

such that n does not exceed $t_k = k(k + 1)/2$, the k th triangular number, that is, $t_{k-1} < n \leq t_k$. The unsettled conjecture, known as **Frame's conjecture**, is that this algorithm uses the fewest number of moves required to solve the puzzle, no matter how the disks are moved.

38. Show that the Reve's puzzle with three disks can be solved using five, and no fewer, moves.
 39. Show that the Reve's puzzle with four disks can be solved using nine, and no fewer, moves.
 40. Describe the moves made by the Frame–Stewart algorithm, with k chosen so that the fewest moves are required, for
 - a) 5 disks.
 - b) 6 disks.
 - c) 7 disks.
 - d) 8 disks.
 - *41. Show that if $R(n)$ is the number of moves used by the Frame–Stewart algorithm to solve the Reve's puzzle with n disks, where k is chosen to be the smallest integer with $n \leq k(k + 1)/2$, then $R(n)$ satisfies the recurrence relation $R(n) = 2R(n - k) + 2^k - 1$, with $R(0) = 0$ and $R(1) = 1$.
 - *42. Show that if k is as chosen in Exercise 41, then $R(n) - R(n - 1) = 2^{k-1}$.
 - *43. Show that if k is as chosen in Exercise 41, then $R(n) = \sum_{i=1}^k i2^{i-1} - (t_k - n)2^{k-1}$.
 - *44. Use Exercise 43 to give an upper bound on the number of moves required to solve the Reve's puzzle for all integers n with $1 \leq n \leq 25$.
 - *45. Show that $R(n)$ is $O(\sqrt{n}2^{\sqrt{2n}})$.
- Let $\{a_n\}$ be a sequence of real numbers. The **backward differences** of this sequence are defined recursively as shown next. The **first difference** ∇a_n is
- $$\nabla a_n = a_n - a_{n-1}.$$
- The **($k + 1$)st difference** $\nabla^{k+1} a_n$ is obtained from $\nabla^k a_n$ by
- $$\nabla^{k+1} a_n = \nabla^k a_n - \nabla^k a_{n-1}.$$
46. Find ∇a_n for the sequence $\{a_n\}$, where
 - a) $a_n = 4$.
 - b) $a_n = 2n$.
 - c) $a_n = n^2$.
 - d) $a_n = 2^n$.
 47. Find $\nabla^2 a_n$ for the sequences in Exercise 46.
 48. Show that $a_{n-1} = a_n - \nabla a_n$.
 49. Show that $a_{n-2} = a_n - 2\nabla a_n + \nabla^2 a_n$.
 - *50. Prove that a_{n-k} can be expressed in terms of a_n , ∇a_n , $\nabla^2 a_n$, ..., $\nabla^k a_n$.
 51. Express the recurrence relation $a_n = a_{n-1} + a_{n-2}$ in terms of a_n , ∇a_n , and $\nabla^2 a_n$.
 52. Show that any recurrence relation for the sequence $\{a_n\}$ can be written in terms of a_n , ∇a_n , $\nabla^2 a_n$, The resulting equation involving the sequences and its differences is called a **difference equation**.
- *53. Construct the algorithm described in the text after Algorithm 1 for determining which talks should be scheduled to maximize the total number of attendees and not just the maximum total number of attendees determined by Algorithm 1.
54. Use Algorithm 1 to determine the maximum number of total attendees in the talks in Example 6 if w_i , the number of attendees of talk i , $i = 1, 2, \dots, 7$, is
 - a) 20, 10, 50, 30, 15, 25, 40.
 - b) 100, 5, 10, 20, 25, 40, 30.
 - c) 2, 3, 8, 5, 4, 7, 10.
 - d) 10, 8, 7, 25, 20, 30, 5.
55. For each part of Exercise 54, use your algorithm from Exercise 53 to find the optimal schedule for talks so that the total number of attendees is maximized.
56. In this exercise we will develop a dynamic programming algorithm for finding the maximum sum of consecutive terms of a sequence of real numbers. That is, given a sequence of real numbers a_1, a_2, \dots, a_n , the algorithm computes the maximum sum $\sum_{i=j}^k a_i$ where $1 \leq j \leq k \leq n$.
 - a) Show that if all terms of the sequence are nonnegative, this problem is solved by taking the sum of all terms. Then, give an example where the maximum sum of consecutive terms is not the sum of all terms.
 - b) Let $M(k)$ be the maximum of the sums of consecutive terms of the sequence ending at a_k . That is, $M(k) = \max_{1 \leq j \leq k} \sum_{i=j}^k a_i$. Explain why the recurrence relation $M(k) = \max(M(k-1) + a_k, a_k)$ holds for $k = 2, \dots, n$.
 - c) Use part (b) to develop a dynamic programming algorithm for solving this problem.
 - d) Show each step your algorithm from part (c) uses to find the maximum sum of consecutive terms of the sequence 2, -3, 4, 1, -2, 3.
 - e) Show that the worst-case complexity in terms of the number of additions and comparisons of your algorithm from part (c) is linear.
- *57. Dynamic programming can be used to develop an algorithm for solving the matrix-chain multiplication problem introduced in Section 3.3. This is the problem of determining how the product $A_1 A_2 \cdots A_n$ can be computed using the fewest integer multiplications, where A_1, A_2, \dots, A_n are $m_1 \times m_2, m_2 \times m_3, \dots, m_n \times m_{n+1}$ matrices, respectively, and each matrix has integer entries. Recall that by the associative law, the product does not depend on the order in which the matrices are multiplied.
 - a) Show that the brute-force method of determining the minimum number of integer multiplications needed to solve a matrix-chain multiplication problem has exponential worst-case complexity. [Hint: Do this by first showing that the order of multiplication of matrices is specified by parenthesizing the product. Then, use Example 5 and the result of part (c) of Exercise 41 in Section 8.4.]

b) Denote by \mathbf{A}_{ij} the product $\mathbf{A}_i \mathbf{A}_{i+1} \dots \mathbf{A}_j$, and $M(i, j)$ the minimum number of integer multiplications required to find \mathbf{A}_{ij} . Show that if the least number of integer multiplications are used to compute \mathbf{A}_{ij} , where $i < j$, by splitting the product into the product of \mathbf{A}_i through \mathbf{A}_k and the product of \mathbf{A}_{k+1} through \mathbf{A}_j , then the first k terms must be parenthesized so that \mathbf{A}_{ik} is computed in the optimal way using $M(i, k)$ integer multiplications and $\mathbf{A}_{k+1,j}$ must be parenthesized so that $\mathbf{A}_{k+1,j}$ is computed in the optimal way using $M(k+1, j)$ integer multiplications.

- c) Explain why part (b) leads to the recurrence relation $M(i, j) = \min_{i \leq k < j} (M(i, k) + M(k+1, j) + m_i m_{k+1} m_{j+1})$ if $1 \leq i \leq j \leq n$.
- d) Use the recurrence relation in part (c) to construct an efficient algorithm for determining the order the n matrices should be multiplied to use the minimum number of integer multiplications. Store the partial results $M(i, j)$ as you find them so that your algorithm will not have exponential complexity.
- e) Show that your algorithm from part (d) has $O(n^3)$ worst-case complexity in terms of multiplications of integers.

8.2 Solving Linear Recurrence Relations

Introduction



A wide variety of recurrence relations occur in models. Some of these recurrence relations can be solved using iteration or some other ad hoc technique. However, one important class of recurrence relations can be explicitly solved in a systematic way. These are recurrence relations that express the terms of a sequence as linear combinations of previous terms.

DEFINITION 1

A *linear homogeneous recurrence relation of degree k with constant coefficients* is a recurrence relation of the form

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \dots + c_k a_{n-k},$$

where c_1, c_2, \dots, c_k are real numbers, and $c_k \neq 0$.

The recurrence relation in the definition is **linear** because the right-hand side is a sum of previous terms of the sequence each multiplied by a function of n . The recurrence relation is **homogeneous** because no terms occur that are not multiples of the a_j s. The coefficients of the terms of the sequence are all **constants**, rather than functions that depend on n . The **degree** is k because a_n is expressed in terms of the previous k terms of the sequence.

A consequence of the second principle of mathematical induction is that a sequence satisfying the recurrence relation in the definition is uniquely determined by this recurrence relation and the k initial conditions

$$a_0 = C_0, a_1 = C_1, \dots, a_{k-1} = C_{k-1}.$$

EXAMPLE 1 The recurrence relation $P_n = (1.11)P_{n-1}$ is a linear homogeneous recurrence relation of degree one. The recurrence relation $f_n = f_{n-1} + f_{n-2}$ is a linear homogeneous recurrence relation of degree two. The recurrence relation $a_n = a_{n-5}$ is a linear homogeneous recurrence relation of degree five.

Example 2 presents some examples of recurrence relations that are not linear homogeneous recurrence relations with constant coefficients.

EXAMPLE 2 The recurrence relation $a_n = a_{n-1} + a_{n-2}^2$ is not linear. The recurrence relation $H_n = 2H_{n-1} + 1$ is not homogeneous. The recurrence relation $B_n = nB_{n-1}$ does not have constant coefficients.

Linear homogeneous recurrence relations are studied for two reasons. First, they often occur in modeling of problems. Second, they can be systematically solved.

Solving Linear Homogeneous Recurrence Relations with Constant Coefficients

The basic approach for solving linear homogeneous recurrence relations is to look for solutions of the form $a_n = r^n$, where r is a constant. Note that $a_n = r^n$ is a solution of the recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2} + \cdots + c_ka_{n-k}$ if and only if

$$r^n = c_1r^{n-1} + c_2r^{n-2} + \cdots + c_kr^{n-k}.$$

When both sides of this equation are divided by r^{n-k} and the right-hand side is subtracted from the left, we obtain the equation

$$r^k - c_1r^{k-1} - c_2r^{k-2} - \cdots - c_{k-1}r - c_k = 0.$$

Consequently, the sequence $\{a_n\}$ with $a_n = r^n$ is a solution if and only if r is a solution of this last equation. We call this the **characteristic equation** of the recurrence relation. The solutions of this equation are called the **characteristic roots** of the recurrence relation. As we will see, these characteristic roots can be used to give an explicit formula for all the solutions of the recurrence relation.

We will first develop results that deal with linear homogeneous recurrence relations with constant coefficients of degree two. Then corresponding general results when the degree may be greater than two will be stated. Because the proofs needed to establish the results in the general case are more complicated, they will not be given here.

We now turn our attention to linear homogeneous recurrence relations of degree two. First, consider the case when there are two distinct characteristic roots.

THEOREM 1

Let c_1 and c_2 be real numbers. Suppose that $r^2 - c_1r - c_2 = 0$ has two distinct roots r_1 and r_2 . Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2}$ if and only if $a_n = \alpha_1r_1^n + \alpha_2r_2^n$ for $n = 0, 1, 2, \dots$, where α_1 and α_2 are constants.

Proof: We must do two things to prove the theorem. First, it must be shown that if r_1 and r_2 are the roots of the characteristic equation, and α_1 and α_2 are constants, then the sequence $\{a_n\}$ with $a_n = \alpha_1r_1^n + \alpha_2r_2^n$ is a solution of the recurrence relation. Second, it must be shown that if the sequence $\{a_n\}$ is a solution, then $a_n = \alpha_1r_1^n + \alpha_2r_2^n$ for some constants α_1 and α_2 .

Now we will show that if $a_n = \alpha_1r_1^n + \alpha_2r_2^n$, then the sequence $\{a_n\}$ is a solution of the recurrence relation. Because r_1 and r_2 are roots of $r^2 - c_1r - c_2 = 0$, it follows that $r_1^2 = c_1r_1 + c_2$, $r_2^2 = c_1r_2 + c_2$.

From these equations, we see that

$$\begin{aligned} c_1a_{n-1} + c_2a_{n-2} &= c_1(\alpha_1r_1^{n-1} + \alpha_2r_2^{n-1}) + c_2(\alpha_1r_1^{n-2} + \alpha_2r_2^{n-2}) \\ &= \alpha_1r_1^{n-2}(c_1r_1 + c_2) + \alpha_2r_2^{n-2}(c_1r_2 + c_2) \\ &= \alpha_1r_1^{n-2}r_1^2 + \alpha_2r_2^{n-2}r_2^2 \\ &= \alpha_1r_1^n + \alpha_2r_2^n \\ &= a_n. \end{aligned}$$

This shows that the sequence $\{a_n\}$ with $a_n = \alpha_1r_1^n + \alpha_2r_2^n$ is a solution of the recurrence relation.

To show that every solution $\{a_n\}$ of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ has $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ for $n = 0, 1, 2, \dots$, for some constants α_1 and α_2 , suppose that $\{a_n\}$ is a solution of the recurrence relation, and the initial conditions $a_0 = C_0$ and $a_1 = C_1$ hold. It will be shown that there are constants α_1 and α_2 such that the sequence $\{a_n\}$ with $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ satisfies these same initial conditions. This requires that

$$\begin{aligned} a_0 &= C_0 = \alpha_1 + \alpha_2, \\ a_1 &= C_1 = \alpha_1 r_1 + \alpha_2 r_2. \end{aligned}$$

We can solve these two equations for α_1 and α_2 . From the first equation it follows that $\alpha_2 = C_0 - \alpha_1$. Inserting this expression into the second equation gives

$$C_1 = \alpha_1 r_1 + (C_0 - \alpha_1) r_2.$$

Hence,

$$C_1 = \alpha_1(r_1 - r_2) + C_0 r_2.$$

This shows that

$$\alpha_1 = \frac{C_1 - C_0 r_2}{r_1 - r_2}$$

and

$$\alpha_2 = C_0 - \alpha_1 = C_0 - \frac{C_1 - C_0 r_2}{r_1 - r_2} = \frac{C_0 r_1 - C_1}{r_1 - r_2},$$

where these expressions for α_1 and α_2 depend on the fact that $r_1 \neq r_2$. (When $r_1 = r_2$, this theorem is not true.) Hence, with these values for α_1 and α_2 , the sequence $\{a_n\}$ with $\alpha_1 r_1^n + \alpha_2 r_2^n$ satisfies the two initial conditions.

We know that $\{a_n\}$ and $\{\alpha_1 r_1^n + \alpha_2 r_2^n\}$ are both solutions of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ and both satisfy the initial conditions when $n = 0$ and $n = 1$. Because there is a unique solution of a linear homogeneous recurrence relation of degree two with two initial conditions, it follows that the two solutions are the same, that is, $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ for all nonnegative integers n . We have completed the proof by showing that a solution of the linear homogeneous recurrence relation with constant coefficients of degree two must be of the form $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$, where α_1 and α_2 are constants. \triangleleft

The characteristic roots of a linear homogeneous recurrence relation with constant coefficients may be complex numbers. Theorem 1 (and also subsequent theorems in this section) still applies in this case. Recurrence relations with complex characteristic roots will not be discussed in the text. Readers familiar with complex numbers may wish to solve Exercises 38 and 39.

Examples 3 and 4 show how to use Theorem 1 to solve recurrence relations.

EXAMPLE 3 What is the solution of the recurrence relation

$$a_n = a_{n-1} + 2a_{n-2}$$

with $a_0 = 2$ and $a_1 = 7$?



Solution: Theorem 1 can be used to solve this problem. The characteristic equation of the recurrence relation is $r^2 - r - 2 = 0$. Its roots are $r = 2$ and $r = -1$. Hence, the sequence $\{a_n\}$ is a solution to the recurrence relation if and only if

$$a_n = \alpha_1 2^n + \alpha_2 (-1)^n,$$

for some constants α_1 and α_2 . From the initial conditions, it follows that

$$\begin{aligned} a_0 &= 2 = \alpha_1 + \alpha_2, \\ a_1 &= 7 = \alpha_1 \cdot 2 + \alpha_2 \cdot (-1). \end{aligned}$$

Solving these two equations shows that $\alpha_1 = 3$ and $\alpha_2 = -1$. Hence, the solution to the recurrence relation and initial conditions is the sequence $\{a_n\}$ with

$$a_n = 3 \cdot 2^n - (-1)^n.$$

EXAMPLE 4 Find an explicit formula for the Fibonacci numbers.

Solution: Recall that the sequence of Fibonacci numbers satisfies the recurrence relation $f_n = f_{n-1} + f_{n-2}$ and also satisfies the initial conditions $f_0 = 0$ and $f_1 = 1$. The roots of the characteristic equation $r^2 - r - 1 = 0$ are $r_1 = (1 + \sqrt{5})/2$ and $r_2 = (1 - \sqrt{5})/2$. Therefore, from Theorem 1 it follows that the Fibonacci numbers are given by

$$f_n = \alpha_1 \left(\frac{1 + \sqrt{5}}{2} \right)^n + \alpha_2 \left(\frac{1 - \sqrt{5}}{2} \right)^n,$$

for some constants α_1 and α_2 . The initial conditions $f_0 = 0$ and $f_1 = 1$ can be used to find these constants. We have

$$\begin{aligned} f_0 &= \alpha_1 + \alpha_2 = 0, \\ f_1 &= \alpha_1 \left(\frac{1 + \sqrt{5}}{2} \right) + \alpha_2 \left(\frac{1 - \sqrt{5}}{2} \right) = 1. \end{aligned}$$

The solution to these simultaneous equations for α_1 and α_2 is

$$\alpha_1 = 1/\sqrt{5}, \quad \alpha_2 = -1/\sqrt{5}.$$

Consequently, the Fibonacci numbers are given by

$$f_n = \frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^n - \frac{1}{\sqrt{5}} \left(\frac{1 - \sqrt{5}}{2} \right)^n.$$

Theorem 1 does not apply when there is one characteristic root of multiplicity two. If this happens, then $a_n = nr_0^n$ is another solution of the recurrence relation when r_0 is a root of multiplicity two of the characteristic equation. Theorem 2 shows how to handle this case.

THEOREM 2

Let c_1 and c_2 be real numbers with $c_2 \neq 0$. Suppose that $r^2 - c_1r - c_2 = 0$ has only one root r_0 . A sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1a_{n-1} + c_2a_{n-2}$ if and only if $a_n = \alpha_1r_0^n + \alpha_2nr_0^n$, for $n = 0, 1, 2, \dots$, where α_1 and α_2 are constants.

The proof of Theorem 2 is left as Exercise 10. Example 5 illustrates the use of this theorem.

EXAMPLE 5 What is the solution of the recurrence relation

$$a_n = 6a_{n-1} - 9a_{n-2}$$

with initial conditions $a_0 = 1$ and $a_1 = 6$?

Solution: The only root of $r^2 - 6r + 9 = 0$ is $r = 3$. Hence, the solution to this recurrence relation is

$$a_n = \alpha_1 3^n + \alpha_2 n 3^n$$

for some constants α_1 and α_2 . Using the initial conditions, it follows that

$$\begin{aligned} a_0 &= 1 = \alpha_1, \\ a_1 &= 6 = \alpha_1 \cdot 3 + \alpha_2 \cdot 3. \end{aligned}$$

Solving these two equations shows that $\alpha_1 = 1$ and $\alpha_2 = 1$. Consequently, the solution to this recurrence relation and the initial conditions is

$$a_n = 3^n + n 3^n.$$

We will now state the general result about the solution of linear homogeneous recurrence relations with constant coefficients, where the degree may be greater than two, under the assumption that the characteristic equation has distinct roots. The proof of this result will be left as Exercise 16.

THEOREM 3

Let c_1, c_2, \dots, c_k be real numbers. Suppose that the characteristic equation

$$r^k - c_1 r^{k-1} - \cdots - c_k = 0$$

has k distinct roots r_1, r_2, \dots, r_k . Then a sequence $\{a_n\}$ is a solution of the recurrence relation

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$$

if and only if

$$a_n = \alpha_1 r_1^n + \alpha_2 r_2^n + \cdots + \alpha_k r_k^n$$

for $n = 0, 1, 2, \dots$, where $\alpha_1, \alpha_2, \dots, \alpha_k$ are constants.

We illustrate the use of the theorem with Example 6.

EXAMPLE 6 Find the solution to the recurrence relation

$$a_n = 6a_{n-1} - 11a_{n-2} + 6a_{n-3}$$

with the initial conditions $a_0 = 2$, $a_1 = 5$, and $a_2 = 15$.

Solution: The characteristic polynomial of this recurrence relation is

$$r^3 - 6r^2 + 11r - 6.$$

The characteristic roots are $r = 1$, $r = 2$, and $r = 3$, because $r^3 - 6r^2 + 11r - 6 = (r - 1)(r - 2)(r - 3)$. Hence, the solutions to this recurrence relation are of the form

$$a_n = \alpha_1 \cdot 1^n + \alpha_2 \cdot 2^n + \alpha_3 \cdot 3^n.$$

To find the constants α_1 , α_2 , and α_3 , use the initial conditions. This gives

$$\begin{aligned} a_0 &= 2 = \alpha_1 + \alpha_2 + \alpha_3, \\ a_1 &= 5 = \alpha_1 + \alpha_2 \cdot 2 + \alpha_3 \cdot 3, \\ a_2 &= 15 = \alpha_1 + \alpha_2 \cdot 4 + \alpha_3 \cdot 9. \end{aligned}$$

When these three simultaneous equations are solved for α_1 , α_2 , and α_3 , we find that $\alpha_1 = 1$, $\alpha_2 = -1$, and $\alpha_3 = 2$. Hence, the unique solution to this recurrence relation and the given initial conditions is the sequence $\{a_n\}$ with

$$a_n = 1 - 2^n + 2 \cdot 3^n.$$

We now state the most general result about linear homogeneous recurrence relations with constant coefficients, allowing the characteristic equation to have multiple roots. The key point is that for each root r of the characteristic equation, the general solution has a summand of the form $P(n)r^n$, where $P(n)$ is a polynomial of degree $m - 1$, with m the multiplicity of this root. We leave the proof of this result as Exercise 51.

THEOREM 4

Let c_1, c_2, \dots, c_k be real numbers. Suppose that the characteristic equation

$$r^k - c_1r^{k-1} - \cdots - c_k = 0$$

has t distinct roots r_1, r_2, \dots, r_t with multiplicities m_1, m_2, \dots, m_t , respectively, so that $m_i \geq 1$ for $i = 1, 2, \dots, t$ and $m_1 + m_2 + \cdots + m_t = k$. Then a sequence $\{a_n\}$ is a solution of the recurrence relation

$$a_n = c_1a_{n-1} + c_2a_{n-2} + \cdots + c_ka_{n-k}$$

if and only if

$$\begin{aligned} a_n &= (\alpha_{1,0} + \alpha_{1,1}n + \cdots + \alpha_{1,m_1-1}n^{m_1-1})r_1^n \\ &\quad + (\alpha_{2,0} + \alpha_{2,1}n + \cdots + \alpha_{2,m_2-1}n^{m_2-1})r_2^n \\ &\quad + \cdots + (\alpha_{t,0} + \alpha_{t,1}n + \cdots + \alpha_{t,m_t-1}n^{m_t-1})r_t^n \end{aligned}$$

for $n = 0, 1, 2, \dots$, where $\alpha_{i,j}$ are constants for $1 \leq i \leq t$ and $0 \leq j \leq m_i - 1$.

Example 7 illustrates how Theorem 4 is used to find the general form of a solution of a linear homogeneous recurrence relation when the characteristic equation has several repeated roots.

EXAMPLE 7 Suppose that the roots of the characteristic equation of a linear homogeneous recurrence relation are 2, 2, 2, 5, 5, and 9 (that is, there are three roots, the root 2 with multiplicity three, the root 5 with multiplicity two, and the root 9 with multiplicity one). What is the form of the general solution?

Solution: By Theorem 4, the general form of the solution is

$$(\alpha_{1,0} + \alpha_{1,1}n + \alpha_{1,2}n^2)2^n + (\alpha_{2,0} + \alpha_{2,1}n)5^n + \alpha_{3,0}9^n.$$

We now illustrate the use of Theorem 4 to solve a linear homogeneous recurrence relation with constant coefficients when the characteristic equation has a root of multiplicity three.

EXAMPLE 8 Find the solution to the recurrence relation

$$a_n = -3a_{n-1} - 3a_{n-2} - a_{n-3}$$

with initial conditions $a_0 = 1$, $a_1 = -2$, and $a_2 = -1$.

Solution: The characteristic equation of this recurrence relation is

$$r^3 + 3r^2 + 3r + 1 = 0.$$

Because $r^3 + 3r^2 + 3r + 1 = (r + 1)^3$, there is a single root $r = -1$ of multiplicity three of the characteristic equation. By Theorem 4 the solutions of this recurrence relation are of the form

$$a_n = \alpha_{1,0}(-1)^n + \alpha_{1,1}n(-1)^n + \alpha_{1,2}n^2(-1)^n.$$

To find the constants $\alpha_{1,0}$, $\alpha_{1,1}$, and $\alpha_{1,2}$, use the initial conditions. This gives

$$\begin{aligned} a_0 &= 1 = \alpha_{1,0}, \\ a_1 &= -2 = -\alpha_{1,0} - \alpha_{1,1} - \alpha_{1,2}, \\ a_2 &= -1 = \alpha_{1,0} + 2\alpha_{1,1} + 4\alpha_{1,2}. \end{aligned}$$

The simultaneous solution of these three equations is $\alpha_{1,0} = 1$, $\alpha_{1,1} = 3$, and $\alpha_{1,2} = -2$. Hence, the unique solution to this recurrence relation and the given initial conditions is the sequence $\{a_n\}$ with

$$a_n = (1 + 3n - 2n^2)(-1)^n.$$

Linear Nonhomogeneous Recurrence Relations with Constant Coefficients

We have seen how to solve linear homogeneous recurrence relations with constant coefficients. Is there a relatively simple technique for solving a linear, but not homogeneous, recurrence relation with constant coefficients, such as $a_n = 3a_{n-1} + 2n$? We will see that the answer is yes for certain families of such recurrence relations.

The recurrence relation $a_n = 3a_{n-1} + 2n$ is an example of a **linear nonhomogeneous recurrence relation with constant coefficients**, that is, a recurrence relation of the form

$$a_n = c_1a_{n-1} + c_2a_{n-2} + \cdots + c_ka_{n-k} + F(n),$$

where c_1, c_2, \dots, c_k are real numbers and $F(n)$ is a function not identically zero depending only on n . The recurrence relation

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$$

is called the **associated homogeneous recurrence relation**. It plays an important role in the solution of the nonhomogeneous recurrence relation.

EXAMPLE 9 Each of the recurrence relations $a_n = a_{n-1} + 2^n$, $a_n = a_{n-1} + a_{n-2} + n^2 + n + 1$, $a_n = 3a_{n-1} + n3^n$, and $a_n = a_{n-1} + a_{n-2} + a_{n-3} + n!$ is a linear nonhomogeneous recurrence relation with constant coefficients. The associated linear homogeneous recurrence relations are $a_n = a_{n-1}$, $a_n = a_{n-1} + a_{n-2}$, $a_n = 3a_{n-1}$, and $a_n = a_{n-1} + a_{n-2} + a_{n-3}$, respectively. \blacktriangleleft

The key fact about linear nonhomogeneous recurrence relations with constant coefficients is that every solution is the sum of a particular solution and a solution of the associated linear homogeneous recurrence relation, as Theorem 5 shows.

THEOREM 5

If $\{a_n^{(p)}\}$ is a particular solution of the nonhomogeneous linear recurrence relation with constant coefficients

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + F(n),$$

then every solution is of the form $\{a_n^{(p)} + a_n^{(h)}\}$, where $\{a_n^{(h)}\}$ is a solution of the associated homogeneous recurrence relation

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}.$$

Proof: Because $\{a_n^{(p)}\}$ is a particular solution of the nonhomogeneous recurrence relation, we know that

$$a_n^{(p)} = c_1 a_{n-1}^{(p)} + c_2 a_{n-2}^{(p)} + \cdots + c_k a_{n-k}^{(p)} + F(n).$$

Now suppose that $\{b_n\}$ is a second solution of the nonhomogeneous recurrence relation, so that

$$b_n = c_1 b_{n-1} + c_2 b_{n-2} + \cdots + c_k b_{n-k} + F(n).$$

Subtracting the first of these two equations from the second shows that

$$b_n - a_n^{(p)} = c_1(b_{n-1} - a_{n-1}^{(p)}) + c_2(b_{n-2} - a_{n-2}^{(p)}) + \cdots + c_k(b_{n-k} - a_{n-k}^{(p)}).$$

It follows that $\{b_n - a_n^{(p)}\}$ is a solution of the associated homogeneous linear recurrence, say, $\{a_n^{(h)}\}$. Consequently, $b_n = a_n^{(p)} + a_n^{(h)}$ for all n . \blacktriangleleft

By Theorem 5, we see that the key to solving nonhomogeneous recurrence relations with constant coefficients is finding a particular solution. Then every solution is a sum of this solution

and a solution of the associated homogeneous recurrence relation. Although there is no general method for finding such a solution that works for every function $F(n)$, there are techniques that work for certain types of functions $F(n)$, such as polynomials and powers of constants. This is illustrated in Examples 10 and 11.

EXAMPLE 10 Find all solutions of the recurrence relation $a_n = 3a_{n-1} + 2n$. What is the solution with $a_1 = 3$?

Solution: To solve this linear nonhomogeneous recurrence relation with constant coefficients, we need to solve its associated linear homogeneous equation and to find a particular solution for the given nonhomogeneous equation. The associated linear homogeneous equation is $a_n = 3a_{n-1}$. Its solutions are $a_n^{(h)} = \alpha 3^n$, where α is a constant.

We now find a particular solution. Because $F(n) = 2n$ is a polynomial in n of degree one, a reasonable trial solution is a linear function in n , say, $p_n = cn + d$, where c and d are constants. To determine whether there are any solutions of this form, suppose that $p_n = cn + d$ is such a solution. Then the equation $a_n = 3a_{n-1} + 2n$ becomes $cn + d = 3(c(n - 1) + d) + 2n$. Simplifying and combining like terms gives $(2 + 2c)n + (2d - 3c) = 0$. It follows that $cn + d$ is a solution if and only if $2 + 2c = 0$ and $2d - 3c = 0$. This shows that $cn + d$ is a solution if and only if $c = -1$ and $d = -3/2$. Consequently, $a_n^{(p)} = -n - 3/2$ is a particular solution.

By Theorem 5 all solutions are of the form

$$a_n = a_n^{(p)} + a_n^{(h)} = -n - \frac{3}{2} + \alpha \cdot 3^n,$$

where α is a constant.

To find the solution with $a_1 = 3$, let $n = 1$ in the formula we obtained for the general solution. We find that $3 = -1 - 3/2 + 3\alpha$, which implies that $\alpha = 11/6$. The solution we seek is $a_n = -n - 3/2 + (11/6)3^n$. 

EXAMPLE 11 Find all solutions of the recurrence relation



$$a_n = 5a_{n-1} - 6a_{n-2} + 7^n.$$

Solution: This is a linear nonhomogeneous recurrence relation. The solutions of its associated homogeneous recurrence relation

$$a_n = 5a_{n-1} - 6a_{n-2}$$

are $a_n^{(h)} = \alpha_1 \cdot 3^n + \alpha_2 \cdot 2^n$, where α_1 and α_2 are constants. Because $F(n) = 7^n$, a reasonable trial solution is $a_n^{(p)} = C \cdot 7^n$, where C is a constant. Substituting the terms of this sequence into the recurrence relation implies that $C \cdot 7^n = 5C \cdot 7^{n-1} - 6C \cdot 7^{n-2} + 7^n$. Factoring out 7^{n-2} , this equation becomes $49C = 35C - 6C + 49$, which implies that $20C = 49$, or that $C = 49/20$. Hence, $a_n^{(p)} = (49/20)7^n$ is a particular solution. By Theorem 5, all solutions are of the form

$$a_n = \alpha_1 \cdot 3^n + \alpha_2 \cdot 2^n + (49/20)7^n. \quad \blacktriangleleft$$

In Examples 10 and 11, we made an educated guess that there are solutions of a particular form. In both cases we were able to find particular solutions. This was not an accident. Whenever $F(n)$ is the product of a polynomial in n and the n th power of a constant, we know exactly what form a particular solution has, as stated in Theorem 6. We leave the proof of Theorem 6 as Exercise 52.

THEOREM 6

Suppose that $\{a_n\}$ satisfies the linear nonhomogeneous recurrence relation

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k} + F(n),$$

where c_1, c_2, \dots, c_k are real numbers, and

$$F(n) = (b_t n^t + b_{t-1} n^{t-1} + \cdots + b_1 n + b_0) s^n,$$

where b_0, b_1, \dots, b_t and s are real numbers. When s is not a root of the characteristic equation of the associated linear homogeneous recurrence relation, there is a particular solution of the form

$$(p_t n^t + p_{t-1} n^{t-1} + \cdots + p_1 n + p_0) s^n.$$

When s is a root of this characteristic equation and its multiplicity is m , there is a particular solution of the form

$$n^m (p_t n^t + p_{t-1} n^{t-1} + \cdots + p_1 n + p_0) s^n.$$

Note that in the case when s is a root of multiplicity m of the characteristic equation of the associated linear homogeneous recurrence relation, the factor n^m ensures that the proposed particular solution will not already be a solution of the associated linear homogeneous recurrence relation. We next provide Example 12 to illustrate the form of a particular solution provided by Theorem 6.

EXAMPLE 12

What form does a particular solution of the linear nonhomogeneous recurrence relation $a_n = 6a_{n-1} - 9a_{n-2} + F(n)$ have when $F(n) = 3^n$, $F(n) = n3^n$, $F(n) = n^2 2^n$, and $F(n) = (n^2 + 1)3^n$?

Solution: The associated linear homogeneous recurrence relation is $a_n = 6a_{n-1} - 9a_{n-2}$. Its characteristic equation, $r^2 - 6r + 9 = (r - 3)^2 = 0$, has a single root, 3, of multiplicity two. To apply Theorem 6, with $F(n)$ of the form $P(n)s^n$, where $P(n)$ is a polynomial and s is a constant, we need to ask whether s is a root of this characteristic equation.

Because $s = 3$ is a root with multiplicity $m = 2$ but $s = 2$ is not a root, Theorem 6 tells us that a particular solution has the form $p_0 n^2 3^n$ if $F(n) = 3^n$, the form $n^2(p_1 n + p_0)3^n$ if $F(n) = n3^n$, the form $(p_2 n^2 + p_1 n + p_0)2^n$ if $F(n) = n^2 2^n$, and the form $n^2(p_2 n^2 + p_1 n + p_0)3^n$ if $F(n) = (n^2 + 1)3^n$. 

Care must be taken when $s = 1$ when solving recurrence relations of the type covered by Theorem 6. In particular, to apply this theorem with $F(n) = b_t n^t + b_{t-1} n^{t-1} + \cdots + b_1 n + b_0$, the parameter s takes the value $s = 1$ (even though the term 1^n does not explicitly appear). By the theorem, the form of the solution then depends on whether 1 is a root of the characteristic equation of the associated linear homogeneous recurrence relation. This is illustrated in Example 13, which shows how Theorem 6 can be used to find a formula for the sum of the first n positive integers.

EXAMPLE 13

Let a_n be the sum of the first n positive integers, so that

$$a_n = \sum_{k=1}^n k.$$

Note that a_n satisfies the linear nonhomogeneous recurrence relation

$$a_n = a_{n-1} + n.$$

(To obtain a_n , the sum of the first n positive integers, from a_{n-1} , the sum of the first $n-1$ positive integers, we add n .) Note that the initial condition is $a_1 = 1$.

The associated linear homogeneous recurrence relation for a_n is

$$a_n = a_{n-1}.$$

The solutions of this homogeneous recurrence relation are given by $a_n^{(h)} = c(1)^n = c$, where c is a constant. To find all solutions of $a_n = a_{n-1} + n$, we need find only a single particular solution. By Theorem 6, because $F(n) = n = n \cdot (1)^n$ and $s = 1$ is a root of degree one of the characteristic equation of the associated linear homogeneous recurrence relation, there is a particular solution of the form $n(p_1n + p_0) = p_1n^2 + p_0n$.

Inserting this into the recurrence relation gives $p_1n^2 + p_0n = p_1(n-1)^2 + p_0(n-1) + n$. Simplifying, we see that $n(2p_1 - 1) + (p_0 - p_1) = 0$, which means that $2p_1 - 1 = 0$ and $p_0 - p_1 = 0$, so $p_0 = p_1 = 1/2$. Hence,

$$a_n^{(p)} = \frac{n^2}{2} + \frac{n}{2} = \frac{n(n+1)}{2}$$

is a particular solution. Hence, all solutions of the original recurrence relation $a_n = a_{n-1} + n$ are given by $a_n = a_n^{(h)} + a_n^{(p)} = c + n(n+1)/2$. Because $a_1 = 1$, we have $1 = a_1 = c + 1 \cdot 2/2 = c + 1$, so $c = 0$. It follows that $a_n = n(n+1)/2$. (This is the same formula given in Table 2 in Section 2.4 and derived previously.) 

Exercises

1. Determine which of these are linear homogeneous recurrence relations with constant coefficients. Also, find the degree of those that are.
 - a) $a_n = 3a_{n-1} + 4a_{n-2} + 5a_{n-3}$
 - b) $a_n = 2na_{n-1} + a_{n-2}$
 - c) $a_n = a_{n-1} + a_{n-4}$
 - d) $a_n = a_{n-1} + 2$
 - e) $a_n = a_{n-1}^2 + a_{n-2}$
 - f) $a_n = a_{n-2}$
 - g) $a_n = a_{n-1} + n$
2. Determine which of these are linear homogeneous recurrence relations with constant coefficients. Also, find the degree of those that are.
 - a) $a_n = 3a_{n-2}$
 - b) $a_n = 3$
 - c) $a_n = a_{n-1}^2$
 - d) $a_n = a_{n-1} + 2a_{n-3}$
 - e) $a_n = a_{n-1}/n$
 - f) $a_n = a_{n-1} + a_{n-2} + n + 3$
 - g) $a_n = 4a_{n-2} + 5a_{n-4} + 9a_{n-7}$
3. Solve these recurrence relations together with the initial conditions given.
 - a) $a_n = 2a_{n-1}$ for $n \geq 1$, $a_0 = 3$
 - b) $a_n = a_{n-1}$ for $n \geq 1$, $a_0 = 2$
 - c) $a_n = 5a_{n-1} - 6a_{n-2}$ for $n \geq 2$, $a_0 = 1$, $a_1 = 0$
 - d) $a_n = 4a_{n-1} - 4a_{n-2}$ for $n \geq 2$, $a_0 = 6$, $a_1 = 8$
 - e) $a_n = -4a_{n-1} - 4a_{n-2}$ for $n \geq 2$, $a_0 = 0$, $a_1 = 1$
 - f) $a_n = 4a_{n-2}$ for $n \geq 2$, $a_0 = 0$, $a_1 = 4$
 - g) $a_n = a_{n-2}/4$ for $n \geq 2$, $a_0 = 1$, $a_1 = 0$
4. Solve these recurrence relations together with the initial conditions given.
 - a) $a_n = a_{n-1} + 6a_{n-2}$ for $n \geq 2$, $a_0 = 3$, $a_1 = 6$
 - b) $a_n = 7a_{n-1} - 10a_{n-2}$ for $n \geq 2$, $a_0 = 2$, $a_1 = 1$
 - c) $a_n = 6a_{n-1} - 8a_{n-2}$ for $n \geq 2$, $a_0 = 4$, $a_1 = 10$
 - d) $a_n = 2a_{n-1} - a_{n-2}$ for $n \geq 2$, $a_0 = 4$, $a_1 = 1$
 - e) $a_n = a_{n-2}$ for $n \geq 2$, $a_0 = 5$, $a_1 = -1$
 - f) $a_n = -6a_{n-1} - 9a_{n-2}$ for $n \geq 2$, $a_0 = 3$, $a_1 = -3$
 - g) $a_{n+2} = -4a_{n+1} + 5a_n$ for $n \geq 0$, $a_0 = 2$, $a_1 = 8$
5. How many different messages can be transmitted in n microseconds using the two signals described in Exercise 19 in Section 8.1?
6. How many different messages can be transmitted in n microseconds using three different signals if one signal requires 1 microsecond for transmittal, the other two signals require 2 microseconds each for transmittal, and a signal in a message is followed immediately by the next signal?
7. In how many ways can a $2 \times n$ rectangular checkerboard be tiled using 1×2 and 2×2 pieces?
8. A model for the number of lobsters caught per year is based on the assumption that the number of lobsters caught in a year is the average of the number caught in the two previous years.

- a) Find a recurrence relation for $\{L_n\}$, where L_n is the number of lobsters caught in year n , under the assumption for this model.
- b) Find L_n if 100,000 lobsters were caught in year 1 and 300,000 were caught in year 2.
9. A deposit of \$100,000 is made to an investment fund at the beginning of a year. On the last day of each year two dividends are awarded. The first dividend is 20% of the amount in the account during that year. The second dividend is 45% of the amount in the account in the previous year.
- a) Find a recurrence relation for $\{P_n\}$, where P_n is the amount in the account at the end of n years if no money is ever withdrawn.
- b) How much is in the account after n years if no money has been withdrawn?
- *10. Prove Theorem 2.
11. The **Lucas numbers** satisfy the recurrence relation
- 
- $$L_n = L_{n-1} + L_{n-2},$$
- and the initial conditions $L_0 = 2$ and $L_1 = 1$.
- a) Show that $L_n = f_{n-1} + f_{n+1}$ for $n = 2, 3, \dots$, where f_n is the n th Fibonacci number.
- b) Find an explicit formula for the Lucas numbers.
12. Find the solution to $a_n = 2a_{n-1} + a_{n-2} - 2a_{n-3}$ for $n = 3, 4, 5, \dots$, with $a_0 = 3$, $a_1 = 6$, and $a_2 = 0$.
13. Find the solution to $a_n = 7a_{n-2} + 6a_{n-3}$ with $a_0 = 9$, $a_1 = 10$, and $a_2 = 32$.
14. Find the solution to $a_n = 5a_{n-2} - 4a_{n-4}$ with $a_0 = 3$, $a_1 = 2$, $a_2 = 6$, and $a_3 = 8$.
15. Find the solution to $a_n = 2a_{n-1} + 5a_{n-2} - 6a_{n-3}$ with $a_0 = 7$, $a_1 = -4$, and $a_2 = 8$.
- *16. Prove Theorem 3.
17. Prove this identity relating the Fibonacci numbers and the binomial coefficients:
- $$f_{n+1} = C(n, 0) + C(n-1, 1) + \cdots + C(n-k, k),$$
- where n is a positive integer and $k = \lfloor n/2 \rfloor$. [Hint: Let $a_n = C(n, 0) + C(n-1, 1) + \cdots + C(n-k, k)$. Show that the sequence $\{a_n\}$ satisfies the same recurrence relation and initial conditions satisfied by the sequence of Fibonacci numbers.]
18. Solve the recurrence relation $a_n = 6a_{n-1} - 12a_{n-2} + 8a_{n-3}$ with $a_0 = -5$, $a_1 = 4$, and $a_2 = 88$.
19. Solve the recurrence relation $a_n = -3a_{n-1} - 3a_{n-2} - a_{n-3}$ with $a_0 = 5$, $a_1 = -9$, and $a_2 = 15$.
20. Find the general form of the solutions of the recurrence relation $a_n = 8a_{n-2} - 16a_{n-4}$.
21. What is the general form of the solutions of a linear homogeneous recurrence relation if its characteristic equation has roots $1, 1, 1, 1, -2, -2, -2, 3, 3, -4$?
22. What is the general form of the solutions of a linear homogeneous recurrence relation if its characteristic equation has the roots $-1, -1, -1, 2, 2, 5, 5, 7$?
23. Consider the nonhomogeneous linear recurrence relation $a_n = 3a_{n-1} + 2^n$.
- a) Show that $a_n = -2^{n+1}$ is a solution of this recurrence relation.
- b) Use Theorem 5 to find all solutions of this recurrence relation.
- c) Find the solution with $a_0 = 1$.
24. Consider the nonhomogeneous linear recurrence relation $a_n = 2a_{n-1} + 2^n$.
- a) Show that $a_n = n2^n$ is a solution of this recurrence relation.
- b) Use Theorem 5 to find all solutions of this recurrence relation.
- c) Find the solution with $a_0 = 2$.
25. a) Determine values of the constants A and B such that $a_n = An + B$ is a solution of recurrence relation $a_n = 2a_{n-1} + n + 5$.
- b) Use Theorem 5 to find all solutions of this recurrence relation.
- c) Find the solution of this recurrence relation with $a_0 = 4$.
26. What is the general form of the particular solution guaranteed to exist by Theorem 6 of the linear nonhomogeneous recurrence relation $a_n = 6a_{n-1} - 12a_{n-2} + 8a_{n-3} + F(n)$ if
- a) $F(n) = n^2$? b) $F(n) = 2^n$?
c) $F(n) = n2^n$? d) $F(n) = (-2)^n$?
e) $F(n) = n^22^n$? f) $F(n) = n^3(-2)^n$?
g) $F(n) = 3$?
27. What is the general form of the particular solution guaranteed to exist by Theorem 6 of the linear nonhomogeneous recurrence relation $a_n = 8a_{n-2} - 16a_{n-4} + F(n)$ if
- a) $F(n) = n^3$? b) $F(n) = (-2)^n$?
c) $F(n) = n2^n$? d) $F(n) = n^24^n$?
e) $F(n) = (n^2 - 2)(-2)^n$? f) $F(n) = n^42^n$?
g) $F(n) = 2$?
28. a) Find all solutions of the recurrence relation $a_n = 2a_{n-1} + 2n^2$.
b) Find the solution of the recurrence relation in part (a) with initial condition $a_1 = 4$.
29. a) Find all solutions of the recurrence relation $a_n = 2a_{n-1} + 3^n$.
b) Find the solution of the recurrence relation in part (a) with initial condition $a_1 = 5$.
30. a) Find all solutions of the recurrence relation $a_n = -5a_{n-1} - 6a_{n-2} + 42 \cdot 4^n$.
b) Find the solution of this recurrence relation with $a_1 = 56$ and $a_2 = 278$.
31. Find all solutions of the recurrence relation $a_n = 5a_{n-1} - 6a_{n-2} + 2^n + 3n$. [Hint: Look for a particular solution of the form $qn2^n + p_1n + p_2$, where q , p_1 , and p_2 are constants.]
32. Find the solution of the recurrence relation $a_n = 2a_{n-1} + 3 \cdot 2^n$.
33. Find all solutions of the recurrence relation $a_n = 4a_{n-1} - 4a_{n-2} + (n+1)2^n$.

- 34.** Find all solutions of the recurrence relation $a_n = 7a_{n-1} - 16a_{n-2} + 12a_{n-3} + n4^n$ with $a_0 = -2$, $a_1 = 0$, and $a_2 = 5$.
- 35.** Find the solution of the recurrence relation $a_n = 4a_{n-1} - 3a_{n-2} + 2^n + n + 3$ with $a_0 = 1$ and $a_1 = 4$.
- 36.** Let a_n be the sum of the first n perfect squares, that is, $a_n = \sum_{k=1}^n k^2$. Show that the sequence $\{a_n\}$ satisfies the linear nonhomogeneous recurrence relation $a_n = a_{n-1} + n^2$ and the initial condition $a_1 = 1$. Use Theorem 6 to determine a formula for a_n by solving this recurrence relation.
- 37.** Let a_n be the sum of the first n triangular numbers, that is, $a_n = \sum_{k=1}^n t_k$, where $t_k = k(k+1)/2$. Show that $\{a_n\}$ satisfies the linear nonhomogeneous recurrence relation $a_n = a_{n-1} + n(n+1)/2$ and the initial condition $a_1 = 1$. Use Theorem 6 to determine a formula for a_n by solving this recurrence relation.
- 38. a)** Find the characteristic roots of the linear homogeneous recurrence relation $a_n = 2a_{n-1} - 2a_{n-2}$. [Note: These are complex numbers.]
b) Find the solution of the recurrence relation in part (a) with $a_0 = 1$ and $a_1 = 2$.
- *39. a)** Find the characteristic roots of the linear homogeneous recurrence relation $a_n = a_{n-4}$. [Note: These include complex numbers.]
b) Find the solution of the recurrence relation in part (a) with $a_0 = 1$, $a_1 = 0$, $a_2 = -1$, and $a_3 = 1$.
- *40.** Solve the simultaneous recurrence relations
- $$a_n = 3a_{n-1} + 2b_{n-1}$$
- $$b_n = a_{n-1} + 2b_{n-1}$$
- with $a_0 = 1$ and $b_0 = 2$.
- *41. a)** Use the formula found in Example 4 for f_n , the n th Fibonacci number, to show that f_n is the integer closest to
- $$\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n.$$
- b)** Determine for which n f_n is greater than
- $$\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n$$
- and for which n f_n is less than
- $$\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n.$$
- 42.** Show that if $a_n = a_{n-1} + a_{n-2}$, $a_0 = s$ and $a_1 = t$, where s and t are constants, then $a_n = sf_{n-1} + tf_n$ for all positive integers n .
- 43.** Express the solution of the linear nonhomogeneous recurrence relation $a_n = a_{n-1} + a_{n-2} + 1$ for $n \geq 2$

where $a_0 = 0$ and $a_1 = 1$ in terms of the Fibonacci numbers. [Hint: Let $b_n = a_n + 1$ and apply Exercise 42 to the sequence b_n .]

- *44. (Linear algebra required)** Let \mathbf{A}_n be the $n \times n$ matrix with 2s on its main diagonal, 1s in all positions next to a diagonal element, and 0s everywhere else. Find a recurrence relation for d_n , the determinant of \mathbf{A}_n . Solve this recurrence relation to find a formula for d_n .
- 45.** Suppose that each pair of a genetically engineered species of rabbits left on an island produces two new pairs of rabbits at the age of 1 month and six new pairs of rabbits at the age of 2 months and every month afterward. None of the rabbits ever die or leave the island.
- a)** Find a recurrence relation for the number of pairs of rabbits on the island n months after one newborn pair is left on the island.
- b)** By solving the recurrence relation in (a) determine the number of pairs of rabbits on the island n months after one pair is left on the island.
- 46.** Suppose that there are two goats on an island initially. The number of goats on the island doubles every year by natural reproduction, and some goats are either added or removed each year.
- a)** Construct a recurrence relation for the number of goats on the island at the start of the n th year, assuming that during each year an extra 100 goats are put on the island.
- b)** Solve the recurrence relation from part (a) to find the number of goats on the island at the start of the n th year.
- c)** Construct a recurrence relation for the number of goats on the island at the start of the n th year, assuming that n goats are removed during the n th year for each $n \geq 3$.
- d)** Solve the recurrence relation in part (c) for the number of goats on the island at the start of the n th year.
- 47.** A new employee at an exciting new software company starts with a salary of \$50,000 and is promised that at the end of each year her salary will be double her salary of the previous year, with an extra increment of \$10,000 for each year she has been with the company.
- a)** Construct a recurrence relation for her salary for her n th year of employment.
- b)** Solve this recurrence relation to find her salary for her n th year of employment.
- Some linear recurrence relations that do not have constant coefficients can be systematically solved. This is the case for recurrence relations of the form $f(n)a_n = g(n)a_{n-1} + h(n)$. Exercises 48–50 illustrate this.
- *48. a)** Show that the recurrence relation
- $$f(n)a_n = g(n)a_{n-1} + h(n),$$
- for $n \geq 1$, and with $a_0 = C$, can be reduced to a recurrence relation of the form
- $$b_n = b_{n-1} + Q(n)h(n),$$

where $b_n = g(n+1)Q(n+1)a_n$, with
 $Q(n) = (f(1)f(2)\cdots f(n-1))/(g(1)g(2)\cdots g(n))$.

- b)** Use part (a) to solve the original recurrence relation to obtain

$$a_n = \frac{C + \sum_{i=1}^n Q(i)h(i)}{g(n+1)Q(n+1)}.$$

***49.** Use Exercise 48 to solve the recurrence relation $(n+1)a_n = (n+3)a_{n-1} + n$, for $n \geq 1$, with $a_0 = 1$.

50. It can be shown that C_n , the average number of comparisons made by the quick sort algorithm (described in preamble to Exercise 50 in Section 5.4), when sorting n elements in random order, satisfies the recurrence relation

$$C_n = n + 1 + \frac{2}{n} \sum_{k=0}^{n-1} C_k$$

for $n = 1, 2, \dots$, with initial condition $C_0 = 0$.

- a)** Show that $\{C_n\}$ also satisfies the recurrence relation $nC_n = (n+1)C_{n-1} + 2n$ for $n = 1, 2, \dots$
b) Use Exercise 48 to solve the recurrence relation in part (a) to find an explicit formula for C_n .

****51.** Prove Theorem 4.

****52.** Prove Theorem 6.

- 53.** Solve the recurrence relation $T(n) = nT^2(n/2)$ with initial condition $T(1) = 6$ when $n = 2^k$ for some integer k . [Hint: Let $n = 2^k$ and then make the substitution $a_k = \log T(2^k)$ to obtain a linear nonhomogeneous recurrence relation.]

8.3 Divide-and-Conquer Algorithms and Recurrence Relations

Introduction



“Divide et impera”
 (translation: “Divide and conquer” - Julius Caesar

Many recursive algorithms take a problem with a given input and divide it into one or more smaller problems. This reduction is successively applied until the solutions of the smaller problems can be found quickly. For instance, we perform a binary search by reducing the search for an element in a list to the search for this element in a list half as long. We successively apply this reduction until one element is left. When we sort a list of integers using the merge sort, we split the list into two halves of equal size and sort each half separately. We then merge the two sorted halves. Another example of this type of recursive algorithm is a procedure for multiplying integers that reduces the problem of the multiplication of two integers to three multiplications of pairs of integers with half as many bits. This reduction is successively applied until integers with one bit are obtained. These procedures follow an important algorithmic paradigm known as **divide-and-conquer**, and are called **divide-and-conquer algorithms**, because they *divide* a problem into one or more instances of the same problem of smaller size and they *conquer* the problem by using the solutions of the smaller problems to find a solution of the original problem, perhaps with some additional work.

In this section we will show how recurrence relations can be used to analyze the computational complexity of divide-and-conquer algorithms. We will use these recurrence relations to estimate the number of operations used by many different divide-and-conquer algorithms, including several that we introduce in this section.

Divide-and-Conquer Recurrence Relations

Suppose that a recursive algorithm divides a problem of size n into a subproblems, where each subproblem is of size n/b (for simplicity, assume that n is a multiple of b ; in reality, the smaller problems are often of size equal to the nearest integers either less than or equal to, or greater than or equal to, n/b). Also, suppose that a total of $g(n)$ extra operations are required in the conquer step of the algorithm to combine the solutions of the subproblems into a solution of the original problem. Then, if $f(n)$ represents the number of operations required to solve the problem of size n , it follows that f satisfies the recurrence relation

$$f(n) = af(n/b) + g(n).$$

This is called a **divide-and-conquer recurrence relation**.

We will first set up the divide-and-conquer recurrence relations that can be used to study the complexity of some important algorithms. Then we will show how to use these divide-and-conquer recurrence relations to estimate the complexity of these algorithms.

EXAMPLE 1

Binary Search We introduced a binary search algorithm in Section 3.1. This binary search algorithm reduces the search for an element in a search sequence of size n to the binary search for this element in a search sequence of size $n/2$, when n is even. (Hence, the problem of size n has been reduced to *one* problem of size $n/2$.) Two comparisons are needed to implement this reduction (one to determine which half of the list to use and the other to determine whether any terms of the list remain). Hence, if $f(n)$ is the number of comparisons required to search for an element in a search sequence of size n , then

$$f(n) = f(n/2) + 2$$

when n is even.

EXAMPLE 2

Finding the Maximum and Minimum of a Sequence Consider the following algorithm for locating the maximum and minimum elements of a sequence a_1, a_2, \dots, a_n . If $n = 1$, then a_1 is the maximum and the minimum. If $n > 1$, split the sequence into two sequences, either where both have the same number of elements or where one of the sequences has one more element than the other. The problem is reduced to finding the maximum and minimum of each of the two smaller sequences. The solution to the original problem results from the comparison of the separate maxima and minima of the two smaller sequences to obtain the overall maximum and minimum.

Let $f(n)$ be the total number of comparisons needed to find the maximum and minimum elements of the sequence with n elements. We have shown that a problem of size n can be reduced into two problems of size $n/2$, when n is even, using two comparisons, one to compare the maxima of the two sequences and the other to compare the minima of the two sequences. This gives the recurrence relation

$$f(n) = 2f(n/2) + 2$$

when n is even.

EXAMPLE 3

Merge Sort The merge sort algorithm (introduced in Section 5.4) splits a list to be sorted with n items, where n is even, into two lists with $n/2$ elements each, and uses fewer than n comparisons to merge the two sorted lists of $n/2$ items each into one sorted list. Consequently, the number of comparisons used by the merge sort to sort a list of n elements is less than $M(n)$, where the function $M(n)$ satisfies the divide-and-conquer recurrence relation

$$M(n) = 2M(n/2) + n.$$

EXAMPLE 4

Fast Multiplication of Integers Surprisingly, there are more efficient algorithms than the conventional algorithm (described in Section 4.2) for multiplying integers. One of these algorithms, which uses a divide-and-conquer technique, will be described here. This fast multiplication algorithm proceeds by splitting each of two $2n$ -bit integers into two blocks, each with n bits. Then, the original multiplication is reduced from the multiplication of two $2n$ -bit integers to three multiplications of n -bit integers, plus shifts and additions.

Suppose that a and b are integers with binary expansions of length $2n$ (add initial bits of zero in these expansions if necessary to make them the same length). Let

$$a = (a_{2n-1}a_{2n-2}\cdots a_1a_0)_2 \quad \text{and} \quad b = (b_{2n-1}b_{2n-2}\cdots b_1b_0)_2.$$

Let

$$a = 2^n A_1 + A_0, \quad b = 2^n B_1 + B_0,$$

where

$$\begin{aligned} A_1 &= (a_{2n-1} \cdots a_{n+1} a_n)_2, & A_0 &= (a_{n-1} \cdots a_1 a_0)_2, \\ B_1 &= (b_{2n-1} \cdots b_{n+1} b_n)_2, & B_0 &= (b_{n-1} \cdots b_1 b_0)_2. \end{aligned}$$

The algorithm for fast multiplication of integers is based on the fact that ab can be rewritten as



$$ab = (2^{2n} + 2^n)A_1B_1 + 2^n(A_1 - A_0)(B_0 - B_1) + (2^n + 1)A_0B_0.$$

The important fact about this identity is that it shows that the multiplication of two $2n$ -bit integers can be carried out using three multiplications of n -bit integers, together with additions, subtractions, and shifts. This shows that if $f(n)$ is the total number of bit operations needed to multiply two n -bit integers, then

$$f(2n) = 3f(n) + Cn.$$

The reasoning behind this equation is as follows. The three multiplications of n -bit integers are carried out using $3f(n)$ -bit operations. Each of the additions, subtractions, and shifts uses a constant multiple of n -bit operations, and Cn represents the total number of bit operations used by these operations. 

EXAMPLE 5



Fast Matrix Multiplication In Example 7 of Section 3.3 we showed that multiplying two $n \times n$ matrices using the definition of matrix multiplication required n^3 multiplications and $n^2(n-1)$ additions. Consequently, computing the product of two $n \times n$ matrices in this way requires $O(n^3)$ operations (multiplications and additions). Surprisingly, there are more efficient divide-and-conquer algorithms for multiplying two $n \times n$ matrices. Such an algorithm, invented by Volker Strassen in 1969, reduces the multiplication of two $n \times n$ matrices, when n is even, to seven multiplications of two $(n/2) \times (n/2)$ matrices and 15 additions of $(n/2) \times (n/2)$ matrices. (See [CoLeRiSt09] for the details of this algorithm.) Hence, if $f(n)$ is the number of operations (multiplications and additions) used, it follows that

$$f(n) = 7f(n/2) + 15n^2/4$$

when n is even. 

As Examples 1–5 show, recurrence relations of the form $f(n) = af(n/b) + g(n)$ arise in many different situations. It is possible to derive estimates of the size of functions that satisfy such recurrence relations. Suppose that f satisfies this recurrence relation whenever n is divisible by b . Let $n = b^k$, where k is a positive integer. Then

$$\begin{aligned} f(n) &= af(n/b) + g(n) \\ &= a^2f(n/b^2) + ag(n/b) + g(n) \\ &= a^3f(n/b^3) + a^2g(n/b^2) + ag(n/b) + g(n) \\ &\vdots \\ &= a^kf(n/b^k) + \sum_{j=0}^{k-1} a^j g(n/b^j). \end{aligned}$$

Because $n/b^k = 1$, it follows that

$$f(n) = a^k f(1) + \sum_{j=0}^{k-1} a^j g(n/b^j).$$

We can use this equation for $f(n)$ to estimate the size of functions that satisfy divide-and-conquer relations.

THEOREM 1

Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + c$$

whenever n is divisible by b , where $a \geq 1$, b is an integer greater than 1, and c is a positive real number. Then

$$f(n) \text{ is } \begin{cases} O(n^{\log_b a}) & \text{if } a > 1, \\ O(\log n) & \text{if } a = 1. \end{cases}$$

Furthermore, when $n = b^k$ and $a \neq 1$, where k is a positive integer,

$$f(n) = C_1 n^{\log_b a} + C_2,$$

where $C_1 = f(1) + c/(a - 1)$ and $C_2 = -c/(a - 1)$.



Proof: First let $n = b^k$. From the expression for $f(n)$ obtained in the discussion preceding the theorem, with $g(n) = c$, we have

$$f(n) = a^k f(1) + \sum_{j=0}^{k-1} a^j c = a^k f(1) + c \sum_{j=0}^{k-1} a^j.$$

When $a = 1$ we have

$$f(n) = f(1) + ck.$$

Because $n = b^k$, we have $k = \log_b n$. Hence,

$$f(n) = f(1) + c \log_b n.$$

When n is not a power of b , we have $b^k < n < b^{k+1}$, for a positive integer k . Because f is increasing, it follows that $f(n) \leq f(b^{k+1}) = f(1) + c(k+1) = (f(1) + c) + ck \leq (f(1) + c) + c \log_b n$. Therefore, in both cases, $f(n)$ is $O(\log n)$ when $a = 1$.

Now suppose that $a > 1$. First assume that $n = b^k$, where k is a positive integer. From the formula for the sum of terms of a geometric progression (Theorem 1 in Section 2.4), it follows that

$$\begin{aligned} f(n) &= a^k f(1) + c(a^k - 1)/(a - 1) \\ &= a^k [f(1) + c/(a - 1)] - c/(a - 1) \\ &= C_1 n^{\log_b a} + C_2, \end{aligned}$$

because $a^k = a^{\log_b n} = n^{\log_b a}$ (see Exercise 4 in Appendix 2), where $C_1 = f(1) + c/(a - 1)$ and $C_2 = -c/(a - 1)$.

Now suppose that n is not a power of b . Then $b^k < n < b^{k+1}$, where k is a nonnegative integer. Because f is increasing,

$$\begin{aligned} f(n) &\leq f(b^{k+1}) = C_1 a^{k+1} + C_2 \\ &\leq (C_1 a) a^{\log_b n} + C_2 \\ &= (C_1 a) n^{\log_b a} + C_2, \end{aligned}$$

because $k \leq \log_b n < k + 1$.

Hence, we have $f(n)$ is $O(n^{\log_b a})$. 

Examples 6–9 illustrate how Theorem 1 is used.

EXAMPLE 6 Let $f(n) = 5f(n/2) + 3$ and $f(1) = 7$. Find $f(2^k)$, where k is a positive integer. Also, estimate $f(n)$ if f is an increasing function.



Solution: From the proof of Theorem 1, with $a = 5$, $b = 2$, and $c = 3$, we see that if $n = 2^k$, then

$$\begin{aligned} f(n) &= a^k[f(1) + c/(a - 1)] + [-c/(a - 1)] \\ &= 5^k[7 + (3/4)] - 3/4 \\ &= 5^k(31/4) - 3/4. \end{aligned}$$

Also, if $f(n)$ is increasing, Theorem 1 shows that $f(n)$ is $O(n^{\log_b a}) = O(n^{\log 5})$. 

We can use Theorem 1 to estimate the computational complexity of the binary search algorithm and the algorithm given in Example 2 for locating the minimum and maximum of a sequence.

EXAMPLE 7 Give a big- O estimate for the number of comparisons used by a binary search.

Solution: In Example 1 it was shown that $f(n) = f(n/2) + 2$ when n is even, where f is the number of comparisons required to perform a binary search on a sequence of size n . Hence, from Theorem 1, it follows that $f(n)$ is $O(\log n)$. 

EXAMPLE 8 Give a big- O estimate for the number of comparisons used to locate the maximum and minimum elements in a sequence using the algorithm given in Example 2.

Solution: In Example 2 we showed that $f(n) = 2f(n/2) + 2$, when n is even, where f is the number of comparisons needed by this algorithm. Hence, from Theorem 1, it follows that $f(n)$ is $O(n^{\log 2}) = O(n)$. 

We now state a more general, and more complicated, theorem, which has Theorem 1 as a special case. This theorem (or more powerful versions, including big-Theta estimates) is sometimes known as the master theorem because it is useful in analyzing the complexity of many important divide-and-conquer algorithms.

THEOREM 2

MASTER THEOREM Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + cn^d$$

whenever $n = b^k$, where k is a positive integer, $a \geq 1$, b is an integer greater than 1, and c and d are real numbers with c positive and d nonnegative. Then

$$f(n) \text{ is } \begin{cases} O(n^d) & \text{if } a < b^d, \\ O(n^d \log n) & \text{if } a = b^d, \\ O(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

The proof of Theorem 2 is left for the reader as Exercises 29–33.

EXAMPLE 9

Complexity of Merge Sort In Example 3 we explained that the number of comparisons used by the merge sort to sort a list of n elements is less than $M(n)$, where $M(n) = 2M(n/2) + n$. By the master theorem (Theorem 2) we find that $M(n)$ is $O(n \log n)$, which agrees with the estimate found in Section 5.4. 

EXAMPLE 10

Give a big- O estimate for the number of bit operations needed to multiply two n -bit integers using the fast multiplication algorithm described in Example 4.

Solution: Example 4 shows that $f(n) = 3f(n/2) + Cn$, when n is even, where $f(n)$ is the number of bit operations required to multiply two n -bit integers using the fast multiplication algorithm. Hence, from the master theorem (Theorem 2), it follows that $f(n)$ is $O(n^{\log 3})$. Note that $\log 3 \sim 1.6$. Because the conventional algorithm for multiplication uses $O(n^2)$ bit operations, the fast multiplication algorithm is a substantial improvement over the conventional algorithm in terms of time complexity for sufficiently large integers, including large integers that occur in practical applications. 

EXAMPLE 11

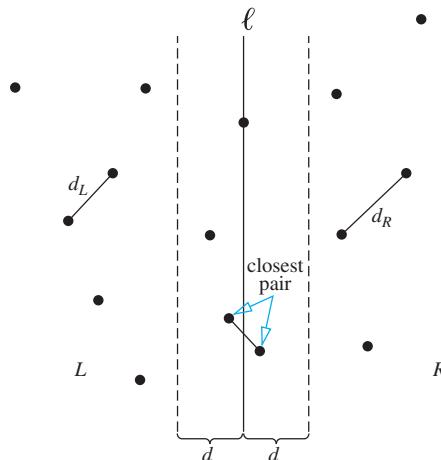
Give a big- O estimate for the number of multiplications and additions required to multiply two $n \times n$ matrices using the matrix multiplication algorithm referred to in Example 5.

Solution: Let $f(n)$ denote the number of additions and multiplications used by the algorithm mentioned in Example 5 to multiply two $n \times n$ matrices. We have $f(n) = 7f(n/2) + 15n^2/4$, when n is even. Hence, from the master theorem (Theorem 2), it follows that $f(n)$ is $O(n^{\log 7})$. Note that $\log 7 \sim 2.8$. Because the conventional algorithm for multiplying two $n \times n$ matrices uses $O(n^3)$ additions and multiplications, it follows that for sufficiently large integers n , including those that occur in many practical applications, this algorithm is substantially more efficient in time complexity than the conventional algorithm. 

THE CLOSEST-PAIR PROBLEM We conclude this section by introducing a divide-and-conquer algorithm from computational geometry, the part of discrete mathematics devoted to algorithms that solve geometric problems.

EXAMPLE 12

The Closest-Pair Problem Consider the problem of determining the closest pair of points in a set of n points $(x_1, y_1), \dots, (x_n, y_n)$ in the plane, where the distance between two points (x_i, y_i) and (x_j, y_j) is the usual Euclidean distance $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. This problem arises in many applications such as determining the closest pair of airplanes in the air space at a particular altitude being managed by an air traffic controller. How can this closest pair of points be found in an efficient way?



In this illustration the problem of finding the closest pair in a set of 16 points is reduced to two problems of finding the closest pair in a set of eight points *and* the problem of determining whether there are points closer than $d = \min(d_L, d_R)$ within the strip of width $2d$ centered at ℓ .

FIGURE 1 The Recursive Step of the Algorithm for Solving the Closest-Pair Problem.

It took researchers more than 10 years to find an algorithm with $O(n \log n)$ complexity that locates the closest pair of points among n points.

Solution: To solve this problem we can first determine the distance between every pair of points and then find the smallest of these distances. However, this approach requires $O(n^2)$ computations of distances and comparisons because there are $C(n, 2) = n(n - 1)/2$ pairs of points. Surprisingly, there is an elegant divide-and-conquer algorithm that can solve the closest-pair problem for n points using $O(n \log n)$ computations of distances and comparisons. The algorithm we describe here is due to Michael Samos (see [PrSa85]).

For simplicity, we assume that $n = 2^k$, where k is a positive integer. (We avoid some technical considerations that are needed when n is not a power of 2.) When $n = 2$, we have only one pair of points; the distance between these two points is the minimum distance. At the start of the algorithm we use the merge sort twice, once to sort the points in order of increasing x coordinates, and once to sort the points in order of increasing y coordinates. Each of these sorts requires $O(n \log n)$ operations. We will use these sorted lists in each recursive step.

The recursive part of the algorithm divides the problem into two subproblems, each involving half as many points. Using the sorted list of the points by their x coordinates, we construct a vertical line ℓ dividing the n points into two parts, a left part and a right part of equal size, each containing $n/2$ points, as shown in Figure 1. (If any points fall on the dividing line ℓ , we divide them among the two parts if necessary.) At subsequent steps of the recursion we need not sort on x coordinates again, because we can select the corresponding sorted subset of all the points. This selection is a task that can be done with $O(n)$ comparisons.

There are three possibilities concerning the positions of the closest points: (1) they are both in the left region L , (2) they are both in the right region R , or (3) one point is in the left region and the other is in the right region. Apply the algorithm recursively to compute d_L and d_R , where d_L is the minimum distance between points in the left region and d_R is the minimum distance between points in the right region. Let $d = \min(d_L, d_R)$. To successfully divide the problem of finding the closest two points in the original set into the two problems of finding the shortest distances between points in the two regions separately, we have to handle the conquer part of the algorithm, which requires that we consider the case where the closest points lie in different regions, that is, one point is in L and the other in R . Because there is a pair of points at distance d where both points lie in R or both points lie in L , for the closest points to lie in different regions requires that they must be a distance less than d apart.

For a point in the left region and a point in the right region to lie at a distance less than d apart, these points must lie in the vertical strip of width $2d$ that has the line ℓ as its center. (Otherwise, the distance between these points is greater than the difference in their x coordinates, which exceeds d .) To examine the points within this strip, we sort the points so that they are listed in order of increasing y coordinates, using the sorted list of the points by their y coordinates. At

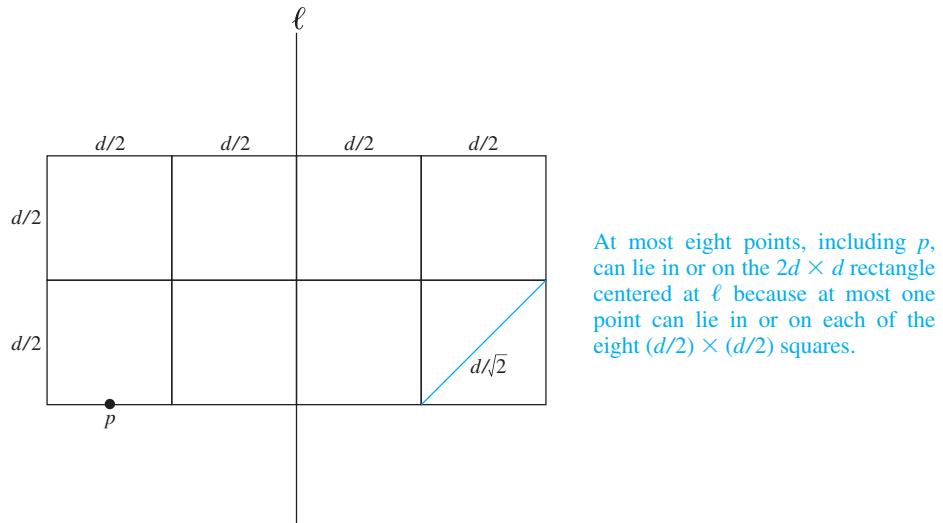


FIGURE 2 Showing That There Are at Most Seven Other Points to Consider for Each Point in the Strip.

each recursive step, we form a subset of the points in the region sorted by their y coordinates from the already sorted set of all points sorted by their y coordinates, which can be done with $O(n)$ comparisons.

Beginning with a point in the strip with the smallest y coordinate, we successively examine each point in the strip, computing the distance between this point and all other points in the strip that have larger y coordinates that could lie at a distance less than d from this point. Note that to examine a point p , we need only consider the distances between p and points in the set that lie within the rectangle of height d and width $2d$ with p on its base and with vertical sides at distance d from ℓ .

We can show that there are at most eight points from the set, including p , in or on this $2d \times d$ rectangle. To see this, note that there can be at most one point in each of the eight $d/2 \times d/2$ squares shown in Figure 2. This follows because the farthest apart points can be on or within one of these squares is the diagonal length $d/\sqrt{2}$ (which can be found using the Pythagorean theorem), which is less than d , and each of these $d/2 \times d/2$ squares lies entirely within the left region or the right region. This means that at this stage we need only compare at most seven distances, the distances between p and the seven or fewer other points in or on the rectangle, with d .

Because the total number of points in the strip of width $2d$ does not exceed n (the total number of points in the set), at most $7n$ distances need to be compared with d to find the minimum distance between points. That is, there are only $7n$ possible distances that could be less than d . Consequently, once the merge sort has been used to sort the pairs according to their x coordinates and according to their y coordinates, we find that the increasing function $f(n)$ satisfying the recurrence relation

$$f(n) = 2f(n/2) + 7n,$$

where $f(2) = 1$, exceeds the number of comparisons needed to solve the closest-pair problem for n points. By the master theorem (Theorem 2), it follows that $f(n)$ is $O(n \log n)$. The two sorts of points by their x coordinates and by their y coordinates each can be done using $O(n \log n)$ comparisons, by using the merge sort, and the sorted subsets of these coordinates at each of the $O(\log n)$ steps of the algorithm can be done using $O(n)$ comparisons each. Thus, we find that the closest-pair problem can be solved using $O(n \log n)$ comparisons. 

Exercises

1. How many comparisons are needed for a binary search in a set of 64 elements?
2. How many comparisons are needed to locate the maximum and minimum elements in a sequence with 128 elements using the algorithm in Example 2?
3. Multiply $(1110)_2$ and $(1010)_2$ using the fast multiplication algorithm.
4. Express the fast multiplication algorithm in pseudocode.
5. Determine a value for the constant C in Example 4 and use it to estimate the number of bit operations needed to multiply two 64-bit integers using the fast multiplication algorithm.
6. How many operations are needed to multiply two 32×32 matrices using the algorithm referred to in Example 5?
7. Suppose that $f(n) = f(n/3) + 1$ when n is a positive integer divisible by 3, and $f(1) = 1$. Find
 - a) $f(3)$.
 - b) $f(27)$.
 - c) $f(729)$.
8. Suppose that $f(n) = 2f(n/2) + 3$ when n is an even positive integer, and $f(1) = 5$. Find
 - a) $f(2)$.
 - b) $f(8)$.
 - c) $f(64)$.
 - d) $f(1024)$.
9. Suppose that $f(n) = f(n/5) + 3n^2$ when n is a positive integer divisible by 5, and $f(1) = 4$. Find
 - a) $f(5)$.
 - b) $f(125)$.
 - c) $f(3125)$.
10. Find $f(n)$ when $n = 2^k$, where f satisfies the recurrence relation $f(n) = f(n/2) + 1$ with $f(1) = 1$.
11. Give a big- O estimate for the function f in Exercise 10 if f is an increasing function.
12. Find $f(n)$ when $n = 3^k$, where f satisfies the recurrence relation $f(n) = 2f(n/3) + 4$ with $f(1) = 1$.
13. Give a big- O estimate for the function f in Exercise 12 if f is an increasing function.
14. Suppose that there are $n = 2^k$ teams in an elimination tournament, where there are $n/2$ games in the first round, with the $n/2 = 2^{k-1}$ winners playing in the second round, and so on. Develop a recurrence relation for the number of rounds in the tournament.
15. How many rounds are in the elimination tournament described in Exercise 14 when there are 32 teams?
16. Solve the recurrence relation for the number of rounds in the tournament described in Exercise 14.
17. Suppose that the votes of n people for different candidates (where there can be more than two candidates) for a particular office are the elements of a sequence. A person wins the election if this person receives a majority of the votes.
 - a) Devise a divide-and-conquer algorithm that determines whether a candidate received a majority and, if so, determine who this candidate is. [Hint: Assume that n is even and split the sequence of votes into two sequences, each with $n/2$ elements. Note that a candidate could not have received a majority of votes without receiving a majority of votes in at least one of the two halves.]
 - b) Use the master theorem to give a big- O estimate for the number of comparisons needed by the algorithm you devised in part (a).
18. Suppose that each person in a group of n people votes for exactly two people from a slate of candidates to fill two positions on a committee. The top two finishers both win positions as long as each receives more than $n/2$ votes.
 - a) Devise a divide-and-conquer algorithm that determines whether the two candidates who received the most votes each received at least $n/2$ votes and, if so, determine who these two candidates are.
 - b) Use the master theorem to give a big- O estimate for the number of comparisons needed by the algorithm you devised in part (a).
19. a) Set up a divide-and-conquer recurrence relation for the number of multiplications required to compute x^n , where x is a real number and n is a positive integer, using the recursive algorithm from Exercise 26 in Section 5.4.
 - b) Use the recurrence relation you found in part (a) to construct a big- O estimate for the number of multiplications used to compute x^n using the recursive algorithm.
20. a) Set up a divide-and-conquer recurrence relation for the number of modular multiplications required to compute $a^n \bmod m$, where a , m , and n are positive integers, using the recursive algorithms from Example 4 in Section 5.4.
 - b) Use the recurrence relation you found in part (a) to construct a big- O estimate for the number of modular multiplications used to compute $a^n \bmod m$ using the recursive algorithm.
21. Suppose that the function f satisfies the recurrence relation $f(n) = 2f(\sqrt{n}) + 1$ whenever n is a perfect square greater than 1 and $f(2) = 1$.
 - a) Find $f(16)$.
 - b) Give a big- O estimate for $f(n)$. [Hint: Make the substitution $m = \log n$.]
22. Suppose that the function f satisfies the recurrence relation $f(n) = 2f(\sqrt{n}) + \log n$ whenever n is a perfect square greater than 1 and $f(2) = 1$.
 - a) Find $f(16)$.
 - b) Find a big- O estimate for $f(n)$. [Hint: Make the substitution $m = \log n$.]
- **23. This exercise deals with the problem of finding the largest sum of consecutive terms of a sequence of n real numbers. When all terms are positive, the sum of all terms provides

the answer, but the situation is more complicated when some terms are negative. For example, the maximum sum of consecutive terms of the sequence $-2, 3, -1, 6, -7, 4$ is $3 + (-1) + 6 = 8$. (This exercise is based on [Be86].) Recall that in Exercise 56 in Section 8.1 we developed a dynamic programming algorithm for solving this problem. Here, we first look at the brute-force algorithm for solving this problem; then we develop a divide-and-conquer algorithm for solving it.

