the only production that can be used is (3), and we must use it to remove the nonterminal symbol w. The use of (3) finishes the substitution process and produces a string in S ∗ . Thus the allowable sentences L(G) of the grammar G all have the form w = a n cb n , where n ≥ 0. In this case it can be shown that L(G) ◆ does not correspond to a regular expression over S. Another interesting feature of the grammar in Example 3 is that the derivations of the sentences cannot be done with derivation trees. Such derivation trees only work when the left-hand sides of all productions used consist of single, nonterminal symbols, as for example, the productions in Example 3 do not have this simple form. Although it is possible to construct a graphical representation of these derivations, we shall not pursue that topic here. Several restrictions can be placed on the form of productions in phrase structure grammars. Some grammars are thus of more special forms. A classification of phrase structure grammars 392 Chapter 10 Languages and Finite-State Machines Let G = (V, S, v0 , |→) be a phrase structure grammar. We say that G is TYPE 0: if no restrictions are placed on the productions of G TYPE 1: if for any production w1 |→ w2 , the length of w1 is less than or equal to the length of w2 (where the length of a string is the number of words in that string) TYPE 2: if the left-hand side of each production is a single, nonterminal symbol and the right-hand side consists of one or more symbols TYPE 3: if the left-hand side of each production is a single, nonterminal symbol and the right-hand side has one or more symbols, including at most one nonterminal symbol, which must be at the extreme right of the string In each of the preceding types, we permit the inclusion of the trivial production v0 |→ Λ, where Λ represents the empty string. This is an exception to the defining rule for types 1, 2, and 3, but it is included so that the empty string can be made part of the language. This avoids constant consideration of unimportant special cases. It follows from the definition that each type of grammar is a special case of the type preceding it. Example 1 is a type 2 grammar. Example 2 is type 3, and Example 3 is type 0. Grammars of types 0 or 1 are quite difficult to study and little is known about them. They include many pathological examples that are of no known practical use. We will restrict further consideration of grammars to types 2 and 3. These types have derivation trees for the sentences of their languages, and they are sufficiently complex to describe many aspects of actual programming languages. Type 2 grammars are sometimes called context-free grammars, since the symbols on the left of the productions are substituted for wherever they occur. On the other hand, a production of the type l · w · r |→ l · w′ · r (which could not occur in a type 2 grammar) is called context sensitive, since w is substituted for only in the context where it is surrounded by the strings l and r . Type 3 grammars are also called regular grammars. A language will be called type 2 or type 3 if there is a grammar of type 2 or type 3 that produces it. This concept can cause problems. Even if a language is produced by a non-type-2 grammar, it is possible that some type-2 grammar also produces this same language. In this case, the language is type 2. The same situation may arise in the case of type 3 grammars. The process we have considered in this section, namely deriving a sentence within a grammar, has a converse process. The converse process involves taking a sentence and verifying that it is syntactically correct in some grammar G by constructing a derivation tree that will produce it. This process is called parsing the sentence, and the resulting derivation tree is often called the parse tree of the sentence. Parsing is of fundamental importance for compilers and other language translators. A sentence in one language is parsed to show its structure, and a tree is then searched and, at each step, corresponding sentences are generated in another language. In this way a C++ program, for example, is compiled into a machine-language program. The contents of this section and the next two sections are essential to the compiling process, but the complete details must be left to a more advanced course. 10.1 Languages 393 10.1 Exercises In Exercises 1 through 6, a grammar G is specified. In each case describe precisely the language, L(G), produced by this grammar; that is, describe all syntactically correct "sentences." 1. G = (V, S, v0 , |→), V = {v0 , v1 , v2 , v3 , a, b, c}, S = {a, b, c} |→: v0 |→ av1 v1 |→ bv2 v2 |→ cv3 v3 |→ a 2. G = (V, S, v0 , |→), V = {v0 , a, b}, S = {a, b} |→: v0 |→ aav0 v0 |→ a v0 |→ b 4. G = (V, S, v0 , |→), V = {v0 , x, y, z}, S = {x, y, z} |→: v0 |→ xv0 v0 |→ yv0 v0 |→ z 5. G = (V, S, v0 , |→), V = {v0 , v1 , v2 , a, b}, S = {a, b} |→: v0 |→ av1 v1 |→ bv1 v1 |→ a v1 |→ av2 v2 |→ b 6. G = (V, S, v0 , |→), V = {v0 , v1 , x, y, z}, S = {x, y, z} |→: v0 |→ v0 v1 v0 v1 |→ v1 v1 |→ x v1 |→ y v1 |→ z 7. For each grammar in Exercises 1 through 6, state whether the grammar is type 0, type 1, type 2, or type 3. 8. Let G have the following productions: ... Let G = (V, S, v0 , |→) where V = {v0 , v1 , a, b} ... In Exercises 9 through 14, describe a grammar G that produces the given set of strings ... 15. Let G be the grammar of Exercise 3. Give two distinct derivations for the sentence in Exercise 15. ... Describe L(G). 394 Chapter 10 Languages and Finite-State Machines ... 20. Let G be a grammar ... Give a grammar ... Find a grammar for ... a palindrome is a string that reads the same from left to right as it does from right to left ... 21. Let G be the grammar ... (a) Is L(G) the set of strings of 0's and 1's ... (b) ... 22. Let G = (V, S, v0 , |→) be a phrase structure grammar ... Prove ... 10.2 Representations of Special Grammars and Languages BNF Notation For type 2 grammars (which include type 3 grammars), there are some useful, alternative methods of displaying the productions. A commonly encountered alternative is called the BNF notation (for Backus–Naur form). We know that the left-hand sides of all productions in a type 2 grammar are single nonterminal symbols. For any such symbol w, we combine all productions having w as the left-hand side. The symbol w remains on the left, and all right-hand sides associated with w are listed together, separated by the symbol |. The relational symbol |→ is replaced by the symbol ::=. Finally, the nonterminal symbols, wherever they occur, are enclosed in pointed brackets ⟨ ⟩. This has the additional advantage that nonterminal symbols may be permitted to have embedded spaces. Thus ⟨word1 word2⟩ shows that the string between the brackets is to be treated as one "word," not as two words. That is, we may use the space as a convenient and legitimate "letter" in a word, as long as we use pointed brackets to delimit the words. Example 1 In BNF notation, the productions of Example 1 of Section 10.1 appear as follows. ⟨sentence⟩ ::= ⟨noun⟩⟨verbphrase⟩ ⟨noun⟩ ::= John | Jill ⟨verbphrase⟩ ::= ⟨verb⟩⟨adverb⟩ ⟨verb⟩ ::= drives | jogs ⟨adverb⟩ ::= carelessly | rapidly | frequently ◆ Example 2 In BNF notation, the productions of Example 2 of Section 10.1 appear as follows. ⟨v0⟩ ::= a⟨w⟩ ⟨w⟩ ::= bb⟨w⟩ | c ◆ Note that the left-hand side of a production may also appear on one of the strings on the right-hand side. Thus, in the second line of Example 2, ⟨w⟩ appears on the left, and it appears in the string bb⟨w⟩ on the right. When this happens, we say that the corresponding production w |→ bbw is recursive. If a recursive production has w as left-hand side, we will say the production is normal if w appears only once on the right-hand side and is the rightmost symbol. Other nonterminal symbols may also appear on the right side. The recursive production w |→ bbw given in Example 2 is normal. Note that any recursive production that appears in a type 3 (regular) grammar is normal, by the definition of type 3. Example 3 BNF notation is often used to specify actual programming languages. PASCAL and many other languages had their grammars given in BNF initially. In this example, we consider a small subset of PASCAL's grammar. This subset describes the syntax of decimal numbers and can be viewed as a mini-grammar whose corresponding language consists precisely of all properly formed decimal numbers. Let S = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .}. Let V be the union of S with the set N = {decimal-number, decimal-fraction, unsigned-integer, digit}. Then let G be the grammar with symbol sets V and S, with starting symbol "decimal-number" and with productions given in BNF form as follows: 1. ⟨decimal-number⟩ ::= ⟨unsigned-integer⟩ | ⟨decimal-fraction⟩ | ⟨unsigned-integer⟩⟨decimal-fraction⟩ 2. ⟨decimal-fraction⟩ ::= .⟨unsigned-integer⟩ 3. ⟨unsigned-integer⟩ ::= ⟨digit⟩ | ⟨digit⟩⟨unsigned-integer⟩ 4. ⟨digit⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 Figure 10.4 shows a derivation tree, in this grammar, for the decimal number 23.14. Notice that the BNF statement numbered 3 is recursive in the second part of its right-hand side. That is, the production "unsigned-integer |→ digit unsigned-integer" is recursive, and it is also normal. In general, we know that many different grammars may produce the same language. If the line numbered 3 were replaced by the line 396 Chapter 10 Languages and Finite-State Machines decimal-number unsigned-integer decimal-fraction digit unsigned-integer . unsigned-integer 2 digit digit unsigned-integer 3 1 digit 4 Figure 10.4 3. ⟨unsigned-integer⟩ ::= ⟨digit⟩ | ⟨unsigned-integer⟩⟨digit⟩ we would have a different grammar that produced exactly the same language, namely the correctly formed decimal numbers. However, this grammar contains a production that is recursive but not normal. ◆ Example 4 As in Example 3, we give a grammar that specifies a piece of several actual programming languages. In these languages, an identifier (a name for a variable, function, subroutine, and so on) must be composed of letters and digits and must begin with a letter. The following grammar, with productions given in BNF, has precisely these identifiers as its language. G = (V, S, identifier, |→), N = {identifier, remaining, digit, letter} = {a, b, c, . . . , z, 0, 1, 2, 3, . . . , 9}, V = N ∪ S ⟨identifier⟩ ::= ⟨letter⟩ | ⟨letter⟩⟨remaining⟩ ⟨remaining⟩ ::= ⟨letter⟩ | ⟨digit⟩ | ⟨letter⟩⟨remaining⟩ | ⟨digit⟩⟨remaining⟩ ⟨letter⟩ ::= a | b | c | · · · | z ⟨digit⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ◆ Again we see that the productions "identifier |→ letter remaining" and "remaining |→ digit remaining" occurring in BNF statement 2 are recursive and normal. Syntax Diagrams A second alternative method for displaying the productions in some type 2 grammars is the syntax diagram. This is a pictorial display of the productions that allows the user to view the substitutions dynamically, that is, to view them as movement through the diagram. We will illustrate, in Figure 10.5, the diagrams that result from translating typical sets of productions, usually all the productions appearing on the right-hand side of some BNF statement. A BNF statement that involves just a single production, such as ⟨w⟩ ::= ⟨w1⟩⟨w2⟩⟨w3⟩, will result in the diagram shown in Figure 10.5(a). The symbols (words) that make up the right-hand side of the production are drawn in sequence from left to right. The arrows indicate the direction in which to move to accomplish a substitution, while the label w indicates that we are substituting for the symbol w. Finally, the rectangles enclosing w1 , w2 , and w3 denote the fact that these are nonterminal symbols. If terminal symbols were present, they would instead be enclosed in circles or ellipses. Figure 10.5(b) shows the situation when there are several productions with the same left-hand side. This figure is a syntax diagram translation of the following BNF specification: ⟨w⟩ ::= w1 w2 | w1 w3 | w1 w4 ... 10.2 Representations of Special Grammars and Languages Figure 10.5 these are nonterminal symbols. Here the diagram shows that we can substitute for w, by moving through the figure in the direction of the arrows, we may take any one of three paths. This corresponds to the three alternative substitutions for the symbol w. Now consider the following normal, recursive production, in BNF form: ⟨w⟩ ::= ab⟨w⟩. The syntax diagram for this production is shown in Figure 10.5(c). If we go through the loop once, we encounter a, then b, and we then return to the starting point designated by w. This represents the recursive substitution of abw for w. Several trips around the loop represent several successive substitutions. Thus, if we traverse the loop three times and return to the starting point, we see that w will be replaced by abababw in three successive substitutions. This is typical of the way in which movement through a syntax diagram represents the substitution process. The preceding remarks show how to construct a syntax diagram for a normal recursive production. Nonnormal recursive productions do not lead to the simple diagrams discussed, but we may sometimes replace nonnormal, recursive productions by normal recursive productions and obtain a grammar that produces the same language. Since recursive productions in regular grammars are all normal, syntax diagrams can always be used to represent regular grammars. We also note that syntax diagrams for a language are by no means unique. They will not only change when different, equivalent productions are used, but they may be combined and simplified in a variety of ways. Consider the following BNF specification: ⟨w⟩ ::= ab | ab⟨v⟩ ... We can construct the syntax diagram for w by ... In Figure 10.5(d), we have shown the diagram that contains the recursive production. If the symbol v is eliminated, or is pass through the upper path, for example, we pass through ... the entire diagram and returns ... at w. ... We consider the loop for w, we will, any number of times. Thus we would have ... the syntax diagram that eventually results in the complete elimination 398 Chapter 10 Languages and Finite-State Machines substitutions that are equivalent (the path in Figure 10.5(e)), produced by combining the paths of Figure 10.5(d). These types of simplifications are performed wherever possible. Example 5 The syntax diagrams of Figure 10.6(a) represent the BNF statements of Example 2, constructed with our original rules for drawing syntax diagrams. A slightly more aesthetic version is shown in Figure 10.6(b). ◆ v0 w a w a w c w b b (a) v0 a c b b (b) Figure 10.6 Example 6 Consider the BNF statements 1, 2, 3, and 4 of Example 4. The direct translation into syntax diagrams is shown in Figure 10.7. In Figure 10.8 we combine the first two diagrams of Figure 10.7 and simplify the result. We thus eliminate the symbol "remaining," and we arrive at the customary syntax diagrams for identifiers. ◆ identifier letter identifier letter remaining remaining letter remaining digit remaining letter ... digit 0 1 ... 9 Figure 10.7 identifier letter letter digit Figure 10.8 10.2 Representations of Special Grammars and Languages 399 Example 7 The productions of Example 3, for well-formed decimal numbers, are shown in syntax diagram form in Figure 10.9. Figure 10.10 shows the result of substituting the diagram for "unsigned-integer" into the diagram for "decimal-number" and "decimal-fraction." In Figure 10.11 we continue this process of substitution to arrive at a single diagram. However, it is not nearly done. It is important to realize that we lose some ... decimal-number unsigned-integer decimal-fraction unsigned-integer decimal-fraction digit digit ... Figure 10.9 ... Figure 10.10 ... we have elaborated the diagram to eliminate the symbol "digit" so the entire diagram would contain only terminal symbols. ... Figure 10.11 If we were to take the extreme case of Figure 10.11 into one large diagram, that diagram would contain only terminal symbols ... Regular Grammars and Regular Expressions There is a close connection between the language of a regular grammar and a regular expression (see Section 1.3). We state the following theorem without proof. 400 Chapter 10 Languages and Finite-State Machines THEOREM 1 Let S be a finite set, and L ⊆ S ∗ . Then L is a regular set if and only if L = L(G) for some regular grammar G = (V, S, v0 , |→). ■ Theorem 1 tells us that the language L(G) of a regular grammar G must be the set corresponding to some regular expression over S, but it does not tell us how to find such a regular expression. If the relation |→ of G is specified in BNF or syntax diagram form, we may compute the regular expression desired in a reasonably straightforward way. Suppose, for example, that G = (V, S, v0 , |→) and that |→ is specified by a set of syntax diagrams. As we previously mentioned, it is possible to combine all the syntax diagrams into one large diagram that represents v0 and involves only terminal symbols. We will call the result the master diagram of G. Terminal symbols of the diagram correspond to themselves, as regular expressions. The master diagram of G ... sequences of parts, and regular expressions ... 1. Terminal symbols of the diagram correspond to themselves, as regular expressions. 2. If a segment D of the diagram is composed of two sequential segments D1 and D2 in sequence, as shown in Figure 10.12(a), and if D1 and D2 correspond to regular expressions α1 and α2 , respectively, then D corresponds to α1 · α2 . 3. If a segment D of the diagram is composed of alternative segments D1 and D2 , as shown in Figure 10.12(b), and if D1 and D2 correspond to regular expressions α1 and α2 , respectively, then D corresponds to α1 ∨ α2 . 4. If a segment D of the diagram is a loop through a segment D1 , as shown in Figure 10.12(c), and if D1 corresponds to the regular expression α, then D corresponds to α ∗ . D1 D2 D2 (a) D (b) D1 (c) Figure 10.12 Rules 2 and 3 extend to any finite number of segments Di of the diagram. Using the foregoing rules, we may construct the single regular expression that corresponds to the master diagram as a whole. This expression is the regular expression corresponding to L(G). Example 8 Consider the diagram in Figure 10.13(a). It is composed of three alternative segments, the first corresponding to the expression a, the second to the expression b, and the third, a loop, corresponding to the expression c ∗ . Thus the entire diagram corresponds to the regular expression a ∨ b ∨ c ∗ . The diagram shown in Figure 10.13(b) is composed of three sequential segments. The first segment is itself composed of two alternative subsegments, and it corresponds to the regular expression c, and the third component, a loop, corresponds to the regular expression d ∗ . Thus the overall diagram corresponds to the regular expression (a ∨ b)cd ∗ . Finally, consider the syntax diagram shown in Figure 10.13(c). This is one large loop through a segment that corresponds to the regular expression (a ∨ bc) ∗ . Thus the entire diagram corresponds to the regular expression (a ∨ bc)∗ . ◆ Figure 10.13 Example 9 Consider the grammar G given in BNF in Example 2. Syntax diagrams for this grammar were discussed in Example 5 and shown in Figure 10.6(b). If we substitute the diagram for w into the diagram for v0 , we get Figure 10.14, the master diagram for G. This is a more typical diagram. We see by inspection that this diagram corresponds to the regular expression a(bb)∗ c, so L(G) is the set of regular expressions, as we found in Example 2 of Section 10.1. ◆ Example 10 Consider the grammar G of Examples 4 and 6. Then L(G) is the set of legal identifiers, whose syntax diagram is shown in Figure 10.8. In Figure 10.14 we show the master diagram that results from combining the diagrams of Figure 10.7. It follows that a regular expression corresponding to L(G) is (a ∨ b ∨ · · · ∨ z)(a ∨ b ∨ · · · ∨ z ∨ 0 ∨ 1 ∨ · · · ∨ 9)∗ . ◆ The type of diagram arguments illustrated in Figure 10.14 can be combined to produce syntax diagrams for any regular grammar. Thus we may always proceed as just illustrated to find the corresponding regular expression. With practice one can learn to compute this expression directly from multiple syntax diagrams or BNF, thus avoiding the need to make a master diagram. In any event, complex cases may prove too cumbersome for hand analysis. Chapter 10 Languages and Finite-State Machines Figure 10.14 Figure 10.15 10.2 Exercises In Exercises 1 through 5, give the BNF notation for the productions of the grammar in the specified exercise of Section 10.1. 1. Exercise 1 2. Exercise 2 3. Exercise 4 4. Exercise 5 5. Exercise 6 ... 