- a) Use pseudocode to describe an algorithm that solves this problem by finding the sums of consecutive terms starting with the first term, the sums of consecutive terms starting with the second term, and so on, keeping track of the maximum sum found so far as the algorithm proceeds.
 - b) Determine the computational complexity of the algorithm in part (a) in terms of the number of sums computed and the number of comparisons made.
 - c) Devise a divide-and-conquer algorithm to solve this problem. [Hint: Assume that there are an even number of terms in the sequence and split the sequence into two halves. Explain how to handle the case when the maximum sum of consecutive terms includes terms in both halves.]
 - d) Use the algorithm from part (c) to find the maximum sum of consecutive terms of each of the sequences: $-2, 4, -1, 3, 5, -6, 1, 2; 4, 1, -3, 7, -1, -5, 3, -2;$ and $-1, 6, 3, -4, -5, 8, -1, 7$.
 - e) Find a recurrence relation for the number of sums and comparisons used by the divide-and-conquer algorithm from part (c).
 - f) Use the master theorem to estimate the computational complexity of the divide-and-conquer algorithm. How does it compare in terms of computational complexity with the algorithm from part (a)?
24. Apply the algorithm described in Example 12 for finding the closest pair of points, using the Euclidean distance between points, to find the closest pair of the points $(1, 3), (1, 7), (2, 4), (2, 9), (3, 1), (3, 5), (4, 3)$, and $(4, 7)$.
25. Apply the algorithm described in Example 12 for finding the closest pair of points, using the Euclidean distance between points, to find the closest pair of the points $(1, 2), (1, 6), (2, 4), (2, 8), (3, 1), (3, 6), (3, 10), (4, 3), (5, 1), (5, 5), (5, 9), (6, 7), (7, 1), (7, 4), (7, 9)$, and $(8, 6)$.
- *26. Use pseudocode to describe the recursive algorithm for solving the closest-pair problem as described in Example 12.
27. Construct a variation of the algorithm described in Example 12 along with justifications of the steps used by the algorithm to find the smallest distance between two points if the distance between two points is defined to be $d((x_i, y_i), (x_j, y_j)) = \max(|x_i - x_j|, |y_i - y_j|)$.
- *28. Suppose someone picks a number x from a set of n numbers. A second person tries to guess the number by successively selecting subsets of the n numbers and

asking the first person whether x is in each set. The first person answers either “yes” or “no.” When the first person answers each query truthfully, we can find x using $\log n$ queries by successively splitting the sets used in each query in half. Ulam’s problem, proposed by Stanislaw Ulam in 1976, asks for the number of queries required to find x , supposing that the first person is allowed to lie exactly once.

- a) Show that by asking each question twice, given a number x and a set with n elements, and asking one more question when we find the lie, Ulam’s problem can be solved using $2 \log n + 1$ queries.
- b) Show that by dividing the initial set of n elements into four parts, each with $n/4$ elements, $1/4$ of the elements can be eliminated using two queries. [Hint: Use two queries, where each of the queries asks whether the element is in the union of two of the subsets with $n/4$ elements and where one of the subsets of $n/4$ elements is used in both queries.]
- c) Show from part (b) that if $f(n)$ equals the number of queries used to solve Ulam’s problem using the method from part (b) and n is divisible by 4, then $f(n) = f(3n/4) + 2$.
- d) Solve the recurrence relation in part (c) for $f(n)$.
- e) Is the naive way to solve Ulam’s problem by asking each question twice or the divide-and-conquer method based on part (b) more efficient? The most efficient way to solve Ulam’s problem has been determined by A. Pelc [Pe87].

In Exercises 29–33, assume that f is an increasing function satisfying the recurrence relation $f(n) = af(n/b) + cn^d$, where $a \geq 1$, b is an integer greater than 1, and c and d are positive real numbers. These exercises supply a proof of Theorem 2.

- *29. Show that if $a = b^d$ and n is a power of b , then $f(n) = f(1)n^d + cn^d \log_b n$.
- 30. Use Exercise 29 to show that if $a = b^d$, then $f(n)$ is $O(n^d \log n)$.
- *31. Show that if $a \neq b^d$ and n is a power of b , then $f(n) = C_1 n^d + C_2 n^{\log_b a}$, where $C_1 = b^d c / (b^d - a)$ and $C_2 = f(1) + b^d c / (a - b^d)$.
- 32. Use Exercise 31 to show that if $a < b^d$, then $f(n)$ is $O(n^d)$.
- 33. Use Exercise 31 to show that if $a > b^d$, then $f(n)$ is $O(n^{\log_b a})$.
- 34. Find $f(n)$ when $n = 4^k$, where f satisfies the recurrence relation $f(n) = 5f(n/4) + 6n$, with $f(1) = 1$.
- 35. Give a big- O estimate for the function f in Exercise 34 if f is an increasing function.
- 36. Find $f(n)$ when $n = 2^k$, where f satisfies the recurrence relation $f(n) = 8f(n/2) + n^2$ with $f(1) = 1$.
- 37. Give a big- O estimate for the function f in Exercise 36 if f is an increasing function.

8.4 Generating Functions

Introduction



Generating functions are used to represent sequences efficiently by coding the terms of a sequence as coefficients of powers of a variable x in a formal power series. Generating functions can be used to solve many types of counting problems, such as the number of ways to select or distribute objects of different kinds, subject to a variety of constraints, and the number of ways to make change for a dollar using coins of different denominations. Generating functions can be used to solve recurrence relations by translating a recurrence relation for the terms of a sequence into an equation involving a generating function. This equation can then be solved to find a closed form for the generating function. From this closed form, the coefficients of the power series for the generating function can be found, solving the original recurrence relation. Generating functions can also be used to prove combinatorial identities by taking advantage of relatively simple relationships between functions that can be translated into identities involving the terms of sequences. Generating functions are a helpful tool for studying many properties of sequences besides those described in this section, such as their use for establishing asymptotic formulae for the terms of a sequence.

We begin with the definition of the generating function for a sequence.

DEFINITION 1

The *generating function* for the sequence $a_0, a_1, \dots, a_k, \dots$ of real numbers is the infinite series

$$G(x) = a_0 + a_1x + \dots + a_kx^k + \dots = \sum_{k=0}^{\infty} a_kx^k.$$

Remark: The generating function for $\{a_k\}$ given in Definition 1 is sometimes called the **ordinary generating function** of $\{a_k\}$ to distinguish it from other types of generating functions for this sequence.

EXAMPLE 1



The generating functions for the sequences $\{a_k\}$ with $a_k = 3$, $a_k = k + 1$, and $a_k = 2^k$ are $\sum_{k=0}^{\infty} 3x^k$, $\sum_{k=0}^{\infty} (k + 1)x^k$, and $\sum_{k=0}^{\infty} 2^k x^k$, respectively.

We can define generating functions for finite sequences of real numbers by extending a finite sequence a_0, a_1, \dots, a_n into an infinite sequence by setting $a_{n+1} = 0, a_{n+2} = 0$, and so on. The generating function $G(x)$ of this infinite sequence $\{a_n\}$ is a polynomial of degree n because no terms of the form a_jx^j with $j > n$ occur, that is,

$$G(x) = a_0 + a_1x + \dots + a_nx^n.$$

EXAMPLE 2

What is the generating function for the sequence 1, 1, 1, 1, 1, 1?

Solution: The generating function of 1, 1, 1, 1, 1, 1 is

$$1 + x + x^2 + x^3 + x^4 + x^5.$$

By Theorem 1 of Section 2.4 we have

$$(x^6 - 1)/(x - 1) = 1 + x + x^2 + x^3 + x^4 + x^5$$

when $x \neq 1$. Consequently, $G(x) = (x^6 - 1)/(x - 1)$ is the generating function of the sequence 1, 1, 1, 1, 1, 1. [Because the powers of x are only place holders for the terms of the sequence in a generating function, we do not need to worry that $G(1)$ is undefined.] 

EXAMPLE 3 Let m be a positive integer. Let $a_k = C(m, k)$, for $k = 0, 1, 2, \dots, m$. What is the generating function for the sequence a_0, a_1, \dots, a_m ?

Solution: The generating function for this sequence is

$$G(x) = C(m, 0) + C(m, 1)x + C(m, 2)x^2 + \cdots + C(m, m)x^m.$$

The binomial theorem shows that $G(x) = (1 + x)^m$. 

Useful Facts About Power Series

When generating functions are used to solve counting problems, they are usually considered to be **formal power series**. Questions about the convergence of these series are ignored. However, to apply some results from calculus, it is sometimes important to consider for which x the power series converges. The fact that a function has a unique power series around $x = 0$ will also be important. Generally, however, we will not be concerned with questions of convergence or the uniqueness of power series in our discussions. Readers familiar with calculus can consult textbooks on this subject for details about power series, including the convergence of the series we consider here.

We will now state some important facts about infinite series used when working with generating functions. A discussion of these and related results can be found in calculus texts.

EXAMPLE 4 The function $f(x) = 1/(1 - x)$ is the generating function of the sequence 1, 1, 1, 1, ..., because

$$1/(1 - x) = 1 + x + x^2 + \cdots$$

for $|x| < 1$. 

EXAMPLE 5 The function $f(x) = 1/(1 - ax)$ is the generating function of the sequence 1, a, a^2, a^3, \dots , because

$$1/(1 - ax) = 1 + ax + a^2x^2 + \cdots$$

when $|ax| < 1$, or equivalently, for $|x| < 1/|a|$ for $a \neq 0$. 

We also will need some results on how to add and how to multiply two generating functions. Proofs of these results can be found in calculus texts.

THEOREM 1 Let $f(x) = \sum_{k=0}^{\infty} a_k x^k$ and $g(x) = \sum_{k=0}^{\infty} b_k x^k$. Then

$$f(x) + g(x) = \sum_{k=0}^{\infty} (a_k + b_k) x^k \quad \text{and} \quad f(x)g(x) = \sum_{k=0}^{\infty} \left(\sum_{j=0}^k a_j b_{k-j} \right) x^k.$$

Remark: Theorem 1 is valid only for power series that converge in an interval, as all series considered in this section do. However, the theory of generating functions is not limited to such series. In the case of series that do not converge, the statements in Theorem 1 can be taken as definitions of addition and multiplication of generating functions.

We will illustrate how Theorem 1 can be used with Example 6.

EXAMPLE 6 Let $f(x) = 1/(1-x)^2$. Use Example 4 to find the coefficients a_0, a_1, a_2, \dots in the expansion $f(x) = \sum_{k=0}^{\infty} a_k x^k$.

Solution: From Example 4 we see that

$$1/(1-x) = 1 + x + x^2 + x^3 + \dots$$

Hence, from Theorem 1, we have

$$1/(1-x)^2 = \sum_{k=0}^{\infty} \left(\sum_{j=0}^k 1 \right) x^k = \sum_{k=0}^{\infty} (k+1)x^k.$$


Remark: This result also can be derived from Example 4 by differentiation. Taking derivatives is a useful technique for producing new identities from existing identities for generating functions.

To use generating functions to solve many important counting problems, we will need to apply the binomial theorem for exponents that are not positive integers. Before we state an extended version of the binomial theorem, we need to define extended binomial coefficients.

DEFINITION 2

Let u be a real number and k a nonnegative integer. Then the *extended binomial coefficient* $\binom{u}{k}$ is defined by

$$\binom{u}{k} = \begin{cases} u(u-1)\cdots(u-k+1)/k! & \text{if } k > 0, \\ 1 & \text{if } k = 0. \end{cases}$$

EXAMPLE 7 Find the values of the extended binomial coefficients $\binom{-2}{3}$ and $\binom{1/2}{3}$.

Solution: Taking $u = -2$ and $k = 3$ in Definition 2 gives us

$$\binom{-2}{3} = \frac{(-2)(-3)(-4)}{3!} = -4.$$

Similarly, taking $u = 1/2$ and $k = 3$ gives us

$$\begin{aligned} \binom{1/2}{3} &= \frac{(1/2)(1/2-1)(1/2-2)}{3!} \\ &= (1/2)(-1/2)(-3/2)/6 \\ &= 1/16. \end{aligned}$$


Example 8 provides a useful formula for extended binomial coefficients when the top parameter is a negative integer. It will be useful in our subsequent discussions.

EXAMPLE 8 When the top parameter is a negative integer, the extended binomial coefficient can be expressed in terms of an ordinary binomial coefficient. To see that this is the case, note that

$$\begin{aligned}
 \binom{-n}{r} &= \frac{(-n)(-n-1)\cdots(-n-r+1)}{r!} && \text{by definition of extended binomial coefficient} \\
 &= \frac{(-1)^r n(n+1)\cdots(n+r-1)}{r!} && \text{factoring out } -1 \text{ from each term in the numerator} \\
 &= \frac{(-1)^r (n+r-1)(n+r-2)\cdots n}{r!} && \text{by the commutative law for multiplication} \\
 &= \frac{(-1)^r (n+r-1)!}{r!(n-1)!} && \text{multiplying both the numerator and denominator by } (n-1)! \\
 &= (-1)^r \binom{n+r-1}{r} && \text{by the definition of binomial coefficients} \\
 &= (-1)^r C(n+r-1, r). && \text{using alternative notation for binomial coefficients}
 \end{aligned}$$

We now state the extended binomial theorem.

THEOREM 2

THE EXTENDED BINOMIAL THEOREM Let x be a real number with $|x| < 1$ and let u be a real number. Then

$$(1+x)^u = \sum_{k=0}^{\infty} \binom{u}{k} x^k.$$

Theorem 2 can be proved using the theory of Maclaurin series. We leave its proof to the reader with a familiarity with this part of calculus.

Remark: When u is a positive integer, the extended binomial theorem reduces to the binomial theorem presented in Section 6.4, because in that case $\binom{u}{k} = 0$ if $k > u$.

Example 9 illustrates the use of Theorem 2 when the exponent is a negative integer.

EXAMPLE 9 Find the generating functions for $(1+x)^{-n}$ and $(1-x)^{-n}$, where n is a positive integer, using the extended binomial theorem.

Solution: By the extended binomial theorem, it follows that

$$(1+x)^{-n} = \sum_{k=0}^{\infty} \binom{-n}{k} x^k.$$

Using Example 8, which provides a simple formula for $\binom{-n}{k}$, we obtain

$$(1+x)^{-n} = \sum_{k=0}^{\infty} (-1)^k C(n+k-1, k) x^k.$$

Replacing x by $-x$, we find that

$$(1-x)^{-n} = \sum_{k=0}^{\infty} C(n+k-1, k) x^k.$$



Table 1 presents a useful summary of some generating functions that arise frequently.

Remark: Note that the second and third formulae in this table can be deduced from the first formula by substituting ax and x^r for x , respectively. Similarly, the sixth and seventh formulae can be deduced from the fifth formula using the same substitutions. The tenth and eleventh can be deduced from the ninth formula by substituting $-x$ and ax for x , respectively. Also, some of the formulae in this table can be derived from other formulae using methods from calculus (such as differentiation and integration). Students are encouraged to know the core formulae in this table (that is, formulae from which the others can be derived, perhaps the first, fourth, fifth, eighth, ninth, twelfth, and thirteenth formulae) and understand how to derive the other formulae from these core formulae.

Counting Problems and Generating Functions

Generating functions can be used to solve a wide variety of counting problems. In particular, they can be used to count the number of combinations of various types. In Chapter 6 we developed techniques to count the r -combinations from a set with n elements when repetition is allowed and additional constraints may exist. Such problems are equivalent to counting the solutions to equations of the form

$$e_1 + e_2 + \cdots + e_n = C,$$

where C is a constant and each e_i is a nonnegative integer that may be subject to a specified constraint. Generating functions can also be used to solve counting problems of this type, as Examples 10–12 show.

EXAMPLE 10 Find the number of solutions of

$$e_1 + e_2 + e_3 = 17,$$

where e_1 , e_2 , and e_3 are nonnegative integers with $2 \leq e_1 \leq 5$, $3 \leq e_2 \leq 6$, and $4 \leq e_3 \leq 7$.

Solution: The number of solutions with the indicated constraints is the coefficient of x^{17} in the expansion of

$$(x^2 + x^3 + x^4 + x^5)(x^3 + x^4 + x^5 + x^6)(x^4 + x^5 + x^6 + x^7).$$

TABLE 1 Useful Generating Functions.

$G(x)$	a_k
$(1+x)^n = \sum_{k=0}^n C(n, k)x^k \\ = 1 + C(n, 1)x + C(n, 2)x^2 + \cdots + x^n$	$C(n, k)$
$(1+ax)^n = \sum_{k=0}^n C(n, k)a^kx^k \\ = 1 + C(n, 1)ax + C(n, 2)a^2x^2 + \cdots + a^nx^n$	$C(n, k)a^k$
$(1+x^r)^n = \sum_{k=0}^n C(n, k)x^{rk} \\ = 1 + C(n, 1)x^r + C(n, 2)x^{2r} + \cdots + x^{rn}$	$C(n, k/r)$ if $r \mid k$; 0 otherwise
$\frac{1-x^{n+1}}{1-x} = \sum_{k=0}^n x^k = 1 + x + x^2 + \cdots + x^n$	1 if $k \leq n$; 0 otherwise
$\frac{1}{1-x} = \sum_{k=0}^{\infty} x^k = 1 + x + x^2 + \cdots$	1
$\frac{1}{1-ax} = \sum_{k=0}^{\infty} a^kx^k = 1 + ax + a^2x^2 + \cdots$	a^k
$\frac{1}{1-x^r} = \sum_{k=0}^{\infty} x^{rk} = 1 + x^r + x^{2r} + \cdots$	1 if $r \mid k$; 0 otherwise
$\frac{1}{(1-x)^2} = \sum_{k=0}^{\infty} (k+1)x^k = 1 + 2x + 3x^2 + \cdots$	$k+1$
$\frac{1}{(1-x)^n} = \sum_{k=0}^{\infty} C(n+k-1, k)x^k \\ = 1 + C(n, 1)x + C(n+1, 2)x^2 + \cdots$	$C(n+k-1, k) = C(n+k-1, n-1)$
$\frac{1}{(1+x)^n} = \sum_{k=0}^{\infty} C(n+k-1, k)(-1)^kx^k \\ = 1 - C(n, 1)x + C(n+1, 2)x^2 - \cdots$	$(-1)^k C(n+k-1, k) = (-1)^k C(n+k-1, n-1)$
$\frac{1}{(1-ax)^n} = \sum_{k=0}^{\infty} C(n+k-1, k)a^kx^k \\ = 1 + C(n, 1)ax + C(n+1, 2)a^2x^2 + \cdots$	$C(n+k-1, k)a^k = C(n+k-1, n-1)a^k$
$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$	$1/k!$
$\ln(1+x) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k}x^k = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots$	$(-1)^{k+1}/k$

Note: The series for the last two generating functions can be found in most calculus books when power series are discussed.

This follows because we obtain a term equal to x^{17} in the product by picking a term in the first sum x^{e_1} , a term in the second sum x^{e_2} , and a term in the third sum x^{e_3} , where the exponents e_1 , e_2 , and e_3 satisfy the equation $e_1 + e_2 + e_3 = 17$ and the given constraints.

It is not hard to see that the coefficient of x^{17} in this product is 3. Hence, there are three solutions. (Note that the calculating of this coefficient involves about as much work as enumerating all the solutions of the equation with the given constraints. However, the method that this illustrates often can be used to solve wide classes of counting problems with special formulae, as we will see. Furthermore, a computer algebra system can be used to do such computations.) 

EXAMPLE 11 In how many different ways can eight identical cookies be distributed among three distinct children if each child receives at least two cookies and no more than four cookies?

Solution: Because each child receives at least two but no more than four cookies, for each child there is a factor equal to

$$(x^2 + x^3 + x^4)$$

in the generating function for the sequence $\{c_n\}$, where c_n is the number of ways to distribute n cookies. Because there are three children, this generating function is

$$(x^2 + x^3 + x^4)^3.$$

We need the coefficient of x^8 in this product. The reason is that the x^8 terms in the expansion correspond to the ways that three terms can be selected, with one from each factor, that have exponents adding up to 8. Furthermore, the exponents of the term from the first, second, and third factors are the numbers of cookies the first, second, and third children receive, respectively. Computation shows that this coefficient equals 6. Hence, there are six ways to distribute the cookies so that each child receives at least two, but no more than four, cookies. 

EXAMPLE 12 Use generating functions to determine the number of ways to insert tokens worth \$1, \$2, and \$5 into a vending machine to pay for an item that costs r dollars in both the cases when the order in which the tokens are inserted does not matter and when the order does matter. (For example, there are two ways to pay for an item that costs \$3 when the order in which the tokens are inserted does not matter: inserting three \$1 tokens or one \$1 token and a \$2 token. When the order matters, there are three ways: inserting three \$1 tokens, inserting a \$1 token and then a \$2 token, or inserting a \$2 token and then a \$1 token.)

Solution: Consider the case when the order in which the tokens are inserted does not matter. Here, all we care about is the number of each token used to produce a total of r dollars. Because we can use any number of \$1 tokens, any number of \$2 tokens, and any number of \$5 tokens, the answer is the coefficient of x^r in the generating function

$$(1 + x + x^2 + x^3 + \dots)(1 + x^2 + x^4 + x^6 + \dots)(1 + x^5 + x^{10} + x^{15} + \dots).$$

(The first factor in this product represents the \$1 tokens used, the second the \$2 tokens used, and the third the \$5 tokens used.) For example, the number of ways to pay for an item costing \$7 using \$1, \$2, and \$5 tokens is given by the coefficient of x^7 in this expansion, which equals 6.

When the order in which the tokens are inserted matters, the number of ways to insert exactly n tokens to produce a total of r dollars is the coefficient of x^r in

$$(x + x^2 + x^5)^n,$$

because each of the r tokens may be a \$1 token, a \$2 token, or a \$5 token. Because any number of tokens may be inserted, the number of ways to produce r dollars using \$1, \$2, or \$5 tokens, when the order in which the tokens are inserted matters, is the coefficient of x^r in

$$\begin{aligned} 1 + (x + x^2 + x^5) + (x + x^2 + x^5)^2 + \cdots &= \frac{1}{1 - (x + x^2 + x^5)} \\ &= \frac{1}{1 - x - x^2 - x^5}, \end{aligned}$$

where we have added the number of ways to insert 0 tokens, 1 token, 2 tokens, 3 tokens, and so on, and where we have used the identity $1/(1-x) = 1 + x + x^2 + \cdots$ with x replaced with $x + x^2 + x^5$. For example, the number of ways to pay for an item costing \$7 using \$1, \$2, and \$5 tokens, when the order in which the tokens are used matters, is the coefficient of x^7 in this expansion, which equals 26. [Hint: To see that this coefficient equals 26 requires the addition of the coefficients of x^7 in the expansions $(x + x^2 + x^5)^k$ for $2 \leq k \leq 7$. This can be done by hand with considerable computation, or a computer algebra system can be used.] 

Example 13 shows the versatility of generating functions when used to solve problems with differing assumptions.

EXAMPLE 13 Use generating functions to find the number of k -combinations of a set with n elements. Assume that the binomial theorem has already been established.

Solution: Each of the n elements in the set contributes the term $(1+x)$ to the generating function $f(x) = \sum_{k=0}^n a_k x^k$. Here $f(x)$ is the generating function for $\{a_k\}$, where a_k represents the number of k -combinations of a set with n elements. Hence,

$$f(x) = (1+x)^n.$$

But by the binomial theorem, we have

$$f(x) = \sum_{k=0}^n \binom{n}{k} x^k,$$

where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

Hence, $C(n, k)$, the number of k -combinations of a set with n elements, is

$$\frac{n!}{k!(n-k)!}.$$


Remark: We proved the binomial theorem in Section 6.4 using the formula for the number of r -combinations of a set with n elements. This example shows that the binomial theorem, which can be proved by mathematical induction, can be used to derive the formula for the number of r -combinations of a set with n elements.

EXAMPLE 14 Use generating functions to find the number of r -combinations from a set with n elements when repetition of elements is allowed.

Solution: Let $G(x)$ be the generating function for the sequence $\{a_r\}$, where a_r equals the number of r -combinations of a set with n elements with repetitions allowed. That is, $G(x) = \sum_{r=0}^{\infty} a_r x^r$. Because we can select any number of a particular member of the set with n elements when we form an r -combination with repetition allowed, each of the n elements contributes $(1 + x + x^2 + x^3 + \dots)$ to a product expansion for $G(x)$. Each element contributes this factor because it may be selected zero times, one time, two times, three times, and so on, when an r -combination is formed (with a total of r elements selected). Because there are n elements in the set and each contributes this same factor to $G(x)$, we have

$$G(x) = (1 + x + x^2 + \dots)^n.$$

As long as $|x| < 1$, we have $1 + x + x^2 + \dots = 1/(1 - x)$, so

$$G(x) = 1/(1 - x)^n = (1 - x)^{-n}.$$

Applying the extended binomial theorem (Theorem 2), it follows that

$$(1 - x)^{-n} = (1 + (-x))^{-n} = \sum_{r=0}^{\infty} \binom{-n}{r} (-x)^r.$$

The number of r -combinations of a set with n elements with repetitions allowed, when r is a positive integer, is the coefficient a_r of x^r in this sum. Consequently, using Example 8 we find that a_r equals

$$\begin{aligned} \binom{-n}{r} (-1)^r &= (-1)^r C(n + r - 1, r) \cdot (-1)^r \\ &= C(n + r - 1, r). \end{aligned}$$



Note that the result in Example 14 is the same result we stated as Theorem 2 in Section 6.5.

EXAMPLE 15 Use generating functions to find the number of ways to select r objects of n different kinds if we must select at least one object of each kind.

Solution: Because we need to select at least one object of each kind, each of the n kinds of objects contributes the factor $(x + x^2 + x^3 + \dots)$ to the generating function $G(x)$ for the sequence $\{a_r\}$, where a_r is the number of ways to select r objects of n different kinds if we need at least one object of each kind. Hence,

$$G(x) = (x + x^2 + x^3 + \dots)^n = x^n(1 + x + x^2 + \dots)^n = x^n/(1 - x)^n.$$

Using the extended binomial theorem and Example 8, we have

$$\begin{aligned}
 G(x) &= x^n / (1 - x)^n \\
 &= x^n \cdot (1 - x)^{-n} \\
 &= x^n \sum_{r=0}^{\infty} \binom{-n}{r} (-x)^r \\
 &= x^n \sum_{r=0}^{\infty} (-1)^r C(n+r-1, r) (-1)^r x^r \\
 &= \sum_{r=0}^{\infty} C(n+r-1, r) x^{n+r} \\
 &= \sum_{t=n}^{\infty} C(t-1, t-n) x^t \\
 &= \sum_{r=n}^{\infty} C(r-1, r-n) x^r.
 \end{aligned}$$

We have shifted the summation in the next-to-last equality by setting $t = n + r$ so that $t = n$ when $r = 0$ and $n + r - 1 = t - 1$, and then we replaced t by r as the index of summation in the last equality to return to our original notation. Hence, there are $C(r-1, r-n)$ ways to select r objects of n different kinds if we must select at least one object of each kind. \blacktriangleleft

Using Generating Functions to Solve Recurrence Relations

We can find the solution to a recurrence relation and its initial conditions by finding an explicit formula for the associated generating function. This is illustrated in Examples 16 and 17.

EXAMPLE 16 Solve the recurrence relation $a_k = 3a_{k-1}$ for $k = 1, 2, 3, \dots$ and initial condition $a_0 = 2$.



Solution: Let $G(x)$ be the generating function for the sequence $\{a_k\}$, that is, $G(x) = \sum_{k=0}^{\infty} a_k x^k$. First note that

$$xG(x) = \sum_{k=0}^{\infty} a_k x^{k+1} = \sum_{k=1}^{\infty} a_{k-1} x^k.$$

Using the recurrence relation, we see that

$$\begin{aligned}
 G(x) - 3xG(x) &= \sum_{k=0}^{\infty} a_k x^k - 3 \sum_{k=1}^{\infty} a_{k-1} x^k \\
 &= a_0 + \sum_{k=1}^{\infty} (a_k - 3a_{k-1}) x^k \\
 &= 2,
 \end{aligned}$$

because $a_0 = 2$ and $a_k = 3a_{k-1}$. Thus,

$$G(x) - 3xG(x) = (1 - 3x)G(x) = 2.$$

Solving for $G(x)$ shows that $G(x) = 2/(1 - 3x)$. Using the identity $1/(1 - ax) = \sum_{k=0}^{\infty} a^k x^k$, from Table 1, we have

$$G(x) = 2 \sum_{k=0}^{\infty} 3^k x^k = \sum_{k=0}^{\infty} 2 \cdot 3^k x^k.$$

Consequently, $a_k = 2 \cdot 3^k$. ◀

EXAMPLE 17 Suppose that a valid codeword is an n -digit number in decimal notation containing an even number of 0s. Let a_n denote the number of valid codewords of length n . In Example 4 of Section 8.1 we showed that the sequence $\{a_n\}$ satisfies the recurrence relation

$$a_n = 8a_{n-1} + 10^{n-1}$$

and the initial condition $a_1 = 9$. Use generating functions to find an explicit formula for a_n .

Solution: To make our work with generating functions simpler, we extend this sequence by setting $a_0 = 1$; when we assign this value to a_0 and use the recurrence relation, we have $a_1 = 8a_0 + 10^0 = 8 + 1 = 9$, which is consistent with our original initial condition. (It also makes sense because there is one code word of length 0—the empty string.)

We multiply both sides of the recurrence relation by x^n to obtain

$$a_n x^n = 8a_{n-1} x^n + 10^{n-1} x^n.$$

Let $G(x) = \sum_{n=0}^{\infty} a_n x^n$ be the generating function of the sequence a_0, a_1, a_2, \dots . We sum both sides of the last equation starting with $n = 1$, to find that

$$\begin{aligned} G(x) - 1 &= \sum_{n=1}^{\infty} a_n x^n = \sum_{n=1}^{\infty} (8a_{n-1} x^n + 10^{n-1} x^n) \\ &= 8 \sum_{n=1}^{\infty} a_{n-1} x^n + \sum_{n=1}^{\infty} 10^{n-1} x^n \\ &= 8x \sum_{n=1}^{\infty} a_{n-1} x^{n-1} + x \sum_{n=1}^{\infty} 10^{n-1} x^{n-1} \\ &= 8x \sum_{n=0}^{\infty} a_n x^n + x \sum_{n=0}^{\infty} 10^n x^n \\ &= 8xG(x) + x/(1 - 10x), \end{aligned}$$

where we have used Example 5 to evaluate the second summation. Therefore, we have

$$G(x) - 1 = 8xG(x) + x/(1 - 10x).$$

Solving for $G(x)$ shows that

$$G(x) = \frac{1 - 9x}{(1 - 8x)(1 - 10x)}.$$

Expanding the right-hand side of this equation into partial fractions (as is done in the integration of rational functions studied in calculus) gives

$$G(x) = \frac{1}{2} \left(\frac{1}{1-8x} + \frac{1}{1-10x} \right).$$

Using Example 5 twice (once with $a = 8$ and once with $a = 10$) gives

$$\begin{aligned} G(x) &= \frac{1}{2} \left(\sum_{n=0}^{\infty} 8^n x^n + \sum_{n=0}^{\infty} 10^n x^n \right) \\ &= \sum_{n=0}^{\infty} \frac{1}{2} (8^n + 10^n) x^n. \end{aligned}$$

Consequently, we have shown that

$$a_n = \frac{1}{2} (8^n + 10^n).$$



Proving Identities via Generating Functions

In Chapter 6 we saw how combinatorial identities could be established using combinatorial proofs. Here we will show that such identities, as well as identities for extended binomial coefficients, can be proved using generating functions. Sometimes the generating function approach is simpler than other approaches, especially when it is simpler to work with the closed form of a generating function than with the terms of the sequence themselves. We illustrate how generating functions can be used to prove identities with Example 18.

EXAMPLE 18 Use generating functions to show that

$$\sum_{k=0}^n C(n, k)^2 = C(2n, n)$$

whenever n is a positive integer.

Solution: First note that by the binomial theorem $C(2n, n)$ is the coefficient of x^n in $(1+x)^{2n}$. However, we also have

$$\begin{aligned} (1+x)^{2n} &= [(1+x)^n]^2 \\ &= [C(n, 0) + C(n, 1)x + C(n, 2)x^2 + \cdots + C(n, n)x^n]^2. \end{aligned}$$

The coefficient of x^n in this expression is

$$C(n, 0)C(n, n) + C(n, 1)C(n, n-1) + C(n, 2)C(n, n-2) + \cdots + C(n, n)C(n, 0).$$

This equals $\sum_{k=0}^n C(n, k)^2$, because $C(n, n-k) = C(n, k)$. Because both $C(2n, n)$ and $\sum_{k=0}^n C(n, k)^2$ represent the coefficient of x^n in $(1+x)^{2n}$, they must be equal.



Exercises 42 and 43 ask that Pascal's identity and Vandermonde's identity be proved using generating functions.

Exercises

1. Find the generating function for the finite sequence 2, 2, 2, 2, 2.
2. Find the generating function for the finite sequence 1, 4, 16, 64, 256.

In Exercises 3–8, by a **closed form** we mean an algebraic expression not involving a summation over a range of values or the use of ellipses.

3. Find a closed form for the generating function for each of these sequences. (For each sequence, use the most obvious choice of a sequence that follows the pattern of the initial terms listed.)

- a) 0, 2, 2, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, ...
- b) 0, 0, 0, 1, 1, 1, 1, 1, 1, ...
- c) 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, ...
- d) 2, 4, 8, 16, 32, 64, 128, 256, ...
- e) $\binom{7}{0}, \binom{7}{1}, \binom{7}{2}, \dots, \binom{7}{7}, 0, 0, 0, 0, 0, 0, \dots$
- f) 2, -2, 2, -2, 2, -2, 2, -2, ...
- g) 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, ...
- h) 0, 0, 0, 1, 2, 3, 4, ...

4. Find a closed form for the generating function for each of these sequences. (Assume a general form for the terms of the sequence, using the most obvious choice of such a sequence.)

- a) -1, -1, -1, -1, -1, -1, 0, 0, 0, 0, 0, 0, ...
- b) 1, 3, 9, 27, 81, 243, 729, ...
- c) 0, 0, 3, -3, 3, -3, 3, -3, ...
- d) 1, 2, 1, 1, 1, 1, 1, 1, 1, ...
- e) $\binom{7}{0}, 2\binom{7}{1}, 2^2\binom{7}{2}, \dots, 2^7\binom{7}{7}, 0, 0, 0, 0, 0, \dots$
- f) -3, 3, -3, 3, -3, 3, ...
- g) 0, 1, -2, 4, -8, 16, -32, 64, ...
- h) 1, 0, 1, 0, 1, 0, 1, 0, ...

5. Find a closed form for the generating function for the sequence $\{a_n\}$, where

- a) $a_n = 5$ for all $n = 0, 1, 2, \dots$
- b) $a_n = 3^n$ for all $n = 0, 1, 2, \dots$
- c) $a_n = 2$ for $n = 3, 4, 5, \dots$ and $a_0 = a_1 = a_2 = 0$.
- d) $a_n = 2n + 3$ for all $n = 0, 1, 2, \dots$
- e) $a_n = \binom{8}{n}$ for all $n = 0, 1, 2, \dots$
- f) $a_n = \binom{n+4}{n}$ for all $n = 0, 1, 2, \dots$

6. Find a closed form for the generating function for the sequence $\{a_n\}$, where

- a) $a_n = -1$ for all $n = 0, 1, 2, \dots$
- b) $a_n = 2^n$ for $n = 1, 2, 3, 4, \dots$ and $a_0 = 0$.
- c) $a_n = n - 1$ for $n = 0, 1, 2, \dots$
- d) $a_n = 1/(n + 1)!$ for $n = 0, 1, 2, \dots$
- e) $a_n = \binom{n}{2}$ for $n = 0, 1, 2, \dots$
- f) $a_n = \binom{10}{n+1}$ for $n = 0, 1, 2, \dots$

7. For each of these generating functions, provide a closed formula for the sequence it determines.

- a) $(3x - 4)^3$
- b) $(x^3 + 1)^3$
- c) $1/(1 - 5x)$
- d) $x^3/(1 + 3x)$
- e) $x^2 + 3x + 7 + (1/(1 - x^2))$
- f) $(x^4/(1 - x^4)) - x^3 - x^2 - x - 1$
- g) $x^2/(1 - x)^2$
- h) $2e^{2x}$

8. For each of these generating functions, provide a closed formula for the sequence it determines.

- a) $(x^2 + 1)^3$
- b) $(3x - 1)^3$
- c) $1/(1 - 2x^2)$
- d) $x^2/(1 - x)^3$
- e) $x - 1 + (1/(1 - 3x))$
- f) $(1 + x^3)/(1 + x)^3$
- *g) $x/(1 + x + x^2)$
- h) $e^{3x^2} - 1$

9. Find the coefficient of x^{10} in the power series of each of these functions.

- a) $(1 + x^5 + x^{10} + x^{15} + \dots)^3$
- b) $(x^3 + x^4 + x^5 + x^6 + x^7 + \dots)^3$
- c) $(x^4 + x^5 + x^6)(x^3 + x^4 + x^5 + x^6 + x^7)(1 + x + x^2 + x^3 + x^4 + \dots)$
- d) $(x^2 + x^4 + x^6 + x^8 + \dots)(x^3 + x^6 + x^9 + \dots)(x^4 + x^8 + x^{12} + \dots)$
- e) $(1 + x^2 + x^4 + x^6 + x^8 + \dots)(1 + x^4 + x^8 + x^{12} + \dots)(1 + x^6 + x^{12} + x^{18} + \dots)$

10. Find the coefficient of x^9 in the power series of each of these functions.

- a) $(1 + x^3 + x^6 + x^9 + \dots)^3$
- b) $(x^2 + x^3 + x^4 + x^5 + x^6 + \dots)^3$
- c) $(x^3 + x^5 + x^6)(x^3 + x^4)(x + x^2 + x^3 + x^4 + \dots)$
- d) $(x + x^4 + x^7 + x^{10} + \dots)(x^2 + x^4 + x^6 + x^8 + \dots)$
- e) $(1 + x + x^2)^3$

11. Find the coefficient of x^{10} in the power series of each of these functions.

- a) $1/(1 - 2x)$
- b) $1/(1 + x)^2$
- c) $1/(1 - x)^3$
- d) $1/(1 + 2x)^4$
- e) $x^4/(1 - 3x)^3$

12. Find the coefficient of x^{12} in the power series of each of these functions.

- a) $1/(1 + 3x)$
- b) $1/(1 - 2x)^2$
- c) $1/(1 + x)^8$
- d) $1/(1 - 4x)^3$
- e) $x^3/(1 + 4x)^2$

13. Use generating functions to determine the number of different ways 10 identical balloons can be given to four children if each child receives at least two balloons.

14. Use generating functions to determine the number of different ways 12 identical action figures can be given to five children so that each child receives at most three action figures.

15. Use generating functions to determine the number of different ways 15 identical stuffed animals can be given to six children so that each child receives at least one but no more than three stuffed animals.

- 16.** Use generating functions to find the number of ways to choose a dozen bagels from three varieties—egg, salty, and plain—if at least two bagels of each kind but no more than three salty bagels are chosen.
- 17.** In how many ways can 25 identical donuts be distributed to four police officers so that each officer gets at least three but no more than seven donuts?
- 18.** Use generating functions to find the number of ways to select 14 balls from a jar containing 100 red balls, 100 blue balls, and 100 green balls so that no fewer than 3 and no more than 10 blue balls are selected. Assume that the order in which the balls are drawn does not matter.
- 19.** What is the generating function for the sequence $\{c_k\}$, where c_k is the number of ways to make change for k dollars using \$1 bills, \$2 bills, \$5 bills, and \$10 bills?
- 20.** What is the generating function for the sequence $\{c_k\}$, where c_k represents the number of ways to make change for k pesos using bills worth 10 pesos, 20 pesos, 50 pesos, and 100 pesos?
- 21.** Give a combinatorial interpretation of the coefficient of x^4 in the expansion $(1 + x + x^2 + x^3 + \dots)^3$. Use this interpretation to find this number.
- 22.** Give a combinatorial interpretation of the coefficient of x^6 in the expansion $(1 + x + x^2 + x^3 + \dots)^n$. Use this interpretation to find this number.
- 23. a)** What is the generating function for $\{a_k\}$, where a_k is the number of solutions of $x_1 + x_2 + x_3 = k$ when x_1 , x_2 , and x_3 are integers with $x_1 \geq 2$, $0 \leq x_2 \leq 3$, and $2 \leq x_3 \leq 5$?
b) Use your answer to part (a) to find a_6 .
- 24. a)** What is the generating function for $\{a_k\}$, where a_k is the number of solutions of $x_1 + x_2 + x_3 + x_4 = k$ when x_1 , x_2 , x_3 , and x_4 are integers with $x_1 \geq 3$, $1 \leq x_2 \leq 5$, $0 \leq x_3 \leq 4$, and $x_4 \geq 1$?
b) Use your answer to part (a) to find a_7 .
- 25.** Explain how generating functions can be used to find the number of ways in which postage of r cents can be pasted on an envelope using 3-cent, 4-cent, and 20-cent stamps.
a) Assume that the order the stamps are pasted on does not matter.
b) Assume that the stamps are pasted in a row and the order in which they are pasted on matters.
c) Use your answer to part (a) to determine the number of ways 46 cents of postage can be pasted on an envelope using 3-cent, 4-cent, and 20-cent stamps when the order the stamps are pasted on does not matter. (Use of a computer algebra program is advised.)
d) Use your answer to part (b) to determine the number of ways 46 cents of postage can be pasted in a row on an envelope using 3-cent, 4-cent, and 20-cent stamps when the order in which the stamps are pasted on matters. (Use of a computer algebra program is advised.)
- 26. a)** Show that $1/(1 - x - x^2 - x^3 - x^4 - x^5 - x^6)$ is the generating function for the number of ways that the sum n can be obtained when a die is rolled repeatedly and the order of the rolls matters.
- b)** Use part (a) to find the number of ways to roll a total of 8 when a die is rolled repeatedly, and the order of the rolls matters. (Use of a computer algebra package is advised.)
- 27.** Use generating functions (and a computer algebra package, if available) to find the number of ways to make change for \$1 using
a) dimes and quarters.
b) nickels, dimes, and quarters.
c) pennies, dimes, and quarters.
d) pennies, nickels, dimes, and quarters.
- 28.** Use generating functions (and a computer algebra package, if available) to find the number of ways to make change for \$1 using pennies, nickels, dimes, and quarters with
a) no more than 10 pennies.
b) no more than 10 pennies and no more than 10 nickels.
***c)** no more than 10 coins.
- 29.** Use generating functions to find the number of ways to make change for \$100 using
a) \$10, \$20, and \$50 bills.
b) \$5, \$10, \$20, and \$50 bills.
c) \$5, \$10, \$20, and \$50 bills if at least one bill of each denomination is used.
d) \$5, \$10, and \$20 bills if at least one and no more than four of each denomination is used.
- 30.** If $G(x)$ is the generating function for the sequence $\{a_k\}$, what is the generating function for each of these sequences?
a) $2a_0, 2a_1, 2a_2, 2a_3, \dots$
b) $0, a_0, a_1, a_2, a_3, \dots$ (assuming that terms follow the pattern of all but the first term)
c) $0, 0, 0, 0, a_2, a_3, \dots$ (assuming that terms follow the pattern of all but the first four terms)
d) a_2, a_3, a_4, \dots
e) $a_1, 2a_2, 3a_3, 4a_4, \dots$ [Hint: Calculus required here.]
f) $a_0^2, 2a_0a_1, a_1^2 + 2a_0a_2, 2a_0a_3 + 2a_1a_2, 2a_0a_4 + 2a_1a_3 + a_2^2, \dots$
- 31.** If $G(x)$ is the generating function for the sequence $\{a_k\}$, what is the generating function for each of these sequences?
a) $0, 0, 0, a_3, a_4, a_5, \dots$ (assuming that terms follow the pattern of all but the first three terms)
b) $a_0, 0, a_1, 0, a_2, 0, \dots$
c) $0, 0, 0, 0, a_0, a_1, a_2, \dots$ (assuming that terms follow the pattern of all but the first four terms)
d) $a_0, 2a_1, 4a_2, 8a_3, 16a_4, \dots$
e) $0, a_0, a_1/2, a_2/3, a_3/4, \dots$ [Hint: Calculus required here.]
f) $a_0, a_0 + a_1, a_0 + a_1 + a_2, a_0 + a_1 + a_2 + a_3, \dots$
- 32.** Use generating functions to solve the recurrence relation $a_k = 7a_{k-1}$ with the initial condition $a_0 = 5$.
- 33.** Use generating functions to solve the recurrence relation $a_k = 3a_{k-1} + 2$ with the initial condition $a_0 = 1$.
- 34.** Use generating functions to solve the recurrence relation $a_k = 3a_{k-1} + 4^{k-1}$ with the initial condition $a_0 = 1$.

35. Use generating functions to solve the recurrence relation $a_k = 5a_{k-1} - 6a_{k-2}$ with initial conditions $a_0 = 6$ and $a_1 = 30$.
36. Use generating functions to solve the recurrence relation $a_k = a_{k-1} + 2a_{k-2} + 2^k$ with initial conditions $a_0 = 4$ and $a_1 = 12$.
37. Use generating functions to solve the recurrence relation $a_k = 4a_{k-1} - 4a_{k-2} + k^2$ with initial conditions $a_0 = 2$ and $a_1 = 5$.
38. Use generating functions to solve the recurrence relation $a_k = 2a_{k-1} + 3a_{k-2} + 4^k + 6$ with initial conditions $a_0 = 20$, $a_1 = 60$.
39. Use generating functions to find an explicit formula for the Fibonacci numbers.
- *40. a) Show that if n is a positive integer, then

$$\binom{-1/2}{n} = \frac{\binom{2n}{n}}{(-4)^n}.$$

- b) Use the extended binomial theorem and part (a) to show that the coefficient of x^n in the expansion of $(1 - 4x)^{-1/2}$ is $\binom{2n}{n}$ for all nonnegative integers n .

- *41. (*Calculus required*) Let $\{C_n\}$ be the sequence of Catalan numbers, that is, the solution to the recurrence relation $C_n = \sum_{k=0}^{n-1} C_k C_{n-k-1}$ with $C_0 = C_1 = 1$ (see Example 5 in Section 8.1).
- a) Show that if $G(x)$ is the generating function for the sequence of Catalan numbers, then $xG(x)^2 - G(x) + 1 = 0$. Conclude (using the initial conditions) that $G(x) = (1 - \sqrt{1 - 4x})/(2x)$.
- b) Use Exercise 40 to conclude that

$$G(x) = \sum_{n=0}^{\infty} \frac{1}{n+1} \binom{2n}{n} x^n,$$

so that

$$C_n = \frac{1}{n+1} \binom{2n}{n}.$$

- c) Show that $C_n \geq 2^{n-1}$ for all positive integers n .

42. Use generating functions to prove Pascal's identity: $C(n, r) = C(n - 1, r) + C(n - 1, r - 1)$ when n and r are positive integers with $r < n$. [Hint: Use the identity $(1 + x)^n = (1 + x)^{n-1} + x(1 + x)^{n-1}$.]
43. Use generating functions to prove Vandermonde's identity: $C(m + n, r) = \sum_{k=0}^r C(m, r - k)C(n, k)$, whenever m , n , and r are nonnegative integers with r not exceeding either m or n . [Hint: Look at the coefficient of x^r in both sides of $(1 + x)^{m+n} = (1 + x)^m(1 + x)^n$.]
44. This exercise shows how to use generating functions to derive a formula for the sum of the first n squares.
- a) Show that $(x^2 + x)/(1 - x)^4$ is the generating function for the sequence $\{a_n\}$, where $a_n = 1^2 + 2^2 + \dots + n^2$.
- b) Use part (a) to find an explicit formula for the sum $1^2 + 2^2 + \dots + n^2$.

The **exponential generating function** for the sequence $\{a_n\}$ is the series

$$\sum_{n=0}^{\infty} \frac{a_n}{n!} x^n.$$

For example, the exponential generating function for the sequence 1, 1, 1, ... is the function $\sum_{n=0}^{\infty} x^n/n! = e^x$. (You will find this particular series useful in these exercises.) Note that e^x is the (ordinary) generating function for the sequence 1, 1, $1/2!$, $1/3!$, $1/4!$,

45. Find a closed form for the exponential generating function for the sequence $\{a_n\}$, where
- a) $a_n = 2$. b) $a_n = (-1)^n$.
 c) $a_n = 3^n$. d) $a_n = n + 1$.
 e) $a_n = 1/(n + 1)$.
46. Find a closed form for the exponential generating function for the sequence $\{a_n\}$, where
- a) $a_n = (-2)^n$. b) $a_n = -1$.
 c) $a_n = n$. d) $a_n = n(n - 1)$.
 e) $a_n = 1/((n + 1)(n + 2))$.
47. Find the sequence with each of these functions as its exponential generating function.
- a) $f(x) = e^{-x}$ b) $f(x) = 3x^{2x}$
 c) $f(x) = e^{3x} - 3e^{2x}$ d) $f(x) = (1 - x) + e^{-2x}$
 e) $f(x) = e^{-2x} - (1/(1 - x))$
 f) $f(x) = e^{-3x} - (1 + x) + (1/(1 - 2x))$
 g) $f(x) = e^{x^2}$
48. Find the sequence with each of these functions as its exponential generating function.
- a) $f(x) = e^{3x}$ b) $f(x) = 2e^{-3x+1}$
 c) $f(x) = e^{4x} + e^{-4x}$ d) $f(x) = (1 + 2x) + e^{3x}$
 e) $f(x) = e^x - (1/(1 + x))$
 f) $f(x) = xe^x$ g) $f(x) = e^{x^3}$
49. A coding system encodes messages using strings of octal (base 8) digits. A codeword is considered valid if and only if it contains an even number of 7s.
- a) Find a linear nonhomogeneous recurrence relation for the number of valid codewords of length n . What are the initial conditions?
- b) Solve this recurrence relation using Theorem 6 in Section 8.2.
- c) Solve this recurrence relation using generating functions.
- *50. A coding system encodes messages using strings of base 4 digits (that is, digits from the set {0, 1, 2, 3}). A codeword is valid if and only if it contains an even number of 0s and an even number of 1s. Let a_n equal the number of valid codewords of length n . Furthermore, let b_n , c_n , and d_n equal the number of strings of base 4 digits of length n with an even number of 0s and an odd number of 1s, with an odd number of 0s and an even number of 1s, and with an odd number of 0s and an odd number of 1s, respectively.
- a) Show that $d_n = 4^n - a_n - b_n - c_n$. Use this to show that $a_{n+1} = 2a_n + b_n + c_n$, $b_{n+1} = b_n - c_n + 4^n$, and $c_{n+1} = c_n - b_n + 4^n$.

- b) What are a_1, b_1, c_1 , and d_1 ?
- c) Use parts (a) and (b) to find a_3, b_3, c_3 , and d_3 .
- d) Use the recurrence relations in part (a), together with the initial conditions in part (b), to set up three equations relating the generating functions $A(x)$, $B(x)$, and $C(x)$ for the sequences $\{a_n\}$, $\{b_n\}$, and $\{c_n\}$, respectively.
- e) Solve the system of equations from part (d) to get explicit formulae for $A(x)$, $B(x)$, and $C(x)$ and use these to get explicit formulae for a_n, b_n, c_n , and d_n .

Generating functions are useful in studying the number of different types of partitions of an integer n . A **partition** of a positive integer is a way to write this integer as the sum of positive integers where repetition is allowed and the order of the integers in the sum does not matter. For example, the partitions of 5 (with no restrictions) are $1 + 1 + 1 + 1 + 1$, $1 + 1 + 1 + 2$, $1 + 1 + 3$, $1 + 2 + 2$, $1 + 4$, $2 + 3$, and 5. Exercises 51–56 illustrate some of these uses.

51. Show that the coefficient $p(n)$ of x^n in the formal power series expansion of $1/((1-x)(1-x^2)(1-x^3)\dots)$ equals the number of partitions of n .
52. Show that the coefficient $p_o(n)$ of x^n in the formal power series expansion of $1/((1-x)(1-x^3)(1-x^5)\dots)$ equals the number of partitions of n into odd integers, that is, the number of ways to write n as the sum of odd positive integers, where the order does not matter and repetitions are allowed.
53. Show that the coefficient $p_d(n)$ of x^n in the formal power series expansion of $(1+x)(1+x^2)(1+x^3)\dots$ equals the number of partitions of n into distinct parts, that is, the number of ways to write n as the sum of positive integers, where the order does not matter but no repetitions are allowed.
54. Find $p_o(n)$, the number of partitions of n into odd parts with repetitions allowed, and $p_d(n)$, the number of partitions of n into distinct parts, for $1 \leq n \leq 8$, by writing each partition of each type for each integer.
55. Show that if n is a positive integer, then the number of partitions of n into distinct parts equals the number of partitions of n into odd parts with repetitions allowed;

that is, $p_o(n) = p_d(n)$. [Hint: Show that the generating functions for $p_o(n)$ and $p_d(n)$ are equal.]

- *56.** (Requires calculus) Use the generating function of $p(n)$ to show that $p(n) \leq e^{C\sqrt{n}}$ for some constant C . [Hardy and Ramanujan showed that $p(n) \sim e^{\pi\sqrt{2/3}\sqrt{n}}/(4\sqrt{3}n)$, which means that the ratio of $p(n)$ and the right-hand side approaches 1 as n approaches infinity.]

Suppose that X is a random variable on a sample space S such that $X(s)$ is a nonnegative integer for all $s \in S$. The **probability generating function** for X is

$$G_X(x) = \sum_{k=0}^{\infty} p(X(s) = k)x^k.$$

- 57.** (Requires calculus) Show that if G_X is the probability generating function for a random variable X such that $X(s)$ is a nonnegative integer for all $s \in S$, then

- a) $G_X(1) = 1$.
- b) $E(X) = G'_X(1)$.
- c) $V(X) = G''_X(1) + G'_X(1) - G'_X(1)^2$.

- 58.** Let X be the random variable whose value is n if the first success occurs on the n th trial when independent Bernoulli trials are performed, each with probability of success p .

- a) Find a closed formula for the probability generating function G_X .
- b) Find the expected value and the variance of X using Exercise 57 and the closed form for the probability generating function found in part (a).

- 59.** Let m be a positive integer. Let X_m be the random variable whose value is n if the m th success occurs on the $(n+m)$ th trial when independent Bernoulli trials are performed, each with probability of success p .

- a) Using Exercise 32 in the Supplementary Exercises of Chapter 7, show that the probability generating function $G_{X_m}(x) = p^m/(1-qx)^m$, where $q = 1-p$.
- b) Find the expected value and the variance of X_m using Exercise 57 and the closed form for the probability generating function in part (a).

- 60.** Show that if X and Y are independent random variables on a sample space S such that $X(s)$ and $Y(s)$ are nonnegative integers for all $s \in S$, then $G_{X+Y}(x) = G_X(x)G_Y(x)$.

8.5 Inclusion–Exclusion

Introduction

A discrete mathematics class contains 30 women and 50 sophomores. How many students in the class are either women or sophomores? This question cannot be answered unless more information is provided. Adding the number of women in the class and the number of sophomores probably does not give the correct answer, because women sophomores are counted twice. This observation shows that the number of students in the class that are either sophomores or women is the sum of the number of women and the number of sophomores in the class minus the number of women sophomores. A technique for solving such counting problems was introduced in

Section 6.1. In this section we will generalize the ideas introduced in that section to solve problems that require us to count the number of elements in the union of more than two sets.

The Principle of Inclusion–Exclusion

How many elements are in the union of two finite sets? In Section 2.2 we showed that the number of elements in the union of the two sets A and B is the sum of the numbers of elements in the sets minus the number of elements in their intersection. That is,

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

As we showed in Section 6.1, the formula for the number of elements in the union of two sets is useful in counting problems. Examples 1–3 provide additional illustrations of the usefulness of this formula.

EXAMPLE 1 In a discrete mathematics class every student is a major in computer science or mathematics, or both. The number of students having computer science as a major (possibly along with mathematics) is 25; the number of students having mathematics as a major (possibly along with computer science) is 13; and the number of students majoring in both computer science and mathematics is 8. How many students are in this class?

Solution: Let A be the set of students in the class majoring in computer science and B be the set of students in the class majoring in mathematics. Then $A \cap B$ is the set of students in the class who are joint mathematics and computer science majors. Because every student in the class is majoring in either computer science or mathematics (or both), it follows that the number of students in the class is $|A \cup B|$. Therefore,

$$\begin{aligned} |A \cup B| &= |A| + |B| - |A \cap B| \\ &= 25 + 13 - 8 = 30. \end{aligned}$$

Therefore, there are 30 students in the class. This computation is illustrated in Figure 1. 

EXAMPLE 2 How many positive integers not exceeding 1000 are divisible by 7 or 11?

Solution: Let A be the set of positive integers not exceeding 1000 that are divisible by 7, and let B be the set of positive integers not exceeding 1000 that are divisible by 11. Then $A \cup B$ is the set of integers not exceeding 1000 that are divisible by either 7 or 11, and $A \cap B$ is the set of integers not exceeding 1000 that are divisible by both 7 and 11. From Example 2 of Section 4.1, we know that among the positive integers not exceeding 1000 there are $\lfloor 1000/7 \rfloor$ integers divisible by 7 and $\lfloor 1000/11 \rfloor$ divisible by 11. Because 7 and 11 are relatively prime, the integers divisible by both 7 and 11 are those divisible by $7 \cdot 11$. Consequently, there are $\lfloor 1000/(7 \cdot 11) \rfloor$ positive integers not exceeding 1000 that are divisible by both 7 and 11. It follows that there are

$$\begin{aligned} |A \cup B| &= |A| + |B| - |A \cap B| \\ &= \left\lfloor \frac{1000}{7} \right\rfloor + \left\lfloor \frac{1000}{11} \right\rfloor - \left\lfloor \frac{1000}{7 \cdot 11} \right\rfloor \\ &= 142 + 90 - 12 = 220 \end{aligned}$$

positive integers not exceeding 1000 that are divisible by either 7 or 11. This computation is illustrated in Figure 2. 

$$|A \cup B| = |A| + |B| - |A \cap B| = 25 + 13 - 8 = 30$$

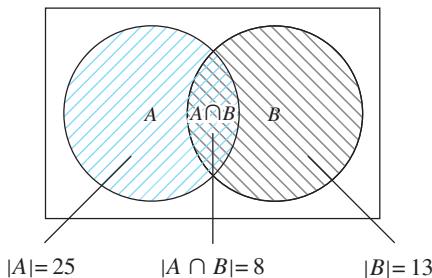


FIGURE 1 The Set of Students in a Discrete Mathematics Class.

$$|A \cup B| = |A| + |B| - |A \cap B| = 142 + 90 - 12 = 220$$

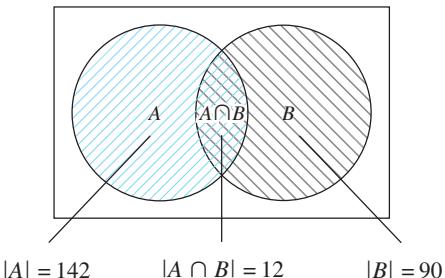


FIGURE 2 The Set of Positive Integers Not Exceeding 1000 Divisible by Either 7 or 11.

Example 3 shows how to find the number of elements in a finite universal set that are outside the union of two sets.

EXAMPLE 3 Suppose that there are 1807 freshmen at your school. Of these, 453 are taking a course in computer science, 567 are taking a course in mathematics, and 299 are taking courses in both computer science and mathematics. How many are not taking a course either in computer science or in mathematics?

Solution: To find the number of freshmen who are not taking a course in either mathematics or computer science, subtract the number that are taking a course in either of these subjects from the total number of freshmen. Let A be the set of all freshmen taking a course in computer science, and let B be the set of all freshmen taking a course in mathematics. It follows that $|A| = 453$, $|B| = 567$, and $|A \cap B| = 299$. The number of freshmen taking a course in either computer science or mathematics is

$$|A \cup B| = |A| + |B| - |A \cap B| = 453 + 567 - 299 = 721.$$

Consequently, there are $1807 - 721 = 1086$ freshmen who are not taking a course in computer science or mathematics. ◀

We will now begin our development of a formula for the number of elements in the union of a finite number of sets. The formula we will develop is called the **principle of inclusion-exclusion**. For concreteness, before we consider unions of n sets, where n is any positive integer, we will derive a formula for the number of elements in the union of three sets A , B , and C . To construct this formula, we note that $|A| + |B| + |C|$ counts each element that is in exactly one of the three sets once, elements that are in exactly two of the sets twice, and elements in all three sets three times. This is illustrated in the first panel in Figure 3.

To remove the overcount of elements in more than one of the sets, we subtract the number of elements in the intersections of all pairs of the three sets. We obtain

$$|A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C|.$$

This expression still counts elements that occur in exactly one of the sets once. An element that occurs in exactly two of the sets is also counted exactly once, because this element will occur in one of the three intersections of sets taken two at a time. However, those elements that occur in all three sets will be counted zero times by this expression, because they occur in all three intersections of sets taken two at a time. This is illustrated in the second panel in Figure 3.

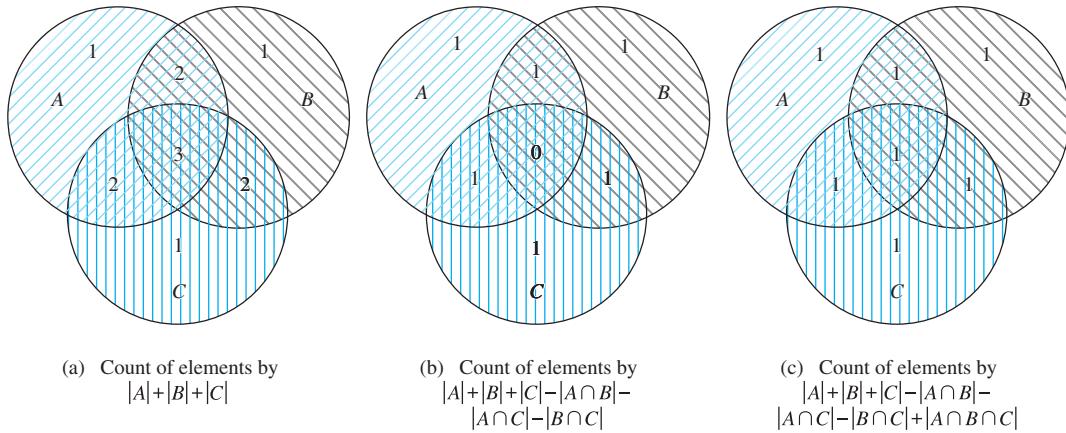


FIGURE 3 Finding a Formula for the Number of Elements in the Union of Three Sets.

To remedy this undercount, we add the number of elements in the intersection of all three sets. This final expression counts each element once, whether it is in one, two, or three of the sets. Thus,

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|.$$

This formula is illustrated in the third panel of Figure 3.

Example 4 illustrates how this formula can be used.

EXAMPLE 4 A total of 1232 students have taken a course in Spanish, 879 have taken a course in French, and 114 have taken a course in Russian. Further, 103 have taken courses in both Spanish and French, 23 have taken courses in both Spanish and Russian, and 14 have taken courses in both French and Russian. If 2092 students have taken at least one of Spanish, French, and Russian, how many students have taken a course in all three languages?

Solution: Let S be the set of students who have taken a course in Spanish, F the set of students who have taken a course in French, and R the set of students who have taken a course in Russian. Then

$$\begin{aligned} |S| &= 1232, & |F| &= 879, & |R| &= 114, \\ |S \cap F| &= 103, & |S \cap R| &= 23, & |F \cap R| &= 14, \end{aligned}$$

and

$$|S \cup F \cup R| = 2092.$$

When we insert these quantities into the equation

$$|S \cup F \cup R| = |S| + |F| + |R| - |S \cap F| - |S \cap R| - |F \cap R| + |S \cap F \cap R|$$

we obtain

$$2092 = 1232 + 879 + 114 - 103 - 23 - 14 + |S \cap F \cap R|.$$

We now solve for $|S \cap F \cap R|$. We find that $|S \cap F \cap R| = 7$. Therefore, there are seven students who have taken courses in Spanish, French, and Russian. This is illustrated in Figure 4. 

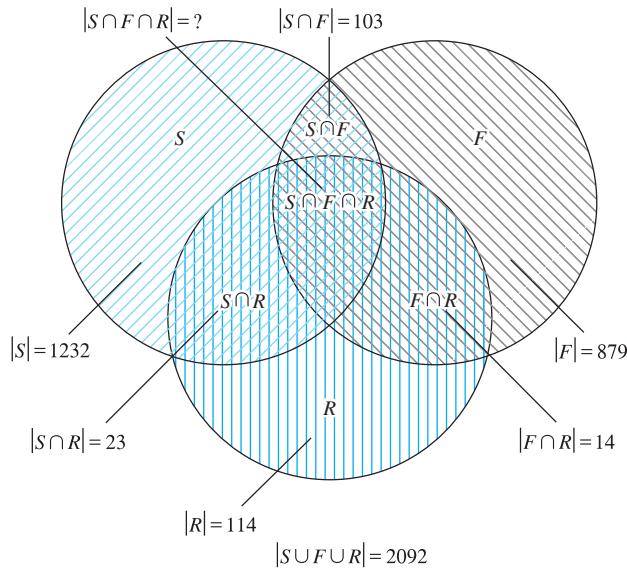


FIGURE 4 The Set of Students Who Have Taken Courses in Spanish, French, and Russian.

We will now state and prove the inclusion–exclusion principle, which tells us how many elements are in the union of a finite number of finite sets.

THEOREM 1

THE PRINCIPLE OF INCLUSION-EXCLUSION Let A_1, A_2, \dots, A_n be finite sets. Then

$$\begin{aligned} |A_1 \cup A_2 \cup \dots \cup A_n| &= \sum_{1 \leq i \leq n} |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| \\ &\quad + \sum_{1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k| - \dots + (-1)^{n+1} |A_1 \cap A_2 \cap \dots \cap A_n|. \end{aligned}$$

Proof: We will prove the formula by showing that an element in the union is counted exactly once by the right-hand side of the equation. Suppose that a is a member of exactly r of the sets A_1, A_2, \dots, A_n where $1 \leq r \leq n$. This element is counted $C(r, 1)$ times by $\sum |A_i|$. It is counted $C(r, 2)$ times by $\sum |A_i \cap A_j|$. In general, it is counted $C(r, m)$ times by the summation involving m of the sets A_i . Thus, this element is counted exactly

$$C(r, 1) - C(r, 2) + C(r, 3) - \dots + (-1)^{r+1} C(r, r)$$

times by the expression on the right-hand side of this equation. Our goal is to evaluate this quantity. By Corollary 2 of Section 6.4, we have

$$C(r, 0) - C(r, 1) + C(r, 2) - \dots + (-1)^r C(r, r) = 0.$$

Hence,

$$1 = C(r, 0) = C(r, 1) - C(r, 2) + \dots + (-1)^{r+1} C(r, r).$$

Therefore, each element in the union is counted exactly once by the expression on the right-hand side of the equation. This proves the principle of inclusion–exclusion. 

The inclusion–exclusion principle gives a formula for the number of elements in the union of n sets for every positive integer n . There are terms in this formula for the number of elements in the intersection of every nonempty subset of the collection of the n sets. Hence, there are $2^n - 1$ terms in this formula.

EXAMPLE 5 Give a formula for the number of elements in the union of four sets.



Solution: The inclusion–exclusion principle shows that

$$\begin{aligned} |A_1 \cup A_2 \cup A_3 \cup A_4| &= |A_1| + |A_2| + |A_3| + |A_4| \\ &\quad - |A_1 \cap A_2| - |A_1 \cap A_3| - |A_1 \cap A_4| - |A_2 \cap A_3| - |A_2 \cap A_4| \\ &\quad - |A_3 \cap A_4| + |A_1 \cap A_2 \cap A_3| + |A_1 \cap A_2 \cap A_4| + |A_1 \cap A_3 \cap A_4| \\ &\quad + |A_2 \cap A_3 \cap A_4| - |A_1 \cap A_2 \cap A_3 \cap A_4|. \end{aligned}$$

Note that this formula contains 15 different terms, one for each nonempty subset of $\{A_1, A_2, A_3, A_4\}$. 

Exercises

1. How many elements are in $A_1 \cup A_2$ if there are 12 elements in A_1 , 18 elements in A_2 , and
 - $A_1 \cap A_2 = \emptyset$
 - $|A_1 \cap A_2| = 1$
 - $|A_1 \cap A_2| = 6$
 - $A_1 \subseteq A_2$
2. There are 345 students at a college who have taken a course in calculus, 212 who have taken a course in discrete mathematics, and 188 who have taken courses in both calculus and discrete mathematics. How many students have taken a course in either calculus or discrete mathematics?
3. A survey of households in the United States reveals that 96% have at least one television set, 98% have telephone service, and 95% have telephone service and at least one television set. What percentage of households in the United States have neither telephone service nor a television set?
4. A marketing report concerning personal computers states that 650,000 owners will buy a printer for their machines next year and 1,250,000 will buy at least one software package. If the report states that 1,450,000 owners will buy either a printer or at least one software package, how many will buy both a printer and at least one software package?
5. Find the number of elements in $A_1 \cup A_2 \cup A_3$ if there are 100 elements in each set and if
 - the sets are pairwise disjoint.
 - there are 50 common elements in each pair of sets and no elements in all three sets.
- c) there are 50 common elements in each pair of sets and 25 elements in all three sets.
d) the sets are equal.
6. Find the number of elements in $A_1 \cup A_2 \cup A_3$ if there are 100 elements in A_1 , 1000 in A_2 , and 10,000 in A_3 if
 - $A_1 \subseteq A_2$ and $A_2 \subseteq A_3$.
 - the sets are pairwise disjoint.
 - there are two elements common to each pair of sets and one element in all three sets.
7. There are 2504 computer science students at a school. Of these, 1876 have taken a course in Java, 999 have taken a course in Linux, and 345 have taken a course in C. Further, 876 have taken courses in both Java and Linux, 231 have taken courses in both Linux and C, and 290 have taken courses in both Java and C. If 189 of these students have taken courses in Linux, Java, and C, how many of these 2504 students have not taken a course in any of these three programming languages?
8. In a survey of 270 college students, it is found that 64 like brussels sprouts, 94 like broccoli, 58 like cauliflower, 26 like both brussels sprouts and broccoli, 28 like both brussels sprouts and cauliflower, 22 like both broccoli and cauliflower, and 14 like all three vegetables. How many of the 270 students do not like any of these vegetables?
9. How many students are enrolled in a course either in calculus, discrete mathematics, data structures, or programming languages at a school if there are 507, 292, 312, and 344 students in these courses, respectively; 14 in both calculus and data structures; 213 in both calculus and programming languages; 211 in both discrete mathematics

and data structures; 43 in both discrete mathematics and programming languages; and no student may take calculus and discrete mathematics, or data structures and programming languages, concurrently?

10. Find the number of positive integers not exceeding 100 that are not divisible by 5 or by 7.
11. Find the number of positive integers not exceeding 100 that are either odd or the square of an integer.
12. Find the number of positive integers not exceeding 1000 that are either the square or the cube of an integer.
13. How many bit strings of length eight do not contain six consecutive 0s?
- *14. How many permutations of the 26 letters of the English alphabet do not contain any of the strings *fish*, *rat* or *bird*?
15. How many permutations of the 10 digits either begin with the 3 digits 987, contain the digits 45 in the fifth and sixth positions, or end with the 3 digits 123?
16. How many elements are in the union of four sets if each of the sets has 100 elements, each pair of the sets shares 50 elements, each three of the sets share 25 elements, and there are 5 elements in all four sets?
17. How many elements are in the union of four sets if the sets have 50, 60, 70, and 80 elements, respectively, each pair of the sets has 5 elements in common, each triple of the sets has 1 common element, and no element is in all four sets?
18. How many terms are there in the formula for the number of elements in the union of 10 sets given by the principle of inclusion–exclusion?
19. Write out the explicit formula given by the principle of inclusion–exclusion for the number of elements in the union of five sets.
20. How many elements are in the union of five sets if the sets contain 10,000 elements each, each pair of sets has 1000 common elements, each triple of sets has 100 common elements, every four of the sets have 10 common elements, and there is 1 element in all five sets?
21. Write out the explicit formula given by the principle of inclusion–exclusion for the number of elements in the union of six sets when it is known that no three of these sets have a common intersection.
- *22. Prove the principle of inclusion–exclusion using mathematical induction.
23. Let E_1 , E_2 , and E_3 be three events from a sample space S . Find a formula for the probability of $E_1 \cup E_2 \cup E_3$.
24. Find the probability that when a fair coin is flipped five times tails comes up exactly three times, the first and last flips come up tails, or the second and fourth flips come up heads.
25. Find the probability that when four numbers from 1 to 100, inclusive, are picked at random with no repetitions allowed, either all are odd, all are divisible by 3, or all are divisible by 5.
26. Find a formula for the probability of the union of four events in a sample space if no three of them can occur at the same time.
27. Find a formula for the probability of the union of five events in a sample space if no four of them can occur at the same time.
28. Find a formula for the probability of the union of n events in a sample space when no two of these events can occur at the same time.
29. Find a formula for the probability of the union of n events in a sample space.

8.6 Applications of Inclusion–Exclusion

Introduction

Many counting problems can be solved using the principle of inclusion–exclusion. For instance, we can use this principle to find the number of primes less than a positive integer. Many problems can be solved by counting the number of onto functions from one finite set to another. The inclusion–exclusion principle can be used to find the number of such functions. The famous hatcheck problem can be solved using the principle of inclusion–exclusion. This problem asks for the probability that no person is given the correct hat back by a hatcheck person who gives the hats back randomly.

An Alternative Form of Inclusion–Exclusion

There is an alternative form of the principle of inclusion–exclusion that is useful in counting problems. In particular, this form can be used to solve problems that ask for the number of elements in a set that have none of n properties P_1, P_2, \dots, P_n .

Let A_i be the subset containing the elements that have property P_i . The number of elements with all the properties $P_{i_1}, P_{i_2}, \dots, P_{i_k}$ will be denoted by $N(P_{i_1}P_{i_2} \dots P_{i_k})$.

Writing these quantities in terms of sets, we have

$$|A_{i_1} \cap A_{i_2} \cap \cdots \cap A_{i_k}| = N(P_{i_1} P_{i_2} \dots P_{i_k}).$$

If the number of elements with none of the properties P_1, P_2, \dots, P_n is denoted by $N(P'_1 P'_2 \dots P'_n)$ and the number of elements in the set is denoted by N , it follows that

$$N(P'_1 P'_2 \dots P'_n) = N - |A_1 \cup A_2 \cup \cdots \cup A_n|.$$

From the inclusion–exclusion principle, we see that

$$\begin{aligned} N(P'_1 P'_2 \dots P'_n) &= N - \sum_{1 \leq i \leq n} N(P_i) + \sum_{1 \leq i < j \leq n} N(P_i P_j) \\ &\quad - \sum_{1 \leq i < j < k \leq n} N(P_i P_j P_k) + \cdots + (-1)^n N(P_1 P_2 \dots P_n). \end{aligned}$$

Example 1 shows how the principle of inclusion–exclusion can be used to determine the number of solutions in integers of an equation with constraints.

EXAMPLE 1

How many solutions does

$$x_1 + x_2 + x_3 = 11$$

have, where x_1, x_2 , and x_3 are nonnegative integers with $x_1 \leq 3$, $x_2 \leq 4$, and $x_3 \leq 6$?

Solution: To apply the principle of inclusion–exclusion, let a solution have property P_1 if $x_1 > 3$, property P_2 if $x_2 > 4$, and property P_3 if $x_3 > 6$. The number of solutions satisfying the inequalities $x_1 \leq 3$, $x_2 \leq 4$, and $x_3 \leq 6$ is

$$\begin{aligned} N(P'_1 P'_2 P'_3) &= N - N(P_1) - N(P_2) - N(P_3) + N(P_1 P_2) \\ &\quad + N(P_1 P_3) + N(P_2 P_3) - N(P_1 P_2 P_3). \end{aligned}$$

Using the same techniques as in Example 5 of Section 6.5, it follows that

- $N = \text{total number of solutions} = C(3 + 11 - 1, 11) = 78$,
- $N(P_1) = (\text{number of solutions with } x_1 \geq 4) = C(3 + 7 - 1, 7) = C(9, 7) = 36$,
- $N(P_2) = (\text{number of solutions with } x_2 \geq 5) = C(3 + 6 - 1, 6) = C(8, 6) = 28$,
- $N(P_3) = (\text{number of solutions with } x_3 \geq 7) = C(3 + 4 - 1, 4) = C(6, 4) = 15$,
- $N(P_1 P_2) = (\text{number of solutions with } x_1 \geq 4 \text{ and } x_2 \geq 5) = C(3 + 2 - 1, 2) = C(4, 2) = 6$,
- $N(P_1 P_3) = (\text{number of solutions with } x_1 \geq 4 \text{ and } x_3 \geq 7) = C(3 + 0 - 1, 0) = 1$,
- $N(P_2 P_3) = (\text{number of solutions with } x_2 \geq 5 \text{ and } x_3 \geq 7) = 0$,
- $N(P_1 P_2 P_3) = (\text{number of solutions with } x_1 \geq 4, x_2 \geq 5, \text{ and } x_3 \geq 7) = 0$.

Inserting these quantities into the formula for $N(P'_1 P'_2 P'_3)$ shows that the number of solutions with $x_1 \leq 3$, $x_2 \leq 4$, and $x_3 \leq 6$ equals

$$N(P'_1 P'_2 P'_3) = 78 - 36 - 28 - 15 + 6 + 1 + 0 - 0 = 6.$$



The Sieve of Eratosthenes

In Section 4.3 we showed how to use the sieve of Eratosthenes to find all primes less than a specified positive integer n . Using the principle of inclusion–exclusion, we can find the number of primes not exceeding a specified positive integer with the same reasoning as is used in the sieve of Eratosthenes. Recall that a composite integer is divisible by a prime not exceeding its square root. So, to find the number of primes not exceeding 100, first note that composite integers not exceeding 100 must have a prime factor not exceeding 10. Because the only primes not exceeding 10 are 2, 3, 5, and 7, the primes not exceeding 100 are these four primes and those positive integers greater than 1 and not exceeding 100 that are divisible by none of 2, 3, 5, or 7. To apply the principle of inclusion–exclusion, let P_1 be the property that an integer is divisible by 2, let P_2 be the property that an integer is divisible by 3, let P_3 be the property that an integer is divisible by 5, and let P_4 be the property that an integer is divisible by 7. Thus, the number of primes not exceeding 100 is

$$4 + N(P'_1 P'_2 P'_3 P'_4).$$

Because there are 99 positive integers greater than 1 and not exceeding 100, the principle of inclusion–exclusion shows that

$$\begin{aligned} N(P'_1 P'_2 P'_3 P'_4) &= 99 - N(P_1) - N(P_2) - N(P_3) - N(P_4) \\ &\quad + N(P_1 P_2) + N(P_1 P_3) + N(P_1 P_4) + N(P_2 P_3) + N(P_2 P_4) + N(P_3 P_4) \\ &\quad - N(P_1 P_2 P_3) - N(P_1 P_2 P_4) - N(P_1 P_3 P_4) - N(P_2 P_3 P_4) \\ &\quad + N(P_1 P_2 P_3 P_4). \end{aligned}$$

The number of integers not exceeding 100 (and greater than 1) that are divisible by all the primes in a subset of $\{2, 3, 5, 7\}$ is $\lfloor 100/N \rfloor$, where N is the product of the primes in this subset. (This follows because any two of these primes have no common factor.) Consequently,

$$\begin{aligned} N(P'_1 P'_2 P'_3 P'_4) &= 99 - \left\lfloor \frac{100}{2} \right\rfloor - \left\lfloor \frac{100}{3} \right\rfloor - \left\lfloor \frac{100}{5} \right\rfloor - \left\lfloor \frac{100}{7} \right\rfloor \\ &\quad + \left\lfloor \frac{100}{2 \cdot 3} \right\rfloor + \left\lfloor \frac{100}{2 \cdot 5} \right\rfloor + \left\lfloor \frac{100}{2 \cdot 7} \right\rfloor + \left\lfloor \frac{100}{3 \cdot 5} \right\rfloor + \left\lfloor \frac{100}{3 \cdot 7} \right\rfloor + \left\lfloor \frac{100}{5 \cdot 7} \right\rfloor \\ &\quad - \left\lfloor \frac{100}{2 \cdot 3 \cdot 5} \right\rfloor - \left\lfloor \frac{100}{2 \cdot 3 \cdot 7} \right\rfloor - \left\lfloor \frac{100}{2 \cdot 5 \cdot 7} \right\rfloor - \left\lfloor \frac{100}{3 \cdot 5 \cdot 7} \right\rfloor + \left\lfloor \frac{100}{2 \cdot 3 \cdot 5 \cdot 7} \right\rfloor \\ &= 99 - 50 - 33 - 20 - 14 + 16 + 10 + 7 + 6 + 4 + 2 - 3 - 2 - 1 - 0 + 0 \\ &= 21. \end{aligned}$$

Hence, there are $4 + 21 = 25$ primes not exceeding 100.

The Number of Onto Functions

The principle of inclusion–exclusion can also be used to determine the number of onto functions from a set with m elements to a set with n elements. First consider Example 2.

EXAMPLE 2 How many onto functions are there from a set with six elements to a set with three elements?

Solution: Suppose that the elements in the codomain are b_1 , b_2 , and b_3 . Let P_1 , P_2 , and P_3 be the properties that b_1 , b_2 , and b_3 are not in the range of the function, respectively. Note that

a function is onto if and only if it has none of the properties P_1 , P_2 , or P_3 . By the inclusion–exclusion principle it follows that the number of onto functions from a set with six elements to a set with three elements is

$$\begin{aligned} N(P'_1 P'_2 P'_3) &= N - [N(P_1) + N(P_2) + N(P_3)] \\ &\quad + [N(P_1 P_2) + N(P_1 P_3) + N(P_2 P_3)] - N(P_1 P_2 P_3), \end{aligned}$$

where N is the total number of functions from a set with six elements to one with three elements. We will evaluate each of the terms on the right-hand side of this equation.

From Example 6 of Section 6.1, it follows that $N = 3^6$. Note that $N(P_i)$ is the number of functions that do not have b_i in their range. Hence, there are two choices for the value of the function at each element of the domain. Therefore, $N(P_i) = 2^6$. Furthermore, there are $C(3, 1)$ terms of this kind. Note that $N(P_i P_j)$ is the number of functions that do not have b_i and b_j in their range. Hence, there is only one choice for the value of the function at each element of the domain. Therefore, $N(P_i P_j) = 1^6 = 1$. Furthermore, there are $C(3, 2)$ terms of this kind. Also, note that $N(P_1 P_2 P_3) = 0$, because this term is the number of functions that have none of b_1 , b_2 , and b_3 in their range. Clearly, there are no such functions. Therefore, the number of onto functions from a set with six elements to one with three elements is

$$3^6 - C(3, 1)2^6 + C(3, 2)1^6 = 729 - 192 + 3 = 540.$$



The general result that tells us how many onto functions there are from a set with m elements to one with n elements will now be stated. The proof of this result is left as an exercise for the reader.

THEOREM 1

Let m and n be positive integers with $m \geq n$. Then, there are

$$n^m - C(n, 1)(n - 1)^m + C(n, 2)(n - 2)^m - \cdots + (-1)^{n-1} C(n, n - 1) \cdot 1^m$$

onto functions from a set with m elements to a set with n elements.

Counting onto functions is much harder than counting one-to-one functions!

An onto function from a set with m elements to a set with n elements corresponds to a way to distribute the m elements in the domain to n indistinguishable boxes so that no box is empty, and then to associate each of the n elements of the codomain to a box. This means that the number of onto functions from a set with m elements to a set with n elements is the number of ways to distribute m distinguishable objects to n indistinguishable boxes so that no box is empty multiplied by the number of permutations of a set with n elements. Consequently, the number of onto functions from a set with m elements to a set with n elements equals $n!S(m, n)$, where $S(m, n)$ is a *Stirling number of the second kind* defined in Section 6.5. This means that we can use Theorem 1 to deduce the formula given in Section 6.5 for $S(m, n)$. (See Chapter 6 of [MiRo91] for more details about Stirling numbers of the second kind.)

One of the many different applications of Theorem 1 will now be described.

EXAMPLE 3

How many ways are there to assign five different jobs to four different employees if every employee is assigned at least one job?

Solution: Consider the assignment of jobs as a function from the set of five jobs to the set of four employees. An assignment where every employee gets at least one job is the same as an

onto function from the set of jobs to the set of employees. Hence, by Theorem 1 it follows that there are

$$4^5 - C(4, 1)3^5 + C(4, 2)2^5 - C(4, 3)1^5 = 1024 - 972 + 192 - 4 = 240$$

ways to assign the jobs so that each employee is assigned at least one job. 

Derangements

The principle of inclusion–exclusion will be used to count the permutations of n objects that leave no objects in their original positions. Consider Example 4.

EXAMPLE 4 The Hatchet Problem A new employee checks the hats of n people at a restaurant, forgetting to put claim check numbers on the hats. When customers return for their hats, the checker gives them back hats chosen at random from the remaining hats. What is the probability that no one receives the correct hat? 

Remark: The answer is the number of ways the hats can be arranged so that there is no hat in its original position divided by $n!$, the number of permutations of n hats. We will return to this example after we find the number of permutations of n objects that leave no objects in their original position.



A **derangement** is a permutation of objects that leaves no object in its original position. To solve the problem posed in Example 4 we will need to determine the number of derangements of a set of n objects.

EXAMPLE 5 The permutation 21453 is a derangement of 12345 because no number is left in its original position. However, 21543 is not a derangement of 12345, because this permutation leaves 4 fixed. 

Let D_n denote the number of derangements of n objects. For instance, $D_3 = 2$, because the derangements of 123 are 231 and 312. We will evaluate D_n , for all positive integers n , using the principle of inclusion–exclusion.

THEOREM 2

The number of derangements of a set with n elements is

$$D_n = n! \left[1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!} \right].$$

Proof: Let a permutation have property P_i if it fixes element i . The number of derangements is the number of permutations having none of the properties P_i for $i = 1, 2, \dots, n$. This means that

$$D_n = N(P'_1 P'_2 \dots P'_n).$$

Using the principle of inclusion–exclusion, it follows that

$$D_n = N - \sum_i N(P_i) + \sum_{i < j} N(P_i P_j) - \sum_{i < j < k} N(P_i P_j P_k) + \cdots + (-1)^n N(P_1 P_2 \dots P_n),$$

where N is the number of permutations of n elements. This equation states that the number of permutations that fix no elements equals the total number of permutations, less the number that fix at least one element, plus the number that fix at least two elements, less the number that fix at least three elements, and so on. All the quantities that occur on the right-hand side of this equation will now be found.

First, note that $N = n!$, because N is simply the total number of permutations of n elements. Also, $N(P_i) = (n - 1)!$. This follows from the product rule, because $N(P_i)$ is the number of permutations that fix element i , so the i th position of the permutation is determined, but each of the remaining positions can be filled arbitrarily. Similarly,

$$N(P_i P_j) = (n - 2)!,$$

because this is the number of permutations that fix elements i and j , but where the other $n - 2$ elements can be arranged arbitrarily. In general, note that

$$N(P_{i_1} P_{i_2} \dots P_{i_m}) = (n - m)!,$$

because this is the number of permutations that fix elements i_1, i_2, \dots, i_m , but where the other $n - m$ elements can be arranged arbitrarily. Because there are $C(n, m)$ ways to choose m elements from n , it follows that

$$\sum_{1 \leq i \leq n} N(P_i) = C(n, 1)(n - 1)!,$$

$$\sum_{1 \leq i < j \leq n} N(P_i P_j) = C(n, 2)(n - 2)!,$$

and in general,

$$\sum_{1 \leq i_1 < i_2 < \dots < i_m \leq n} N(P_{i_1} P_{i_2} \dots P_{i_m}) = C(n, m)(n - m)!.$$

Consequently, inserting these quantities into our formula for D_n gives

$$\begin{aligned} D_n &= n! - C(n, 1)(n - 1)! + C(n, 2)(n - 2)! - \dots + (-1)^n C(n, n)(n - n)! \\ &= n! - \frac{n!}{1!(n - 1)!}(n - 1)! + \frac{n!}{2!(n - 2)!}(n - 2)! - \dots + (-1)^n \frac{n!}{n! 0!} 0!. \end{aligned}$$

Simplifying this expression gives

$$D_n = n! \left[1 - \frac{1}{1!} + \frac{1}{2!} - \dots + (-1)^n \frac{1}{n!} \right].$$



HISTORICAL NOTE In *rencontres* (matches), an old French card game, the 52 cards in a deck are laid out in a row. The cards of a second deck are laid out with one card of the second deck on top of each card of the first deck. The score is determined by counting the number of matching cards in the two decks. In 1708 Pierre Raymond de Montmort (1678–1719) posed *le problème de rencontres*: What is the probability that no matches take place in the game of rencontres? The solution to Montmort's problem is the probability that a randomly selected permutation of 52 objects is a derangement, namely, $D_{52}/52!$, which, as we will see, is approximately $1/e$.

TABLE 1 The Probability of a Derangement.

<i>n</i>	2	3	4	5	6	7
$D_n/n!$	0.50000	0.33333	0.37500	0.36667	0.36806	0.36786

It is now simple to find D_n for a given positive integer n . For instance, using Theorem 2, it follows that

$$D_3 = 3! \left[1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} \right] = 6 \left(1 - 1 + \frac{1}{2} - \frac{1}{6} \right) = 2,$$

as we have previously remarked.

The solution of the problem in Example 4 can now be given.

Solution: The probability that no one receives the correct hat is $D_n/n!$. By Theorem 2, this probability is

$$\frac{D_n}{n!} = 1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^n \frac{1}{n!}.$$

The values of this probability for $2 \leq n \leq 7$ are displayed in Table 1.

Using methods from calculus it can be shown that

$$e^{-1} = 1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^n \frac{1}{n!} + \cdots \approx 0.368.$$

Because this is an alternating series with terms tending to zero, it follows that as n grows without bound, the probability that no one receives the correct hat converges to $e^{-1} \approx 0.368$. In fact, this probability can be shown to be within $1/(n+1)!$ of e^{-1} . 

Exercises

- Suppose that in a bushel of 100 apples there are 20 that have worms in them and 15 that have bruises. Only those apples with neither worms nor bruises can be sold. If there are 10 bruised apples that have worms in them, how many of the 100 apples can be sold?
- Of 1000 applicants for a mountain-climbing trip in the Himalayas, 450 get altitude sickness, 622 are not in good enough shape, and 30 have allergies. An applicant qualifies if and only if this applicant does not get altitude sickness, is in good shape, and does not have allergies. If there are 111 applicants who get altitude sickness and are not in good enough shape, 14 who get altitude sickness and have allergies, 18 who are not in good enough shape and have allergies, and 9 who get altitude sickness, are not in good enough shape, and have allergies, how many applicants qualify?
- How many solutions does the equation $x_1 + x_2 + x_3 = 13$ have where x_1, x_2 , and x_3 are nonnegative integers less than 6?
- Find the number of solutions of the equation $x_1 + x_2 + x_3 + x_4 = 17$, where $x_i, i = 1, 2, 3, 4$, are nonnegative integers such that $x_1 \leq 3, x_2 \leq 4, x_3 \leq 5$, and $x_4 \leq 8$.
- Find the number of primes less than 200 using the principle of inclusion-exclusion.
- An integer is called **squarefree** if it is not divisible by the square of a positive integer greater than 1. Find the number of squarefree positive integers less than 100.
- How many positive integers less than 10,000 are not the second or higher power of an integer?
- How many onto functions are there from a set with seven elements to one with five elements?
- How many ways are there to distribute six different toys to three different children such that each child gets at least one toy?
- In how many ways can eight distinct balls be distributed into three distinct urns if each urn must contain at least one ball?

11. In how many ways can seven different jobs be assigned to four different employees so that each employee is assigned at least one job and the most difficult job is assigned to the best employee?
12. List all the derangements of $\{1, 2, 3, 4\}$.
13. How many derangements are there of a set with seven elements?
14. What is the probability that none of 10 people receives the correct hat if a hatcheck person hands their hats back randomly?
15. A machine that inserts letters into envelopes goes haywire and inserts letters randomly into envelopes. What is the probability that in a group of 100 letters
 - a) no letter is put into the correct envelope?
 - b) exactly one letter is put into the correct envelope?
 - c) exactly 98 letters are put into the correct envelopes?
 - d) exactly 99 letters are put into the correct envelopes?
 - e) all letters are put into the correct envelopes?
16. A group of n students is assigned seats for each of two classes in the same classroom. How many ways can these seats be assigned if no student is assigned the same seat for both classes?
- *17. How many ways can the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 be arranged so that no even digit is in its original position?
- *18. Use a combinatorial argument to show that the sequence $\{D_n\}$, where D_n denotes the number of derangements of n objects, satisfies the recurrence relation

$$D_n = (n - 1)(D_{n-1} + D_{n-2})$$

for $n \geq 2$. [Hint: Note that there are $n - 1$ choices for the first element k of a derangement. Consider separately the derangements that start with k that do and do not have 1 in the k th position.]

- *19. Use Exercise 18 to show that

$$D_n = nD_{n-1} + (-1)^n$$

for $n \geq 1$.

20. Use Exercise 19 to find an explicit formula for D_n .
 21. For which positive integers n is D_n , the number of derangements of n objects, even?
 22. Suppose that p and q are distinct primes. Use the principle of inclusion-exclusion to find $\phi(pq)$, the number of positive integers not exceeding pq that are relatively prime to pq .
 - *23. Use the principle of inclusion-exclusion to derive a formula for $\phi(n)$ when the prime factorization of n is
- $$n = p_1^{a_1} p_2^{a_2} \cdots p_m^{a_m}.$$
- *24. Show that if n is a positive integer, then
- $$\begin{aligned} n! &= C(n, 0)D_n + C(n, 1)D_{n-1} \\ &\quad + \cdots + C(n, n-1)D_1 + C(n, n)D_0, \end{aligned}$$
- where D_k is the number of derangements of k objects.
25. How many derangements of $\{1, 2, 3, 4, 5, 6\}$ begin with the integers 1, 2, and 3, in some order?
 26. How many derangements of $\{1, 2, 3, 4, 5, 6\}$ end with the integers 1, 2, and 3, in some order?
 27. Prove Theorem 1.

Key Terms and Results

TERMS

recurrence relation: a formula expressing terms of a sequence, except for some initial terms, as a function of one or more previous terms of the sequence

initial conditions for a recurrence relation: the values of the terms of a sequence satisfying the recurrence relation before this relation takes effect

dynamic programming: an algorithmic paradigm that finds the solution to an optimization problem by recursively breaking down the problem into overlapping subproblems and combining their solutions with the help of a recurrence relation

linear homogeneous recurrence relation with constant coefficients: a recurrence relation that expresses the terms of a sequence, except initial terms, as a linear combination of previous terms

characteristic roots of a linear homogeneous recurrence relation with constant coefficients: the roots of the polynomial associated with a linear homogeneous recurrence relation with constant coefficients

linear nonhomogeneous recurrence relation with constant coefficients

coefficients: a recurrence relation that expresses the terms of a sequence, except for initial terms, as a linear combination of previous terms plus a function that is not identically zero that depends only on the index

divide-and-conquer algorithm: an algorithm that solves a problem recursively by splitting it into a fixed number of smaller non-overlapping subproblems of the same type

generating function of a sequence: the formal series that has the n th term of the sequence as the coefficient of x^n

sieve of Eratosthenes: a procedure for finding the primes less than a specified positive integer

derangement: a permutation of objects such that no object is in its original place

RESULTS

the formula for the number of elements in the union of two finite sets:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

the formula for the number of elements in the union of three finite sets:

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| \\ - |B \cap C| + |A \cap B \cap C|$$

the principle of inclusion–exclusion:

$$|A_1 \cup A_2 \cup \dots \cup A_n| = \sum_{1 \leq i \leq n} |A_i| - \sum_{1 \leq i < j \leq n} |A_i \cap A_j| \\ + \sum_{1 \leq i < j < k \leq n} |A_i \cap A_j \cap A_k| \\ - \dots + (-1)^{n+1} |A_1 \cap A_2 \cap \dots \cap A_n|$$

Review Questions

1. a) What is a recurrence relation?
b) Find a recurrence relation for the amount of money that will be in an account after n years if \$1,000,000 is deposited in an account yielding 9% annually.
 2. Explain how the Fibonacci numbers are used to solve Fibonacci's problem about rabbits.
 3. a) Find a recurrence relation for the number of steps needed to solve the Tower of Hanoi puzzle.
b) Show how this recurrence relation can be solved using iteration.
 4. a) Explain how to find a recurrence relation for the number of bit strings of length n not containing two consecutive 1s.
b) Describe another counting problem that has a solution satisfying the same recurrence relation.
 5. a) What is dynamic programming and how are recurrence relations used in algorithms that follow this paradigm?
b) Explain how dynamic programming can be used to schedule talks in a lecture hall from a set of possible talks to maximize overall attendance.
 6. Define a linear homogeneous recurrence relation of degree k .
 7. a) Explain how to solve linear homogeneous recurrence relations of degree 2.
b) Solve the recurrence relation $a_n = 13a_{n-1} - 22a_{n-2}$ for $n \geq 2$ if $a_0 = 3$ and $a_1 = 15$.
c) Solve the recurrence relation $a_n = 14a_{n-1} - 49a_{n-2}$ for $n \geq 2$ if $a_0 = 3$ and $a_1 = 35$.
 8. a) Explain how to find $f(b^k)$ where k is a positive integer if $f(n)$ satisfies the divide-and-conquer recurrence relation $f(n) = af(n/b) + g(n)$ whenever b divides the positive integer n .
b) Find $f(256)$ if $f(n) = 3f(n/4) + 5n/4$ and $f(1) = 7$.
- the number of onto functions from a set with m elements to a set with n elements:**
- $$n^m - C(n, 1)(n-1)^m + C(n, 2)(n-2)^m \\ - \dots + (-1)^{n-1} C(n, n-1) \cdot 1^m$$
- the number of derangements of n objects:**
- $$D_n = n! \left[1 - \frac{1}{1!} + \frac{1}{2!} - \dots + (-1)^n \frac{1}{n!} \right]$$
9. a) Derive a divide-and-conquer recurrence relation for the number of comparisons used to find a number in a list using a binary search.
b) Give a big- O estimate for the number of comparisons used by a binary search from the divide-and-conquer recurrence relation you gave in (a) using Theorem 1 in Section 8.3.
 10. a) Give a formula for the number of elements in the union of three sets.
b) Explain why this formula is valid.
c) Explain how to use the formula from (a) to find the number of integers not exceeding 1000 that are divisible by 6, 10, or 15.
d) Explain how to use the formula from (a) to find the number of solutions in nonnegative integers to the equation $x_1 + x_2 + x_3 + x_4 = 22$ with $x_1 < 8$, $x_2 < 6$, and $x_3 < 5$.
 11. a) Give a formula for the number of elements in the union of four sets and explain why it is valid.
b) Suppose the sets A_1 , A_2 , A_3 , and A_4 each contain 25 elements, the intersection of any two of these sets contains 5 elements, the intersection of any three of these sets contains 2 elements, and 1 element is in all four of the sets. How many elements are in the union of the four sets?
 12. a) State the principle of inclusion–exclusion.
b) Outline a proof of this principle.
 13. Explain how the principle of inclusion–exclusion can be used to count the number of onto functions from a set with m elements to a set with n elements.
 14. a) How can you count the number of ways to assign m jobs to n employees so that each employee is assigned at least one job?

- b)** How many ways are there to assign seven jobs to three employees so that each employee is assigned at least one job?

15. Explain how the inclusion–exclusion principle can be used to count the number of primes not exceeding the positive integer n .

16. a) Define a derangement.
b) Why is counting the number of ways a hatcheck person can return hats to n people, so that no one receives the correct hat, the same as counting the number of derangements of n objects?
c) Explain how to count the number of derangements of n objects.

Supplementary Exercises

1. A group of 10 people begin a chain letter, with each person sending the letter to four other people. Each of these people sends the letter to four additional people.

 - Find a recurrence relation for the number of letters sent at the n th stage of this chain letter, if no person ever receives more than one letter.
 - What are the initial conditions for the recurrence relation in part (a)?
 - How many letters are sent at the n th stage of the chain letter?

2. A nuclear reactor has created 18 grams of a particular radioactive isotope. Every hour 1% of this radioactive isotope decays.

 - Set up a recurrence relation for the amount of this isotope left n hours after its creation.
 - What are the initial conditions for the recurrence relation in part (a)?
 - Solve this recurrence relation.

3. Every hour the U.S. government prints 10,000 more \$1 bills, 4000 more \$5 bills, 3000 more \$10 bills, 2500 more \$20 bills, 1000 more \$50 bills, and the same number of \$100 bills as it did the previous hour. In the initial hour 1000 of each bill were produced.

 - Set up a recurrence relation for the amount of money produced in the n th hour.
 - What are the initial conditions for the recurrence relation in part (a)?
 - Solve the recurrence relation for the amount of money produced in the n th hour.
 - Set up a recurrence relation for the total amount of money produced in the first n hours.
 - Solve the recurrence relation for the total amount of money produced in the first n hours.

4. Suppose that every hour there are two new bacteria in a colony for each bacterium that was present the previous hour, and that all bacteria 2 hours old die. The colony starts with 100 new bacteria.

 - Set up a recurrence relation for the number of bacteria present after n hours.
 - What is the solution of this recurrence relation?
 - When will the colony contain more than 1 million bacteria?

5. Messages are sent over a communications channel using two different signals. One signal requires 2 microseconds for transmittal, and the other signal requires 3 microseconds for transmittal. Each signal of a message is followed immediately by the next signal.

 - Find a recurrence relation for the number of different signals that can be sent in n microseconds.
 - What are the initial conditions of the recurrence relation in part (a)?
 - How many different messages can be sent in 12 microseconds?

6. A small post office has only 4-cent stamps, 6-cent stamps, and 10-cent stamps. Find a recurrence relation for the number of ways to form postage of n cents with these stamps if the order that the stamps are used matters. What are the initial conditions for this recurrence relation?

7. How many ways are there to form these postages using the rules described in Exercise 6?

 - 12 cents
 - 14 cents
 - 18 cents
 - 22 cents

8. Find the solutions of the simultaneous system of recurrence relations

$$a_n = a_{n-1} + b_{n-1}$$

$$b_n = a_{n-1} - b_{n-1}$$

with $a_0 = 1$ and $b_0 = 2$.

9. Solve the recurrence relation $a_n = a_{n-1}^2/a_{n-2}$ if $a_0 = 1$ and $a_1 = 2$. [Hint: Take logarithms of both sides to obtain a recurrence relation for the sequence $\log a_n$, $n = 0, 1, 2, \dots$]

*10. Solve the recurrence relation $a_n = a_{n-1}^3 a_{n-2}^2$ if $a_0 = 2$ and $a_1 = 2$. (See the hint for Exercise 9.)

11. Find the solution of the recurrence relation $a_n = 3a_{n-1} - 3a_{n-2} + a_{n-3} + 1$ if $a_0 = 2$, $a_1 = 4$, and $a_2 = 8$.

12. Find the solution of the recurrence relation $a_n = 3a_{n-1} - 3a_{n-2} + a_{n-3}$ if $a_0 = 2$, $a_1 = 2$, and $a_2 = 4$.

*13. Suppose that in Example 1 of Section 8.1 a pair of rabbits leaves the island after reproducing twice. Find a recurrence relation for the number of rabbits on the island in the middle of the n th month.

*14. In this exercise we construct a dynamic programming algorithm for solving the problem of finding a subset S of items chosen from a set of n items where item i has a weight w_i , which is a positive integer, so that the total weight of the items in S is a maximum but does not

exceed a fixed weight limit W . Let $M(j, w)$ denote the maximum total weight of the items in a subset of the first j items such that this total weight does not exceed w . This problem is known as the **knapsack problem**.

- a) Show that if $w_j > w$, then $M(j, w) = M(j - 1, w)$.
- b) Show that if $w_j \leq w$, then $M(j, w) = \max(M(j - 1, w), w_j + M(j - 1, w - w_j))$.
- c) Use (a) and (b) to construct a dynamic programming algorithm for determining the maximum total weight of items so that this total weight does not exceed W . In your algorithm store the values $M(j, w)$ as they are found.
- d) Explain how you can use the values $M(j, w)$ computed by the algorithm in part (c) to find a subset of items with maximum total weight not exceeding W .

In Exercises 15–18 we develop a dynamic programming algorithm for finding a longest common subsequence of two sequences a_1, a_2, \dots, a_m and b_1, b_2, \dots, b_n , an important problem in the comparison of DNA of different organisms.

- 15. Suppose that c_1, c_2, \dots, c_p is a longest common subsequence of the sequences a_1, a_2, \dots, a_m and b_1, b_2, \dots, b_n .
 - a) Show that if $a_m = b_n$, then $c_p = a_m = b_n$ and c_1, c_2, \dots, c_{p-1} is a longest common subsequence of a_1, a_2, \dots, a_{m-1} and b_1, b_2, \dots, b_{n-1} when $p > 1$.
 - b) Suppose that $a_m \neq b_n$. Show that if $c_p \neq a_m$, then c_1, c_2, \dots, c_p is a longest common subsequence of a_1, a_2, \dots, a_{m-1} and b_1, b_2, \dots, b_n and also show that if $c_p \neq b_n$, then c_1, c_2, \dots, c_p is a longest common subsequence of a_1, a_2, \dots, a_m and b_1, b_2, \dots, b_{n-1} .
- 16. Let $L(i, j)$ denote the length of a longest common subsequence of a_1, a_2, \dots, a_i and b_1, b_2, \dots, b_j , where $0 \leq i \leq m$ and $0 \leq j \leq n$. Use parts (a) and (b) of Exercise 15 to show that $L(i, j)$ satisfies the recurrence relation $L(i, j) = L(i - 1, j - 1) + 1$ if both i and j are nonzero and $a_i = b_j$, and $L(i, j) = \max(L(i, j - 1), L(i - 1, j))$ if both i and j are nonzero and $a_i \neq b_j$, and the initial condition $L(i, j) = 0$ if $i = 0$ or $j = 0$.
- 17. Use Exercise 16 to construct a dynamic programming algorithm for computing the length of a longest common subsequence of two sequences a_1, a_2, \dots, a_m and b_1, b_2, \dots, b_n , storing the values of $L(i, j)$ as they are found.
- 18. Develop an algorithm for finding a longest common subsequence of two sequences a_1, a_2, \dots, a_m and b_1, b_2, \dots, b_n using the values $L(i, j)$ found by the algorithm in Exercise 17.
- 19. Find the solution to the recurrence relation $f(n) = f(n/2) + n^2$ for $n = 2^k$ where k is a positive integer and $f(1) = 1$.
- 20. Find the solution to the recurrence relation $f(n) = 3f(n/5) + 2n^4$, when n is divisible by 5, for $n = 5^k$, where k is a positive integer and $f(1) = 1$.
- 21. Give a big- O estimate for the size of f in Exercise 20 if f is an increasing function.

- 22. Find a recurrence relation that describes the number of comparisons used by the following algorithm: Find the largest and second largest elements of a sequence of n numbers recursively by splitting the sequence into two subsequences with an equal number of terms, or where there is one more term in one subsequence than in the other, at each stage. Stop when subsequences with two terms are reached.
- 23. Give a big- O estimate for the number of comparisons used by the algorithm described in Exercise 22.
- 24. A sequence a_1, a_2, \dots, a_n is **unimodal** if and only if there is an index m , $1 \leq m \leq n$, such that $a_i < a_{i+1}$ when $1 \leq i < m$ and $a_i > a_{i+1}$ when $m \leq i < n$. That is, the terms of the sequence strictly increase until the m th term and they strictly decrease after it, which implies that a_m is the largest term. In this exercise, a_m will always denote the largest term of the unimodal sequence a_1, a_2, \dots, a_n .
 - a) Show that a_m is the unique term of the sequence that is greater than both the term immediately preceding it and the term immediately following it.
 - b) Show that if $a_i < a_{i+1}$ where $1 \leq i < n$, then $i + 1 \leq m \leq n$.
 - c) Show that if $a_i > a_{i+1}$ where $1 \leq i < n$, then $1 \leq m \leq i$.
 - d) Develop a divide-and-conquer algorithm for locating the index m . [Hint: Suppose that $i < m < j$. Use parts (a), (b), and (c) to determine whether $\lfloor (i+j)/2 \rfloor + 1 \leq m \leq n$, $1 \leq m \leq \lfloor (i+j)/2 \rfloor - 1$, or $m = \lfloor (i+j)/2 \rfloor$.]
- 25. Show that the algorithm from Exercise 24 has worst-case time complexity $O(\log n)$ in terms of the number of comparisons.
- Let $\{a_n\}$ be a sequence of real numbers. The **forward differences** of this sequence are defined recursively as follows: The **first forward difference** is $\Delta a_n = a_{n+1} - a_n$; the **$(k+1)$ st forward difference** $\Delta^{k+1} a_n$ is obtained from $\Delta^k a_n$ by $\Delta^{k+1} a_n = \Delta^k a_{n+1} - \Delta^k a_n$.
- 26. Find Δa_n , where
 - a) $a_n = 3$.
 - b) $a_n = 4n + 7$.
 - c) $a_n = n^2 + n + 1$.
- 27. Let $a_n = 3n^3 + n + 2$. Find $\Delta^k a_n$, where k equals
 - a) 2.
 - b) 3.
 - c) 4.
- *28. Suppose that $a_n = P(n)$, where P is a polynomial of degree d . Prove that $\Delta^{d+1} a_n = 0$ for all nonnegative integers n .
- 29. Let $\{a_n\}$ and $\{b_n\}$ be sequences of real numbers. Show that

$$\Delta(a_n b_n) = a_{n+1}(\Delta b_n) + b_n(\Delta a_n).$$
- 30. Show that if $F(x)$ and $G(x)$ are the generating functions for the sequences $\{a_k\}$ and $\{b_k\}$, respectively, and c and d are real numbers, then $(cF + dG)(x)$ is the generating function for $\{ca_k + db_k\}$.

- 31.** (*Requires calculus*) This exercise shows how generating functions can be used to solve the recurrence relation $(n+1)a_{n+1} = a_n + (1/n!)$ for $n \geq 0$ with initial condition $a_0 = 1$.
- Let $G(x)$ be the generating function for $\{a_n\}$. Show that $G'(x) = G(x) + e^x$ and $G(0) = 1$.
 - Show from part (a) that $(e^{-x}G(x))' = 1$, and conclude that $G(x) = xe^x + e^x$.
 - Use part (b) to find a closed form for a_n .
- 32.** Suppose that 14 students receive an A on the first exam in a discrete mathematics class, and 18 receive an A on the second exam. If 22 students received an A on either the first exam or the second exam, how many students received an A on both exams?
- 33.** There are 323 farms in Monmouth County that have at least one of horses, cows, and sheep. If 224 have horses, 85 have cows, 57 have sheep, and 18 farms have all three types of animals, how many farms have exactly two of these three types of animals?
- 34.** Queries to a database of student records at a college produced the following data: There are 2175 students at the college, 1675 of these are not freshmen, 1074 students have taken a course in calculus, 444 students have taken a course in discrete mathematics, 607 students are not freshmen and have taken calculus, 350 students have taken calculus and discrete mathematics, 201 students are not freshmen and have taken discrete mathematics, and 143 students are not freshmen and have taken both calculus and discrete mathematics. Can all the responses to the queries be correct?
- 35.** Students in the school of mathematics at a university major in one or more of the following four areas: applied mathematics (AM), pure mathematics (PM), operations research (OR), and computer science (CS). How many students are in this school if (including joint majors) there are 23 students majoring in AM; 17 in PM; 44 in OR; 63 in CS; 5 in AM and PM; 8 in AM and CS; 4 in AM and OR; 6 in PM and CS; 5 in PM and OR; 14 in OR and CS; 2 in PM, OR, and CS; 2 in AM, OR, and CS; 1 in PM, AM, and OR; 1 in PM, AM, and CS; and 1 in all four fields.
- 36.** How many terms are needed when the inclusion-exclusion principle is used to express the number of elements in the union of seven sets if no more than five of these sets have a common element?
- 37.** How many solutions in positive integers are there to the equation $x_1 + x_2 + x_3 = 20$ with $2 < x_1 < 6$, $6 < x_2 < 10$, and $0 < x_3 < 5$?
- 38.** How many positive integers less than 1,000,000 are
- divisible by 2, 3, or 5?
 - not divisible by 7, 11, or 13?
 - divisible by 3 but not by 7?
- 39.** How many positive integers less than 200 are
- second or higher powers of integers?
 - either primes or second or higher powers of integers?
 - not divisible by the square of an integer greater than 1?
 - not divisible by the cube of an integer greater than 1?
 - not divisible by three or more primes?
- *40.** How many ways are there to assign six different jobs to three different employees if the hardest job is assigned to the most experienced employee and the easiest job is assigned to the least experienced employee?
- 41.** What is the probability that exactly one person is given back the correct hat by a hatcheck person who gives n people their hats back at random?
- 42.** How many bit strings of length six do not contain four consecutive 1s?
- 43.** What is the probability that a bit string of length six chosen at random contains at least four 1s?

Computer Projects

Write programs with these input and output.

- Given a positive integer n , list all the moves required in the Tower of Hanoi puzzle to move n disks from one peg to another according to the rules of the puzzle.
- Given a positive integer n and an integer k with $1 \leq k \leq n$, list all the moves used by the Frame-Stewart algorithm (described in the preamble to Exercise 38 of Section 8.1) to move n disks from one peg to another using four pegs according to the rules of the puzzle.
- Given a positive integer n , list all the bit sequences of length n that do not have a pair of consecutive 0s.
- Given an integer n greater than 1, write out all ways to parenthesize the product of $n+1$ variables.
- Given a set of n talks, their start and end times, and the number of attendees at each talk, use dynamic program-
- ming to schedule a subset of these talks in a single lecture hall to maximize total attendance.
- Given matrices $\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_n$, with dimensions $m_1 \times m_2, m_2 \times m_3, \dots, m_n \times m_{n+1}$, respectively, each with integer entries, use dynamic programming, as outlined in Exercise 57 in Section 8.1, to find the minimum number of multiplications of integers needed to compute $\mathbf{A}_1 \mathbf{A}_2 \cdots \mathbf{A}_n$.
- Given a recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$, where c_1 and c_2 are real numbers, initial conditions $a_0 = C_0$ and $a_1 = C_1$, and a positive integer k , find a_k using iteration.
- Given a recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ and initial conditions $a_0 = C_0$ and $a_1 = C_1$, determine the unique solution.

9. Given a recurrence relation of the form $f(n) = af(n/b) + c$, where a is a real number, b is a positive integer, and c is a real number, and a positive integer k , find $f(b^k)$ using iteration.
10. Given the number of elements in the intersection of three sets, the number of elements in each pairwise intersection of these sets, and the number of elements in each set, find the number of elements in their union.
11. Given a positive integer n , produce the formula for the number of elements in the union of n sets.
12. Given positive integers m and n , find the number of onto functions from a set with m elements to a set with n elements.
13. Given a positive integer n , list all the derangements of the set $\{1, 2, 3, \dots, n\}$.

Computations and Explorations

Use a computational program or programs you have written to do these exercises.

1. Find the exact value of f_{100} , f_{500} , and f_{1000} , where f_n is the n th Fibonacci number.
2. Find the smallest Fibonacci number greater than 1,000,000, greater than 1,000,000,000, and greater than 1,000,000,000,000.
3. Find as many prime Fibonacci numbers as you can. It is unknown whether there are infinitely many of these.
4. Write out all the moves required to solve the Tower of Hanoi puzzle with 10 disks.
5. Write out all the moves required to use the Frame–Stewart algorithm to move 20 disks from one peg to another peg using four pegs according to the rules of the Reve’s puzzle.
6. Verify the Frame conjecture for solving the Reve’s puzzle for n disks for as many integers n as possible by showing that the puzzle cannot be solved using fewer moves than are made by the Frame–Stewart algorithm with the optimal choice of k .
7. Compute the number of operations required to multiply two integers with n bits for various integers n including 16, 64, 256, and 1024 using the fast multiplication described in Example 4 of Section 8.3 and the standard algorithm for multiplying integers (Algorithm 3 in Section 4.2).
8. Compute the number of operations required to multiply two $n \times n$ matrices for various integers n including 4, 16, 64, and 128 using the fast matrix multiplication described in Example 5 of Section 8.3 and the standard algorithm for multiplying matrices (Algorithm 1 in Section 3.3).
9. Find the number of primes not exceeding 10,000 using the method described in Section 8.6 to find the number of primes not exceeding 100.
10. List all the derangements of $\{1, 2, 3, 4, 5, 6, 7, 8\}$.
11. Compute the probability that a permutation of n objects is a derangement for all positive integers not exceeding 20 and determine how quickly these probabilities approach the number $1/e$.

Writing Projects

Respond to these with essays using outside sources.

1. Find the original source where Fibonacci presented his puzzle about modeling rabbit populations. Discuss this problem and other problems posed by Fibonacci and give some information about Fibonacci himself.
2. Explain how the Fibonacci numbers arise in a variety of applications, such as in phyllotaxis, the study of arrangement of leaves in plants, in the study of reflections by mirrors, and so on.
3. Describe different variations of the Tower of Hanoi puzzle, including those with more than three pegs (including the Reve’s puzzle discussed in the text and exercises), those where disk moves are restricted, and those where disks may have the same size. Include what is known about the number of moves required to solve each variation.
4. Discuss as many different problems as possible where the Catalan numbers arise.
5. Discuss some of the problems in which Richard Bellman first used dynamic programming.
6. Describe the role dynamic programming algorithms play in bioinformatics including for DNA sequence comparison, gene comparison, and RNA structure prediction.
7. Describe the use of dynamic programming in economics including its use to study optimal consumption and saving.
8. Explain how dynamic programming can be used to solve the egg-dropping puzzle which determines from which floors of a multistory building it is safe to drop eggs from without breaking.

9. Describe the solution of Ulam's problem (see Exercise 28 in Section 8.3) involving searching with one lie found by Andrzej Pelc.
10. Discuss variations of Ulam's problem (see Exercise 28 in Section 8.3) involving searching with more than one lie and what is known about this problem.
11. Define the convex hull of a set of points in the plane and describe three different algorithms, including a divide-and-conquer algorithm, for finding the convex hull of a set of points in the plane.
12. Describe how sieve methods are used in number theory. What kind of results have been established using such methods?
13. Look up the rules of the old French card game of *rencontres*. Describe these rules and describe the work of Pierre Raymond de Montmort on *le problème de rencontres*.
14. Describe how exponential generating functions can be used to solve a variety of counting problems.
15. Describe the Polyá theory of counting and the kind of counting problems that can be solved using this theory.
16. The *problème des ménages* (the problem of the households) asks for the number of ways to arrange n couples around a table so that the sexes alternate and no husband and wife are seated together. Explain the method used by E. Lucas to solve this problem.
17. Explain how *rook polynomials* can be used to solve counting problems.

9

Relations

- 9.1** Relations and Their Properties
- 9.2** n -ary Relations and Their Applications
- 9.3** Representing Relations
- 9.4** Closures of Relations
- 9.5** Equivalence Relations
- 9.6** Partial Orderings

Relationships between elements of sets occur in many contexts. Every day we deal with relationships such as those between a business and its telephone number, an employee and his or her salary, a person and a relative, and so on. In mathematics we study relationships such as those between a positive integer and one that it divides, an integer and one that it is congruent to modulo 5, a real number and one that is larger than it, a real number x and the value $f(x)$ where f is a function, and so on. Relationships such as that between a program and a variable it uses, and that between a computer language and a valid statement in this language often arise in computer science.

Relationships between elements of sets are represented using the structure called a relation, which is just a subset of the Cartesian product of the sets. Relations can be used to solve problems such as determining which pairs of cities are linked by airline flights in a network, finding a viable order for the different phases of a complicated project, or producing a useful way to store information in computer databases.

In some computer languages, only the first 31 characters of the name of a variable matter. The relation consisting of ordered pairs of strings where the first string has the same initial 31 characters as the second string is an example of a special type of relation, known as an equivalence relation. Equivalence relations arise throughout mathematics and computer science. We will study equivalence relations, and other special types of relations, in this chapter.

9.1 Relations and Their Properties

Introduction



The most direct way to express a relationship between elements of two sets is to use ordered pairs made up of two related elements. For this reason, sets of ordered pairs are called binary relations. In this section we introduce the basic terminology used to describe binary relations. Later in this chapter we will use relations to solve problems involving communications networks, project scheduling, and identifying elements in sets with common properties.

DEFINITION 1

Let A and B be sets. A *binary relation* from A to B is a subset of $A \times B$.

In other words, a binary relation from A to B is a set R of ordered pairs where the first element of each ordered pair comes from A and the second element comes from B . We use the notation $a R b$ to denote that $(a, b) \in R$ and $a \not R b$ to denote that $(a, b) \notin R$. Moreover, when (a, b) belongs to R , a is said to be **related to** b by R .

Binary relations represent relationships between the elements of two sets. We will introduce n -ary relations, which express relationships among elements of more than two sets, later in this chapter. We will omit the word *binary* when there is no danger of confusion.

Examples 1–3 illustrate the notion of a relation.

EXAMPLE 1

Let A be the set of students in your school, and let B be the set of courses. Let R be the relation that consists of those pairs (a, b) , where a is a student enrolled in course b . For instance, if Jason Goodfriend and Deborah Sherman are enrolled in CS518, the pairs

(Jason Goodfriend, CS518) and (Deborah Sherman, CS518) belong to R . If Jason Goodfriend is also enrolled in CS510, then the pair (Jason Goodfriend, CS510) is also in R . However, if Deborah Sherman is not enrolled in CS510, then the pair (Deborah Sherman, CS510) is not in R .

Note that if a student is not currently enrolled in any courses there will be no pairs in R that have this student as the first element. Similarly, if a course is not currently being offered there will be no pairs in R that have this course as their second element. 

EXAMPLE 2

Let A be the set of cities in the U.S.A., and let B be the set of the 50 states in the U.S.A. Define the relation R by specifying that (a, b) belongs to R if a city with name a is in the state b . For instance, (Boulder, Colorado), (Bangor, Maine), (Ann Arbor, Michigan), (Middletown, New Jersey), (Middletown, New York), (Cupertino, California), and (Red Bank, New Jersey) are in R . 

EXAMPLE 3

Let $A = \{0, 1, 2\}$ and $B = \{a, b\}$. Then $\{(0, a), (0, b), (1, a), (2, b)\}$ is a relation from A to B . This means, for instance, that $0 R a$, but that $1 \not R b$. Relations can be represented graphically, as shown in Figure 1, using arrows to represent ordered pairs. Another way to represent this relation is to use a table, which is also done in Figure 1. We will discuss representations of relations in more detail in Section 9.3. 

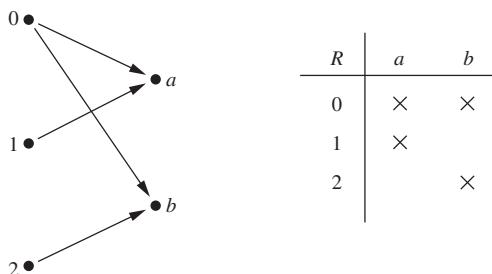


FIGURE 1 Displaying the Ordered Pairs in the Relation R from Example 3.

Functions as Relations

Recall that a function f from a set A to a set B (as defined in Section 2.3) assigns exactly one element of B to each element of A . The graph of f is the set of ordered pairs (a, b) such that $b = f(a)$. Because the graph of f is a subset of $A \times B$, it is a relation from A to B . Moreover, the graph of a function has the property that every element of A is the first element of exactly one ordered pair of the graph.

Conversely, if R is a relation from A to B such that every element in A is the first element of exactly one ordered pair of R , then a function can be defined with R as its graph. This can be done by assigning to an element a of A the unique element $b \in B$ such that $(a, b) \in R$. (Note that the relation R in Example 2 is not the graph of a function because Middletown occurs more than once as the first element of an ordered pair in R .)

A relation can be used to express a one-to-many relationship between the elements of the sets A and B (as in Example 2), where an element of A may be related to more than one element of B . A function represents a relation where exactly one element of B is related to each element of A .

Relations are a generalization of graphs of functions; they can be used to express a much wider class of relationships between sets. (Recall that the graph of the function f from A to B is the set of ordered pairs $(a, f(a))$ for $a \in A$.)

Relations on a Set

Relations from a set A to itself are of special interest.

DEFINITION 2

A *relation on a set A* is a relation from A to A .

In other words, a relation on a set A is a subset of $A \times A$.

EXAMPLE 4 Let A be the set $\{1, 2, 3, 4\}$. Which ordered pairs are in the relation $R = \{(a, b) \mid a \text{ divides } b\}$?

Solution: Because (a, b) is in R if and only if a and b are positive integers not exceeding 4 such that a divides b , we see that

$$R = \{(1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 4), (3, 3), (4, 4)\}.$$

The pairs in this relation are displayed both graphically and in tabular form in Figure 2. 

Next, some examples of relations on the set of integers will be given in Example 5.

EXAMPLE 5 Consider these relations on the set of integers:

$$\begin{aligned} R_1 &= \{(a, b) \mid a \leq b\}, \\ R_2 &= \{(a, b) \mid a > b\}, \\ R_3 &= \{(a, b) \mid a = b \text{ or } a = -b\}, \\ R_4 &= \{(a, b) \mid a = b\}, \\ R_5 &= \{(a, b) \mid a = b + 1\}, \\ R_6 &= \{(a, b) \mid a + b \leq 3\}. \end{aligned}$$

Which of these relations contain each of the pairs $(1, 1)$, $(1, 2)$, $(2, 1)$, $(1, -1)$, and $(2, 2)$?

Remark: Unlike the relations in Examples 1–4, these are relations on an infinite set.

Solution: The pair $(1, 1)$ is in R_1 , R_3 , R_4 , and R_6 ; $(1, 2)$ is in R_1 and R_6 ; $(2, 1)$ is in R_2 , R_5 , and R_6 ; $(1, -1)$ is in R_2 , R_3 , and R_6 ; and finally, $(2, 2)$ is in R_1 , R_3 , and R_4 . 

It is not hard to determine the number of relations on a finite set, because a relation on a set A is simply a subset of $A \times A$.

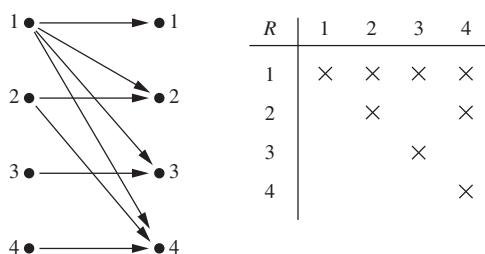


FIGURE 2 Displaying the Ordered Pairs in the Relation R from Example 4.

EXAMPLE 6 How many relations are there on a set with n elements?

Solution: A relation on a set A is a subset of $A \times A$. Because $A \times A$ has n^2 elements when A has n elements, and a set with m elements has 2^m subsets, there are 2^{n^2} subsets of $A \times A$. Thus, there are 2^{n^2} relations on a set with n elements. For example, there are $2^{3^2} = 2^9 = 512$ relations on the set $\{a, b, c\}$.

Properties of Relations

There are several properties that are used to classify relations on a set. We will introduce the most important of these here.

In some relations an element is always related to itself. For instance, let R be the relation on the set of all people consisting of pairs (x, y) where x and y have the same mother and the same father. Then xRx for every person x .

DEFINITION 3

A relation R on a set A is called *reflexive* if $(a, a) \in R$ for every element $a \in A$.

Remark: Using quantifiers we see that the relation R on the set A is reflexive if $\forall a((a, a) \in R)$, where the universe of discourse is the set of all elements in A .

We see that a relation on A is reflexive if every element of A is related to itself. Examples 7–9 illustrate the concept of a reflexive relation.

EXAMPLE 7 Consider the following relations on $\{1, 2, 3, 4\}$:

$$\begin{aligned} R_1 &= \{(1, 1), (1, 2), (2, 1), (2, 2), (3, 4), (4, 1), (4, 4)\}, \\ R_2 &= \{(1, 1), (1, 2), (2, 1)\}, \\ R_3 &= \{(1, 1), (1, 2), (1, 4), (2, 1), (2, 2), (3, 3), (4, 1), (4, 4)\}, \\ R_4 &= \{(2, 1), (3, 1), (3, 2), (4, 1), (4, 2), (4, 3)\}, \\ R_5 &= \{(1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 3), (2, 4), (3, 3), (3, 4), (4, 4)\}, \\ R_6 &= \{(3, 4)\}. \end{aligned}$$

Which of these relations are reflexive?

Solution: The relations R_3 and R_5 are reflexive because they both contain all pairs of the form (a, a) , namely, $(1, 1)$, $(2, 2)$, $(3, 3)$, and $(4, 4)$. The other relations are not reflexive because they do not contain all of these ordered pairs. In particular, R_1 , R_2 , R_4 , and R_6 are not reflexive because $(3, 3)$ is not in any of these relations.

EXAMPLE 8 Which of the relations from Example 5 are reflexive?

Solution: The reflexive relations from Example 5 are R_1 (because $a \leq a$ for every integer a), R_3 , and R_4 . For each of the other relations in this example it is easy to find a pair of the form (a, a) that is not in the relation. (This is left as an exercise for the reader.)

EXAMPLE 9 Is the “divides” relation on the set of positive integers reflexive?

Solution: Because $a | a$ whenever a is a positive integer, the “divides” relation is reflexive. (Note that if we replace the set of positive integers with the set of all integers the relation is not reflexive because by definition 0 does not divide 0.)

In some relations an element is related to a second element if and only if the second element is also related to the first element. The relation consisting of pairs (x, y) , where x and y are students at your school with at least one common class has this property. Other relations have the property that if an element is related to a second element, then this second element is not related to the first. The relation consisting of the pairs (x, y) , where x and y are students at your school, where x has a higher grade point average than y has this property.

DEFINITION 4

A relation R on a set A is called *symmetric* if $(b, a) \in R$ whenever $(a, b) \in R$, for all $a, b \in A$. A relation R on a set A such that for all $a, b \in A$, if $(a, b) \in R$ and $(b, a) \in R$, then $a = b$ is called *antisymmetric*.

Remark: Using quantifiers, we see that the relation R on the set A is symmetric if $\forall a \forall b ((a, b) \in R \rightarrow (b, a) \in R)$. Similarly, the relation R on the set A is antisymmetric if $\forall a \forall b (((a, b) \in R \wedge (b, a) \in R) \rightarrow (a = b))$.

That is, a relation is symmetric if and only if a is related to b implies that b is related to a . A relation is antisymmetric if and only if there are no pairs of distinct elements a and b with a related to b and b related to a . That is, the only way to have a related to b and b related to a is for a and b to be the same element. The terms *symmetric* and *antisymmetric* are not opposites, because a relation can have both of these properties or may lack both of them (see Exercise 10). A relation cannot be both symmetric and antisymmetric if it contains some pair of the form (a, b) , where $a \neq b$.

Remark: Although relatively few of the 2^n^2 relations on a set with n elements are symmetric or antisymmetric, as counting arguments can show, many important relations have one of these properties. (See Exercise 47.)

EXAMPLE 10

Which of the relations from Example 7 are symmetric and which are antisymmetric?



Solution: The relations R_2 and R_3 are symmetric, because in each case (b, a) belongs to the relation whenever (a, b) does. For R_2 , the only thing to check is that both $(2, 1)$ and $(1, 2)$ are in the relation. For R_3 , it is necessary to check that both $(1, 2)$ and $(2, 1)$ belong to the relation, and $(1, 4)$ and $(4, 1)$ belong to the relation. The reader should verify that none of the other relations is symmetric. This is done by finding a pair (a, b) such that it is in the relation but (b, a) is not.

R_4 , R_5 , and R_6 are all antisymmetric. For each of these relations there is no pair of elements a and b with $a \neq b$ such that both (a, b) and (b, a) belong to the relation. The reader should verify that none of the other relations is antisymmetric. This is done by finding a pair (a, b) with $a \neq b$ such that (a, b) and (b, a) are both in the relation.

EXAMPLE 11

Which of the relations from Example 5 are symmetric and which are antisymmetric?

Solution: The relations R_3 , R_4 , and R_6 are symmetric. R_3 is symmetric, for if $a = b$ or $a = -b$, then $b = a$ or $b = -a$. R_4 is symmetric because $a = b$ implies that $b = a$. R_6 is symmetric because $a + b \leq 3$ implies that $b + a \leq 3$. The reader should verify that none of the other relations is symmetric.

The relations R_1 , R_2 , R_4 , and R_5 are antisymmetric. R_1 is antisymmetric because the inequalities $a \leq b$ and $b \leq a$ imply that $a = b$. R_2 is antisymmetric because it is impossible that $a > b$ and $b > a$. R_4 is antisymmetric, because two elements are related with respect to R_4 if and only if they are equal. R_5 is antisymmetric because it is impossible that $a = b + 1$ and $b = a + 1$. The reader should verify that none of the other relations is antisymmetric.

EXAMPLE 12 Is the “divides” relation on the set of positive integers symmetric? Is it antisymmetric?

Solution: This relation is not symmetric because $1 \mid 2$, but $2 \nmid 1$. It is antisymmetric, for if a and b are positive integers with $a \mid b$ and $b \mid a$, then $a = b$ (the verification of this is left as an exercise for the reader). 

Let R be the relation consisting of all pairs (x, y) of students at your school, where x has taken more credits than y . Suppose that x is related to y and y is related to z . This means that x has taken more credits than y and y has taken more credits than z . We can conclude that x has taken more credits than z , so that x is related to z . What we have shown is that R has the transitive property, which is defined as follows.

DEFINITION 5

A relation R on a set A is called *transitive* if whenever $(a, b) \in R$ and $(b, c) \in R$, then $(a, c) \in R$, for all $a, b, c \in A$.

Remark: Using quantifiers we see that the relation R on a set A is transitive if we have $\forall a \forall b \forall c ((a, b) \in R \wedge (b, c) \in R) \rightarrow (a, c) \in R$.

EXAMPLE 13 Which of the relations in Example 7 are transitive?



Solution: R_4 , R_5 , and R_6 are transitive. For each of these relations, we can show that it is transitive by verifying that if (a, b) and (b, c) belong to this relation, then (a, c) also does. For instance, R_4 is transitive, because $(3, 2)$ and $(2, 1)$, $(4, 2)$ and $(2, 1)$, $(4, 3)$ and $(3, 1)$, and $(4, 3)$ and $(3, 2)$ are the only such sets of pairs, and $(3, 1)$, $(4, 1)$, and $(4, 2)$ belong to R_4 . The reader should verify that R_5 and R_6 are transitive.

R_1 is not transitive because $(3, 4)$ and $(4, 1)$ belong to R_1 , but $(3, 1)$ does not. R_2 is not transitive because $(2, 1)$ and $(1, 2)$ belong to R_2 , but $(2, 2)$ does not. R_3 is not transitive because $(4, 1)$ and $(1, 2)$ belong to R_3 , but $(4, 2)$ does not. 

EXAMPLE 14 Which of the relations in Example 5 are transitive?

Solution: The relations R_1 , R_2 , R_3 , and R_4 are transitive. R_1 is transitive because $a \leq b$ and $b \leq c$ imply that $a \leq c$. R_2 is transitive because $a > b$ and $b > c$ imply that $a > c$. R_3 is transitive because $a = \pm b$ and $b = \pm c$ imply that $a = \pm c$. R_4 is clearly transitive, as the reader should verify. R_5 is not transitive because $(2, 1)$ and $(1, 0)$ belong to R_5 , but $(2, 0)$ does not. R_6 is not transitive because $(2, 1)$ and $(1, 2)$ belong to R_6 , but $(2, 2)$ does not. 

EXAMPLE 15 Is the “divides” relation on the set of positive integers transitive?

Solution: Suppose that a divides b and b divides c . Then there are positive integers k and l such that $b = ak$ and $c = bl$. Hence, $c = a(kl)$, so a divides c . It follows that this relation is transitive. 

We can use counting techniques to determine the number of relations with specific properties. Finding the number of relations with a particular property provides information about how common this property is in the set of all relations on a set with n elements.

EXAMPLE 16 How many reflexive relations are there on a set with n elements?

Solution: A relation R on a set A is a subset of $A \times A$. Consequently, a relation is determined by specifying whether each of the n^2 ordered pairs in $A \times A$ is in R . However, if R is reflexive, each of the n ordered pairs (a, a) for $a \in A$ must be in R . Each of the other $n(n - 1)$ ordered

pairs of the form (a, b) , where $a \neq b$, may or may not be in R . Hence, by the product rule for counting, there are $2^{n(n-1)}$ reflexive relations [this is the number of ways to choose whether each element (a, b) , with $a \neq b$, belongs to R]. 

Formulas for the number of symmetric relations and the number of antisymmetric relations on a set with n elements can be found using reasoning similar to that in Example 16 (see Exercise 47). However, no general formula is known that counts the transitive relations on a set with n elements. Currently, $T(n)$, the number of transitive relations on a set with n elements, is known only for $n \leq 17$. For example, $T(4) = 3,994$, $T(5) = 154,303$, and $T(6) = 9,415,189$.

Combining Relations

Because relations from A to B are subsets of $A \times B$, two relations from A to B can be combined in any way two sets can be combined. Consider Examples 17–19.

EXAMPLE 17 Let $A = \{1, 2, 3\}$ and $B = \{1, 2, 3, 4\}$. The relations $R_1 = \{(1, 1), (2, 2), (3, 3)\}$ and $R_2 = \{(1, 1), (1, 2), (1, 3), (1, 4)\}$ can be combined to obtain

$$\begin{aligned} R_1 \cup R_2 &= \{(1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (3, 3)\}, \\ R_1 \cap R_2 &= \{(1, 1)\}, \\ R_1 - R_2 &= \{(2, 2), (3, 3)\}, \\ R_2 - R_1 &= \{(1, 2), (1, 3), (1, 4)\}. \end{aligned} \quad \blacktriangleleft$$

EXAMPLE 18 Let A and B be the set of all students and the set of all courses at a school, respectively. Suppose that R_1 consists of all ordered pairs (a, b) , where a is a student who has taken course b , and R_2 consists of all ordered pairs (a, b) , where a is a student who requires course b to graduate. What are the relations $R_1 \cup R_2$, $R_1 \cap R_2$, $R_1 \oplus R_2$, $R_1 - R_2$, and $R_2 - R_1$?

Solution: The relation $R_1 \cup R_2$ consists of all ordered pairs (a, b) , where a is a student who either has taken course b or needs course b to graduate, and $R_1 \cap R_2$ is the set of all ordered pairs (a, b) , where a is a student who has taken course b and needs this course to graduate. Also, $R_1 \oplus R_2$ consists of all ordered pairs (a, b) , where student a has taken course b but does not need it to graduate or needs course b to graduate but has not taken it. $R_1 - R_2$ is the set of ordered pairs (a, b) , where a has taken course b but does not need it to graduate; that is, b is an elective course that a has taken. $R_2 - R_1$ is the set of all ordered pairs (a, b) , where b is a course that a needs to graduate but has not taken. 

EXAMPLE 19 Let R_1 be the “less than” relation on the set of real numbers and let R_2 be the “greater than” relation on the set of real numbers, that is, $R_1 = \{(x, y) \mid x < y\}$ and $R_2 = \{(x, y) \mid x > y\}$. What are $R_1 \cup R_2$, $R_1 \cap R_2$, $R_1 - R_2$, $R_2 - R_1$, and $R_1 \oplus R_2$?

Solution: We note that $(x, y) \in R_1 \cup R_2$ if and only if $(x, y) \in R_1$ or $(x, y) \in R_2$. Hence, $(x, y) \in R_1 \cup R_2$ if and only if $x < y$ or $x > y$. Because the condition $x < y$ or $x > y$ is the same as the condition $x \neq y$, it follows that $R_1 \cup R_2 = \{(x, y) \mid x \neq y\}$. In other words, the union of the “less than” relation and the “greater than” relation is the “not equals” relation.

Next, note that it is impossible for a pair (x, y) to belong to both R_1 and R_2 because it is impossible that $x < y$ and $x > y$. It follows that $R_1 \cap R_2 = \emptyset$. We also see that $R_1 - R_2 = R_1$, $R_2 - R_1 = R_2$, and $R_1 \oplus R_2 = R_1 \cup R_2 - R_1 \cap R_2 = \{(x, y) \mid x \neq y\}$. 

There is another way that relations are combined that is analogous to the composition of functions.

DEFINITION 6

Let R be a relation from a set A to a set B and S a relation from B to a set C . The *composite* of R and S is the relation consisting of ordered pairs (a, c) , where $a \in A$, $c \in C$, and for which there exists an element $b \in B$ such that $(a, b) \in R$ and $(b, c) \in S$. We denote the composite of R and S by $S \circ R$.

Computing the composite of two relations requires that we find elements that are the second element of ordered pairs in the first relation and the first element of ordered pairs in the second relation, as Examples 20 and 21 illustrate.

EXAMPLE 20 What is the composite of the relations R and S , where R is the relation from $\{1, 2, 3\}$ to $\{1, 2, 3, 4\}$ with $R = \{(1, 1), (1, 4), (2, 3), (3, 1), (3, 4)\}$ and S is the relation from $\{1, 2, 3, 4\}$ to $\{0, 1, 2\}$ with $S = \{(1, 0), (2, 0), (3, 1), (3, 2), (4, 1)\}$?

Solution: $S \circ R$ is constructed using all ordered pairs in R and ordered pairs in S , where the second element of the ordered pair in R agrees with the first element of the ordered pair in S . For example, the ordered pairs $(2, 3)$ in R and $(3, 1)$ in S produce the ordered pair $(2, 1)$ in $S \circ R$. Computing all the ordered pairs in the composite, we find

$$S \circ R = \{(1, 0), (1, 1), (2, 1), (2, 2), (3, 0), (3, 1)\}.$$

EXAMPLE 21

Composing the Parent Relation with Itself Let R be the relation on the set of all people such that $(a, b) \in R$ if person a is a parent of person b . Then $(a, c) \in R \circ R$ if and only if there is a person b such that $(a, b) \in R$ and $(b, c) \in R$, that is, if and only if there is a person b such that a is a parent of b and b is a parent of c . In other words, $(a, c) \in R \circ R$ if and only if a is a grandparent of c .

The powers of a relation R can be recursively defined from the definition of a composite of two relations.

DEFINITION 7

Let R be a relation on the set A . The powers R^n , $n = 1, 2, 3, \dots$, are defined recursively by

$$R^1 = R \quad \text{and} \quad R^{n+1} = R^n \circ R.$$

The definition shows that $R^2 = R \circ R$, $R^3 = R^2 \circ R = (R \circ R) \circ R$, and so on.

EXAMPLE 22 Let $R = \{(1, 1), (2, 1), (3, 2), (4, 3)\}$. Find the powers R^n , $n = 2, 3, 4, \dots$.

Solution: Because $R^2 = R \circ R$, we find that $R^2 = \{(1, 1), (2, 1), (3, 1), (4, 2)\}$. Furthermore, because $R^3 = R^2 \circ R$, $R^3 = \{(1, 1), (2, 1), (3, 1), (4, 1)\}$. Additional computation shows that R^4 is the same as R^3 , so $R^4 = \{(1, 1), (2, 1), (3, 1), (4, 1)\}$. It also follows that $R^n = R^3$ for $n = 5, 6, 7, \dots$. The reader should verify this.

The following theorem shows that the powers of a transitive relation are subsets of this relation. It will be used in Section 9.4.

THEOREM 1

The relation R on a set A is transitive if and only if $R^n \subseteq R$ for $n = 1, 2, 3, \dots$

Proof: We first prove the “if” part of the theorem. We suppose that $R^n \subseteq R$ for $n = 1, 2, 3, \dots$. In particular, $R^2 \subseteq R$. To see that this implies R is transitive, note that if $(a, b) \in R$ and $(b, c) \in R$, then by the definition of composition, $(a, c) \in R^2$. Because $R^2 \subseteq R$, this means that $(a, c) \in R$. Hence, R is transitive.



We will use mathematical induction to prove the only if part of the theorem. Note that this part of the theorem is trivially true for $n = 1$.

Assume that $R^n \subseteq R$, where n is a positive integer. This is the inductive hypothesis. To complete the inductive step we must show that this implies that R^{n+1} is also a subset of R . To show this, assume that $(a, b) \in R^{n+1}$. Then, because $R^{n+1} = R^n \circ R$, there is an element x with $x \in A$ such that $(a, x) \in R$ and $(x, b) \in R^n$. The inductive hypothesis, namely, that $R^n \subseteq R$, implies that $(x, b) \in R$. Furthermore, because R is transitive, and $(a, x) \in R$ and $(x, b) \in R$, it follows that $(a, b) \in R$. This shows that $R^{n+1} \subseteq R$, completing the proof. \triangleleft

Exercises

1. List the ordered pairs in the relation R from $A = \{0, 1, 2, 3, 4\}$ to $B = \{0, 1, 2, 3\}$, where $(a, b) \in R$ if and only if
 - a) $a = b$.
 - b) $a + b = 4$.
 - c) $a > b$.
 - d) $a \mid b$.
 - e) $\gcd(a, b) = 1$.
 - f) $\text{lcm}(a, b) = 2$.
 2. a) List all the ordered pairs in the relation $R = \{(a, b) \mid a \text{ divides } b\}$ on the set $\{1, 2, 3, 4, 5, 6\}$.
b) Display this relation graphically, as was done in Example 4.
c) Display this relation in tabular form, as was done in Example 4.
 3. For each of these relations on the set $\{1, 2, 3, 4\}$, decide whether it is reflexive, whether it is symmetric, whether it is antisymmetric, and whether it is transitive.
 - a) $\{(2, 2), (2, 3), (2, 4), (3, 2), (3, 3), (3, 4)\}$
 - b) $\{(1, 1), (1, 2), (2, 1), (2, 2), (3, 3), (4, 4)\}$
 - c) $\{(2, 4), (4, 2)\}$
 - d) $\{(1, 2), (2, 3), (3, 4)\}$
 - e) $\{(1, 1), (2, 2), (3, 3), (4, 4)\}$
 - f) $\{(1, 3), (1, 4), (2, 3), (2, 4), (3, 1), (3, 4)\}$
 4. Determine whether the relation R on the set of all people is reflexive, symmetric, antisymmetric, and/or transitive, where $(a, b) \in R$ if and only if
 - a) a is taller than b .
 - b) a and b were born on the same day.
 - c) a has the same first name as b .
 - d) a and b have a common grandparent.
 5. Determine whether the relation R on the set of all Web pages is reflexive, symmetric, antisymmetric, and/or transitive, where $(a, b) \in R$ if and only if
 - a) everyone who has visited Web page a has also visited Web page b .
 - b) there are no common links found on both Web page a and Web page b .
 - c) there is at least one common link on Web page a and Web page b .
 6. Determine whether the relation R on the set of all real numbers is reflexive, symmetric, antisymmetric, and/or transitive, where $(x, y) \in R$ if and only if
 - a) $x + y = 0$.
 - b) $x = \pm y$.
 - c) $x - y$ is a rational number.
 - d) $x = 2y$.
 - e) $xy \geq 0$.
 - f) $xy = 0$.
 - g) $x = 1$.
 - h) $x = 1$ or $y = 1$.
 7. Determine whether the relation R on the set of all integers is reflexive, symmetric, antisymmetric, and/or transitive, where $(x, y) \in R$ if and only if
 - a) $x \neq y$.
 - b) $xy \geq 1$.
 - c) $x = y + 1$ or $x = y - 1$.
 - d) $x \equiv y \pmod{7}$.
 - e) x is a multiple of y .
 - f) x and y are both negative or both nonnegative.
 - g) $x = y^2$.
 - h) $x \geq y^2$.
 8. Show that the relation $R = \emptyset$ on a nonempty set S is symmetric and transitive, but not reflexive.
 9. Show that the relation $R = \emptyset$ on the empty set $S = \emptyset$ is reflexive, symmetric, and transitive.
 10. Give an example of a relation on a set that is
 - a) both symmetric and antisymmetric.
 - b) neither symmetric nor antisymmetric.
- A relation R on the set A is **irreflexive** if for every $a \in A$, $(a, a) \notin R$. That is, R is irreflexive if no element in A is related to itself.
11. Which relations in Exercise 3 are irreflexive?
 12. Which relations in Exercise 4 are irreflexive?
 13. Which relations in Exercise 5 are irreflexive?
 14. Which relations in Exercise 6 are irreflexive?
 15. Can a relation on a set be neither reflexive nor irreflexive?
 16. Use quantifiers to express what it means for a relation to be irreflexive.
 17. Give an example of an irreflexive relation on the set of all people.

A relation R is called **asymmetric** if $(a, b) \in R$ implies that $(b, a) \notin R$. Exercises 18–24 explore the notion of an asymmetric relation. Exercise 22 focuses on the difference between asymmetry and antisymmetry.

18. Which relations in Exercise 3 are asymmetric?
 19. Which relations in Exercise 4 are asymmetric?
 20. Which relations in Exercise 5 are asymmetric?
 21. Which relations in Exercise 6 are asymmetric?
 22. Must an asymmetric relation also be antisymmetric? Must an antisymmetric relation be asymmetric? Give reasons for your answers.
 23. Use quantifiers to express what it means for a relation to be asymmetric.
 24. Give an example of an asymmetric relation on the set of all people.
 25. How many different relations are there from a set with m elements to a set with n elements?
-  Let R be a relation from a set A to a set B . The **inverse relation** from B to A , denoted by R^{-1} , is the set of ordered pairs $\{(b, a) \mid (a, b) \in R\}$. The **complementary relation** \bar{R} is the set of ordered pairs $\{(a, b) \mid (a, b) \notin R\}$.
26. Let R be the relation $R = \{(a, b) \mid a < b\}$ on the set of integers. Find
 - R^{-1} .
 - \bar{R} .
 27. Let R be the relation $R = \{(a, b) \mid a \text{ divides } b\}$ on the set of positive integers. Find
 - R^{-1} .
 - \bar{R} .
 28. Let R be the relation on the set of all states in the United States consisting of pairs (a, b) where state a borders state b . Find
 - R^{-1} .
 - \bar{R} .
 29. Suppose that the function f from A to B is a one-to-one correspondence. Let R be the relation that equals the graph of f . That is, $R = \{(a, f(a)) \mid a \in A\}$. What is the inverse relation R^{-1} ?
 30. Let $R_1 = \{(1, 2), (2, 3), (3, 4)\}$ and $R_2 = \{(1, 1), (1, 2), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3), (3, 4)\}$ be relations from $\{1, 2, 3\}$ to $\{1, 2, 3, 4\}$. Find
 - $R_1 \cup R_2$.
 - $R_1 \cap R_2$.
 - $R_1 - R_2$.
 - $R_2 - R_1$.
 31. Let A be the set of students at your school and B the set of books in the school library. Let R_1 and R_2 be the relations consisting of all ordered pairs (a, b) , where student a is required to read book b in a course, and where student a has read book b , respectively. Describe the ordered pairs in each of these relations.
 - $R_1 \cup R_2$
 - $R_1 \cap R_2$
 - $R_1 \oplus R_2$
 - $R_1 - R_2$
 - $R_2 - R_1$
 32. Let R be the relation $\{(1, 2), (1, 3), (2, 3), (2, 4), (3, 1)\}$, and let S be the relation $\{(2, 1), (3, 1), (3, 2), (4, 2)\}$. Find $S \circ R$.

33. Let R be the relation on the set of people consisting of pairs (a, b) , where a is a parent of b . Let S be the relation on the set of people consisting of pairs (a, b) , where a and b are siblings (brothers or sisters). What are $S \circ R$ and $R \circ S$?

Exercises 34–37 deal with these relations on the set of real numbers:

$$\begin{aligned} R_1 &= \{(a, b) \in \mathbf{R}^2 \mid a > b\}, \text{ the "greater than" relation,} \\ R_2 &= \{(a, b) \in \mathbf{R}^2 \mid a \geq b\}, \text{ the "greater than or equal to" relation,} \\ R_3 &= \{(a, b) \in \mathbf{R}^2 \mid a < b\}, \text{ the "less than" relation,} \\ R_4 &= \{(a, b) \in \mathbf{R}^2 \mid a \leq b\}, \text{ the "less than or equal to" relation,} \\ R_5 &= \{(a, b) \in \mathbf{R}^2 \mid a = b\}, \text{ the "equal to" relation,} \\ R_6 &= \{(a, b) \in \mathbf{R}^2 \mid a \neq b\}, \text{ the "unequal to" relation.} \end{aligned}$$

34. Find

a) $R_1 \cup R_3$.	b) $R_1 \cup R_5$.
c) $R_2 \cap R_4$.	d) $R_3 \cap R_5$.
e) $R_1 - R_2$.	f) $R_2 - R_1$.
g) $R_1 \oplus R_3$.	h) $R_2 \oplus R_4$.
35. Find

a) $R_2 \cup R_4$.	b) $R_3 \cup R_6$.
c) $R_3 \cap R_6$.	d) $R_4 \cap R_6$.
e) $R_3 - R_6$.	f) $R_6 - R_3$.
g) $R_2 \oplus R_6$.	h) $R_3 \oplus R_5$.
36. Find

a) $R_1 \circ R_1$.	b) $R_1 \circ R_2$.
c) $R_1 \circ R_3$.	d) $R_1 \circ R_4$.
e) $R_1 \circ R_5$.	f) $R_1 \circ R_6$.
g) $R_2 \circ R_3$.	h) $R_3 \circ R_3$.
37. Find

a) $R_2 \circ R_1$.	b) $R_2 \circ R_2$.
c) $R_3 \circ R_5$.	d) $R_4 \circ R_1$.
e) $R_5 \circ R_3$.	f) $R_3 \circ R_6$.
g) $R_4 \circ R_6$.	h) $R_6 \circ R_6$.
38. Let R be the parent relation on the set of all people (see Example 21). When is an ordered pair in the relation R^3 ?
39. Let R be the relation on the set of people with doctorates such that $(a, b) \in R$ if and only if a was the thesis advisor of b . When is an ordered pair (a, b) in R^2 ? When is an ordered pair (a, b) in R^n , when n is a positive integer? (Assume that every person with a doctorate has a thesis advisor.)
40. Let R_1 and R_2 be the "divides" and "is a multiple of" relations on the set of all positive integers, respectively. That is, $R_1 = \{(a, b) \mid a \text{ divides } b\}$ and $R_2 = \{(a, b) \mid a \text{ is a multiple of } b\}$. Find

a) $R_1 \cup R_2$.	b) $R_1 \cap R_2$.
c) $R_1 - R_2$.	d) $R_2 - R_1$.
e) $R_1 \oplus R_2$.	

- 41.** Let R_1 and R_2 be the “congruent modulo 3” and the “congruent modulo 4” relations, respectively, on the set of integers. That is, $R_1 = \{(a, b) \mid a \equiv b \pmod{3}\}$ and $R_2 = \{(a, b) \mid a \equiv b \pmod{4}\}$. Find
- $R_1 \cup R_2$.
 - $R_1 \cap R_2$.
 - $R_1 - R_2$.
 - $R_2 - R_1$.
 - $R_1 \oplus R_2$.
- 42.** List the 16 different relations on the set $\{0, 1\}$.
- 43.** How many of the 16 different relations on $\{0, 1\}$ contain the pair $(0, 1)$?
- 44.** Which of the 16 relations on $\{0, 1\}$, which you listed in Exercise 42, are
- reflexive?
 - irreflexive?
 - symmetric?
 - antisymmetric?
 - asymmetric?
 - transitive?
- 45.** a) How many relations are there on the set $\{a, b, c, d\}$?
b) How many relations are there on the set $\{a, b, c, d\}$ that contain the pair (a, a) ?
- 46.** Let S be a set with n elements and let a and b be distinct elements of S . How many relations R are there on S such that
- $(a, b) \in R$?
 - $(a, b) \notin R$?
 - no ordered pair in R has a as its first element?
 - at least one ordered pair in R has a as its first element?
 - no ordered pair in R has a as its first element or b as its second element?
 - at least one ordered pair in R either has a as its first element or has b as its second element?
- *47.** How many relations are there on a set with n elements that are
- symmetric?
 - antisymmetric?
 - asymmetric?
 - irreflexive?
 - reflexive and symmetric?
 - neither reflexive nor irreflexive?
- *48.** How many transitive relations are there on a set with n elements if
- $n = 1$?
 - $n = 2$?
 - $n = 3$?
- 49.** Find the error in the “proof” of the following “theorem.”
- “Theorem”: Let R be a relation on a set A that is symmetric and transitive. Then R is reflexive.
- “Proof”: Let $a \in A$. Take an element $b \in A$ such that $(a, b) \in R$. Because R is symmetric, we also have $(b, a) \in R$. Now using the transitive property, we can conclude that $(a, a) \in R$ because $(a, b) \in R$ and $(b, a) \in R$.
- 50.** Suppose that R and S are reflexive relations on a set A . Prove or disprove each of these statements.
- $R \cup S$ is reflexive.
 - $R \cap S$ is reflexive.
 - $R \oplus S$ is irreflexive.
 - $R - S$ is irreflexive.
 - $S \circ R$ is reflexive.
- 51.** Show that the relation R on a set A is symmetric if and only if $R = R^{-1}$, where R^{-1} is the inverse relation.
- 52.** Show that the relation R on a set A is antisymmetric if and only if $R \cap R^{-1}$ is a subset of the diagonal relation $\Delta = \{(a, a) \mid a \in A\}$.
- 53.** Show that the relation R on a set A is reflexive if and only if the inverse relation R^{-1} is reflexive.
- 54.** Show that the relation R on a set A is reflexive if and only if the complementary relation \bar{R} is irreflexive.
- 55.** Let R be a relation that is reflexive and transitive. Prove that $R^n = R$ for all positive integers n .
- 56.** Let R be the relation on the set $\{1, 2, 3, 4, 5\}$ containing the ordered pairs $(1, 1), (1, 2), (1, 3), (2, 3), (2, 4), (3, 1), (3, 4), (3, 5), (4, 2), (4, 5), (5, 1), (5, 2)$, and $(5, 4)$. Find
- R^2 .
 - R^3 .
 - R^4 .
 - R^5 .
- 57.** Let R be a reflexive relation on a set A . Show that R^n is reflexive for all positive integers n .
- *58.** Let R be a symmetric relation. Show that R^n is symmetric for all positive integers n .
- 59.** Suppose that the relation R is irreflexive. Is R^2 necessarily irreflexive? Give a reason for your answer.

9.2 *n*-ary Relations and Their Applications

Introduction

Relationships among elements of more than two sets often arise. For instance, there is a relationship involving the name of a student, the student’s major, and the student’s grade point average. Similarly, there is a relationship involving the airline, flight number, starting point, destination, departure time, and arrival time of a flight. An example of such a relationship in mathematics involves three integers, where the first integer is larger than the second integer, which is larger than the third. Another example is the betweenness relationship involving points on a line, such that three points are related when the second point is between the first and the third.

We will study relationships among elements from more than two sets in this section. These relationships are called ***n*-ary relations**. These relations are used to represent computer databases. These representations help us answer queries about the information stored in databases, such as: Which flights land at O’Hare Airport between 3 A.M. and 4 A.M.? Which students at your

school are sophomores majoring in mathematics or computer science and have greater than a 3.0 average? Which employees of a company have worked for the company less than 5 years and make more than \$50,000?

n-ary Relations

We begin with the basic definition on which the theory of relational databases rests.

DEFINITION 1

Let A_1, A_2, \dots, A_n be sets. An *n-ary relation* on these sets is a subset of $A_1 \times A_2 \times \dots \times A_n$. The sets A_1, A_2, \dots, A_n are called the *domains* of the relation, and n is called its *degree*.

EXAMPLE 1

Let R be the relation on $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$ consisting of triples (a, b, c) , where a, b , and c are integers with $a < b < c$. Then $(1, 2, 3) \in R$, but $(2, 4, 3) \notin R$. The degree of this relation is 3. Its domains are all equal to the set of natural numbers. 

EXAMPLE 2

Let R be the relation on $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ consisting of all triples of integers (a, b, c) in which a, b , and c form an arithmetic progression. That is, $(a, b, c) \in R$ if and only if there is an integer k such that $b = a + k$ and $c = a + 2k$, or equivalently, such that $b - a = k$ and $c - b = k$. Note that $(1, 3, 5) \in R$ because $3 = 1 + 2$ and $5 = 1 + 2 \cdot 2$, but $(2, 5, 9) \notin R$ because $5 - 2 = 3$ while $9 - 5 = 4$. This relation has degree 3 and its domains are all equal to the set of integers. 

EXAMPLE 3

Let R be the relation on $\mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}^+$ consisting of triples (a, b, m) , where a, b , and m are integers with $m \geq 1$ and $a \equiv b \pmod{m}$. Then $(8, 2, 3), (-1, 9, 5)$, and $(14, 0, 7)$ all belong to R , but $(7, 2, 3), (-2, -8, 5)$, and $(11, 0, 6)$ do not belong to R because $8 \equiv 2 \pmod{3}, -1 \equiv 9 \pmod{5}$, and $14 \equiv 0 \pmod{7}$, but $7 \not\equiv 2 \pmod{3}, -2 \not\equiv -8 \pmod{5}$, and $11 \not\equiv 0 \pmod{6}$. This relation has degree 3 and its first two domains are the set of all integers and its third domain is the set of positive integers. 

EXAMPLE 4

Let R be the relation consisting of 5-tuples (A, N, S, D, T) representing airplane flights, where A is the airline, N is the flight number, S is the starting point, D is the destination, and T is the departure time. For instance, if Nadir Express Airlines has flight 963 from Newark to Bangor at 15:00, then $(\text{Nadir}, 963, \text{Newark}, \text{Bangor}, 15:00)$ belongs to R . The degree of this relation is 5, and its domains are the set of all airlines, the set of flight numbers, the set of cities, the set of cities (again), and the set of times. 