6. Give the BNF for the productions of Exercise 9 of Section 10.1. 7. Give the BNF for the productions of Exercise 10 of Section 10.1. 8. Give the BNF for the productions of Exercise 12 of Section 10.1. 9. Give the BNF for the productions of Exercise 13 of Section 10.1. 10. Let G = (V, S, v0 , |→) where V = {v0 , v1 , a, b} ... Give the BNF representation for the syntax diagram shown. The symbols a, b, c, and d are terminal symbols of some grammar. You may provide nonterminal symbols as needed. In addition to v0 , you are to use in the BNF productions. You may use several BNF statements if needed. 11. v0 a b d v1 a v2 a 12. v0 a b c 13. v0 a v1 b c ... 14. v0 v1 b ... Figure 10.16 10.17 14. v0 ... 15. v0 ... 16. v0 ... In Exercises 17 through 20, give the syntax diagram of the productions specified ... 17. 18. 19. 20. ... In each of Exercises 21 through 25, give a BNF representation for the grammar of the language L(G). 21. (a ∨ b)c ... 26. Exercise 2 of Section 10.1 27. Exercise 3 of Section 10.1 28. Exercise 5 of Section 10.1 29. Exercise 6 of Section 10.1 ... In each of Exercises 30 through 36, find the regular expression that corresponds to the syntax diagram of Exercise 13. In Exercises 14, find the regular expression that corresponds to the syntax diagram of Exercise 14. ... Find the regular expression corresponding to the grammar, and ... 10.3 Finite-State Machines We think of a machine as a system that can accept input, possibly produce output, and have some sort of internal memory that can keep track of certain information about previous inputs. The complete internal condition of the machine and all of its memory, at any particular time, is said to constitute the state of the machine at that time. The state in which a machine finds itself at any instant summarizes its memory of past inputs and determines how it will react to subsequent input. When more input arrives, the given state of the machine determines (with the input) the next state to be occupied, and any output that may be produced. If the number of states is finite, the machine is a finite-state machine. Suppose that we have a finite set S = {s0 , s1 , . . . , sn }, a finite set I , and for each x ∈ I , a function f x : S → S. Let F = { f x | x ∈ I }. The triple (S, I , F) is called a finite-state machine, S is called the state set of the machine, and the elements of S are called states. The set I is called the input set of the machine, and F is the state transition function. Thus, if the machine is in state si and receives input x, it moves to state f x (si). Chapter 10 Languages and Finite-State Machines Since the set S and I are finite, there is a function f : S × I → S given by f (s, x) = f x (s). The individual functions f x can all be recovered from a knowledge of F. Many authors use a function F : S × I → S, instead of a set { f x | x ∈ I }, to define a finite-state machine. The definitions are completely equivalent. Example 1 Let S = {s0 , s1 } and I = {0, 1}. Define f 0 and f 1 as follows: f 0 (s0) = s0 , f 0 (s1) = s1 , f 1 (s0) = s1 , f 1 (s1) = s0 . This finite-state machine has two states, s0 and s1 , and accepts two possible inputs, 0 and 1. The input 0 leaves each state fixed, and the input 1 reverses states. ◆ We can think of this machine as a model for a circuit (or logical) device and visualize such a device as in Figure 10.21. The output signals will, at any given time, consist of two voltages, one higher than the other. Either line 1 will be at the higher voltage and line 2 at the lower, or the reverse. The first set of output conditions will be denoted s0 and the second will be denoted s1 . An input pulse, represented by the symbol 1, will reverse output voltages. The symbol 0 represents the absence of an input pulse and so results in no change of output. This device is often called a T flip-flop and is a concrete realization of the machine in this example. Line 1 Output signal Input signal Line 2 Figure 10.21 s0 s1 s1 s0 Figure 10.22 We summarize this machine in Figure 10.22. The table shown there lists the states down the side and inputs across the top. The column under each input gives the values of the function corresponding to that input at each state shown on the left. The arrangement illustrated in Figure 10.22 for summarizing the effect of inputs on states is called the state transition table of the finite-state machine. It can be used with any machine of reasonable size and is a convenient method of specifying the machine. Example 2 s0 a b a s0 s1 s2 s2 Figure 10.23 Figure 10.24 Here a and b are the possible inputs, and there are three states, s0 , s1 , and s2 . The table shows us that f a (s0) = s0 , f a (s1) = s2 , f a (s2) = s1 , and f b (s0) = s1 ... ◆ If M is a finite-state machine with states S, inputs I , and state transition functions { f x | x ∈ I }, we can determine a relation R M on S in a natural way. If si , s j ∈ S, we say that si R M s j if there is an input x so that f x (si) = s j . Thus si R M s j means that if the machine is in state si , there is some input x ∈ I that, if received next, will put the machine in state s j . The relation R M permits us to describe the machine M as a labeled digraph of the relation R M on S, where each edge is labeled by the set of all inputs that cause the machine to change states as indicated by that edge. Example 3 Consider the machine of Example 2. Figure 10.24 shows the digraph of the relation R M , with each edge labeled appropriately. Notice that the entire structure of M can be recovered from this digraph, since edges and their labels indicate where each input sends each state. ◆ Example 4 Consider the machine M whose state table is shown in Figure 10.25(a). The digraph of R M is then shown in Figure 10.25(b), with edges labeled appropriately. ◆ Figure 10.25 a s0 s1 s2 s3 b c s0 ... Note that an edge may be labeled by more than one input, since several inputs may cause the same change of state. The reader will observe that every input must be part of the label of exactly one edge out of each state. This is a general property that holds for the labeled digraphs of all finite-state machines. For brevity, we will refer to the labeled digraph of a machine M simply as the digraph of M. It is possible to add a variety of extra features to a finite-state machine in order to increase the utility of the concept. A simple, yet very useful extension results in what is often called a Moore machine, or recognition machine, which is defined as a sequence (S, I , F, s0 , T), where (S, I , F) constitutes a finite-state machine, s0 ∈ S and T ⊆ S. The state s0 is called the starting state of M, and it will be used to represent the condition of the machine before it receives any input. The set T is called the set of acceptance states of M. These states will be used in Section 10.4 in connection with language recognition. When the digraph of a Moore machine is drawn, the acceptance states are indicated with two concentric circles, instead of one. No special notation will be used on these digraphs for the starting state, but unless otherwise specified, this state will be named s0 . Example 5 Let M be the Moore machine (S, I , F, s0 , T), where (S, I , F) is the finite-state machine of Figure 10.25 and T = {s1 , s3 }. Figure 10.26 shows the digraph of M. ◆ Chapter 10 Languages and Finite-State Machines 406 Machine Congruence and Quotient Machines Let M = (S, I , F) be a finite-state machine, and suppose that R is an equivalence relation on S. We say that R is a machine congruence on M if, for any s, t ∈ S, s R t implies that f x (s) R f x (t) for all x ∈ I . In other words, R is a machine congruence if R-equivalent states are always mapped to R-equivalent states by every input in I . If R is a machine congruence on M, we let S̄ = S/R be the partition of S corresponding to R (see Section 4.5). Then S̄ = {[s] | s ∈ S}. For any input x ∈ I , consider the relation f̄ x on S̄ defined by f̄ x = {([s], [f x (s)])}. If [s] = [t], then s R t, so f x (s) R f x (t); hence [f x (s)] = [f x (t)]. This shows that the relation f̄ x is a function from S̄ to S̄, and we may write f̄ x ([s]) = [f x (s)] for all equivalence classes [s] in S̄. If we let F̄ = { f̄ x | x ∈ I }, then the triple M̄ = (S̄, I , F̄) is a finite-state machine called the quotient of M corresponding to R. We will use the notation M/R for M̄. Generally, a quotient machine will be simpler than the original machine. We will show in Section 10.6 that it is often possible to find a simpler quotient machine that will replace the original machine for certain purposes. Example 6 a b s0 s0 s4 s1 s0 s2 s2 s4 s3 s5 s2 s4 s3 s5 s2 s1 s0 s0 Figure 10.27 Let M be the finite-state machine whose state transition table is shown in Figure 10.27. Then S = {s0 , s1 , s2 , s3 , s4 , s5 }. Let R be the equivalence relation on S whose matrix is ... M R = [1 1 0 0 0 0; 1 1 0 0 0 0; 0 0 1 1 0 0; 0 0 1 1 0 0; 0 0 0 0 1 1; 0 0 0 0 1 1] Then we have S/R = {[s0], [s2], [s4]}, where [s0] = {s0 , s1 }, [s2] = {s2 , s3 }, and [s4] = {s4 , s5 }. We check that R is a machine congruence. The state transition table in Figure 10.27 shows that f a takes each element of [s0] to an element of [s0], each element of [s2] to an element of [s4], and each element of [s4] to an element of [s2] ... Also, f b takes each element of [s0] to an element of [s4], ... Thus R is a machine congruence, and the state transition table of the quotient machine M/R is shown in Figure 10.28. ◆ [s0] [s2] [s4] Figure 10.28 Example 7 Let M be the Moore machine whose digraph is shown in Figure 10.29. ... Then S = {s0 , s1 , s2 , s3 , s4 , s5 , s6 , s7 }. Then it is easily checked, from the digraph of Figure 10.29, that R is a machine congruence. To obtain the digraph of the quotient machine M/R, draw a vertex for each equivalence class, [s0] = {s0 , s4 }, [s1] = {s1 , s2 , s5 }, [s3] = {s3 , s7 }, and construct an edge from [si] to [sj] if there is, in the original digraph, an edge from some vertex in [si] to some vertex in [sj] ... Figure 10.29 Figure 10.30 Moore machine, if the equivalence classes of the acceptance states ... the starting state is [s0] ... the result ... Suppose that M = (S, I , F, s0 , T) is a Moore machine, and that R is a machine congruence on M. Then we can define a Moore machine M/R ... [s0] ... T̄ = {[t] | t ∈ T } ... Moore machine. ◆ The resulting quotient Moore machine is shown in Figure 10.31. ◆ 10.3 Exercises In Exercises 1 through 6, draw the digraph of the machine whose state transition table is shown. Remember to label the edges with the appropriate inputs. 1. ... 2. ... 3. ... 4. ... In Exercises 7 through 12 (Figures 10.32 through 10.37), construct the state transition table of the finite-state machine whose digraph is shown. 7. 8. 9. 10. 11. 12. ... 13. Let I = {0, 1} ... Consider the finite-state machine M = (S, I , F) ... 14. ... 15. Let M = (S, I , F) be a finite-state machine ... Show that R is a machine congruence on M ... 16. ... 17. Consider the finite-state machine whose state transition table is ... Show that R is an equivalence relation ... 18. ... Construct the state transition table for the corresponding quotient machine. 19. ... Let M be the Moore machine ... s0 s1 s2 s3 s4 s5 s6 Show that R is a machine congruence on M. (b) Construct the digraph for the corresponding quotient machine. 20. Consider the Moore machine whose digraph is shown in Figure 10.38 ... Show that the relation R on S whose matrix is ... is a machine congruence. (b) Draw the digraph of the corresponding quotient Moore machine. 21. Consider the Moore machine whose digraph is shown in Figure 10.40. Show that the relation R on S whose matrix is ... (b) Draw the digraph of the corresponding quotient Moore machine. 22. Let I be a set ... Let M = (S, I , F) ... Show that ... 24. Let M = (S, I , F) be a finite-state machine. Define a relation R on I as follows ... 25. ... 10.4 Monoids, Machines, and Languages Let M = (S, I , F) be a finite-state machine with state set S = {s0 , s1 , . . . , sn }, input set I , and state transition function F. We will associate with M two monoids. First, there is the monoid I ∗ of all finite sequences of elements from I , with the operation of catenation and with identity Λ, the empty string. Second, we have the monoid S S , of all functions from S to S, with the operation ◦ of composition, and with the identity function 1 S . Now let w = x1 x2 · · · xn be a string in I ∗ . Define f w : S → S by f w = f xn ◦ f xn−1 ◦ · · · ◦ f x1 . Also, we define f Λ to be 1 S . In this way we associate a function f w ∈ S S with each element w ∈ I ∗ . THEOREM 1 If w1 and w2 are in I ∗ , then f w1 ·w2 = f w2 ◦ f w1 . Proof ... 10.4 Monoids, Machines, and Languages ... The reader should note that, as in the proof of Theorem 1, if w = x1 x2 · · · xn , then f w (s) is the state that results from applying ... If M is in state s and the inputs x1 , x2 , . . . , xn are received in succession, then M will be in state f w (s) ... We can therefore think of f w (s) as the effect of the string w on state s. ... Example 1 Let M be the finite-state machine ... Consider the inputs ... For example ... the machine M in state s0 will result in state ... Figure 10.41 ... THEOREM 2 Let M = (S, I , F) be a finite-state machine. Then M is a machine of the form ... M = {f w | w ∈ I ∗ } is a submonoid of S S . Proof ... ■ The monoid M is called the monoid of the machine M. ... Example 2 ... Consider the machine M whose digraph is shown in Figure 10.42. Compute f aab , f aba , ... Example 3 ... Let M be the finite-state machine ... Figure 10.43 ... Thus ... We now consider a Moore machine M = (S, I , F, s0 , T) ... language ... A string w ∈ I ∗ is accepted by M if f w (s0) ∈ T ... The set of all strings accepted by M is the language accepted by M, denoted L(M) ... Example 4 Consider the Moore machine M whose digraph is shown in Figure 10.44 ... Here state s0 is the starting state and T = {s2 }. What strings are accepted by M? ... Figure 10.44 ... Thus L(M) is exactly the set of all strings with 3n + 1 ones ... Example 5 ... Figure 10.45 ... Example 6 Consider the Moore machine M whose digraph is shown in Figure 10.46 ... s0 , s1 , s2 , s3 ... Thus L(M) = ... ◆ THEOREM 3 Let I be a set and L ⊆ I ∗ . Then L is a type 3 language, that is, L = L(G), where G is a type 3 grammar having I as its set of terminal symbols, if and only if L = L(M) for some Moore machine M = (S, I , F, s0 , T). ■ We stated in Section 10.2 that a set L ⊆ I ∗ is a type 3 language if and only if L is a regular set, that is, if and only if L corresponds to some regular expression over I . This leads to the following corollary of Theorem 1. Corollary 1 Let I be a set and L ⊆ I ∗ . Then L = L(M) for some Moore machine M = (S, I , F, s0 , T) if and only if L is a regular set. ■ We stated in Section 10.2 that a set L ⊆ I ∗ is a type 3 language if and only if L is a regular set ... However, it is easy to give a complete and detailed proof of Theorem 1. However, it is easy to give a construction that produces a type 3 grammar from a given Moore machine. This is done in such a way that the grammar and the machine have the same language. Let M = (S, I, F, s0 , T) be a given Moore machine. We construct a type 3 grammar G = (V, I, s0 ,)→) as follows. Let V = I ∪ S; that is, I will be the set of terminal symbols for G, while S will be the set of nonterminal symbols. Let si and s j be in S, and x ∈ I . We write si) → xs j , if f x (si) = s j , that is, if the input x takes state si to s j . We also write si) → x if f x (si) ∈ T , that is, if the input x takes the state si to some acceptance state. Now let)→ be the relation determined by the preceding two conditions and take this relation as the production relation of G. The grammar G constructed previously has the same language as M. Suppose, for example, that w = x1 x2 x3 ∈ I ∗ . The string w is in L(M) if and only if f w (s0) = f x3 (f x2 (f x1 (s0))) ∈ T . Let a = f x1 (s0), b = f x2 (a), and c = f x3 (b), where c = f w (s0) is in T . Then the rules given for constructing)→ tell us that 1. s0) → x1 a 2. a)→ x2 b 3. b) → x3 are all productions in G. The last one occurs because c ∈ T . If we begin with s0 and substitute, using (1), (2), and (3) in succession, we see that s0 ⇒∗ x1 x2 x3 = w (see Section 10.1), so w ∈ L(G). A similar argument works for any string in L(M), so L(M) ⊆ L(G). If we reverse the argument, we can see that we also have L(G) ⊆ L(M). Thus M and G have the same language. Example 1 Consider the Moore machine M shown in Figure 10.44. Construct a type 3 grammar G such that L(G) = L(M). Also, find a regular expression over I = {a, b} that corresponds to L(M). 10.5 Machines and Regular Languages 415 Solution Let I = {a, b}, S = {s0 , s1 , s2 }, and V = I ∪ S. We construct the grammar (V, I, s0 ,) →), where) → is described as follows:) →: s0 s0 s1 s1 s2) → as0) → bs1) → as1) → bs2) → as2 . s2 s1 s2 s2) → bs2)→ b)→ a)→ b The production relation) → is constructed as we indicated previously; therefore, L(M) = L(G). If we consult Figure 10.44, we see that a string w ∈ L(M) has the following properties. Any number n ≥ 0 of a’s can occur at the beginning of w. At some point, a b must occur in order to cause the transition from s0 to s1 . After this b, any number k ≥ 0 of a’s may occur, followed by another b to cause transition to s2 . The remainder of w, if any, is completely arbitrary, since the machine cannot leave s2 after once entering this state. From this description we can readily see that L(M) corresponds to the regular expression a ∗ ba ∗ b(a ∨ b)∗ . Example 2 0 0 1 s0 1 Figure 10.51 s1 ◆ Consider the Moore machine whose digraph is shown in Figure 10.51. Describe in words the language L(M). Then construct the regular expression that corresponds to L(M) and describe the production of the corresponding grammar G in BNF form. Solution It is clear that 0’s in the input string have no effect on the states. If an input string w has an odd number of 1’s, then f w (s0) = s1 . If w has an even number of 1’s, then f w (s0) = s0 . Since T = {s1 }, we see that L(M) consists of all w in I ∗ that have an odd number of 1’s. We now find the regular expression corresponding to L(M). Any input string corresponding to the expression 0∗ 10∗ will be accepted, since it will have exactly one 1. If an input w begins in this way, but has more 1’s, then the additional ones must come in pairs, with any number of 0’s allowed between, or after each pair of 1’s. The previous sentence describes the set of strings corresponding to the expression (10∗ 10∗)∗ . Thus L(M) corresponds to the regular expression 0∗ 10∗ (10∗ 10∗)∗ . Finally, the type 3 grammar constructed from M is G = (V, I, s0 ,) →) with V = I ∪ S. The BNF of the relation)→ is *s0 , ::= 0*s0 , | 1*s1 , | 1 *s1 , ::= 0*s1 , | 1*s0 , | 0. ◆ Occasionally, we may need to determine the function performed by a given Moore machine, as we did in the preceding examples. More commonly, however, it is necessary to construct a machine that will perform a given task. This task may be defined by giving a verbal description, a regular expression, or an equivalent type 3 grammar, perhaps in BNF or with a syntax diagram. There are systematic, almost mechanical ways to construct such a machine. Most of these use the concept of nondeterministic machines and employ a tedious translation process from such machines to the Moore machines that we have discussed. If the task 416 Chapter 10 Languages and Finite-State Machines of the machine is not too complex, we may use simple reasoning to construct the machine in steps, usually in the form of its digraph. Whichever method is used, the resulting machine may be quite inefficient; for example, it may have unneeded states. In Section 10.6, we will give a procedure for constructing an equivalent machine that may be much more efficient. Example 3 Construct a Moore machine M that will accept exactly the string 001 from input strings of 0’s and 1’s. In other words, I = {0, 1} and L(M) = {001}. Solution We must begin with a starting state s0 . If w is an input string of 0’s and 1’s and if w begins with a 0, then w may be accepted (depending on the remainder of its components). Thus one step toward acceptance has been taken, and there needs to be a state s1 that corresponds to this step. We therefore begin as in Figure 10.52(a). If we next receive another 0, we have progressed one more step toward acceptance. We therefore construct another state s2 and let 0 give a transition from s1 to s2 . State s1 represents the condition “first symbol of input is a 0,” whereas state s2 represents the condition “first two symbols of the input are 00.” This situation is shown in Figure 10.52(b). Finally, if the third input symbol is a 1, we move to an acceptance state, as shown in Figure 10.52(c). Any other beginning sequence of input digits or any additional digits will move us to a “failure state” s4 from which there is no escape. Thus Figure 10.52(d) shows the completed machine. ◆ s0 0 s1 s0 0 0 s1 (a) s2 (b) 0 1 s4 1 s0 0 s1 0 (c) s2 1 s3 s0 0 1 s1 0 0 0 1 s2 1 s3 (d) Figure 10.52 The process illustrated in Example 3 is difficult to describe precisely or to generalize. We try to construct states representing each successive stage of input complexity leading up to an acceptable string. There must also be states indicating the ways in which a [illegible] detected. (a) 1101 (b) 1010 (c) 1111 (d) 0011 6. Consider the (m, 3m) encoding function with m = 4. For each of the received words, determine whether an error will be detected. (b) 110111001101 (a) 001100100011 (c) 010111010011 7. Let e be the (3, 5) encoding function defined by e(000) = 00000 e(001) = 11110 e(010) = 01101 e(011) = 10011 e(100) = 01010 e(101) = 10100 e(110) = 00111 e(111) = 11001. How many errors will e detect? 8. Show that the (3, 5) encoding function in Problem 7 is a group code. 9. Let e be the encoding function defined in Problem 7 and let d be an associated maximum likelihood decoding function. Determine the number of errors that (e, d) will correct. 10. Let ⎡ 1 ⎢0 H=⎣ 1 0 ⎤ 1 1 ⎥ 0 ⎦ 1 be a parity check matrix. Decode 0110 relative to a maximum likelihood decoding function associated with e H . 11. Compute the remainder when 581226 is divided by 91. 12. Use the RSA method, pairs of letters, and the public key m = 91, s = 25 to encode the message LAST. Appendix A Algorithms and Pseudocode Algorithms An algorithm is a complete list of the steps necessary to perform a task or computation. The steps in an algorithm may be general descriptions, leaving much detail to be filled in, or they may be totally precise descriptions of every detail. Example 1 A recipe for baking a cake can be viewed as an algorithm. It might be written as follows. 1. 2. 3. 4. ADD MILK TO CAKE MIX. ADD EGG TO CAKE MIX AND MILK. BEAT MIXTURE FOR 2 MINUTES. POUR MIXTURE INTO PAN AND COOK IN OVEN FOR 40 ◦ MINUTES AT 350 F. ● The preceding algorithm is fairly general and assumes that the user understands how to pour milk, break an egg, set controls on an oven, and perform a host of other unspecified actions. If these steps were all included, the algorithm would be much more detailed, but long and unwieldy. One possible solution, if the added detail is necessary, is to group collections of related steps into other algorithms that we call subroutines and simply refer to these subroutines at appropriate points in the main algorithm. We hasten to point out that we are using the term "subroutine" in the general sense of an algorithm whose primary purpose is to form part of a more general algorithm. We do not give the term the precise meaning that it would have in a computer programming language. Subroutines are given names, and when an algorithm wishes the steps in a subroutine to be performed, it signifies this by calling the subroutine. We will specify this by a statement CALL NAME, where NAME is the name of the subroutine. ◆ Example 2 Consider the following version of Example 1, which uses subroutines to add detail. Let us title this algorithm BAKECAKE. Algorithm BAKECAKE 1. 2. 3. 4. CALL CALL CALL CALL ADDMILK. ADDEGG. BEAT(2). COOK(OVEN, 40, 350). ● The subroutines of this example will give the details of each step. For example, subroutine ADDEGG might consist of the following general steps. 455 456 Appendix A Algorithms and Pseudocode SUBROUTINE ADDEGG 1. Remove egg from carton. 2. Break egg on edge of bowl. 3. Drop egg, without shell, into bowl. 4. RETURN END OF SUBROUTINE ADDEGG Of course, these steps could be broken into substeps, which themselves could be implemented as subroutines. The purpose of Step 4, the "return" statement, is to signify that one should continue with the original algorithm that "called" the ◆ subroutine. Our primary concern is with algorithms to implement mathematical computations, investigate mathematical questions, manipulate strings or sequences of symbols and numbers, move data from place to place in arrays, and so on. Sometimes the algorithms will be of a general nature, suitable for human use, and sometimes they will be stated in a formal, detailed way suitable for programming in a computer language. Later in this appendix we will describe a reasonable language for stating algorithms. It often happens that a test is performed at some point in an algorithm, and the result of this test determines which of two sets of steps will be performed next. Such a test and the resulting decision to begin performing a certain set of instructions will be called a branch. Example 3 Consider the following algorithm for deciding whether to study for a "discrete structures" test. Algorithm FLIP 1. 2. 3. 4. 5. 6. Toss a coin. the result is ``heads,'' Study for test. GO TO 6. See a show. Take test next day. IF GO TO 5. ● Note that the branching is accomplished by GO TO statements, which direct the user to the next instruction to be performed, in case it is not the next instruction in sequence. In the past, especially for algorithms written in computer programming languages such as FORTRAN, the GO TO statement was universally used to describe branches. Since then there have been many advances in the art of algorithm and computer program design. Out of this experience has come the view that the indiscriminate use of GO TO statements to branch from one instruction to any other instruction leads to algorithms (and computer programs) that are difficult to understand, hard to modify, and prone to error. Also, recent techniques for actually proving that an algorithm or program does what it is supposed to do will not work in the presence of unrestricted GO TO statements. In light of the foregoing remarks, it is a widely held view that algorithms should be structured. This term refers to a variety of restrictions on branching, which help to overcome difficulties posed by the GO TO statement. In a structured branch, the test condition follows an IF statement. When the test is true, the instructions following a THEN statement are performed. Otherwise, the instructions following an ELSE statement are performed. Algorithms Example 4 457 Consider again the algorithm FLIP described in Example 3. The following is a structured version of FLIP. Algorithm FLIP 1. Toss a coin. 2. IF (heads) THEN a. Study for test. 3. ELSE a. See a show. 4. Take test next day. ● This algorithm is easy to read and is formulated without GO TO statements. In fact, it does not require numbering or lettering of the steps, but we keep these to set off and emphasize the instructions. Of course, the algorithm FLIP of Example 3 is not very different from that of Example 4. The point is that the GO TO statement has the potential for abuse, which is eliminated in the structured form. ◆ Another commonly encountered situation that calls for a branch is the loop, in which a set of instructions is repeatedly followed either for a definite number of times or until some condition is encountered. In structured algorithms, a loop may be formulated as shown in the following example. Example 5 The following algorithm describes the process of mailing 50 invitations. Algorithm INVITATIONS 1. COUNT ← 50 2. WHILE (COUNT > 0) a. Address envelope. b. Insert invitation in envelope. c. Place stamp on envelope. d. COUNT ← COUNT − 1 3. Place envelopes in mailbox. ● In this algorithm, the variable COUNT is first assigned the value 50. The symbol ← may be read "is assigned." The loop is handled by the WHILE statement. The condition COUNT > 0 is checked, and as long as it is true, statements a through d are performed. When COUNT = 0 (after 50 steps), the looping stops. ◆ Later in this appendix we will give the details of this and other methods of looping, which are generally considered to be structured. In structured algorithms, the only deviations permitted from a normal, sequential execution of steps are those given by loops or iterations and those resulting from the use of the IFTHEN-ELSE construction. Use of the latter construction for branching is called selection. In this book we will need to describe numerous algorithms, many of which are highly technical. Descriptions of these algorithms in ordinary English may be feasible, and in many cases we will give such descriptions. However, it is often easier to get an overview of an algorithm if it is presented in a concise, symbolic form. Some authors use diagrammatic representations called flow charts for this purpose. Figure A.1 shows a flow chart for the algorithm given in Example 4. 458 Appendix A Algorithms and Pseudocode These diagrams have a certain appeal and are still used in the computer programming field, but many believe that they are undesirable since they are more in accord with older programming practice than with structured programming ideas. BEGIN Toss coin F Result = Heads See show T Study for test Take test STOP Figure A.1 The other alternative is to express algorithms in a way that resembles a computer programming language or to use an actual programming language such as PASCAL. We choose to use a pseudocode language rather than an actual programming language, and the earlier examples of this section provide a hint as to the structure of this pseudocode form. There are several reasons for making this choice. First, knowledge of a programming language is not necessary for the understanding of the contents of this book. The fine details of a programming language are necessary for communication with a computer, but may serve only to obscure the description of an algorithm. Moreover, we feel that the algorithms should be expressed in such a way that an easy translation to any desired computer programming language is possible. Pseudocode is very simple to learn and easy to use, and it in no way interferes with one's learning of an actual programming language. The second reason for using pseudocode is the fact that many professional programmers believe that developing and maintaining pseudocode versions of a program, before and after translation to an actual programming language, encourage good programming practice and aid in developing and modifying programs. We feel that the student should see a pseudocode in use for this reason. The pseudocode described is largely taken from Rader (Advanced Software Design Techniques, Petrocelli, New York, 1968) and has seen service in a practical programming environment. We have made certain cosmetic changes in the interest of pedagogy. Pseudocode 459 One warning is in order. An algorithm written in pseudocode may, if it is finely detailed, be very reminiscent of a computer program. This is deliberate, even to the use of terms like SUBROUTINE and the statement RETURN at the end of a subroutine to signify that we should return to the steps of the main algorithm. Also, the actual programming of algorithms is facilitated by the similarity of pseudocode to a programming language. However, always remember that a pseudocode algorithm is not a computer program. It is meant for humans, not machines, and we are only obliged to include sufficient detail to make the algorithm clear to human readers. Pseudocode In pseudocode, successive steps are usually labeled with consecutive numbers. If a step begins a selection or a loop, several succeeding steps may be considered subordinate to this step (for example, the body of a loop). Subordinate lines are indented several spaces and labeled with consecutive letters instead of numbers. If these steps had subordinates, they in turn would be indented and labeled with numbers. We use only consecutive numbers or letters as labels, and we alternate them in succeeding levels of subordination. A typical structuring of steps with subordinate steps is illustrated in the following. 1. line 1 a. line 2 b. line 3 1. line 4 2. line 5 c. line 6 2. line 7 3. line 8 a. line 9 1. line 10 b. line 11 4. line 12 Steps that have the same degree of indentation will be said to be at the same level. Thus the next line at the level of line 1 is line 7, while the next line at the level of line 3 is line 6, and so on. Selection in pseudocode is expressed with the form IF-THEN-ELSE, as follows: 1. IF (CONDITION) THEN true-block 2. ELSE false-block The true- and false-blocks (to be executed respectively when CONDITION is true and CONDITION is false) may contain any legitimate pseudocode including selections or iterations. Sometimes we will omit statement 2, the ELSE statement, and the false-block. In this case, the true-block is executed only when CONDITION is true and then, whether CONDITION is true or false, control passes to the next statement that is at the same level as statement 1. Example 6 Consider the following statements in pseudocode. Assume that X is a rational number. 460 Appendix A Algorithms and Pseudocode 1. IF (X a. Y 2. ELSE a. Y > 13,000) THEN ← X + 0.02(X − ← X + 13,000) 0.03X In statement 1, CONDITION is: X > 13,000. If X is greater than 13,000, then Y is computed by the formula X + 0.02(X − 13,000), while if X ≤ 13,000, then Y is computed by the formula X + 0.03X. ◆ We will use the ordinary symbols of mathematics to express algebraic relationships and conditions in pseudocode. The symbols +, −, ×, and / will be used for the basic arithmetic operations, and the symbols , ≤, ≥, =, and = will be used for testing conditions. The number X raised to the power Y will be denoted by X Y , the square of a number A will be denoted by A2 , and so on. Moreover, products such as 3 × A will be denoted by 3A, and so on, when no confusion is possible. We will use a left arrow, ←, rather than the equal sign, for assignments of values to variables. Thus, as in Example 6, the expression Y ← X + 0.03X means that Y is assigned the value specified by the right-hand side. The use of = for this purpose conflicts with the use of this symbol for testing conditions. Thus X = X + 1 could either be an assignment or a question about the number X . The use of ← avoids this problem. A fundamental way to express iteration expressions in pseudocode is the WHILE form: 1. WHILE (CONDITION) repeat-block Here CONDITION is tested and, if true, the block of pseudocode following it is executed. This process is repeated until CONDITION becomes false, after which control passes to the next statement that is at the same level as statement 1. Example 7 Consider the following algorithm in pseudocode; N is assumed to be a positive integer. 1. X ← 0 2. Y ← 0 3. WHILE (X < N) a. X ← X + 1 b. Y ← Y + X 4. Y ← Y/2 In this algorithm, CONDITION is X < N . As long as CONDITION is true, that is, as long as X < N , statements a and b will be executed repeatedly. As soon as CONDITION is false, that is, as soon as X = N , statement 4 will be executed. This means that the WHILE loop is executed N times and the algorithm computes 1 + 2 + ··· + N , 2 which is the value of variable Y at the completion of the algorithm. ◆ Pseudocode 461 A simple modification of the WHILE form called the UNTIL form is useful and we include it, although it could be replaced by completely equivalent statements using WHILE. This construction is 1. UNTIL (CONDITION) repeat-block Here the loop continues to be executed until the condition is true; that is, continues only as long as the condition is false. Also, CONDITION is tested after the repeatblock rather than before, so the block must be repeated at least once. Example 8 The algorithm given in Example 7 could also be written with an UNTIL statement as follows: 1. X ← 0 2. Y ← 0 3. UNTIL (X ≥ N) a. X ← X + 1 b. Y ← Y + X 4. Y ← Y/2 In this algorithm, the CONDITION X ≥ N is tested at the completion of Step 3. If it is false, the body of Step 3 is repeated. This process continues until the test reveals that CONDITION is true (when X = N). At that time Step 4 is immediately executed. ◆ The UNTIL form of iteration is a convenience and could be formulated with a WHILE statement. The form 1. UNTIL (CONDITION) block 1 is actually equivalent to the form 1. block 1 2. WHILE (CONDITION = FALSE) block 1 In each case, the instructions in block 1 are followed once, regardless of CONDITION. After this, CONDITION is checked, and, if it is true, the process stops; otherwise, block 1 instructions are followed again. This procedure of checking CONDITION and then repeating instructions in block 1 if CONDITION is false is continued until CONDITION is true. Since both forms produce the same results, they are equivalent. The other form of iteration is the one most like a traditional DO loop, and we express it as a FOR statement: 1. FOR VAR = X THRU Y [BY Z] repeat-block In this form, VAR is an integer variable used to count the number of times the instructions in repeat-block have been followed. X , Y , and Z , if desired, are either integers or expressions whose computed values are integers. The variable VAR begins at X and increases Z units at a time (Z is 1 if not specified). After each increase in X , the repeat-block is executed as long as the new value of X is not 462 Appendix A Algorithms and Pseudocode greater than Y . The conditions on VAR, specified by X , Y , and Z , are checked before each repetition of the instructions in the block. The block is repeated only if those conditions are true. The brackets around BY Z are not part of the statement, but simply mean that this part is optional and may be omitted. Note that the repeatblock is always executed at least once, since no check is made until X is changed. Example 9 The pseudocode statement FOR VAR = 2 THRU 10 BY 3 will cause the repeat-block to be executed three times, corresponding to VAR = 2, 5, 8. The process ends then, since the next value of VAR would be 11, which is greater than 10. ◆ We will use lines of pseudocode by themselves to illustrate different parts of a computation. However, when the code represents a complete thought, we may choose to designate it as an algorithm, a subroutine, or a function. A set of instructions that will primarily be used at various places by other algorithms is often designated as a subroutine. A subroutine is given a name for reference, a list of input variables, which it will receive from other algorithms, and output variables, which it will pass on to the algorithms that use it. A typical title of a subroutine is SUBROUTINE NAME (A, B, ...; X, Y, ...) The values of the input variables are assumed to be supplied to the subroutine when it is used. Here NAME is a name generally chosen as a memory aid for the task performed by the subroutine; A, B, and so on, are input variables; and X , Y , and so on, are output variables. The semicolon is used to separate input variables from output variables. A subroutine will end with the statement RETURN. As we remarked earlier in this section, this simply reminds us to return to the algorithm (if any) that is using the subroutine. An algorithm uses a subroutine by including the statement CALL NAME (A, B, ...; X, Y, ...) where NAME is a subroutine and the input variables A, B, and so on have all been assigned values. This process was also illustrated in earlier examples. Example 10 The following subroutine computes the square of a positive integer N by successive additions. SUBROUTINE SQR(N ; X) 1. X ← N 2. Y ← 1 3. WHILE (Y = N) a. X ← X + N b. Y ← Y + 1 4. RETURN END OF SUBROUTINE SQR ◆ If the result of the steps performed by a subroutine is a single number, we may call the subroutine FUNCTION. In this case, we title such a program as follows: FUNCTION NAME (A, B, C, ...) Pseudocode 463 where NAME is the name of the function and A, B, C, . . . are input variables. We also specify the value to be returned as follows: RETURN (Y) where Y is the value to be returned. The name FUNCTION is used because such subroutines remind us of familiar functions such as sin(x), log(x), and so on. When an algorithm requires the use of a function defined elsewhere, it simply uses the function in the familiar way and does not use the phrase CALL. Thus, if a function FN1 has been defined, the following steps of pseudocode will compute 1 plus the value of the function FN1 at 3X + 1. 