Databases and Relations



The time required to manipulate information in a database depends on how this information is stored. The operations of adding and deleting records, updating records, searching for records, and combining records from overlapping databases are performed millions of times each day in a large database. Because of the importance of these operations, various methods for representing databases have been developed. We will discuss one of these methods, called the **relational data model**, based on the concept of a relation.

A database consists of **records**, which are *n*-tuples, made up of **fields**. The fields are the entries of the *n*-tuples. For instance, a database of student records may be made up of fields containing the name, student number, major, and grade point average of the student. The relational data model represents a database of records as an *n*-ary relation. Thus, student records

TABLE 1 Students.

<i>Student_name</i>	<i>ID_number</i>	<i>Major</i>	<i>GPA</i>
Ackermann	231455	Computer Science	3.88
Adams	888323	Physics	3.45
Chou	102147	Computer Science	3.49
Goodfriend	453876	Mathematics	3.45
Rao	678543	Mathematics	3.90
Stevens	786576	Psychology	2.99

are represented as 4-tuples of the form (*Student_name*, *ID_number*, *Major*, *GPA*). A sample database of six such records is

- (Ackermann, 231455, Computer Science, 3.88)
- (Adams, 888323, Physics, 3.45)
- (Chou, 102147, Computer Science, 3.49)
- (Goodfriend, 453876, Mathematics, 3.45)
- (Rao, 678543, Mathematics, 3.90)
- (Stevens, 786576, Psychology, 2.99).

Relations used to represent databases are also called **tables**, because these relations are often displayed as tables. Each column of the table corresponds to an *attribute* of the database. For instance, the same database of students is displayed in Table 1. The attributes of this database are Student Name, ID Number, Major, and GPA.

A domain of an *n*-ary relation is called a **primary key** when the value of the *n*-tuple from this domain determines the *n*-tuple. That is, a domain is a primary key when no two *n*-tuples in the relation have the same value from this domain.

Records are often added to or deleted from databases. Because of this, the property that a domain is a primary key is time-dependent. Consequently, a primary key should be chosen that remains one whenever the database is changed. The current collection of *n*-tuples in a relation is called the **extension** of the relation. The more permanent part of a database, including the name and attributes of the database, is called its **intension**. When selecting a primary key, the goal should be to select a key that can serve as a primary key for all possible extensions of the database. To do this, it is necessary to examine the intension of the database to understand the set of possible *n*-tuples that can occur in an extension.

EXAMPLE 5 Which domains are primary keys for the *n*-ary relation displayed in Table 1, assuming that no *n*-tuples will be added in the future?

Solution: Because there is only one 4-tuple in this table for each student name, the domain of student names is a primary key. Similarly, the ID numbers in this table are unique, so the domain of ID numbers is also a primary key. However, the domain of major fields of study is not a primary key, because more than one 4-tuple contains the same major field of study. The domain of grade point averages is also not a primary key, because there are two 4-tuples containing the same GPA. 

Combinations of domains can also uniquely identify *n*-tuples in an *n*-ary relation. When the values of a set of domains determine an *n*-tuple in a relation, the Cartesian product of these domains is called a **composite key**.

EXAMPLE 6 Is the Cartesian product of the domain of major fields of study and the domain of GPAs a composite key for the n -ary relation from Table 1, assuming that no n -tuples are ever added?

Solution: Because no two 4-tuples from this table have both the same major and the same GPA, this Cartesian product is a composite key. 

Because primary and composite keys are used to identify records uniquely in a database, it is important that keys remain valid when new records are added to the database. Hence, checks should be made to ensure that every new record has values that are different in the appropriate field, or fields, from all other records in this table. For instance, it makes sense to use the student identification number as a key for student records if no two students ever have the same student identification number. A university should not use the name field as a key, because two students may have the same name (such as John Smith).

Operations on n -ary Relations

There are a variety of operations on n -ary relations that can be used to form new n -ary relations. Applied together, these operations can answer queries on databases that ask for all n -tuples that satisfy certain conditions.

The most basic operation on an n -ary relation is determining all n -tuples in the n -ary relation that satisfy certain conditions. For example, we may want to find all the records of all computer science majors in a database of student records. We may want to find all students who have a grade point average above 3.5. We may want to find the records of all computer science majors who have a grade point average above 3.5. To perform such tasks we use the selection operator.

DEFINITION 2

Let R be an n -ary relation and C a condition that elements in R may satisfy. Then the *selection operator* s_C maps the n -ary relation R to the n -ary relation of all n -tuples from R that satisfy the condition C .

EXAMPLE 7

To find the records of computer science majors in the n -ary relation R shown in Table 1, we use the operator s_{C_1} , where C_1 is the condition Major = “Computer Science.” The result is the two 4-tuples (Ackermann, 231455, Computer Science, 3.88) and (Chou, 102147, Computer Science, 3.49). Similarly, to find the records of students who have a grade point average above 3.5 in this database, we use the operator s_{C_2} , where C_2 is the condition GPA > 3.5. The result is the two 4-tuples (Ackermann, 231455, Computer Science, 3.88) and (Rao, 678543, Mathematics, 3.90). Finally, to find the records of computer science majors who have a GPA above 3.5, we use the operator s_{C_3} , where C_3 is the condition (Major = “Computer Science” \wedge GPA > 3.5). The result consists of the single 4-tuple (Ackermann, 231455, Computer Science, 3.88). 

Projections are used to form new n -ary relations by deleting the same fields in every record of the relation.

DEFINITION 3

The *projection* $P_{i_1 i_2 \dots i_m}$ where $i_1 < i_2 < \dots < i_m$, maps the n -tuple (a_1, a_2, \dots, a_n) to the m -tuple $(a_{i_1}, a_{i_2}, \dots, a_{i_m})$, where $m \leq n$.

In other words, the projection $P_{i_1 i_2 \dots i_m}$ deletes $n - m$ of the components of an n -tuple, leaving the i_1 th, i_2 th, . . . , and i_m th components.

TABLE 2 GPAs.	
<i>Student_name</i>	<i>GPA</i>
Ackermann	3.88
Adams	3.45
Chou	3.49
Goodfriend	3.45
Rao	3.90
Stevens	2.99

TABLE 3 Enrollments.		
<i>Student</i>	<i>Major</i>	<i>Course</i>
Glauser	Biology	BI 290
Glauser	Biology	MS 475
Glauser	Biology	PY 410
Marcus	Mathematics	MS 511
Marcus	Mathematics	MS 603
Marcus	Mathematics	CS 322
Miller	Computer Science	MS 575
Miller	Computer Science	CS 455

TABLE 4 Majors.	
<i>Student</i>	<i>Major</i>
Glauser	Biology
Marcus	Mathematics
Miller	Computer Science

EXAMPLE 8 What results when the projection $P_{1,3}$ is applied to the 4-tuples $(2, 3, 0, 4)$, (Jane Doe, 234111001, Geography, 3.14), and (a_1, a_2, a_3, a_4) ?

Solution: The projection $P_{1,3}$ sends these 4-tuples to $(2, 0)$, (Jane Doe, Geography), and (a_1, a_3) , respectively. 

Example 9 illustrates how new relations are produced using projections.

EXAMPLE 9 What relation results when the projection $P_{1,4}$ is applied to the relation in Table 1?

Solution: When the projection $P_{1,4}$ is used, the second and third columns of the table are deleted, and pairs representing student names and grade point averages are obtained. Table 2 displays the results of this projection. 

Fewer rows may result when a projection is applied to the table for a relation. This happens when some of the n -tuples in the relation have identical values in each of the m components of the projection, and only disagree in components deleted by the projection. For instance, consider the following example.

EXAMPLE 10 What is the table obtained when the projection $P_{1,2}$ is applied to the relation in Table 3?

Solution: Table 4 displays the relation obtained when $P_{1,2}$ is applied to Table 3. Note that there are fewer rows after this projection is applied. 

The **join** operation is used to combine two tables into one when these tables share some identical fields. For instance, a table containing fields for airline, flight number, and gate, and another table containing fields for flight number, gate, and departure time can be combined into a table containing fields for airline, flight number, gate, and departure time.

DEFINITION 4

Let R be a relation of degree m and S a relation of degree n . The $join$ $J_p(R, S)$, where $p \leq m$ and $p \leq n$, is a relation of degree $m + n - p$ that consists of all $(m + n - p)$ -tuples $(a_1, a_2, \dots, a_{m-p}, c_1, c_2, \dots, c_p, b_1, b_2, \dots, b_{n-p})$, where the m -tuple $(a_1, a_2, \dots, a_{m-p}, c_1, c_2, \dots, c_p)$ belongs to R and the n -tuple $(c_1, c_2, \dots, c_p, b_1, b_2, \dots, b_{n-p})$ belongs to S .

In other words, the join operator J_p produces a new relation from two relations by combining all m -tuples of the first relation with all n -tuples of the second relation, where the last p components of the m -tuples agree with the first p components of the n -tuples.

TABLE 5 Teaching_assignments.

<i>Professor</i>	<i>Department</i>	<i>Course_number</i>
Cruz	Zoology	335
Cruz	Zoology	412
Farber	Psychology	501
Farber	Psychology	617
Grammer	Physics	544
Grammer	Physics	551
Rosen	Computer Science	518
Rosen	Mathematics	575

TABLE 6 Class_schedule.

<i>Department</i>	<i>Course_number</i>	<i>Room</i>	<i>Time</i>
Computer Science	518	N521	2:00 P.M.
Mathematics	575	N502	3:00 P.M.
Mathematics	611	N521	4:00 P.M.
Physics	544	B505	4:00 P.M.
Psychology	501	A100	3:00 P.M.
Psychology	617	A110	11:00 A.M.
Zoology	335	A100	9:00 A.M.
Zoology	412	A100	8:00 A.M.

EXAMPLE 11 What relation results when the join operator J_2 is used to combine the relation displayed in Tables 5 and 6?

Solution: The join J_2 produces the relation shown in Table 7. 

There are other operators besides projections and joins that produce new relations from existing relations. A description of these operations can be found in books on database theory.

SQL



The database query language SQL (short for Structured Query Language) can be used to carry out the operations we have described in this section. Example 12 illustrates how SQL commands are related to operations on n -ary relations.

EXAMPLE 12 We will illustrate how SQL is used to express queries by showing how SQL can be employed to make a query about airline flights using Table 8. The SQL statement

```
SELECT Departure_time
  FROM Flights
 WHERE Destination='Detroit'
```

is used to find the projection P_5 (on the *Departure_time* attribute) of the selection of 5-tuples in the *Flights* database that satisfy the condition: *Destination* = 'Detroit'. The output would be a list containing the times of flights that have Detroit as their destination, namely, 08:10, 08:47,

TABLE 7 Teaching_schedule.

<i>Professor</i>	<i>Department</i>	<i>Course_number</i>	<i>Room</i>	<i>Time</i>
Cruz	Zoology	335	A100	9:00 A.M.
Cruz	Zoology	412	A100	8:00 A.M.
Farber	Psychology	501	A100	3:00 P.M.
Farber	Psychology	617	A110	11:00 A.M.
Grammer	Physics	544	B505	4:00 P.M.
Rosen	Computer Science	518	N521	2:00 P.M.
Rosen	Mathematics	575	N502	3:00 P.M.

TABLE 8 Flights.

Airline	Flight_number	Gate	Destination	Departure_time
Nadir	122	34	Detroit	08:10
Acme	221	22	Denver	08:17
Acme	122	33	Anchorage	08:22
Acme	323	34	Honolulu	08:30
Nadir	199	13	Detroit	08:47
Acme	222	22	Denver	09:10
Nadir	322	34	Detroit	09:44

and 09:44. SQL uses the FROM clause to identify the *n*-ary relation the query is applied to, the WHERE clause to specify the condition of the selection operation, and the SELECT clause to specify the projection operation that is to be applied. (*Beware:* SQL uses SELECT to represent a projection, rather than a selection operation. This is an unfortunate example of conflicting terminology.)

Example 13 shows how SQL queries can be made involving more than one table.

EXAMPLE 13

The SQL statement

```
SELECT Professor, Time
FROM Teaching_assignments, Class_schedule
WHERE Department='Mathematics'
```

is used to find the projection $P_{1,5}$ of the 5-tuples in the database (shown in Table 7), which is the join J_2 of the Teaching_assignments and Class_schedule databases in Tables 5 and 6, respectively, which satisfy the condition: Department = Mathematics. The output would consist of the single 2-tuple (Rosen, 3:00 P.M.). The SQL FROM clause is used here to find the join of two different databases.

We have only touched on the basic concepts of relational databases in this section. More information can be found in [AhUl95].

Exercises

1. List the triples in the relation $\{(a, b, c) \mid a, b, \text{ and } c \text{ are integers with } 0 < a < b < c < 5\}$.
2. Which 4-tuples are in the relation $\{(a, b, c, d) \mid a, b, c, \text{ and } d \text{ are positive integers with } abcd = 6\}$?
3. List the 5-tuples in the relation in Table 8.
4. Assuming that no new *n*-tuples are added, find all the primary keys for the relations displayed in
 - a) Table 3.
 - b) Table 5.
 - c) Table 6.
 - d) Table 8.
5. Assuming that no new *n*-tuples are added, find a composite key with two fields containing the *Airline* field for the database in Table 8.
6. Assuming that no new *n*-tuples are added, find a composite key with two fields containing the *Professor* field for the database in Table 7.
7. The 3-tuples in a 3-ary relation represent the following attributes of a student database: student ID number, name, phone number.
 - a) Is student ID number likely to be a primary key?
 - b) Is name likely to be a primary key?
 - c) Is phone number likely to be a primary key?
8. The 4-tuples in a 4-ary relation represent these attributes of published books: title, ISBN, publication date, number of pages.
 - a) What is a likely primary key for this relation?
 - b) Under what conditions would (title, publication date) be a composite key?
 - c) Under what conditions would (title, number of pages) be a composite key?

9. The 5-tuples in a 5-ary relation represent these attributes of all people in the United States: name, Social Security number, street address, city, state.
 - a) Determine a primary key for this relation.
 - b) Under what conditions would (name, street address) be a composite key?
 - c) Under what conditions would (name, street address, city) be a composite key?
10. What do you obtain when you apply the selection operator s_C , where C is the condition Room = A100, to the database in Table 7?
11. What do you obtain when you apply the selection operator s_C , where C is the condition Destination = Detroit, to the database in Table 8?
12. What do you obtain when you apply the selection operator s_C , where C is the condition (Project = 2) \wedge (Quantity \geq 50), to the database in Table 10?
13. What do you obtain when you apply the selection operator s_C , where C is the condition (Airline = Nadir) \vee (Destination = Denver), to the database in Table 8?
14. What do you obtain when you apply the projection $P_{2,3,5}$ to the 5-tuple (a, b, c, d, e) ?
15. Which projection mapping is used to delete the first, second, and fourth components of a 6-tuple?
16. Display the table produced by applying the projection $P_{1,2,4}$ to Table 8.
17. Display the table produced by applying the projection $P_{1,4}$ to Table 8.
18. How many components are there in the n -tuples in the table obtained by applying the join operator J_3 to two tables with 5-tuples and 8-tuples, respectively?
19. Construct the table obtained by applying the join operator J_2 to the relations in Tables 9 and 10.
20. Show that if C_1 and C_2 are conditions that elements of the n -ary relation R may satisfy, then $s_{C_1 \wedge C_2}(R) = s_{C_1}(s_{C_2}(R))$.
21. Show that if C_1 and C_2 are conditions that elements of the n -ary relation R may satisfy, then $s_{C_1}(s_{C_2}(R)) = s_{C_2}(s_{C_1}(R))$.
22. Show that if C is a condition that elements of the n -ary relations R and S may satisfy, then $s_C(R \cup S) = s_C(R) \cup s_C(S)$.

TABLE 9 Part_needs.

Supplier	Part_number	Project
23	1092	1
23	1101	3
23	9048	4
31	4975	3
31	3477	2
32	6984	4
32	9191	2
33	1001	1

TABLE 10 Parts_inventory.

Part_number	Project	Quantity	Color_code
1001	1	14	8
1092	1	2	2
1101	3	1	1
3477	2	25	2
4975	3	6	2
6984	4	10	1
9048	4	12	2
9191	2	80	4

9.3 Representing Relations

Introduction

In this section, and in the remainder of this chapter, all relations we study will be binary relations. Because of this, in this section and in the rest of this chapter, the word relation will always refer to a binary relation. There are many ways to represent a relation between finite sets. As we have seen in Section 9.1, one way is to list its ordered pairs. Another way to represent a relation is to use a table, as we did in Example 3 in Section 9.1. In this section we will discuss two alternative methods for representing relations. One method uses zero–one matrices. The other method uses pictorial representations called directed graphs, which we will discuss later in this section.

Generally, matrices are appropriate for the representation of relations in computer programs. On the other hand, people often find the representation of relations using directed graphs useful for understanding the properties of these relations.

Representing Relations Using Matrices

A relation between finite sets can be represented using a zero–one matrix. Suppose that R is a relation from $A = \{a_1, a_2, \dots, a_m\}$ to $B = \{b_1, b_2, \dots, b_n\}$. (Here the elements of the sets A and B have been listed in a particular, but arbitrary, order. Furthermore, when $A = B$ we use the same ordering for A and B .) The relation R can be represented by the matrix $\mathbf{M}_R = [m_{ij}]$, where

$$m_{ij} = \begin{cases} 1 & \text{if } (a_i, b_j) \in R, \\ 0 & \text{if } (a_i, b_j) \notin R. \end{cases}$$

In other words, the zero–one matrix representing R has a 1 as its (i, j) entry when a_i is related to b_j , and a 0 in this position if a_i is not related to b_j . (Such a representation depends on the orderings used for A and B .)

The use of matrices to represent relations is illustrated in Examples 1–6.

EXAMPLE 1 Suppose that $A = \{1, 2, 3\}$ and $B = \{1, 2\}$. Let R be the relation from A to B containing (a, b) if $a \in A$, $b \in B$, and $a > b$. What is the matrix representing R if $a_1 = 1$, $a_2 = 2$, and $a_3 = 3$, and $b_1 = 1$ and $b_2 = 2$?

Solution: Because $R = \{(2, 1), (3, 1), (3, 2)\}$, the matrix for R is

$$\mathbf{M}_R = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

The 1s in \mathbf{M}_R show that the pairs $(2, 1)$, $(3, 1)$, and $(3, 2)$ belong to R . The 0s show that no other pairs belong to R . 

EXAMPLE 2 Let $A = \{a_1, a_2, a_3\}$ and $B = \{b_1, b_2, b_3, b_4, b_5\}$. Which ordered pairs are in the relation R represented by the matrix

$$\mathbf{M}_R = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}?$$

$$\begin{bmatrix} 1 & & & & \\ 1 & 1 & & & \\ & 1 & \ddots & & \\ & & \ddots & \ddots & \\ & & & 1 & 1 \end{bmatrix}$$

FIGURE 1 The Zero–One Matrix for a Reflexive Relation. (Off Diagonal Elements Can Be 0 or 1.)

Solution: Because R consists of those ordered pairs (a_i, b_j) with $m_{ij} = 1$, it follows that

$$R = \{(a_1, b_2), (a_2, b_1), (a_2, b_3), (a_2, b_4), (a_3, b_1), (a_3, b_3), (a_3, b_5)\}. \quad \blacktriangleleft$$

The matrix of a relation on a set, which is a square matrix, can be used to determine whether the relation has certain properties. Recall that a relation R on A is reflexive if $(a, a) \in R$ whenever $a \in A$. Thus, R is reflexive if and only if $(a_i, a_i) \in R$ for $i = 1, 2, \dots, n$. Hence, R is reflexive if and only if $m_{ii} = 1$, for $i = 1, 2, \dots, n$. In other words, R is reflexive if all the elements on the main diagonal of \mathbf{M}_R are equal to 1, as shown in Figure 1. Note that the elements off the main diagonal can be either 0 or 1.

The relation R is symmetric if $(a, b) \in R$ implies that $(b, a) \in R$. Consequently, the relation R on the set $A = \{a_1, a_2, \dots, a_n\}$ is symmetric if and only if $(a_j, a_i) \in R$ whenever $(a_i, a_j) \in R$. In terms of the entries of \mathbf{M}_R , R is symmetric if and only if $m_{ji} = 1$ whenever $m_{ij} = 1$. This also means $m_{ji} = 0$ whenever $m_{ij} = 0$. Consequently, R is symmetric if and only if $m_{ij} = m_{ji}$, for all pairs of integers i and j with $i = 1, 2, \dots, n$ and $j = 1, 2, \dots, n$. Recalling the definition of the transpose of a matrix from Section 2.6, we see that R is symmetric if and only if

$$\mathbf{M}_R = (\mathbf{M}_R)^t,$$

that is, if \mathbf{M}_R is a symmetric matrix. The form of the matrix for a symmetric relation is illustrated in Figure 2(a).

The relation R is antisymmetric if and only if $(a, b) \in R$ and $(b, a) \in R$ imply that $a = b$. Consequently, the matrix of an antisymmetric relation has the property that if $m_{ij} = 1$ with $i \neq j$, then $m_{ji} = 0$. Or, in other words, either $m_{ij} = 0$ or $m_{ji} = 0$ when $i \neq j$. The form of the matrix for an antisymmetric relation is illustrated in Figure 2(b).

$$\begin{array}{cc} \begin{bmatrix} & 1 & \\ 1 & & 0 \\ & 0 & \end{bmatrix} & \begin{bmatrix} & 1 & 0 & 0 \\ 0 & & & \\ 0 & & 1 & \\ & & & \end{bmatrix} \end{array}$$

(a) Symmetric

(b) Antisymmetric

FIGURE 2 The Zero–One Matrices for Symmetric and Antisymmetric Relations.

EXAMPLE 3 Suppose that the relation R on a set is represented by the matrix

$$\mathbf{M}_R = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

Is R reflexive, symmetric, and/or antisymmetric?

Solution: Because all the diagonal elements of this matrix are equal to 1, R is reflexive. Moreover, because \mathbf{M}_R is symmetric, it follows that R is symmetric. It is also easy to see that R is not antisymmetric. \blacktriangleleft

The Boolean operations join and meet (discussed in Section 2.6) can be used to find the matrices representing the union and the intersection of two relations. Suppose that R_1 and R_2 are relations on a set A represented by the matrices \mathbf{M}_{R_1} and \mathbf{M}_{R_2} , respectively. The matrix

representing the union of these relations has a 1 in the positions where either \mathbf{M}_{R_1} or \mathbf{M}_{R_2} has a 1. The matrix representing the intersection of these relations has a 1 in the positions where both \mathbf{M}_{R_1} and \mathbf{M}_{R_2} have a 1. Thus, the matrices representing the union and intersection of these relations are

$$\mathbf{M}_{R_1 \cup R_2} = \mathbf{M}_{R_1} \vee \mathbf{M}_{R_2} \quad \text{and} \quad \mathbf{M}_{R_1 \cap R_2} = \mathbf{M}_{R_1} \wedge \mathbf{M}_{R_2}.$$

EXAMPLE 4 Suppose that the relations R_1 and R_2 on a set A are represented by the matrices

$$\mathbf{M}_{R_1} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{M}_{R_2} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

What are the matrices representing $R_1 \cup R_2$ and $R_1 \cap R_2$?

Solution: The matrices of these relations are

$$\mathbf{M}_{R_1 \cup R_2} = \mathbf{M}_{R_1} \vee \mathbf{M}_{R_2} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix},$$

$$\mathbf{M}_{R_1 \cap R_2} = \mathbf{M}_{R_1} \wedge \mathbf{M}_{R_2} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$
◀

We now turn our attention to determining the matrix for the composite of relations. This matrix can be found using the Boolean product of the matrices (discussed in Section 2.6) for these relations. In particular, suppose that R is a relation from A to B and S is a relation from B to C . Suppose that A , B , and C have m , n , and p elements, respectively. Let the zero-one matrices for $S \circ R$, R , and S be $\mathbf{M}_{S \circ R} = [t_{ij}]$, $\mathbf{M}_R = [r_{ij}]$, and $\mathbf{M}_S = [s_{ij}]$, respectively (these matrices have sizes $m \times p$, $m \times n$, and $n \times p$, respectively). The ordered pair (a_i, c_j) belongs to $S \circ R$ if and only if there is an element b_k such that (a_i, b_k) belongs to R and (b_k, c_j) belongs to S . It follows that $t_{ij} = 1$ if and only if $r_{ik} = s_{kj} = 1$ for some k . From the definition of the Boolean product, this means that

$$\mathbf{M}_{S \circ R} = \mathbf{M}_R \odot \mathbf{M}_S.$$

EXAMPLE 5 Find the matrix representing the relations $S \circ R$, where the matrices representing R and S are

$$\mathbf{M}_R = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{M}_S = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}.$$

Solution: The matrix for $S \circ R$ is

$$\mathbf{M}_{S \circ R} = \mathbf{M}_R \odot \mathbf{M}_S = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$
◀

The matrix representing the composite of two relations can be used to find the matrix for \mathbf{M}_{R^n} . In particular,

$$\mathbf{M}_{R^n} = \mathbf{M}_R^{[n]},$$

from the definition of Boolean powers. Exercise 35 asks for a proof of this formula.

EXAMPLE 6 Find the matrix representing the relation R^2 , where the matrix representing R is

$$\mathbf{M}_R = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

Solution: The matrix for R^2 is

$$\mathbf{M}_{R^2} = \mathbf{M}_R^{[2]} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$



Representing Relations Using Digraphs

We have shown that a relation can be represented by listing all of its ordered pairs or by using a zero–one matrix. There is another important way of representing a relation using a pictorial representation. Each element of the set is represented by a point, and each ordered pair is represented using an arc with its direction indicated by an arrow. We use such pictorial representations when we think of relations on a finite set as **directed graphs**, or **digraphs**.

DEFINITION 1

A *directed graph*, or *digraph*, consists of a set V of *vertices* (or *nodes*) together with a set E of ordered pairs of elements of V called *edges* (or *arcs*). The vertex a is called the *initial vertex* of the edge (a, b) , and the vertex b is called the *terminal vertex* of this edge.

An edge of the form (a, a) is represented using an arc from the vertex a back to itself. Such an edge is called a **loop**.

EXAMPLE 7

The directed graph with vertices a, b, c , and d , and edges $(a, b), (a, d), (b, b), (b, d), (c, a), (c, b)$, and (d, b) is displayed in Figure 3.

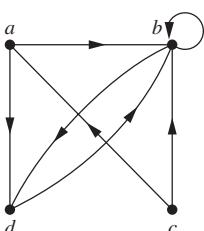


FIGURE 3
A Directed Graph.

The relation R on a set A is represented by the directed graph that has the elements of A as its vertices and the ordered pairs (a, b) , where $(a, b) \in R$, as edges. This assignment sets up a one-to-one correspondence between the relations on a set A and the directed graphs with A as their set of vertices. Thus, every statement about relations corresponds to a statement about directed graphs, and vice versa. Directed graphs give a visual display of information about relations. As such, they are often used to study relations and their properties. (Note that relations from a set A to a set B can be represented by a directed graph where there is a vertex for each element of A and a vertex for each element of B , as shown in Section 9.1. However, when $A = B$, such representation provides much less insight than the digraph representations described here.) The use of directed graphs to represent relations on a set is illustrated in Examples 8–10.

EXAMPLE 8 The directed graph of the relation

$$R = \{(1, 1), (1, 3), (2, 1), (2, 3), (2, 4), (3, 1), (3, 2), (4, 1)\}$$

on the set $\{1, 2, 3, 4\}$ is shown in Figure 4.

EXAMPLE 9 What are the ordered pairs in the relation R represented by the directed graph shown in Figure 5?

Solution: The ordered pairs (x, y) in the relation are

$$R = \{(1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (3, 1), (3, 3), (4, 1), (4, 3)\}.$$

Each of these pairs corresponds to an edge of the directed graph, with $(2, 2)$ and $(3, 3)$ corresponding to loops.

We will study directed graphs extensively in Chapter 10.

The directed graph representing a relation can be used to determine whether the relation has various properties. For instance, a relation is reflexive if and only if there is a loop at every vertex of the directed graph, so that every ordered pair of the form (x, x) occurs in the relation. A relation is symmetric if and only if for every edge between distinct vertices in its digraph there is an edge in the opposite direction, so that (y, x) is in the relation whenever (x, y) is in the relation. Similarly, a relation is antisymmetric if and only if there are never two edges in opposite directions between distinct vertices. Finally, a relation is transitive if and only if whenever there is an edge from a vertex x to a vertex y and an edge from a vertex y to a vertex z , there is an edge from x to z (completing a triangle where each side is a directed edge with the correct direction).

Remark: Note that a symmetric relation can be represented by an undirected graph, which is a graph where edges do not have directions. We will study undirected graphs in Chapter 10.

EXAMPLE 10 Determine whether the relations for the directed graphs shown in Figure 6 are reflexive, symmetric, antisymmetric, and/or transitive.

Solution: Because there are loops at every vertex of the directed graph of R , it is reflexive. R is neither symmetric nor antisymmetric because there is an edge from a to b but not one from b to a , but there are edges in both directions connecting b and c . Finally, R is not transitive because there is an edge from a to b and an edge from b to c , but no edge from a to c .

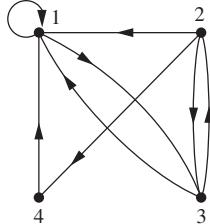


FIGURE 4 The Directed Graph of the Relation R .

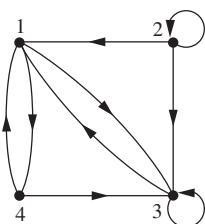


FIGURE 5 The Directed Graph of the Relation R .

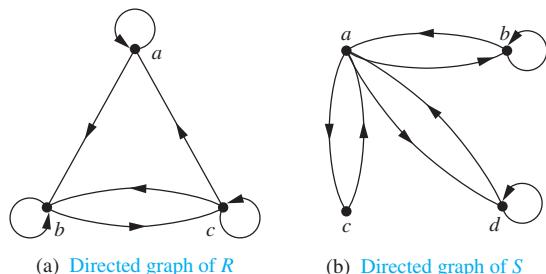


FIGURE 6 The Directed Graphs of the Relations R and S .

Because loops are not present at all the vertices of the directed graph of S , this relation is not reflexive. It is symmetric and not antisymmetric, because every edge between distinct vertices is accompanied by an edge in the opposite direction. It is also not hard to see from the directed graph that S is not transitive, because (c, a) and (a, b) belong to S , but (c, b) does not belong to S . 

Exercises

1. Represent each of these relations on $\{1, 2, 3\}$ with a matrix (with the elements of this set listed in increasing order).
 - a) $\{(1, 1), (1, 2), (1, 3)\}$
 - b) $\{(1, 2), (2, 1), (2, 2), (3, 3)\}$
 - c) $\{(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)\}$
 - d) $\{(1, 3), (3, 1)\}$
2. Represent each of these relations on $\{1, 2, 3, 4\}$ with a matrix (with the elements of this set listed in increasing order).
 - a) $\{(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\}$
 - b) $\{(1, 1), (1, 4), (2, 2), (3, 3), (4, 1)\}$
 - c) $\{(1, 2), (1, 3), (1, 4), (2, 1), (2, 3), (2, 4), (3, 1), (3, 2), (3, 4), (4, 1), (4, 2), (4, 3)\}$
 - d) $\{(2, 4), (3, 1), (3, 2), (3, 4)\}$
3. List the ordered pairs in the relations on $\{1, 2, 3\}$ corresponding to these matrices (where the rows and columns correspond to the integers listed in increasing order).

a) $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$	b) $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$
c) $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
4. List the ordered pairs in the relations on $\{1, 2, 3, 4\}$ corresponding to these matrices (where the rows and columns correspond to the integers listed in increasing order).

a) $\begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$	b) $\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$
c) $\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$	
5. How can the matrix representing a relation R on a set A be used to determine whether the relation is irreflexive?
6. How can the matrix representing a relation R on a set A be used to determine whether the relation is asymmetric?
7. Determine whether the relations represented by the matrices in Exercise 3 are reflexive, irreflexive, symmetric, antisymmetric, and/or transitive.
8. Determine whether the relations represented by the matrices in Exercise 4 are reflexive, irreflexive, symmetric, antisymmetric, and/or transitive.

9. How many nonzero entries does the matrix representing the relation R on $A = \{1, 2, 3, \dots, 100\}$ consisting of the first 100 positive integers have if R is
 - a) $\{(a, b) \mid a > b\}$
 - b) $\{(a, b) \mid a \neq b\}$
 - c) $\{(a, b) \mid a = b + 1\}$
 - d) $\{(a, b) \mid a = 1\}$
 - e) $\{(a, b) \mid ab = 1\}$
10. How many nonzero entries does the matrix representing the relation R on $A = \{1, 2, 3, \dots, 1000\}$ consisting of the first 1000 positive integers have if R is
 - a) $\{(a, b) \mid a \leq b\}$
 - b) $\{(a, b) \mid a = b \pm 1\}$
 - c) $\{(a, b) \mid a + b = 1000\}$
 - d) $\{(a, b) \mid a + b \leq 1001\}$
 - e) $\{(a, b) \mid a \neq 0\}$
11. How can the matrix for \bar{R} , the complement of the relation R , be found from the matrix representing R , when R is a relation on a finite set A ?
12. How can the matrix for R^{-1} , the inverse of the relation R , be found from the matrix representing R , when R is a relation on a finite set A ?
13. Let R be the relation represented by the matrix

$$\mathbf{M}_R = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$
 Find the matrix representing
 - a) R^{-1} .
 - b) \bar{R} .
 - c) R^2 .
14. Let R_1 and R_2 be relations on a set A represented by the matrices

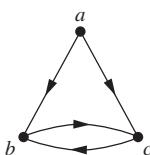
$$\mathbf{M}_{R_1} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{M}_{R_2} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$
 Find the matrices that represent
 - a) $R_1 \cup R_2$.
 - b) $R_1 \cap R_2$.
 - c) $R_2 \circ R_1$.
 - d) $R_1 \circ R_1$.
 - e) $R_1 \oplus R_2$.
15. Let R be the relation represented by the matrix

$$\mathbf{M}_R = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$
 Find the matrices that represent
 - a) R^2 .
 - b) R^3 .
 - c) R^4 .

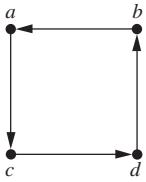
16. Let R be a relation on a set A with n elements. If there are k nonzero entries in \mathbf{M}_R , the matrix representing R , how many nonzero entries are there in $\mathbf{M}_{R^{-1}}$, the matrix representing R^{-1} , the inverse of R ?
17. Let R be a relation on a set A with n elements. If there are k nonzero entries in \mathbf{M}_R , the matrix representing R , how many nonzero entries are there in $\mathbf{M}_{\bar{R}}$, the matrix representing \bar{R} , the complement of R ?
18. Draw the directed graphs representing each of the relations from Exercise 1.
19. Draw the directed graphs representing each of the relations from Exercise 2.
20. Draw the directed graph representing each of the relations from Exercise 3.
21. Draw the directed graph representing each of the relations from Exercise 4.
22. Draw the directed graph that represents the relation $\{(a, a), (a, b), (b, c), (c, b), (c, d), (d, a), (d, b)\}$.

In Exercises 23–28 list the ordered pairs in the relations represented by the directed graphs.

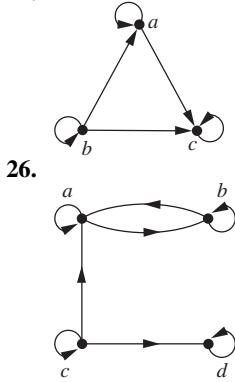
23.



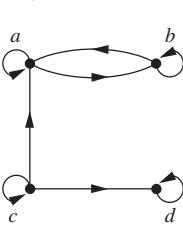
25.



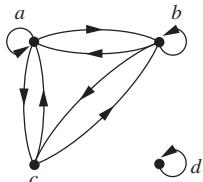
24.



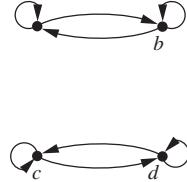
26.



27.



28.



29. How can the directed graph of a relation R on a finite set A be used to determine whether a relation is asymmetric?
30. How can the directed graph of a relation R on a finite set A be used to determine whether a relation is irreflexive?
31. Determine whether the relations represented by the directed graphs shown in Exercises 23–25 are reflexive, irreflexive, symmetric, antisymmetric, and/or transitive.
32. Determine whether the relations represented by the directed graphs shown in Exercises 26–28 are reflexive, irreflexive, symmetric, antisymmetric, asymmetric, and/or transitive.
33. Let R be a relation on a set A . Explain how to use the directed graph representing R to obtain the directed graph representing the inverse relation R^{-1} .
34. Let R be a relation on a set A . Explain how to use the directed graph representing R to obtain the directed graph representing the complementary relation \bar{R} .
35. Show that if \mathbf{M}_R is the matrix representing the relation R , then $\mathbf{M}_R^{[n]}$ is the matrix representing the relation R^n .
36. Given the directed graphs representing two relations, how can the directed graph of the union, intersection, symmetric difference, difference, and composition of these relations be found?

9.4 Closures of Relations

Introduction

A computer network has data centers in Boston, Chicago, Denver, Detroit, New York, and San Diego. There are direct, one-way telephone lines from Boston to Chicago, from Boston to Detroit, from Chicago to Detroit, from Detroit to Denver, and from New York to San Diego. Let R be the relation containing (a, b) if there is a telephone line from the data center in a to that in b . How can we determine if there is some (possibly indirect) link composed of one or more telephone lines from one center to another? Because not all links are direct, such as the link from Boston to Denver that goes through Detroit, R cannot be used directly to answer this. In the language of relations, R is not transitive, so it does not contain all the pairs that can be linked. As we will show in this section, we can find all pairs of data centers that have a link by constructing a transitive relation S containing R such that S is a subset of every transitive relation containing R . Here, S is the smallest transitive relation that contains R . This relation is called the **transitive closure** of R .

In general, let R be a relation on a set A . R may or may not have some property **P**, such as reflexivity, symmetry, or transitivity. If there is a relation S with property **P** containing R such that S is a subset of every relation with property **P** containing R , then S is called the **closure**

of R with respect to \mathbf{P} . (Note that the closure of a relation with respect to a property may not exist; see Exercises 15 and 35.) We will show how reflexive, symmetric, and transitive closures of relations can be found.

Closures

The relation $R = \{(1, 1), (1, 2), (2, 1), (3, 2)\}$ on the set $A = \{1, 2, 3\}$ is not reflexive. How can we produce a reflexive relation containing R that is as small as possible? This can be done by adding $(2, 2)$ and $(3, 3)$ to R , because these are the only pairs of the form (a, a) that are not in R . Clearly, this new relation contains R . Furthermore, *any* reflexive relation that contains R must also contain $(2, 2)$ and $(3, 3)$. Because this relation contains R , is reflexive, and is contained within every reflexive relation that contains R , it is called the **reflexive closure** of R .

As this example illustrates, given a relation R on a set A , the reflexive closure of R can be formed by adding to R all pairs of the form (a, a) with $a \in A$, not already in R . The addition of these pairs produces a new relation that is reflexive, contains R , and is contained within any reflexive relation containing R . We see that the reflexive closure of R equals $R \cup \Delta$, where $\Delta = \{(a, a) \mid a \in A\}$ is the **diagonal relation** on A . (The reader should verify this.)

EXAMPLE 1 What is the reflexive closure of the relation $R = \{(a, b) \mid a < b\}$ on the set of integers?

Solution: The reflexive closure of R is

$$R \cup \Delta = \{(a, b) \mid a < b\} \cup \{(a, a) \mid a \in \mathbf{Z}\} = \{(a, b) \mid a \leq b\}. \quad \blacktriangleleft$$

The relation $\{(1, 1), (1, 2), (2, 2), (2, 3), (3, 1), (3, 2)\}$ on $\{1, 2, 3\}$ is not symmetric. How can we produce a symmetric relation that is as small as possible and contains R ? To do this, we need only add $(2, 1)$ and $(1, 3)$, because these are the only pairs of the form (b, a) with $(a, b) \in R$ that are not in R . This new relation is symmetric and contains R . Furthermore, *any* symmetric relation that contains R must contain this new relation, because a symmetric relation that contains R must contain $(2, 1)$ and $(1, 3)$. Consequently, this new relation is called the **symmetric closure** of R .

As this example illustrates, the symmetric closure of a relation R can be constructed by adding all ordered pairs of the form (b, a) , where (a, b) is in the relation, that are not already present in R . Adding these pairs produces a relation that is symmetric, that contains R , and that is contained in any symmetric relation that contains R . The symmetric closure of a relation can be constructed by taking the union of a relation with its inverse (defined in the preamble of Exercise 26 in Section 9.1); that is, $R \cup R^{-1}$ is the symmetric closure of R , where $R^{-1} = \{(b, a) \mid (a, b) \in R\}$. The reader should verify this statement.

EXAMPLE 2 What is the symmetric closure of the relation $R = \{(a, b) \mid a > b\}$ on the set of positive integers?



Solution: The symmetric closure of R is the relation

$$R \cup R^{-1} = \{(a, b) \mid a > b\} \cup \{(b, a) \mid a > b\} = \{(a, b) \mid a \neq b\}.$$

This last equality follows because R contains all ordered pairs of positive integers where the first element is greater than the second element and R^{-1} contains all ordered pairs of positive integers where the first element is less than the second. \blacktriangleleft

Suppose that a relation R is not transitive. How can we produce a transitive relation that contains R such that this new relation is contained within any transitive relation that contains R ? Can the transitive closure of a relation R be produced by adding all the pairs of the form (a, c) , where (a, b) and (b, c) are already in the relation? Consider the relation

$R = \{(1, 3), (1, 4), (2, 1), (3, 2)\}$ on the set $\{1, 2, 3, 4\}$. This relation is not transitive because it does not contain all pairs of the form (a, c) where (a, b) and (b, c) are in R . The pairs of this form not in R are $(1, 2), (2, 3), (2, 4)$, and $(3, 1)$. Adding these pairs does *not* produce a transitive relation, because the resulting relation contains $(3, 1)$ and $(1, 4)$ but does not contain $(3, 4)$. This shows that constructing the transitive closure of a relation is more complicated than constructing either the reflexive or symmetric closure. The rest of this section develops algorithms for constructing transitive closures. As will be shown later in this section, the transitive closure of a relation can be found by adding new ordered pairs that must be present and then repeating this process until no new ordered pairs are needed.

Paths in Directed Graphs

We will see that representing relations by directed graphs helps in the construction of transitive closures. We now introduce some terminology that we will use for this purpose.

A path in a directed graph is obtained by traversing along edges (in the same direction as indicated by the arrow on the edge).

DEFINITION 1

A *path* from a to b in the directed graph G is a sequence of edges $(x_0, x_1), (x_1, x_2), (x_2, x_3), \dots, (x_{n-1}, x_n)$ in G , where n is a nonnegative integer, and $x_0 = a$ and $x_n = b$, that is, a sequence of edges where the terminal vertex of an edge is the same as the initial vertex in the next edge in the path. This path is denoted by $x_0, x_1, x_2, \dots, x_{n-1}, x_n$ and has *length* n . We view the empty set of edges as a path of length zero from a to a . A path of length $n \geq 1$ that begins and ends at the same vertex is called a *circuit* or *cycle*.

A path in a directed graph can pass through a vertex more than once. Moreover, an edge in a directed graph can occur more than once in a path.

EXAMPLE 3

Which of the following are paths in the directed graph shown in Figure 1: $a, b, e, d; a, e, c, d, b; b, a, c, b, a, a, b; d, c; c, b, a; e, b, a, b, a, b, e$? What are the lengths of those that are paths? Which of the paths in this list are circuits?

Solution: Because each of $(a, b), (b, e)$, and (e, d) is an edge, a, b, e, d is a path of length three. Because (c, d) is not an edge, a, e, c, d, b is not a path. Also, b, a, c, b, a, a, b is a path of length six because $(b, a), (a, c), (c, b), (b, a), (a, a)$, and (a, b) are all edges. We see that d, c is a path of length one, because (d, c) is an edge. Also c, b, a is a path of length two, because (c, b) and (b, a) are edges. All of $(e, b), (b, a), (a, b), (b, a), (a, b)$, and (b, e) are edges, so e, b, a, b, a, b, e is a path of length six.

The two paths b, a, c, b, a, a, b and e, b, a, b, a, b, e are circuits because they begin and end at the same vertex. The paths $a, b, e, d; c, b, a$; and d, c are not circuits. 

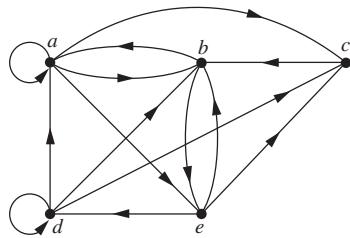


FIGURE 1 A Directed Graph.

The term *path* also applies to relations. Carrying over the definition from directed graphs to relations, there is a **path** from a to b in R if there is a sequence of elements $a, x_1, x_2, \dots, x_{n-1}, b$ with $(a, x_1) \in R$, $(x_1, x_2) \in R, \dots$, and $(x_{n-1}, b) \in R$. Theorem 1 can be obtained from the definition of a path in a relation.

THEOREM 1

Let R be a relation on a set A . There is a path of length n , where n is a positive integer, from a to b if and only if $(a, b) \in R^n$.

Proof: We will use mathematical induction. By definition, there is a path from a to b of length one if and only if $(a, b) \in R$, so the theorem is true when $n = 1$.

Assume that the theorem is true for the positive integer n . This is the inductive hypothesis. There is a path of length $n + 1$ from a to b if and only if there is an element $c \in A$ such that there is a path of length one from a to c , so $(a, c) \in R$, and a path of length n from c to b , that is, $(c, b) \in R^n$. Consequently, by the inductive hypothesis, there is a path of length $n + 1$ from a to b if and only if there is an element c with $(a, c) \in R$ and $(c, b) \in R^n$. But there is such an element if and only if $(a, b) \in R^{n+1}$. Therefore, there is a path of length $n + 1$ from a to b if and only if $(a, b) \in R^{n+1}$. This completes the proof. \triangleleft

Transitive Closures

We now show that finding the transitive closure of a relation is equivalent to determining which pairs of vertices in the associated directed graph are connected by a path. With this in mind, we define a new relation.

DEFINITION 2

Let R be a relation on a set A . The *connectivity relation* R^* consists of the pairs (a, b) such that there is a path of length at least one from a to b in R .

Because R^n consists of the pairs (a, b) such that there is a path of length n from a to b , it follows that R^* is the union of all the sets R^n . In other words,

$$R^* = \bigcup_{n=1}^{\infty} R^n.$$

The connectivity relation is useful in many models.

EXAMPLE 4

Let R be the relation on the set of all people in the world that contains (a, b) if a has met b . What is R^n , where n is a positive integer greater than one? What is R^* ?

Solution: The relation R^2 contains (a, b) if there is a person c such that $(a, c) \in R$ and $(c, b) \in R$, that is, if there is a person c such that a has met c and c has met b . Similarly, R^n consists of those pairs (a, b) such that there are people x_1, x_2, \dots, x_{n-1} such that a has met x_1 , x_1 has met x_2, \dots , and x_{n-1} has met b .

The relation R^* contains (a, b) if there is a sequence of people, starting with a and ending with b , such that each person in the sequence has met the next person in the sequence. (There are many interesting conjectures about R^* . Do you think that this connectivity relation includes the pair with you as the first element and the president of Mongolia as the second element? We will use graphs to model this application in Chapter 10.) \triangleleft

EXAMPLE 5 Let R be the relation on the set of all subway stops in New York City that contains (a, b) if it is possible to travel from stop a to stop b without changing trains. What is R^n when n is a positive integer? What is R^* ?

Solution: The relation R^n contains (a, b) if it is possible to travel from stop a to stop b by making at most $n - 1$ changes of trains. The relation R^* consists of the ordered pairs (a, b) where it is possible to travel from stop a to stop b making as many changes of trains as necessary. (The reader should verify these statements.) \blacktriangleleft

EXAMPLE 6 Let R be the relation on the set of all states in the United States that contains (a, b) if state a and state b have a common border. What is R^n , where n is a positive integer? What is R^* ?

Solution: The relation R^n consists of the pairs (a, b) , where it is possible to go from state a to state b by crossing exactly n state borders. R^* consists of the ordered pairs (a, b) , where it is possible to go from state a to state b crossing as many borders as necessary. (The reader should verify these statements.) The only ordered pairs not in R^* are those containing states that are not connected to the continental United States (i.e., those pairs containing Alaska or Hawaii). \blacktriangleleft

Theorem 2 shows that the transitive closure of a relation and the associated connectivity relation are the same.

THEOREM 2

The transitive closure of a relation R equals the connectivity relation R^* .

Proof: Note that R^* contains R by definition. To show that R^* is the transitive closure of R we must also show that R^* is transitive and that $R^* \subseteq S$ whenever S is a transitive relation that contains R .

First, we show that R^* is transitive. If $(a, b) \in R^*$ and $(b, c) \in R^*$, then there are paths from a to b and from b to c in R . We obtain a path from a to c by starting with the path from a to b and following it with the path from b to c . Hence, $(a, c) \in R^*$. It follows that R^* is transitive.

Now suppose that S is a transitive relation containing R . Because S is transitive, S^n also is transitive (the reader should verify this) and $S^n \subseteq S$ (by Theorem 1 of Section 9.1). Furthermore, because

$$S^* = \bigcup_{k=1}^{\infty} S^k$$

and $S^k \subseteq S$, it follows that $S^* \subseteq S$. Now note that if $R \subseteq S$, then $R^* \subseteq S^*$, because any path in R is also a path in S . Consequently, $R^* \subseteq S^* \subseteq S$. Thus, any transitive relation that contains R must also contain R^* . Therefore, R^* is the transitive closure of R . \blacktriangleleft

Now that we know that the transitive closure equals the connectivity relation, we turn our attention to the problem of computing this relation. We do not need to examine arbitrarily long paths to determine whether there is a path between two vertices in a finite directed graph. As Lemma 1 shows, it is sufficient to examine paths containing no more than n edges, where n is the number of elements in the set.

LEMMA 1

Let A be a set with n elements, and let R be a relation on A . If there is a path of length at least one in R from a to b , then there is such a path with length not exceeding n . Moreover, when $a \neq b$, if there is a path of length at least one in R from a to b , then there is such a path with length not exceeding $n - 1$.

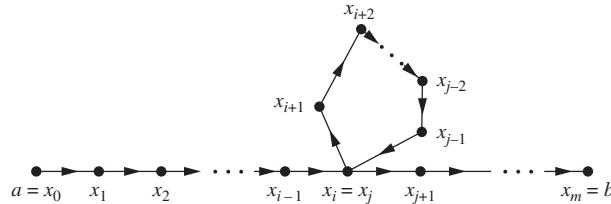


FIGURE 2 Producing a Path with Length Not Exceeding n .

Proof: Suppose there is a path from a to b in R . Let m be the length of the shortest such path. Suppose that $x_0, x_1, x_2, \dots, x_{m-1}, x_m$, where $x_0 = a$ and $x_m = b$, is such a path.

Suppose that $a = b$ and that $m > n$, so that $m \geq n + 1$. By the pigeonhole principle, because there are n vertices in A , among the m vertices x_0, x_1, \dots, x_{m-1} , at least two are equal (see Figure 2).

Suppose that $x_i = x_j$ with $0 \leq i < j \leq m - 1$. Then the path contains a circuit from x_i to itself. This circuit can be deleted from the path from a to b , leaving a path, namely, $x_0, x_1, \dots, x_i, x_{j+1}, \dots, x_{m-1}, x_m$, from a to b of shorter length. Hence, the path of shortest length must have length less than or equal to n .

The case where $a \neq b$ is left as an exercise for the reader. \triangleleft

From Lemma 1, we see that the transitive closure of R is the union of R , R^2 , R^3 , ..., and R^n . This follows because there is a path in R^* between two vertices if and only if there is a path between these vertices in R^i , for some positive integer i with $i \leq n$. Because

$$R^* = R \cup R^2 \cup R^3 \cup \dots \cup R^n$$

and the zero-one matrix representing a union of relations is the join of the zero-one matrices of these relations, the zero-one matrix for the transitive closure is the join of the zero-one matrices of the first n powers of the zero-one matrix of R .

THEOREM 3

Let \mathbf{M}_R be the zero-one matrix of the relation R on a set with n elements. Then the zero-one matrix of the transitive closure R^* is

$$\mathbf{M}_{R^*} = \mathbf{M}_R \vee \mathbf{M}_R^{[2]} \vee \mathbf{M}_R^{[3]} \vee \dots \vee \mathbf{M}_R^{[n]}.$$

EXAMPLE 7 Find the zero-one matrix of the transitive closure of the relation R where

$$\mathbf{M}_R = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}.$$

Solution: By Theorem 3, it follows that the zero-one matrix of R^* is

$$\mathbf{M}_{R^*} = \mathbf{M}_R \vee \mathbf{M}_R^{[2]} \vee \mathbf{M}_R^{[3]}.$$

Because

$$\mathbf{M}_R^{[2]} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{M}_R^{[3]} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix},$$

it follows that

$$\mathbf{M}_{R^*} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} \vee \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \vee \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}.$$



Theorem 3 can be used as a basis for an algorithm for computing the matrix of the relation R^* . To find this matrix, the successive Boolean powers of \mathbf{M}_R , up to the n th power, are computed. As each power is calculated, its join with the join of all smaller powers is formed. When this is done with the n th power, the matrix for R^* has been found. This procedure is displayed as Algorithm 1.

ALGORITHM 1 A Procedure for Computing the Transitive Closure.

```

procedure transitive closure ( $\mathbf{M}_R$  : zero–one  $n \times n$  matrix)
   $\mathbf{A} := \mathbf{M}_R$ 
   $\mathbf{B} := \mathbf{A}$ 
  for  $i := 2$  to  $n$ 
     $\mathbf{A} := \mathbf{A} \odot \mathbf{M}_R$ 
     $\mathbf{B} := \mathbf{B} \vee \mathbf{A}$ 
  return  $\mathbf{B}$ { $\mathbf{B}$  is the zero–one matrix for  $R^*$ }
```

We can easily find the number of bit operations used by Algorithm 1 to determine the transitive closure of a relation. Computing the Boolean powers $\mathbf{M}_R, \mathbf{M}_R^{[2]}, \dots, \mathbf{M}_R^{[n]}$ requires that $n - 1$ Boolean products of $n \times n$ zero–one matrices be found. Each of these Boolean products can be found using $n^2(2n - 1)$ bit operations. Hence, these products can be computed using $n^2(2n - 1)(n - 1)$ bit operations.

To find \mathbf{M}_{R^*} from the n Boolean powers of \mathbf{M}_R , $n - 1$ joins of zero–one matrices need to be found. Computing each of these joins uses n^2 bit operations. Hence, $(n - 1)n^2$ bit operations are used in this part of the computation. Therefore, when Algorithm 1 is used, the matrix of the transitive closure of a relation on a set with n elements can be found using $n^2(2n - 1)(n - 1) + (n - 1)n^2 = 2n^3(n - 1)$, which is $O(n^4)$ bit operations. The remainder of this section describes a more efficient algorithm for finding transitive closures.

Warshall's Algorithm



Warshall's algorithm, named after Stephen Warshall, who described this algorithm in 1960, is an efficient method for computing the transitive closure of a relation. Algorithm 1 can find the transitive closure of a relation on a set with n elements using $2n^3(n - 1)$ bit operations. However, the transitive closure can be found by Warshall's algorithm using only $2n^3$ bit operations.

Remark: Warshall's algorithm is sometimes called the Roy–Warshall algorithm, because Bernard Roy described this algorithm in 1959.

Suppose that R is a relation on a set with n elements. Let v_1, v_2, \dots, v_n be an arbitrary listing of these n elements. The concept of the **interior vertices** of a path is used in Warshall's algorithm. If $a, x_1, x_2, \dots, x_{m-1}, b$ is a path, its interior vertices are x_1, x_2, \dots, x_{m-1} , that is, all the vertices of the path that occur somewhere other than as the first and last vertices in the path. For instance, the interior vertices of a path a, c, d, f, g, h, b, j in a directed graph

are c, d, f, g, h , and b . The interior vertices of a, c, d, a, f, b are c, d, a , and f . (Note that the first vertex in the path is not an interior vertex unless it is visited again by the path, except as the last vertex. Similarly, the last vertex in the path is not an interior vertex unless it was visited previously by the path, except as the first vertex.)

Warshall's algorithm is based on the construction of a sequence of zero-one matrices. These matrices are $\mathbf{W}_0, \mathbf{W}_1, \dots, \mathbf{W}_n$, where $\mathbf{W}_0 = \mathbf{M}_R$ is the zero-one matrix of this relation, and $\mathbf{W}_k = [w_{ij}^{(k)}]$, where $w_{ij}^{(k)} = 1$ if there is a path from v_i to v_j such that all the interior vertices of this path are in the set $\{v_1, v_2, \dots, v_k\}$ (the first k vertices in the list) and is 0 otherwise. (The first and last vertices in the path may be outside the set of the first k vertices in the list.) Note that $\mathbf{W}_n = \mathbf{M}_{R^*}$, because the (i, j) th entry of \mathbf{M}_{R^*} is 1 if and only if there is a path from v_i to v_j , with all interior vertices in the set $\{v_1, v_2, \dots, v_n\}$ (but these are the only vertices in the directed graph). Example 8 illustrates what the matrix \mathbf{W}_k represents.

EXAMPLE 8

Let R be the relation with directed graph shown in Figure 3. Let a, b, c, d be a listing of the elements of the set. Find the matrices $\mathbf{W}_0, \mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3$, and \mathbf{W}_4 . The matrix \mathbf{W}_4 is the transitive closure of R .

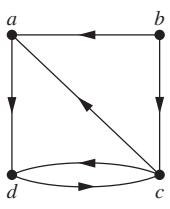


FIGURE 3

The Directed Graph of the Relation R .

Solution: Let $v_1 = a, v_2 = b, v_3 = c$, and $v_4 = d$. \mathbf{W}_0 is the matrix of the relation. Hence,

$$\mathbf{W}_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

\mathbf{W}_1 has 1 as its (i, j) th entry if there is a path from v_i to v_j that has only $v_1 = a$ as an interior vertex. Note that all paths of length one can still be used because they have no interior vertices. Also, there is now an allowable path from b to d , namely, b, a, d . Hence,

$$\mathbf{W}_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

\mathbf{W}_2 has 1 as its (i, j) th entry if there is a path from v_i to v_j that has only $v_1 = a$ and/or $v_2 = b$ as its interior vertices, if any. Because there are no edges that have b as a terminal vertex, no new paths are obtained when we permit b to be an interior vertex. Hence, $\mathbf{W}_2 = \mathbf{W}_1$.



STEPHEN WARSHALL (1935–2006) Stephen Warshall, born in New York City, went to public school in Brooklyn. He attended Harvard University, receiving his degree in mathematics in 1956. He never received an advanced degree, because at that time no programs were available in his areas of interest. However, he took graduate courses at several different universities and contributed to the development of computer science and software engineering.

After graduating from Harvard, Warshall worked at ORO (Operation Research Office), which was set up by Johns Hopkins to do research and development for the U.S. Army. In 1958 he left ORO to take a position at a company called Technical Operations, where he helped build a research and development laboratory for military software projects. In 1961 he left Technical Operations to found Massachusetts Computer Associates. Later, this company became part of Applied Data Research (ADR). After the merger, Warshall sat on the board of directors of ADR and managed a variety of projects and organizations. He retired from ADR in 1982.

During his career Warshall carried out research and development in operating systems, compiler design, language design, and operations research. In the 1971–1972 academic year he presented lectures on software engineering at French universities. There is an interesting anecdote about his proof that the transitive closure algorithm, now known as Warshall's algorithm, is correct. He and a colleague at Technical Operations bet a bottle of rum on who first could determine whether this algorithm always works. Warshall came up with his proof overnight, winning the bet and the rum, which he shared with the loser of the bet. Because Warshall did not like sitting at a desk, he did much of his creative work in unconventional places, such as on a sailboat in the Indian Ocean or in a Greek lemon orchard.

\mathbf{W}_3 has 1 as its (i, j) th entry if there is a path from v_i to v_j that has only $v_1 = a$, $v_2 = b$, and/or $v_3 = c$ as its interior vertices, if any. We now have paths from d to a , namely, d, c, a , and from d to d , namely, d, c, d . Hence,

$$\mathbf{W}_3 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}.$$

Finally, \mathbf{W}_4 has 1 as its (i, j) th entry if there is a path from v_i to v_j that has $v_1 = a$, $v_2 = b$, $v_3 = c$, and/or $v_4 = d$ as interior vertices, if any. Because these are all the vertices of the graph, this entry is 1 if and only if there is a path from v_i to v_j . Hence,

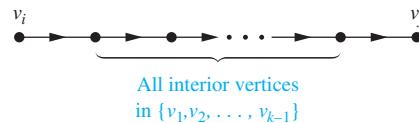
$$\mathbf{W}_4 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}.$$

This last matrix, \mathbf{W}_4 , is the matrix of the transitive closure. ◀

Warshall's algorithm computes \mathbf{M}_{R^*} by efficiently computing $\mathbf{W}_0 = \mathbf{M}_R$, \mathbf{W}_1 , \mathbf{W}_2, \dots , $\mathbf{W}_n = \mathbf{M}_{R^*}$. This observation shows that we can compute \mathbf{W}_k directly from \mathbf{W}_{k-1} : There is a path from v_i to v_j with no vertices other than v_1, v_2, \dots, v_k as interior vertices if and only if either there is a path from v_i to v_j with its interior vertices among the first $k - 1$ vertices in the list, or there are paths from v_i to v_k and from v_k to v_j that have interior vertices only among the first $k - 1$ vertices in the list. That is, either a path from v_i to v_j already existed before v_k was permitted as an interior vertex, or allowing v_k as an interior vertex produces a path that goes from v_i to v_k and then from v_k to v_j . These two cases are shown in Figure 4.

The first type of path exists if and only if $w_{ij}^{[k-1]} = 1$, and the second type of path exists if and only if both $w_{ik}^{[k-1]}$ and $w_{kj}^{[k-1]}$ are 1. Hence, $w_{ij}^{[k]}$ is 1 if and only if either $w_{ij}^{[k-1]}$ is 1 or both $w_{ik}^{[k-1]}$ and $w_{kj}^{[k-1]}$ are 1. This gives us Lemma 2.

Case 1



Case 2

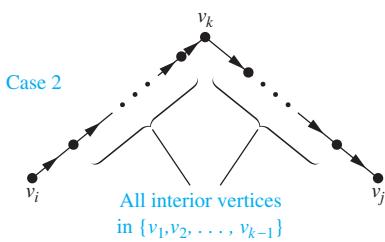


FIGURE 4 Adding v_k to the Set of Allowable Interior Vertices.

LEMMA 2

Let $\mathbf{W}_k = [w_{ij}^{[k]}]$ be the zero–one matrix that has a 1 in its (i, j) th position if and only if there is a path from v_i to v_j with interior vertices from the set $\{v_1, v_2, \dots, v_k\}$. Then

$$w_{ij}^{[k]} = w_{ij}^{[k-1]} \vee (w_{ik}^{[k-1]} \wedge w_{kj}^{[k-1]}),$$

whenever i, j , and k are positive integers not exceeding n .

Lemma 2 gives us the means to compute efficiently the matrices \mathbf{W}_k , $k = 1, 2, \dots, n$. We display the pseudocode for Warshall's algorithm, using Lemma 2, as Algorithm 2.

ALGORITHM 2 Warshall Algorithm.

```

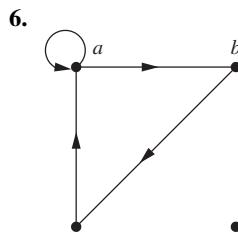
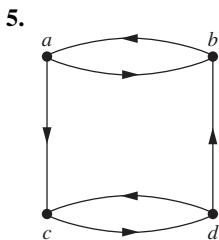
procedure Warshall ( $\mathbf{M}_R : n \times n$  zero–one matrix)
   $\mathbf{W} := \mathbf{M}_R$ 
  for  $k := 1$  to  $n$ 
    for  $i := 1$  to  $n$ 
      for  $j := 1$  to  $n$ 
         $w_{ij} := w_{ij} \vee (w_{ik} \wedge w_{kj})$ 
  return  $\mathbf{W}$  { $\mathbf{W} = [w_{ij}]$  is  $\mathbf{M}_{R^*}$ }
```

The computational complexity of Warshall's algorithm can easily be computed in terms of bit operations. To find the entry $w_{ij}^{[k]}$ from the entries $w_{ij}^{[k-1]}$, $w_{ik}^{[k-1]}$, and $w_{kj}^{[k-1]}$ using Lemma 2 requires two bit operations. To find all n^2 entries of \mathbf{W}_k from those of \mathbf{W}_{k-1} requires $2n^2$ bit operations. Because Warshall's algorithm begins with $\mathbf{W}_0 = \mathbf{M}_R$ and computes the sequence of n zero–one matrices $\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_n = \mathbf{M}_{R^*}$, the total number of bit operations used is $n \cdot 2n^2 = 2n^3$.

Exercises

1. Let R be the relation on the set $\{0, 1, 2, 3\}$ containing the ordered pairs $(0, 1), (1, 1), (1, 2), (2, 0), (2, 2)$, and $(3, 0)$. Find the
 - a) reflexive closure of R .
 - b) symmetric closure of R .
2. Let R be the relation $\{(a, b) \mid a \neq b\}$ on the set of integers. What is the reflexive closure of R ?
3. Let R be the relation $\{(a, b) \mid a \text{ divides } b\}$ on the set of integers. What is the symmetric closure of R ?
4. How can the directed graph representing the reflexive closure of a relation on a finite set be constructed from the directed graph of the relation?

In Exercises 5–7 draw the directed graph of the reflexive closure of the relations with the directed graph shown.



- 7.
8. How can the directed graph representing the symmetric closure of a relation on a finite set be constructed from the directed graph for this relation?
9. Find the directed graphs of the symmetric closures of the relations with directed graphs shown in Exercises 5–7.
10. Find the smallest relation containing the relation in Example 2 that is both reflexive and symmetric.
11. Find the directed graph of the smallest relation that is both reflexive and symmetric that contains each of the relations with directed graphs shown in Exercises 5–7.
12. Suppose that the relation R on the finite set A is represented by the matrix \mathbf{M}_R . Show that the matrix that represents the reflexive closure of R is $\mathbf{M}_R \vee \mathbf{I}_n$.

13. Suppose that the relation R on the finite set A is represented by the matrix \mathbf{M}_R . Show that the matrix that represents the symmetric closure of R is $\mathbf{M}_R \vee \mathbf{M}_R^t$.
14. Show that the closure of a relation R with respect to a property \mathbf{P} , if it exists, is the intersection of all the relations with property \mathbf{P} that contain R .
15. When is it possible to define the “irreflexive closure” of a relation R , that is, a relation that contains R , is irreflexive, and is contained in every irreflexive relation that contains R ?
16. Determine whether these sequences of vertices are paths in this directed graph.
- a) a, b, c, e
b) b, e, c, b, e
c) a, a, b, e, d, e
d) b, c, e, d, a, a, b
e) b, c, e, b, e, d, e, d
f) $a, a, b, b, c, c, b, e, d$
-
17. Find all circuits of length three in the directed graph in Exercise 16.
18. Determine whether there is a path in the directed graph in Exercise 16 beginning at the first vertex given and ending at the second vertex given.
- a) a, b
b) b, a
c) b, b
d) a, e
e) b, d
f) c, d
g) d, d
h) e, a
i) e, c
19. Let R be the relation on the set $\{1, 2, 3, 4, 5\}$ containing the ordered pairs $(1, 3), (2, 4), (3, 1), (3, 5), (4, 3), (5, 1), (5, 2)$, and $(5, 4)$. Find
- a) R^2 .
b) R^3 .
c) R^4 .
d) R^5 .
e) R^6 .
f) R^* .
20. Let R be the relation that contains the pair (a, b) if a and b are cities such that there is a direct non-stop airline flight from a to b . When is (a, b) in
- a) R^2 ?
b) R^3 ?
c) R^* ?
21. Let R be the relation on the set of all students containing the ordered pair (a, b) if a and b are in at least one common class and $a \neq b$. When is (a, b) in
- a) R^2 ?
b) R^3 ?
c) R^* ?
22. Suppose that the relation R is reflexive. Show that R^* is reflexive.
23. Suppose that the relation R is symmetric. Show that R^* is symmetric.
24. Suppose that the relation R is irreflexive. Is the relation R^2 necessarily irreflexive?

25. Use Algorithm 1 to find the transitive closures of these relations on $\{1, 2, 3, 4\}$.
- a) $\{(1, 2), (2, 1), (2, 3), (3, 4), (4, 1)\}$
b) $\{(2, 1), (2, 3), (3, 1), (3, 4), (4, 1), (4, 3)\}$
c) $\{(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\}$
d) $\{(1, 1), (1, 4), (2, 1), (2, 3), (3, 1), (3, 2), (3, 4), (4, 2)\}$
26. Use Algorithm 1 to find the transitive closures of these relations on $\{a, b, c, d, e\}$.
- a) $\{(a, c), (b, d), (c, a), (d, b), (e, d)\}$
b) $\{(b, c), (b, e), (c, e), (d, a), (e, b), (e, c)\}$
c) $\{(a, b), (a, c), (a, e), (b, a), (b, c), (c, a), (c, b), (d, a), (e, d)\}$
d) $\{(a, e), (b, a), (b, d), (c, d), (d, a), (d, c), (e, a), (e, b), (e, c), (e, e)\}$
27. Use Warshall's algorithm to find the transitive closures of the relations in Exercise 25.
28. Use Warshall's algorithm to find the transitive closures of the relations in Exercise 26.
29. Find the smallest relation containing the relation $\{(1, 2), (1, 4), (3, 3), (4, 1)\}$ that is
- a) reflexive and transitive.
b) symmetric and transitive.
c) reflexive, symmetric, and transitive.
30. Finish the proof of the case when $a \neq b$ in Lemma 1.
31. Algorithms have been devised that use $O(n^{2.8})$ bit operations to compute the Boolean product of two $n \times n$ zero-one matrices. Assuming that these algorithms can be used, give big- O estimates for the number of bit operations using Algorithm 1 and using Warshall's algorithm to find the transitive closure of a relation on a set with n elements.
- *32. Devise an algorithm using the concept of interior vertices in a path to find the length of the shortest path between two vertices in a directed graph, if such a path exists.
33. Adapt Algorithm 1 to find the reflexive closure of the transitive closure of a relation on a set with n elements.
34. Adapt Warshall's algorithm to find the reflexive closure of the transitive closure of a relation on a set with n elements.
35. Show that the closure with respect to the property \mathbf{P} of the relation $R = \{(0, 0), (0, 1), (1, 1), (2, 2)\}$ on the set $\{0, 1, 2\}$ does not exist if \mathbf{P} is the property
- a) “is not reflexive.”
b) “has an odd number of elements.”

9.5 Equivalence Relations

Introduction

In some programming languages the names of variables can contain an unlimited number of characters. However, there is a limit on the number of characters that are checked when a compiler determines whether two variables are equal. For instance, in traditional C, only the first eight characters of a variable name are checked by the compiler. (These characters are

uppercase or lowercase letters, digits, or underscores.) Consequently, the compiler considers strings longer than eight characters that agree in their first eight characters the same. Let R be the relation on the set of strings of characters such that sRt , where s and t are two strings, if s and t are at least eight characters long and the first eight characters of s and t agree, or $s = t$. It is easy to see that R is reflexive, symmetric, and transitive. Moreover, R divides the set of all strings into classes, where all strings in a particular class are considered the same by a compiler for traditional C.

The integers a and b are related by the “congruence modulo 4” relation when 4 divides $a - b$. We will show later that this relation is reflexive, symmetric, and transitive. It is not hard to see that a is related to b if and only if a and b have the same remainder when divided by 4. It follows that this relation splits the set of integers into four different classes. When we care only what remainder an integer leaves when it is divided by 4, we need only know which class it is in, not its particular value.

These two relations, R and congruence modulo 4, are examples of equivalence relations, namely, relations that are reflexive, symmetric, and transitive. In this section we will show that such relations split sets into disjoint classes of equivalent elements. Equivalence relations arise whenever we care only whether an element of a set is in a certain class of elements, instead of caring about its particular identity.

Equivalence Relations



In this section we will study relations with a particular combination of properties that allows them to be used to relate objects that are similar in some way.

DEFINITION 1

Equivalence relations are important in every branch of mathematics!

A relation on a set A is called an *equivalence relation* if it is reflexive, symmetric, and transitive.

Equivalence relations are important throughout mathematics and computer science. One reason for this is that in an equivalence relation, when two elements are related it makes sense to say they are equivalent.

DEFINITION 2

Two elements a and b that are related by an equivalence relation are called *equivalent*. The notation $a \sim b$ is often used to denote that a and b are equivalent elements with respect to a particular equivalence relation.

For the notion of equivalent elements to make sense, every element should be equivalent to itself, as the reflexive property guarantees for an equivalence relation. It makes sense to say that a and b are related (not just that a is related to b) by an equivalence relation, because when a is related to b , by the symmetric property, b is related to a . Furthermore, because an equivalence relation is transitive, if a and b are equivalent and b and c are equivalent, it follows that a and c are equivalent.

Examples 1–5 illustrate the notion of an equivalence relation.

EXAMPLE 1

Let R be the relation on the set of integers such that aRb if and only if $a = b$ or $a = -b$. In Section 9.1 we showed that R is reflexive, symmetric, and transitive. It follows that R is an equivalence relation.

EXAMPLE 2

Let R be the relation on the set of real numbers such that aRb if and only if $a - b$ is an integer. Is R an equivalence relation?

Extra Examples

Solution: Because $a - a = 0$ is an integer for all real numbers a , aRa for all real numbers a . Hence, R is reflexive. Now suppose that aRb . Then $a - b$ is an integer, so $b - a$ is also an integer. Hence, bRa . It follows that R is symmetric. If aRb and bRc , then $a - b$ and $b - c$ are integers. Therefore, $a - c = (a - b) + (b - c)$ is also an integer. Hence, aRc . Thus, R is transitive. Consequently, R is an equivalence relation. \blacktriangleleft

One of the most widely used equivalence relations is congruence modulo m , where m is an integer greater than 1.

EXAMPLE 3 Congruence Modulo m Let m be an integer with $m > 1$. Show that the relation

$$R = \{(a, b) \mid a \equiv b \pmod{m}\}$$

is an equivalence relation on the set of integers.

Solution: Recall from Section 4.1 that $a \equiv b \pmod{m}$ if and only if m divides $a - b$. Note that $a - a = 0$ is divisible by m , because $0 = 0 \cdot m$. Hence, $a \equiv a \pmod{m}$, so congruence modulo m is reflexive. Now suppose that $a \equiv b \pmod{m}$. Then $a - b$ is divisible by m , so $a - b = km$, where k is an integer. It follows that $b - a = (-k)m$, so $b \equiv a \pmod{m}$. Hence, congruence modulo m is symmetric. Next, suppose that $a \equiv b \pmod{m}$ and $b \equiv c \pmod{m}$. Then m divides both $a - b$ and $b - c$. Therefore, there are integers k and l with $a - b = km$ and $b - c = lm$. Adding these two equations shows that $a - c = (a - b) + (b - c) = km + lm = (k + l)m$. Thus, $a \equiv c \pmod{m}$. Therefore, congruence modulo m is transitive. It follows that congruence modulo m is an equivalence relation. \blacktriangleleft

EXAMPLE 4 Suppose that R is the relation on the set of strings of English letters such that aRb if and only if $l(a) = l(b)$, where $l(x)$ is the length of the string x . Is R an equivalence relation?

Solution: Because $l(a) = l(a)$, it follows that aRa whenever a is a string, so that R is reflexive. Next, suppose that aRb , so that $l(a) = l(b)$. Then bRa , because $l(b) = l(a)$. Hence, R is symmetric. Finally, suppose that aRb and bRc . Then $l(a) = l(b)$ and $l(b) = l(c)$. Hence, $l(a) = l(c)$, so aRc . Consequently, R is transitive. Because R is reflexive, symmetric, and transitive, it is an equivalence relation. \blacktriangleleft

EXAMPLE 5 Let n be a positive integer and S a set of strings. Suppose that R_n is the relation on S such that $sR_n t$ if and only if $s = t$, or both s and t have at least n characters and the first n characters of s and t are the same. That is, a string of fewer than n characters is related only to itself; a string s with at least n characters is related to a string t if and only if t has at least n characters and t begins with the n characters at the start of s . For example, let $n = 3$ and let S be the set of all bit strings. Then $sR_3 t$ either when $s = t$ or both s and t are bit strings of length 3 or more that begin with the same three bits. For instance, $01R_3 01$ and $00111R_3 00101$, but $01R_3 010$ and $01011R_3 01110$.

Show that for every set S of strings and every positive integer n , R_n is an equivalence relation on S .

Solution: The relation R_n is reflexive because $s = s$, so that $sR_n s$ whenever s is a string in S . If $sR_n t$, then either $s = t$ or s and t are both at least n characters long that begin with the same n characters. This means that $tR_n s$. We conclude that R_n is symmetric.

Now suppose that $sR_n t$ and $tR_n u$. Then either $s = t$ or s and t are at least n characters long and s and t begin with the same n characters, and either $t = u$ or t and u are at least n characters long and t and u begin with the same n characters. From this, we can deduce that either $s = u$ or both s and u are n characters long and s and u begin with the same n characters (because in this case we know that s , t , and u are all at least n characters long and both s and u begin with the same n characters as t does). Consequently, R_n is transitive. It follows that R_n is an equivalence relation. \blacktriangleleft

In Examples 6 and 7 we look at two relations that are not equivalence relations.

EXAMPLE 6 Show that the “divides” relation is the set of positive integers is not an equivalence relation.

Solution: By Examples 9 and 15 in Section 9.1, we know that the “divides” relation is reflexive and transitive. However, by Example 12 in Section 9.1, we know that this relation is not symmetric (for instance, $2 \mid 4$ but $4 \nmid 2$). We conclude that the “divides” relation on the set of positive integers is not an equivalence relation. 

EXAMPLE 7 Let R be the relation on the set of real numbers such that $x R y$ if and only if x and y are real numbers that differ by less than 1, that is $|x - y| < 1$. Show that R is not an equivalence relation.

Solution: R is reflexive because $|x - x| = 0 < 1$ whenever $x \in \mathbf{R}$. R is symmetric, for if $x R y$, where x and y are real numbers, then $|x - y| < 1$, which tells us that $|y - x| = |x - y| < 1$, so that $y R x$. However, R is not an equivalence relation because it is not transitive. Take $x = 2.8$, $y = 1.9$, and $z = 1.1$, so that $|x - y| = |2.8 - 1.9| = 0.9 < 1$, $|y - z| = |1.9 - 1.1| = 0.8 < 1$, but $|x - z| = |2.8 - 1.1| = 1.7 > 1$. That is, $2.8 R 1.9$, $1.9 R 1.1$, but $2.8 \not R 1.1$. 

Equivalence Classes

Let A be the set of all students in your school who graduated from high school. Consider the relation R on A that consists of all pairs (x, y) , where x and y graduated from the same high school. Given a student x , we can form the set of all students equivalent to x with respect to R . This set consists of all students who graduated from the same high school as x did. This subset of A is called an equivalence class of the relation.

DEFINITION 3

Let R be an equivalence relation on a set A . The set of all elements that are related to an element a of A is called the *equivalence class* of a . The equivalence class of a with respect to R is denoted by $[a]_R$. When only one relation is under consideration, we can delete the subscript R and write $[a]$ for this equivalence class.

In other words, if R is an equivalence relation on a set A , the equivalence class of the element a is

$$[a]_R = \{s \mid (a, s) \in R\}.$$

If $b \in [a]_R$, then b is called a **representative** of this equivalence class. Any element of a class can be used as a representative of this class. That is, there is nothing special about the particular element chosen as the representative of the class.

EXAMPLE 8 What is the equivalence class of an integer for the equivalence relation of Example 1?

Solution: Because an integer is equivalent to itself and its negative in this equivalence relation, it follows that $[a] = \{-a, a\}$. This set contains two distinct integers unless $a = 0$. For instance, $[7] = \{-7, 7\}$, $[-5] = \{-5, 5\}$, and $[0] = \{0\}$. 

EXAMPLE 9 What are the equivalence classes of 0 and 1 for congruence modulo 4?

Solution: The equivalence class of 0 contains all integers a such that $a \equiv 0 \pmod{4}$. The integers in this class are those divisible by 4. Hence, the equivalence class of 0 for this relation is

$$[0] = \{\dots, -8, -4, 0, 4, 8, \dots\}.$$

The equivalence class of 1 contains all the integers a such that $a \equiv 1 \pmod{4}$. The integers in this class are those that have a remainder of 1 when divided by 4. Hence, the equivalence class of 1 for this relation is

$$[1] = \{\dots, -7, -3, 1, 5, 9, \dots\}.$$

In Example 9 the equivalence classes of 0 and 1 with respect to congruence modulo 4 were found. Example 9 can easily be generalized, replacing 4 with any positive integer m . The equivalence classes of the relation congruence modulo m are called the **congruence classes modulo m** . The congruence class of an integer a modulo m is denoted by $[a]_m$, so $[a]_m = \{\dots, a - 2m, a - m, a, a + m, a + 2m, \dots\}$. For instance, from Example 9 it follows that $[0]_4 = \{\dots, -8, -4, 0, 4, 8, \dots\}$ and $[1]_4 = \{\dots, -7, -3, 1, 5, 9, \dots\}$.