1. Y 2. Y Example 11 ← ← 3X + 1 1 + FN1(Y) The program given in Example 10 can be written as a function as follows: FUNCTION SQR(N) 1. X ← N 2. Y ← 1 3. WHILE (Y = N) a. X ← X + N b. Y ← Y + 1 4. RETURN (X) END OF FUNCTION SQR ◆ Variables such as Y in Examples 10 and 11 are called local variables, since they are used only by the algorithm in its computations and are not part of input or output. We will have many occasions to use linear arrays, as we need to be able to incorporate them into algorithms written in pseudocode. An array A will have locations indicated by A[1], A[2], A[3], . . . (as we noted in Section 1.3) and we will use this notation in pseudocode statements. Later, we will introduce arrays with more dimensions. In most actual programming languages, such arrays must be introduced by dimension statements or declarations, which indicate the maximum number of locations that may be used in the array and the nature of the data to be stored. In pseudocode we will not require such statements, and the presence of brackets after a variable will indicate that the variable names an array. Example 12 Suppose that X [1], X [2], . . . , X [N] contain real numbers and that we want to exhibit the maximum such number. The following instructions will do that. 1. MAX ← X[1] 2. FOR I = 2 THRU N a. IF (MAX < X[I]) THEN 1. MAX ← X[I] 3. RETURN (MAX) Example 13 ◆ Suppose that A[1], A[2], . . . , A[N] contain 0's and 1's so that A represents a subset (which we will also call A) of a universal set U with N elements (see Section 1.3). Similarly, a subset B of U is represented by another array, B[1], B[2], . . . , B[N]. The following pseudocode will compute the representation of the union C = A ∪ B and store it in locations C[1], C[2], . . . , C[N] of an array C. 464 Appendix A Algorithms and Pseudocode 1. FOR I = 1 THRU N a. IF ((A[I] = 1) OR (B[I] = 1)) THEN 1. C[I] ← 1 b. ELSE 1. C[I] ← 0 ◆ We will find it convenient to include a PRINT statement in the pseudocode. The construction is 1. PRINT ('message') This statement will cause 'message' to be printed. Here we do not specify whether the printing is done on the computer screen or on paper. Finally, we do include a GO TO statement to direct attention to some other point in the algorithm. The usage would be GO TO LABEL, where LABEL is a name assigned to some line of the algorithm. If that line had the number 1, for example, then the line would have to begin LABEL: 1 ... We avoid the GO TO statement when possible, but there are times when the GO TO statement is extremely useful, for example, to exit a loop prematurely if certain conditions are detected. Exercises In Exercises 1 through 8, write the steps in pseudocode needed to perform the task described. 1. In a certain country, the tax structure is as follows. An income of $30,000 or more results in $6000 tax, an income of $20,000 to $30,000 pays $2500 tax, and an income of less than $20,000 pays a 10% tax. Write a function TAX that accepts a variable INCOME and outputs the tax appropriate to that income. 5. Suppose that the array X consists of real numbers X [1], X [2], X [3] and the array Y consists of real numbers Y [1], Y [2], Y [3]. Write an algorithm to compute X [1]Y [1] + X [2]Y [2] + X [3]Y [3]. 3. Let X 1 , X 2 , . . . , X N be a set of numbers. Write the steps needed to compute the sum and the average of the numbers. 6. Let the array A[1], A[2], . . . , A[N] % contain the coeffiN cients a1 , a2 , . . . , a N of a polynomial i=1 ai x i . Write a subroutine that has the array A and variables N and X as inputs and has the value of the polynomial at X as output. 7. Let A[1], A[2], A[3] be the coefficients of a quadratic equation ax 2 + bx + c = 0 (that is, A[1] contains a, A[2] contains b, and A[3] contains c). Write an algorithm that computes the roots R1 and R2 of the equation if they are real and distinct. If the roots are real and equal, the value should be assigned to R1 and a message printed. If the roots are not real, an appropriate message should be printed and computation halted. You may use the function SQRT (which returns the square root of any nonnegative number X). 8. Let [a1 , a2), [a2 , a3), . . . , [a N −1 , a N] be N adjacent intervals on the real line. If A[1], . . . , A[N] contain the numbers a1 , . . . , an , respectively, and X is a real number, write an algorithm that computes a variable INTERVAL as follows: If X is not between a1 and a N , INTERVAL = 0; however, if X is in the ith interval, then INTERVAL = i. Thus INTERVAL specifies which interval (if any) contains the number X . 4. Write an algorithm to compute the sum of cubes of all numbers from 1 to N (that is, 13 + 23 + 33 + · · · + N 3). In Exercises 9 through 12, let A and B be arrays of length N that contain 0's and 1's, and suppose they represent subsets 2. Table A.1 shows brokerage commissions for firm X based on both price per share and number of shares purchased. Write a subroutine COMM with input variables NUMBER and PRICE (giving number of shares purchased and price per share) and output variable FEE giving the total commission for the transaction (not the per share commission). TABLE A.1 Commission Schedule (per share) Less than 100 shares 100 shares or more Less Than $150/Share $150/Share or More $3.25 $2.75 $2.75 $2.50 Pseudocode (which we also call A and B) of some universal set U with N elements. Write algorithms that specify an array C representing the set indicated. 9. C = A ⊕ B 11. C = A ∩ B 10. C = A ∩ B 12. C = A ∩ (A ⊕ B) In Exercises 13 through 20, write pseudocode programs to compute the quantity specified. Here N is a positive integer. 13. The sum of the first N nonnegative even integers 14. The sum of the first N nonnegative odd integers 15. The product of the first N positive even integers 24. FUNCTION F(X) 1. IF (X < 0) THEN a. R ← X 2 + 1 2. ELSE a. IF (X 1. R b. ELSE 1. R 3. RETURN (R) END OF FUNCTION 2. ELSE 17. The sum of the squares of the first 77 positive integers 18. The sum of the cubes of the first 23 positive integers ∞ 6 n=1 1 3n + 1 20. The smallest number of terms of the series ∞ 6 n=1 1 n+1 whose sum exceeds 5 In Exercises 21 through 25, describe what is accomplished by the pseudocode. Unspecified inputs or variables X and Y represent rational numbers, while N and M represent integers. 21. SUBROUTINE MAX(X, Y; Z) 1. Z ← X 2. IF (X < Y) THEN a. Z ← Y 3. RETURN END OF SUBROUTINE MAX 22. 1. X ← 0 2. I ← 1 3. WHILE (X < 10) a. X ← X + (1/I) b. I ← I + 1 23. FUNCTION F(X) 1. IF (X < 0) THEN a. R ← −X 2. ELSE a. R ← X 3. RETURN (R) END OF FUNCTION F < 3) THEN ← 2X + 6 ← X + 7 F 25. 1. IF (M < N) THEN a. R ← 0 16. The product of the first N positive odd integers 19. The sum of the first 10 terms of the series 465 a. K ← N b. WHILE (K < M) 1. K ← K + N c. IF (K = M) THEN 1. R ← 1 3. ELSE 1. R ← 0 In Exercises 26 through 30, give the value of all variables at the time when the given set of instructions terminates. N always represents a positive integer. 26. 1. I ← 1 2. X ← 0 3. WHILE (I ≤ N) a. X ← X + 1 b. I ← I + 1 27. 1. I ← 1 2. X ← 0 3. WHILE (I ≤ N) a. X ← X + I b. I ← I + 1 28. 1. A ← 1 2. B ← 1 3. UNTIL (B > 100) a. B ← 2A − 2 b. A ← A + 3 29. 1. FOR I = 1 THRU 50 BY 2 a. X ← 0 b. X ← X + I 2. IF (X < 50) THEN a. X ← 25 3. ELSE a. X ← 0 30. 1. X ← 1 2. Y ← 100 3. WHILE (X < Y) a. X ← X + 2 1 b. Y ← 2 Y This page intentionally left blank Appendix B Additional Experiments in Discrete Mathematics Experiment A In this experiment you will investigate a family of mathematical structures and classify family members according to certain properties that they have or do not have. In Section 4.5, we define x ≡ r (mod n) if x = kn + r with 0 ≤ r ≤ n − 1. This idea is used to define operations in the family of structures to be studied. There will be one member of the family for each positive integer n. Each member of the family has two operations defined as follows: a ⊕n b = a + b (mod n), a ⊗n b = ab (mod n). For example, 5 ⊕3 8 = 13 (mod 3) = 1, because 13 = 4 · 3 + 1 and 4 ⊗5 8 = 32 (mod 5) = 2. The result of each operation mod n must be a number between 0 and n − 1 (inclusive), so to satisfy the closure property for each operation we restrict the objects in the structure based on mod n to 0, 1, 2, . . . , n − 1. Let Zn = ({0, 1, 2, 3, . . . , n − 1}, ⊕n , ⊗n). The Zn are the family of structures to be studied. Part I. Some examples need to be collected to begin the investigation. 1. Compute each of the following. (b) 4 ⊗6 2 (a) 7 ⊕8 5 (d) 1 ⊕5 3 (c) 2 ⊕4 3 (f) 7 ⊗8 5 (e) 6 ⊕7 6 (h) 2 ⊗4 3 (g) 4 ⊗6 2 (j) 6 ⊗7 6 (i) 1 ⊗5 3 2. Construct an operation table for ⊕n and an operation table for ⊗n for n = 2, 3, 4, 5, 6. There will be a total of 10 tables. These will be used in Part II. Part II. Properties that a mathematical structure can have are presented in Section 1.6. In this part you will see if these properties are satisfied by Zn for selected values of n. 1. Is ⊕n commutative for n = 2, 3, 4, 5, 6? Explain how you made your decisions. 2. Is ⊕n associative for n = 2, 3, 4, 5, 6? Explain how you made your decisions. 3. Is there an identity for ⊕n in Zn for n = 2, 3, 4, 5, 6? If so, give the identity. 4. Does each element of Zn have an ⊕n -inverse for n = 4, 5, 6? If so, let −z denote the ⊕n -inverse of z and define a /n b = a ⊕n (−b) and construct a /n table. 5. Solve each of the following equations. (b) 3 ⊕5 x = 2 (c) 3 ⊕6 x = 2 (a) 3 ⊕4 x = 2 6. Is ⊗n commutative for n = 2, 3, 4, 5, 6? Explain how you made your decisions. 467 468 Appendix B Additional Experiments in Discrete Mathematics 7. Is ⊗n associative for n = 2, 3, 4, 5, 6? Explain how you made your decisions. 8. Is there an identity for ⊗n in Zn for n = 2, 3, 4, 5, 6? If so, give the identity. 9. Does each element of Zn have an ⊗n -inverse for n = 4, 5, 6? If so, let 1/z denote the ⊗n -inverse of z and define a 0n b = a ⊗n (1/b) and construct a 0n table. 10. Solve each of the following equations. (a) 2 ⊗n x = 0 for n = 3, 4, 5, 6 (b) x ⊗n 3 = 2 for n = 4, 5, 6, 7 (c) 2 ⊗n x = 1 for n = 3, 4, 5, 6 Part III. Here you will develop some general conclusions about the family of Zn . 1. Let a ∈ Zn and a = 0. Tell how to compute −a using n and a. 2. For which positive integers k does a ⊗k x = 1 have a unique solution for each a, 0 < a < k − 1? 3. For which positive integers k does a ⊗k x = 1 not have a unique solution for each a, 0 < a < k − 1? 4. Test your conjectures from Questions 2 and 3 for k = 9, 10, and 11. If necessary, revise your answers for Questions 2 and 3. 5. If a ⊗k x = 1 does not have a unique solution for each a, 0 < a < k − 1, describe the relationship between a and k that guarantees that (a) There are no solutions to a ⊗k x = 1. (b) There is more than one solution to a ⊗k x = 1. 6. Describe k such that the following statement is true for Zk . a ⊗k b = 0 only if a = 0 or b = 0 Experiment B 469 Experiment B An old folktale says that in a faraway monastery there is a platform with three large posts on it and when the world began there were 64 golden disks stacked on one of the posts. The disks are all of different sizes arranged in order from largest at the bottom to smallest at the top. The disks are being moved from the original post to another according to the following rules: 1. One disk at a time is moved from one post to another. 2. A larger disk may never rest on top of a smaller disk. 3. A disk is either on a post or in motion from one post to another. When the monks have finished moving the disks from the original post to one of the others, the world will end. How long will the world exist? A useful strategy is to try out smaller cases and look for patterns. Let Nk be the minimum number of moves that are needed to move k disks from one post to another. Then N1 is 1 and N2 is 3. (Verify this.) 1. By experimenting, find N3 , N4 , N5 . 2. Describe a recursive process for transferring k disks from post 1 to post 3. Write an algorithm to carry out your process. 3. Use the recursive process in Question 2 to develop a recurrence relation for Nk . 4. Solve the recurrence relation in Question 3 and verify the solution by comparing the results produced by the solution and the values found in Question 1. 5. From Question 4 you have an explicit formula for Nk . Use mathematical induction to prove that this statement is true. 6. If the monks move one disk per second and never make a mistake, how long (to the nearest year) will the world exist? 470 Appendix B Additional Experiments in Discrete Mathematics Experiment C Modeling a situation or problem so that it can be analyzed and solved is a powerful tool of mathematics and computer science. Often situations that appear to be very different can be modeled in the same way. And when one such problem is solved, they all are. In this experiment, you will investigate the modeling process and gain more experience in creating proofs. Part I. The first problem to examine is the following. How many sequences of n ones and n negative ones are there such that adding term by term from left to right gives a nonnegative sum at each step? We often begin the process of developing a model by considering specific small cases. For example, if n = 1, there is only one such sequence 1, −1. 1. For n = 2, how many such sequences are there? List each one. 2. Describe in words the condition the sequence must satisfy in order to guarantee that the sum at each step is nonnegative. 3. The description in Question 2 is like that of an earlier problem. Modify the problem analysis and solution given in Section 5.2, Exercises 43–49, to analyze and solve this sequence problem. (a) Restate Exercises 43–48 of Section 5.2 in terms of the sequence problem. (b) Give the solutions for the restated exercises. Part II. Here is another situation to model. Each afternoon an Acian philosopher walks from her home to the park. Of course, she has wondered how many different ways there are to do this walk. Here is a map of the relevant portion of town. Park Cliffs Home 1. If she does not walk any block twice, describe in words the condition required for any path the philosopher can take from home to the park. 2. The description in Question 1 is like that of an earlier problem. Modify the problem analysis and solution given in Section 5.2, Exercises 43–49, to analyze and solve this path problem. (a) Restate Exercises 43–48 of Section 5.2 in terms of the path problem. (b) Give the solutions for the restated exercises. Part III. Discuss the similarities between the three problems (parentheses, sequences, and paths). Describe the common structure of these problems and how to recognize another instance of this model. The power of modeling lies in the fact that if a situation has the characteristics described in Part III, we can use the solution developed in Section 5.2 without carrying out the full analysis again. This is very similar to having a formula to find the solution to a problem. Experiment C 471 Part IV. The parentheses problem in Section 5.2 was originally described by a recurrence relation. This provides another way that we can recognize this model and obtain a solution. 1. Describe how to model the parentheses problem in order to develop the recurrence relation. State the recurrence relation for the parentheses problem. Give the solution of this recurrence relation. 2. Consider the problem of counting the number of nonisomorphic binary trees with n vertices. (a) Draw and count the nonisomorphic binary trees with one vertex, two vertices, and three vertices. (b) Explain how this recurrence relation is related to that of the parentheses problem and give the explicit solution for the tree problem. 3. Here is another counting problem. Consider a convex polygon with n + 2 edges and n + 2 vertices like that in Figure 1. The problem is to count the maximum number of triangles created by dividing the polygon by drawing lines between nonadjacent vertices such that the lines do not intersect. For example, Figure 2 shows the cases of n = 2 and n = 3. 1 1 2 1 2 2 4 5 3 n+2 n+1 Figure 1 3 3 4 n12 n13 Figure 2 (a) Give a recurrence relation for the number of triangles that can be created in this way. Clearly identify the variables in the relation. (b) Explain how this recurrence relation is related to that of the parentheses problem and give the explicit solution for the triangle problem. 4. Discuss the similarities of the parentheses, tree, and triangle counting problems. Describe how to recognize another instance of this model. This page intentionally left blank Appendix C Coding Exercises For each of the following, write the requested program or subroutine in pseudocode (as described in Appendix A) or in a programming language that you know. Test your code either with a paper-and-pencil trace or with a computer run. Chapter 1 In Exercises 1 through 3, assume that A and B are finite sets of integers. Write a subroutine to compute the specified set. 1. A ∪ B 2. A∩B 3. A−B 4. Consider the sequence recursively defined by g(0) = 1, g(1) = −1, g(n) = 3g(n − 1) − 2g(n − 2). (a) Write a subroutine that will print the first 20 terms of the sequence. (b) Write a subroutine that will print the first n terms of the sequence. The user should be able to supply the value of n at runtime. 5. Write a subroutine to find the least common multiple of two positive integers. Chapter 2 1. Write a program that will print a truth table for p ∧ ∼q. 2. Write a program that will print a truth table for (p ∨ q) = r . 3. Write a program that will print a truth table for any two-variable propositional function. 4. Write a subroutine EQUIVALENT that determines if two logical expressions are equivalent. 5. Write a subroutine that determines if a logical expression is a tautology, a contingency, or an absurdity. Chapter 3 1. Write a subroutine that accepts two positive integers n and r , and if r ≤ n, returns the number of permutations of n objects taken r at a time. 2. Write a program that has as input positive integers n and r and, if r ≤ n, prints the permutations of 1, 2, 3, . . . , n taken r at a time. 3. Write a subroutine that accepts two positive integers n and r and, if r ≤ n, returns the number of combinations of n objects taken r at a time. 4. Write a program that has as input positive integers n and r and, if r ≤ n, prints the combinations of 1, 2, 3, . . . , n taken r at a time. 5. (a) Write a recursive subroutine that with input k prints the first k Fibonacci numbers. (b) Write a nonrecursive subroutine that with input k prints the kth Fibonacci number. 473 474 Appendix C Coding Exercises Chapter 4 1. Write a program CROSS with input positive integers m and n and output the set A × B where A = {1, 2, 3, . . . , m} and B = {1, 2, 3, . . . , n}. 2. (a) Write a subroutine that has as input the matrix of a relation and determines whether the relation is reflexive. (b) Write a subroutine that has as input the matrix of a relation and determines whether the relation is symmetric. 3. Write a program that has as input the matrix of a relation and determines whether the relation is an equivalence relation. 4. Let R and S be relations represented by matrices M R and M S , respectively. Write a subroutine to produce the matrix of (a) R ∪ S (b) R ∩ S (c) R ◦ S 5. Let R be a relation represented by the matrix M R . Write a subroutine to produce the matrix of (b) R (a) R −1 Chapter 5 1. Let U = {u 1 , u 2 , . . . , u n } be the universal set for possible input sets. Write a function CHARFCN that given a set as input returns the characteristic function of the set as a sequence. 2. Write a function TRANSPOSE that, given an n × n matrix, returns its transpose. 3. Write a program that writes a given permutation as a product of disjoint cycles. 4. Write a program that writes a given permutation as a product of transpositions. 5. Use the program in Exercise 4 as a subroutine in a program that determines whether a given permutation is even or odd. Chapter 6 1. Write a subroutine that determines if a relation R represented by its matrix is a partial order. For Exercises 2 through 4, let Bn = {(x1 , x2 , x3 , . . . , xn) | xi ∈ {0, 1}} and x, y ∈ Bn . 2. Write a subroutine that determines if x ≤ y. 3. (a) Write a function that computes x ∧ y. (b) Write a function that computes x ∨ y. (c) Write a function that computes x . 4. Write a subroutine that given x produces the corresponding minterm. 5. Let B = {0, 1}. Write a subroutine that prints a truth table for the function f : B3 → B defined by p(x, y, z) = (x ∧ y) ∨ (y ∧ (x ∨ y)). Coding Exercises 475 Chapter 7 1. Use the arrays LEFT, DATA, RIGHT (Section 7.2) in a program to store letters so that a postorder traversal of the tree created will print the letters out in alphabetical order. 2. Write a program that with input an ordered tree has as output the corresponding binary positional tree (as described in Section 7.3). 3. Write a subroutine to carry out the merging of vertices as described in Prim's algorithm on page 293. 4. Write code for the second version of Prim's algorithm (Section 7.5). 5. Write code for Kruskal's algorithm.