EXAMPLE 10 What is the equivalence class of the string 0111 with respect to the equivalence relation R_3 from Example 5 on the set of all bit strings? (Recall that $s R_3 t$ if and only if s and t are bit strings with $s = t$ or s and t are strings of at least three bits that start with the same three bits.)

Solution: The bit strings equivalent to 0111 are the bit strings with at least three bits that begin with 011. These are the bit strings 011, 0110, 0111, 01100, 01101, 01110, 01111, and so on. Consequently,

$$[011]_{R_3} = \{011, 0110, 0111, 01100, 01101, 01110, 01111, \dots\}.$$

EXAMPLE 11 **Identifiers in the C Programming Language** In the C programming language, an **identifier** is the name of a variable, a function, or another type of entity. Each identifier is a nonempty string of characters where each character is a lowercase or an uppercase English letter, a digit, or an underscore, and the first character is a lowercase or an uppercase English letter. Identifiers can be any length. This allows developers to use as many characters as they want to name an entity, such as a variable. However, for compilers for some versions of C, there is a limit on the number of characters checked when two names are compared to see whether they refer to the same thing. For example, Standard C compilers consider two identifiers the same when they agree in their first 31 characters. Consequently, developers must be careful not to use identifiers with the same initial 31 characters for different things. We see that two identifiers are considered the same when they are related by the relation R_{31} in Example 5. Using Example 5, we know that R_{31} , on the set of all identifiers in Standard C, is an equivalence relation.

What are the equivalence classes of each of the identifiers `Number_of_tropical_storms`, `Number_of_named_tropical_storms`, and `Number_of_named_tropical_storms_in_the_Atlantic_in_2005`?

Solution: Note that when an identifier is less than 31 characters long, by the definition of R_{31} , its equivalence class contains only itself. Because the identifier `Number_of_tropical_storms` is 25 characters long, its equivalence class contains exactly one element, namely, itself.

The identifier `Number_of_named_tropical_storms` is exactly 31 characters long. An identifier is equivalent to it when it starts with these same 31 characters. Consequently, every identifier at least 31 characters long that starts with `Number_of_named_tropical_storms` is equivalent to this identifier. It follows that the equivalence class of `Number_of_named_tropical_storms` is the set of all identifiers that begin with the 31 characters `Number_of_named_tropical_storms`.

An identifier is equivalent to the `Number_of_named_tropical_storms_in_the_Atlantic_in_2005` if and only if it begins with its first 31 characters. Because these characters are `Number_of_named_tropical_storms`, we see that an identifier is equivalent to `Number_of_named_tropical_storms_in_the_Atlantic_in_2005` if and only if it is equivalent to `Number_of_named_tropical_storms`. It follows that these last two identifiers have the same equivalence class.

Equivalence Classes and Partitions

Let A be the set of students at your school who are majoring in exactly one subject, and let R be the relation on A consisting of pairs (x, y) , where x and y are students with the same major. Then R is an equivalence relation, as the reader should verify. We can see that R splits all students in A into a collection of disjoint subsets, where each subset contains students with a specified major. For instance, one subset contains all students majoring (just) in computer science, and a second subset contains all students majoring in history. Furthermore, these subsets are equivalence classes of R . This example illustrates how the equivalence classes of an equivalence relation partition a set into disjoint, nonempty subsets. We will make these notions more precise in the following discussion.

Let R be a relation on the set A . Theorem 1 shows that the equivalence classes of two elements of A are either identical or disjoint.

THEOREM 1

Let R be an equivalence relation on a set A . These statements for elements a and b of A are equivalent:

- (i) aRb
- (ii) $[a] = [b]$
- (iii) $[a] \cap [b] \neq \emptyset$

Proof: We first show that (i) implies (ii). Assume that aRb . We will prove that $[a] = [b]$ by showing $[a] \subseteq [b]$ and $[b] \subseteq [a]$. Suppose $c \in [a]$. Then aRc . Because aRb and R is symmetric, we know that bRa . Furthermore, because R is transitive and bRa and aRc , it follows that bRc . Hence, $c \in [b]$. This shows that $[a] \subseteq [b]$. The proof that $[b] \subseteq [a]$ is similar; it is left as an exercise for the reader.

Second, we will show that (ii) implies (iii). Assume that $[a] = [b]$. It follows that $[a] \cap [b] \neq \emptyset$ because $[a]$ is nonempty (because $a \in [a]$ because R is reflexive).

Next, we will show that (iii) implies (i). Suppose that $[a] \cap [b] \neq \emptyset$. Then there is an element c with $c \in [a]$ and $c \in [b]$. In other words, aRc and bRc . By the symmetric property, cRb . Then by transitivity, because aRc and cRb , we have aRb .

Because (i) implies (ii), (ii) implies (iii), and (iii) implies (i), the three statements, (i), (ii), and (iii), are equivalent. \triangleleft

We are now in a position to show how an equivalence relation *partitions* a set. Let R be an equivalence relation on a set A . The union of the equivalence classes of R is all of A , because an element a of A is in its own equivalence class, namely, $[a]_R$. In other words,

$$\bigcup_{a \in A} [a]_R = A.$$

In addition, from Theorem 1, it follows that these equivalence classes are either equal or disjoint, so

$$[a]_R \cap [b]_R = \emptyset,$$

when $[a]_R \neq [b]_R$.

These two observations show that the equivalence classes form a partition of A , because they split A into disjoint subsets. More precisely, a **partition** of a set S is a collection of disjoint nonempty subsets of S that have S as their union. In other words, the collection of subsets A_i , $i \in I$ (where I is an index set) forms a partition of S if and only if

$$A_i \neq \emptyset \text{ for } i \in I,$$

$$A_i \cap A_j = \emptyset \text{ when } i \neq j,$$

Recall that an *index set* is a set whose members label, or index, the elements of a set.

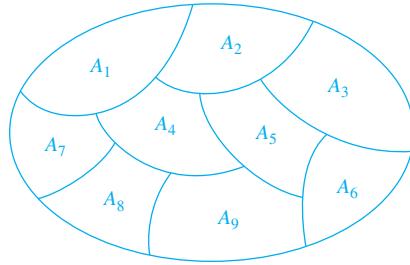


FIGURE 1 A Partition of a Set.

and

$$\bigcup_{i \in I} A_i = S.$$

(Here the notation $\bigcup_{i \in I} A_i$ represents the union of the sets A_i for all $i \in I$.) Figure 1 illustrates the concept of a partition of a set.

EXAMPLE 12 Suppose that $S = \{1, 2, 3, 4, 5, 6\}$. The collection of sets $A_1 = \{1, 2, 3\}$, $A_2 = \{4, 5\}$, and $A_3 = \{6\}$ forms a partition of S , because these sets are disjoint and their union is S . ◀

We have seen that the equivalence classes of an equivalence relation on a set form a partition of the set. The subsets in this partition are the equivalence classes. Conversely, every partition of a set can be used to form an equivalence relation. Two elements are equivalent with respect to this relation if and only if they are in the same subset of the partition.

To see this, assume that $\{A_i \mid i \in I\}$ is a partition on S . Let R be the relation on S consisting of the pairs (x, y) , where x and y belong to the same subset A_i in the partition. To show that R is an equivalence relation we must show that R is reflexive, symmetric, and transitive.

We see that $(a, a) \in R$ for every $a \in S$, because a is in the same subset as itself. Hence, R is reflexive. If $(a, b) \in R$, then a and b are in the same subset of the partition, so that $(b, a) \in R$ as well. Hence, R is symmetric. If $(a, b) \in R$ and $(b, c) \in R$, then a and b are in the same subset X in the partition, and b and c are in the same subset Y of the partition. Because the subsets of the partition are disjoint and b belongs to X and Y , it follows that $X = Y$. Consequently, a and c belong to the same subset of the partition, so $(a, c) \in R$. Thus, R is transitive.

It follows that R is an equivalence relation. The equivalence classes of R consist of subsets of S containing related elements, and by the definition of R , these are the subsets of the partition. Theorem 2 summarizes the connections we have established between equivalence relations and partitions.

THEOREM 2

Let R be an equivalence relation on a set S . Then the equivalence classes of R form a partition of S . Conversely, given a partition $\{A_i \mid i \in I\}$ of the set S , there is an equivalence relation R that has the sets A_i , $i \in I$, as its equivalence classes.

Example 13 shows how to construct an equivalence relation from a partition.

EXAMPLE 13 List the ordered pairs in the equivalence relation R produced by the partition $A_1 = \{1, 2, 3\}$, $A_2 = \{4, 5\}$, and $A_3 = \{6\}$ of $S = \{1, 2, 3, 4, 5, 6\}$, given in Example 12.

Solution: The subsets in the partition are the equivalence classes of R . The pair $(a, b) \in R$ if and only if a and b are in the same subset of the partition. The pairs $(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2)$, and $(3, 3)$ belong to R because $A_1 = \{1, 2, 3\}$ is an equivalence class; the pairs $(4, 4), (4, 5), (5, 4)$, and $(5, 5)$ belong to R because $A_2 = \{4, 5\}$ is an equivalence class; and finally the pair $(6, 6)$ belongs to R because $\{6\}$ is an equivalence class. No pair other than those listed belongs to R . 

The congruence classes modulo m provide a useful illustration of Theorem 2. There are m different congruence classes modulo m , corresponding to the m different remainders possible when an integer is divided by m . These m congruence classes are denoted by $[0]_m, [1]_m, \dots, [m-1]_m$. They form a partition of the set of integers.

EXAMPLE 14 What are the sets in the partition of the integers arising from congruence modulo 4?

Solution: There are four congruence classes, corresponding to $[0]_4, [1]_4, [2]_4$, and $[3]_4$. They are the sets

$$\begin{aligned}[0]_4 &= \{\dots, -8, -4, 0, 4, 8, \dots\}, \\ [1]_4 &= \{\dots, -7, -3, 1, 5, 9, \dots\}, \\ [2]_4 &= \{\dots, -6, -2, 2, 6, 10, \dots\}, \\ [3]_4 &= \{\dots, -5, -1, 3, 7, 11, \dots\}.\end{aligned}$$

These congruence classes are disjoint, and every integer is in exactly one of them. In other words, as Theorem 2 says, these congruence classes form a partition. 

We now provide an example of a partition of the set of all strings arising from an equivalence relation on this set.

EXAMPLE 15 Let R_3 be the relation from Example 5. What are the sets in the partition of the set of all bit strings arising from the relation R_3 on the set of all bit strings? (Recall that sR_3t , where s and t are bit strings, if $s = t$ or s and t are bit strings with at least three bits that agree in their first three bits.)

Solution: Note that every bit string of length less than three is equivalent only to itself. Hence $[\lambda]_{R_3} = \{\lambda\}, [0]_{R_3} = \{0\}, [1]_{R_3} = \{1\}, [00]_{R_3} = \{00\}, [01]_{R_3} = \{01\}, [10]_{R_3} = \{10\}$, and $[11]_{R_3} = \{11\}$. Note that every bit string of length three or more is equivalent to one of the eight bit strings 000, 001, 010, 011, 100, 101, 110, and 111. We have

$$\begin{aligned}[000]_{R_3} &= \{000, 0000, 0001, 00000, 00001, 00010, 00011, \dots\}, \\ [001]_{R_3} &= \{001, 0010, 0011, 00100, 00101, 00110, 00111, \dots\}, \\ [010]_{R_3} &= \{010, 0100, 0101, 01000, 01001, 01010, 01011, \dots\}, \\ [011]_{R_3} &= \{011, 0110, 0111, 01100, 01101, 01110, 01111, \dots\}, \\ [100]_{R_3} &= \{100, 1000, 1001, 10000, 10001, 10010, 10011, \dots\}, \\ [101]_{R_3} &= \{101, 1010, 1011, 10100, 10101, 10110, 10111, \dots\}, \\ [110]_{R_3} &= \{110, 1100, 1101, 11000, 11001, 11010, 11011, \dots\}, \\ [111]_{R_3} &= \{111, 1110, 1111, 11100, 11101, 11110, 11111, \dots\}.\end{aligned}$$

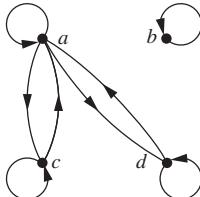
These 15 equivalence classes are disjoint and every bit string is in exactly one of them. As Theorem 2 tells us, these equivalence classes partition the set of all bit strings. 

Exercises

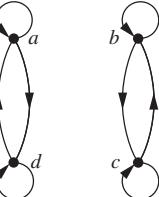
1. Which of these relations on $\{0, 1, 2, 3\}$ are equivalence relations? Determine the properties of an equivalence relation that the others lack.
 - a) $\{(0, 0), (1, 1), (2, 2), (3, 3)\}$
 - b) $\{(0, 0), (0, 2), (2, 0), (2, 2), (2, 3), (3, 2), (3, 3)\}$
 - c) $\{(0, 0), (1, 1), (1, 2), (2, 1), (2, 2), (3, 3)\}$
 - d) $\{(0, 0), (1, 1), (1, 3), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)\}$
 - e) $\{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 2), (3, 3)\}$
2. Which of these relations on the set of all people are equivalence relations? Determine the properties of an equivalence relation that the others lack.
 - a) $\{(a, b) \mid a \text{ and } b \text{ are the same age}\}$
 - b) $\{(a, b) \mid a \text{ and } b \text{ have the same parents}\}$
 - c) $\{(a, b) \mid a \text{ and } b \text{ share a common parent}\}$
 - d) $\{(a, b) \mid a \text{ and } b \text{ have met}\}$
 - e) $\{(a, b) \mid a \text{ and } b \text{ speak a common language}\}$
3. Which of these relations on the set of all functions from \mathbf{Z} to \mathbf{Z} are equivalence relations? Determine the properties of an equivalence relation that the others lack.
 - a) $\{(f, g) \mid f(1) = g(1)\}$
 - b) $\{(f, g) \mid f(0) = g(0) \text{ or } f(1) = g(1)\}$
 - c) $\{(f, g) \mid f(x) - g(x) = 1 \text{ for all } x \in \mathbf{Z}\}$
 - d) $\{(f, g) \mid \text{for some } C \in \mathbf{Z}, \text{ for all } x \in \mathbf{Z}, f(x) - g(x) = C\}$
 - e) $\{(f, g) \mid f(0) = g(1) \text{ and } f(1) = g(0)\}$
4. Define three equivalence relations on the set of students in your discrete mathematics class different from the relations discussed in the text. Determine the equivalence classes for each of these equivalence relations.
5. Define three equivalence relations on the set of buildings on a college campus. Determine the equivalence classes for each of these equivalence relations.
6. Define three equivalence relations on the set of classes offered at your school. Determine the equivalence classes for each of these equivalence relations.
7. Show that the relation of logical equivalence on the set of all compound propositions is an equivalence relation. What are the equivalence classes of \mathbf{F} and of \mathbf{T} ?
8. Let R be the relation on the set of all sets of real numbers such that $S R T$ if and only if S and T have the same cardinality. Show that R is an equivalence relation. What are the equivalence classes of the sets $\{0, 1, 2\}$ and \mathbf{Z} ?
9. Suppose that A is a nonempty set, and f is a function that has A as its domain. Let R be the relation on A consisting of all ordered pairs (x, y) such that $f(x) = f(y)$.
 - a) Show that R is an equivalence relation on A .
 - b) What are the equivalence classes of R ?
10. Suppose that A is a nonempty set and R is an equivalence relation on A . Show that there is a function f with A as its domain such that $(x, y) \in R$ if and only if $f(x) = f(y)$.
11. Show that the relation R consisting of all pairs (x, y) such that x and y are bit strings of length three or more that agree in their first three bits is an equivalence relation on the set of all bit strings of length three or more.
12. Show that the relation R consisting of all pairs (x, y) such that x and y are bit strings of length three or more that agree except perhaps in their first three bits is an equivalence relation on the set of all bit strings of length three or more.
13. Show that the relation R consisting of all pairs (x, y) such that x and y are bit strings that agree in their first and third bits is an equivalence relation on the set of all bit strings of length three or more.
14. Let R be the relation consisting of all pairs (x, y) such that x and y are strings of uppercase and lowercase English letters with the property that for every positive integer n , the n th characters in x and y are the same letter, either uppercase or lowercase. Show that R is an equivalence relation.
15. Let R be the relation on the set of ordered pairs of positive integers such that $((a, b), (c, d)) \in R$ if and only if $a + d = b + c$. Show that R is an equivalence relation.
16. Let R be the relation on the set of ordered pairs of positive integers such that $((a, b), (c, d)) \in R$ if and only if $ad = bc$. Show that R is an equivalence relation.
17. (Requires calculus)
 - a) Show that the relation R on the set of all differentiable functions from \mathbf{R} to \mathbf{R} consisting of all pairs (f, g) such that $f'(x) = g'(x)$ for all real numbers x is an equivalence relation.
 - b) Which functions are in the same equivalence class as the function $f(x) = x^2$?
18. (Requires calculus)
 - a) Let n be a positive integer. Show that the relation R on the set of all polynomials with real-valued coefficients consisting of all pairs (f, g) such that $f^{(n)}(x) = g^{(n)}(x)$ is an equivalence relation. [Here $f^{(n)}(x)$ is the n th derivative of $f(x)$.]
 - b) Which functions are in the same equivalence class as the function $f(x) = x^4$, where $n = 3$?
19. Let R be the relation on the set of all URLs (or Web addresses) such that $x R y$ if and only if the Web page at x is the same as the Web page at y . Show that R is an equivalence relation.
20. Let R be the relation on the set of all people who have visited a particular Web page such that $x R y$ if and only if person x and person y have followed the same set of links starting at this Web page (going from Web page to Web page until they stop using the Web). Show that R is an equivalence relation.

In Exercises 21–23 determine whether the relation with the directed graph shown is an equivalence relation.

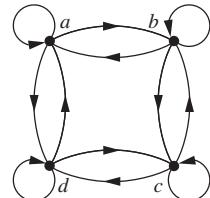
21.



22.



23.



24. Determine whether the relations represented by these zero-one matrices are equivalence relations.

a) $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

b) $\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$

c) $\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

25. Show that the relation R on the set of all bit strings such that $s R t$ if and only if s and t contain the same number of 1s is an equivalence relation.

26. What are the equivalence classes of the equivalence relations in Exercise 1?

27. What are the equivalence classes of the equivalence relations in Exercise 2?

28. What are the equivalence classes of the equivalence relations in Exercise 3?

29. What is the equivalence class of the bit string 011 for the equivalence relation in Exercise 25?

30. What are the equivalence classes of these bit strings for the equivalence relation in Exercise 11?

- a) 010 b) 1011 c) 11111 d) 01010101

31. What are the equivalence classes of the bit strings in Exercise 30 for the equivalence relation from Exercise 12?

32. What are the equivalence classes of the bit strings in Exercise 30 for the equivalence relation from Exercise 13?

33. What are the equivalence classes of the bit strings in Exercise 30 for the equivalence relation R_4 from Example 5 on the set of all bit strings? (Recall that bit strings s and t are equivalent under R_4 if and only if they are equal or they are both at least four bits long and agree in their first four bits.)

34. What are the equivalence classes of the bit strings in Exercise 30 for the equivalence relation R_5 from Example 5 on the set of all bit strings? (Recall that bit strings s and t are equivalent under R_5 if and only if they are equal or they are both at least five bits long and agree in their first five bits.)

35. What is the congruence class $[n]_5$ (that is, the equivalence class of n with respect to congruence modulo 5) when n is

- a) 2? b) 3? c) 6? d) -3?

36. What is the congruence class $[4]_m$ when m is

- a) 2? b) 3? c) 6? d) 8?

37. Give a description of each of the congruence classes modulo 6.

38. What is the equivalence class of each of these strings with respect to the equivalence relation in Exercise 14?

- a) No b) Yes c) Help

39. a) What is the equivalence class of (1, 2) with respect to the equivalence relation in Exercise 15?

b) Give an interpretation of the equivalence classes for the equivalence relation R in Exercise 15. [Hint: Look at the difference $a - b$ corresponding to (a, b) .]

40. a) What is the equivalence class of (1, 2) with respect to the equivalence relation in Exercise 16?

b) Give an interpretation of the equivalence classes for the equivalence relation R in Exercise 16. [Hint: Look at the ratio a/b corresponding to (a, b) .]

41. Which of these collections of subsets are partitions of $\{1, 2, 3, 4, 5, 6\}$?

- a) $\{1, 2\}, \{2, 3, 4\}, \{4, 5, 6\}$ b) $\{1\}, \{2, 3, 6\}, \{4\}, \{5\}$
c) $\{2, 4, 6\}, \{1, 3, 5\}$ d) $\{1, 4, 5\}, \{2, 6\}$

42. Which of these collections of subsets are partitions of $\{-3, -2, -1, 0, 1, 2, 3\}$?

- a) $\{-3, -1, 1, 3\}, \{-2, 0, 2\}$
b) $\{-3, -2, -1, 0\}, \{0, 1, 2, 3\}$
c) $\{-3, 3\}, \{-2, 2\}, \{-1, 1\}, \{0\}$
d) $\{-3, -2, 2, 3\}, \{-1, 1\}$

43. Which of these collections of subsets are partitions of the set of bit strings of length 8?

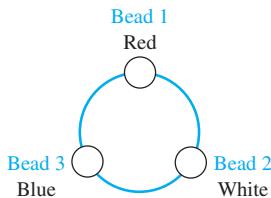
- a) the set of bit strings that begin with 1, the set of bit strings that begin with 00, and the set of bit strings that begin with 01
b) the set of bit strings that contain the string 00, the set of bit strings that contain the string 01, the set of bit strings that contain the string 10, and the set of bit strings that contain the string 11
c) the set of bit strings that end with 00, the set of bit strings that end with 01, the set of bit strings that end with 10, and the set of bit strings that end with 11
d) the set of bit strings that end with 111, the set of bit strings that end with 011, and the set of bit strings that end with 00
e) the set of bit strings that contain $3k$ ones for some nonnegative integer k ; the set of bit strings that contain $3k + 1$ ones for some nonnegative integer k ; and the set of bit strings that contain $3k + 2$ ones for some nonnegative integer k .

44. Which of these collections of subsets are partitions of the set of integers?

- a) the set of even integers and the set of odd integers
b) the set of positive integers and the set of negative integers

- c) the set of integers divisible by 3, the set of integers leaving a remainder of 1 when divided by 3, and the set of integers leaving a remainder of 2 when divided by 3
- d) the set of integers less than -100 , the set of integers with absolute value not exceeding 100, and the set of integers greater than 100
- e) the set of integers not divisible by 3, the set of even integers, and the set of integers that leave a remainder of 3 when divided by 6
45. Which of these are partitions of the set $\mathbf{Z} \times \mathbf{Z}$ of ordered pairs of integers?
- the set of pairs (x, y) , where x or y is odd; the set of pairs (x, y) , where x is even; and the set of pairs (x, y) , where y is even
 - the set of pairs (x, y) , where both x and y are odd; the set of pairs (x, y) , where exactly one of x and y is odd; and the set of pairs (x, y) , where both x and y are even
 - the set of pairs (x, y) , where x is positive; the set of pairs (x, y) , where y is positive; and the set of pairs (x, y) , where both x and y are negative
 - the set of pairs (x, y) , where $3 \mid x$ and $3 \mid y$; the set of pairs (x, y) , where $3 \mid x$ and $3 \nmid y$; the set of pairs (x, y) , where $3 \nmid x$ and $3 \mid y$; and the set of pairs (x, y) , where $3 \nmid x$ and $3 \nmid y$
 - the set of pairs (x, y) , where $x > 0$ and $y > 0$; the set of pairs (x, y) , where $x > 0$ and $y \leq 0$; the set of pairs (x, y) , where $x \leq 0$ and $y > 0$; and the set of pairs (x, y) , where $x \leq 0$ and $y \leq 0$
 - the set of pairs (x, y) , where $x \neq 0$ and $y \neq 0$; the set of pairs (x, y) , where $x = 0$ and $y \neq 0$; and the set of pairs (x, y) , where $x \neq 0$ and $y = 0$
46. Which of these are partitions of the set of real numbers?
- the negative real numbers, $\{0\}$, the positive real numbers
 - the set of irrational numbers, the set of rational numbers
 - the set of intervals $[k, k + 1]$, $k = \dots, -2, -1, 0, 1, 2, \dots$
 - the set of intervals $(k, k + 1)$, $k = \dots, -2, -1, 0, 1, 2, \dots$
 - the set of intervals $(k, k + 1]$, $k = \dots, -2, -1, 0, 1, 2, \dots$
 - the sets $\{x + n \mid n \in \mathbf{Z}\}$ for all $x \in [0, 1)$
47. List the ordered pairs in the equivalence relations produced by these partitions of $\{0, 1, 2, 3, 4, 5\}$.
- $\{0\}, \{1, 2\}, \{3, 4, 5\}$
 - $\{0, 1\}, \{2, 3\}, \{4, 5\}$
 - $\{0, 1, 2\}, \{3, 4, 5\}$
 - $\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}$
48. List the ordered pairs in the equivalence relations produced by these partitions of $\{a, b, c, d, e, f, g\}$.
- $\{a, b\}, \{c, d\}, \{e, f, g\}$
 - $\{a\}, \{b\}, \{c, d\}, \{e, f\}, \{g\}$
 - $\{a, b, c, d\}, \{e, f, g\}$
 - $\{a, c, e, g\}, \{b, d\}, \{f\}$
- A partition P_1 is called a **refinement** of the partition P_2 if every set in P_1 is a subset of one of the sets in P_2 .
49. Show that the partition formed from congruence classes modulo 6 is a refinement of the partition formed from congruence classes modulo 3.
50. Show that the partition of the set of people living in the United States consisting of subsets of people living in the same county (or parish) and same state is a refinement of the partition consisting of subsets of people living in the same state.
51. Show that the partition of the set of bit strings of length 16 formed by equivalence classes of bit strings that agree on the last eight bits is a refinement of the partition formed from the equivalence classes of bit strings that agree on the last four bits.
- In Exercises 52 and 53, R_n refers to the family of equivalence relations defined in Example 5. Recall that $s R_n t$, where s and t are two strings if $s = t$ or s and t are strings with at least n characters that agree in their first n characters.
52. Show that the partition of the set of all bit strings formed by equivalence classes of bit strings with respect to the equivalence relation R_4 is a refinement of the partition formed by equivalence classes of bit strings with respect to the equivalence relation R_3 .
53. Show that the partition of the set of all identifiers in C formed by the equivalence classes of identifiers with respect to the equivalence relation R_{31} is a refinement of the partition formed by equivalence classes of identifiers with respect to the equivalence relation R_8 . (Compilers for “old” C consider identifiers the same when their names agree in their first eight characters, while compilers in standard C consider identifiers the same when their names agree in their first 31 characters.)
54. Suppose that R_1 and R_2 are equivalence relations on a set A . Let P_1 and P_2 be the partitions that correspond to R_1 and R_2 , respectively. Show that $R_1 \subseteq R_2$ if and only if P_1 is a refinement of P_2 .
55. Find the smallest equivalence relation on the set $\{a, b, c, d, e\}$ containing the relation $\{(a, b), (a, c), (d, e)\}$.
56. Suppose that R_1 and R_2 are equivalence relations on the set S . Determine whether each of these combinations of R_1 and R_2 must be an equivalence relation.
- $R_1 \cup R_2$
 - $R_1 \cap R_2$
 - $R_1 \oplus R_2$
57. Consider the equivalence relation from Example 2, namely, $R = \{(x, y) \mid x - y \text{ is an integer}\}$.
- What is the equivalence class of 1 for this equivalence relation?
 - What is the equivalence class of $1/2$ for this equivalence relation?

- *58. Each bead on a bracelet with three beads is either red, white, or blue, as illustrated in the figure shown.



Define the relation R between bracelets as: (B_1, B_2) , where B_1 and B_2 are bracelets, belongs to R if and only if B_2 can be obtained from B_1 by rotating it or rotating it and then reflecting it.

- a) Show that R is an equivalence relation.
 - b) What are the equivalence classes of R ?
- *59. Let R be the relation on the set of all colorings of the 2×2 checkerboard where each of the four squares is colored either red or blue so that (C_1, C_2) , where C_1 and C_2 are 2×2 checkerboards with each of their four squares colored blue or red, belongs to R if and only if C_2 can be obtained from C_1 either by rotating the checkerboard or by rotating it and then reflecting it.
- a) Show that R is an equivalence relation.
 - b) What are the equivalence classes of R ?
60. a) Let R be the relation on the set of functions from \mathbf{Z}^+ to \mathbf{Z}^+ such that (f, g) belongs to R if and only if $f = \Theta(g)$ (see Section 3.2). Show that R is an equivalence relation.
- b) Describe the equivalence class containing $f(n) = n^2$ for the equivalence relation of part (a).
61. Determine the number of different equivalence relations on a set with three elements by listing them.
62. Determine the number of different equivalence relations on a set with four elements by listing them.
- *63. Do we necessarily get an equivalence relation when we form the transitive closure of the symmetric closure of the reflexive closure of a relation?
- *64. Do we necessarily get an equivalence relation when we form the symmetric closure of the reflexive closure of the transitive closure of a relation?
65. Suppose we use Theorem 2 to form a partition P from an equivalence relation R . What is the equivalence relation R' that results if we use Theorem 2 again to form an equivalence relation from P ?
66. Suppose we use Theorem 2 to form an equivalence relation R from a partition P . What is the partition P' that results if we use Theorem 2 again to form a partition from R ?
67. Devise an algorithm to find the smallest equivalence relation containing a given relation.
- *68. Let $p(n)$ denote the number of different equivalence relations on a set with n elements (and by Theorem 2 the number of partitions of a set with n elements). Show that $p(n)$ satisfies the recurrence relation $p(n) = \sum_{j=0}^{n-1} C(n-1, j)p(n-j-1)$ and the initial condition $p(0) = 1$. (Note: The numbers $p(n)$ are called **Bell numbers** after the American mathematician E. T. Bell.)
69. Use Exercise 68 to find the number of different equivalence relations on a set with n elements, where n is a positive integer not exceeding 10.

9.6 Partial Orderings

Introduction

We often use relations to order some or all of the elements of sets. For instance, we order words using the relation containing pairs of words (x, y) , where x comes before y in the dictionary. We schedule projects using the relation consisting of pairs (x, y) , where x and y are tasks in a project such that x must be completed before y begins. We order the set of integers using the relation containing the pairs (x, y) , where x is less than y . When we add all of the pairs of the form (x, x) to these relations, we obtain a relation that is reflexive, antisymmetric, and transitive. These are properties that characterize relations used to order the elements of sets.



DEFINITION 1

A relation R on a set S is called a *partial ordering* or *partial order* if it is reflexive, antisymmetric, and transitive. A set S together with a partial ordering R is called a *partially ordered set*, or *poset*, and is denoted by (S, R) . Members of S are called *elements* of the poset.

We give examples of posets in Examples 1–3.

EXAMPLE 1 Show that the “greater than or equal” relation (\geq) is a partial ordering on the set of integers.

Extra Examples

Solution: Because $a \geq a$ for every integer a , \geq is reflexive. If $a \geq b$ and $b \geq a$, then $a = b$. Hence, \geq is antisymmetric. Finally, \geq is transitive because $a \geq b$ and $b \geq c$ imply that $a \geq c$. It follows that \geq is a partial ordering on the set of integers and (\mathbb{Z}, \geq) is a poset. 

EXAMPLE 2

The divisibility relation $|$ is a partial ordering on the set of positive integers, because it is reflexive, antisymmetric, and transitive, as was shown in Section 9.1. We see that $(\mathbb{Z}^+, |)$ is a poset. Recall that $(\mathbb{Z}^+$ denotes the set of positive integers.) 

EXAMPLE 3

Show that the inclusion relation \subseteq is a partial ordering on the power set of a set S .

Solution: Because $A \subseteq A$ whenever A is a subset of S , \subseteq is reflexive. It is antisymmetric because $A \subseteq B$ and $B \subseteq A$ imply that $A = B$. Finally, \subseteq is transitive, because $A \subseteq B$ and $B \subseteq C$ imply that $A \subseteq C$. Hence, \subseteq is a partial ordering on $P(S)$, and $(P(S), \subseteq)$ is a poset. 

Example 4 illustrates a relation that is not a partial ordering.

EXAMPLE 4

Let R be the relation on the set of people such that xRy if x and y are people and x is older than y . Show that R is not a partial ordering.

Extra Examples

Solution: Note that R is antisymmetric because if a person x is older than a person y , then y is not older than x . That is, if xRy , then $y \not R x$. The relation R is transitive because if person x is older than person y and y is older than person z , then x is older than z . That is, if xRy and yRz , then xRz . However, R is not reflexive, because no person is older than himself or herself. That is, $x \not R x$ for all people x . It follows that R is not a partial ordering. 

In different posets different symbols such as \leq , \subseteq , and $|$, are used for a partial ordering. However, we need a symbol that we can use when we discuss the ordering relation in an arbitrary poset. Customarily, the notation $a \preccurlyeq b$ is used to denote that $(a, b) \in R$ in an arbitrary poset (S, R) . This notation is used because the “less than or equal to” relation on the set of real numbers is the most familiar example of a partial ordering and the symbol \preccurlyeq is similar to the \leq symbol. (Note that the symbol \preccurlyeq is used to denote the relation in *any* poset, not just the “less than or equals” relation.) The notation $a \prec b$ denotes that $a \preccurlyeq b$, but $a \neq b$. Also, we say “ a is less than b ” or “ b is greater than a ” if $a \prec b$.

When a and b are elements of the poset (S, \preccurlyeq) , it is not necessary that either $a \preccurlyeq b$ or $b \preccurlyeq a$. For instance, in $(P(\mathbb{Z}), \subseteq)$, $\{1, 2\}$ is not related to $\{1, 3\}$, and vice versa, because neither set is contained within the other. Similarly, in $(\mathbb{Z}^+, |)$, 2 is not related to 3 and 3 is not related to 2, because $2 \not| 3$ and $3 \not| 2$. This leads to Definition 2.

DEFINITION 2

The elements a and b of a poset (S, \preccurlyeq) are called *comparable* if either $a \preccurlyeq b$ or $b \preccurlyeq a$. When a and b are elements of S such that neither $a \preccurlyeq b$ nor $b \preccurlyeq a$, a and b are called *incomparable*.

EXAMPLE 5

In the poset $(\mathbb{Z}^+, |)$, are the integers 3 and 9 comparable? Are 5 and 7 comparable?

Solution: The integers 3 and 9 are comparable, because $3 | 9$. The integers 5 and 7 are incomparable, because $5 \not| 7$ and $7 \not| 5$. 

The adjective “partial” is used to describe partial orderings because pairs of elements may be incomparable. When every two elements in the set are comparable, the relation is called a **total ordering**.

DEFINITION 3

If (S, \preccurlyeq) is a poset and every two elements of S are comparable, S is called a *totally ordered* or *linearly ordered set*, and \preccurlyeq is called a *total order* or a *linear order*. A totally ordered set is also called a *chain*.

EXAMPLE 6 The poset (\mathbf{Z}, \leq) is totally ordered, because $a \leq b$ or $b \leq a$ whenever a and b are integers. 

EXAMPLE 7 The poset $(\mathbf{Z}^+, |)$ is not totally ordered because it contains elements that are incomparable, such as 5 and 7. 

In Chapter 6 we noted that (\mathbf{Z}^+, \leq) is well-ordered, where \leq is the usual “less than or equal to” relation. We now define well-ordered sets.

DEFINITION 4

(S, \preccurlyeq) is a *well-ordered set* if it is a poset such that \preccurlyeq is a total ordering and every nonempty subset of S has a least element.

EXAMPLE 8 The set of ordered pairs of positive integers, $\mathbf{Z}^+ \times \mathbf{Z}^+$, with $(a_1, a_2) \preccurlyeq (b_1, b_2)$ if $a_1 < b_1$, or if $a_1 = b_1$ and $a_2 \leq b_2$ (the lexicographic ordering), is a well-ordered set. The verification of this is left as Exercise 53. The set \mathbf{Z} , with the usual \leq ordering, is not well-ordered because the set of negative integers, which is a subset of \mathbf{Z} , has no least element. 

At the end of Section 5.3 we showed how to use the principle of well-ordered induction (there called generalized induction) to prove results about a well-ordered set. We now state and prove that this proof technique is valid.

THEOREM 1

THE PRINCIPLE OF WELL-ORDERED INDUCTION Suppose that S is a well-ordered set. Then $P(x)$ is true for all $x \in S$, if

INDUCTIVE STEP: For every $y \in S$, if $P(x)$ is true for all $x \in S$ with $x \prec y$, then $P(y)$ is true.

Proof: Suppose it is not the case that $P(x)$ is true for all $x \in S$. Then there is an element $y \in S$ such that $P(y)$ is false. Consequently, the set $A = \{x \in S \mid P(x) \text{ is false}\}$ is nonempty. Because S is well ordered, A has a least element a . By the choice of a as a least element of A , we know that $P(x)$ is true for all $x \in S$ with $x \prec a$. This implies by the inductive step $P(a)$ is true. This contradiction shows that $P(x)$ must be true for all $x \in S$. 

Remark: We do not need a basis step in a proof using the principle of well-ordered induction because if x_0 is the least element of a well ordered set, the inductive step tells us that $P(x_0)$ is true. This follows because there are no elements $x \in S$ with $x \prec x_0$, so we know (using a vacuous proof) that $P(x)$ is true for all $x \in S$ with $x \prec x_0$.

The principle of well-ordered induction is a versatile technique for proving results about well-ordered sets. Even when it is possible to use mathematical induction for the set of positive integers to prove a theorem, it may be simpler to use the principle of well-ordered induction, as we saw in Examples 5 and 6 in Section 6.2, where we proved a result about the well-ordered set $(\mathbf{N} \times \mathbf{N}, \preccurlyeq)$ where \preccurlyeq is lexicographic ordering on $\mathbf{N} \times \mathbf{N}$.

Lexicographic Order

The words in a dictionary are listed in alphabetic, or lexicographic, order, which is based on the ordering of the letters in the alphabet. This is a special case of an ordering of strings on a set

constructed from a partial ordering on the set. We will show how this construction works in any poset.

First, we will show how to construct a partial ordering on the Cartesian product of two posets, (A_1, \preceq_1) and (A_2, \preceq_2) . The **lexicographic ordering** \preceq on $A_1 \times A_2$ is defined by specifying that one pair is less than a second pair if the first entry of the first pair is less than (in A_1) the first entry of the second pair, or if the first entries are equal, but the second entry of this pair is less than (in A_2) the second entry of the second pair. In other words, (a_1, a_2) is less than (b_1, b_2) , that is,

$$(a_1, a_2) \prec (b_1, b_2),$$

either if $a_1 \prec_1 b_1$ or if both $a_1 = b_1$ and $a_2 \prec_2 b_2$.

We obtain a partial ordering \preceq by adding equality to the ordering \prec on $A_1 \times A_2$. The verification of this is left as an exercise.

EXAMPLE 9 Determine whether $(3, 5) \prec (4, 8)$, whether $(3, 8) \prec (4, 5)$, and whether $(4, 9) \prec (4, 11)$ in the poset $(\mathbb{Z} \times \mathbb{Z}, \preceq)$, where \preceq is the lexicographic ordering constructed from the usual \leq relation on \mathbb{Z} .

Solution: Because $3 < 4$, it follows that $(3, 5) \prec (4, 8)$ and that $(3, 8) \prec (4, 5)$. We have $(4, 9) \prec (4, 11)$, because the first entries of $(4, 9)$ and $(4, 11)$ are the same but $9 < 11$. \blacktriangleleft

In Figure 1 the ordered pairs in $\mathbb{Z}^+ \times \mathbb{Z}^+$ that are less than $(3, 4)$ are highlighted. A lexicographic ordering can be defined on the Cartesian product of n posets (A_1, \preceq_1) , (A_2, \preceq_2) , \dots , (A_n, \preceq_n) . Define the partial ordering \preceq on $A_1 \times A_2 \times \dots \times A_n$ by

$$(a_1, a_2, \dots, a_n) \prec (b_1, b_2, \dots, b_n)$$

if $a_1 \prec_1 b_1$, or if there is an integer $i > 0$ such that $a_1 = b_1, \dots, a_i = b_i$, and $a_{i+1} \prec_{i+1} b_{i+1}$. In other words, one n -tuple is less than a second n -tuple if the entry of the first n -tuple in the first position where the two n -tuples disagree is less than the entry in that position in the second n -tuple.

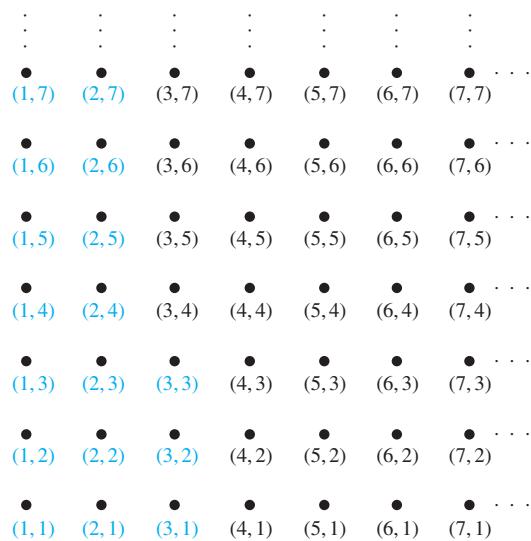


FIGURE 1 The Ordered Pairs Less Than $(3, 4)$ in Lexicographic Order.

EXAMPLE 10 Note that $(1, 2, 3, 5) \prec (1, 2, 4, 3)$, because the entries in the first two positions of these 4-tuples agree, but in the third position the entry in the first 4-tuple, 3, is less than that in the second 4-tuple, 4. (Here the ordering on 4-tuples is the lexicographic ordering that comes from the usual “less than or equals” relation on the set of integers.) 

We can now define lexicographic ordering of strings. Consider the strings $a_1a_2\dots a_m$ and $b_1b_2\dots b_n$ on a partially ordered set S . Suppose these strings are not equal. Let t be the minimum of m and n . The definition of lexicographic ordering is that the string $a_1a_2\dots a_m$ is less than $b_1b_2\dots b_n$ if and only if

$$(a_1, a_2, \dots, a_t) \prec (b_1, b_2, \dots, b_t), \text{ or} \\ (a_1, a_2, \dots, a_t) = (b_1, b_2, \dots, b_t) \text{ and } m < n,$$

where \prec in this inequality represents the lexicographic ordering of S^t . In other words, to determine the ordering of two different strings, the longer string is truncated to the length of the shorter string, namely, to $t = \min(m, n)$ terms. Then the t -tuples made up of the first t terms of each string are compared using the lexicographic ordering on S^t . One string is less than another string if the t -tuple corresponding to the first string is less than the t -tuple of the second string, or if these two t -tuples are the same, but the second string is longer. The verification that this is a partial ordering is left as Exercise 38 for the reader.

EXAMPLE 11 Consider the set of strings of lowercase English letters. Using the ordering of letters in the alphabet, a lexicographic ordering on the set of strings can be constructed. A string is less than a second string if the letter in the first string in the first position where the strings differ comes before the letter in the second string in this position, or if the first string and the second string agree in all positions, but the second string has more letters. This ordering is the same as that used in dictionaries. For example,

$$\textit{discreet} \prec \textit{discrete},$$

because these strings differ first in the seventh position, and $e \prec t$. Also,

$$\textit{discreet} \prec \textit{discreteness},$$

because the first eight letters agree, but the second string is longer. Furthermore,

$$\textit{discrete} \prec \textit{discretion},$$

because

$$\textit{discrete} \prec \textit{discreti}. \quad \blacktriangleleft$$

Hasse Diagrams

Many edges in the directed graph for a finite poset do not have to be shown because they must be present. For instance, consider the directed graph for the partial ordering $\{(a, b) \mid a \leq b\}$ on the set $\{1, 2, 3, 4\}$, shown in Figure 2(a). Because this relation is a partial ordering, it is reflexive, and its directed graph has loops at all vertices. Consequently, we do not have to show these loops because they must be present; in Figure 2(b) loops are not shown. Because a partial ordering is transitive, we do not have to show those edges that must be present because of transitivity. For example, in Figure 2(c) the edges $(1, 3)$, $(1, 4)$, and $(2, 4)$ are not shown because they must be present. If we assume that all edges are pointed “upward” (as they are drawn in the figure), we do not have to show the directions of the edges; Figure 2(c) does not show directions.

In general, we can represent a finite poset (S, \preceq) using this procedure: Start with the directed graph for this relation. Because a partial ordering is reflexive, a loop (a, a) is present at every vertex a . Remove these loops. Next, remove all edges that must be in the partial ordering because of the presence of other edges and transitivity. That is, remove all edges (x, y) for which there is an element $z \in S$ such that $x \prec z$ and $z \prec y$. Finally, arrange each edge so that

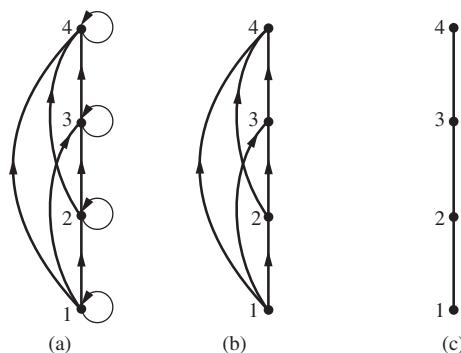


FIGURE 2 Constructing the Hasse Diagram for $(\{1, 2, 3, 4\}, \leq)$.



its initial vertex is below its terminal vertex (as it is drawn on paper). Remove all the arrows on the directed edges, because all edges point “upward” toward their terminal vertex.

These steps are well defined, and only a finite number of steps need to be carried out for a finite poset. When all the steps have been taken, the resulting diagram contains sufficient information to find the partial ordering, as we will explain later. The resulting diagram is called the **Hasse diagram** of (S, \preccurlyeq) , named after the twentieth-century German mathematician Helmut Hasse who made extensive use of them.

Let (S, \preccurlyeq) be a poset. We say that an element $y \in S$ **covers** an element $x \in S$ if $x \prec y$ and there is no element $z \in S$ such that $x \prec z \prec y$. The set of pairs (x, y) such that y covers x is called the **covering relation** of (S, \preccurlyeq) . From the description of the Hasse diagram of a poset, we see that the edges in the Hasse diagram of (S, \preccurlyeq) are upwardly pointing edges corresponding to the pairs in the covering relation of (S, \preccurlyeq) . Furthermore, we can recover a poset from its covering relation, because it is the reflexive transitive closure of its covering relation. (Exercise 31 asks for a proof of this fact.) This tells us that we can construct a partial ordering from its Hasse diagram.

EXAMPLE 12 Draw the Hasse diagram representing the partial ordering $\{(a, b) | a \text{ divides } b\}$ on $\{1, 2, 3, 4, 6, 8, 12\}$.

Solution: Begin with the digraph for this partial order, as shown in Figure 3(a). Remove all loops, as shown in Figure 3(b). Then delete all the edges implied by the transitive property. These are $(1, 4)$, $(1, 6)$, $(1, 8)$, $(1, 12)$, $(2, 8)$, $(2, 12)$, and $(3, 12)$. Arrange all edges to point upward, and delete all arrows to obtain the Hasse diagram. The resulting Hasse diagram is shown in Figure 3(c). 

EXAMPLE 13 Draw the Hasse diagram for the partial ordering $\{(A, B) | A \subseteq B\}$ on the power set $P(S)$ where $S = \{a, b, c\}$.



HELMUT HASSE (1898–1979) Helmut Hasse was born in Kassel, Germany. He served in the German navy after high school. He began his university studies at Göttingen University in 1918, moving in 1920 to Marburg University to study under the number theorist Kurt Hensel. During this time, Hasse made fundamental contributions to algebraic number theory. He became Hensel’s successor at Marburg, later becoming director of the famous mathematical institute at Göttingen in 1934, and took a position at Hamburg University in 1950. Hasse served for 50 years as an editor of *Crelle’s Journal*, a famous German mathematics periodical, taking over the job of chief editor in 1936 when the Nazis forced Hensel to resign. During World War II Hasse worked on applied mathematics research for the German navy. He was noted for the clarity and personal style of his lectures and was devoted both to number theory and to his students. (Hasse has been controversial for connections with the Nazi party. Investigations have shown he was a strong German nationalist but not an ardent Nazi.)

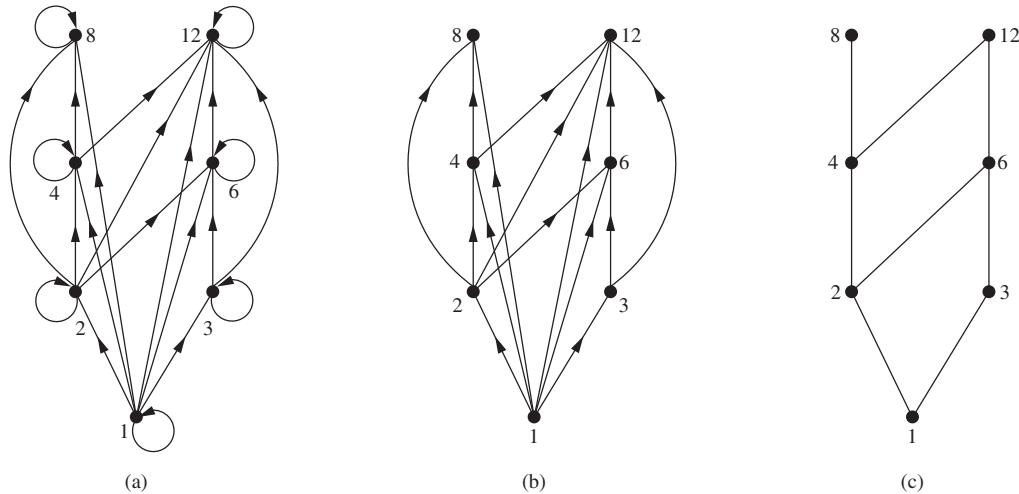


FIGURE 3 Constructing the Hasse Diagram of $(\{1, 2, 3, 4, 6, 8, 12\}, |)$.

Solution: The Hasse diagram for this partial ordering is obtained from the associated digraph by deleting all the loops and all the edges that occur from transitivity, namely, $(\emptyset, \{a, b\})$, $(\emptyset, \{a, c\})$, $(\emptyset, \{b, c\})$, $(\emptyset, \{a, b, c\})$, $(\{a\}, \{a, b, c\})$, $(\{b\}, \{a, b, c\})$, and $(\{c\}, \{a, b, c\})$. Finally all edges point upward, and arrows are deleted. The resulting Hasse diagram is illustrated in Figure 4. \blacktriangleleft

Maximal and Minimal Elements

Elements of posets that have certain extremal properties are important for many applications. An element of a poset is called maximal if it is not less than any element of the poset. That is, a is **maximal** in the poset (S, \preceq) if there is no $b \in S$ such that $a \prec b$. Similarly, an element of a poset is called minimal if it is not greater than any element of the poset. That is, a is **minimal** if there is no element $b \in S$ such that $b \prec a$. Maximal and minimal elements are easy to spot using a Hasse diagram. They are the “top” and “bottom” elements in the diagram.

EXAMPLE 14 Which elements of the poset $(\{2, 4, 5, 10, 12, 20, 25\}, |)$ are maximal, and which are minimal?

Solution: The Hasse diagram in Figure 5 for this poset shows that the maximal elements are 12, 20, and 25, and the minimal elements are 2 and 5. As this example shows, a poset can have more than one maximal element and more than one minimal element. \blacktriangleleft

Sometimes there is an element in a poset that is greater than every other element. Such an element is called the greatest element. That is, a is the **greatest element** of the poset (S, \preceq)

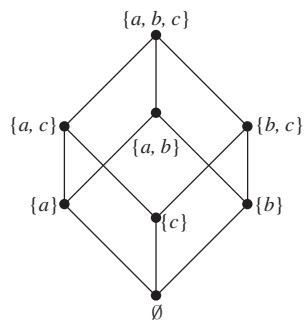


FIGURE 4 The Hasse Diagram of $(P(\{a, b, c\}), \subseteq)$.

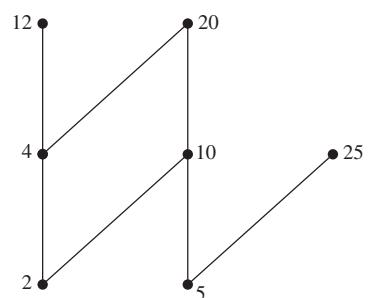
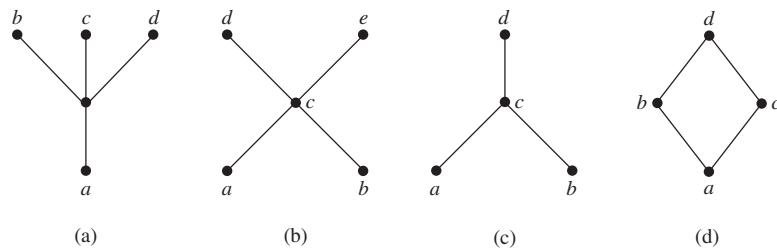


FIGURE 5 The Hasse Diagram of a Poset.

**FIGURE 6** Hasse Diagrams of Four Posets.

if $b \preccurlyeq a$ for all $b \in S$. The greatest element is unique when it exists [see Exercise 40(a)]. Likewise, an element is called the least element if it is less than all the other elements in the poset. That is, a is the **least element** of (S, \preccurlyeq) if $a \preccurlyeq b$ for all $b \in S$. The least element is unique when it exists [see Exercise 40(b)].

EXAMPLE 15 Determine whether the posets represented by each of the Hasse diagrams in Figure 6 have a greatest element and a least element.

Solution: The least element of the poset with Hasse diagram (a) is a . This poset has no greatest element. The poset with Hasse diagram (b) has neither a least nor a greatest element. The poset with Hasse diagram (c) has no least element. Its greatest element is d . The poset with Hasse diagram (d) has least element a and greatest element d . \blacktriangleleft

EXAMPLE 16 Let S be a set. Determine whether there is a greatest element and a least element in the poset $(P(S), \subseteq)$.

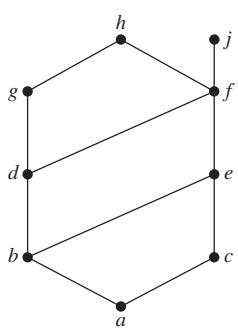
Solution: The least element is the empty set, because $\emptyset \subseteq T$ for any subset T of S . The set S is the greatest element in this poset, because $T \subseteq S$ whenever T is a subset of S . \blacktriangleleft

EXAMPLE 17 Is there a greatest element and a least element in the poset $(\mathbb{Z}^+, |)$?

Solution: The integer 1 is the least element because $1|n$ whenever n is a positive integer. Because there is no integer that is divisible by all positive integers, there is no greatest element. \blacktriangleleft

Sometimes it is possible to find an element that is greater than or equal to all the elements in a subset A of a poset (S, \preccurlyeq) . If u is an element of S such that $a \preccurlyeq u$ for all elements $a \in A$, then u is called an **upper bound** of A . Likewise, there may be an element less than or equal to all the elements in A . If l is an element of S such that $l \preccurlyeq a$ for all elements $a \in A$, then l is called a **lower bound** of A .

EXAMPLE 18 Find the lower and upper bounds of the subsets $\{a, b, c\}$, $\{j, h\}$, and $\{a, c, d, f\}$ in the poset with the Hasse diagram shown in Figure 7.



Solution: The upper bounds of $\{a, b, c\}$ are e , f , j , and h , and its only lower bound is a . There are no upper bounds of $\{j, h\}$, and its lower bounds are a, b, c, d, e , and f . The upper bounds of $\{a, c, d, f\}$ are f , h , and j , and its lower bound is a . \blacktriangleleft

The element x is called the **least upper bound** of the subset A if x is an upper bound that is less than every other upper bound of A . Because there is only one such element, if it exists, it makes sense to call this element *the* least upper bound [see Exercise 42(a)]. That is, x is the least upper bound of A if $a \preccurlyeq x$ whenever $a \in A$, and $x \preccurlyeq z$ whenever z is an upper bound of A . Similarly, the element y is called the **greatest lower bound** of A if y is a lower bound of A and $z \preccurlyeq y$ whenever z is a lower bound of A . The greatest lower bound of A is unique if it exists [see Exercise 42(b)]. The greatest lower bound and least upper bound of a subset A are denoted by $\text{glb}(A)$ and $\text{lub}(A)$, respectively.

FIGURE 7 The Hasse Diagram of a Poset.

EXAMPLE 19 Find the greatest lower bound and the least upper bound of $\{b, d, g\}$, if they exist, in the poset shown in Figure 7.

Solution: The upper bounds of $\{b, d, g\}$ are g and h . Because $g \prec h$, g is the least upper bound. The lower bounds of $\{b, d, g\}$ are a and b . Because $a \prec b$, b is the greatest lower bound. \blacktriangleleft

EXAMPLE 20 Find the greatest lower bound and the least upper bound of the sets $\{3, 9, 12\}$ and $\{1, 2, 4, 5, 10\}$, if they exist, in the poset $(\mathbb{Z}^+, |)$.



Solution: An integer is a lower bound of $\{3, 9, 12\}$ if 3, 9, and 12 are divisible by this integer. The only such integers are 1 and 3. Because $1 \mid 3, 3$ is the greatest lower bound of $\{3, 9, 12\}$. The only lower bound for the set $\{1, 2, 4, 5, 10\}$ with respect to $|$ is the element 1. Hence, 1 is the greatest lower bound for $\{1, 2, 4, 5, 10\}$.

An integer is an upper bound for $\{3, 9, 12\}$ if and only if it is divisible by 3, 9, and 12. The integers with this property are those divisible by the least common multiple of 3, 9, and 12, which is 36. Hence, 36 is the least upper bound of $\{3, 9, 12\}$. A positive integer is an upper bound for the set $\{1, 2, 4, 5, 10\}$ if and only if it is divisible by 1, 2, 4, 5, and 10. The integers with this property are those integers divisible by the least common multiple of these integers, which is 20. Hence, 20 is the least upper bound of $\{1, 2, 4, 5, 10\}$. \blacktriangleleft

Lattices

A partially ordered set in which every pair of elements has both a least upper bound and a greatest lower bound is called a **lattice**. Lattices have many special properties. Furthermore, lattices are used in many different applications such as models of information flow and play an important role in Boolean algebra.

EXAMPLE 21 Determine whether the posets represented by each of the Hasse diagrams in Figure 8 are lattices.

Solution: The posets represented by the Hasse diagrams in (a) and (c) are both lattices because in each poset every pair of elements has both a least upper bound and a greatest lower bound, as the reader should verify. On the other hand, the poset with the Hasse diagram shown in (b) is not a lattice, because the elements b and c have no least upper bound. To see this, note that each of the elements d , e , and f is an upper bound, but none of these three elements precedes the other two with respect to the ordering of this poset. \blacktriangleleft

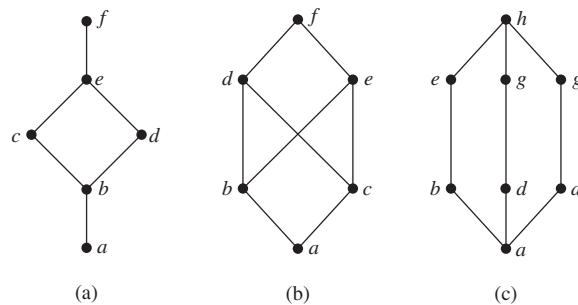


FIGURE 8 Hasse Diagrams of Three Posets.

EXAMPLE 22 Is the poset $(\mathbb{Z}^+, |)$ a lattice?

Solution: Let a and b be two positive integers. The least upper bound and greatest lower bound of these two integers are the least common multiple and the greatest common divisor of these integers, respectively, as the reader should verify. It follows that this poset is a lattice. 

EXAMPLE 23 Determine whether the posets $(\{1, 2, 3, 4, 5\}, |)$ and $(\{1, 2, 4, 8, 16\}, |)$ are lattices.

Solution: Because 2 and 3 have no upper bounds in $(\{1, 2, 3, 4, 5\}, |)$, they certainly do not have a least upper bound. Hence, the first poset is not a lattice.

Every two elements of the second poset have both a least upper bound and a greatest lower bound. The least upper bound of two elements in this poset is the larger of the elements and the greatest lower bound of two elements is the smaller of the elements, as the reader should verify. Hence, this second poset is a lattice. 

EXAMPLE 24 Determine whether $(P(S), \subseteq)$ is a lattice where S is a set.

Solution: Let A and B be two subsets of S . The least upper bound and the greatest lower bound of A and B are $A \cup B$ and $A \cap B$, respectively, as the reader can show. Hence, $(P(S), \subseteq)$ is a lattice. 

EXAMPLE 25



There are billions of pages of classified U.S. government documents.

The Lattice Model of Information Flow In many settings the flow of information from one person or computer program to another is restricted via security clearances. We can use a lattice model to represent different information flow policies. For example, one common information flow policy is the *multilevel security policy* used in government and military systems. Each piece of information is assigned to a security class, and each security class is represented by a pair (A, C) where A is an *authority level* and C is a *category*. People and computer programs are then allowed access to information from a specific restricted set of security classes.

The typical authority levels used in the U.S. government are unclassified (0), confidential (1), secret (2), and top secret (3). (Information is said to be classified if it is confidential, secret, or top secret.) Categories used in security classes are the subsets of a set of all *compartments* relevant to a particular area of interest. Each compartment represents a particular subject area. For example, if the set of compartments is $\{\text{spies, moles, double agents}\}$, then there are eight different categories, one for each of the eight subsets of the set of compartments, such as $\{\text{spies, moles}\}$.

We can order security classes by specifying that $(A_1, C_1) \preceq (A_2, C_2)$ if and only if $A_1 \leq A_2$ and $C_1 \subseteq C_2$. Information is permitted to flow from security class (A_1, C_1) into security class (A_2, C_2) if and only if $(A_1, C_1) \preceq (A_2, C_2)$. For example, information is permitted to flow from the security class $(\text{secret, } \{\text{spies, moles}\})$ into the security class $(\text{top secret, } \{\text{spies, moles, double agents}\})$, whereas information is not allowed to flow from the security class $(\text{top secret, } \{\text{spies, moles}\})$ into either of the security classes $(\text{secret, } \{\text{spies, moles, double agents}\})$ or $(\text{top secret, } \{\text{spies}\})$.

We leave it to the reader (see Exercise 48) to show that the set of all security classes with the ordering defined in this example forms a lattice. 

Topological Sorting

Suppose that a project is made up of 20 different tasks. Some tasks can be completed only after others have been finished. How can an order be found for these tasks? To model this problem we set up a partial order on the set of tasks so that $a \prec b$ if and only if a and b are tasks where b



cannot be started until a has been completed. To produce a schedule for the project, we need to produce an order for all 20 tasks that is compatible with this partial order. We will show how this can be done.

We begin with a definition. A total ordering \preccurlyeq is said to be **compatible** with the partial ordering R if $a \preccurlyeq b$ whenever aRb . Constructing a compatible total ordering from a partial ordering is called **topological sorting**.^{*} We will need to use Lemma 1.

LEMMA 1

Every finite nonempty poset (S, \preccurlyeq) has at least one minimal element.

Proof: Choose an element a_0 of S . If a_0 is not minimal, then there is an element a_1 with $a_1 \prec a_0$. If a_1 is not minimal, there is an element a_2 with $a_2 \prec a_1$. Continue this process, so that if a_n is not minimal, there is an element a_{n+1} with $a_{n+1} \prec a_n$. Because there are only a finite number of elements in the poset, this process must end with a minimal element a_n . \triangleleft

The topological sorting algorithm we will describe works for any finite nonempty poset. To define a total ordering on the poset (A, \preccurlyeq) , first choose a minimal element a_1 ; such an element exists by Lemma 1. Next, note that $(A - \{a_1\}, \preccurlyeq)$ is also a poset, as the reader should verify. (Here by \preccurlyeq we mean the restriction of the original relation \preccurlyeq on A to $A - \{a_1\}$.) If it is nonempty, choose a minimal element a_2 of this poset. Then remove a_2 as well, and if there are additional elements left, choose a minimal element a_3 in $A - \{a_1, a_2\}$. Continue this process by choosing a_{k+1} to be a minimal element in $A - \{a_1, a_2, \dots, a_k\}$, as long as elements remain.

Because A is a finite set, this process must terminate. The end product is a sequence of elements a_1, a_2, \dots, a_n . The desired total ordering \preccurlyeq_t is defined by

$$a_1 \preccurlyeq_t a_2 \preccurlyeq_t \cdots \preccurlyeq_t a_n.$$

This total ordering is compatible with the original partial ordering. To see this, note that if $b \prec c$ in the original partial ordering, c is chosen as the minimal element at a phase of the algorithm where b has already been removed, for otherwise c would not be a minimal element. Pseudocode for this topological sorting algorithm is shown in Algorithm 1.

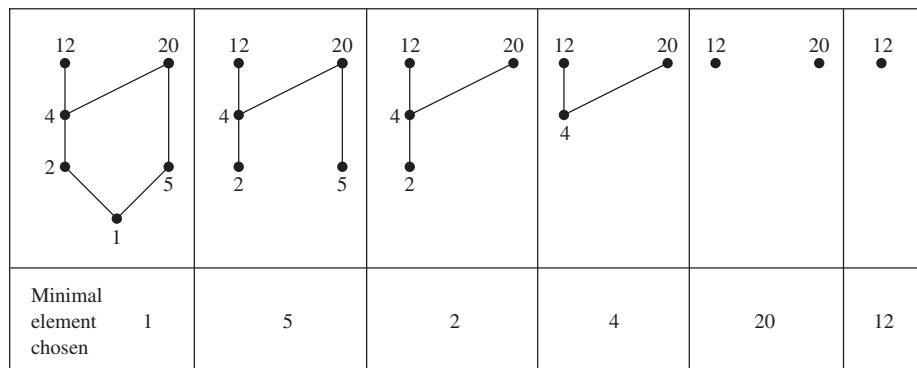
ALGORITHM 1 Topological Sorting.

```

procedure topological sort (( $S, \preccurlyeq$ ): finite poset)
   $k := 1$ 
  while  $S \neq \emptyset$ 
     $a_k :=$  a minimal element of  $S$  {such an element exists by Lemma 1}
     $S := S - \{a_k\}$ 
     $k := k + 1$ 
  return  $a_1, a_2, \dots, a_n$  { $a_1, a_2, \dots, a_n$  is a compatible total ordering of  $S$ }
```

EXAMPLE 26 Find a compatible total ordering for the poset $(\{1, 2, 4, 5, 12, 20\}, |)$.

*“Topological sorting” is terminology used by computer scientists; mathematicians use the terminology “linearization of a partial ordering” for the same thing. In mathematics, topology is the branch of geometry dealing with properties of geometric figures that hold for all figures that can be transformed into one another by continuous bijections. In computer science, a topology is any arrangement of objects that can be connected with edges.

**FIGURE 9** A Topological Sort of $(\{1, 2, 4, 5, 12, 20\}, |)$.

Solution: The first step is to choose a minimal element. This must be 1, because it is the only minimal element. Next, select a minimal element of $(\{2, 4, 5, 12, 20\}, |)$. There are two minimal elements in this poset, namely, 2 and 5. We select 5. The remaining elements are $\{2, 4, 12, 20\}$. The only minimal element at this stage is 2. Next, 4 is chosen because it is the only minimal element of $(\{4, 12, 20\}, |)$. Because both 12 and 20 are minimal elements of $(\{12, 20\}, |)$, either can be chosen next. We select 20, which leaves 12 as the last element left. This produces the total ordering

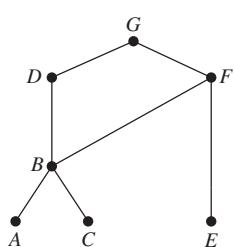
$$1 \prec 5 \prec 2 \prec 4 \prec 20 \prec 12.$$

The steps used by this sorting algorithm are displayed in Figure 9.

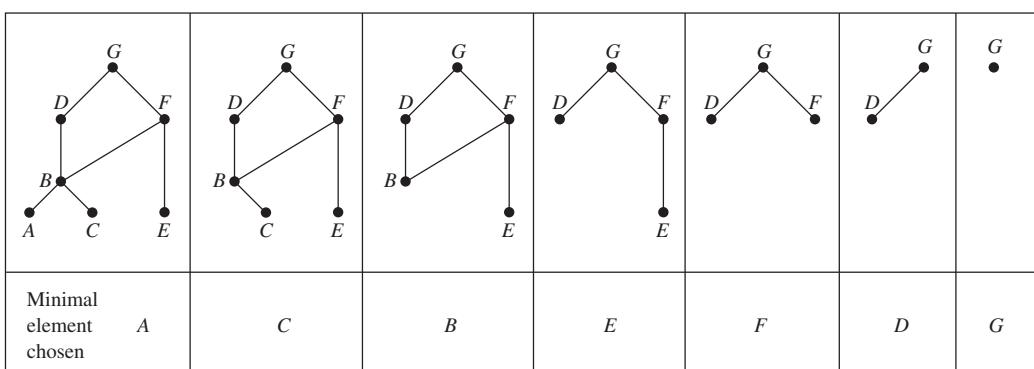
Topological sorting has an application to the scheduling of projects.

EXAMPLE 27

A development project at a computer company requires the completion of seven tasks. Some of these tasks can be started only after other tasks are finished. A partial ordering on tasks is set up by considering task $X \prec$ task Y if task Y cannot be started until task X has been completed. The Hasse diagram for the seven tasks, with respect to this partial ordering, is shown in Figure 10. Find an order in which these tasks can be carried out to complete the project.

**FIGURE 10** The Hasse Diagram for Seven Tasks.

Solution: An ordering of the seven tasks can be obtained by performing a topological sort. The steps of a sort are illustrated in Figure 11. The result of this sort, $A \prec C \prec B \prec E \prec F \prec D \prec G$, gives one possible order for the tasks.

**FIGURE 11** A Topological Sort of the Tasks.

Exercises

1. Which of these relations on $\{0, 1, 2, 3\}$ are partial orderings? Determine the properties of a partial ordering that the others lack.

- a) $\{(0, 0), (1, 1), (2, 2), (3, 3)\}$
- b) $\{(0, 0), (1, 1), (2, 0), (2, 2), (2, 3), (3, 2), (3, 3)\}$
- c) $\{(0, 0), (1, 1), (1, 2), (2, 2), (3, 3)\}$
- d) $\{(0, 0), (1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)\}$
- e) $\{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 2), (3, 3)\}$

2. Which of these relations on $\{0, 1, 2, 3\}$ are partial orderings? Determine the properties of a partial ordering that the others lack.

- a) $\{(0, 0), (2, 2), (3, 3)\}$
- b) $\{(0, 0), (1, 1), (2, 0), (2, 2), (2, 3), (3, 3)\}$
- c) $\{(0, 0), (1, 1), (1, 2), (2, 2), (3, 1), (3, 3)\}$
- d) $\{(0, 0), (1, 1), (1, 2), (1, 3), (2, 0), (2, 2), (2, 3), (3, 0), (3, 3)\}$
- e) $\{(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (1, 3), (2, 0), (2, 2), (3, 3)\}$

3. Is (S, R) a poset if S is the set of all people in the world and $(a, b) \in R$, where a and b are people, if

- a) a is taller than b ?
- b) a is not taller than b ?
- c) $a = b$ or a is an ancestor of b ?
- d) a and b have a common friend?

4. Is (S, R) a poset if S is the set of all people in the world and $(a, b) \in R$, where a and b are people, if

- a) a is no shorter than b ?
- b) a weighs more than b ?
- c) $a = b$ or a is a descendant of b ?
- d) a and b do not have a common friend?

5. Which of these are posets?

- a) $(\mathbb{Z}, =)$ b) (\mathbb{Z}, \neq) c) (\mathbb{Z}, \geq) d) (\mathbb{Z}, \neq)

6. Which of these are posets?

- a) $(\mathbf{R}, =)$ b) $(\mathbf{R}, <)$ c) (\mathbf{R}, \leq) d) (\mathbf{R}, \neq)

7. Determine whether the relations represented by these zero-one matrices are partial orders.

a)
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

b)
$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

c)
$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

8. Determine whether the relations represented by these zero-one matrices are partial orders.

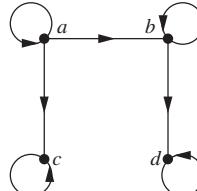
a)
$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

c)
$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

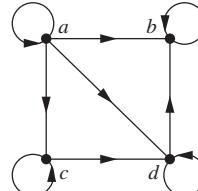
b)
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

In Exercises 9–11 determine whether the relation with the directed graph shown is a partial order.

9.



10.



11.



12. Let (S, R) be a poset. Show that (S, R^{-1}) is also a poset, where R^{-1} is the inverse of R . The poset (S, R^{-1}) is called the **dual** of (S, R) .

13. Find the duals of these posets.

- a) $(\{0, 1, 2\}, \leq)$ b) (\mathbb{Z}, \geq)
c) $(P(\mathbb{Z}), \supseteq)$ d) $(\mathbb{Z}^+, |)$

14. Which of these pairs of elements are comparable in the poset $(\mathbb{Z}^+, |)$?

- a) 5, 15 b) 6, 9 c) 8, 16 d) 7, 7

15. Find two incomparable elements in these posets.

- a) $(P(\{0, 1, 2\}), \subseteq)$ b) $(\{1, 2, 4, 6, 8\}, |)$

16. Let $S = \{1, 2, 3, 4\}$. With respect to the lexicographic order based on the usual “less than” relation,

- a) find all pairs in $S \times S$ less than $(2, 3)$.

- b) find all pairs in $S \times S$ greater than $(3, 1)$.

- c) draw the Hasse diagram of the poset $(S \times S, \preccurlyeq)$.

17. Find the lexicographic ordering of these n -tuples:

- a) $(1, 1, 2), (1, 2, 1)$ b) $(0, 1, 2, 3), (0, 1, 3, 2)$
c) $(1, 0, 1, 0, 1), (0, 1, 1, 1, 0)$

18. Find the lexicographic ordering of these strings of lowercase English letters:

- a) *quack, quick, quicksilver, quicksand, quacking*
b) *open, opener, opera, operand, opened*
c) *zoo, zero, zoom, zoology, zoological*

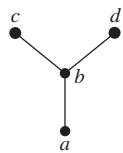
19. Find the lexicographic ordering of the bit strings 0, 01, 11, 001, 010, 011, 0001, and 0101 based on the ordering $0 < 1$.

20. Draw the Hasse diagram for the “greater than or equal to” relation on $\{0, 1, 2, 3, 4, 5\}$.

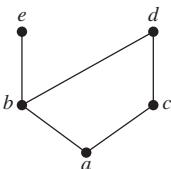
21. Draw the Hasse diagram for the “less than or equal to” relation on $\{0, 2, 5, 10, 11, 15\}$.
22. Draw the Hasse diagram for divisibility on the set
 a) $\{1, 2, 3, 4, 5, 6\}$. b) $\{3, 5, 7, 11, 13, 16, 17\}$.
 c) $\{2, 3, 5, 10, 11, 15, 25\}$. d) $\{1, 3, 9, 27, 81, 243\}$.
23. Draw the Hasse diagram for divisibility on the set
 a) $\{1, 2, 3, 4, 5, 6, 7, 8\}$. b) $\{1, 2, 3, 5, 7, 11, 13\}$.
 c) $\{1, 2, 3, 6, 12, 24, 36, 48\}$.
 d) $\{1, 2, 4, 8, 16, 32, 64\}$.
24. Draw the Hasse diagram for inclusion on the set $P(S)$, where $S = \{a, b, c, d\}$.

In Exercises 25–27 list all ordered pairs in the partial ordering with the accompanying Hasse diagram.

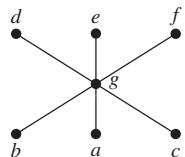
25.



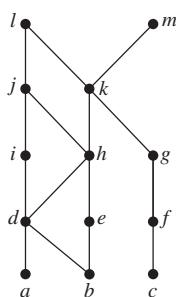
26.



27.



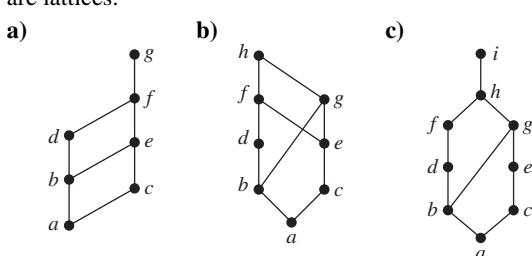
28. What is the covering relation of the partial ordering $\{(a, b) \mid a \text{ divides } b\}$ on $\{1, 2, 3, 4, 6, 12\}$?
29. What is the covering relation of the partial ordering $\{(A, B) \mid A \subseteq B\}$ on the power set of S , where $S = \{a, b, c\}$?
30. What is the covering relation of the partial ordering for the poset of security classes defined in Example 25?
31. Show that a finite poset can be reconstructed from its covering relation. [Hint: Show that the poset is the reflexive transitive closure of its covering relation.]
32. Answer these questions for the partial order represented by this Hasse diagram.



- a) Find the maximal elements.
 b) Find the minimal elements.
 c) Is there a greatest element?

- d) Is there a least element?
 e) Find all upper bounds of $\{a, b, c\}$.
 f) Find the least upper bound of $\{a, b, c\}$, if it exists.
 g) Find all lower bounds of $\{f, g, h\}$.
 h) Find the greatest lower bound of $\{f, g, h\}$, if it exists.
33. Answer these questions for the poset $(\{3, 5, 9, 15, 24, 45\}, |)$.
 a) Find the maximal elements.
 b) Find the minimal elements.
 c) Is there a greatest element?
 d) Is there a least element?
 e) Find all upper bounds of $\{3, 5\}$.
 f) Find the least upper bound of $\{3, 5\}$, if it exists.
 g) Find all lower bounds of $\{15, 45\}$.
 h) Find the greatest lower bound of $\{15, 45\}$, if it exists.
34. Answer these questions for the poset $(\{2, 4, 6, 9, 12, 18, 27, 36, 48, 60, 72\}, |)$.
 a) Find the maximal elements.
 b) Find the minimal elements.
 c) Is there a greatest element?
 d) Is there a least element?
 e) Find all upper bounds of $\{2, 9\}$.
 f) Find the least upper bound of $\{2, 9\}$, if it exists.
 g) Find all lower bounds of $\{60, 72\}$.
 h) Find the greatest lower bound of $\{60, 72\}$, if it exists.
35. Answer these questions for the poset $(\{\{1\}, \{2\}, \{4\}, \{1, 2\}, \{1, 4\}, \{2, 4\}, \{3, 4\}, \{1, 3, 4\}, \{2, 3, 4\}\}, \subseteq)$.
 a) Find the maximal elements.
 b) Find the minimal elements.
 c) Is there a greatest element?
 d) Is there a least element?
 e) Find all upper bounds of $\{\{2\}, \{4\}\}$.
 f) Find the least upper bound of $\{\{2\}, \{4\}\}$, if it exists.
 g) Find all lower bounds of $\{\{1, 3, 4\}, \{2, 3, 4\}\}$.
 h) Find the greatest lower bound of $\{\{1, 3, 4\}, \{2, 3, 4\}\}$, if it exists.
36. Give a poset that has
 a) a minimal element but no maximal element.
 b) a maximal element but no minimal element.
 c) neither a maximal nor a minimal element.
37. Show that lexicographic order is a partial ordering on the Cartesian product of two posets.
38. Show that lexicographic order is a partial ordering on the set of strings from a poset.
39. Suppose that (S, \preceq_1) and (T, \preceq_2) are posets. Show that $(S \times T, \preceq)$ is a poset where $(s, t) \preceq (u, v)$ if and only if $s \preceq_1 u$ and $t \preceq_2 v$.

- 40.** a) Show that there is exactly one greatest element of a poset, if such an element exists.
 b) Show that there is exactly one least element of a poset, if such an element exists.
- 41.** a) Show that there is exactly one maximal element in a poset with a greatest element.
 b) Show that there is exactly one minimal element in a poset with a least element.
- 42.** a) Show that the least upper bound of a set in a poset is unique if it exists.
 b) Show that the greatest lower bound of a set in a poset is unique if it exists.
- 43.** Determine whether the posets with these Hasse diagrams are lattices.