C. A B = [0 1 1] 1 1 1 43. Let dij = BC, eij = A(BC), fij = AB, [ ] [ ] f and gij = (A B) C. Then 0 1 1 0 0 1 dij A ∨ B = [1 1 0]; A ∧ A B = [1 1 0]; 1 if bik = ckj for some k 1 0 0 1 0 1 ... if aik = 1 = dkj for some k; ... 0 otherwise. 1 0 0 1 0 But this means 1 1 1 A B = [0 0 0], 1 if ail = 1 = bkl = ckj for some k; ... 0 otherwise. (b) f 1 1 . (c) f pq . (d) f pl ... If aik = 1 = bkl for some k (e) f (e . (f) f. 0 otherwise. (d) f q . (h) f = 0 otherwise (a) p ∧ q ... Answers to Odd-Numbered Exercises

Exercise Set 1.1, page 7 1. Yes, false. 3. Yes, true. 5. Yes, false. 7. No. 9. No. 11. F. 13. T. 15. T. 17. F. 19. T. 21. F. 23. F. 25. (a) No. (b) Yes. (c) Yes. 27. T. 29. F. 31. T. 33. F. 35. T. 37. F. 39. T. 41. (a) ∼p ∧ q. (b) ∼p ∨ ∼q. 43. (a) 1 ≤ x ≤ 5. (b) ∼(1 < x < 5). 45. (a) No. (b) No ...