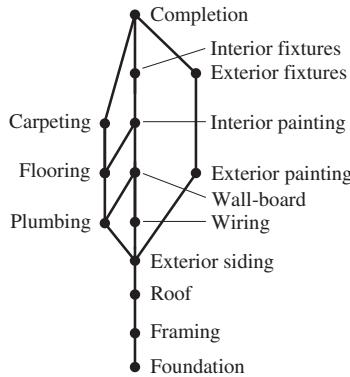


- 44.** Determine whether these posets are lattices.
- $(\{1, 3, 6, 9, 12\}, |)$
 - $(\{1, 5, 25, 125\}, |)$
 - (\mathbf{Z}, \geq)
 - $(P(S), \supseteq)$, where $P(S)$ is the power set of a set S
- 45.** Show that every nonempty finite subset of a lattice has a least upper bound and a greatest lower bound.
- 46.** Show that if the poset (S, R) is a lattice then the dual poset (S, R^{-1}) is also a lattice.
- 47.** In a company, the lattice model of information flow is used to control sensitive information with security classes represented by ordered pairs (A, C) . Here A is an authority level, which may be nonproprietary (0), proprietary (1), restricted (2), or registered (3). A category C is a subset of the set of all projects $\{\text{Cheetah}, \text{Impala}, \text{Puma}\}$. (Names of animals are often used as code names for projects in companies.)
- Is information permitted to flow from $(\text{Proprietary}, \{\text{Cheetah}, \text{Puma}\})$ into $(\text{Restricted}, \{\text{Puma}\})$?
 - Is information permitted to flow from $(\text{Restricted}, \{\text{Cheetah}\})$ into $(\text{Registered}, \{\text{Cheetah}, \text{Impala}\})$?
 - Into which classes is information from $(\text{Proprietary}, \{\text{Cheetah}, \text{Puma}\})$ permitted to flow?
 - From which classes is information permitted to flow into the security class $(\text{Restricted}, \{\text{Impala}, \text{Puma}\})$?
- 48.** Show that the set S of security classes (A, C) is a lattice, where A is a positive integer representing an authority class and C is a subset of a finite set of compartments, with $(A_1, C_1) \preceq (A_2, C_2)$ if and only if $A_1 \leq A_2$ and $C_1 \subseteq C_2$. [Hint: First show that (S, \preceq) is a poset and then show that the least upper bound and greatest lower bound of (A_1, C_1) and (A_2, C_2) are $(\max(A_1, A_2), C_1 \cup C_2)$ and $(\min(A_1, A_2), C_1 \cap C_2)$, respectively.]

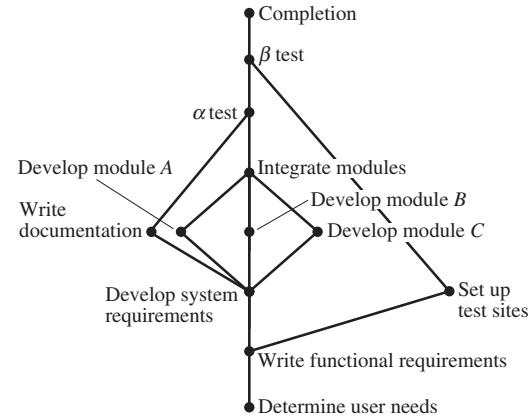
***49.** Show that the set of all partitions of a set S with the relation $P_1 \preceq P_2$ if the partition P_1 is a refinement of the partition P_2 is a lattice. (See the preamble to Exercise 49 of Section 9.5.)

- 50.** Show that every totally ordered set is a lattice.
- 51.** Show that every finite lattice has a least element and a greatest element.
- 52.** Give an example of an infinite lattice with
- neither a least nor a greatest element.
 - a least but not a greatest element.
 - a greatest but not a least element.
 - both a least and a greatest element.
- 53.** Verify that $(\mathbf{Z}^+ \times \mathbf{Z}^+, \preceq)$ is a well-ordered set, where \preceq is lexicographic order, as claimed in Example 8.
- 54.** Determine whether each of these posets is well-ordered.
- (S, \leq) , where $S = \{10, 11, 12, \dots\}$
 - $(\mathbf{Q} \cap [0, 1], \leq)$ (the set of rational numbers between 0 and 1 inclusive)
 - (S, \leq) , where S is the set of positive rational numbers with denominators not exceeding 3
 - (\mathbf{Z}^-, \geq) , where \mathbf{Z}^- is the set of negative integers
- A poset (R, \preceq) is **well-founded** if there is no infinite decreasing sequence of elements in the poset, that is, elements x_1, x_2, \dots, x_n such that $\dots < x_n < \dots < x_2 < x_1$. A poset (R, \preceq) is **dense** if for all $x \in S$ and $y \in S$ with $x < y$, there is an element $z \in R$ such that $x < z < y$.
- 55.** Show that the poset (\mathbf{Z}, \preceq) , where $x < y$ if and only if $|x| < |y|$ is well-founded but is not a totally ordered set.
- 56.** Show that a dense poset with at least two elements that are comparable is not well-founded.
- 57.** Show that the poset of rational numbers with the usual “less than or equal to” relation, (\mathbf{Q}, \leq) , is a dense poset.
- *58.** Show that the set of strings of lowercase English letters with lexicographic order is neither well-founded nor dense.
- 59.** Show that a poset is well-ordered if and only if it is totally ordered and well-founded.
- 60.** Show that a finite nonempty poset has a maximal element.
- 61.** Find a compatible total order for the poset with the Hasse diagram shown in Exercise 32.
- 62.** Find a compatible total order for the divisibility relation on the set $\{1, 2, 3, 6, 8, 12, 24, 36\}$.
- 63.** Find all compatible total orderings for the poset $(\{1, 2, 4, 5, 12, 20\}, |)$ from Example 26.
- 64.** Find all compatible total orderings for the poset with the Hasse diagram in Exercise 27.
- 65.** Find all possible orders for completing the tasks in the development project in Example 27.

- 66.** Schedule the tasks needed to build a house, by specifying their order, if the Hasse diagram representing these tasks is as shown in the figure.



- 67.** Find an ordering of the tasks of a software project if the Hasse diagram for the tasks of the project is as shown.



Key Terms and Results

TERMS

binary relation from A to B : a subset of $A \times B$

relation on A : a binary relation from A to itself (i.e., a subset of $A \times A$)

$S \circ R$: composite of R and S

R^{-1} : inverse relation of R

R^n : n th power of R

reflexive: a relation R on A is reflexive if $(a, a) \in R$ for all $a \in A$

symmetric: a relation R on A is symmetric if $(b, a) \in R$ whenever $(a, b) \in R$

antisymmetric: a relation R on A is antisymmetric if $a = b$ whenever $(a, b) \in R$ and $(b, a) \in R$

transitive: a relation R on A is transitive if $(a, b) \in R$ and $(b, c) \in R$ implies that $(a, c) \in R$

n -ary relation on A_1, A_2, \dots, A_n : a subset of $A_1 \times A_2 \times \dots \times A_n$

relational data model: a model for representing databases using n -ary relations

primary key: a domain of an n -ary relation such that an n -tuple is uniquely determined by its value for this domain

composite key: the Cartesian product of domains of an n -ary relation such that an n -tuple is uniquely determined by its values in these domains

selection operator: a function that selects the n -tuples in an n -ary relation that satisfy a specified condition

projection: a function that produces relations of smaller degree from an n -ary relation by deleting fields

join: a function that combines n -ary relations that agree on certain fields

directed graph or digraph: a set of elements called vertices and ordered pairs of these elements, called edges

loop: an edge of the form (a, a)

closure of a relation R with respect to a property P : the relation S (if it exists) that contains R , has property P , and is contained within any relation that contains R and has property P

path in a digraph: a sequence of edges $(a, x_1), (x_1, x_2), \dots, (x_{n-2}, x_{n-1}), (x_{n-1}, b)$ such that the terminal vertex of each edge is the initial vertex of the succeeding edge in the sequence

circuit (or cycle) in a digraph: a path that begins and ends at the same vertex

R^* (connectivity relation): the relation consisting of those ordered pairs (a, b) such that there is a path from a to b

equivalence relation: a reflexive, symmetric, and transitive relation

equivalent: if R is an equivalence relation, a is equivalent to b if aRb

$[a]_R$ (equivalence class of a with respect to R): the set of all elements of A that are equivalent to a

$[a]_m$ (congruence class modulo m): the set of integers congruent to a modulo m

partition of a set S : a collection of pairwise disjoint nonempty subsets that have S as their union

partial ordering: a relation that is reflexive, antisymmetric, and transitive

poset (S, R): a set S and a partial ordering R on this set

comparable: the elements a and b in the poset (A, \preccurlyeq) are comparable if $a \preccurlyeq b$ or $b \preccurlyeq a$

incomparable: elements in a poset that are not comparable

total (or linear) ordering: a partial ordering for which every pair of elements are comparable

totally (or linearly) ordered set: a poset with a total (or linear) ordering

well-ordered set: a poset (S, \preccurlyeq) , where \preccurlyeq is a total order and every nonempty subset of S has a least element

lexicographic order: a partial ordering of Cartesian products or strings

Hasse diagram: a graphical representation of a poset where loops and all edges resulting from the transitive property are not shown, and the direction of the edges is indicated by the position of the vertices

maximal element: an element of a poset that is not less than any other element of the poset

minimal element: an element of a poset that is not greater than any other element of the poset

greatest element: an element of a poset greater than all other elements in this set

least element: an element of a poset less than all other elements in this set

upper bound of a set: an element in a poset greater than all other elements in the set

lower bound of a set: an element in a poset less than all other elements in the set

least upper bound of a set: an upper bound of the set that is less than all other upper bounds

greatest lower bound of a set: a lower bound of the set that is greater than all other lower bounds

lattice: a partially ordered set in which every two elements have a greatest lower bound and a least upper bound

compatible total ordering for a partial ordering: a total ordering that contains the given partial ordering

topological sort: the construction of a total ordering compatible with a given partial ordering

RESULTS

The reflexive closure of a relation R on the set A equals $R \cup \Delta$, where $\Delta = \{(a, a) \mid a \in A\}$.

The symmetric closure of a relation R on the set A equals $R \cup R^{-1}$, where $R^{-1} = \{(b, a) \mid (a, b) \in R\}$.

The transitive closure of a relation equals the connectivity relation formed from this relation.

Warshall's algorithm for finding the transitive closure of a relation

Let R be an equivalence relation. Then the following three statements are equivalent: (1) $a R b$; (2) $[a]_R \cap [b]_R \neq \emptyset$; (3) $[a]_R = [b]_R$.

The equivalence classes of an equivalence relation on a set A form a partition of A . Conversely, an equivalence relation can be constructed from any partition so that the equivalence classes are the subsets in the partition.

The principle of well-ordered induction

The topological sorting algorithm

Review Questions

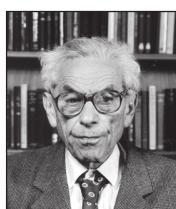
1. a) What is a relation on a set?
b) How many relations are there on a set with n elements?
2. a) What is a reflexive relation?
b) What is a symmetric relation?
c) What is an antisymmetric relation?
d) What is a transitive relation?
3. Give an example of a relation on the set $\{1, 2, 3, 4\}$ that is
 - a) reflexive, symmetric, and not transitive.
 - b) not reflexive, symmetric, and transitive.
 - c) reflexive, antisymmetric, and not transitive.
 - d) reflexive, symmetric, and transitive.
 - e) reflexive, antisymmetric, and transitive.
4. a) How many reflexive relations are there on a set with n elements?
b) How many symmetric relations are there on a set with n elements?
c) How many antisymmetric relations are there on a set with n elements?
5. a) Explain how an n -ary relation can be used to represent information about students at a university.
b) How can the 5-ary relation containing names of students, their addresses, telephone numbers, majors, and grade point averages be used to form a 3-ary relation containing the names of students, their majors, and their grade point averages?
- c) How can the 4-ary relation containing names of students, their addresses, telephone numbers, and majors and the 4-ary relation containing names of students, their student numbers, majors, and numbers of credit hours be combined into a single n -ary relation?
6. a) Explain how to use a zero-one matrix to represent a relation on a finite set.
b) Explain how to use the zero-one matrix representing a relation to determine whether the relation is reflexive, symmetric, and/or antisymmetric.
7. a) Explain how to use a directed graph to represent a relation on a finite set.
b) Explain how to use the directed graph representing a relation to determine whether a relation is reflexive, symmetric, and/or antisymmetric.
8. a) Define the reflexive closure and the symmetric closure of a relation.
b) How can you construct the reflexive closure of a relation?
c) How can you construct the symmetric closure of a relation?
d) Find the reflexive closure and the symmetric closure of the relation $\{(1, 2), (2, 3), (2, 4), (3, 1)\}$ on the set $\{1, 2, 3, 4\}$.
9. a) Define the transitive closure of a relation.
b) Can the transitive closure of a relation be obtained by including all pairs (a, c) such that (a, b) and (b, c) belong to the relation?

- c) Describe two algorithms for finding the transitive closure of a relation.
- d) Find the transitive closure of the relation $\{(1,1), (1,3), (2,1), (2,3), (2,4), (3,2), (3,4), (4,1)\}$.
10. a) Define an equivalence relation.
 b) Which relations on the set $\{a, b, c, d\}$ are equivalence relations and contain (a, b) and (b, d) ?
11. a) Show that congruence modulo m is an equivalence relation whenever m is a positive integer.
 b) Show that the relation $\{(a, b) \mid a \equiv \pm b \pmod{7}\}$ is an equivalence relation on the set of integers.
12. a) What are the equivalence classes of an equivalence relation?
 b) What are the equivalence classes of the “congruent modulo 5” relation?
 c) What are the equivalence classes of the equivalence relation in Question 11(b)?
13. Explain the relationship between equivalence relations on a set and partitions of this set.
14. a) Define a partial ordering.
 b) Show that the divisibility relation on the set of positive integers is a partial order.
15. Explain how partial orderings on the sets A_1 and A_2 can be used to define a partial ordering on the set $A_1 \times A_2$.
16. a) Explain how to construct the Hasse diagram of a partial order on a finite set.
 b) Draw the Hasse diagram of the divisibility relation on the set $\{2, 3, 5, 9, 12, 15, 18\}$.
17. a) Define a maximal element of a poset and the greatest element of a poset.
 b) Give an example of a poset that has three maximal elements.
 c) Give an example of a poset with a greatest element.
18. a) Define a lattice.
 b) Give an example of a poset with five elements that is a lattice and an example of a poset with five elements that is not a lattice.
19. a) Show that every finite subset of a lattice has a greatest lower bound and a least upper bound.
 b) Show that every lattice with a finite number of elements has a least element and a greatest element.
20. a) Define a well-ordered set.
 b) Describe an algorithm for producing a totally ordered set compatible with a given partially ordered set.
 c) Explain how the algorithm from (b) can be used to order the tasks in a project if tasks are done one at a time and each task can be done only after one or more of the other tasks have been completed.

Supplementary Exercises

- Let S be the set of all strings of English letters. Determine whether these relations are reflexive, irreflexive, symmetric, antisymmetric, and/or transitive.
 - $R_1 = \{(a, b) \mid a$ and b have no letters in common
 - $R_2 = \{(a, b) \mid a$ and b are not the same length
 - $R_3 = \{(a, b) \mid a$ is longer than $b\}$
- Construct a relation on the set $\{a, b, c, d\}$ that is
 - reflexive, symmetric, but not transitive.
 - irreflexive, symmetric, and transitive.
 - irreflexive, antisymmetric, and not transitive.
 - reflexive, neither symmetric nor antisymmetric, and transitive.
 - neither reflexive, irreflexive, symmetric, antisymmetric, nor transitive.
- Show that the relation R on $\mathbf{Z} \times \mathbf{Z}$ defined by $(a, b) R (c, d)$ if and only if $a + d = b + c$ is an equivalence relation.
- Show that a subset of an antisymmetric relation is also antisymmetric.
- Let R be a reflexive relation on a set A . Show that $R \subseteq R^2$.
- Suppose that R_1 and R_2 are reflexive relations on a set A . Show that $R_1 \oplus R_2$ is irreflexive.
- Suppose that R_1 and R_2 are reflexive relations on a set A . Is $R_1 \cap R_2$ also reflexive? Is $R_1 \cup R_2$ also reflexive?
- Suppose that R is a symmetric relation on a set A . Is \bar{R} also symmetric?
- Let R_1 and R_2 be symmetric relations. Is $R_1 \cap R_2$ also symmetric? Is $R_1 \cup R_2$ also symmetric?
- A relation R is called **circular** if $a R b$ and $b R c$ imply that $c R a$. Show that R is reflexive and circular if and only if it is an equivalence relation.
- Show that a primary key in an n -ary relation is a primary key in any projection of this relation that contains this key as one of its fields.
- Is the primary key in an n -ary relation also a primary key in a larger relation obtained by taking the join of this relation with a second relation?
- Show that the reflexive closure of the symmetric closure of a relation is the same as the symmetric closure of its reflexive closure.
- Let R be the relation on the set of all mathematicians that contains the ordered pair (a, b) if and only if a and b have written a published mathematical paper together.
 - Describe the relation R^2 .
 - Describe the relation R^* .
 - The **Erdős number** of a mathematician is 1 if this mathematician wrote a paper with the prolific Hungarian mathematician Paul Erdős, it is 2 if this mathematician did not write a joint paper with Erdős but wrote a joint paper with someone who wrote a joint paper with Erdős, and so on (except that the Erdős number of Erdős himself is 0). Give a definition of the Erdős number in terms of paths in R .

- 15.** a) Give an example to show that the transitive closure of the symmetric closure of a relation is not necessarily the same as the symmetric closure of the transitive closure of this relation.
 b) Show, however, that the transitive closure of the symmetric closure of a relation must contain the symmetric closure of the transitive closure of this relation.
- 16.** a) Let S be the set of subroutines of a computer program. Define the relation R by $\mathbf{P}R\mathbf{Q}$ if subroutine \mathbf{P} calls subroutine \mathbf{Q} during its execution. Describe the transitive closure of R .
 b) For which subroutines \mathbf{P} does (\mathbf{P}, \mathbf{P}) belong to the transitive closure of R ?
 c) Describe the reflexive closure of the transitive closure of R .
- 17.** Suppose that R and S are relations on a set A with $R \subseteq S$ such that the closures of R and S with respect to a property \mathbf{P} both exist. Show that the closure of R with respect to \mathbf{P} is a subset of the closure of S with respect to \mathbf{P} .
- 18.** Show that the symmetric closure of the union of two relations is the union of their symmetric closures.
- *19.** Devise an algorithm, based on the concept of interior vertices, that finds the length of the longest path between two vertices in a directed graph, or determines that there are arbitrarily long paths between these vertices.
- 20.** Which of these are equivalence relations on the set of all people?
- a)** $\{(x, y) \mid x \text{ and } y \text{ have the same sign of the zodiac}\}$
b) $\{(x, y) \mid x \text{ and } y \text{ were born in the same year}\}$
c) $\{(x, y) \mid x \text{ and } y \text{ have been in the same city}\}$
- *21.** How many different equivalence relations with exactly three different equivalence classes are there on a set with five elements?
- 22.** Show that $\{(x, y) \mid x - y \in \mathbf{Q}\}$ is an equivalence relation on the set of real numbers, where \mathbf{Q} denotes the set of rational numbers. What are $[1]$, $[\frac{1}{2}]$, and $[\pi]$?
- 23.** Suppose that $P_1 = \{A_1, A_2, \dots, A_m\}$ and $P_2 = \{B_1, B_2, \dots, B_n\}$ are both partitions of the set S . Show that the collection of nonempty subsets of the form $A_i \cap B_j$ is a partition of S that is a refinement of both P_1 and P_2 (see the preamble to Exercise 49 of Section 9.5).
- *24.** Show that the transitive closure of the symmetric closure of the reflexive closure of a relation R is the smallest equivalence relation that contains R .
- 25.** Let $\mathbf{R}(S)$ be the set of all relations on a set S . Define the relation \preccurlyeq on $\mathbf{R}(S)$ by $R_1 \preccurlyeq R_2$ if $R_1 \subseteq R_2$, where R_1 and R_2 are relations on S . Show that $(\mathbf{R}(S), \preccurlyeq)$ is a poset.
- 26.** Let $\mathbf{P}(S)$ be the set of all partitions of the set S . Define the relation \preccurlyeq on $\mathbf{P}(S)$ by $P_1 \preccurlyeq P_2$ if P_1 is a refinement of P_2 (see Exercise 49 of Section 9.5). Show that $(\mathbf{P}(S), \preccurlyeq)$ is a poset.



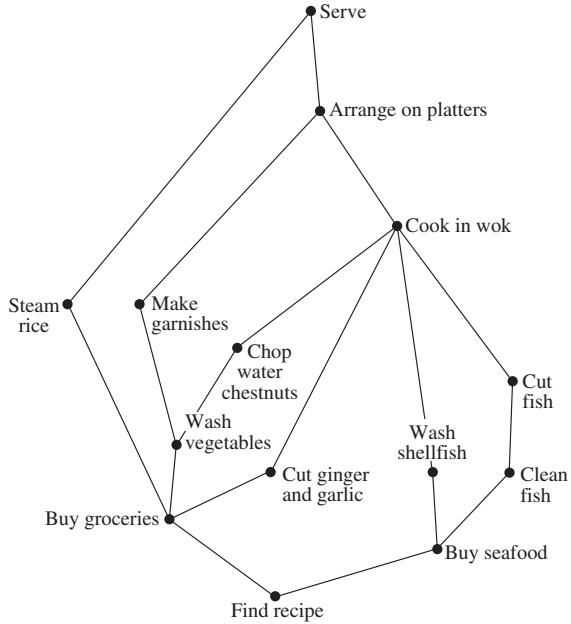
PAUL ERDŐS (1913–1996) Paul Erdős, born in Budapest, Hungary, was the son of two high school mathematics teachers. He was a child prodigy; at age 3 he could multiply three-digit numbers in his head, and at 4 he discovered negative numbers on his own. Because his mother did not want to expose him to contagious diseases, he was mostly home-schooled. At 17 Erdős entered Eötvös University, graduating four years later with a Ph.D. in mathematics. After graduating he spent four years at Manchester, England, on a postdoctoral fellowship. In 1938 he went to the United States because of the difficult political situation in Hungary, especially for Jews. He spent much of his time in the United States, except for 1954 to 1962, when he was banned as part of the paranoia of the McCarthy era. He also spent considerable time in Israel.

Erdős made many significant contributions to combinatorics and to number theory. One of the discoveries of which he was most proud is his elementary proof (in the sense that it does not use any complex analysis) of the prime number theorem, which provides an estimate for the number of primes not exceeding a fixed positive integer. He also participated in the modern development of the Ramsey theory.

Erdős traveled extensively throughout the world to work with other mathematicians, visiting conferences, universities, and research laboratories. He had no permanent home. He devoted himself almost entirely to mathematics, traveling from one mathematician to the next, proclaiming “My brain is open.” Erdős was the author or coauthor of more than 1500 papers and had more than 500 coauthors. Copies of his articles are kept by Ron Graham, a famous discrete mathematician with whom he collaborated extensively and who took care of many of his worldly needs.

Erdős offered rewards, ranging from \$10 to \$10,000, for the solution of problems that he found particularly interesting, with the size of the reward depending on the difficulty of the problem. He paid out close to \$4000. Erdős had his own special language, using such terms as “epsilon” (child), “boss” (woman), “slave” (man), “captured” (married), “liberated” (divorced), “Supreme Fascist” (God), “Sam” (United States), and “Joe” (Soviet Union). Although he was curious about many things, he concentrated almost all his energy on mathematical research. He had no hobbies and no full-time job. He never married and apparently remained celibate. Erdős was extremely generous, donating much of the money he collected from prizes, awards, and stipends for scholarships and to worthwhile causes. He traveled extremely lightly and did not like having many material possessions.

27. Schedule the tasks needed to cook a Chinese meal by specifying their order, if the Hasse diagram representing these tasks is as shown here.



A subset of a poset such that every two elements of this subset are comparable is called a **chain**. A subset of a poset is called an **antichain** if every two elements of this subset are incomparable.

28. Find all chains in the posets with the Hasse diagrams shown in Exercises 25–27 in Section 9.6.
29. Find all antichains in the posets with the Hasse diagrams shown in Exercises 25–27 in Section 9.6.
30. Find an antichain with the greatest number of elements in the poset with the Hasse diagram of Exercise 32 in Section 9.6.
31. Show that every maximal chain in a finite poset (S, \preccurlyeq) contains a minimal element of S . (A maximal chain is a chain that is not a subset of a larger chain.)
- **32. Show that every finite poset can be partitioned into k chains, where k is the largest number of elements in an antichain in this poset.
- *33. Show that in any group of $mn + 1$ people there is either a list of $m + 1$ people where a person in the list (except for the first person listed) is a descendant of the previous person on the list, or there are $n + 1$ people such that none of these people is a descendant of any of the other n people. [Hint: Use Exercise 32.]
- Suppose that (S, \preccurlyeq) is a well-founded partially ordered set. The **principle of well-founded induction** states that $P(x)$ is true for all $x \in S$ if $\forall x(\forall y(y \prec x \rightarrow P(y)) \rightarrow P(x))$.
34. Show that no separate basis case is needed for the principle of well-founded induction. That is, $P(u)$ is true for all minimal elements u in S if $\forall x(\forall y(y \prec x \rightarrow P(y)) \rightarrow P(x))$.

- *35. Show that the principle of well-founded induction is valid.

A relation R on a set A is a **quasi-ordering** on A if R is reflexive and transitive.

36. Let R be the relation on the set of all functions from \mathbf{Z}^+ to \mathbf{Z}^+ such that (f, g) belongs to R if and only if f is $O(g)$. Show that R is a quasi-ordering.
37. Let R be a quasi-ordering on a set A . Show that $R \cap R^{-1}$ is an equivalence relation.
- *38. Let R be a quasi-ordering and let S be the relation on the set of equivalence classes of $R \cap R^{-1}$ such that (C, D) belongs to S , where C and D are equivalence classes of R , if and only if there are elements c of C and d of D such that (c, d) belongs to R . Show that S is a partial ordering.
- Let L be a lattice. Define the **meet** (\wedge) and **join** (\vee) operations by $x \wedge y = \text{glb}(x, y)$ and $x \vee y = \text{lub}(x, y)$.
39. Show that the following properties hold for all elements x, y , and z of a lattice L .
- $x \wedge y = y \wedge x$ and $x \vee y = y \vee x$ (**commutative laws**)
 - $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ and $(x \vee y) \vee z = x \vee (y \vee z)$ (**associative laws**)
 - $x \wedge (x \vee y) = x$ and $x \vee (x \wedge y) = x$ (**absorption laws**)
 - $x \wedge x = x$ and $x \vee x = x$ (**idempotent laws**)
40. Show that if x and y are elements of a lattice L , then $x \vee y = y$ if and only if $x \wedge y = x$.
- A lattice L is **bounded** if it has both an **upper bound**, denoted by 1, such that $x \preccurlyeq 1$ for all $x \in L$ and a **lower bound**, denoted by 0, such that $0 \preccurlyeq x$ for all $x \in L$.
41. Show that if L is a bounded lattice with upper bound 1 and lower bound 0 then these properties hold for all elements $x \in L$.
- $x \vee 1 = 1$
 - $x \wedge 1 = x$
 - $x \vee 0 = x$
 - $x \wedge 0 = 0$
42. Show that every finite lattice is bounded.
- A lattice is called **distributive** if $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ and $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ for all x, y , and z in L .
- *43. Give an example of a lattice that is not distributive.
44. Show that the lattice $(P(S), \subseteq)$ where $P(S)$ is the power set of a finite set S is distributive.
45. Is the lattice $(\mathbf{Z}^+, |)$ distributive?
- The **complement** of an element a of a bounded lattice L with upper bound 1 and lower bound 0 is an element b such that $a \vee b = 1$ and $a \wedge b = 0$. Such a lattice is **complemented** if every element of the lattice has a complement.
46. Give an example of a finite lattice where at least one element has more than one complement and at least one element has no complement.
47. Show that the lattice $(P(S), \subseteq)$ where $P(S)$ is the power set of a finite set S is complemented.

- *48. Show that if L is a finite distributive lattice, then an element of L has at most one complement.

The game of Chomp, introduced in Example 12 in Section 1.8, can be generalized for play on any finite partially ordered set (S, \preceq) with a least element a . In this game, a move consists of selecting an element x in S and removing x and all elements larger than it from S . The loser is the player who is forced to select the least element a .

49. Show that the game of Chomp with cookies arranged in an $m \times n$ rectangular grid, described in Example 12 in Section 1.8, is the same as the game of Chomp on the poset $(S, |)$, where S is the set of all positive integers that divide $p^{m-1}q^{n-1}$, where p and q are distinct primes.
50. Show that if (S, \preceq) has a greatest element b , then a winning strategy for Chomp on this poset exists. [Hint: Generalize the argument in Example 12 in Section 1.8.]

Computer Projects

Write programs with these input and output.

1. Given the matrix representing a relation on a finite set, determine whether the relation is reflexive and/or irreflexive.
2. Given the matrix representing a relation on a finite set, determine whether the relation is symmetric and/or anti-symmetric.
3. Given the matrix representing a relation on a finite set, determine whether the relation is transitive.
4. Given a positive integer n , display all the relations on a set with n elements.
- *5. Given a positive integer n , determine the number of transitive relations on a set with n elements.
- *6. Given a positive integer n , determine the number of equivalence relations on a set with n elements.
- *7. Given a positive integer n , display all the equivalence relations on the set of the n smallest positive integers.
8. Given an n -ary relation, find the projection of this relation when specified fields are deleted.
9. Given an m -ary relation and an n -ary relation, and a set of common fields, find the join of these relations with respect to these common fields.

10. Given the matrix representing a relation on a finite set, find the matrix representing the reflexive closure of this relation.
11. Given the matrix representing a relation on a finite set, find the matrix representing the symmetric closure of this relation.
12. Given the matrix representing a relation on a finite set, find the matrix representing the transitive closure of this relation by computing the join of the Boolean powers of the matrix representing the relation.
13. Given the matrix representing a relation on a finite set, find the matrix representing the transitive closure of this relation using Warshall's algorithm.
14. Given the matrix representing a relation on a finite set, find the matrix representing the smallest equivalence relation containing this relation.
15. Given a partial ordering on a finite set, find a total ordering compatible with it using topological sorting.

Computations and Explorations

Use a computational program or programs you have written to do these exercises.

1. Display all the different relations on a set with four elements.
2. Display all the different reflexive and symmetric relations on a set with six elements.
3. Display all the reflexive and transitive relations on a set with five elements.
- *4. Determine how many transitive relations there are on a set with n elements for all positive integers n with $n \leq 7$.
5. Find the transitive closure of a relation of your choice on a set with at least 20 elements. Either use a relation that

corresponds to direct links in a particular transportation or communications network or use a randomly generated relation.

6. Compute the number of different equivalence relations on a set with n elements for all positive integers n not exceeding 20.
7. Display all the equivalence relations on a set with seven elements.
- *8. Display all the partial orders on a set with five elements.
- *9. Display all the lattices on a set with five elements.

Writing Projects

Respond to these with essays using outside sources.

1. Discuss the concept of a fuzzy relation. How are fuzzy relations used?
2. Describe the basic principles of relational databases, going beyond what was covered in Section 9.2. How widely used are relational databases as compared with other types of databases?
3. Look up the original papers by Warshall and by Roy (in French) in which they develop algorithms for finding transitive closures. Discuss their approaches. Why do you suppose that what we call Warshall's algorithm was discovered independently by more than one person?
4. Describe how equivalence classes can be used to define the rational numbers as classes of pairs of integers and how the basic arithmetic operations on rational numbers can be defined following this approach. (See Exercise 40 in Section 9.5.)
5. Explain how Helmut Hasse used what we now call Hasse diagrams.
6. Describe some of the mechanisms used to enforce information flow policies in computer operating systems.
7. Discuss the use of the Program Evaluation and Review Technique (PERT) to schedule the tasks of a large complicated project. How widely is PERT used?
8. Discuss the use of the Critical Path Method (CPM) to find the shortest time for the completion of a project. How widely is CPM used?
9. Discuss the concept of *duality* in a lattice. Explain how duality can be used to establish new results.
10. Explain what is meant by a *modular lattice*. Describe some of the properties of modular lattices and describe how modular lattices arise in the study of projective geometry.

10

Graphs

- 10.1** Graphs and Graph Models
- 10.2** Graph Terminology and Special Types of Graphs
- 10.3** Representing Graphs and Graph Isomorphism
- 10.4** Connectivity
- 10.5** Euler and Hamilton Paths
- 10.6** Shortest-Path Problems
- 10.7** Planar Graphs
- 10.8** Graph Coloring

Graphs are discrete structures consisting of vertices and edges that connect these vertices. There are different kinds of graphs, depending on whether edges have directions, whether multiple edges can connect the same pair of vertices, and whether loops are allowed. Problems in almost every conceivable discipline can be solved using graph models. We will give examples to illustrate how graphs are used as models in a variety of areas. For instance, we will show how graphs are used to represent the competition of different species in an ecological niche, how graphs are used to represent who influences whom in an organization, and how graphs are used to represent the outcomes of round-robin tournaments. We will describe how graphs can be used to model acquaintanceships between people, collaboration between researchers, telephone calls between telephone numbers, and links between websites. We will show how graphs can be used to model roadmaps and the assignment of jobs to employees of an organization.

Using graph models, we can determine whether it is possible to walk down all the streets in a city without going down a street twice, and we can find the number of colors needed to color the regions of a map. Graphs can be used to determine whether a circuit can be implemented on a planar circuit board. We can distinguish between two chemical compounds with the same molecular formula but different structures using graphs. We can determine whether two computers are connected by a communications link using graph models of computer networks. Graphs with weights assigned to their edges can be used to solve problems such as finding the shortest path between two cities in a transportation network. We can also use graphs to schedule exams and assign channels to television stations. This chapter will introduce the basic concepts of graph theory and present many different graph models. To solve the wide variety of problems that can be studied using graphs, we will introduce many different graph algorithms. We will also study the complexity of these algorithms.

10.1 Graphs and Graph Models

We begin with the definition of a graph.

DEFINITION 1

A graph $G = (V, E)$ consists of V , a nonempty set of *vertices* (or *nodes*) and E , a set of *edges*. Each edge has either one or two vertices associated with it, called its *endpoints*. An edge is said to *connect* its endpoints.

Remark: The set of vertices V of a graph G may be infinite. A graph with an infinite vertex set or an infinite number of edges is called an **infinite graph**, and in comparison, a graph with a finite vertex set and a finite edge set is called a **finite graph**. In this book we will usually consider only finite graphs.

Now suppose that a network is made up of data centers and communication links between computers. We can represent the location of each data center by a point and each communications link by a line segment, as shown in Figure 1.

This computer network can be modeled using a graph in which the vertices of the graph represent the data centers and the edges represent communication links. In general, we visualize

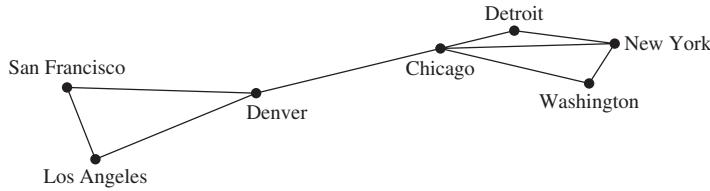


FIGURE 1 A Computer Network.

graphs by using points to represent vertices and line segments, possibly curved, to represent edges, where the endpoints of a line segment representing an edge are the points representing the endpoints of the edge. When we draw a graph, we generally try to draw edges so that they do not cross. However, this is not necessary because any depiction using points to represent vertices and any form of connection between vertices can be used. Indeed, there are some graphs that cannot be drawn in the plane without edges crossing (see Section 10.7). The key point is that the way we draw a graph is arbitrary, as long as the correct connections between vertices are depicted.

Note that each edge of the graph representing this computer network connects two different vertices. That is, no edge connects a vertex to itself. Furthermore, no two different edges connect the same pair of vertices. A graph in which each edge connects two different vertices and where no two edges connect the same pair of vertices is called a **simple graph**. Note that in a simple graph, each edge is associated to an unordered pair of vertices, and no other edge is associated to this same edge. Consequently, when there is an edge of a simple graph associated to $\{u, v\}$, we can also say, without possible confusion, that $\{u, v\}$ is an edge of the graph.

A computer network may contain multiple links between data centers, as shown in Figure 2. To model such networks we need graphs that have more than one edge connecting the same pair of vertices. Graphs that may have **multiple edges** connecting the same vertices are called **multigraphs**. When there are m different edges associated to the same unordered pair of vertices $\{u, v\}$, we also say that $\{u, v\}$ is an edge of multiplicity m . That is, we can think of this set of edges as m different copies of an edge $\{u, v\}$.

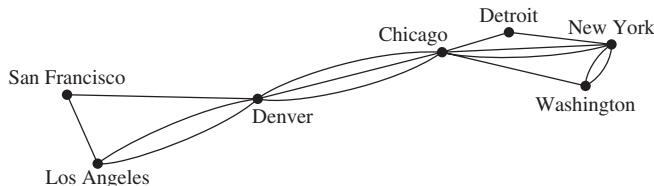


FIGURE 2 A Computer Network with Multiple Links between Data Centers.

Sometimes a communications link connects a data center with itself, perhaps a feedback loop for diagnostic purposes. Such a network is illustrated in Figure 3. To model this network we

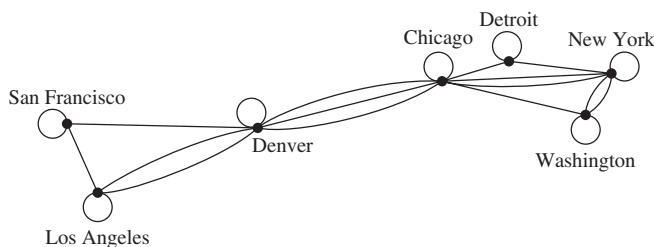


FIGURE 3 A Computer Network with Diagnostic Links.

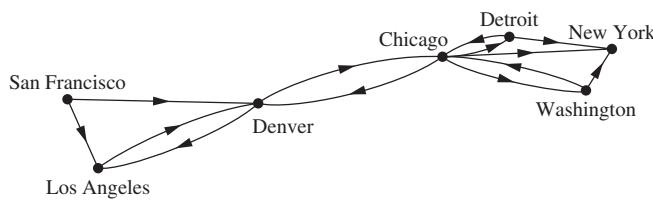


FIGURE 4 A Communications Network with One-Way Communications Links.

need to include edges that connect a vertex to itself. Such edges are called **loops**, and sometimes we may even have more than one loop at a vertex. Graphs that may include loops, and possibly multiple edges connecting the same pair of vertices or a vertex to itself, are sometimes called **pseudographs**.

So far the graphs we have introduced are **undirected graphs**. Their edges are also said to be **undirected**. However, to construct a graph model, we may find it necessary to assign directions to the edges of a graph. For example, in a computer network, some links may operate in only one direction (such links are called single duplex lines). This may be the case if there is a large amount of traffic sent to some data centers, with little or no traffic going in the opposite direction. Such a network is shown in Figure 4.

To model such a computer network we use a directed graph. Each edge of a directed graph is associated to an ordered pair. The definition of directed graph we give here is more general than the one we used in Chapter 9, where we used directed graphs to represent relations.

DEFINITION 2

A *directed graph* (or *digraph*) (V, E) consists of a nonempty set of vertices V and a set of *directed edges* (or *arcs*) E . Each directed edge is associated with an ordered pair of vertices. The directed edge associated with the ordered pair (u, v) is said to *start* at u and *end* at v .

When we depict a directed graph with a line drawing, we use an arrow pointing from u to v to indicate the direction of an edge that starts at u and ends at v . A directed graph may contain loops and it may contain multiple directed edges that start and end at the same vertices. A directed graph may also contain directed edges that connect vertices u and v in both directions; that is, when a digraph contains an edge from u to v , it may also contain one or more edges from v to u . Note that we obtain a directed graph when we assign a direction to each edge in an undirected graph. When a directed graph has no loops and has no multiple directed edges, it is called a **simple directed graph**. Because a simple directed graph has at most one edge associated to each ordered pair of vertices (u, v) , we call (u, v) an edge if there is an edge associated to it in the graph.

In some computer networks, multiple communication links between two data centers may be present, as illustrated in Figure 5. Directed graphs that may have **multiple directed edges** from a vertex to a second (possibly the same) vertex are used to model such networks. We call such graphs **directed multigraphs**. When there are m directed edges, each associated to an ordered pair of vertices (u, v) , we say that (u, v) is an edge of **multiplicity** m .

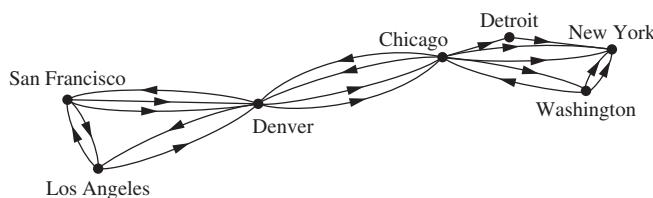


FIGURE 5 A Computer Network with Multiple One-Way Links.

TABLE 1 Graph Terminology.

Type	Edges	Multiple Edges Allowed?	Loops Allowed?
Simple graph	Undirected	No	No
Multigraph	Undirected	Yes	No
Pseudograph	Undirected	Yes	Yes
Simple directed graph	Directed	No	No
Directed multigraph	Directed	Yes	Yes
Mixed graph	Directed and undirected	Yes	Yes

For some models we may need a graph where some edges are undirected, while others are directed. A graph with both directed and undirected edges is called a **mixed graph**. For example, a mixed graph might be used to model a computer network containing links that operate in both directions and other links that operate only in one direction.

This terminology for the various types of graphs is summarized in Table 1. We will sometimes use the term **graph** as a general term to describe graphs with directed or undirected edges (or both), with or without loops, and with or without multiple edges. At other times, when the context is clear, we will use the term **graph** to refer only to undirected graphs.



Because of the relatively modern interest in graph theory, and because it has applications to a wide variety of disciplines, many different terminologies of graph theory have been introduced. The reader should determine how such terms are being used whenever they are encountered. The terminology used by mathematicians to describe graphs has been increasingly standardized, but the terminology used to discuss graphs when they are used in other disciplines is still quite varied. Although the terminology used to describe graphs may vary, three key questions can help us understand the structure of a graph:

- Are the edges of the graph undirected or directed (or both)?
- If the graph is undirected, are multiple edges present that connect the same pair of vertices?
- If the graph is directed, are multiple directed edges present?
- Are loops present?

Answering such questions helps us understand graphs. It is less important to remember the particular terminology used.

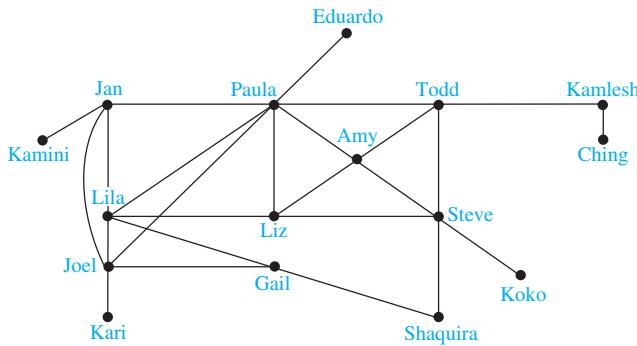
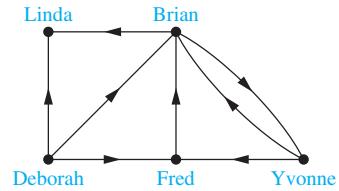
Graph Models



Can you find a subject to which graph theory has not been applied?

Graphs are used in a wide variety of models. We began this section by describing how to construct graph models of communications networks linking data centers. We will complete this section by describing some diverse graph models for some interesting applications. We will return to many of these applications later in this chapter and in Chapter 11. We will introduce additional graph models in subsequent sections of this and later chapters. Also, recall that directed graph models for some applications were introduced in Chapter 9. When we build a graph model, we need to make sure that we have correctly answered the three key questions we posed about the structure of a graph.

SOCIAL NETWORKS Graphs are extensively used to model social structures based on different kinds of relationships between people or groups of people. These social structures, and the graphs that represent them, are known as **social networks**. In these graph models, individuals or organizations are represented by vertices; relationships between individuals or organizations are represented by edges. The study of social networks is an extremely active multidisciplinary area, and many different types of relationships between people have been studied using them.

**FIGURE 6** An Acquaintanceship Graph.**FIGURE 7** An Influence Graph.

We will introduce some of the most commonly studied social networks here. More information about social networks can be found in [Ne10] and [EaK110].

EXAMPLE 1

Acquaintanceship and Friendship Graphs We can use a simple graph to represent whether two people know each other, that is, whether they are acquainted, or whether they are friends (either in the real world or in the virtual world via a social networking site such as Facebook). Each person in a particular group of people is represented by a vertex. An undirected edge is used to connect two people when these people know each other, when we are concerned only with acquaintanceship, or whether they are friends. No multiple edges and usually no loops are used. (If we want to include the notion of self-knowledge, we would include loops.) A small acquaintanceship graph is shown in Figure 6. The acquaintanceship graph of all people in the world has more than six billion vertices and probably more than one trillion edges! We will discuss this graph further in Section 10.4. 

EXAMPLE 2

Influence Graphs In studies of group behavior it is observed that certain people can influence the thinking of others. A directed graph called an **influence graph** can be used to model this behavior. Each person of the group is represented by a vertex. There is a directed edge from vertex a to vertex b when the person represented by vertex a can influence the person represented by vertex b . This graph does not contain loops and it does not contain multiple directed edges. An example of an influence graph for members of a group is shown in Figure 7. In the group modeled by this influence graph, Deborah cannot be influenced, but she can influence Brian, Fred, and Linda. Also, Yvonne and Brian can influence each other. 

EXAMPLE 3

Collaboration Graphs A **collaboration graph** is used to model social networks where two people are related by working together in a particular way. Collaboration graphs are simple graphs, as edges in these graphs are undirected and there are no multiple edges or loops. Vertices in these graphs represent people; two people are connected by an undirected edge when the people have collaborated. There are no loops nor multiple edges in these graphs. The **Hollywood graph** is a collaborator graph that represents actors by vertices and connects two actors with an edge if they have worked together on a movie or television show. The Hollywood graph is a huge graph with more than 1.5 million vertices (as of early 2011). We will discuss some aspects of the Hollywood graph later in Section 10.4.

In an **academic collaboration graph**, vertices represent people (perhaps restricted to members of a certain academic community), and edges link two people if they have jointly published a paper. The collaboration graph for people who have published research papers in mathematics was found in 2004 to have more than 400,000 vertices and 675,000 edges, and these numbers have grown considerably since then. We will have more to say about this graph in Section 10.4. Collaboration graphs have also been used in sports, where two professional athletes are considered to have collaborated if they have ever played on the same team during a regular season of their sport. 

COMMUNICATION NETWORKS We can model different communications networks using vertices to represent devices and edges to represent the particular type of communications links of interest. We have already modeled a data network in the first part of this section.

EXAMPLE 4



Call Graphs Graphs can be used to model telephone calls made in a network, such as a long-distance telephone network. In particular, a directed multigraph can be used to model calls where each telephone number is represented by a vertex and each telephone call is represented by a directed edge. The edge representing a call starts at the telephone number from which the call was made and ends at the telephone number to which the call was made. We need directed edges because the direction in which the call is made matters. We need multiple directed edges because we want to represent each call made from a particular telephone number to a second number.

A small telephone call graph is displayed in Figure 8(a), representing seven telephone numbers. This graph shows, for instance, that three calls have been made from 732-555-1234 to 732-555-9876 and two in the other direction, but no calls have been made from 732-555-4444 to any of the other six numbers except 732-555-0011. When we care only whether there has been a call connecting two telephone numbers, we use an undirected graph with an edge connecting telephone numbers when there has been a call between these numbers. This version of the call graph is displayed in Figure 8(b).

Call graphs that model actual calling activities can be huge. For example, one call graph studied at AT&T, which models calls during 20 days, has about 290 million vertices and 4 billion edges. We will discuss call graphs further in Section 10.4.

INFORMATION NETWORKS Graphs can be used to model various networks that link particular types of information. Here, we will describe how to model the World Wide Web using a graph. We will also describe how to use a graph to model the citations in different types of documents.

EXAMPLE 5



The Web Graph The World Wide Web can be modeled as a directed graph where each Web page is represented by a vertex and where an edge starts at the Web page a and ends at the Web page b if there is a link on a pointing to b . Because new Web pages are created and others removed somewhere on the Web almost every second, the Web graph changes on an almost continual basis. Many people are studying the properties of the Web graph to better understand the nature of the Web. We will return to Web graphs in Section 10.4, and in Chapter 11 we will explain how the Web graph is used by the Web crawlers that search engines use to create indexes of Web pages.

EXAMPLE 6

Citation Graphs Graphs can be used to represent citations in different types of documents, including academic papers, patents, and legal opinions. In such graphs, each document is represented by a vertex, and there is an edge from one document to a second document if the

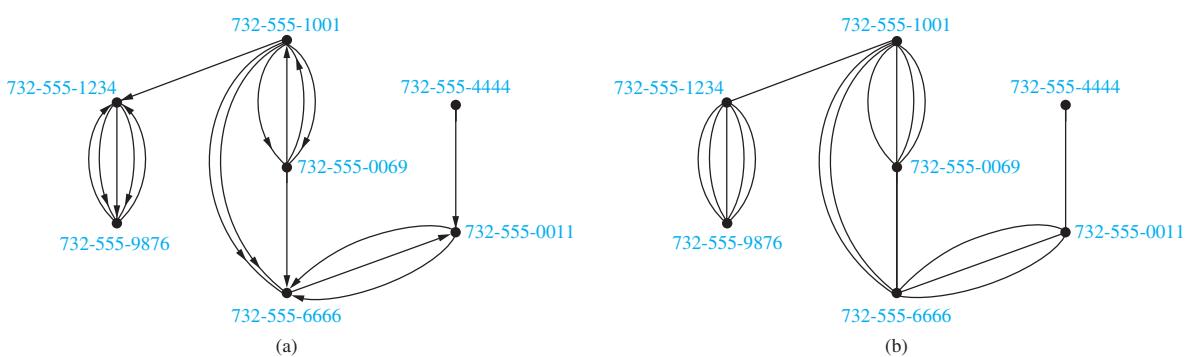


FIGURE 8 A Call Graph.

first document cites the second in its citation list. (In an academic paper, the citation list is the bibliography, or list of references; in a patent it is the list of previous patents that are cited; and in a legal opinion it is the list of previous opinions cited.) A citation graph is a directed graph without loops or multiple edges.

SOFTWARE DESIGN APPLICATIONS Graph models are useful tools in the design of software. We will briefly describe two of these models here.

EXAMPLE 7 Module Dependency Graphs One of the most important tasks in designing software is how to structure a program into different parts, or modules. Understanding how the different modules of a program interact is essential not only for program design, but also for testing and maintenance of the resulting software. A **module dependency graph** provides a useful tool for understanding how different modules of a program interact. In a program dependency graph, each module is represented by a vertex. There is a directed edge from a module to a second module if the second module depends on the first. An example of a program dependency graph for a web browser is shown in Figure 9.

EXAMPLE 8 Precedence Graphs and Concurrent Processing Computer programs can be executed more rapidly by executing certain statements concurrently. It is important not to execute a statement that requires results of statements not yet executed. The dependence of statements on previous statements can be represented by a directed graph. Each statement is represented by a vertex, and there is an edge from one statement to a second statement if the second statement cannot be executed before the first statement. This resulting graph is called a **precedence graph**. A computer program and its graph are displayed in Figure 10. For instance, the graph shows that statement S_5 cannot be executed before statements S_1 , S_2 , and S_4 are executed.

TRANSPORTATION NETWORKS We can use graphs to model many different types of transportation networks, including road, air, and rail networks, as well shipping networks.

EXAMPLE 9 Airline Routes We can model airline networks by representing each airport by a vertex. In particular, we can model all the flights by a particular airline each day using a directed edge to represent each flight, going from the vertex representing the departure airport to the vertex representing the destination airport. The resulting graph will generally be a directed multigraph, as there may be multiple flights from one airport to some other airport during the same day.

EXAMPLE 10 Road Networks Graphs can be used to model road networks. In such models, vertices represent intersections and edges represent roads. When all roads are two-way and there is at most one road connecting two intersections, we can use a simple undirected graph to model the road network. However, we will often want to model road networks when some roads are one-way and when there may be more than one road between two intersections. To build such models, we use undirected edges to represent two-way roads and we use directed edges to represent

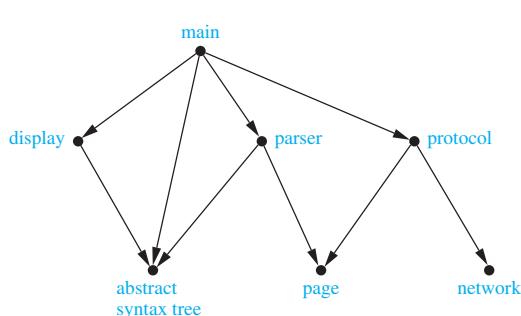


FIGURE 9 A Module Dependency Graph.

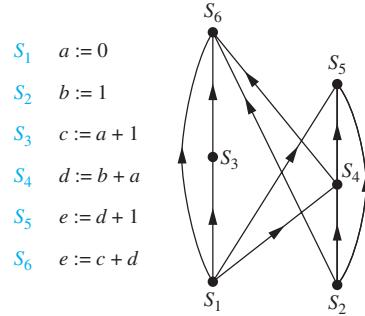


FIGURE 10 A Precedence Graph.

one-way roads. Multiple undirected edges represent multiple two-way roads connecting the same two intersections. Multiple directed edges represent multiple one-way roads that start at one intersection and end at a second intersection. Loops represent loop roads. Mixed graphs are needed to model road networks that include both one-way and two-way roads.

BIOLOGICAL NETWORKS Many aspects of the biological sciences can be modeled using graphs.

EXAMPLE 11



Niche Overlap Graphs in Ecology Graphs are used in many models involving the interaction of different species of animals. For instance, the competition between species in an ecosystem can be modeled using a **niche overlap graph**. Each species is represented by a vertex. An undirected edge connects two vertices if the two species represented by these vertices compete (that is, some of the food resources they use are the same). A niche overlap graph is a simple graph because no loops or multiple edges are needed in this model. The graph in Figure 11 models the ecosystem of a forest. We see from this graph that squirrels and raccoons compete but that crows and shrews do not.

EXAMPLE 12



Protein Interaction Graphs A protein interaction in a living cell occurs when two or more proteins in that cell bind to perform a biological function. Because protein interactions are crucial for most biological functions, many scientists work on discovering new proteins and understanding interactions between proteins. Protein interactions within a cell can be modeled using a **protein interaction graph** (also called a **protein–protein interaction network**), an undirected graph in which each protein is represented by a vertex, with an edge connecting the vertices representing each pair of proteins that interact. It is a challenging problem to determine genuine protein interactions in a cell, as experiments often produce false positives, which conclude that two proteins interact when they really do not. Protein interaction graphs can be used to deduce important biological information, such as by identifying the most important proteins for various functions and the functionality of newly discovered proteins.

Because there are thousands of different proteins in a typical cell, the protein interaction graph of a cell is extremely large and complex. For example, yeast cells have more than 6,000 proteins, and more than 80,000 interactions between them are known, and human cells have more than 100,000 proteins, with perhaps as many as 1,000,000 interactions between them. Additional vertices and edges are added to a protein interaction graph when new proteins and interactions between proteins are discovered. Because of the complexity of protein interaction graphs, they are often split into smaller graphs called modules that represent groups of proteins that are involved in a particular function of a cell. Figure 12 illustrates a module of the protein interaction graph described in [Bo04], comprising the complex of proteins that degrade RNA in human cells. To learn more about protein interaction graphs, see [Bo04], [Ne10], and [Hu07].

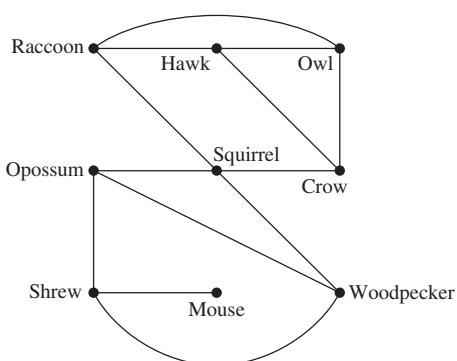


FIGURE 11 A Niche Overlap Graph.

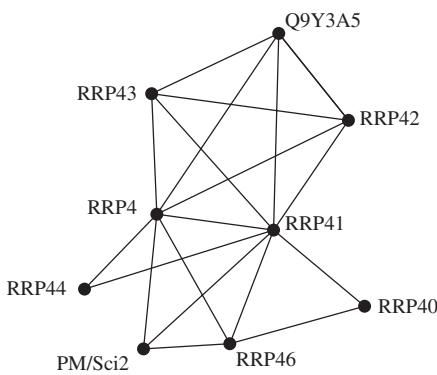


FIGURE 12 A Module of a Protein Interaction Graph.

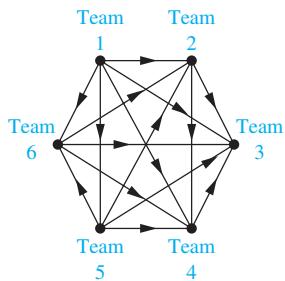


FIGURE 13 A Graph Model of a Round-Robin Tournament.

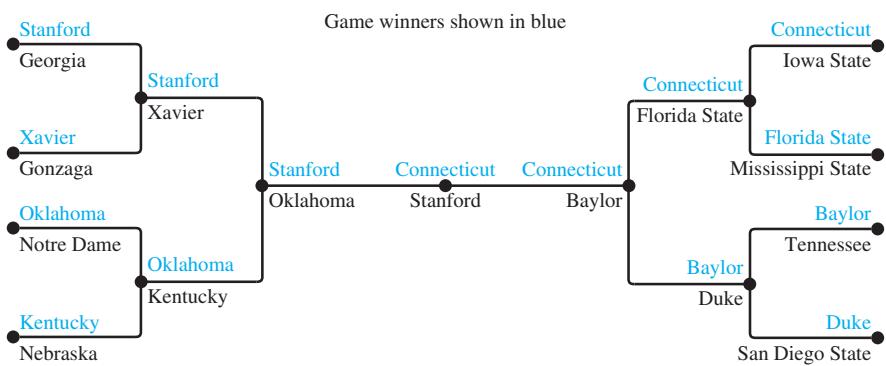


FIGURE 14 A Single-Elimination Tournament.

TOURNAMENTS We now give some examples that show how graphs can also be used to model different kinds of tournaments.

EXAMPLE 13

Round-Robin Tournaments A tournament where each team plays every other team exactly once and no ties are allowed is called a **round-robin tournament**. Such tournaments can be modeled using directed graphs where each team is represented by a vertex. Note that (a, b) is an edge if team a beats team b . This graph is a simple directed graph, containing no loops or multiple directed edges (because no two teams play each other more than once). Such a directed graph model is presented in Figure 13. We see that Team 1 is undefeated in this tournament, and Team 3 is winless. 

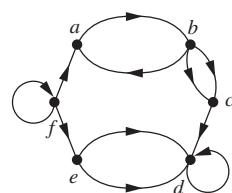
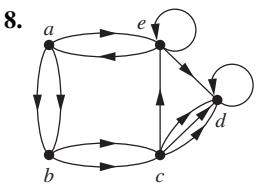
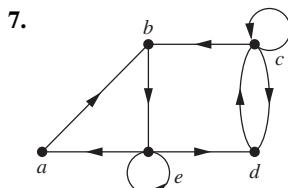
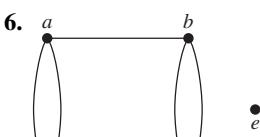
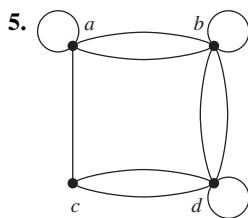
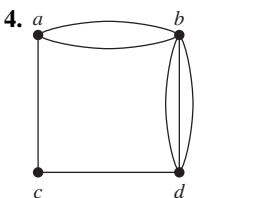
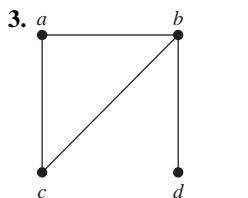
EXAMPLE 14

Single-Elimination Tournaments A tournament where each contestant is eliminated after one loss is called a **single-elimination tournament**. Single-elimination tournaments are often used in sports, including tennis championships and the yearly NCAA basketball championship. We can model such a tournament using a vertex to represent each game and a directed edge to connect a game to the next game the winner of this game played in. The graph in Figure 14 represents the games played by the final 16 teams in the 2010 NCAA women's basketball tournament. 

Exercises

- Draw graph models, stating the type of graph (from Table 1) used, to represent airline routes where every day there are four flights from Boston to Newark, two flights from Newark to Boston, three flights from Newark to Miami, two flights from Miami to Newark, one flight from Newark to Detroit, two flights from Detroit to Newark, three flights from Newark to Washington, two flights from Washington to Newark, and one flight from Washington to Miami, with
 - an edge between vertices representing cities that have a flight between them (in either direction).
 - an edge between vertices representing cities for each flight that operates between them (in either direction).
 - an edge between vertices representing cities for each flight that operates between them (in either direction), plus a loop for a special sightseeing trip that takes off and lands in Miami.
- What kind of graph (from Table 1) can be used to model a highway system between major cities where
 - there is an edge between the vertices representing cities if there is an interstate highway between them?
 - there is an edge between the vertices representing cities for each interstate highway between them?
 - there is an edge between the vertices representing cities for each interstate highway between them, and there is a loop at the vertex representing a city if there is an interstate highway that circles this city?

For Exercises 3–9, determine whether the graph shown has directed or undirected edges, whether it has multiple edges, and whether it has one or more loops. Use your answers to determine the type of graph in Table 1 this graph is.



10. For each undirected graph in Exercises 3–9 that is not simple, find a set of edges to remove to make it simple.

11. Let G be a simple graph. Show that the relation R on the set of vertices of G such that uRv if and only if there is an edge associated to $\{u, v\}$ is a symmetric, irreflexive relation on G .

12. Let G be an undirected graph with a loop at every vertex. Show that the relation R on the set of vertices of G such that uRv if and only if there is an edge associated to $\{u, v\}$ is a symmetric, reflexive relation on G .

13. The **intersection graph** of a collection of sets A_1, A_2, \dots, A_n is the graph that has a vertex for each of these sets and has an edge connecting the vertices representing two sets if these sets have a nonempty intersection. Construct the intersection graph of these collections of sets.

a) $A_1 = \{0, 2, 4, 6, 8\}, A_2 = \{0, 1, 2, 3, 4\}, A_3 = \{1, 3, 5, 7, 9\}, A_4 = \{5, 6, 7, 8, 9\}, A_5 = \{0, 1, 8, 9\}$

b) $A_1 = \{\dots, -4, -3, -2, -1, 0\}, A_2 = \{\dots, -2, -1, 0, 1, 2, \dots\}, A_3 = \{\dots, -6, -4, -2, 0, 2, 4, 6, \dots\}, A_4 = \{\dots, -5, -3, -1, 1, 3, 5, \dots\}, A_5 = \{\dots, -6, -3, 0, 3, 6, \dots\}$

c) $A_1 = \{x \mid x < 0\}, A_2 = \{x \mid -1 < x < 0\}, A_3 = \{x \mid 0 < x < 1\}, A_4 = \{x \mid -1 < x < 1\}, A_5 = \{x \mid x > -1\}, A_6 = \mathbf{R}$

14. Use the niche overlap graph in Figure 11 to determine the species that compete with hawks.
15. Construct a niche overlap graph for six species of birds, where the hermit thrush competes with the robin and with the blue jay, the robin also competes with the mockingbird, the mockingbird also competes with the blue jay, and the nuthatch competes with the hairy wood-pecker.
16. Draw the acquaintanceship graph that represents that Tom and Patricia, Tom and Hope, Tom and Sandy, Tom and Amy, Tom and Marika, Jeff and Patricia, Jeff and Mary, Patricia and Hope, Amy and Hope, and Amy and Marika know each other, but none of the other pairs of people listed know each other.
17. We can use a graph to represent whether two people were alive at the same time. Draw such a graph to represent whether each pair of the mathematicians and computer scientists with biographies in the first five chapters of this book who died before 1900 were contemporaneous. (Assume two people lived at the same time if they were alive during the same year.)
18. Who can influence Fred and whom can Fred influence in the influence graph in Example 2?
19. Construct an influence graph for the board members of a company if the President can influence the Director of Research and Development, the Director of Marketing, and the Director of Operations; the Director of Research and Development can influence the Director of Operations; the Director of Marketing can influence the Director of Operations; and no one can influence, or be influenced by, the Chief Financial Officer.
20. Which other teams did Team 4 beat and which teams beat Team 4 in the round-robin tournament represented by the graph in Figure 13?
21. In a round-robin tournament the Tigers beat the Blue Jays, the Tigers beat the Cardinals, the Tigers beat the Orioles, the Blue Jays beat the Cardinals, the Blue Jays beat the Orioles, and the Cardinals beat the Orioles. Model this outcome with a directed graph.
22. Construct the call graph for a set of seven telephone numbers 555-0011, 555-1221, 555-1333, 555-8888, 555-2222, 555-0091, and 555-1200 if there were three calls from 555-0011 to 555-8888 and two calls from 555-8888 to 555-0011, two calls from 555-2222 to 555-0091, two calls from 555-1221 to each of the other numbers, and one call from 555-1333 to each of 555-0011, 555-1221, and 555-1200.
23. Explain how the two telephone call graphs for calls made during the month of January and calls made during the month of February can be used to determine the new telephone numbers of people who have changed their telephone numbers.

- 24.** a) Explain how graphs can be used to model electronic mail messages in a network. Should the edges be directed or undirected? Should multiple edges be allowed? Should loops be allowed?
 b) Describe a graph that models the electronic mail sent in a network in a particular week.
- 25.** How can a graph that models e-mail messages sent in a network be used to find people who have recently changed their primary e-mail address?
- 26.** How can a graph that models e-mail messages sent in a network be used to find electronic mail mailing lists used to send the same message to many different e-mail addresses?
- 27.** Describe a graph model that represents whether each person at a party knows the name of each other person at the party. Should the edges be directed or undirected? Should multiple edges be allowed? Should loops be allowed?
- 28.** Describe a graph model that represents a subway system in a large city. Should edges be directed or undirected? Should multiple edges be allowed? Should loops be allowed?
- 29.** For each course at a university, there may be one or more other courses that are its prerequisites. How can a graph be used to model these courses and which courses are prerequisites for which courses? Should edges be directed or undirected? Looking at the graph model, how can we find courses that do not have any prerequisites and how can we find courses that are not the prerequisite for any other courses?
- 30.** Describe a graph model that represents the positive recommendations of movie critics, using vertices to represent both these critics and all movies that are currently being shown.
- 31.** Describe a graph model that represents traditional marriages between men and women. Does this graph have any special properties?
- 32.** Which statements must be executed before S_6 is executed in the program in Example 8? (Use the precedence graph in Figure 10.)
- 33.** Construct a precedence graph for the following program:
- ```

 $S_1: x := 0$
 $S_2: x := x + 1$
 $S_3: y := 2$
 $S_4: z := y$
 $S_5: x := x + 2$
 $S_6: y := x + z$
 $S_7: z := 4$

```
- 34.** Describe a discrete structure based on a graph that can be used to model airline routes and their flight times. [Hint: Add structure to a directed graph.]
- 35.** Describe a discrete structure based on a graph that can be used to model relationships between pairs of individuals in a group, where each individual may either like, dislike, or be neutral about another individual, and the reverse relationship may be different. [Hint: Add structure to a directed graph. Treat separately the edges in opposite directions between vertices representing two individuals.]
- 36.** Describe a graph model that can be used to represent all forms of electronic communication between two people in a single graph. What kind of graph is needed?

## 10.2 Graph Terminology and Special Types of Graphs

### Introduction



We introduce some of the basic vocabulary of graph theory in this section. We will use this vocabulary later in this chapter when we solve many different types of problems. One such problem involves determining whether a graph can be drawn in the plane so that no two of its edges cross. Another example is deciding whether there is a one-to-one correspondence between the vertices of two graphs that produces a one-to-one correspondence between the edges of the graphs. We will also introduce several important families of graphs often used as examples and in models. Several important applications will be described where these special types of graphs arise.

### Basic Terminology

First, we give some terminology that describes the vertices and edges of undirected graphs.

#### DEFINITION 1

Two vertices  $u$  and  $v$  in an undirected graph  $G$  are called *adjacent* (or *neighbors*) in  $G$  if  $u$  and  $v$  are endpoints of an edge  $e$  of  $G$ . Such an edge  $e$  is called *incident with* the vertices  $u$  and  $v$  and  $e$  is said to *connect*  $u$  and  $v$ .

We will also find useful terminology describing the set of vertices adjacent to a particular vertex of a graph.

### DEFINITION 2

The set of all neighbors of a vertex  $v$  of  $G = (V, E)$ , denoted by  $N(v)$ , is called the *neighborhood* of  $v$ . If  $A$  is a subset of  $V$ , we denote by  $N(A)$  the set of all vertices in  $G$  that are adjacent to at least one vertex in  $A$ . So,  $N(A) = \bigcup_{v \in A} N(v)$ .

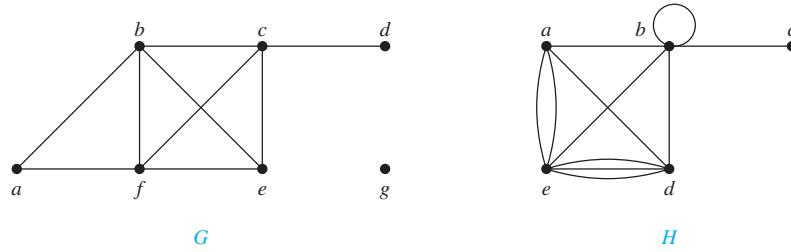
To keep track of how many edges are incident to a vertex, we make the following definition.

### DEFINITION 3

The *degree* of a vertex in an undirected graph is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex  $v$  is denoted by  $\deg(v)$ .

**EXAMPLE 1** What are the degrees and what are the neighborhoods of the vertices in the graphs  $G$  and  $H$  displayed in Figure 1?

**Solution:** In  $G$ ,  $\deg(a) = 2$ ,  $\deg(b) = \deg(c) = \deg(f) = 4$ ,  $\deg(d) = 1$ ,  $\deg(e) = 3$ , and  $\deg(g) = 0$ . The neighborhoods of these vertices are  $N(a) = \{b, f\}$ ,  $N(b) = \{a, c, e, f\}$ ,  $N(c) = \{b, d, e, f\}$ ,  $N(d) = \{c\}$ ,  $N(e) = \{b, c, f\}$ ,  $N(f) = \{a, b, c, e\}$ , and  $N(g) = \emptyset$ . In  $H$ ,  $\deg(a) = 4$ ,  $\deg(b) = \deg(e) = 6$ ,  $\deg(c) = 1$ , and  $\deg(d) = 5$ . The neighborhoods of these vertices are  $N(a) = \{b, d, e\}$ ,  $N(b) = \{a, b, c, d, e\}$ ,  $N(c) = \{b\}$ ,  $N(d) = \{a, b, e\}$ , and  $N(e) = \{a, b, d\}$ . 



**FIGURE 1** The Undirected Graphs  $G$  and  $H$ .

A vertex of degree zero is called **isolated**. It follows that an isolated vertex is not adjacent to any vertex. Vertex  $g$  in graph  $G$  in Example 1 is isolated. A vertex is **pendant** if and only if it has degree one. Consequently, a pendant vertex is adjacent to exactly one other vertex. Vertex  $d$  in graph  $G$  in Example 1 is pendant.

Examining the degrees of vertices in a graph model can provide useful information about the model, as Example 2 shows.

### EXAMPLE 2

What does the degree of a vertex in a niche overlap graph (introduced in Example 11 in Section 10.1) represent? Which vertices in this graph are pendant and which are isolated? Use the niche overlap graph shown in Figure 11 of Section 10.1 to interpret your answers.

**Solution:** There is an edge between two vertices in a niche overlap graph if and only if the two species represented by these vertices compete. Hence, the degree of a vertex in a niche overlap graph is the number of species in the ecosystem that compete with the species represented by this vertex. A vertex is pendant if the species competes with exactly one other species in the

ecosystem. Finally, the vertex representing a species is isolated if this species does not compete with any other species in the ecosystem.

For instance, the degree of the vertex representing the squirrel in the niche overlap graph in Figure 11 in Section 10.1 is four, because the squirrel competes with four other species: the crow, the opossum, the raccoon, and the woodpecker. In this niche overlap graph, the mouse is the only species represented by a pendant vertex, because the mouse competes only with the shrew and all other species compete with at least two other species. There are no isolated vertices in the graph in this niche overlap graph because every species in this ecosystem competes with at least one other species.  $\blacktriangleleft$

What do we get when we add the degrees of all the vertices of a graph  $G = (V, E)$ ? Each edge contributes two to the sum of the degrees of the vertices because an edge is incident with exactly two (possibly equal) vertices. This means that the sum of the degrees of the vertices is twice the number of edges. We have the result in Theorem 1, which is sometimes called the handshaking theorem (and is also often known as the handshaking lemma), because of the analogy between an edge having two endpoints and a handshake involving two hands.

### THEOREM 1

**THE HANDSHAKING THEOREM** Let  $G = (V, E)$  be an undirected graph with  $m$  edges. Then

$$2m = \sum_{v \in V} \deg(v).$$

(Note that this applies even if multiple edges and loops are present.)

### EXAMPLE 3

How many edges are there in a graph with 10 vertices each of degree six?

**Solution:** Because the sum of the degrees of the vertices is  $6 \cdot 10 = 60$ , it follows that  $2m = 60$  where  $m$  is the number of edges. Therefore,  $m = 30$ .  $\blacktriangleleft$

Theorem 1 shows that the sum of the degrees of the vertices of an undirected graph is even. This simple fact has many consequences, one of which is given as Theorem 2.

### THEOREM 2

An undirected graph has an even number of vertices of odd degree.

**Proof:** Let  $V_1$  and  $V_2$  be the set of vertices of even degree and the set of vertices of odd degree, respectively, in an undirected graph  $G = (V, E)$  with  $m$  edges. Then

$$2m = \sum_{v \in V} \deg(v) = \sum_{v \in V_1} \deg(v) + \sum_{v \in V_2} \deg(v).$$

Because  $\deg(v)$  is even for  $v \in V_1$ , the first term in the right-hand side of the last equality is even. Furthermore, the sum of the two terms on the right-hand side of the last equality is even, because this sum is  $2m$ . Hence, the second term in the sum is also even. Because all the terms in this sum are odd, there must be an even number of such terms. Thus, there are an even number of vertices of odd degree.  $\blacktriangleleft$

Terminology for graphs with directed edges reflects the fact that edges in directed graphs have directions.

**DEFINITION 4**

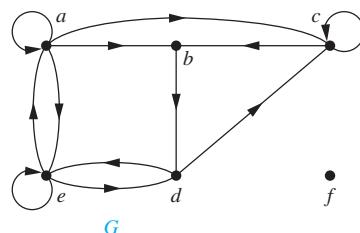
When  $(u, v)$  is an edge of the graph  $G$  with directed edges,  $u$  is said to be *adjacent to*  $v$  and  $v$  is said to be *adjacent from*  $u$ . The vertex  $u$  is called the *initial vertex* of  $(u, v)$ , and  $v$  is called the *terminal* or *end vertex* of  $(u, v)$ . The initial vertex and terminal vertex of a loop are the same.

Because the edges in graphs with directed edges are ordered pairs, the definition of the degree of a vertex can be refined to reflect the number of edges with this vertex as the initial vertex and as the terminal vertex.

**DEFINITION 5**

In a graph with directed edges the *in-degree of a vertex*  $v$ , denoted by  $\deg^-(v)$ , is the number of edges with  $v$  as their terminal vertex. The *out-degree of  $v$* , denoted by  $\deg^+(v)$ , is the number of edges with  $v$  as their initial vertex. (Note that a loop at a vertex contributes 1 to both the in-degree and the out-degree of this vertex.)

**EXAMPLE 4** Find the in-degree and out-degree of each vertex in the graph  $G$  with directed edges shown in Figure 2.



**FIGURE 2** The Directed Graph  $G$ .

**Solution:** The in-degrees in  $G$  are  $\deg^-(a) = 2$ ,  $\deg^-(b) = 2$ ,  $\deg^-(c) = 3$ ,  $\deg^-(d) = 2$ ,  $\deg^-(e) = 3$ , and  $\deg^-(f) = 0$ . The out-degrees are  $\deg^+(a) = 4$ ,  $\deg^+(b) = 1$ ,  $\deg^+(c) = 2$ ,  $\deg^+(d) = 2$ ,  $\deg^+(e) = 3$ , and  $\deg^+(f) = 0$ .  $\blacktriangleleft$

Because each edge has an initial vertex and a terminal vertex, the sum of the in-degrees and the sum of the out-degrees of all vertices in a graph with directed edges are the same. Both of these sums are the number of edges in the graph. This result is stated as Theorem 3.

**THEOREM 3**

Let  $G = (V, E)$  be a graph with directed edges. Then

$$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |E|.$$

There are many properties of a graph with directed edges that do not depend on the direction of its edges. Consequently, it is often useful to ignore these directions. The undirected graph that results from ignoring directions of edges is called the **underlying undirected graph**. A graph with directed edges and its underlying undirected graph have the same number of edges.

## Some Special Simple Graphs

We will now introduce several classes of simple graphs. These graphs are often used as examples and arise in many applications.

**EXAMPLE 5 Complete Graphs** A **complete graph on  $n$  vertices**, denoted by  $K_n$ , is a simple graph that contains exactly one edge between each pair of distinct vertices. The graphs  $K_n$ , for  $n = 1, 2, 3, 4, 5, 6$ , are displayed in Figure 3. A simple graph for which there is at least one pair of distinct vertex not connected by an edge is called **noncomplete**. 

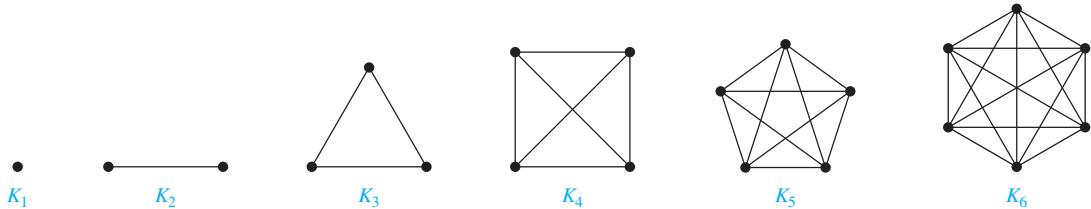


FIGURE 3 The Graphs  $K_n$  for  $1 \leq n \leq 6$ .

**EXAMPLE 6 Cycles** A **cycle**  $C_n$ ,  $n \geq 3$ , consists of  $n$  vertices  $v_1, v_2, \dots, v_n$  and edges  $\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{n-1}, v_n\}$ , and  $\{v_n, v_1\}$ . The cycles  $C_3$ ,  $C_4$ ,  $C_5$ , and  $C_6$  are displayed in Figure 4. 

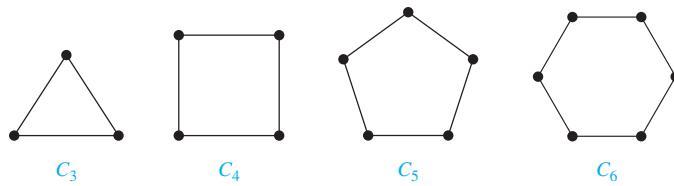


FIGURE 4 The Cycles  $C_3$ ,  $C_4$ ,  $C_5$ , and  $C_6$ .

**EXAMPLE 7 Wheels** We obtain a **wheel**  $W_n$  when we add an additional vertex to a cycle  $C_n$ , for  $n \geq 3$ , and connect this new vertex to each of the  $n$  vertices in  $C_n$ , by new edges. The wheels  $W_3$ ,  $W_4$ ,  $W_5$ , and  $W_6$  are displayed in Figure 5. 

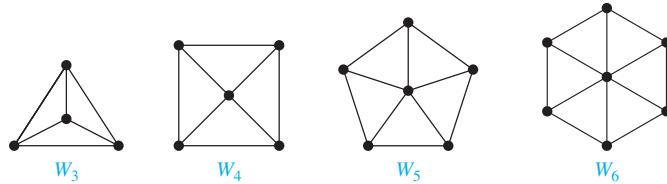
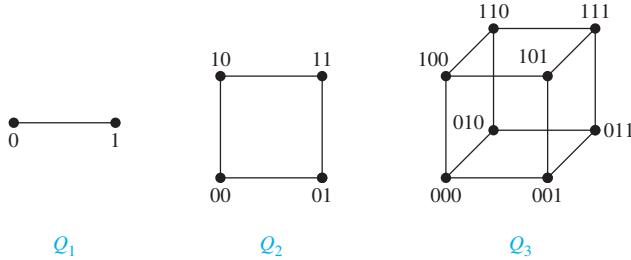


FIGURE 5 The Wheels  $W_3$ ,  $W_4$ ,  $W_5$ , and  $W_6$ .