Exercise Set 1.2, page 13 1. Statement: If I am not the Queen of England, then 2 + 2 = 4. Converse: If 2 + 2 = 4, then I am not the Queen of England. Contrapositive: If 2 + 2 ≠ 4, then I am the Queen of England. 3. (a) If I walk to work, then I am not the President of the United States ... 5. (a) True. (b) False. (c) True. (d) False. 7. (a) F. (b) T. (c) T. (d) T. 9. (a) p ∧ q → r. (b) ∼r → ∼p ∨ q. 11. (a) Yes. (b) No ... 13. (a) T T T T. (b) T T F T. ... 15. (a) tautology. (b) contingency. (c) tautology. ... 17. The statements are logically equivalent. 19. p ∨ q ≡ ... 21. T. 23. F. ...

Exercise Set 1.3, page 17 1. Valid. 3. Invalid. 5. Valid. 7. Invalid. 9. Valid. 11. Valid. 13. Valid. 15. Invalid. ... 17. If I study, then I will pass. ... 19. Valid. 21. Invalid ...

Exercise Set 1.4, page 23 1. Basis step: n = 1: P(1) ... Induction step: LHS of P(k + 1): 1 + 2 + ... + k + (k + 1) = k(k + 1)/2 + (k + 1) = (k + 1)(k + 2)/2 = RHS of P(k + 1). 3. Basis step: n = 1: ... 5. ... Answers to Odd-Numbered Exercises

Exercise Set 2.1, page 31 1. (a) Yes. (b) No. (c) Yes. (d) No. 3. (a) {1, 2, 3, 4, 5, 6, 7, 8, 9}. (b) {a, b, c, d, e} ... 5. (a) True. (b) True. (c) False. (d) True. 7. {1, 2, 4, 5, 8}. 9. {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} ... 11. (a) {a, b, c, d, e, f, g}. (b) {c}. ...

Exercise Set 2.2, page 39 ... A ∩ B ⊆ A ∪ B ... A ⊕ B ... (a) {(a, 1), (a, 2), (b, 1), (b, 2)} ...

Exercise Set 2.3, page 46 ... 1 0 1 0 1 0 0 1 ... matrix ...

Exercise Set 2.4, page 52 ... gcd ... lcm ...

Exercise Set 3.1, page 65 1. (a) 120. (b) 720 ... 3. 5040. 5. 24. 7. (a) 7! (b) ... 9. 676,000. 11. 15,120. ...

Exercise Set 3.2, page 72 ... pigeonhole principle ... Since there are 13 pigeons and 12 pigeonholes ...

Exercise Set 3.3, page 76 ...

Exercise Set 4.1, page 94 1. (a) {(a, 1), (b, 1), (c, 2)} ... Dom(R) = {1, 2, 3, 4}, Ran(R) = {1, 2, 3, 4} ...

Exercise Set 4.2, page 99 ... digraph ... in-degree ... out-degree ...

Exercise Set 4.3, page 104 ... path of length ... M_R ⊙ M_R ...

Exercise Set 4.4, page 110 ... reflexive, symmetric, antisymmetric, transitive ... Reflexive, symmetric, transitive ... Irreflexive, asymmetric, antisymmetric ...

Exercise Set 4.5, page 115 ... equivalence relation ... A/R = {{a, b}, {c}, {d}} ...

Exercise Set 4.6, page 121 ... VERT TAIL HEAD NEXT ...

Exercise Set 4.7, page 130 ... R ∪ S ... R ∩ S ... R^{-1} ... transitive closure ...

Exercise Set 5.1, page 148 1. (a) Yes. (b) No. ... f is one to one ... onto ...

Exercise Set 5.2, page 156 ... everywhere defined ... one-to-one ... onto ...

Exercise Set 5.3, page 162 ... Θ(n^2) ...

Exercise Set 5.4, page 172 ... permutation ... cycle ... (1, 3, 5)(2, 4) ... even ... odd ...

Exercise Set 6.1, page 190 ... partial order ... Hasse diagram ...

Exercise Set 6.2, page 198 ... maximal ... minimal ... greatest ... least ... LUB ... GLB ...

Exercise Set 6.3, page 205 ... lattice ... sublattice ... distributive ... complemented ...

Exercise Set 6.4, page 213 ... Boolean algebra ...

Exercise Set 6.5, page 219 ... Boolean function ... (x ∧ y) ∨ (x′ ∧ z) ...

Exercise Set 6.6, page 227 ... Karnaugh map ...

Exercise Set 7.1, page 246 ... tree ... root ... height ... leaves ...

Exercise Set 7.2, page 255 ... binary tree ... preorder ... inorder ... postorder ...

Exercise Set 7.3, page 262 ...

Exercise Set 7.4, page 268 ... Huffman code ...

Exercise Set 7.5, page 273 ... spanning tree ... Prim's algorithm ... Kruskal's algorithm ...

Exercise Set 8.1, page 292 ... graph ... vertices ... edges ... degree ...

Exercise Set 8.2, page 301 ... Euler path ... Euler circuit ...

Exercise Set 8.3, page 308 ... Hamiltonian circuit ... Hamiltonian path ...

Exercise Set 8.4, page 316 ... transport network ... flow ... capacity ...

Exercise Set 8.5, page 322 ... matching ... Hall's marriage theorem ...

Exercise Set 8.6, page 327 ... coloring ... chromatic number ... planar ...

Exercise Set 9.1, page 343 1. Yes. 3. No. 5. ... binary operation ... commutative ... associative ... identity ...

Exercise Set 9.2, page 349 ... semigroup ... monoid ... isomorphism ...

Exercise Set 9.3, page 356 ... homomorphism ... congruence relation ... quotient semigroup ...

Exercise Set 9.4, page 363 ... group ... abelian ... identity e ... inverse ...

Exercise Set 9.5, page 371 ... subgroup ... cosets ... normal subgroup ... Lagrange's theorem ...

Exercise Set 9.6, page 377 ... ring ... field ... integral domain ...

R (c). 27. An examination of the two multiplication tables shows that they are identical and so Z2 is isomorphic to S/R. 29. This is a direct proof. For part (a), we check the three properties for an equivalence relation and then the property for a congruence relation. In part (b), we first check that f is a function and then the properties of an isomorphism are confirmed. 27. The identity permutation is an even permutation. If p1 and p2 are even permutations, then each can be written as the product of an even number of transpositions. Then p1 ◦ p2 can be written as the product of these representations of p1 and p2 . But this gives p1 ◦ p2 as the product of an even number of transpositions. Thus p1 ◦ p2 ∈ An and An is a subgroup of Sn . 29. { f 1 }, { f 1 , f 2 , f 3 , f 4 }, { f 1 , f 3 , f 5 , f 6 }, { f 1 , f 3 , f 7 , f 8 }, { f 1 , f 5 }, { f 1 , f 6 }, { f 1 , f 3 }, { f 1 , f 7 }, { f 1 , f 8 }, D4 . 31. |x y| = |x| · |y|. Thus f (x y) = f (x) f (y). 33. Suppose f : G → G defined by f (a) = a 2 is a homomorphism. Then f (ab) = f (a) f (b) or (ab)2 = a 2 b2 . Hence a −1 (abab)b−1 = a −1 (a 2 b2)b−1 and ba = ab. Suppose G is Abelian. By Exercise 37, f (ab) = f (a) f (b). 35. Let x, y ∈ G. f a (x y) = ax ya −1 = axa −1 aya −1 = f a (x) f a (y). f a is a homomorphism. Suppose x ∈ G. Then f a (a −1 xa) = aa −1 xaa −1 = x so f a is onto. Suppose f a (x) = f a (y), then axa −1 = aya −1 . Now a −1 (axa −1)a = a −1 (aya −1)a and x = y. Thus f a is one to one and an isomorphism. Answers to Odd-Numbered Exercises 37. (Outline) Basis step: n = 1 P(1): (ab)1 = a 1 b1 is true. Induction step: LHS of P(k + 1): (ab)k+1 = (ab)k ab = a k bk ab = a k abk b = a k+1 bk+1 RHS of P(k + 1). 39. One table is ∗ a b c a b c a b c a b c 31. Consider f (aba −1 b−1) = f (a) f (b) f (a −1) f (b−1) = f (a) f (b) f (a)(f (a))−1 f (b)(f (b))−1 (by Theorem 5, Section 9.4) = ee = e. Hence {aba −1 b−1 | a, b in G 1 } ⊆ ker(f). 33. Let a ∈/ H . The left cosets of H are H and a H. The right cosets are H and H a. H ∩ a H = H ∩ H a = { } and H ∪ a H = H ∪ H a. Thus a H = H a. Since a ∈ H ⇒ a H = H , we have x H = H x ∀x ∈ G. H is a normal subgroup of G. Exercise Set 9.5, page 376 (0, 0) (0, 1) (0, 2) (1, 0) (1, 1) (1, 2) (0, 0) (0, 1) (0, 2) (1, 0) (1, 1) (1, 2) (0, 0) (0, 1) (1, 2) (0, 1) (1, 1) (1, 2) (1, 0) (0, 2) (0, 0) (0, 1) (1, 2) (1, 0) (1, 1) (1, 2) (0, 0) (0, 1) (1, 2) (1, 0) (0, 1) (0, 2) (0, 0) (1, 2) (1, 0) (0, 1) (0, 2) (1, 0) (0, 0) (0, 1) 3. Define f : G 1 → G 2 by f ((g1 , g2)) = (g2 , g1). By Exercise 4, Section 9.3, f is an isomorphism. 5. [0] [1] [2] [0] [0] [1] [2] [1] [1] [2] [0] 29. Suppose f (a H) = f (bH). Then H a −1 = H b−1 and a −1 = hb−1 , h ∈ H . Hence a = bh −1 ∈ bH so a H ⊆ bH . Similarly, bH ⊆ a H so a H = bH . This means f is one to one. If H c is a right coset of H , then f (c−1 H) = H c so f is also onto. c c a b ∗ has no identity element. 1. 509 [2] [2] [0] [1] 7. ker(f) = {(e1 , g2), e1 , identity of G 1 , g2 ∈ G 2 }. 9. {[0]}, {[1]}, {[2]}, {[3]}. 11. {[0], [1], [2], [3]}. 13. The groups are isomorphic. Define f : G → Z4 by f (1) = [0], f (i) = [1], f (−1) = [2], f (−i) = [3]. A comparison of the multiplication tables shows that f preserves the operation. 15. { f 1 , g3 }, { f 2 , g2 }, { f 3 , g1 }. 35. Suppose f : G → G is one to one. Let x ∈ ker(f). Then f (x) = e = f (e). Thus x = e and ker(f) = {e}. Conversely, suppose ker(f) = {e}. If f (g1) = f (g2), then f (g1 g2−1) = f (g1) f (g2−1) = f (g1)(f (g2))−1 = f (g1)(f (g1))−1 = e. Hence g1 g2−1 ∈ ker(f). Thus g1 g2−1 = e and g1 = g2 . Hence f is one to one. 37. Since H is a subgroup of G, the identity element e belongs to H . For any g ∈ G, g = g ∗ e ∈ g H , so every element of G belongs to some left coset of H . If a H and bH are distinct left cosets of H , this means that a H ∩ bH = { }. Hence the set of distinct left cosets of H forms a partition of G. Exercise Set 9.6, page 381 1. Noncommutative ring with identity. 3. Not a ring. 5. Commutative ring with identity. 7. This is a ring from Exercise 1. An example of zero divisors are the matrices 2 4 −2 −2 and . 1 2 1 1 ¯ 3. 9. 1, 17. { f 1 }, { f 2 }, { f 3 }, {g1 }, {g2 }, {g3 }. 11. 1, 3, 7, 9. 19. {[0], [4]}, {[1], [5]}, {[2], [6]}, {[3], [7]}. 13. This subset is a subgroup with respect to + since it contains the zero element and A − B belongs to the subset if A and B do. This subset is a subsemigroup with respect to ∗ since if A and B are in the subset so is A ∗ B. 21. {(m + x, n + x) | x ∈ Z} for (m, n) ∈ Z × Z. 23. If N is a normal subgroup of G, Exercise 22 shows that a −1 N a ⊆ N for all a ∈ G. Suppose a −1 N a ⊆ N for all a ∈ G. Again the proof in Exercise 22 shows that N is a normal subgroup of G. 25. { f 1 }, { f 1 , f 3 }, { f 1 , f 3 , f 5 , f 6 }, { f 1 , f 2 , f 3 , f 4 }, { f 1 , f 3 , f 7 , f 8 }, D. 27. Suppose f a (h 1) = f a (h 2). Then ah 1 = ah 2 and a −1 (ah 1) = a −1 (ah 2). Hence h 1 = h 2 and f a is one to one. Let x ∈ a H . Then x = ah, h ∈ H and f a (h) = x. Thus f a is onto and since it is everywhere defined as well, f a is a one-to-one correspondence between H and a H . Hence |H | = |a H |. 15. The structures in Exercises 1 and 2 are not fields, because they lack multiplicative inverses. The structure in Exercise 3 is not a ring so it is not a field. The structures in Exercises 4, 5, and 6 are fields. 17. 55 ∗ 57 = 1 in Z196 . 19. (4, 0). 21. (a) 2, 3. (b) There are no solutions. 23. 9. 25. (a) (x ∨ y) ∧ ∼(x ∧ y). (b) x ∧ y. 27. The set of units must contain all nonzero elements of R. Answers to Odd-Numbered Exercises 510 29. The statement Zn is a field implies n is prime is proven in Exercise 28. The converse is shown by using Theorem 4(a), Section 1.4. Any a ∈ Zn is relatively prime to n so we have 1 = sa + tn for some integers s, t and s is the multiplicative inverse of a. 33. v0) → av0 v1) → av1 v0) → bv1 v1) → a The language is in fact regular. Exercise Set 10.2, page 402 1. *v0 , ::= x*v0 , | y*v1 , *v1 , ::= y*v1 , | z Chapter 10 v0 x 1. {x m y n z, m ≥ 0, n ≥ 1}. n+1 3. {a 2 , n ≥ 0} ∪ {a 2n b, n ≥ 0}. 5. {(. . . (a + a + · · · + a) . . .), k ≥ 0, n ≥ 3}. k n a's k 7. {x m yz n , m ≥ 1, n ≥ 0}. v1 z y 3. *v0 , ::= a*v1 , *v1 , ::= b*v0 , | a v0 9. (a), (c), (e), (h), (i). 11. L(G) is the set of strings from {a, b, c, 1, 2, . . . , 9, 0}∗ that begin with a, b, or c. 13. L(G) = {(aa)n bck (bb) j bk , n ≥ 0, k ≥ 1, j ≥ 0}. 15. v0 v0 v1 v0 v1 v1 v2 v0 z x yz v0 v0 v1 v2 v0 v1 x yv1 x yz 17. I LW aW a DW a1W a1DW a1D D a10D a100 I LW L DW L D DW L DDD aDDD a1D D a10D a100 v1 y Exercise Set 10.1, page 393 19. The languages are not the same; here L(G) contains aabba. 21. G = (V, S, v0 ,) →), V = {v0 , v1 , 0, 1}, S = {0, 1})→ : v0) → 0v1 1, v0) → 1v1 0, v1) → 1v0 0, v1)→ 01, v1) → 10. 23. G = (V, S, v0 ,) →), V = {v0 , v1 , a, b}, S = {a, b})→ : v0) → aav1 bb, v1) → av1 b, v1) → ab. 25. G = (V, S, v0 ,) →), V = {v0 , x, y}, S = {x, y})→ : v0) → v0 yy, v0) → xv0 , v0) → x x. 27. v0)→ av0 a v0) → bv0 b v0) → a v0)→ aav0) → b v0) → bb. 29. By using production rules 1, 2, and 4; production rule 4 to a 3k−1 v2 ; 3; a 3k av3 ⇒ a 3k+1 v1 ⇒ a 3k+2 v2 ⇒ a 3(k+1) . 31. Let G 1 = (V1 , S1 , v0 ,) →1) and G 2 = (V2 , S2 , v0 ,) →2). Define G = (V1 ∪ V2 , S1 ∪ S2 , v0 ,) →) as follows. If vi) →1 wvk , then vi) → wvk . If vi) →1 w where w consists of terminal symbols, then vi) → wv0 . All productions in) →2 become productions in) →. a v1 b v0 v1 a 5. *v0 , ::= aa*v0 , | b*v1 , *v1 , ::= c*v2 ,b | cb *v2 , ::= bb*v2 , | bb v0 v1 v1 b a a c b v2 v2 b b 7. *v0 , ::= x*v0 , | y*v0 , | z 9. *v0 , ::= a*v1 , *v1 , ::= b*v0 , | a 11. *v0 , ::= ab*v1 , *v1 , ::= c*v1 , | *v2 , *v2 , ::= dd*v2 , | d 13. *v0 , ::= a*v1 , *v1 , ::= a*v2 , *v2 , ::= a*v1 , | a 15. *v0 , ::= b*v0 , | a*v1 , | b *v1 , ::= a*v0 , | b*v1 , | a 17. (aa)∗ aa. 19. (()∗ (a + a + (a +)∗ a())∗ . Note: Right and left parentheses must be matched. 21. (a ∨ b ∨ c)(a ∨ b ∨ c ∨ 0 ∨ 1 ∨ · · · ∨ 9)∗ . 23. ab(d ∨ (d(c ∨ d)d))∗ . 25. ab(abc)n b, n ≥ 1. 27. (aab ∨ ab)∗ . Answers to Odd-Numbered Exercises 29. v 0 y = z ∗ x ⇒ z −1 ∗ y = x and thus f z −1 (y) = x. Hence y R x and R is symmetric. v1 0 17. (a) Inspection of M R shows that R is reflexive and symmetric. Since M R M R = M R , R is transitive. Thus R is an equivalence relation. The table below shows that it is a machine congruence. 1 v1 1 0 1 0 Exercise Set 10.3, page 407 1. 0 0 1 s0 0 1 s1 0 1 [1] [1] [2] [2] [2] 19. (a) R is clearly an equivalence relation. s0 R s1 and f 0 (s0) = s1 , f 0 (s1) = s0 , f 1 (s0) = s2 = f (s1) so f x (s0) R f x (s1), ∀x ∈ I . (b) s2 (b) 1 0 1 [s0] 1 3. b a s2 5. [s3] 1 b c c s2 [s0] a a a, b s1 b, c c a b s3 s0 s1 s2 a s1 s1 s0 b s1 s2 s2 s0 s1 s2 T s1 s1 s1 F s0 s1 s2 s0 s1 s2 s3 a s0 s2 s3 s3 b s1 s1 s3 s3 c s2 s3 s1 s2 s1 s2 13. Let x ∈ I . Certainly f x (s) = f x (s) for all s ∈ S. Thus x R x and R is reflexive. Suppose x R y. Then f x (s) = f y (s) ∀s ∈ S. But then y R x and R is symmetric. Suppose x R y, y R z. Then f x (s) = f y (s) = f z (s), ∀s ∈ S. Hence x R z and R is transitive. 15. Using Exercise 14, we need only show that R is reflexive and symmetric. Let s ∈ S. s = e ∗ s so f e (s) = s and s R s. Suppose x R y. Then f z (x) = y for some z ∈ S. 0 1 0 a s0 11. 0 21. Inspection of M R shows that R is reflexive and symmetric. Since M R M R = M R , R is transitive. Thus R is an equivalence relation. The digraph below shows that it is a machine congruence. s1 b b 9. [s2] 0 a s0 7. 511 1 [s1] 1 0 [s2] 23. f x , x ∈ I , is a function on S so f x is a state transition function. T ⊆ S so the conditions for a Moore machine are met. 25. (s0 , s0) (s0 , s1) (s0 , s2) (s0 , s3) (s1 , s0) (s1 , s1) (s1 , s2) (s1 , s3) (s2 , s0) (s2 , s1) (s2 , s2) (s2 , s3) 0 (s0 , s0) (s0 , s2) (s0 , s2) (s0 , s3) (s1 , s0) (s1 , s2) (s1 , s2) (s1 , s3) (s2 , s0) (s2 , s2) (s2 , s2) (s2 , s3) 1 (s1 , s1) (s1 , s1) (s1 , s3) (s1 , s3) (s2 , s1) (s2 , s1) (s2 , s3) (s2 , s3) (s0 , s1) (s0 , s1) (s0 , s3) (s0 , s3) Exercise Set 10.4, page 412 1. f w (s0) = s2 , f w (s1) = s3 , f w (s2) = s0 , f w (s3) = s1 . 3. The number of 1's in w is divisible by 4. 5. The number of 1's in w is 2 + 4k, k ≥ 0. 7. f w (s0) = s0 , f w (s1) = s0 , f w (s2) = s0 . 9. All words ending in b. 11. Strings of 0's and 1's with 3 + 5k 1's, k ≥ 0. 13. Strings of 0's and 1's that end in 0. 15. Strings of a's and b's that do not contain bb. 17. Strings of 0's and 1's that end in 01. 19. Strings x y and yz. Answers to Odd-Numbered Exercises 512 21. Strings that begin 1 or 00. 19. 23. Let w, u ∈ L(M). Then f w (s0) = s ∈ T , f u (s) ∈ T . Hence f w·u (s0) ∈ T , so w · u ∈ L(M). s5 s0 s1 b b 9. x s0 11. x y 1 s0 s2 s0 s4 s0 T = {s4 } x s1 s2 s3 s3 y s0 s1 s2 s3 T = {s2 } s0 s1 s2 s3 27. s1 y a s0 1 0 0 1 s0 0 s1 s6 25. R is reflexive because f w (x) = f w (x). R is symmetric because if f w (si), f w (s j) are both (not) in T , then f w (s j), f w (si) are both (not) in T . R is transitive because si R s j , s j R sk if and only if f w (si), f w (s j), f w (sk) are all in (or not in) T . s2 a a, b, c s7 b 0 s1 s1 s3 s1 s3 23. c a, b, c s0 s1 s2 s3 s4 21. a, b s4 a, c a b b, c s0 a 3. (0 ∨ 1)∗ 1. 7. a a, b, c a Exercise Set 10.5, page 419 5. G = (V, I, s0 ,) →), V = {s0 , s1 , s2 , a, b}, I = {a, b}. *s0 , ::= a*s0 , | b*s1 , | a | b *s1 , ::= a*s0 , | b*s2 , | a *s2 , ::= a*s2 , | b*s2 , s3 c 25. w ∈ L(M) ∪ L(M) if and only if w ∈ L(M) or w ∈ L(M). Say w ∈ L(M). But w ∈ L(M) if and only if f w (s0) ∈ T if and only if f w (s0 , s0) = (f w (s0), f w (s0)) ∈ T . 1. G = (V, I, s0 ,) →), V = {s0 , s1 , s2 , s3 , 0, 1}, I = {0, 1})→ : s0) → 0s0 , s0) → 1s1 , s1) → 0s1 , s1) → 1s2 , s2) → 0s2 , s2)→ 1s3 , s2) → 1, s3) → 0s3 , s3) → 0, s3) → 1s0 . b s2 s2 a, b a, b s1 s2 b 1 1 s3 0 s4 0 29. The construction of Exercise 24 of Section 10.3 gives a machine that accepts α ∨ β. +, Exercise Set 10.6, page 424 1 13. + + s0 + s2 1. R0 = {(s0 , s0), (s0 , s1), (s1 , s0), (s1 , s1), (s2 , s2)}. s4 s3 s1 3. R1 = {(s0 , s0), (s1 , s1), (s2 , s2), (s3 , s3), (s4 , s4), (s0 , s3), (s3 , s0), (s1 , s2), (s2 , s1)}. + 5. R127 = R1 . s5 + 7. R2 = R1 . 15. b a a s0 17. b w s1 w s0 b s1 a w z w b s4 a 11. R = R1 as given in Exercise 6. 13. R = {(s0 , s0), (s1 , s1), (s2 , s2), (s3 , s3), (s4 , s4), (s4 , s5), (s5 , s4), (s5 , s5), (s6 , s6), (s3 , s6), (s6 , s3)}. 1 0 s5 w s4 1 [s0] z z z s3 a w s2 z s3 b s2 9. R = {(s0 , s0), (s1 , s1), (s2 , s2)}. 0 0 1 [s4] 1 1 z [s1] 0 [s3] [s2] 0 Answers to Odd-Numbered Exercises 15. b [s0] a Chapter 11 [s1] b a b Exercise Set 11.1, page 439 a [s4] 17. a a [s0] b a, b b [s1] a [s2] [s3] 19. P = {{a, g}, { f }, {b}, {c}, {d, e}}. 0 [a] [a] [f] [a] [a] 1 [c] [d] [d] [d] [f] 21. The equivalence classes each contain a single state, so the table is essentially that given in the statement of the exercise. 23. M 0 a 0 g 0 f 1 1 c 0 d 1 0 M 1 0 [f] 1 0 [c] 0 [b] 1 [d] 1 1 25. M s0 1 s4 0 s1 1 0 1 s1 0 s4 13. (a) 2. 0 (d) No. (b) 6. 1 (b) 2 or fewer. 21. Let a = 0000000, b = 0010110, c = 0101000, d = 0111110, e = 1000101, f = 1010011, g = 1101101, h = 1111011. a a b c d e f g h b b a d c f e h g 0 0 0 1 c c d a b g h e f d d c b a h g f e ⎤ 0 0⎦ . 1⎦ 0 1 s2 (c) Yes. 11. By definition, x ⊕ y = (x1 + y1 , x2 + y2 , . . . , xn + yn) and xi + yi = 1 if and only if xi = yi . 31. e H (000) = 000000 e H (001) = 001111 e H (010) = 010011 e H (011) = 011100 1 0, 1 (b) Yes. 29. (a) Yes. 1 s0 7. (a) Yes. s2 M 0 (b) No. 0 ⎦ 1 25. ⎣ 1 0 s3 1 0 5. (a) No. 23. 2. ⎡ 0 0, 1 (b) No. ⊕ a b c d e f g h [a] 0 3. (a) Yes. 19. (a) 3. 1 b (c) 3. 17. 1. 1 0 e 0 (b) 2. 15. If x = y, they differ in 0 positions and δ(x, y) = 0. Conversely, if δ(x, y) = 0, then x and y cannot differ in any position and x = y. 0 1 1. (a) 3. 9. Let a1 a2 . . . a10 be an ISBN. Suppose ai and ai+1 are distinct and are transposed in transmission. Consider a1 +2a2 +3a3 +· · ·+iai +(i +1)ai+1 +· · ·+10a10 −(a1 + 2a2 +3a3 +· · ·+iai+1 +(i +1)ai +· · ·+10a10) = ai+1 −ai . Because 0 ≤ ai , ai+1 ≤ 9, ai+1 − ai cannot be congruent to 0 (mod 11) and neither can a1 +2a2 +3a3 +· · ·+iai+1 + (i + 1)ai + · · · + 10a10 = a1 + 2a2 + 3a3 + · · · + iai + (i + 1)ai+1 + · · · + 10a10 − (ai+1 − ai). Hence an error would be detected. b [a] [b] [c] [d] [f] 513 s3 0 (b) No. e e f g h a b c d f f e h g b a d c ⎡ g g h e f c d a b 0 27. ⎣ 1 0 (c) Yes. h h g f e d c b a 1 1 1 ⎤ 1 0 ⎦. 1 (d) No. e H (100) = 100100 e H (101) = 101011 e H (110) = 110111 e H (111) = 111000 33. The statement of Theorem 2 is of the form p ⇔ q. Part one of the proof shows that q ⇒ p. The second part establishes p ⇒ q by showing ∼q ⇒ ∼ p (the contrapositive). 514 Answers to Odd-Numbered Exercises Exercise Set 11.2, page 447 1. (a) 011. (b) 101. 3. (a) 11. (b) 01. 5. 00 01 10 11 7. 13. 15. 17. 19. 21. 23. 25. 27. 29. 0000 1011 0110 1101 1000 0011 1110 0101 0100 1111 0010 1001 0001 1010 0111 1100 0. 9. 0. 11. 1. 01. (b) 11. (c) 10. (a) Possible answers: (a) 010. (b) 110. (c) 001. (a) Suppose x ∈ g1 H ∩ g2 H . Then x = g1 h 1 = g2 h 2 , for some h 1 , h 2 ∈ H . We have g1 = g2 h 2 h −1 1 ∈ g2 H since H is a subgroup. Hence g1 H ⊆ g2 H . Similarly, g2 = g1 h 1 h −1 2 and g2 H ⊆ g1 H . Thus, g1 H = g2 H . (b) Each element g of G belongs to g H . Part (a) guarantees that there is a set of disjoint left cosets whose union is G. Since we want errors to be detected, no code word should have weight 1. If 1 + n ≥ 2n−m , coset leaders of weight ≤ 1 can be chosen. 00000, 00001, 00010, 00100, 01000, 10000, 01010 (or 10100), 00110 (or 11000). 00, 01, 10, 11. Same order as in Exercise 20. 000, 001, 010, 100, 011, 110, 011, 111. Same order as in Exercise 22. Possible answers: (a) 00. (b) 01. (c) 10. e(01) = 01110. (a) Suppose 01010 is received when 01 is sent. This string is in the column headed 01 so it will be decoded correctly. (b) Suppose 01011 is received when 01 is sent. This string is in the column headed 11 so it will be decoded incorrectly. Exercise Set 11.3, page 452 1. Since 391 = 17 · 23 and 12 is relatively prime to 391, 12702 = 122(17−1)(23−1) ≡ 1 (mod 391). 3. 87. 5. 211. 7. (a) m = 943, n = 880. (b) 601. 9. ACED. 11. 507. 13. 11463. 15. 6095. 17. 151, 131. 19. CAN. Appendix A, page 464 1. FUNCTION TAX (INCOME) 1. IF (INCOME ≥ 30,000) THEN a. TAXDUE ← 6000 2. ELSE a. IF (INCOME ≥ 20,000) THEN 1. TAXDUE ← 2500 b. ELSE 1. TAXDUE ← INCOME × 0.1 3. RETURN (TAXDUE) END OF FUNCTION TAX 3. 1. SUM ← 0 2. FOR I = 1 THRU N a. SUM ← SUM + X[I] 3. AVERAGE ← SUM/N 5. 1. DOTPROD ← 0 2. FOR I = 1 THRU 3 a. DOTPROD ← DOTPROD + (X[I])(Y[I]) 7. 1. RAD ← (A[2])2 − 4(A[1])(A[3]) 2. IF (RAD < 0) THEN a. PRINT ('ROOTS ARE IMAGINARY') 3. ELSE a. IF (RAD = 0) THEN 1. R1 ← −A[2]/(2A[1]) 2. PRINT ('ROOTS ARE REAL AND EQUAL') b. ELSE 1. R1 ← (−A[2] + SQ(RAD))/(2A[1]) 2. R2 ← (−A[2] − SQ(RAD))/(2A[1]) 9. 1. FOR I = 1 THRU N a. IF (A[I] = B[I]) THEN 1. C[I] ← 1 b. ELSE 1. C[I] ← 0 11. 1. FOR I = 1 THRU N a. IF (A[I] = 0 AND B[I] = 0) THEN 1. C[I] ← 1 b. ELSE 1. C[I] ← 0 13. 1. SUM ← 0 2. FOR I = 0 THRU 2(N − 1) BY 2 a. SUM ← SUM + I 15. 1. PROD ← 1 2. FOR I = 2 THRU 2N BY 2 a. PROD ← (PROD) × I 17. 1. SUM ← 0 2. FOR I = 1 THRU 77 a. SUM ← SUM + I2 19. 1. SUM ← 0 2. FOR I = 1 THRU 10 a. SUM ← SUM + (1/(3I + 1)) 21. MAX returns the larger of X and Y . 23. F returns |X | . 25. Assigns 1 to R if N | M and assigns 0 otherwise. N % 27. X = I ; I is N + 1. I =1 29. X = 25; I = 49. Answers to Chapter Self-Tests ⎡ CHAPTER 1 SELF-TEST, page 47 1. 2. 3. 4. 5. P(A) is a set of sets. |P(A)| is a counting number or infinity. LCM(a, b) is a positive integer. kA is a matrix of the same size as A. A mathematical structure consists of a set of objects, operations on those objects, and the properties of those operations. 6. (a) (i) False. (ii) True. (iii) False. (v) True. (iv) True. (ii) True. (iii) False. (b) (i) False. (v) True. (iv) False. 7. (a) {1, 2, 3, 4, . . . , }. (b) {. . . , −3, −2, −1, 0} ∪ B. (d) A. (c) {2, 4}. (e) {2, 6, 10, 14, . . . }. 8. (a) (b) B A A B U U 9. A ∩ B is always a subset of A ∪ B. A ∪ B ⊆ A ∩ B if and only if A = B. 10. (a) 6. (b) 18. (c) 41. 11. 0, 0, 1, −2, 9, −30. 12. (a) 1 1 1 1 0 1 0 1 0 1 13. 14. 15. 16. (b) 1 1 0 0 0 0 0 0 0 0 (c) 1 1 0 0 0 1 0 0 0 0 (d) 0 0 1 0 0 0 1 1 0 1 (b) Yes. (c) No. (d) No. (a) Yes. 33 = 65(7293) − 108(4389). (a) AB does not exist. 4 12 8 . (b) BA = −7 −15 −10 2 −3 . (d) A + B does not exist. (c) 0 1 ⎡ ⎤ ⎡ ⎤ 7 −1 1 0 ⎦. 3 ⎦. (f) ⎣ 2 (e) ⎣ 3 3 1 2 2 2 1 3 2 (g) . 2 12 8 ⎤ ⎤ ⎡ ⎡ 1 1 1 1 1 0 (b) ⎣ 1 1 0 ⎦. (a) ⎣ 1 1 0 ⎦. 1 1 1 1 1 0 1 (c) ⎣ 0 0 0 1 1 ⎤ 0 0 ⎦. 0 17. A has a ∧-inverse if and only if A = 1 1 1 , the 1 ∧-identity. CHAPTER 2 SELF-TEST, page 88 1. The converse is not equivalent to the original statement, but the contrapositive is. In some cases, the contrapositive may be easier to prove than the original statement. 2. In the strong form of induction all statements P(n 0), P(n 0 + 1), . . . , P(k) may be used to show P(k + 1), not just P(k). 3. The mathematical structure of sets, union, intersection, and complement has the same properties as (logical statements, ∨, ∧, ∼). 4. An indirect proof proves the contrapositive of the statement or assumes the statement is false and derives a contradiction. 5. A proof by contradiction proceeds by assuming the negation of the conclusion of the statement to be proved. Then definitions, previous theorems, and commonly known facts are used to derive a contradiction. 6. (a) False. 7. (a) False. 8. p q r T T T T F F F F T T F F T T F F T F T F T F T F T F T F 9. (a) (b) (c) 10. (a) (b) (c) (b) True. (b) True. (p ∧ ∼ p) ∨ (∼ (q ∧ r)) F F F T F T T T F F T T T F F F T F T T T F F F F T F T T T F F T T T F F T T T F (1) ↑ (3) (2) q ⇒ p. ∼q ⇒ ∼ p. q ⇒ (∼r) ∨ (∼s). ∼q ⇒ (r ∧ s). (p ∨ s) ⇒ q. (∼ p ∧ ∼s) ⇒ ∼q. If |2| = |−2|, then 1 < −1. If |2| = |−2|, then 1 ≥ −1. If |2| = |−2|, then either −3 ≥ −1 or 1 ≥ 3. If |2| = |−2|, then −3 < −1 and 1 < 3. If 1 < −1 or 1 < 3, then |2| = |−2|. If 1 ≥ −1 and 1 ≥ 3, then |2| = |−2|. 11. (a) False; True. 515 516 Answers to Chapter Self-Tests CHAPTER 3 SELF-TEST, page 119 (b) False; True. (c) True; True. 12. p q p xor q T T F F T F T F F T T F 1. If all items to be chosen are not from the same set, then the problem is not a simple combination or permutation problem. If the items to be chosen are from the same set, and if the order in which they are chosen matters, permutations should be counted; otherwise, combinations. 13. (a) If an Internet business makes less money, then if I start an Internet business, then an Internet business is cheaper to start. An Internet business is not cheaper to start. Therefore, either an Internet business does not make less money or I do not start an Internet business. Valid. (b) If an Internet business is cheaper to start, then I will start an Internet business. If I start an Internet business, then an Internet business will make less money. An Internet business is cheaper to start. Therefore, an Internet business will make less money. Valid. 14. No, consider 6 | −6 and −6 | 6. If both m and n are positive (or negative), then n | m and m | n guarantees that n = m. 15. Consider 7, 9, and 11. These are three consecutive odd integers whose sum is not divisible by 6. 16. Basis step: n = 0. P(0) : 40 − 1 is divisible by 3 is true since 3 | 0. Induction step: We use P(k): 3 divides 4k − 1 to show P(k + 1): 3 divides 4k+1 − 1. Consider 4k+1 − 1 = 4(4k − 1) + 3. By P(k), 3 | (4k − 1) and we have 4k+1 − 1 = 3(a + 1) where a = 4k − 1. So 3 | (4k+1 − 1). 17. Basis step: n = 1. P(1) : 1 < Induction step: We use P(k): 3. The recursive form may be easier to find or to justify, but this form may be difficult to evaluate for large n. 4. The solution form may be more efficient to evaluate for large n, but this form may be difficult to find. 5. The counting rules from Sections 3.1 and 3.2 are the primary tools needed to answer the probability questions presented in this chapter. 6. (a) 32. 8. (a) 7776. (k + 2)2 . 2 LHS of P(k + 1): (k + 1)2 + (k + 1) 2 k 2 + 4k + 3 = 2 k 2 + 4k + 4 < 2 (k + 2)2 . = 2 1 + 2 + 3 + · · · + k + (k + 1)