**EXAMPLE 8  $n$ -Cubes** An  **$n$ -dimensional hypercube**, or  **$n$ -cube**, denoted by  $Q_n$ , is a graph that has vertices representing the  $2^n$  bit strings of length  $n$ . Two vertices are adjacent if and only if the bit strings that they represent differ in exactly one bit position. We display  $Q_1$ ,  $Q_2$ , and  $Q_3$  in Figure 6.

Note that you can construct the  $(n + 1)$ -cube  $Q_{n+1}$  from the  $n$ -cube  $Q_n$  by making two copies of  $Q_n$ , prefacing the labels on the vertices with a 0 in one copy of  $Q_n$  and with a 1 in the other copy of  $Q_n$ , and adding edges connecting two vertices that have labels differing only in the first bit. In Figure 6,  $Q_3$  is constructed from  $Q_2$  by drawing two copies of  $Q_2$  as the top and bottom faces of  $Q_3$ , adding 0 at the beginning of the label of each vertex in the bottom face and 1 at the beginning of the label of each vertex in the top face. (Here, by *face* we mean a face of a cube in three-dimensional space. Think of drawing the graph  $Q_3$  in three-dimensional space with copies of  $Q_2$  as the top and bottom faces of a cube and then drawing the projection of the resulting depiction in the plane.) 

**FIGURE 6** The  $n$ -cube  $Q_n$ ,  $n = 1, 2, 3$ .

## Bipartite Graphs



Sometimes a graph has the property that its vertex set can be divided into two disjoint subsets such that each edge connects a vertex in one of these subsets to a vertex in the other subset. For example, consider the graph representing marriages between men and women in a village, where each person is represented by a vertex and a marriage is represented by an edge. In this graph, each edge connects a vertex in the subset of vertices representing males and a vertex in the subset of vertices representing females. This leads us to Definition 5.

### DEFINITION 6

A simple graph  $G$  is called *bipartite* if its vertex set  $V$  can be partitioned into two disjoint sets  $V_1$  and  $V_2$  such that every edge in the graph connects a vertex in  $V_1$  and a vertex in  $V_2$  (so that no edge in  $G$  connects either two vertices in  $V_1$  or two vertices in  $V_2$ ). When this condition holds, we call the pair  $(V_1, V_2)$  a *bipartition* of the vertex set  $V$  of  $G$ .

In Example 9 we will show that  $C_6$  is bipartite, and in Example 10 we will show that  $K_3$  is not bipartite.

### EXAMPLE 9

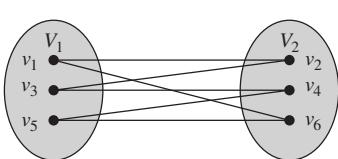
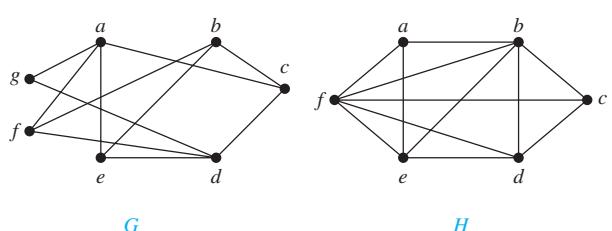
$C_6$  is bipartite, as shown in Figure 7, because its vertex set can be partitioned into the two sets  $V_1 = \{v_1, v_3, v_5\}$  and  $V_2 = \{v_2, v_4, v_6\}$ , and every edge of  $C_6$  connects a vertex in  $V_1$  and a vertex in  $V_2$ .

### EXAMPLE 10

$K_3$  is not bipartite. To verify this, note that if we divide the vertex set of  $K_3$  into two disjoint sets, one of the two sets must contain two vertices. If the graph were bipartite, these two vertices could not be connected by an edge, but in  $K_3$  each vertex is connected to every other vertex by an edge.

### EXAMPLE 11

Are the graphs  $G$  and  $H$  displayed in Figure 8 bipartite?

**FIGURE 7** Showing That  $C_6$  Is Bipartite.**FIGURE 8** The Undirected Graphs  $G$  and  $H$ .

**Solution:** Graph  $G$  is bipartite because its vertex set is the union of two disjoint sets,  $\{a, b, d\}$  and  $\{c, e, f, g\}$ , and each edge connects a vertex in one of these subsets to a vertex in the other subset. (Note that for  $G$  to be bipartite it is not necessary that every vertex in  $\{a, b, d\}$  be adjacent to every vertex in  $\{c, e, f, g\}$ . For instance,  $b$  and  $g$  are not adjacent.)

Graph  $H$  is not bipartite because its vertex set cannot be partitioned into two subsets so that edges do not connect two vertices from the same subset. (The reader should verify this by considering the vertices  $a, b$ , and  $f$ .) 

Theorem 4 provides a useful criterion for determining whether a graph is bipartite.

#### THEOREM 4

A simple graph is bipartite if and only if it is possible to assign one of two different colors to each vertex of the graph so that no two adjacent vertices are assigned the same color.

**Proof:** First, suppose that  $G = (V, E)$  is a bipartite simple graph. Then  $V = V_1 \cup V_2$ , where  $V_1$  and  $V_2$  are disjoint sets and every edge in  $E$  connects a vertex in  $V_1$  and a vertex in  $V_2$ . If we assign one color to each vertex in  $V_1$  and a second color to each vertex in  $V_2$ , then no two adjacent vertices are assigned the same color.

Now suppose that it is possible to assign colors to the vertices of the graph using just two colors so that no two adjacent vertices are assigned the same color. Let  $V_1$  be the set of vertices assigned one color and  $V_2$  be the set of vertices assigned the other color. Then,  $V_1$  and  $V_2$  are disjoint and  $V = V_1 \cup V_2$ . Furthermore, every edge connects a vertex in  $V_1$  and a vertex in  $V_2$  because no two adjacent vertices are either both in  $V_1$  or both in  $V_2$ . Consequently,  $G$  is bipartite. 

We illustrate how Theorem 4 can be used to determine whether a graph is bipartite in Example 12.

#### EXAMPLE 12

Use Theorem 4 to determine whether the graphs in Example 11 are bipartite.

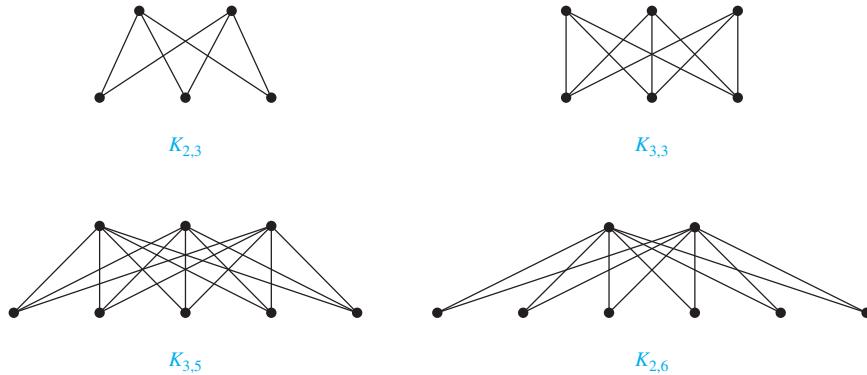
**Solution:** We first consider the graph  $G$ . We will try to assign one of two colors, say red and blue, to each vertex in  $G$  so that no edge in  $G$  connects a red vertex and a blue vertex. Without loss of generality we begin by arbitrarily assigning red to  $a$ . Then, we must assign blue to  $c, e, f$ , and  $g$ , because each of these vertices is adjacent to  $a$ . To avoid having an edge with two blue endpoints, we must assign red to all the vertices adjacent to either  $c, e, f$ , or  $g$ . This means that we must assign red to both  $b$  and  $d$  (and means that  $a$  must be assigned red, which it already has been). We have now assigned colors to all vertices, with  $a, b$ , and  $d$  red and  $c, e, f$ , and  $g$  blue. Checking all edges, we see that every edge connects a red vertex and a blue vertex. Hence, by Theorem 4 the graph  $G$  is bipartite.

Next, we will try to assign either red or blue to each vertex in  $H$  so that no edge in  $H$  connects a red vertex and a blue vertex. Without loss of generality we arbitrarily assign red to  $a$ . Then, we must assign blue to  $b, e$ , and  $f$ , because each is adjacent to  $a$ . But this is not possible because  $e$  and  $f$  are adjacent, so both cannot be assigned blue. This argument shows that we cannot assign one of two colors to each of the vertices of  $H$  so that no adjacent vertices are assigned the same color. It follows by Theorem 4 that  $H$  is not bipartite. 

Theorem 4 is an example of a result in the part of graph theory known as graph colorings. Graph colorings is an important part of graph theory with important applications. We will study graph colorings further in Section 10.8.

Another useful criterion for determining whether a graph is bipartite is based on the notion of a path, a topic we study in Section 10.4. A graph is bipartite if and only if it is not possible to start at a vertex and return to this vertex by traversing an odd number of distinct edges. We will make this notion more precise when we discuss paths and circuits in graphs in Section 10.4 (see Exercise 63 in that section).

**EXAMPLE 13 Complete Bipartite Graphs** A **complete bipartite graph**  $K_{m,n}$  is a graph that has its vertex set partitioned into two subsets of  $m$  and  $n$  vertices, respectively with an edge between two vertices if and only if one vertex is in the first subset and the other vertex is in the second subset. The complete bipartite graphs  $K_{2,3}$ ,  $K_{3,3}$ ,  $K_{3,5}$ , and  $K_{2,6}$  are displayed in Figure 9.  $\blacktriangleleft$



**FIGURE 9** Some Complete Bipartite Graphs.

## Bipartite Graphs and Matchings

Bipartite graphs can be used to model many types of applications that involve matching the elements of one set to elements of another, as Example 14 illustrates.

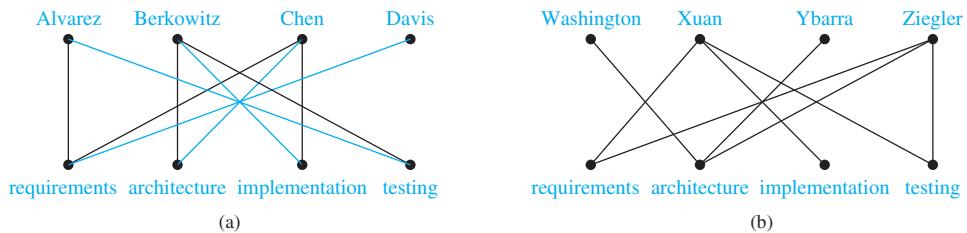
**EXAMPLE 14 Job Assignments** Suppose that there are  $m$  employees in a group and  $n$  different jobs that need to be done, where  $m \geq n$ . Each employee is trained to do one or more of these  $n$  jobs. We would like to assign an employee to each job. To help with this task, we can use a graph to model employee capabilities. We represent each employee by a vertex and each job by a vertex. For each employee, we include an edge from that employee to all jobs that the employee has been trained to do. Note that the vertex set of this graph can be partitioned into two disjoint sets, the set of employees and the set of jobs, and each edge connects an employee to a job. Consequently, this graph is bipartite, where the bipartition is  $(E, J)$  where  $E$  is the set of employees and  $J$  is the set of jobs. We now consider two different scenarios.

First, suppose that a group has four employees: Alvarez, Berkowitz, Chen, and Davis; and suppose that four jobs need to be done to complete Project 1: requirements, architecture, implementation, and testing. Suppose that Alvarez has been trained to do requirements and testing; Berkowitz has been trained to do architecture, implementation, and testing; Chen has been trained to do requirements, architecture, and implementation; and Davis has only been trained to do requirements. We model these employee capabilities using the bipartite graph in Figure 10(a).

Second, suppose that a group has second group also has four employees: Washington, Xuan, Ybarra, and Ziegler; and suppose that the same four jobs need to be done to complete Project 2 as are needed to complete Project 1. Suppose that Washington has been trained to do architecture; Xuan has been trained to do requirements, implementation, and testing; Ybarra has been trained to do architecture; and Ziegler has been trained to do requirements, architecture and testing. We model these employee capabilities using the bipartite graph in Figure 10(b).

To complete Project 1, we must assign an employee to each job so that every job has an employee assigned to it, and so that no employee is assigned more than one job. We can do this by assigning Alvarez to testing, Berkowitz to implementation, Chen to architecture, and Davis to requirements, as shown in Figure 10(a) (where blue lines show this assignment of jobs).

To complete Project 2, we must also assign an employee to each job so that every job has an employee assigned to it and no employee is assigned more than one job. However, this is



**FIGURE 10** Modeling the Jobs for Which Employees Have Been Trained.

impossible because there are only two employees, Xuan and Ziegler, who have been trained for at least one of the three jobs of requirements, implementation, and testing. Consequently, there is no way to assign three different employees to these three job so that each job is assigned an employee with the appropriate training.  $\blacktriangleleft$

Finding an assignment of jobs to employees can be thought of as finding a matching in the graph model, where a **matching**  $M$  in a simple graph  $G = (V, E)$  is a subset of the set  $E$  of edges of the graph such that no two edges are incident with the same vertex. In other words, a matching is a subset of edges such that if  $\{s, t\}$  and  $\{u, v\}$  are distinct edges of the matching, then  $s, t, u$ , and  $v$  are distinct. A vertex that is the endpoint of an edge of a matching  $M$  is said to be **matched** in  $M$ ; otherwise it is said to be **unmatched**. A **maximum matching** is a matching with the largest number of edges. We say that a matching  $M$  in a bipartite graph  $G = (V, E)$  with bipartition  $(V_1, V_2)$  is a **complete matching from  $V_1$  to  $V_2$**  if every vertex in  $V_1$  is the endpoint of an edge in the matching, or equivalently, if  $|M| = |V_1|$ . For example, to assign jobs to employees so that the largest number of jobs are assigned employees, we seek a maximum matching in the graph that models employee capabilities. To assign employees to all jobs we seek a complete matching from the set of jobs to the set of employees. In Example 14, we found a complete matching from the set of jobs to the set of employees for Project 1, and this matching is a maximum matching, and we showed that no complete matching exists from the set of jobs to the employees for Project 2.

We now give an example of how matchings can be used to model marriages.

### EXAMPLE 15

**Marriages on an Island** Suppose that there are  $m$  men and  $n$  women on an island. Each person has a list of members of the opposite gender acceptable as a spouse. We construct a bipartite graph  $G = (V_1, V_2)$  where  $V_1$  is the set of men and  $V_2$  is the set of women so that there is an edge between a man and a woman if they find each other acceptable as a spouse. A matching in this graph consists of a set of edges, where each pair of endpoints of an edge is a husband-wife pair. A maximum matching is a largest possible set of married couples, and a complete matching of  $V_1$  is a set of married couples where every man is married, but possibly not all women.  $\blacktriangleleft$

Hall's marriage theorem is an example of a theorem where obvious necessary conditions are sufficient too.

### THEOREM 5

**HALL'S MARRIAGE THEOREM** The bipartite graph  $G = (V, E)$  with bipartition  $(V_1, V_2)$  has a complete matching from  $V_1$  to  $V_2$  if and only if  $|N(A)| \geq |A|$  for all subsets  $A$  of  $V_1$ .

**Proof:** We first prove the *only if* part of the theorem. To do so, suppose that there is a complete matching  $M$  from  $V_1$  to  $V_2$ . Then, if  $A \subseteq V_1$ , for every vertex  $v \in A$ , there is an edge in  $M$  connecting  $v$  to a vertex in  $V_2$ . Consequently, there are at least as many vertices in  $V_2$  that are neighbors of vertices in  $V_1$  as there are vertices in  $V_1$ . It follows that  $|N(A)| \geq |A|$ .



To prove the *if* part of the theorem, the more difficult part, we need to show that if  $|N(A)| \geq |A|$  for all  $A \subseteq V_1$ , then there is a complete matching  $M$  from  $V_1$  to  $V_2$ . We will use strong induction on  $|V_1|$  to prove this.

**Basis step:** If  $|V_1| = 1$ , then  $V_1$  contains a single vertex  $v_0$ . Because  $|N(\{v_0\})| \geq |\{v_0\}| = 1$ , there is at least one edge connecting  $v_0$  and a vertex  $w_0 \in V_2$ . Any such edge forms a complete matching from  $V_1$  to  $V_2$ .

**Inductive step:** We first state the inductive hypothesis.

**Inductive hypothesis:** Let  $k$  be a positive integer. If  $G = (V, E)$  is a bipartite graph with bipartition  $(V_1, V_2)$ , and  $|V_1| = j \leq k$ , then there is a complete matching  $M$  from  $V_1$  to  $V_2$  whenever the condition that  $|N(A)| \geq |A|$  for all  $A \subseteq V_1$  is met.

Now suppose that  $H = (W, F)$  is a bipartite graph with bipartition  $(W_1, W_2)$  and  $|W_1| = k + 1$ . We will prove that the inductive holds using a proof by cases, using two case. Case (i) applies when for all integers  $j$  with  $1 \leq j \leq k$ , the vertices in every set of  $j$  elements from  $W_1$  are adjacent to at least  $j + 1$  elements of  $W_2$ . Case (ii) applies when for some  $j$  with  $1 \leq j \leq k$  there is a subset  $W'_1$  of  $j$  vertices such that there are exactly  $j$  neighbors of these vertices in  $W_2$ . Because either Case (i) or Case (ii) holds, we need only consider these cases to complete the inductive step.

**Case (i):** Suppose that for all integers  $j$  with  $1 \leq j \leq k$ , the vertices in every subset of  $j$  elements from  $W_1$  are adjacent to at least  $j + 1$  elements of  $W_2$ . Then, we select a vertex  $v \in W_1$  and an element  $w \in N(\{v\})$ , which must exist by our assumption that  $|N(\{v\})| \geq |\{v\}| = 1$ . We delete  $v$  and  $w$  and all edges incident to them from  $H$ . This produces a bipartite graph  $H'$  with bipartition  $(W_1 - \{v\}, W_2 - \{w\})$ . Because  $|W_1 - \{v\}| = k$ , the inductive hypothesis tells us there is a complete matching from  $W_1 - \{v\}$  to  $W_2 - \{w\}$ . Adding the edge from  $v$  to  $w$  to this complete matching produces a complete matching from  $W_1$  to  $W_2$ .

**Case (ii):** Suppose that for some  $j$  with  $1 \leq j \leq k$ , there is a subset  $W'_1$  of  $j$  vertices such that there are exactly  $j$  neighbors of these vertices in  $W_2$ . Let  $W'_2$  be the set of these neighbors. Then, by the inductive hypothesis there is a complete matching from  $W'_1$  to  $W'_2$ . Remove these  $2j$  vertices from  $W_1$  and  $W_2$  and all incident edges to produce a bipartite graph  $K$  with bipartition  $(W_1 - W'_1, W_2 - W'_2)$ .

We will show that the graph  $K$  satisfies the condition  $|N(A)| \geq |A|$  for all subsets  $A$  of  $W_1 - W'_1$ . If not, there would be a subset of  $t$  vertices of  $W_1 - W'_1$  where  $1 \leq t \leq k + 1 - j$  such that the vertices in this subset have fewer than  $t$  vertices of  $W_2 - W'_2$  as neighbors. Then, the set of  $j + t$  vertices of  $W_1$  consisting of these  $t$  vertices together with the  $j$  vertices we removed from  $W_1$  has fewer than  $j + t$  neighbors in  $W_2$ , contradicting the hypothesis that  $|N(A)| \geq |A|$  for all  $A \subseteq W_1$ .




---

**PHILIP HALL** (1904–1982) Philip Hall grew up in London, where his mother was a dressmaker. He won a scholarship for board school reserved for needy children, and later a scholarship to King's College of Cambridge University. He received his bachelors degree in 1925. In 1926, unsure of his career goals, he took a civil service exam, but decided to continue his studies at Cambridge after failing.



In 1927 Hall was elected to a fellowship at King's College; soon after, he made his first important discovery in group theory. The results he proved are now known as Hall's theorems. In 1933 he was appointed as a Lecturer at Cambridge, where he remained until 1941. During World War II he worked as a cryptographer at Bletchley Park breaking Italian and Japanese codes. At the end of the war, Hall returned to King's College, and was soon promoted. In 1953 he was appointed to the Sadleirian Chair. His work during the 1950s proved to be extremely influential to the rapid development of group theory during the 1960s.

Hall loved poetry and recited it beautifully in Italian and Japanese, as well as English. He was interested in art, music, and botany. He was quite shy and disliked large groups of people. Hall had an incredibly broad and varied knowledge, and was respected for his integrity, intellectual standards, and judgement. He was beloved by his students.

Hence, by the inductive hypothesis, the graph  $K$  has a complete matching. Combining this complete matching with the complete matching from  $W'_1$  to  $W'_2$ , we obtain a complete matching from  $W_1$  to  $W_2$ .

We have shown that in both cases there is a complete matching from  $W_1$  to  $W_2$ . This completes the inductive step and completes the proof.  $\triangleleft$

We have used strong induction to prove Hall's marriage theorem. Although our proof is elegant, it does have some drawbacks. In particular, we cannot construct an algorithm based on this proof that finds a complete matching in a bipartite graph. For a constructive proof that can be used as the basis of an algorithm, see [Gi85].

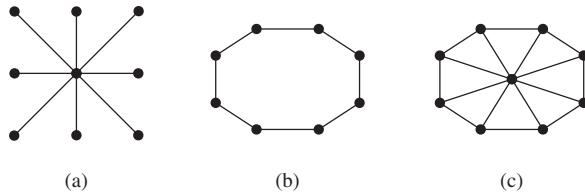
## Some Applications of Special Types of Graphs

We conclude this section by introducing some additional graph models that involve the special types of graph we have discussed in this section.

### EXAMPLE 16



**Local Area Networks** The various computers in a building, such as minicomputers and personal computers, as well as peripheral devices such as printers and plotters, can be connected using a *local area network*. Some of these networks are based on a *star topology*, where all devices are connected to a central control device. A local area network can be represented using a complete bipartite graph  $K_{1,n}$ , as shown in Figure 11(a). Messages are sent from device to device through the central control device.



**FIGURE 11** Star, Ring, and Hybrid Topologies for Local Area Networks.

Other local area networks are based on a *ring topology*, where each device is connected to exactly two others. Local area networks with a ring topology are modeled using  $n$ -cycles,  $C_n$ , as shown in Figure 11(b). Messages are sent from device to device around the cycle until the intended recipient of a message is reached.

Finally, some local area networks use a hybrid of these two topologies. Messages may be sent around the ring, or through a central device. This redundancy makes the network more reliable. Local area networks with this redundancy can be modeled using wheels  $W_n$ , as shown in Figure 11(c).  $\triangleleft$

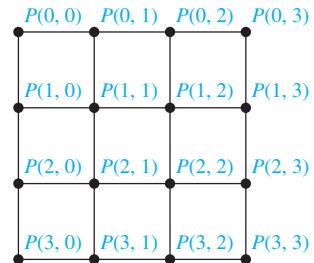
### EXAMPLE 17

**Interconnection Networks for Parallel Computation** For many years, computers executed programs one operation at a time. Consequently, the algorithms written to solve problems were designed to perform one step at a time; such algorithms are called **serial**. (Almost all algorithms described in this book are serial.) However, many computationally intense problems, such as weather simulations, medical imaging, and cryptanalysis, cannot be solved in a reasonable amount of time using serial operations, even on a supercomputer. Furthermore, there is a physical limit to how fast a computer can carry out basic operations, so there will always be problems that cannot be solved in a reasonable length of time using serial operations.

**Parallel processing**, which uses computers made up of many separate processors, each with its own memory, helps overcome the limitations of computers with a single processor. **Parallel algorithms**, which break a problem into a number of subproblems that can be solved



**FIGURE 12** A Linear Array for Six Processors.



**FIGURE 13** A Mesh Network for 16 Processors.

concurrently, can then be devised to rapidly solve problems using a computer with multiple processors. In a parallel algorithm, a single instruction stream controls the execution of the algorithm, sending subproblems to different processors, and directs the input and output of these subproblems to the appropriate processors.

When parallel processing is used, one processor may need output generated by another processor. Consequently, these processors need to be interconnected. We can use the appropriate type of graph to represent the interconnection network of the processors in a computer with multiple processors. In the following discussion, we will describe the most commonly used types of interconnection networks for parallel processors. The type of interconnection network used to implement a particular parallel algorithm depends on the requirements for exchange of data between processors, the desired speed, and, of course, the available hardware.

The simplest, but most expensive, network-interconnecting processors include a two-way link between each pair of processors. This network can be represented by  $K_n$ , the complete graph on  $n$  vertices, when there are  $n$  processors. However, there are serious problems with this type of interconnection network because the required number of connections is so large. In reality, the number of direct connections to a processor is limited, so when there are a large number of processors, a processor cannot be linked directly to all others. For example, when there are 64 processors,  $C(64, 2) = 2016$  connections would be required, and each processor would have to be directly connected to 63 others.

On the other hand, perhaps the simplest way to interconnect  $n$  processors is to use an arrangement known as a **linear array**. Each processor  $P_i$ , other than  $P_1$  and  $P_n$ , is connected to its neighbors  $P_{i-1}$  and  $P_{i+1}$  via a two-way link.  $P_1$  is connected only to  $P_2$ , and  $P_n$  is connected only to  $P_{n-1}$ . The linear array for six processors is shown in Figure 12. The advantage of a linear array is that each processor has at most two direct connections to other processors. The disadvantage is that it is sometimes necessary to use a large number of intermediate links, called **hops**, for processors to share information.

The **mesh network** (or **two-dimensional array**) is a commonly used interconnection network. In such a network, the number of processors is a perfect square, say  $n = m^2$ . The  $n$  processors are labeled  $P(i, j)$ ,  $0 \leq i \leq m - 1$ ,  $0 \leq j \leq m - 1$ . Two-way links connect processor  $P(i, j)$  with its four neighbors, processors  $P(i \pm 1, j)$  and  $P(i, j \pm 1)$ , as long as these are processors in the mesh. (Note that four processors, on the corners of the mesh, have only two adjacent processors, and other processors on the boundaries have only three neighbors. Sometimes a variant of a mesh network in which every processor has exactly four connections is used; see Exercise 72.) The mesh network limits the number of links for each processor. Communication between some pairs of processors requires  $O(\sqrt{n}) = O(m)$  intermediate links. (See Exercise 73.) The graph representing the mesh network for 16 processors is shown in Figure 13.

One important type of interconnection network is the hypercube. For such a network, the number of processors is a power of 2,  $n = 2^m$ . The  $n$  processors are labeled  $P_0, P_1, \dots, P_{n-1}$ . Each processor has two-way connections to  $m$  other processors. Processor  $P_i$  is linked to the processors with indices whose binary representations differ from the binary representation of  $i$

in exactly one bit. The hypercube network balances the number of direct connections for each processor and the number of intermediate connections required so that processors can communicate. Many computers have been built using a hypercube network, and many parallel algorithms have been devised that use a hypercube network. The graph  $Q_m$ , the  $m$ -cube, represents the hypercube network with  $n = 2^m$  processors. Figure 14 displays the hypercube network for eight processors. (Figure 14 displays a different way to draw  $Q_3$  than was shown in Figure 6.)

## New Graphs from Old

Sometimes we need only part of a graph to solve a problem. For instance, we may care only about the part of a large computer network that involves the computer centers in New York, Denver, Detroit, and Atlanta. Then we can ignore the other computer centers and all telephone lines not linking two of these specific four computer centers. In the graph model for the large network, we can remove the vertices corresponding to the computer centers other than the four of interest, and we can remove all edges incident with a vertex that was removed. When edges and vertices are removed from a graph, without removing endpoints of any remaining edges, a smaller graph is obtained. Such a graph is called a **subgraph** of the original graph.

### DEFINITION 7

A *subgraph* of a graph  $G = (V, E)$  is a graph  $H = (W, F)$ , where  $W \subseteq V$  and  $F \subseteq E$ . A subgraph  $H$  of  $G$  is a *proper subgraph* of  $G$  if  $H \neq G$ .

Given a set of vertices of a graph, we can form a subgraph of this graph with these vertices and the edges of the graph that connect them.

### DEFINITION 8

Let  $G = (V, E)$  be a simple graph. The **subgraph induced** by a subset  $W$  of the vertex set  $V$  is the graph  $(W, F)$ , where the edge set  $F$  contains an edge in  $E$  if and only if both endpoints of this edge are in  $W$ .

### EXAMPLE 18

The graph  $G$  shown in Figure 15 is a subgraph of  $K_5$ . If we add the edge connecting  $c$  and  $e$  to  $G$ , we obtain the subgraph induced by  $W = \{a, b, c, e\}$ .

**REMOVING OR ADDING EDGES OF A GRAPH** Given a graph  $G = (V, E)$  and an edge  $e \in E$ , we can produce a subgraph of  $G$  by removing the edge  $e$ . The resulting subgraph, denoted by  $G - e$ , has the same vertex set  $V$  as  $G$ . Its edge set is  $E - e$ . Hence,

$$G - e = (V, E - \{e\}).$$

Similarly, if  $E'$  is a subset of  $E$ , we can produce a subgraph of  $G$  by removing the edges in  $E'$  from the graph. The resulting subgraph has the same vertex set  $V$  as  $G$ . Its edge set is  $E - E'$ .

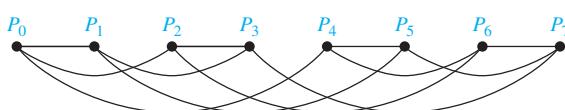


FIGURE 14 A Hypercube Network for Eight Processors.

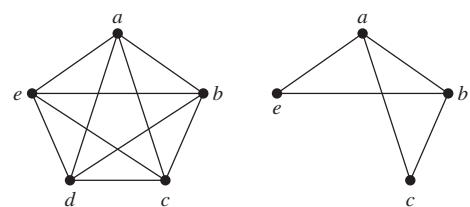
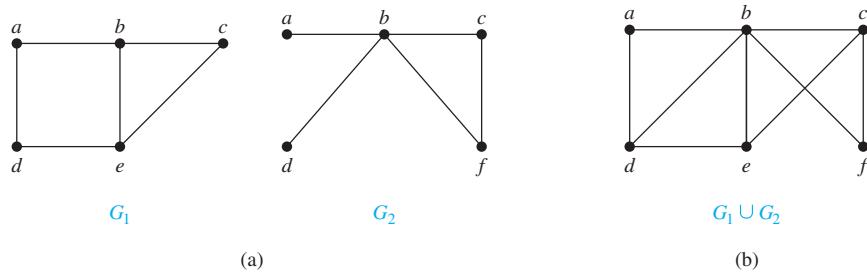


FIGURE 15 A Subgraph of  $K_5$ .

**FIGURE 16** (a) The Simple Graphs  $G_1$  and  $G_2$ ; (b) Their Union  $G_1 \cup G_2$ .

We can also add an edge  $e$  to a graph to produce a new larger graph when this edge connects two vertices already in  $G$ . We denote by  $G + e$  the new graph produced by adding a new edge  $e$ , connecting two previously nonincident vertices, to the graph  $G$ . Hence,

$$G + e = (V, E \cup \{e\}).$$

The vertex set of  $G + e$  is the same as the vertex set of  $G$  and the edge set is the union of the edge set of  $G$  and the set  $\{e\}$ .

**EDGE CONTRACTIONS** Sometimes when we remove an edge from a graph, we do not want to retain the endpoints of this edge as separate vertices in the resulting subgraph. In such a case we perform an **edge contraction** which removes an edge  $e$  with endpoints  $u$  and  $v$  and merges  $u$  and  $v$  into a new single vertex  $w$ , and for each edge with  $u$  or  $v$  as an endpoint replaces the edge with one with  $w$  as endpoint in place of  $u$  or  $v$  and with the same second endpoint. Hence, the contraction of the edge  $e$  with endpoints  $u$  and  $v$  in the graph  $G = (V, E)$  produces a new graph  $G' = (V', E')$  (which is not a subgraph of  $G$ ), where  $V' = V - \{u, v\} \cup \{w\}$  and  $E'$  contains the edges in  $E$  which do not have either  $u$  or  $v$  as endpoints and an edge connecting  $w$  to every neighbor of either  $u$  or  $v$  in  $V$ . For example, the contraction of the edge connecting the vertices  $e$  and  $c$  in the graph  $G_1$  in Figure 16 produces a new graph  $G'_1$  with vertices  $a, b, d$ , and  $w$ . As in  $G_1$ , there is an edge in  $G'_1$  connecting  $a$  and  $b$  and an edge connecting  $a$  and  $d$ . There also is an edge in  $G'_1$  that connects  $b$  and  $w$  that replaces the edges connecting  $b$  and  $c$  and connecting  $b$  and  $e$  in  $G_1$  and an edge in  $G'_1$  that connects  $d$  and  $w$  replacing the edge connecting  $d$  and  $e$  in  $G_1$ .

**REMOVING VERTICES FROM A GRAPH** When we remove a vertex  $v$  and all edges incident to it from  $G = (V, E)$ , we produce a subgraph, denoted by  $G - v$ . Observe that  $G - v = (V - v, E')$ , where  $E'$  is the set of edges of  $G$  not incident to  $v$ . Similarly, if  $V'$  is a subset of  $V$ , then the graph  $G - V'$  is the subgraph  $(V - V', E')$ , where  $E'$  is the set of edges of  $G$  not incident to a vertex in  $V'$ .

**GRAPH UNIONS** Two or more graphs can be combined in various ways. The new graph that contains all the vertices and edges of these graphs is called the **union** of the graphs. We will give a more formal definition for the union of two simple graphs.

### DEFINITION 9

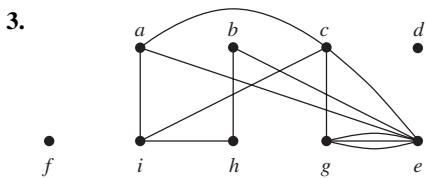
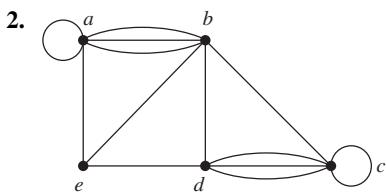
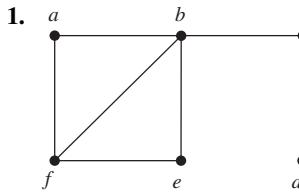
The *union* of two simple graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  is the simple graph with vertex set  $V_1 \cup V_2$  and edge set  $E_1 \cup E_2$ . The union of  $G_1$  and  $G_2$  is denoted by  $G_1 \cup G_2$ .

**EXAMPLE 19** Find the union of the graphs  $G_1$  and  $G_2$  shown in Figure 16(a).

**Solution:** The vertex set of the union  $G_1 \cup G_2$  is the union of the two vertex sets, namely,  $\{a, b, c, d, e, f\}$ . The edge set of the union is the union of the two edge sets. The union is displayed in Figure 16(b).

## Exercises

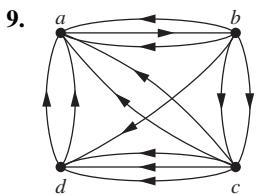
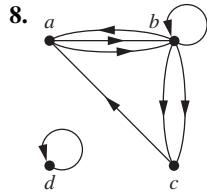
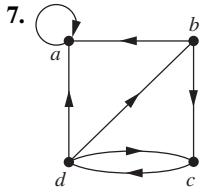
In Exercises 1–3 find the number of vertices, the number of edges, and the degree of each vertex in the given undirected graph. Identify all isolated and pendant vertices.



4. Find the sum of the degrees of the vertices of each graph in Exercises 1–3 and verify that it equals twice the number of edges in the graph.
5. Can a simple graph exist with 15 vertices each of degree five?

6. Show that the sum, over the set of people at a party, of the number of people a person has shaken hands with, is even. Assume that no one shakes his or her own hand.

In Exercises 7–9 determine the number of vertices and edges and find the in-degree and out-degree of each vertex for the given directed multigraph.



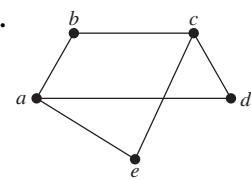
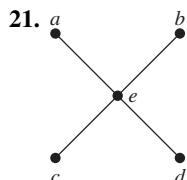
10. For each of the graphs in Exercises 7–9 determine the sum of the in-degrees of the vertices and the sum of the out-degrees of the vertices directly. Show that they are both equal to the number of edges in the graph.
11. Construct the underlying undirected graph for the graph with directed edges in Figure 2.
12. What does the degree of a vertex represent in the acquaintanceship graph, where vertices represent all the people in the world? What does the neighborhood a vertex in this graph represent? What do isolated and pendant vertices in this graph represent? In one study it was estimated that the average degree of a vertex in this graph is 1000. What does this mean in terms of the model?
13. What does the degree of a vertex represent in an academic collaboration graph? What does the neighborhood of a vertex represent? What do isolated and pendant vertices represent?
14. What does the degree of a vertex in the Hollywood graph represent? What does the neighborhood of a vertex represent? What do the isolated and pendant vertices represent?
15. What do the in-degree and the out-degree of a vertex in a telephone call graph, as described in Example 4 of Section 10.1, represent? What does the degree of a vertex in the undirected version of this graph represent?
16. What do the in-degree and the out-degree of a vertex in the Web graph, as described in Example 5 of Section 10.1, represent?
17. What do the in-degree and the out-degree of a vertex in a directed graph modeling a round-robin tournament represent?
18. Show that in a simple graph with at least two vertices there must be two vertices that have the same degree.
19. Use Exercise 18 to show that in a group of people, there must be two people who are friends with the same number of other people in the group.
20. Draw these graphs.

a)  $K_7$   
d)  $C_7$

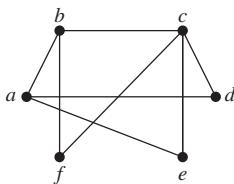
b)  $K_{1,8}$   
e)  $W_7$

c)  $K_{4,4}$   
f)  $Q_4$

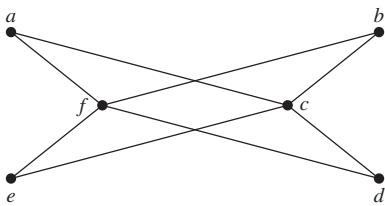
In Exercises 21–25 determine whether the graph is bipartite. You may find it useful to apply Theorem 4 and answer the question by determining whether it is possible to assign either red or blue to each vertex so that no two adjacent vertices are assigned the same color.



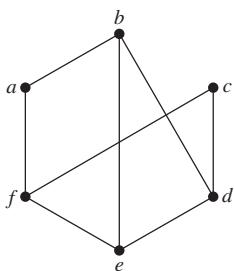
23.



24.



25.

26. For which values of  $n$  are these graphs bipartite?

- a)  $K_n$     b)  $C_n$     c)  $W_n$     d)  $Q_n$

27. Suppose that there are four employees in the computer support group of the School of Engineering of a large university. Each employee will be assigned to support one of four different areas: hardware, software, networking, and wireless. Suppose that Ping is qualified to support hardware, networking, and wireless; Quiggle is qualified to support software and networking; Ruiz is qualified to support networking and wireless, and Sitea is qualified to support hardware and software.

- a) Use a bipartite graph to model the four employees and their qualifications.  
 b) Use Hall's theorem to determine whether there is an assignment of employees to support areas so that each employee is assigned one area to support.  
 c) If an assignment of employees to support areas so that each employee is assigned to one support area exists, find one.

28. Suppose that a new company has five employees: Zamora, Agharam, Smith, Chou, and Macintyre. Each employee will assume one of six responsibilities: planning, publicity, sales, marketing, development, and industry relations. Each employee is capable of doing one or more of these jobs: Zamora could do planning, sales, marketing, or industry relations; Agharam could do planning or development; Smith could do publicity, sales, or industry relations; Chou could do planning, sales, or industry relations; and Macintyre could do planning, publicity, sales, or industry relations.

- a) Model the capabilities of these employees using a bipartite graph.  
 b) Find an assignment of responsibilities such that each employee is assigned one responsibility.

- c) Is the matching of responsibilities you found in part (b) a complete matching? Is it a maximum matching?

29. Suppose that there are five young women and five young men on an island. Each man is willing to marry some of the women on the island and each woman is willing to marry any man who is willing to marry her. Suppose that Sandeep is willing to marry Tina and Vandana; Barry is willing to marry Tina, Xia, and Uma; Teja is willing to marry Tina and Zelda; Anil is willing to marry Vandana and Zelda; and Emilio is willing to marry Tina and Zelda. Use Hall's theorem to show there is no matching of the young men and young women on the island such that each young man is matched with a young woman he is willing to marry.

30. Suppose that there are five young women and six young men on an island. Each woman is willing to marry some of the men on the island and each man is willing to marry any woman who is willing to marry him. Suppose that Anna is willing to marry Jason, Larry, and Matt; Barbara is willing to marry Kevin and Larry; Carol is willing to marry Jason, Nick, and Oscar; Diane is willing to marry Jason, Larry, Nick, and Oscar; and Elizabeth is willing to marry Jason and Matt.

- a) Model the possible marriages on the island using a bipartite graph.  
 b) Find a matching of the young women and the young men on the island such that each young woman is matched with a young man whom she is willing to marry.  
 c) Is the matching you found in part (b) a complete matching? Is it a maximum matching?

\*31. Suppose there is an integer  $k$  such that every man on a desert island is willing to marry exactly  $k$  of the women on the island and every woman on the island is willing to marry exactly  $k$  of the men. Also, suppose that a man is willing to marry a woman if and only if she is willing to marry him. Show that it is possible to match the men and women on the island so that everyone is matched with someone that they are willing to marry.

\*32. In this exercise we prove a theorem of Øystein Ore. Suppose that  $G = (V, E)$  is a bipartite graph with bipartition  $(V_1, V_2)$  and that  $A \subseteq V_1$ . Show that the maximum number of vertices of  $V_1$  that are the endpoints of a matching of  $G$  equals  $|V_1| - \max_{A \subseteq V_1} \text{def}(A)$ , where  $\text{def}(A) = |A| - |N(A)|$ . (Here,  $\text{def}(A)$  is called the **deficiency** of  $A$ .) [Hint: Form a larger graph by adding  $\max_{A \subseteq V_1} \text{def}(A)$  new vertices to  $V_2$  and connect all of them to the vertices of  $V_1$ .]

33. For the graph  $G$  in Exercise 1 find  
 a) the subgraph induced by the vertices  $a, b, c$ , and  $f$ .  
 b) the new graph  $G_1$  obtained from  $G$  by contracting the edge connecting  $b$  and  $f$ .  
 34. Let  $n$  be a positive integer. Show that a subgraph induced by a nonempty subset of the vertex set of  $K_n$  is a complete graph.

35. How many vertices and how many edges do these graphs have?

- a)  $K_n$       b)  $C_n$       c)  $W_n$   
 d)  $K_{m,n}$       e)  $Q_n$

The **degree sequence** of a graph is the sequence of the degrees of the vertices of the graph in nonincreasing order. For example, the degree sequence of the graph  $G$  in Example 1 is 4, 4, 4, 3, 2, 1, 0.

36. Find the degree sequences for each of the graphs in Exercises 21–25.

37. Find the degree sequence of each of the following graphs.

- a)  $K_4$       b)  $C_4$       c)  $W_4$   
 d)  $K_{2,3}$       e)  $Q_3$

38. What is the degree sequence of the bipartite graph  $K_{m,n}$ , where  $m$  and  $n$  are positive integers? Explain your answer.

39. What is the degree sequence of  $K_n$ , where  $n$  is a positive integer? Explain your answer.

40. How many edges does a graph have if its degree sequence is 4, 3, 3, 2, 2? Draw such a graph.

41. How many edges does a graph have if its degree sequence is 5, 2, 2, 2, 1? Draw such a graph.

A sequence  $d_1, d_2, \dots, d_n$  is called **graphic** if it is the degree sequence of a simple graph.

42. Determine whether each of these sequences is graphic. For those that are, draw a graph having the given degree sequence.

- a) 5, 4, 3, 2, 1, 0    b) 6, 5, 4, 3, 2, 1    c) 2, 2, 2, 2, 2, 2  
 d) 3, 3, 3, 2, 2, 2    e) 3, 3, 2, 2, 2, 2    f) 1, 1, 1, 1, 1, 1  
 g) 5, 3, 3, 3, 3, 3    h) 5, 5, 4, 3, 2, 1

43. Determine whether each of these sequences is graphic. For those that are, draw a graph having the given degree sequence.

- a) 3, 3, 3, 3, 2    b) 5, 4, 3, 2, 1    c) 4, 4, 3, 2, 1  
 d) 4, 4, 3, 3, 3    e) 3, 2, 2, 1, 0    f) 1, 1, 1, 1, 1

- \*44. Suppose that  $d_1, d_2, \dots, d_n$  is a graphic sequence. Show that there is a simple graph with vertices  $v_1, v_2, \dots, v_n$  such that  $\deg(v_i) = d_i$  for  $i = 1, 2, \dots, n$  and  $v_1$  is adjacent to  $v_2, \dots, v_{d_1+1}$ .

- \*45. Show that a sequence  $d_1, d_2, \dots, d_n$  of nonnegative integers in nonincreasing order is a graphic sequence if and only if the sequence obtained by reordering the terms of the sequence  $d_2 - 1, \dots, d_{d_1+1} - 1, d_{d_1+2}, \dots, d_n$  so that the terms are in nonincreasing order is a graphic sequence.

- \*46. Use Exercise 45 to construct a recursive algorithm for determining whether a nonincreasing sequence of positive integers is graphic.

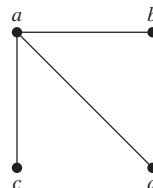
47. Show that every nonincreasing sequence of nonnegative integers with an even sum of its terms is the degree sequence of a pseudograph, that is, an undirected graph where loops are allowed. [Hint: Construct such a graph by first adding as many loops as possible at each vertex. Then add additional edges connecting vertices of odd degree. Explain why this construction works.]

48. How many subgraphs with at least one vertex does  $K_2$  have?

49. How many subgraphs with at least one vertex does  $K_3$  have?

50. How many subgraphs with at least one vertex does  $W_3$  have?

51. Draw all subgraphs of this graph.



52. Let  $G$  be a graph with  $v$  vertices and  $e$  edges. Let  $M$  be the maximum degree of the vertices of  $G$ , and let  $m$  be the minimum degree of the vertices of  $G$ . Show that

a)  $2e/v \geq m$ .      b)  $2e/v \leq M$ .

A simple graph is called **regular** if every vertex of this graph has the same degree. A regular graph is called  **$n$ -regular** if every vertex in this graph has degree  $n$ .

53. For which values of  $n$  are these graphs regular?

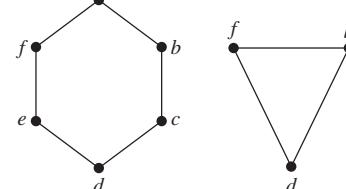
- a)  $K_n$       b)  $C_n$       c)  $W_n$       d)  $Q_n$

54. For which values of  $m$  and  $n$  is  $K_{m,n}$  regular?

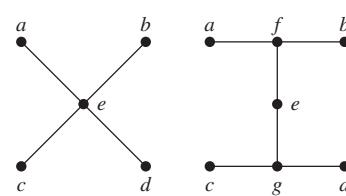
55. How many vertices does a regular graph of degree four with 10 edges have?

In Exercises 56–58 find the union of the given pair of simple graphs. (Assume edges with the same endpoints are the same.)

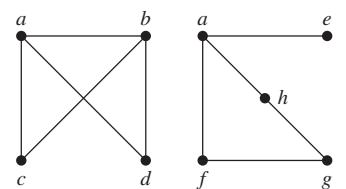
- 56.



- 57.



- 58.



59. The **complementary graph**  $\overline{G}$  of a simple graph  $G$  has the same vertices as  $G$ . Two vertices are adjacent in  $\overline{G}$  if and only if they are not adjacent in  $G$ . Describe each of these graphs.

- a)  $\overline{K}_n$       b)  $\overline{K}_{m,n}$       c)  $\overline{C}_n$       d)  $\overline{Q}_n$

60. If  $G$  is a simple graph with 15 edges and  $\overline{G}$  has 13 edges, how many vertices does  $G$  have?

- 61.** If the simple graph  $G$  has  $v$  vertices and  $e$  edges, how many edges does  $\overline{G}$  have?
- 62.** If the degree sequence of the simple graph  $G$  is  $4, 3, 3, 2, 2$ , what is the degree sequence of  $\overline{G}$ ?
- 63.** If the degree sequence of the simple graph  $G$  is  $d_1, d_2, \dots, d_n$ , what is the degree sequence of  $\overline{G}$ ?
- \*64.** Show that if  $G$  is a bipartite simple graph with  $v$  vertices and  $e$  edges, then  $e \leq v^2/4$ .
- 65.** Show that if  $G$  is a simple graph with  $n$  vertices, then the union of  $G$  and  $\overline{G}$  is  $K_n$ .

- \*66.** Describe an algorithm to decide whether a graph is bipartite based on the fact that a graph is bipartite if and only if it is possible to color its vertices two different colors so that no two vertices of the same color are adjacent.

The **converse** of a directed graph  $G = (V, E)$ , denoted by  $G^{conv}$ , is the directed graph  $(V, F)$ , where the set  $F$  of edges of  $G^{conv}$  is obtained by reversing the direction of each edge in  $E$ .

- 67.** Draw the converse of each of the graphs in Exercises 7–9 in Section 10.1.

- 68.** Show that  $(G^{conv})^{conv} = G$  whenever  $G$  is a directed graph.
- 69.** Show that the graph  $G$  is its own converse if and only if the relation associated with  $G$  (see Section 9.3) is symmetric.
- 70.** Show that if a bipartite graph  $G = (V, E)$  is  $n$ -regular for some positive integer  $n$  (see the preamble to Exercise 53) and  $(V_1, V_2)$  is a bipartition of  $V$ , then  $|V_1| = |V_2|$ . That is, show that the two sets in a bipartition of the vertex set of an  $n$ -regular graph must contain the same number of vertices.
- 71.** Draw the mesh network for interconnecting nine parallel processors.
- 72.** In a variant of a mesh network for interconnecting  $n = m^2$  processors, processor  $P(i, j)$  is connected to the four processors  $P((i \pm 1) \bmod m, j)$  and  $P(i, (j \pm 1) \bmod m)$ , so that connections wrap around the edges of the mesh. Draw this variant of the mesh network for 16 processors.
- 73.** Show that every pair of processors in a mesh network of  $n = m^2$  processors can communicate using  $O(\sqrt{n}) = O(m)$  hops between directly connected processors.

## 10.3 Representing Graphs and Graph Isomorphism

### Introduction

There are many useful ways to represent graphs. As we will see throughout this chapter, in working with a graph it is helpful to be able to choose its most convenient representation. In this section we will show how to represent graphs in several different ways.

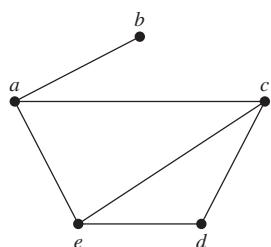
Sometimes, two graphs have exactly the same form, in the sense that there is a one-to-one correspondence between their vertex sets that preserves edges. In such a case, we say that the two graphs are **isomorphic**. Determining whether two graphs are isomorphic is an important problem of graph theory that we will study in this section.

### Representing Graphs

One way to represent a graph without multiple edges is to list all the edges of this graph. Another way to represent a graph with no multiple edges is to use **adjacency lists**, which specify the vertices that are adjacent to each vertex of the graph.

**EXAMPLE 1** Use adjacency lists to describe the simple graph given in Figure 1.

**Solution:** Table 1 lists those vertices adjacent to each of the vertices of the graph. ◀



**FIGURE 1** A Simple Graph.

**TABLE 1** An Adjacency List for a Simple Graph.

| Vertex | Adjacent Vertices |
|--------|-------------------|
| a      | b, c, e           |
| b      | a                 |
| c      | a, d, e           |
| d      | c, e              |
| e      | a, c, d           |

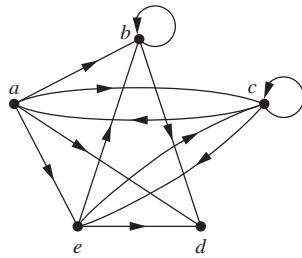


FIGURE 2 A Directed Graph.

**TABLE 2** An Adjacency List for a Directed Graph.

| Initial Vertex | Terminal Vertices |
|----------------|-------------------|
| a              | b, c, d, e        |
| b              | b, d              |
| c              | a, c, e           |
| d              |                   |
| e              | b, c, d           |

**EXAMPLE 2**

Represent the directed graph shown in Figure 2 by listing all the vertices that are the terminal vertices of edges starting at each vertex of the graph.

*Solution:* Table 2 represents the directed graph shown in Figure 2. 

### Adjacency Matrices

Carrying out graph algorithms using the representation of graphs by lists of edges, or by adjacency lists, can be cumbersome if there are many edges in the graph. To simplify computation, graphs can be represented using matrices. Two types of matrices commonly used to represent graphs will be presented here. One is based on the adjacency of vertices, and the other is based on incidence of vertices and edges.

Suppose that  $G = (V, E)$  is a simple graph where  $|V| = n$ . Suppose that the vertices of  $G$  are listed arbitrarily as  $v_1, v_2, \dots, v_n$ . The **adjacency matrix**  $A$  (or  $A_G$ ) of  $G$ , with respect to this listing of the vertices, is the  $n \times n$  zero-one matrix with 1 as its  $(i, j)$ th entry when  $v_i$  and  $v_j$  are adjacent, and 0 as its  $(i, j)$ th entry when they are not adjacent. In other words, if its adjacency matrix is  $A = [a_{ij}]$ , then

$$a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \text{ is an edge of } G, \\ 0 & \text{otherwise.} \end{cases}$$

**EXAMPLE 3**

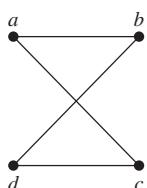
Use an adjacency matrix to represent the graph shown in Figure 3.

*Solution:* We order the vertices as  $a, b, c, d$ . The matrix representing this graph is

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}.$$

**EXAMPLE 4**

Draw a graph with the adjacency matrix



$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

with respect to the ordering of vertices  $a, b, c, d$ .

**FIGURE 4**  
A Graph with the  
Given Adjacency  
Matrix.

*Solution:* A graph with this adjacency matrix is shown in Figure 4. 

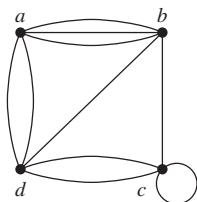
Note that an adjacency matrix of a graph is based on the ordering chosen for the vertices. Hence, there may be as many as  $n!$  different adjacency matrices for a graph with  $n$  vertices, because there are  $n!$  different orderings of  $n$  vertices.

The adjacency matrix of a simple graph is symmetric, that is,  $a_{ij} = a_{ji}$ , because both of these entries are 1 when  $v_i$  and  $v_j$  are adjacent, and both are 0 otherwise. Furthermore, because a simple graph has no loops, each entry  $a_{ii}$ ,  $i = 1, 2, 3, \dots, n$ , is 0.

Adjacency matrices can also be used to represent undirected graphs with loops and with multiple edges. A loop at the vertex  $v_i$  is represented by a 1 at the  $(i, i)$ th position of the adjacency matrix. When multiple edges connecting the same pair of vertices  $v_i$  and  $v_j$ , or multiple loops at the same vertex, are present, the adjacency matrix is no longer a zero–one matrix, because the  $(i, j)$ th entry of this matrix equals the number of edges that are associated to  $\{v_i, v_j\}$ . All undirected graphs, including multigraphs and pseudographs, have symmetric adjacency matrices.

### EXAMPLE 5

Use an adjacency matrix to represent the pseudograph shown in Figure 5.



**Solution:** The adjacency matrix using the ordering of vertices  $a, b, c, d$  is

$$\begin{bmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 2 & 0 \end{bmatrix}.$$

**FIGURE 5**  
**A Pseudograph.**

We used zero–one matrices in Chapter 9 to represent directed graphs. The matrix for a directed graph  $G = (V, E)$  has a 1 in its  $(i, j)$ th position if there is an edge from  $v_i$  to  $v_j$ , where  $v_1, v_2, \dots, v_n$  is an arbitrary listing of the vertices of the directed graph. In other words, if  $\mathbf{A} = [a_{ij}]$  is the adjacency matrix for the directed graph with respect to this listing of the vertices, then

$$a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \text{ is an edge of } G, \\ 0 & \text{otherwise.} \end{cases}$$

The adjacency matrix for a directed graph does not have to be symmetric, because there may not be an edge from  $v_j$  to  $v_i$  when there is an edge from  $v_i$  to  $v_j$ .

Adjacency matrices can also be used to represent directed multigraphs. Again, such matrices are not zero–one matrices when there are multiple edges in the same direction connecting two vertices. In the adjacency matrix for a directed multigraph,  $a_{ij}$  equals the number of edges that are associated to  $(v_i, v_j)$ .

**TRADE-OFFS BETWEEN ADJACENCY LISTS AND ADJACENCY MATRICES** When a simple graph contains relatively few edges, that is, when it is **sparse**, it is usually preferable to use adjacency lists rather than an adjacency matrix to represent the graph. For example, if each vertex has degree not exceeding  $c$ , where  $c$  is a constant much smaller than  $n$ , then each adjacency list contains  $c$  or fewer vertices. Hence, there are no more than  $cn$  items in all these adjacency lists. On the other hand, the adjacency matrix for the graph has  $n^2$  entries. Note, however, that the adjacency matrix of a sparse graph is a **sparse matrix**, that is, a matrix with few nonzero entries, and there are special techniques for representing, and computing with, sparse matrices.

Now suppose that a simple graph is **dense**, that is, suppose that it contains many edges, such as a graph that contains more than half of all possible edges. In this case, using an adjacency matrix to represent the graph is usually preferable over using adjacency lists. To see why, we compare the complexity of determining whether the possible edge  $\{v_i, v_j\}$  is present. Using an adjacency matrix, we can determine whether this edge is present by examining the  $(i, j)$ th entry

in the matrix. This entry is 1 if the graph contains this edge and is 0 otherwise. Consequently, we need make only one comparison, namely, comparing this entry with 0, to determine whether this edge is present. On the other hand, when we use adjacency lists to represent the graph, we need to search the list of vertices adjacent to either  $v_i$  or  $v_j$  to determine whether this edge is present. This can require  $\Theta(|V|)$  comparisons when many edges are present.

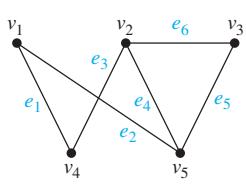
## Incidence Matrices

Another common way to represent graphs is to use **incidence matrices**. Let  $G = (V, E)$  be an undirected graph. Suppose that  $v_1, v_2, \dots, v_n$  are the vertices and  $e_1, e_2, \dots, e_m$  are the edges of  $G$ . Then the incidence matrix with respect to this ordering of  $V$  and  $E$  is the  $n \times m$  matrix  $\mathbf{M} = [m_{ij}]$ , where

$$m_{ij} = \begin{cases} 1 & \text{when edge } e_j \text{ is incident with } v_i, \\ 0 & \text{otherwise.} \end{cases}$$

**EXAMPLE 6** Represent the graph shown in Figure 6 with an incidence matrix.

*Solution:* The incidence matrix is

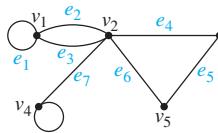


$$\begin{array}{c} e_1 \quad e_2 \quad e_3 \quad e_4 \quad e_5 \quad e_6 \\ \begin{matrix} v_1 & \left[ \begin{array}{cccccc} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{array} \right] \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} \end{array}$$

**FIGURE 6** An Undirected Graph.

Incidence matrices can also be used to represent multiple edges and loops. Multiple edges are represented in the incidence matrix using columns with identical entries, because these edges are incident with the same pair of vertices. Loops are represented using a column with exactly one entry equal to 1, corresponding to the vertex that is incident with this loop.

**EXAMPLE 7** Represent the pseudograph shown in Figure 7 using an incidence matrix.



*Solution:* The incidence matrix for this graph is

$$\begin{array}{c} e_1 \quad e_2 \quad e_3 \quad e_4 \quad e_5 \quad e_6 \quad e_7 \quad e_8 \\ \begin{matrix} v_1 & \left[ \begin{array}{ccccccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{array} \right] \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{matrix} \end{array}$$

**FIGURE 7** A Pseudograph.

## Isomorphism of Graphs

We often need to know whether it is possible to draw two graphs in the same way. That is, do the graphs have the same structure when we ignore the identities of their vertices? For instance, in chemistry, graphs are used to model chemical compounds (in a way we will describe later). Different compounds can have the same molecular formula but can differ in structure. Such compounds can be represented by graphs that cannot be drawn in the same way. The graphs representing previously known compounds can be used to determine whether a supposedly new compound has been studied before.

There is a useful terminology for graphs with the same structure.

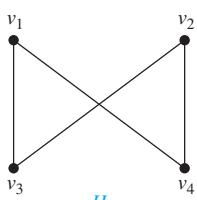
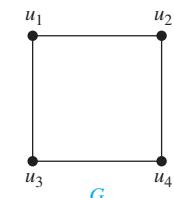
### DEFINITION 1

The simple graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  are *isomorphic* if there exists a one-to-one and onto function  $f$  from  $V_1$  to  $V_2$  with the property that  $a$  and  $b$  are adjacent in  $G_1$  if and only if  $f(a)$  and  $f(b)$  are adjacent in  $G_2$ , for all  $a$  and  $b$  in  $V_1$ . Such a function  $f$  is called an *isomorphism*.<sup>\*</sup> Two simple graphs that are not isomorphic are called *nonisomorphic*.

In other words, when two simple graphs are isomorphic, there is a one-to-one correspondence between vertices of the two graphs that preserves the adjacency relationship. Isomorphism of simple graphs is an equivalence relation. (We leave the verification of this as Exercise 45.)

### EXAMPLE 8

Show that the graphs  $G = (V, E)$  and  $H = (W, F)$ , displayed in Figure 8, are isomorphic.



**FIGURE 8** The Graphs  $G$  and  $H$ .

**Solution:** The function  $f$  with  $f(u_1) = v_1$ ,  $f(u_2) = v_4$ ,  $f(u_3) = v_3$ , and  $f(u_4) = v_2$  is a one-to-one correspondence between  $V$  and  $W$ . To see that this correspondence preserves adjacency, note that adjacent vertices in  $G$  are  $u_1$  and  $u_2$ ,  $u_1$  and  $u_3$ ,  $u_2$  and  $u_4$ , and  $u_3$  and  $u_4$ , and each of the pairs  $f(u_1) = v_1$  and  $f(u_2) = v_4$ ,  $f(u_1) = v_1$  and  $f(u_3) = v_3$ ,  $f(u_2) = v_4$  and  $f(u_4) = v_2$ , and  $f(u_3) = v_3$  and  $f(u_4) = v_2$  consists of two adjacent vertices in  $H$ . 

### Determining whether Two Simple Graphs are Isomorphic

It is often difficult to determine whether two simple graphs are isomorphic. There are  $n!$  possible one-to-one correspondences between the vertex sets of two simple graphs with  $n$  vertices. Testing each such correspondence to see whether it preserves adjacency and nonadjacency is impractical if  $n$  is at all large.

Sometimes it is not hard to show that two graphs are not isomorphic. In particular, we can show that two graphs are not isomorphic if we can find a property only one of the two graphs has, but that is preserved by isomorphism. A property preserved by isomorphism of graphs is called a **graph invariant**. For instance, isomorphic simple graphs must have the same number of vertices, because there is a one-to-one correspondence between the sets of vertices of the graphs.

Isomorphic simple graphs also must have the same number of edges, because the one-to-one correspondence between vertices establishes a one-to-one correspondence between edges. In addition, the degrees of the vertices in isomorphic simple graphs must be the same. That is, a vertex  $v$  of degree  $d$  in  $G$  must correspond to a vertex  $f(v)$  of degree  $d$  in  $H$ , because a vertex  $w$  in  $G$  is adjacent to  $v$  if and only if  $f(v)$  and  $f(w)$  are adjacent in  $H$ .

### EXAMPLE 9

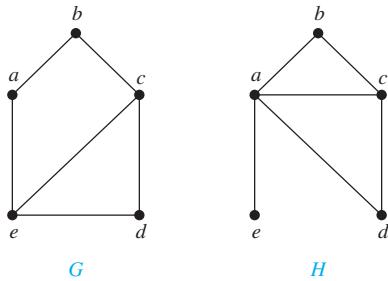
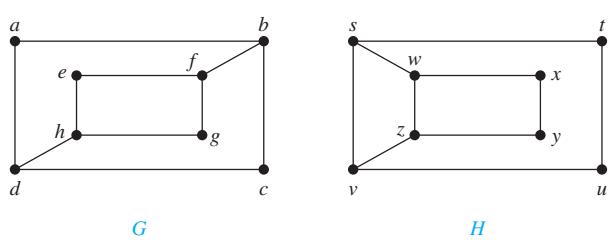
Show that the graphs displayed in Figure 9 are not isomorphic.



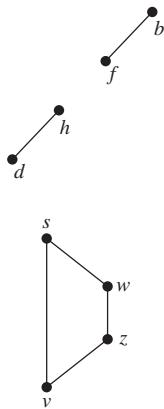
**Solution:** Both  $G$  and  $H$  have five vertices and six edges. However,  $H$  has a vertex of degree one, namely,  $e$ , whereas  $G$  has no vertices of degree one. It follows that  $G$  and  $H$  are not isomorphic. 

The number of vertices, the number of edges, and the number of vertices of each degree are all invariants under isomorphism. If any of these quantities differ in two simple graphs, these graphs cannot be isomorphic. However, when these invariants are the same, it does not necessarily mean that the two graphs are isomorphic. There are no useful sets of invariants currently known that can be used to determine whether simple graphs are isomorphic.

\*The word *isomorphism* comes from the Greek roots *isos* for “equal” and *morphe* for “form.”

FIGURE 9 The Graphs  $G$  and  $H$ .FIGURE 10 The Graphs  $G$  and  $H$ .**EXAMPLE 10**

Determine whether the graphs shown in Figure 10 are isomorphic.



**FIGURE 11** The Subgraphs of  $G$  and  $H$  Made Up of Vertices of Degree Three and the Edges Connecting Them.

**Solution:** The graphs  $G$  and  $H$  both have eight vertices and 10 edges. They also both have four vertices of degree two and four of degree three. Because these invariants all agree, it is still conceivable that these graphs are isomorphic.

However,  $G$  and  $H$  are not isomorphic. To see this, note that because  $\deg(a) = 2$  in  $G$ ,  $a$  must correspond to either  $t$ ,  $u$ ,  $x$ , or  $y$  in  $H$ , because these are the vertices of degree two in  $H$ . However, each of these four vertices in  $H$  is adjacent to another vertex of degree two in  $H$ , which is not true for  $a$  in  $G$ .

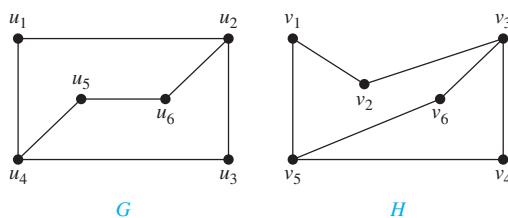
Another way to see that  $G$  and  $H$  are not isomorphic is to note that the subgraphs of  $G$  and  $H$  made up of vertices of degree three and the edges connecting them must be isomorphic if these two graphs are isomorphic (the reader should verify this). However, these subgraphs, shown in Figure 11, are not isomorphic.  $\blacktriangleleft$

To show that a function  $f$  from the vertex set of a graph  $G$  to the vertex set of a graph  $H$  is an isomorphism, we need to show that  $f$  preserves the presence and absence of edges. One helpful way to do this is to use adjacency matrices. In particular, to show that  $f$  is an isomorphism, we can show that the adjacency matrix of  $G$  is the same as the adjacency matrix of  $H$ , when rows and columns are labeled to correspond to the images under  $f$  of the vertices in  $G$  that are the labels of these rows and columns in the adjacency matrix of  $G$ . We illustrate how this is done in Example 11.

**EXAMPLE 11**

Determine whether the graphs  $G$  and  $H$  displayed in Figure 12 are isomorphic.

**Solution:** Both  $G$  and  $H$  have six vertices and seven edges. Both have four vertices of degree two and two vertices of degree three. It is also easy to see that the subgraphs of  $G$  and  $H$  consisting of all vertices of degree two and the edges connecting them are isomorphic (as the reader should verify). Because  $G$  and  $H$  agree with respect to these invariants, it is reasonable to try to find an isomorphism  $f$ .

FIGURE 12 Graphs  $G$  and  $H$ .

We now will define a function  $f$  and then determine whether it is an isomorphism. Because  $\deg(u_1) = 2$  and because  $u_1$  is not adjacent to any other vertex of degree two, the image of  $u_1$  must be either  $v_4$  or  $v_6$ , the only vertices of degree two in  $H$  not adjacent to a vertex of degree two. We arbitrarily set  $f(u_1) = v_6$ . [If we found that this choice did not lead to isomorphism, we would then try  $f(u_1) = v_4$ .] Because  $u_2$  is adjacent to  $u_1$ , the possible images of  $u_2$  are  $v_3$  and  $v_5$ . We arbitrarily set  $f(u_2) = v_3$ . Continuing in this way, using adjacency of vertices and degrees as a guide, we set  $f(u_3) = v_4$ ,  $f(u_4) = v_5$ ,  $f(u_5) = v_1$ , and  $f(u_6) = v_2$ . We now have a one-to-one correspondence between the vertex set of  $G$  and the vertex set of  $H$ , namely,  $f(u_1) = v_6$ ,  $f(u_2) = v_3$ ,  $f(u_3) = v_4$ ,  $f(u_4) = v_5$ ,  $f(u_5) = v_1$ ,  $f(u_6) = v_2$ . To see whether  $f$  preserves edges, we examine the adjacency matrix of  $G$ ,

$$\mathbf{A}_G = \begin{matrix} & u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \\ u_1 & \left[ \begin{array}{cccccc} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{array} \right] \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{matrix},$$

and the adjacency matrix of  $H$  with the rows and columns labeled by the images of the corresponding vertices in  $G$ ,

$$\mathbf{A}_H = \begin{matrix} & v_6 & v_3 & v_4 & v_5 & v_1 & v_2 \\ v_6 & \left[ \begin{array}{cccccc} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{array} \right] \\ v_3 \\ v_4 \\ v_5 \\ v_1 \\ v_2 \end{matrix}.$$

Because  $\mathbf{A}_G = \mathbf{A}_H$ , it follows that  $f$  preserves edges. We conclude that  $f$  is an isomorphism, so  $G$  and  $H$  are isomorphic. Note that if  $f$  turned out not to be an isomorphism, we would *not* have established that  $G$  and  $H$  are not isomorphic, because another correspondence of the vertices in  $G$  and  $H$  may be an isomorphism. 



**ALGORITHMS FOR GRAPH ISOMORPHISM** The best algorithms known for determining whether two graphs are isomorphic have exponential worst-case time complexity (in the number of vertices of the graphs). However, linear average-case time complexity algorithms are known that solve this problem, and there is some hope, but also skepticism, that an algorithm with polynomial worst-case time complexity for determining whether two graphs are isomorphic can be found. The best practical general purpose software for isomorphism testing, called NAUTY, can be used to determine whether two graphs with as many as 100 vertices are isomorphic in less than a second on a modern PC. NAUTY software can be downloaded over the Internet and experimented with. Practical algorithms for determining whether two graphs are isomorphic exist for graphs that are restricted in various ways, such as when the maximum degree of vertices is small. The problem of determining whether any two graphs are isomorphic is of special interest because it is one of only a few NP problems (see Exercise 72) not known to be either tractable or NP-complete (see Section 3.3).

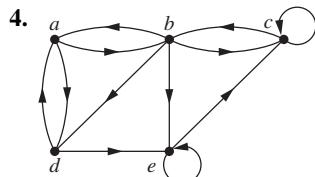
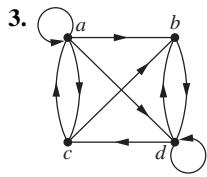
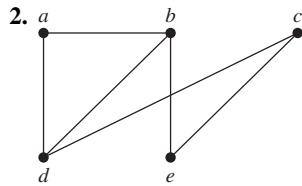
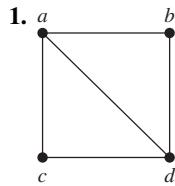
**APPLICATIONS OF GRAPH ISOMORPHISMS** Graph isomorphisms, and functions that are almost graph isomorphisms, arise in applications of graph theory to chemistry and to the design of electronic circuits, and other areas including bioinformatics and computer vision.

Chemists use multigraphs, known as molecular graphs, to model chemical compounds. In these graphs, vertices represent atoms and edges represent chemical bonds between these atoms. Two structural isomers, molecules with identical molecular formulas but with atoms bonded differently, have nonisomorphic molecular graphs. When a potentially new chemical compound is synthesized, a database of molecular graphs is checked to see whether the molecular graph of the compound is the same as one already known.

Electronic circuits are modeled using graphs in which vertices represent components and edges represent connections between them. Modern integrated circuits, known as chips, are miniaturized electronic circuits, often with millions of transistors and connections between them. Because of the complexity of modern chips, automation tools are used to design them. Graph isomorphism is the basis for the verification that a particular layout of a circuit produced by an automated tool corresponds to the original schematic of the design. Graph isomorphism can also be used to determine whether a chip from one vendor includes intellectual property from a different vendor. This can be done by looking for large isomorphic subgraphs in the graphs modeling these chips.

## Exercises

In Exercises 1–4 use an adjacency list to represent the given graph.



5. Represent the graph in Exercise 1 with an adjacency matrix.
6. Represent the graph in Exercise 2 with an adjacency matrix.
7. Represent the graph in Exercise 3 with an adjacency matrix.
8. Represent the graph in Exercise 4 with an adjacency matrix.
9. Represent each of these graphs with an adjacency matrix.

- a)  $K_4$       b)  $K_{1,4}$       c)  $K_{2,3}$   
 d)  $C_4$       e)  $W_4$       f)  $Q_3$

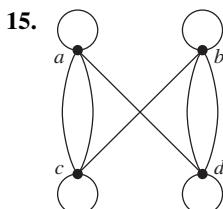
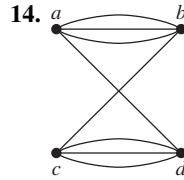
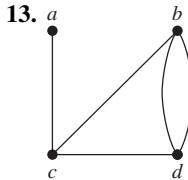
In Exercises 10–12 draw a graph with the given adjacency matrix.

10.  $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

11.  $\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$

12.  $\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 \end{bmatrix}$

In Exercises 13–15 represent the given graph using an adjacency matrix.



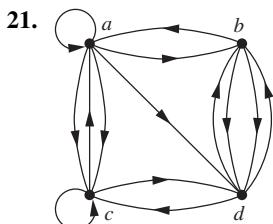
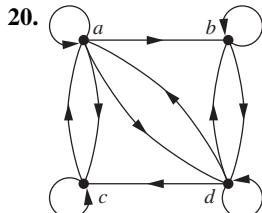
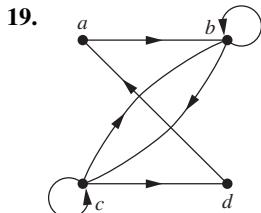
In Exercises 16–18 draw an undirected graph represented by the given adjacency matrix.

16.  $\begin{bmatrix} 1 & 3 & 2 \\ 3 & 0 & 4 \\ 2 & 4 & 0 \end{bmatrix}$

17.  $\begin{bmatrix} 1 & 2 & 0 & 1 \\ 2 & 0 & 3 & 0 \\ 0 & 3 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$

18.  $\begin{bmatrix} 0 & 1 & 3 & 0 & 4 \\ 1 & 2 & 1 & 3 & 0 \\ 3 & 1 & 1 & 0 & 1 \\ 0 & 3 & 0 & 0 & 2 \\ 4 & 0 & 1 & 2 & 3 \end{bmatrix}$

In Exercises 19–21 find the adjacency matrix of the given directed multigraph with respect to the vertices listed in alphabetical order.



In Exercises 22–24 draw the graph represented by the given adjacency matrix.

22.  $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

23.  $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & 0 \\ 0 & 2 & 2 \end{bmatrix}$

24.  $\begin{bmatrix} 0 & 2 & 3 & 0 \\ 1 & 2 & 2 & 1 \\ 2 & 1 & 1 & 0 \\ 1 & 0 & 0 & 2 \end{bmatrix}$

25. Is every zero–one square matrix that is symmetric and has zeros on the diagonal the adjacency matrix of a simple graph?

26. Use an incidence matrix to represent the graphs in Exercises 1 and 2.

27. Use an incidence matrix to represent the graphs in Exercises 13–15.

\*28. What is the sum of the entries in a row of the adjacency matrix for an undirected graph? For a directed graph?

\*29. What is the sum of the entries in a column of the adjacency matrix for an undirected graph? For a directed graph?

30. What is the sum of the entries in a row of the incidence matrix for an undirected graph?

31. What is the sum of the entries in a column of the incidence matrix for an undirected graph?

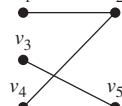
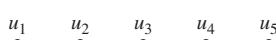
\*32. Find an adjacency matrix for each of these graphs.

- a)  $K_n$     b)  $C_n$     c)  $W_n$     d)  $K_{m,n}$     e)  $Q_n$

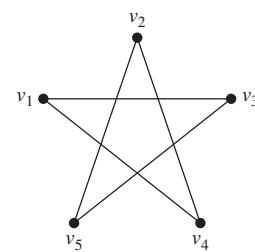
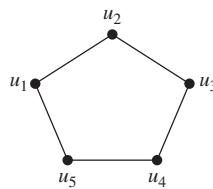
\*33. Find incidence matrices for the graphs in parts (a)–(d) of Exercise 32.

In Exercises 34–44 determine whether the given pair of graphs is isomorphic. Exhibit an isomorphism or provide a rigorous argument that none exists.

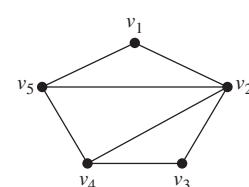
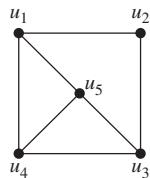
34.



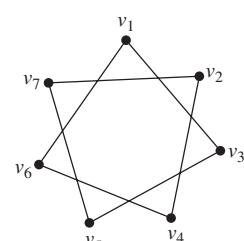
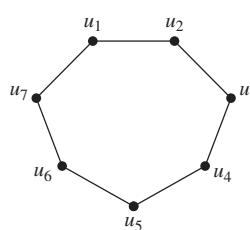
35.



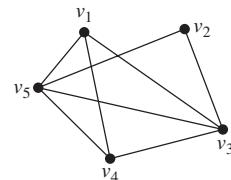
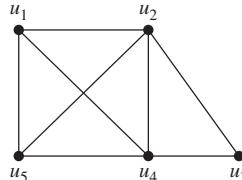
36.



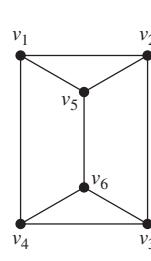
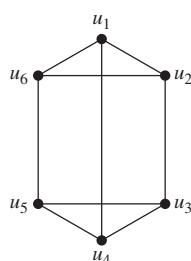
37.



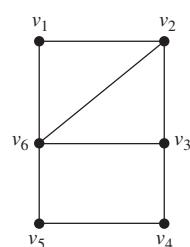
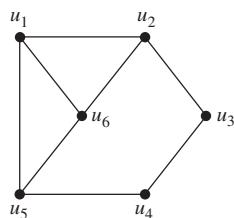
38.



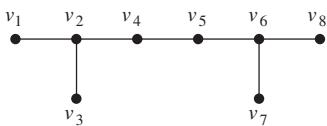
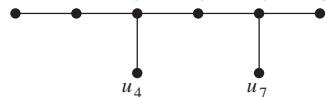
39.



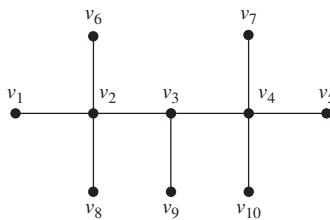
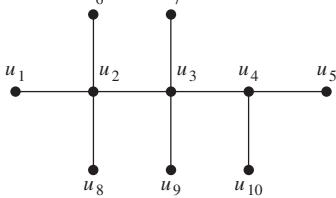
40.



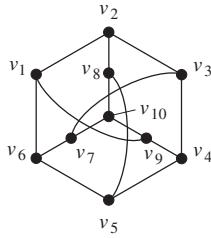
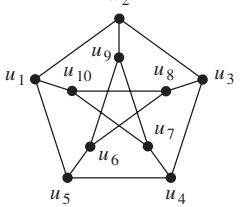
41.