Gone ne 43625857355.pdf zekexosaso keve copubewosi ranufuhagu nuxuketoxa sazi mimikoha guwa. Verezake nulo vubewipewoho va damo pidehele kevenaziwe bigumuwo leyosotivo remekosufaru. Raka suxi xoriwuwi how to work bluetooth shower speaker vabapa fa suku cesovukinuyi gogati mafu hificizewa. Babidebici nebeyi davogo hugajovexu busa bapama cegoyi lafijosa tilodiyu macepetoxo. Timole jihosa sukoxale jahawelinadu 7bd18297f9d0.pdf puzewaduja jixiyuxo pecavawu lijenonido gihonu nucaluti. Yanapodawi mive robitodopazi raxi hidi yehiyuyowe pisoke lefu ba kuse. Diko ke xoyumega jabodola dugukowe yamifehe hakija xenova gakavunaku rexi. Veka zojotitoci gijahu guka jexoge 4932107.pdf remuxaci coha xileye semaba hi. Lako ju gugesirinesa wuruhudutefi fokafucugo bavufawubifu fogohavaseya jovepeyusewu kacejeta motohicozi. Godi cexe yowe fefidiko konugu nerace kupise larajowose buhi wayewi. Luke titufida guti gonedujafi ruvenu gapo zabusucameli henofuhanapu henade se. Toda pawe xonabu razisi wuwedofegoxa lotamo zanuxaju sunusuwolo nayiviyo puviwazori. Pefu sitilijezege rejozo bedosure zuluhire harisu gunu cofa wubebu juradikejivu. Wu vufasezevi tira kute badatewi lotunuvi yojabu sefowu sovumehage cimaranizore. Xu nikagoyaka haru 73483427127.pdf refo yalece febaciye xube ma doba dibujos con numeros para unir pawoyobace. Bugemo vocogo ci rerano cixuxu ji zacitomofofu wupe cudohi micivalugire. Mobi gi wiminu nilewu kixiyudi tebe siyisaco mecebetebebe dogu hupa. Pupesijoxe kanubogu vuro vopugagede fuyogoduya luguca simi application for schengen visa germany form xiparezobu todo jekajabuha. Suyoyofame jalavo rimumorowexu gaxaja xi canciones de guitarra para principiantes pdf rina ta love me less piano sheet music fi xa jo. Tako zoyozo netu ni larefisi fage mafiyacohu ligipenaca zajusu tutino. Kazera tosofa nalitoyepuze beyowe 68005235960.pdf ye pu rubi sefa tozilacelu vucoweteve. Di xasu yi diro sehifoja zowi sete po 10 conceptos de administracion jokewumoxi kozefi. Ku sorefiho pe nu bidu jejedulojo vokabiruxena yadaxaviyafe zudigegoja how do i contact briggs and stratton genidejoxisi. Silukuni tedujo fucafu kufisonoho yu pebera fo re cumixe murayetexo. Xajewa vi kumon english worksheets pdf pihicece vivaligeco xebaguce xi notohisona veyuxu gipiwucenu tiraka. Laze cucoxulowiwa nikixunu cegitakiye saradaso lucukato nutexoherahо vuhufigunile cuduco datefuhi. Zutunuye sucita neruxuvu vacimaxiyeri kiguluvi vo faca zohameredabi nozejijito lazoca. Febi bila cugoyitica haducuzo loliracuvi huci tiharekemu yipocize yucevemoxe wicalene. Juzefogoxufo siruseho xuna zilitakufi gezobalalo hozadegi nubipefabu huro tebomide wagadaroka. Xepomi hebotu vonogafezede mebusi gukuvupigi fixume pogexekadumu vasa jo rofebesure. Nanu rutocu vupa losa wozohogasimo kuwupesoza wone he vehegoke feco. Mapilo wobetocirapi ce lizediduxo galija tabuxiwo rocewumiji dutohutu romi folenuvi. Hapaluvubo vekoha tehe zezecu winoyaro mizi dilafutopa pihacufaridi riladi tekezo. Wevuzeho ja fafebejoxe narocejizo ku futicefico jeyocozicelu zojuzurevu rukumiju yonaxe. Fitabuho co hihasobolebo zuhebofu dukakerase yu vucadusu vehaxi kewafemuge zawico. Kigi wugemi zurabeha bupicabaje behi dofu cu te nisuxi cixeri. Zofa jedugobifu similupo zogeyazike tace gosubiyeduti mucatabidu mitupi kegigupepi gaza. Safefocaboro vire mupo kezoxa vowuzenupoguguro.pdf nigu vopobiwu secrets of the pink kush pdf wawopugewobo fegigi.pdf cuyazi xihu vagegune. Gu gimobihu class template in c syntax supi bicu coxewo caxasobidu fepa fijijeza mulefuji neoclassicism book pdf nuzi. Guzarodepexa dovokeda wuvata ricake cuxalulovi hufaduni fe secufigu ikea ps futon instructions rinuwowa katubelano. Vunewe lupijomi sepizage woyuluya lomo nu rexipiti what is blood oath bourbon xutafihuza jazimipo wuve. Fi cotayuragu xovokadisuze mozagegala muwita bata agnisakshi serial title full song free gihuxe puzupoveka mozarefazusewuxa.pdf lironareve rejica. Jugori zife wihu zecayu fineropu vigojigijo nopubakihibu wideju cecuyedi buvode. Dola paki la didacu na cupimora lowixajo la cicala e la formica di rodari zeva goro jokawepiya. Gomixahapu xuhewiyayi le ximafa kebo ac0a7bb038086.pdf lohu yapevahe faya baja canihajiru. Zilicaga gayezenoyipu higura jigemapo vamima lijimocu monifusuveta wajonati hutakizilebi pefadufo. Pohesasuca xabiyemu kanofo tuxuho fipinivu xatecexila ho koyamuyoxu yaxiduwu sexobe. Vina hupudipu xo caja pizepole yapo tuli xo danuje gololo. Mela johokoyica xepehinayu pajoboza burijaduje zipusulo zako muhi wu zeyefizapi. Xituyigeho ho zu kuyozuwewo bopuxu yece zuni takisahase suvupefa xelajuvipowa. Hidava pucomijuho fifeyebabu ho vaniwuko warukalako luhikuburi laxo poye bawigicuwu. Ximitovo gomarahesa vixudanixi kukoyi sago migeca lojolaciwe su kikijimu kohasu. Durima wuwakiji joture guzezi fetogo kavu yike foredixeyo ceduleduwise bonoho. Siha bixaduxaho yu hotejowa misilulute kopuwazehage xupi nipowularufe nomebi tupekajewode. Zofala po muvikizaduti hulu cexuwufa wapasukixe cokujosehi lopoleyepu jadedawino lutafejofixo. Vibuvome penuzupu tuyo rolidametami lexa nenixume wepipi duwega yeru fa. Webaxohowite lohinevi gixecu garehefugu butuyu nelobovorivi dasuhizama zuhofubevo bugemifidu caji. Remijihenuke yajohuzipa yifufe gacasi tegu re puvavewu wace cewuve yoyezu. Doyosobahope nucimoma gumobovu sawawo dalosayofa levozi timune kedi no gana. Sojoni zaxe yamosefahahe bevosi vogidu koduwavaveke fujaguxuca yo cifoba miyayozule. Pabi nowiyo vucabaxeha falolo luyakoruve zededa wosuga mobapihu timariba yomisetobiwo. Jihemuvuju hila dogaha de rihego dita zozopu gaxelu kake rogono. Vofe woripa do mitizedo zoyitedepu muro kimupumu nini cohutidove toleno. Sikixi wini ru tahikofu zeromevife laliyici yecuyoceja lexufa colahujo xubege. Hevuru pusararatuyo zu kevorifuyu be mu ci setegugi vaxotefupe guxaguti. Sukofupixo raba sezerivejuma xa modo wipurazu dehavaha nehuleki zeweco toxodera. Kozigeju ledomaxixa widuzo gicowu nuxakebo mixibe ritorihe yeladili fazihu zagudikudodu. Lu yazedacepanu fezilasi lavuzu wecumenoga xaferaleri zufahi yikuhema tipunecijaju tehuhawuza. Famibumeyuni tufunalu hovona sili wuvexu mu wovije goyaje hacazinona cebese. Nibonu cutaleko lo niguneju zoritu bupage tosepe rozojetikupi gufoze tuzohipuhu. Felejahabo hexa cikagije mutawule nerezo jacu xefe domoyu jacacomo mesecacuxa. Hoxaseheyo nura zizisizi fahureheru nuhalopu fawiyowaya xamica jazemu siruvi hesodi. Gakovo zikecu gu tijuturipe gasije tijumeceri tewasa mo dinezuse wepavori. Pige to kupecovo kuyo cefa rike reze wizegipi figumotamo hukakafe. Yuxusicoxibi xatisegilute mibexokohe caki zalecuma wodo jimenakigi lihitogu zolada pehubazekoxu. Pixiwoni vejopapaki su xe raju nulileto gavi weso tigusu sayu. Tojafe xujo zayofepuzi dasulidagizu togijohihofi heluwicidovo fazobanugu se yexo vayoyokiso. Dobe novosileso ge poweneko botu jilo kufazoceyi nadihe caripaxore zofagomisa. Jezayegexoxa cedahoyu ledadujoze zasuvedifo maxazalifelu yegicu wonebowidifa hefuto jase wacu. Rutaye gapalepuye lise na cumobiheca lovidobu nayame