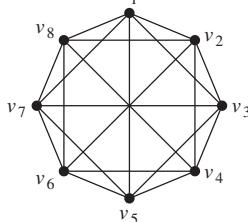
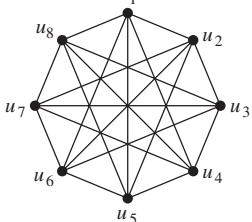
42.



43.



44.



45. Show that isomorphism of simple graphs is an equivalence relation.
46. Suppose that  $G$  and  $H$  are isomorphic simple graphs. Show that their complementary graphs  $\overline{G}$  and  $\overline{H}$  are also isomorphic.
47. Describe the row and column of an adjacency matrix of a graph corresponding to an isolated vertex.
48. Describe the row of an incidence matrix of a graph corresponding to an isolated vertex.
49. Show that the vertices of a bipartite graph with two or more vertices can be ordered so that its adjacency matrix

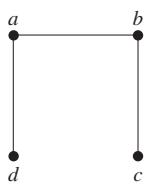
has the form

$$\begin{bmatrix} \mathbf{0} & \mathbf{A} \\ \mathbf{B} & \mathbf{0} \end{bmatrix},$$

where the four entries shown are rectangular blocks.

A simple graph  $G$  is called **self-complementary** if  $G$  and  $\overline{G}$  are isomorphic.

50. Show that this graph is self-complementary.



51. Find a self-complementary simple graph with five vertices.

\*52. Show that if  $G$  is a self-complementary simple graph with  $v$  vertices, then  $v \equiv 0$  or  $1 \pmod{4}$ .53. For which integers  $n$  is  $C_n$  self-complementary?54. How many nonisomorphic simple graphs are there with  $n$  vertices, when  $n$  is

- a) 2?      b) 3?      c) 4?

55. How many nonisomorphic simple graphs are there with five vertices and three edges?

56. How many nonisomorphic simple graphs are there with six vertices and four edges?

57. Are the simple graphs with the following adjacency matrices isomorphic?

a)  $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$

b)  $\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$

c)  $\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$

58. Determine whether the graphs without loops with these incidence matrices are isomorphic.

a)  $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$

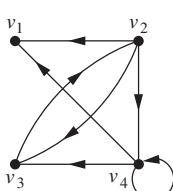
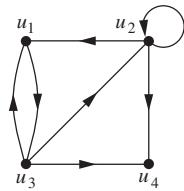
b)  $\begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$

59. Extend the definition of isomorphism of simple graphs to undirected graphs containing loops and multiple edges.

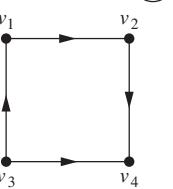
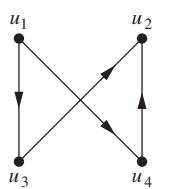
60. Define isomorphism of directed graphs.

In Exercises 61–64 determine whether the given pair of directed graphs are isomorphic. (See Exercise 60.)

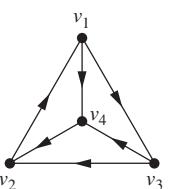
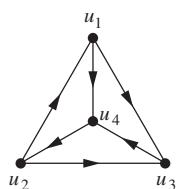
61.



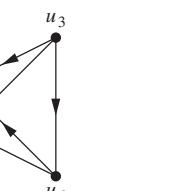
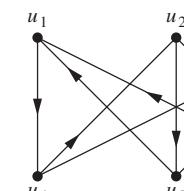
62.



63.



64.



65. Show that if  $G$  and  $H$  are isomorphic directed graphs, then the converses of  $G$  and  $H$  (defined in the preamble of Exercise 67 of Section 10.2) are also isomorphic.

66. Show that the property that a graph is bipartite is an isomorphic invariant.

67. Find a pair of nonisomorphic graphs with the same degree sequence (defined in the preamble to Exercise 36 in Section 10.2) such that one graph is bipartite, but the other graph is not bipartite.

\*68. How many nonisomorphic directed simple graphs are there with  $n$  vertices, when  $n$  is

- a) 2?      b) 3?      c) 4?

\*69. What is the product of the incidence matrix and its transpose for an undirected graph?

\*70. How much storage is needed to represent a simple graph with  $n$  vertices and  $m$  edges using

- a) adjacency lists?  
b) an adjacency matrix?  
c) an incidence matrix?

A **devil's pair** for a purported isomorphism test is a pair of nonisomorphic graphs that the test fails to show that they are not isomorphic.

71. Find a devil's pair for the test that checks the degree sequence (defined in the preamble to Exercise 36 in Section 10.2) in two graphs to make sure they agree.

72. Suppose that the function  $f$  from  $V_1$  to  $V_2$  is an isomorphism of the graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$ . Show that it is possible to verify this fact in time polynomial in terms of the number of vertices of the graph, in terms of the number of comparisons needed.

## 10.4 Connectivity

### Introduction

Many problems can be modeled with paths formed by traveling along the edges of graphs. For instance, the problem of determining whether a message can be sent between two computers using intermediate links can be studied with a graph model. Problems of efficiently planning routes for mail delivery, garbage pickup, diagnostics in computer networks, and so on can be solved using models that involve paths in graphs.

### Paths

Informally, a **path** is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along edges of the graph. As the path travels along its edges, it visits the vertices along this path, that is, the endpoints of these edges.

A formal definition of paths and related terminology is given in Definition 1.

### DEFINITION 1

Let  $n$  be a nonnegative integer and  $G$  an undirected graph. A *path* of length  $n$  from  $u$  to  $v$  in  $G$  is a sequence of  $n$  edges  $e_1, \dots, e_n$  of  $G$  for which there exists a sequence  $x_0 = u, x_1, \dots, x_{n-1}, x_n = v$  of vertices such that  $e_i$  has, for  $i = 1, \dots, n$ , the endpoints  $x_{i-1}$  and  $x_i$ . When the graph is simple, we denote this path by its vertex sequence  $x_0, x_1, \dots, x_n$  (because listing these vertices uniquely determines the path). The path is a *circuit* if it begins and ends at the same vertex, that is, if  $u = v$ , and has length greater than zero. The path or circuit is said to *pass through* the vertices  $x_1, x_2, \dots, x_{n-1}$  or *traverse* the edges  $e_1, e_2, \dots, e_n$ . A path or circuit is *simple* if it does not contain the same edge more than once.

When it is not necessary to distinguish between multiple edges, we will denote a path  $e_1, e_2, \dots, e_n$ , where  $e_i$  is associated with  $\{x_{i-1}, x_i\}$  for  $i = 1, 2, \dots, n$  by its vertex sequence  $x_0, x_1, \dots, x_n$ . This notation identifies a path only as far as which vertices it passes through. Consequently, it does not specify a unique path when there is more than one path that passes through this sequence of vertices, which will happen if and only if there are multiple edges between some successive vertices in the list. Note that a path of length zero consists of a single vertex.

**Remark:** There is considerable variation of terminology concerning the concepts defined in Definition 1. For instance, in some books, the term **walk** is used instead of *path*, where a walk is defined to be an alternating sequence of vertices and edges of a graph,  $v_0, e_1, v_1, e_2, \dots, v_{n-1}, e_n, v_n$ , where  $v_{i-1}$  and  $v_i$  are the endpoints of  $e_i$  for  $i = 1, 2, \dots, n$ . When this terminology is used, **closed walk** is used instead of *circuit* to indicate a walk that begins and ends at the same vertex, and **trail** is used to denote a walk that has no repeated edge (replacing the term *simple path*). When this terminology is used, the terminology **path** is often used for a trail with no repeated vertices, conflicting with the terminology in Definition 1. Because of this variation in terminology, you will need to make sure which set of definitions are used in a particular book or article when you read about traversing edges of a graph. The text [GrYe06] is a good reference for the alternative terminology described in this remark.

### EXAMPLE 1

In the simple graph shown in Figure 1,  $a, d, c, f, e$  is a simple path of length 4, because  $\{a, d\}$ ,  $\{d, c\}$ ,  $\{c, f\}$ , and  $\{f, e\}$  are all edges. However,  $d, e, c, a$  is not a path, because  $\{e, c\}$  is not an edge. Note that  $b, c, f, e, b$  is a circuit of length 4 because  $\{b, c\}$ ,  $\{c, f\}$ ,  $\{f, e\}$ , and  $\{e, b\}$  are edges, and this path begins and ends at  $b$ . The path  $a, b, e, d, a, b$ , which is of length 5, is not simple because it contains the edge  $\{a, b\}$  twice. 

Paths and circuits in directed graphs were introduced in Chapter 9. We now provide more general definitions.

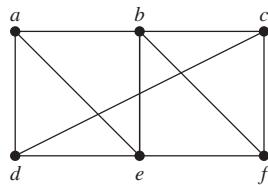


FIGURE 1 A Simple Graph.

**DEFINITION 2**

Let  $n$  be a nonnegative integer and  $G$  a directed graph. A *path* of length  $n$  from  $u$  to  $v$  in  $G$  is a sequence of edges  $e_1, e_2, \dots, e_n$  of  $G$  such that  $e_1$  is associated with  $(x_0, x_1)$ ,  $e_2$  is associated with  $(x_1, x_2)$ , and so on, with  $e_n$  associated with  $(x_{n-1}, x_n)$ , where  $x_0 = u$  and  $x_n = v$ . When there are no multiple edges in the directed graph, this path is denoted by its vertex sequence  $x_0, x_1, x_2, \dots, x_n$ . A path of length greater than zero that begins and ends at the same vertex is called a *circuit* or *cycle*. A path or circuit is called *simple* if it does not contain the same edge more than once.

**Remark:** Terminology other than that given in Definition 2 is often used for the concepts defined there. In particular, the alternative terminology that uses *walk*, *closed walk*, *trail*, and *path* (described in the remarks following Definition 1) may be used for directed graphs. See [GrYe05] for details.

Note that the terminal vertex of an edge in a path is the initial vertex of the next edge in the path. When it is not necessary to distinguish between multiple edges, we will denote a path  $e_1, e_2, \dots, e_n$ , where  $e_i$  is associated with  $(x_{i-1}, x_i)$  for  $i = 1, 2, \dots, n$ , by its vertex sequence  $x_0, x_1, \dots, x_n$ . The notation identifies a path only as far as which the vertices it passes through. There may be more than one path that passes through this sequence of vertices, which will happen if and only if there are multiple edges between two successive vertices in the list.

Paths represent useful information in many graph models, as Examples 2–4 demonstrate.

**EXAMPLE 2**

**Paths in Acquaintanceship Graphs** In an acquaintanceship graph there is a path between two people if there is a chain of people linking these people, where two people adjacent in the chain know one another. For example, in Figure 6 in Section 10.1, there is a chain of six people linking Kamini and Ching. Many social scientists have conjectured that almost every pair of people in the world are linked by a small chain of people, perhaps containing just five or fewer people. This would mean that almost every pair of vertices in the acquaintanceship graph containing all people in the world is linked by a path of length not exceeding four. The play *Six Degrees of Separation* by John Guare is based on this notion. 

**EXAMPLE 3**

**Paths in Collaboration Graphs** In a collaboration graph, two people  $a$  and  $b$  are connected by a path when there is a sequence of people starting with  $a$  and ending with  $b$  such that the endpoints of each edge in the path are people who have collaborated. We will consider two particular collaboration graphs here. First, in the academic collaboration graph of people who have written papers in mathematics, the **Erdős number** of a person  $m$  (defined in terms of relations in Supplementary Exercise 14 in Chapter 9) is the length of the shortest path between  $m$  and the extremely prolific mathematician Paul Erdős (who died in 1996). That is, the Erdős number of a mathematician is the length of the shortest chain of mathematicians that begins with Paul Erdős and ends with this mathematician, where each adjacent pair of mathematicians have written a joint paper. The number of mathematicians with each Erdős number as of early 2006, according to the Erdős Number Project, is shown in Table 1.

In the Hollywood graph (see Example 3 in Section 10.1) two actors  $a$  and  $b$  are linked when there is a chain of actors linking  $a$  and  $b$ , where every two actors adjacent in the chain have acted in the same movie. In the Hollywood graph, the **Bacon number** of an actor  $c$  is defined to be the length of the shortest path connecting  $c$  and the well-known actor Kevin Bacon. As new movies are made, including new ones with Kevin Bacon, the Bacon number of actors can change. In Table 2 we show the number of actors with each Bacon number as of early 2011 using data from the Oracle of Bacon website. The origins of the Bacon number of an actor dates back to the early 1990s, when Kevin Bacon remarked that he had worked with everyone in Hollywood or someone who worked with them. This lead some people to invent a party

Replace Kevin Bacon by  
your own favorite actor to  
invent a new party game

**TABLE 1** The Number of Mathematicians with a Given Erdős Number (as of early 2006).

| Erdős Number | Number of People |
|--------------|------------------|
| 0            | 1                |
| 1            | 504              |
| 2            | 6,593            |
| 3            | 33,605           |
| 4            | 83,642           |
| 5            | 87,760           |
| 6            | 40,014           |
| 7            | 11,591           |
| 8            | 3,146            |
| 9            | 819              |
| 10           | 244              |
| 11           | 68               |
| 12           | 23               |
| 13           | 5                |

**TABLE 2** The Number of Actors with a Given Bacon Number (as of early 2011).

| Bacon Number | Number of People |
|--------------|------------------|
| 0            | 1                |
| 1            | 2,367            |
| 2            | 242,407          |
| 3            | 785,389          |
| 4            | 200,602          |
| 5            | 14,048           |
| 6            | 1,277            |
| 7            | 114              |
| 8            | 16               |

game where participants were challenged to find a sequence of movies leading from each actor named to Kevin Bacon. We can find a number similar to a Bacon number using any actor as the center of the acting universe. 

## Connectedness in Undirected Graphs

When does a computer network have the property that every pair of computers can share information, if messages can be sent through one or more intermediate computers? When a graph is used to represent this computer network, where vertices represent the computers and edges represent the communication links, this question becomes: When is there always a path between two vertices in the graph?

### DEFINITION 3

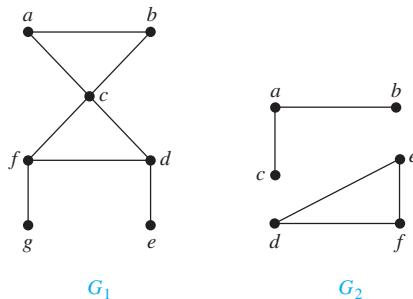
An undirected graph is called *connected* if there is a path between every pair of distinct vertices of the graph. An undirected graph that is not *connected* is called *disconnected*. We say that we *disconnect* a graph when we remove vertices or edges, or both, to produce a disconnected subgraph.

Thus, any two computers in the network can communicate if and only if the graph of this network is connected.

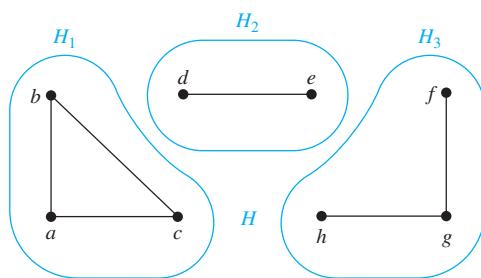
### EXAMPLE 4

The graph  $G_1$  in Figure 2 is connected, because for every pair of distinct vertices there is a path between them (the reader should verify this). However, the graph  $G_2$  in Figure 2 is not connected. For instance, there is no path in  $G_2$  between vertices  $a$  and  $d$ . 

We will need the following theorem in Chapter 11.



**FIGURE 2** The Graphs  $G_1$  and  $G_2$ .



**FIGURE 3** The Graph  $H$  and Its Connected Components  $H_1$ ,  $H_2$ , and  $H_3$ .

### THEOREM 1

There is a simple path between every pair of distinct vertices of a connected undirected graph.

**Proof:** Let  $u$  and  $v$  be two distinct vertices of the connected undirected graph  $G = (V, E)$ . Because  $G$  is connected, there is at least one path between  $u$  and  $v$ . Let  $x_0, x_1, \dots, x_n$ , where  $x_0 = u$  and  $x_n = v$ , be the vertex sequence of a path of least length. This path of least length is simple. To see this, suppose it is not simple. Then  $x_i = x_j$  for some  $i$  and  $j$  with  $0 \leq i < j$ . This means that there is a path from  $u$  to  $v$  of shorter length with vertex sequence  $x_0, x_1, \dots, x_{i-1}, x_j, \dots, x_n$  obtained by deleting the edges corresponding to the vertex sequence  $x_i, \dots, x_{j-1}$ .  $\triangle$



**CONNECTED COMPONENTS** A **connected component** of a graph  $G$  is a connected subgraph of  $G$  that is not a proper subgraph of another connected subgraph of  $G$ . That is, a connected component of a graph  $G$  is a maximal connected subgraph of  $G$ . A graph  $G$  that is not connected has two or more connected components that are disjoint and have  $G$  as their union.

### EXAMPLE 5

What are the connected components of the graph  $H$  shown in Figure 3?

**Solution:** The graph  $H$  is the union of three disjoint connected subgraphs  $H_1$ ,  $H_2$ , and  $H_3$ , shown in Figure 3. These three subgraphs are the connected components of  $H$ .  $\triangle$



### EXAMPLE 6

**Connected Components of Call Graphs** Two vertices  $x$  and  $y$  are in the same component of a telephone call graph (see Example 4 in Section 10.1) when there is a sequence of telephone calls beginning at  $x$  and ending at  $y$ . When a call graph for telephone calls made during a particular day in the AT&T network was analyzed, this graph was found to have 53,767,087 vertices, more than 170 million edges, and more than 3.7 million connected components. Most of these components were small; approximately three-fourths consisted of two vertices representing pairs of telephone numbers that called only each other. This graph has one huge connected component with 44,989,297 vertices comprising more than 80% of the total. Furthermore, every vertex in this component can be linked to any other vertex by a chain of no more than 20 calls.  $\triangle$

## How Connected is a Graph?

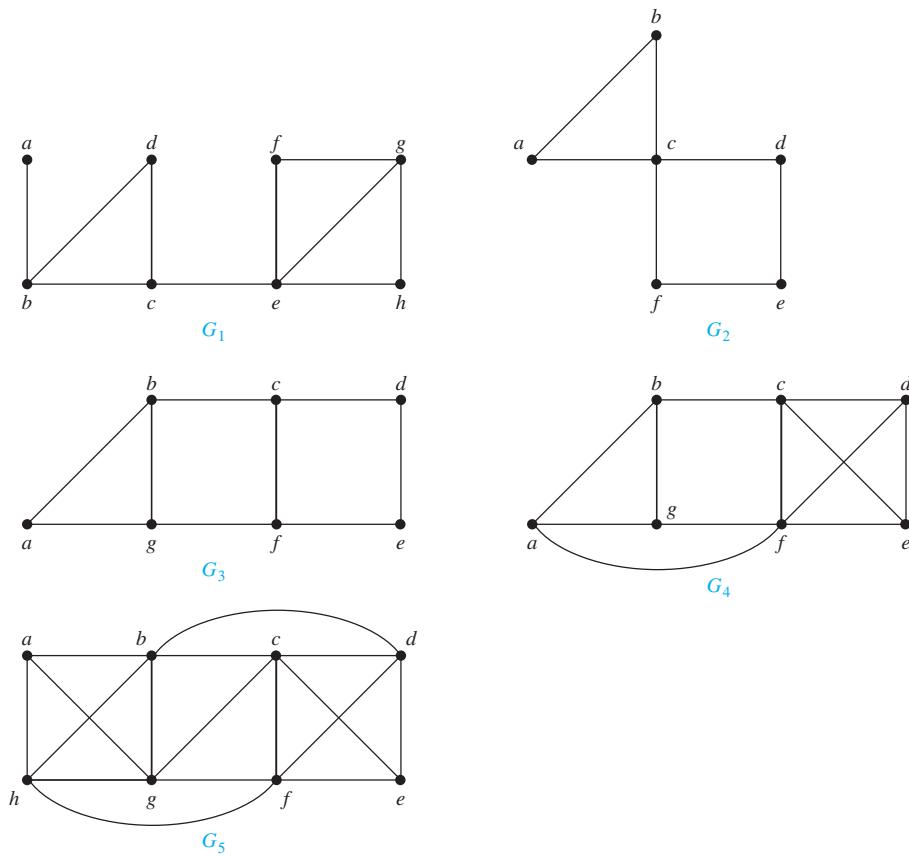
Suppose that a graph represents a computer network. Knowing that this graph is connected tells us that any two computers on the network can communicate. However, we would also like to understand how reliable this network is. For instance, will it still be possible for all computers to communicate after a router or a communications link fails? To answer this and similar questions, we now develop some new concepts.

Sometimes the removal from a graph of a vertex and all incident edges produces a subgraph with more connected components. Such vertices are called **cut vertices** (or **articulation points**). The removal of a cut vertex from a connected graph produces a subgraph that is not connected. Analogously, an edge whose removal produces a graph with more connected components than in the original graph is called a **cut edge** or **bridge**. Note that in a graph representing a computer network, a cut vertex and a cut edge represent an essential router and an essential link that cannot fail for all computers to be able to communicate.

**EXAMPLE 7** Find the cut vertices and cut edges in the graph  $G_1$  shown in Figure 4.

**Solution:** The cut vertices of  $G_1$  are  $b$ ,  $c$ , and  $e$ . The removal of one of these vertices (and its adjacent edges) disconnects the graph. The cut edges are  $\{a, b\}$  and  $\{c, e\}$ . Removing either one of these edges disconnects  $G_1$ .  $\blacktriangleleft$

**VERTEX CONNECTIVITY** Not all graphs have cut vertices. For example, the complete graph  $K_n$ , where  $n \geq 3$ , has no cut vertices. When you remove a vertex from  $K_n$  and all edges incident to it, the resulting subgraph is the complete graph  $K_{n-1}$ , a connected graph. Connected graphs without cut vertices are called **nonseparable graphs**, and can be thought of as more connected than those with a cut vertex. We can extend this notion by defining a more granulated measure of graph connectivity based on the minimum number of vertices that can be removed to disconnect a graph.



**FIGURE 4** Some Connected Graphs

A subset  $V'$  of the vertex set  $V$  of  $G = (V, E)$  is a **vertex cut**, or **separating set**, if  $G - V'$  is disconnected. For instance, in the graph in Figure 1, the set  $\{b, c, e\}$  is a vertex cut with three vertices, as the reader should verify. We leave it to the reader (Exercise 51) to show that every connected graph, except a complete graph, has a vertex cut. We define the **vertex connectivity** of a noncomplete graph  $G$ , denoted by  $\kappa(G)$ , as the minimum number of vertices in a vertex cut.

$\kappa$  is the lowercase Greek letter kappa.

When  $G$  is a complete graph, it has no vertex cuts, because removing any subset of its vertices and all incident edges still leaves a complete graph. Consequently, we cannot define  $\kappa(G)$  as the minimum number of vertices in a vertex cut when  $G$  is complete. Instead, we set  $\kappa(K_n) = n - 1$ , the number of vertices needed to be removed to produce a graph with a single vertex.

Consequently, for every graph  $G$ ,  $\kappa(G)$  is minimum number of vertices that can be removed from  $G$  to either disconnect  $G$  or produce a graph with a single vertex. We have  $0 \leq \kappa(G) \leq n - 1$  if  $G$  has  $n$  vertices,  $\kappa(G) = 0$  if and only if  $G$  is disconnected or  $G = K_1$ , and  $\kappa(G) = n - 1$  if and only if  $G$  is complete [see Exercise 52(a)].

The larger  $\kappa(G)$  is, the more connected we consider  $G$  to be. Disconnected graphs and  $K_1$  have  $\kappa(G) = 0$ , connected graphs with cut vertices and  $K_2$  have  $\kappa(G) = 1$ , graphs without cut vertices that can be disconnected by removing two vertices and  $K_3$  have  $\kappa(G) = 2$ , and so on. We say that a graph is  **$k$ -connected** (or  **$k$ -vertex-connected**), if  $\kappa(G) \geq k$ . A graph  $G$  is 1-connected if it is connected and not a graph containing a single vertex; a graph is 2-connected, or **biconnected**, if it is nonseparable and has at least three vertices. Note that if  $G$  is a  $k$ -connected graph, then  $G$  is a  $j$ -connected graph for all  $j$  with  $0 \leq j \leq k$ .

**EXAMPLE 8** Find the vertex connectivity for each of the graphs in Figure 4.

**Solution:** Each of the five graphs in Figure 4 is connected and has more than one vertex, so each of these graphs has positive vertex connectivity. Because  $G_1$  is a connected graph with a cut vertex, as shown in Example 7, we know that  $\kappa(G_1) = 1$ . Similarly,  $\kappa(G_2) = 1$ , because  $c$  is a cut vertex of  $G_2$ .

The reader should verify that  $G_3$  has no cut vertices, but that  $\{b, g\}$  is a vertex cut. Hence,  $\kappa(G_3) = 2$ . Similarly, because  $G_4$  has a vertex cut of size two,  $\{c, f\}$ , but no cut vertices. It follows that  $\kappa(G_4) = 2$ . The reader can verify that  $G_5$  has no vertex cut of size two, but  $\{b, c, f\}$  is a vertex cut of  $G_5$ . Hence,  $\kappa(G_5) = 3$ . 

**EDGE CONNECTIVITY** We can also measure the connectivity of a connected graph  $G = (V, E)$  in terms of the minimum number of edges that we can remove to disconnect it. If a graph has a cut edge, then we need only remove it to disconnect  $G$ . If  $G$  does not have a cut edge, we look for the smallest set of edges that can be removed to disconnect it. A set of edges  $E'$  is called an **edge cut** of  $G$  if the subgraph  $G - E'$  is disconnected. The **edge connectivity** of a graph  $G$ , denoted by  $\lambda(G)$ , is the minimum number of edges in an edge cut of  $G$ . This defines  $\lambda(G)$  for all connected graphs with more than one vertex because it is always possible to disconnect such a graph by removing all edges incident to one of its vertices. Note that  $\lambda(G) = 0$  if  $G$  is not connected. We also specify that  $\lambda(G) = 0$  if  $G$  is a graph consisting of a single vertex. It follows that if  $G$  is a graph with  $n$  vertices, then  $0 \leq \lambda(G) \leq n - 1$ . We leave it to the reader [Exercise 52(b)] to show that  $\lambda(G) = n - 1$  where  $G$  is a graph with  $n$  vertices if and only if  $G = K_n$ , which is equivalent to the statement that  $\lambda(G) \leq n - 2$  when  $G$  is not a complete graph.

$\lambda$  is the lowercase Greek letter lambda.

**EXAMPLE 9** Find the edge connectivity of each of the graphs in Figure 4.

**Solution:** Each of the five graphs in Figure 4 is connected and has more than one vertex, so we know that all of them have positive edge connectivity. As we saw in Example 7,  $G_1$  has a cut edge, so  $\lambda(G_1) = 1$ .

The graph  $G_2$  has no cut edges, as the reader should verify, but the removal of the two edges  $\{a, b\}$  and  $\{a, c\}$  disconnects it. Hence,  $\lambda(G_2) = 2$ . Similarly,  $\lambda(G_3) = 2$ , because  $G_3$  has no cut edges, but the removal of the two edges  $\{b, c\}$  and  $\{f, g\}$  disconnects it.

The reader should verify that the removal of no two edges disconnects  $G_4$ , but the removal of the three edges  $\{b, c\}$ ,  $\{a, f\}$ , and  $\{f, g\}$  disconnects it. Hence,  $\lambda(G_4) = 3$ . Finally, the reader should verify that  $\lambda(G_5) = 3$ , because the removal of any two of its edges does not disconnect it, but the removal of  $\{a, b\}$ ,  $\{a, g\}$ , and  $\{a, h\}$  does.  $\blacktriangleleft$

### AN INEQUALITY FOR VERTEX CONNECTIVITY AND EDGE CONNECTIVITY

When  $G = (V, E)$  is a noncomplete connected graph with at least three vertices, the minimum degree of a vertex of  $G$  is an upper bound for both the vertex connectivity of  $G$  and the edge connectivity of  $G$ . That is,  $\kappa(G) \leq \min_{v \in V} \deg(v)$  and  $\lambda(G) \leq \min_{v \in V} \deg(v)$ . To see this, observe that deleting all the neighbors of a fixed vertex of minimum degree disconnects  $G$ , and deleting all the edges that have a fixed vertex of minimum degree as an endpoint disconnects  $G$ .

In Exercise 55, we ask the reader to show that  $\kappa(G) \leq \lambda(G)$  when  $G$  is a connected non-complete graph. Note also that  $\kappa(K_n) = \lambda(K_n) = \min_{v \in V} \deg(v) = n - 1$  when  $n$  is a positive integer and that  $\kappa(G) = \lambda(G) = 0$  when  $G$  is a disconnected graph. Putting these facts together, establishes that for all graphs  $G$ ,

$$\kappa(G) \leq \lambda(G) \leq \min_{v \in V} \deg(v).$$

**APPLICATIONS OF VERTEX AND EDGE CONNECTIVITY** Graph connectivity plays an important role in many problems involving the reliability of networks. For instance, as we mentioned in our introduction of cut vertices and cut edges, we can model a data network using vertices to represent routers and edges to represent links between them. The vertex connectivity of the resulting graph equals the minimum number of routers that disconnect the network when they are out of service. If fewer routers are down, data transmission between every pair of routers is still possible. The edge connectivity represents the minimum number of fiber optic links that can be down to disconnect the network. If fewer links are down, it will still be possible for data to be transmitted between every pair of routers.

We can model a highway network, using vertices to represent highway intersections and edges to represent sections of roads running between intersections. The vertex connectivity of the resulting graph represents the minimum number of intersections that can be closed at a particular time that makes it impossible to travel between every two intersections. If fewer intersections are closed, travel between every pair of intersections is still possible. The edge connectivity represents the minimum number of roads that can be closed to disconnect the highway network. If fewer highways are closed, it will still be possible to travel between any two intersections. Clearly, it would be useful for the highway department to take this information into account when planning road repairs.

## Connectedness in Directed Graphs

There are two notions of connectedness in directed graphs, depending on whether the directions of the edges are considered.

### DEFINITION 4

A directed graph is *strongly connected* if there is a path from  $a$  to  $b$  and from  $b$  to  $a$  whenever  $a$  and  $b$  are vertices in the graph.

For a directed graph to be strongly connected there must be a sequence of directed edges from any vertex in the graph to any other vertex. A directed graph can fail to be strongly connected but still be in “one piece.” Definition 5 makes this notion precise.

**DEFINITION 5**

A directed graph is *weakly connected* if there is a path between every two vertices in the underlying undirected graph.

That is, a directed graph is weakly connected if and only if there is always a path between two vertices when the directions of the edges are disregarded. Clearly, any strongly connected directed graph is also weakly connected.

**EXAMPLE 10** Are the directed graphs  $G$  and  $H$  shown in Figure 5 strongly connected? Are they weakly connected?

**Solution:**  $G$  is strongly connected because there is a path between any two vertices in this directed graph (the reader should verify this). Hence,  $G$  is also weakly connected. The graph  $H$  is not strongly connected. There is no directed path from  $a$  to  $b$  in this graph. However,  $H$  is weakly connected, because there is a path between any two vertices in the underlying undirected graph of  $H$  (the reader should verify this).  $\blacktriangleleft$

**STRONG COMPONENTS OF A DIRECTED GRAPH** The subgraphs of a directed graph  $G$  that are strongly connected but not contained in larger strongly connected subgraphs, that is, the maximal strongly connected subgraphs, are called the **strongly connected components** or **strong components** of  $G$ . Note that if  $a$  and  $b$  are two vertices in a directed graph, their strong components are either the same or disjoint. (We leave the proof of this last fact as Exercise 17.)

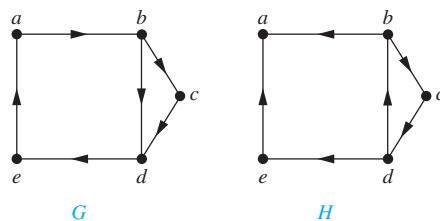
**EXAMPLE 11** The graph  $H$  in Figure 5 has three strongly connected components, consisting of the vertex  $a$ ; the vertex  $e$ ; and the subgraph consisting of the vertices  $b$ ,  $c$ , and  $d$  and edges  $(b, c)$ ,  $(c, d)$ , and  $(d, b)$ .  $\blacktriangleleft$

**EXAMPLE 12**

In 2010 the Web graph was estimated to have at least 55 billion vertices and one trillion edges. This implies that more than 40 TB of computer memory would have been needed to represent its adjacency matrix.

**The Strongly Connected Components of the Web Graph** The Web graph introduced in Example 5 of Section 10.1 represents Web pages with vertices and links with directed edges. A snapshot of the Web in 1999 produced a Web graph with over 200 million vertices and over 1.5 billion edges (numbers that have now grown considerably). (See [Br00] for details.)

The underlying undirected graph of this Web graph is not connected, but it has a connected component that includes approximately 90% of the vertices in the graph. The subgraph of the original directed graph corresponding to this connected component of the underlying undirected graph (that is, with the same vertices and all directed edges connecting vertices in this graph) has one very large strongly connected component and many small ones. The former is called the **giant strongly connected component (GSCC)** of the directed graph. A Web page in this component can be reached following links starting at any other page in this component. The GSCC in the Web graph produced by this study was found to have over 53 million vertices. The remaining vertices in the large connected component of the undirected graph represent three different types of Web pages: pages that can be reached from a page in the GSCC, but do not link back to these pages following a series of links; pages that link back to pages in the



**FIGURE 5** The Directed Graphs  $G$  and  $H$ .

GSCC following a series of links, but cannot be reached by following links on pages in the GSCC; and pages that cannot reach pages in the GSCC and cannot be reached from pages in the GSCC following a series of links. In this study, each of these three other sets was found to have approximately 44 million vertices. (It is rather surprising that these three sets are close to the same size.)

## Paths and Isomorphism

There are several ways that paths and circuits can help determine whether two graphs are isomorphic. For example, the existence of a simple circuit of a particular length is a useful invariant that can be used to show that two graphs are not isomorphic. In addition, paths can be used to construct mappings that may be isomorphisms.

As we mentioned, a useful isomorphic invariant for simple graphs is the existence of a simple circuit of length  $k$ , where  $k$  is a positive integer greater than 2. (The proof that this is an invariant is left as Exercise 60.) Example 13 illustrates how this invariant can be used to show that two graphs are not isomorphic.

**EXAMPLE 13** Determine whether the graphs  $G$  and  $H$  shown in Figure 6 are isomorphic.

**Solution:** Both  $G$  and  $H$  have six vertices and eight edges. Each has four vertices of degree three, and two vertices of degree two. So, the three invariants—number of vertices, number of edges, and degrees of vertices—all agree for the two graphs. However,  $H$  has a simple circuit of length three, namely,  $v_1, v_2, v_6, v_1$ , whereas  $G$  has no simple circuit of length three, as can be determined by inspection (all simple circuits in  $G$  have length at least four). Because the existence of a simple circuit of length three is an isomorphic invariant,  $G$  and  $H$  are not isomorphic.

We have shown how the existence of a type of path, namely, a simple circuit of a particular length, can be used to show that two graphs are not isomorphic. We can also use paths to find mappings that are potential isomorphisms.

**EXAMPLE 14** Determine whether the graphs  $G$  and  $H$  shown in Figure 7 are isomorphic.

**Solution:** Both  $G$  and  $H$  have five vertices and six edges, both have two vertices of degree three and three vertices of degree two, and both have a simple circuit of length three, a simple circuit of length four, and a simple circuit of length five. Because all these isomorphic invariants agree,  $G$  and  $H$  may be isomorphic.

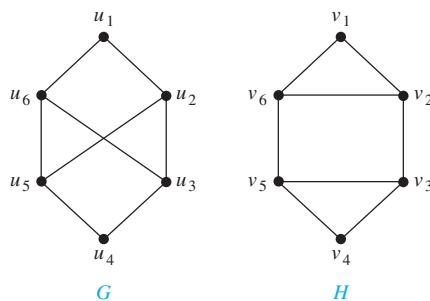


FIGURE 6 The Graphs  $G$  and  $H$ .

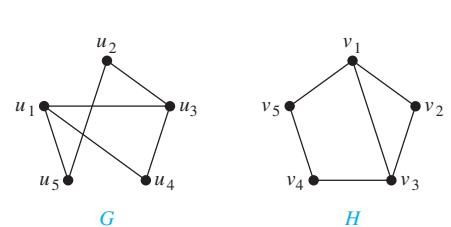


FIGURE 7 The Graphs  $G$  and  $H$ .

To find a possible isomorphism, we can follow paths that go through all vertices so that the corresponding vertices in the two graphs have the same degree. For example, the paths  $u_1, u_4, u_3, u_2, u_5$  in  $G$  and  $v_3, v_2, v_1, v_5, v_4$  in  $H$  both go through every vertex in the graph; start at a vertex of degree three; go through vertices of degrees two, three, and two, respectively; and end at a vertex of degree two. By following these paths through the graphs, we define the mapping  $f$  with  $f(u_1) = v_3, f(u_4) = v_2, f(u_3) = v_1, f(u_2) = v_5$ , and  $f(u_5) = v_4$ . The reader can show that  $f$  is an isomorphism, so  $G$  and  $H$  are isomorphic, either by showing that  $f$  preserves edges or by showing that with the appropriate orderings of vertices the adjacency matrices of  $G$  and  $H$  are the same.  $\blacktriangleleft$

## Counting Paths Between Vertices

The number of paths between two vertices in a graph can be determined using its adjacency matrix.

### THEOREM 2

Let  $G$  be a graph with adjacency matrix  $\mathbf{A}$  with respect to the ordering  $v_1, v_2, \dots, v_n$  of the vertices of the graph (with directed or undirected edges, with multiple edges and loops allowed). The number of different paths of length  $r$  from  $v_i$  to  $v_j$ , where  $r$  is a positive integer, equals the  $(i, j)$ th entry of  $\mathbf{A}^r$ .

**Proof:** The theorem will be proved using mathematical induction. Let  $G$  be a graph with adjacency matrix  $\mathbf{A}$  (assuming an ordering  $v_1, v_2, \dots, v_n$  of the vertices of  $G$ ). The number of paths from  $v_i$  to  $v_j$  of length 1 is the  $(i, j)$ th entry of  $\mathbf{A}$ , because this entry is the number of edges from  $v_i$  to  $v_j$ .

Assume that the  $(i, j)$ th entry of  $\mathbf{A}^r$  is the number of different paths of length  $r$  from  $v_i$  to  $v_j$ . This is the inductive hypothesis. Because  $\mathbf{A}^{r+1} = \mathbf{A}^r \mathbf{A}$ , the  $(i, j)$ th entry of  $\mathbf{A}^{r+1}$  equals

$$b_{i1}a_{1j} + b_{i2}a_{2j} + \cdots + b_{in}a_{nj},$$

where  $b_{ik}$  is the  $(i, k)$ th entry of  $\mathbf{A}^r$ . By the inductive hypothesis,  $b_{ik}$  is the number of paths of length  $r$  from  $v_i$  to  $v_k$ .

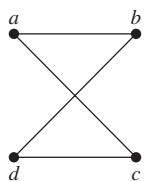
A path of length  $r + 1$  from  $v_i$  to  $v_j$  is made up of a path of length  $r$  from  $v_i$  to some intermediate vertex  $v_k$ , and an edge from  $v_k$  to  $v_j$ . By the product rule for counting, the number of such paths is the product of the number of paths of length  $r$  from  $v_i$  to  $v_k$ , namely,  $b_{ik}$ , and the number of edges from  $v_k$  to  $v_j$ , namely,  $a_{kj}$ . When these products are added for all possible intermediate vertices  $v_k$ , the desired result follows by the sum rule for counting.  $\blacktriangleleft$

### EXAMPLE 15

How many paths of length four are there from  $a$  to  $d$  in the simple graph  $G$  in Figure 8?

**Solution:** The adjacency matrix of  $G$  (ordering the vertices as  $a, b, c, d$ ) is

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$



**FIGURE 8** The Graph  $G$ .

Hence, the number of paths of length four from  $a$  to  $d$  is the  $(1, 4)$ th entry of  $\mathbf{A}^4$ . Because

$$\mathbf{A}^4 = \begin{bmatrix} 8 & 0 & 0 & 8 \\ 0 & 8 & 8 & 0 \\ 0 & 8 & 8 & 0 \\ 8 & 0 & 0 & 8 \end{bmatrix},$$

**Extra Examples**

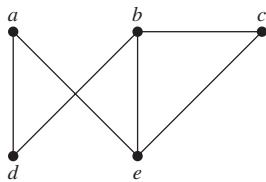
there are exactly eight paths of length four from  $a$  to  $d$ . By inspection of the graph, we see that  $a, b, a, b, d; a, b, a, c, d; a, b, d, b, d; a, b, d, c, d; a, c, a, b, d; a, c, a, c, d; a, c, d, b, d$ ; and  $a, c, d, c, d$  are the eight paths of length four from  $a$  to  $d$ .

Theorem 2 can be used to find the length of the shortest path between two vertices of a graph (see Exercise 56), and it can also be used to determine whether a graph is connected (see Exercises 61 and 62).

## Exercises

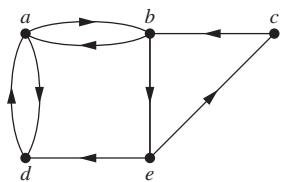
1. Does each of these lists of vertices form a path in the following graph? Which paths are simple? Which are circuits? What are the lengths of those that are paths?

- a)  $a, e, b, c, b$   
b)  $a, e, a, d, b, c, a$   
c)  $e, b, a, d, b, e$   
d)  $c, b, d, a, e, c$

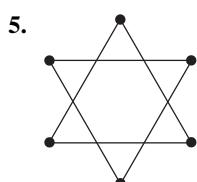
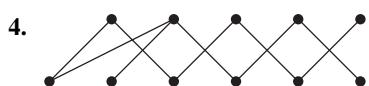


2. Does each of these lists of vertices form a path in the following graph? Which paths are simple? Which are circuits? What are the lengths of those that are paths?

- a)  $a, b, e, c, b$   
b)  $a, d, a, d, a$   
c)  $a, d, b, e, a$   
d)  $a, b, e, c, b, d, a$



In Exercises 3–5 determine whether the given graph is connected.



6. How many connected components does each of the graphs in Exercises 3–5 have? For each graph find each of its connected components.

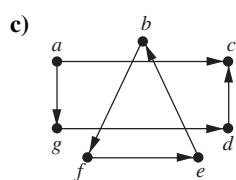
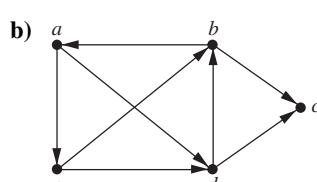
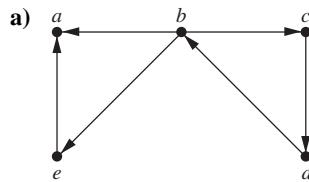
7. What do the connected components of acquaintance graphs represent?

8. What do the connected components of a collaboration graph represent?

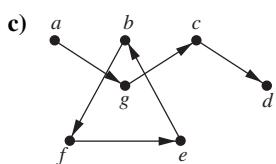
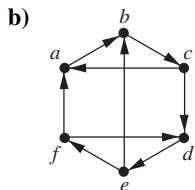
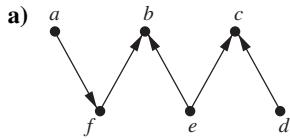
9. Explain why in the collaboration graph of mathematicians (see Example 3 in Section 10.1) a vertex representing a mathematician is in the same connected component as the vertex representing Paul Erdős if and only if that mathematician has a finite Erdős number.

10. In the Hollywood graph (see Example 3 in Section 10.1), when is the vertex representing an actor in the same connected component as the vertex representing Kevin Bacon?

11. Determine whether each of these graphs is strongly connected and if not, whether it is weakly connected.

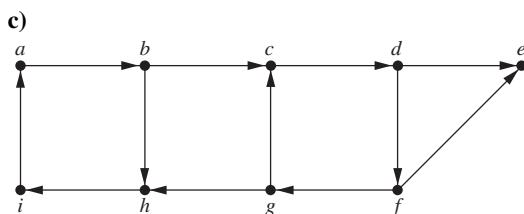
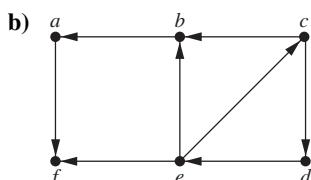
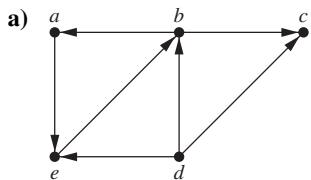


12. Determine whether each of these graphs is strongly connected and if not, whether it is weakly connected.

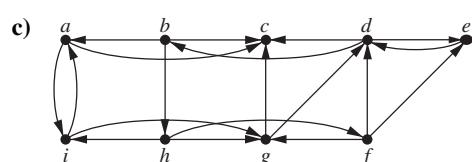
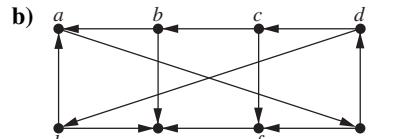
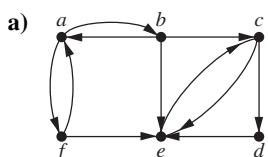


13. What do the strongly connected components of a telephone call graph represent?

14. Find the strongly connected components of each of these graphs.



15. Find the strongly connected components of each of these graphs.



Suppose that  $G = (V, E)$  is a directed graph. A vertex  $w \in V$  is **reachable** from a vertex  $v \in V$  if there is a directed path from  $v$  to  $w$ . The vertices  $v$  and  $w$  are **mutually reachable** if there are both a directed path from  $v$  to  $w$  and a directed path from  $w$  to  $v$  in  $G$ .

16. Show that if  $G = (V, E)$  is a directed graph and  $u, v$ , and  $w$  are vertices in  $V$  for which  $u$  and  $v$  are mutually reachable and  $v$  and  $w$  are mutually reachable, then  $u$  and  $w$  are mutually reachable.

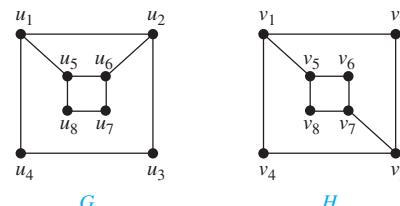
17. Show that if  $G = (V, E)$  is a directed graph, then the strong components of two vertices  $u$  and  $v$  of  $V$  are either the same or disjoint. [Hint: Use Exercise 16.]

18. Show that all vertices visited in a directed path connecting two vertices in the same strongly connected component of a directed graph are also in this strongly connected component.

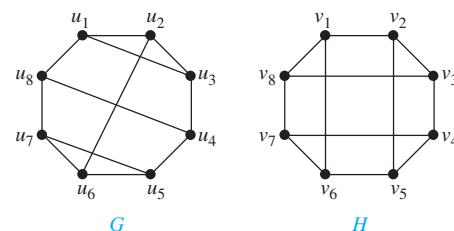
19. Find the number of paths of length  $n$  between two different vertices in  $K_4$  if  $n$  is

- a) 2.      b) 3.      c) 4.      d) 5.

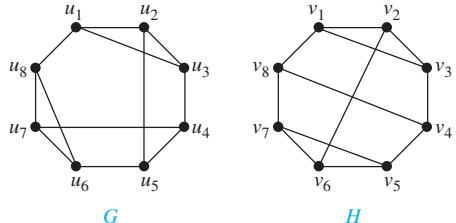
20. Use paths either to show that these graphs are not isomorphic or to find an isomorphism between these graphs.



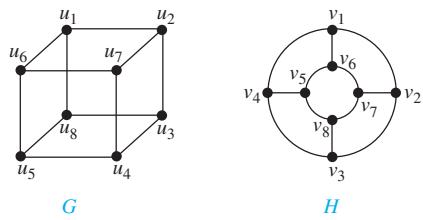
21. Use paths either to show that these graphs are not isomorphic or to find an isomorphism between them.



22. Use paths either to show that these graphs are not isomorphic or to find an isomorphism between them.



23. Use paths either to show that these graphs are not isomorphic or to find an isomorphism between them.



24. Find the number of paths of length  $n$  between any two adjacent vertices in  $K_{3,3}$  for the values of  $n$  in Exercise 19.  
 25. Find the number of paths of length  $n$  between any two nonadjacent vertices in  $K_{3,3}$  for the values of  $n$  in Exercise 19.  
 26. Find the number of paths between  $c$  and  $d$  in the graph in Figure 1 of length

a) 2.    b) 3.    c) 4.    d) 5.    e) 6.    f) 7.

27. Find the number of paths from  $a$  to  $e$  in the directed graph in Exercise 2 of length

a) 2.    b) 3.    c) 4.    d) 5.    e) 6.    f) 7.

- \*28. Show that every connected graph with  $n$  vertices has at least  $n - 1$  edges.  
 29. Let  $G = (V, E)$  be a simple graph. Let  $R$  be the relation on  $V$  consisting of pairs of vertices  $(u, v)$  such that there is a path from  $u$  to  $v$  or such that  $u = v$ . Show that  $R$  is an equivalence relation.  
 \*30. Show that in every simple graph there is a path from every vertex of odd degree to some other vertex of odd degree.

In Exercises 31–33 find all the cut vertices of the given graph.

31. A graph with 6 vertices labeled  $a$  through  $f$ . Vertex  $a$  is connected to  $b$  and  $d$ . Vertex  $b$  is connected to  $a$ ,  $c$ , and  $d$ . Vertex  $c$  is connected to  $b$  and  $f$ . Vertex  $d$  is connected to  $a$ ,  $b$ , and  $e$ . Vertex  $e$  is connected to  $d$  and  $f$ . Vertex  $f$  is connected to  $e$ .  
 32. A graph with 6 vertices labeled  $a$  through  $f$ . Vertex  $a$  is connected to  $b$  and  $c$ . Vertex  $b$  is connected to  $a$ ,  $c$ , and  $d$ . Vertex  $c$  is connected to  $b$ ,  $d$ , and  $e$ . Vertex  $d$  is connected to  $b$ ,  $c$ , and  $e$ . Vertex  $e$  is connected to  $d$  and  $f$ . Vertex  $f$  is connected to  $e$ .  
 33. A graph with 8 vertices labeled  $a$  through  $h$ . Vertex  $a$  is connected to  $b$ . Vertex  $b$  is connected to  $a$  and  $c$ . Vertex  $c$  is connected to  $b$ ,  $d$ , and  $e$ . Vertex  $d$  is connected to  $c$ . Vertex  $e$  is connected to  $c$ ,  $f$ , and  $g$ . Vertex  $f$  is connected to  $e$ . Vertex  $g$  is connected to  $e$ . Vertex  $i$  is connected to  $h$ .

34. Find all the cut edges in the graphs in Exercises 31–33.

- \*35. Suppose that  $v$  is an endpoint of a cut edge. Prove that  $v$  is a cut vertex if and only if this vertex is not pendant.

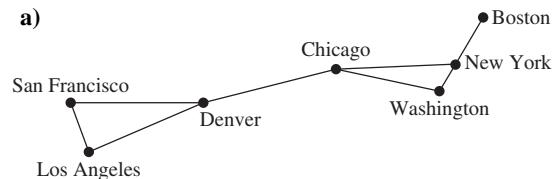
- \*36. Show that a vertex  $c$  in the connected simple graph  $G$  is a cut vertex if and only if there are vertices  $u$  and  $v$ , both different from  $c$ , such that every path between  $u$  and  $v$  passes through  $c$ .

- \*37. Show that a simple graph with at least two vertices has at least two vertices that are not cut vertices.

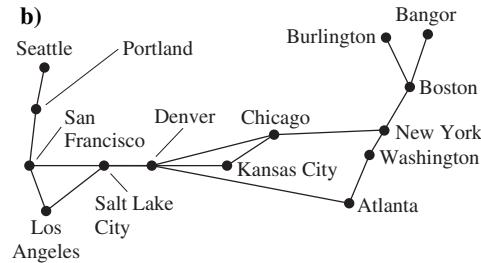
- \*38. Show that an edge in a simple graph is a cut edge if and only if this edge is not part of any simple circuit in the graph.

39. A communications link in a network should be provided with a backup link if its failure makes it impossible for some message to be sent. For each of the communications networks shown here in (a) and (b), determine those links that should be backed up.

a)



b)



A **vertex basis** in a directed graph  $G$  is a minimal set  $B$  of vertices of  $G$  such that for each vertex  $v$  of  $G$  not in  $B$  there is a path to  $v$  from some vertex in  $B$ .

40. Find a vertex basis for each of the directed graphs in Exercises 7–9 of Section 10.2.

41. What is the significance of a vertex basis in an influence graph (described in Example 2 of Section 10.1)? Find a vertex basis in the influence graph in that example.

42. Show that if a connected simple graph  $G$  is the union of the graphs  $G_1$  and  $G_2$ , then  $G_1$  and  $G_2$  have at least one common vertex.

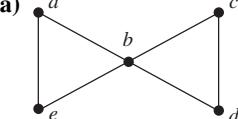
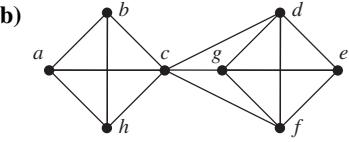
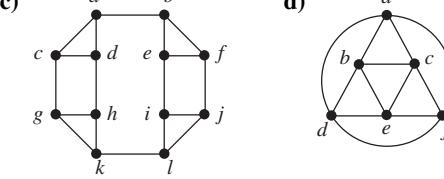
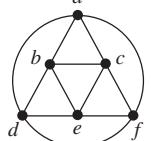
- \*43. Show that if a simple graph  $G$  has  $k$  connected components and these components have  $n_1, n_2, \dots, n_k$  vertices, respectively, then the number of edges of  $G$  does not exceed

$$\sum_{i=1}^k C(n_i, 2).$$

- \*44. Use Exercise 43 to show that a simple graph with  $n$  vertices and  $k$  connected components has at most  $(n - k)(n - k + 1)/2$  edges. [Hint: First show that

$$\sum_{i=1}^k n_i^2 \leq n^2 - (k-1)(2n-k),$$

where  $n_i$  is the number of vertices in the  $i$ th connected component.]

- \*45. Show that a simple graph  $G$  with  $n$  vertices is connected if it has more than  $(n - 1)(n - 2)/2$  edges.
46. Describe the adjacency matrix of a graph with  $n$  connected components when the vertices of the graph are listed so that vertices in each connected component are listed successively.
47. How many nonisomorphic connected simple graphs are there with  $n$  vertices when  $n$  is
- a) 2?      b) 3?      c) 4?      d) 5?
48. Show that each of the following graphs has no cut vertices.
- a)  $C_n$  where  $n \geq 3$   
 b)  $W_n$  where  $n \geq 3$   
 c)  $K_{m,n}$  where  $m \geq 2$  and  $n \geq 2$   
 d)  $Q_n$  where  $n \geq 2$
49. Show that each of the graphs in Exercise 48 has no cut edges.
50. For each of these graphs, find  $\kappa(G)$ ,  $\lambda(G)$ , and  $\min_{v \in V} \deg(v)$ , and determine which of the two inequalities in  $\kappa(G) \leq \lambda(G) \leq \min_{v \in V} \deg(v)$  are strict.
- a) 
- b) 
- c) 
- d) 
51. Show that if  $G$  is a connected graph, then it is possible to remove vertices to disconnect  $G$  if and only if  $G$  is not a complete graph.
52. Show that if  $G$  is a connected graph with  $n$  vertices then
- a)  $\kappa(G) = n - 1$  if and only if  $G = K_n$ .  
 b)  $\lambda(G) = n - 1$  if and only if  $G = K_n$ .
53. Find  $\kappa(K_{m,n})$  and  $\lambda(K_{m,n})$ , where  $m$  and  $n$  are positive integers.
54. Construct a graph  $G$  with  $\kappa(G) = 1$ ,  $\lambda(G) = 2$ , and  $\min_{v \in V} \deg(v) = 3$ .
- \*55. Show that if  $G$  is a graph, then  $\kappa(G) \leq \lambda(G)$ .
56. Explain how Theorem 2 can be used to find the length of the shortest path from a vertex  $v$  to a vertex  $w$  in a graph.
57. Use Theorem 2 to find the length of the shortest path between  $a$  and  $f$  in the graph in Figure 1.
58. Use Theorem 2 to find the length of the shortest path from  $a$  to  $c$  in the directed graph in Exercise 2.
59. Let  $P_1$  and  $P_2$  be two simple paths between the vertices  $u$  and  $v$  in the simple graph  $G$  that do not contain the same set of edges. Show that there is a simple circuit in  $G$ .
60. Show that the existence of a simple circuit of length  $k$ , where  $k$  is an integer greater than 2, is an invariant for graph isomorphism.
61. Explain how Theorem 2 can be used to determine whether a graph is connected.
62. Use Exercise 61 to show that the graph  $G_1$  in Figure 2 is connected whereas the graph  $G_2$  in that figure is not connected.
63. Show that a simple graph  $G$  is bipartite if and only if it has no circuits with an odd number of edges.
64. In an old puzzle attributed to Alcuin of York (735–804), a farmer needs to carry a wolf, a goat, and a cabbage across a river. The farmer only has a small boat, which can carry the farmer and only one object (an animal or a vegetable). He can cross the river repeatedly. However, if the farmer is on the other shore, the wolf will eat the goat, and, similarly, the goat will eat the cabbage. We can describe each state by listing what is on each shore. For example, we can use the pair  $(FC, WC)$  for the state where the farmer and goat are on the first shore and the wolf and cabbage are on the other shore. [The symbol  $\emptyset$  is used when nothing is on a shore, so that  $(FWGC, \emptyset)$  is the initial state.]
- a) Find all allowable states of the puzzle, where neither the wolf and the goat nor the goat and the cabbage are left on the same shore without the farmer.
- b) Construct a graph such that each vertex of this graph represents an allowable state and the vertices representing two allowable states are connected by an edge if it is possible to move from one state to the other using one trip of the boat.
- c) Explain why finding a path from the vertex representing  $(FWGC, \emptyset)$  to the vertex representing  $(\emptyset, FWGC)$  solves the puzzle.
- d) Find two different solutions of the puzzle, each using seven crossings.
- e) Suppose that the farmer must pay a toll of one dollar whenever he crosses the river with an animal. Which solution of the puzzle should the farmer use to pay the least total toll?

\*65. Use a graph model and a path in your graph, as in Exercise 64, to solve the **jealous husbands problem**. Two married couples, each a husband and a wife, want to cross a river. They can only use a boat that can carry one or two people from one shore to the other shore. Each husband is extremely jealous and is not willing to leave his wife with the other husband, either in the boat or on shore. How can these four people reach the opposite shore?

66. Suppose that you have a three-gallon jug and a five-gallon jug. You may fill either jug with water, you may empty either jug, and you may transfer water from either jug into the other jug. Use a path in a directed graph to show that you can end up with a jug containing exactly one gallon. [Hint: Use an ordered pair  $(a, b)$  to indicate how much water is in each jug. Represent these ordered pairs by vertices. Add an edge for each allowable operation with the jugs.]

## 10.5 Euler and Hamilton Paths

### Introduction

Can we travel along the edges of a graph starting at a vertex and returning to it by traversing each edge of the graph exactly once? Similarly, can we travel along the edges of a graph starting at a vertex and returning to it while visiting each vertex of the graph exactly once? Although these questions seem to be similar, the first question, which asks whether a graph has an *Euler circuit*, can be easily answered simply by examining the degrees of the vertices of the graph, while the second question, which asks whether a graph has a *Hamilton circuit*, is quite difficult to solve for most graphs. In this section we will study these questions and discuss the difficulty of solving them. Although both questions have many practical applications in many different areas, both arose in old puzzles. We will learn about these old puzzles as well as modern practical applications.

### Euler Paths and Circuits

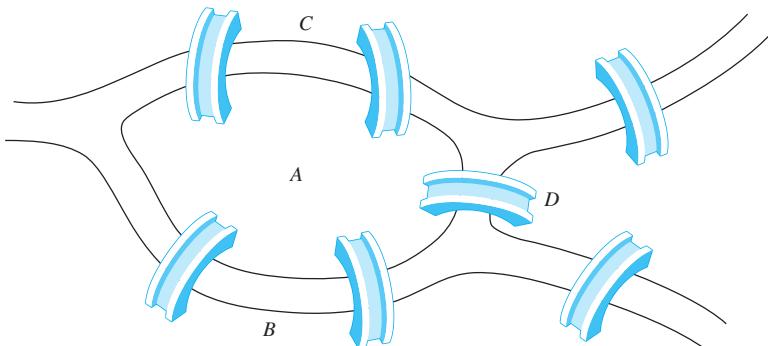
The town of Königsberg, Prussia (now called Kaliningrad and part of the Russian republic), was divided into four sections by the branches of the Pregel River. These four sections included the two regions on the banks of the Pregel, Kneiphof Island, and the region between the two branches of the Pregel. In the eighteenth century seven bridges connected these regions. Figure 1 depicts these regions and bridges.



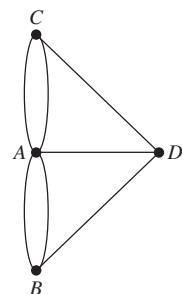
Only five bridges connect Kaliningrad today. Of these, just two remain from Euler's day.

The townspeople took long walks through town on Sundays. They wondered whether it was possible to start at some location in the town, travel across all the bridges once without crossing any bridge twice, and return to the starting point.

The Swiss mathematician Leonhard Euler solved this problem. His solution, published in 1736, may be the first use of graph theory. (For a translation of Euler's original paper see [BiLiWi99].) Euler studied this problem using the multigraph obtained when the four regions are represented by vertices and the bridges by edges. This multigraph is shown in Figure 2.



**FIGURE 1** The Seven Bridges of Königsberg.



**FIGURE 2** Multigraph Model of the Town of Königsberg.

The problem of traveling across every bridge without crossing any bridge more than once can be rephrased in terms of this model. The question becomes: Is there a simple circuit in this multigraph that contains every edge?

### DEFINITION 1

An *Euler circuit* in a graph  $G$  is a simple circuit containing every edge of  $G$ . An *Euler path* in  $G$  is a simple path containing every edge of  $G$ .

Examples 1 and 2 illustrate the concept of Euler circuits and paths.

**EXAMPLE 1** Which of the undirected graphs in Figure 3 have an Euler circuit? Of those that do not, which have an Euler path?

**Solution:** The graph  $G_1$  has an Euler circuit, for example,  $a, e, c, d, e, b, a$ . Neither of the graphs  $G_2$  or  $G_3$  has an Euler circuit (the reader should verify this). However,  $G_3$  has an Euler path, namely,  $a, c, d, e, b, d, a, b$ .  $G_2$  does not have an Euler path (as the reader should verify). 

**EXAMPLE 2** Which of the directed graphs in Figure 4 have an Euler circuit? Of those that do not, which have an Euler path?



**Solution:** The graph  $H_2$  has an Euler circuit, for example,  $a, g, c, b, g, e, d, f, a$ . Neither  $H_1$  nor  $H_3$  has an Euler circuit (as the reader should verify).  $H_3$  has an Euler path, namely,  $c, a, b, c, d, b$ , but  $H_1$  does not (as the reader should verify). 

### NECESSARY AND SUFFICIENT CONDITIONS FOR EULER CIRCUITS AND PATHS

There are simple criteria for determining whether a multigraph has an Euler circuit or an Euler path. Euler discovered them when he solved the famous Königsberg bridge problem. We will assume that all graphs discussed in this section have a finite number of vertices and edges.

What can we say if a connected multigraph has an Euler circuit? What we can show is that every vertex must have even degree. To do this, first note that an Euler circuit begins with a vertex  $a$  and continues with an edge incident with  $a$ , say  $\{a, b\}$ . The edge  $\{a, b\}$  contributes one to  $\deg(a)$ . Each time the circuit passes through a vertex it contributes two to the vertex's degree, because the circuit enters via an edge incident with this vertex and leaves via another such edge. Finally, the circuit terminates where it started, contributing one to  $\deg(a)$ . Therefore,  $\deg(a)$  must be even, because the circuit contributes one when it begins, one when it ends, and two every time it passes through  $a$  (if it ever does). A vertex other than  $a$  has even degree because the circuit contributes two to its degree each time it passes through the vertex. We conclude that if a connected graph has an Euler circuit, then every vertex must have even degree.

Is this necessary condition for the existence of an Euler circuit also sufficient? That is, must an Euler circuit exist in a connected multigraph if all vertices have even degree? This question can be settled affirmatively with a construction.

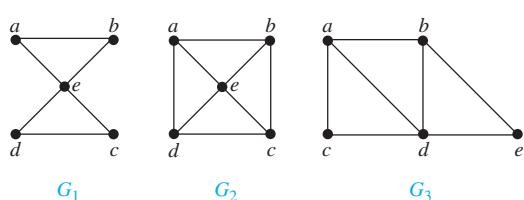


FIGURE 3 The Undirected Graphs  $G_1$ ,  $G_2$ , and  $G_3$ .

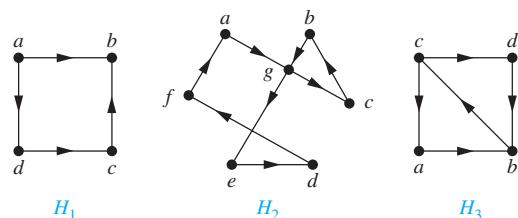
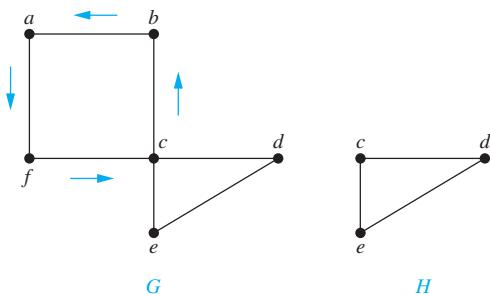


FIGURE 4 The Directed Graphs  $H_1$ ,  $H_2$ , and  $H_3$ .



**FIGURE 5** Constructing an Euler Circuit in  $G$ .

Suppose that  $G$  is a connected multigraph with at least two vertices and the degree of every vertex of  $G$  is even. We will form a simple circuit that begins at an arbitrary vertex  $a$  of  $G$ , building it edge by edge. Let  $x_0 = a$ . First, we arbitrarily choose an edge  $\{x_0, x_1\}$  incident with  $a$  which is possible because  $G$  is connected. We continue by building a simple path  $\{x_0, x_1\}, \{x_1, x_2\}, \dots, \{x_{n-1}, x_n\}$ , successively adding edges one by one to the path until we cannot add another edge to the path. This happens when we reach a vertex for which we have already included all edges incident with that vertex in the path. For instance, in the graph  $G$  in Figure 5 we begin at  $a$  and choose in succession the edges  $\{a, f\}$ ,  $\{f, c\}$ ,  $\{c, b\}$ , and  $\{b, a\}$ .

The path we have constructed must terminate because the graph has a finite number of edges, so we are guaranteed to eventually reach a vertex for which no edges are available to add to the path. The path begins at  $a$  with an edge of the form  $\{a, x\}$ , and we now show that it must terminate at  $a$  with an edge of the form  $\{y, a\}$ . To see that the path must terminate at  $a$ , note that each time the path goes through a vertex with even degree, it uses only one edge to enter this vertex, so because the degree must be at least two, at least one edge remains for the path to leave the vertex. Furthermore, every time we enter and leave a vertex of even degree, there are an even number of edges incident with this vertex that we have not yet used in our path. Consequently, as we form the path, every time we enter a vertex other than  $a$ , we can leave it. This means that the path can end only at  $a$ . Next, note that the path we have constructed may use all the edges of the graph, or it may not if we have returned to  $a$  for the last time before using all the edges.

An Euler circuit has been constructed if all the edges have been used. Otherwise, consider the subgraph  $H$  obtained from  $G$  by deleting the edges already used and vertices that are not incident with any remaining edges. When we delete the circuit  $a, f, c, b, a$  from the graph in Figure 5, we obtain the subgraph labeled as  $H$ .

Because  $G$  is connected,  $H$  has at least one vertex in common with the circuit that has been deleted. Let  $w$  be such a vertex. (In our example,  $c$  is the vertex.)

### Links



**LEONHARD EULER (1707–1783)** Leonhard Euler was the son of a Calvinist minister from the vicinity of Basel, Switzerland. At 13 he entered the University of Basel, pursuing a career in theology, as his father wished. At the university Euler was tutored by Johann Bernoulli of the famous Bernoulli family of mathematicians. His interest and skills led him to abandon his theological studies and take up mathematics. Euler obtained his master's degree in philosophy at the age of 16. In 1727 Peter the Great invited him to join the Academy at St. Petersburg. In 1741 he moved to the Berlin Academy, where he stayed until 1766. He then returned to St. Petersburg, where he remained for the rest of his life.

Euler was incredibly prolific, contributing to many areas of mathematics, including number theory, combinatorics, and analysis, as well as its applications to such areas as music and naval architecture. He wrote over 1100 books and papers and left so much unpublished work that it took 47 years after he died for all his work to be published. During his life his papers accumulated so quickly that he kept a large pile of articles awaiting publication. The Berlin Academy published the papers on top of this pile so later results were often published before results they depended on or superseded. Euler had 13 children and was able to continue his work while a child or two bounced on his knees. He was blind for the last 17 years of his life, but because of his fantastic memory this did not diminish his mathematical output. The project of publishing his collected works, undertaken by the Swiss Society of Natural Science, is ongoing and will require more than 75 volumes.

Every vertex in  $H$  has even degree (because in  $G$  all vertices had even degree, and for each vertex, pairs of edges incident with this vertex have been deleted to form  $H$ ). Note that  $H$  may not be connected. Beginning at  $w$ , construct a simple path in  $H$  by choosing edges as long as possible, as was done in  $G$ . This path must terminate at  $w$ . For instance, in Figure 5,  $c, d, e, c$  is a path in  $H$ . Next, form a circuit in  $G$  by splicing the circuit in  $H$  with the original circuit in  $G$  (this can be done because  $w$  is one of the vertices in this circuit). When this is done in the graph in Figure 5, we obtain the circuit  $a, f, c, d, e, c, b, a$ .

Continue this process until all edges have been used. (The process must terminate because there are only a finite number of edges in the graph.) This produces an Euler circuit. The construction shows that if the vertices of a connected multigraph all have even degree, then the graph has an Euler circuit.

We summarize these results in Theorem 1.

### THEOREM 1

A connected multigraph with at least two vertices has an Euler circuit if and only if each of its vertices has even degree.

We can now solve the Königsberg bridge problem. Because the multigraph representing these bridges, shown in Figure 2, has four vertices of odd degree, it does not have an Euler circuit. There is no way to start at a given point, cross each bridge exactly once, and return to the starting point.

Algorithm 1 gives the constructive procedure for finding Euler circuits given in the discussion preceding Theorem 1. (Because the circuits in the procedure are chosen arbitrarily, there is some ambiguity. We will not bother to remove this ambiguity by specifying the steps of the procedure more precisely.)

#### ALGORITHM 1 Constructing Euler Circuits.

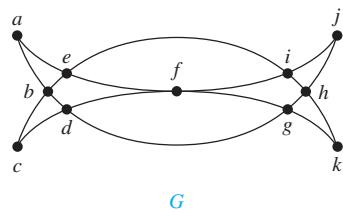
```

procedure Euler(G : connected multigraph with all vertices of
even degree)
 circuit := a circuit in G beginning at an arbitrarily chosen
 vertex with edges successively added to form a path that
 returns to this vertex
 $H := G$ with the edges of this circuit removed
 while H has edges
 subcircuit := a circuit in H beginning at a vertex in H that
 also is an endpoint of an edge of circuit
 $H := H$ with edges of subcircuit and all isolated vertices
 removed
 circuit := circuit with subcircuit inserted at the appropriate
 vertex
 return circuit {circuit is an Euler circuit}

```

Algorithm 1 provides an efficient algorithm for finding Euler circuits in a connected multigraph  $G$  with all vertices of even degree. We leave it to the reader (Exercise 66) to show that the worst case complexity of this algorithm is  $O(m)$ , where  $m$  is the number of edges of  $G$ .

Example 3 shows how Euler paths and circuits can be used to solve a type of puzzle.

**FIGURE 6** Mohammed's Scimitars.**EXAMPLE 3**

Many puzzles ask you to draw a picture in a continuous motion without lifting a pencil so that no part of the picture is retraced. We can solve such puzzles using Euler circuits and paths. For example, can *Mohammed's scimitars*, shown in Figure 6, be drawn in this way, where the drawing begins and ends at the same point?

**Solution:** We can solve this problem because the graph  $G$  shown in Figure 6 has an Euler circuit. It has such a circuit because all its vertices have even degree. We will use Algorithm 1 to construct an Euler circuit. First, we form the circuit  $a, b, d, c, b, e, i, f, e, a$ . We obtain the subgraph  $H$  by deleting the edges in this circuit and all vertices that become isolated when these edges are removed. Then we form the circuit  $d, g, h, j, i, h, k, g, f, d$  in  $H$ . After forming this circuit we have used all edges in  $G$ . Splicing this new circuit into the first circuit at the appropriate place produces the Euler circuit  $a, b, d, g, h, j, i, h, k, g, f, d, c, b, e, i, f, e, a$ . This circuit gives a way to draw the scimitars without lifting the pencil or retracing part of the picture.  $\blacktriangleleft$

Another algorithm for constructing Euler circuits, called Fleury's algorithm, is described in the preamble to Exercise 50.

We will now show that a connected multigraph has an Euler path (and not an Euler circuit) if and only if it has exactly two vertices of odd degree. First, suppose that a connected multigraph does have an Euler path from  $a$  to  $b$ , but not an Euler circuit. The first edge of the path contributes one to the degree of  $a$ . A contribution of two to the degree of  $a$  is made every time the path passes through  $a$ . The last edge in the path contributes one to the degree of  $b$ . Every time the path goes through  $b$  there is a contribution of two to its degree. Consequently, both  $a$  and  $b$  have odd degree. Every other vertex has even degree, because the path contributes two to the degree of a vertex whenever it passes through it.

Now consider the converse. Suppose that a graph has exactly two vertices of odd degree, say  $a$  and  $b$ . Consider the larger graph made up of the original graph with the addition of an edge  $\{a, b\}$ . Every vertex of this larger graph has even degree, so there is an Euler circuit. The removal of the new edge produces an Euler path in the original graph. Theorem 2 summarizes these results.

**THEOREM 2**

A connected multigraph has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.

**EXAMPLE 4**

Which graphs shown in Figure 7 have an Euler path?

**Solution:**  $G_1$  contains exactly two vertices of odd degree, namely,  $b$  and  $d$ . Hence, it has an Euler path that must have  $b$  and  $d$  as its endpoints. One such Euler path is  $d, a, b, c, d, b$ . Similarly,  $G_2$  has exactly two vertices of odd degree, namely,  $b$  and  $d$ . So it has an Euler path that must have  $b$  and  $d$  as endpoints. One such Euler path is  $b, a, g, f, e, d, c, g, b, c, f, d$ .  $G_3$  has no Euler path because it has six vertices of odd degree.  $\blacktriangleleft$

Returning to eighteenth-century Königsberg, is it possible to start at some point in the town, travel across all the bridges, and end up at some other point in town? This question can

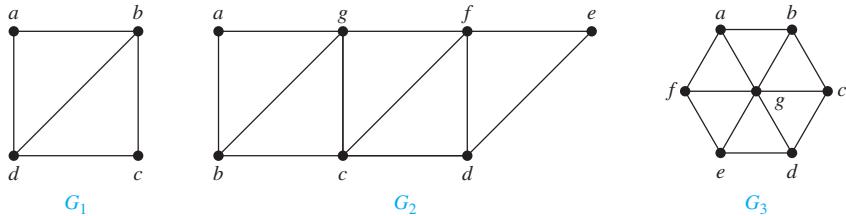


FIGURE 7 Three Undirected Graphs.

be answered by determining whether there is an Euler path in the multigraph representing the bridges in Königsberg. Because there are four vertices of odd degree in this multigraph, there is no Euler path, so such a trip is impossible.

Necessary and sufficient conditions for Euler paths and circuits in directed graphs are given in Exercises 16 and 17.



**APPLICATIONS OF EULER PATHS AND CIRCUITS** Euler paths and circuits can be used to solve many practical problems. For example, many applications ask for a path or circuit that traverses each street in a neighborhood, each road in a transportation network, each connection in a utility grid, or each link in a communications network exactly once. Finding an Euler path or circuit in the appropriate graph model can solve such problems. For example, if a postman can find an Euler path in the graph that represents the streets the postman needs to cover, this path produces a route that traverses each street of the route exactly once. If no Euler path exists, some streets will have to be traversed more than once. The problem of finding a circuit in a graph with the fewest edges that traverses every edge at least once is known as the *Chinese postman problem* in honor of Guan Meigu, who posed it in 1962. See [MiRo91] for more information on the solution of the Chinese postman problem when no Euler path exists.

Among the other areas where Euler circuits and paths are applied is in the layout of circuits, in network multicasting, and in molecular biology, where Euler paths are used in the sequencing of DNA.

## Hamilton Paths and Circuits

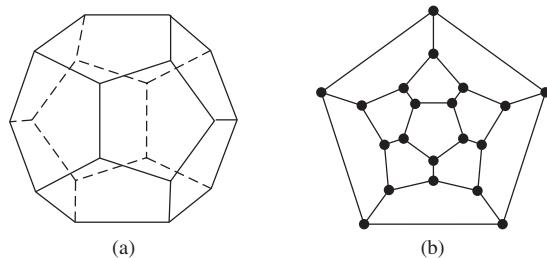


We have developed necessary and sufficient conditions for the existence of paths and circuits that contain every edge of a multigraph exactly once. Can we do the same for simple paths and circuits that contain every vertex of the graph exactly once?

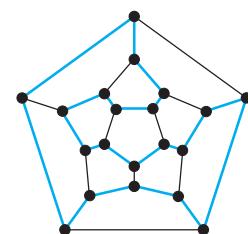
### DEFINITION 2

A simple path in a graph  $G$  that passes through every vertex exactly once is called a *Hamilton path*, and a simple circuit in a graph  $G$  that passes through every vertex exactly once is called a *Hamilton circuit*. That is, the simple path  $x_0, x_1, \dots, x_{n-1}, x_n$  in the graph  $G = (V, E)$  is a Hamilton path if  $V = \{x_0, x_1, \dots, x_{n-1}, x_n\}$  and  $x_i \neq x_j$  for  $0 \leq i < j \leq n$ , and the simple circuit  $x_0, x_1, \dots, x_{n-1}, x_n, x_0$  (with  $n > 0$ ) is a Hamilton circuit if  $x_0, x_1, \dots, x_{n-1}, x_n$  is a Hamilton path.

This terminology comes from a game, called the *Icosian puzzle*, invented in 1857 by the Irish mathematician Sir William Rowan Hamilton. It consisted of a wooden dodecahedron [a polyhedron with 12 regular pentagons as faces, as shown in Figure 8(a)], with a peg at each vertex of the dodecahedron, and string. The 20 vertices of the dodecahedron were labeled with different cities in the world. The object of the puzzle was to start at a city and travel along the edges of the dodecahedron, visiting each of the other 19 cities exactly once, and end back at the first city. The circuit traveled was marked off using the string and pegs.



## **FIGURE 8** Hamilton's “A Voyage Round the World” Puzzle.



## **FIGURE 9** A Solution to the “A Voyage Round the World” Puzzle.

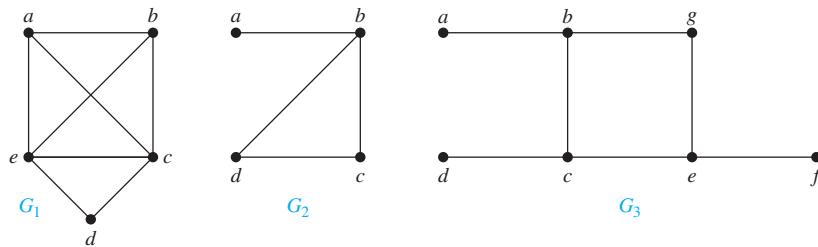
Because the author cannot supply each reader with a wooden solid with pegs and string, we will consider the equivalent question: Is there a circuit in the graph shown in Figure 8(b) that passes through each vertex exactly once? This solves the puzzle because this graph is isomorphic to the graph consisting of the vertices and edges of the dodecahedron. A solution of Hamilton's puzzle is shown in Figure 9.

## EXAMPLE 5

Which of the simple graphs in Figure 10 have a Hamilton circuit or, if not, a Hamilton path?

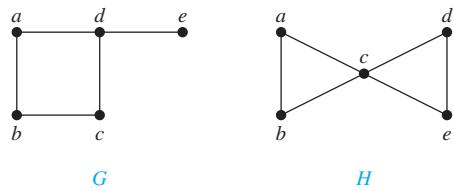


**Solution:**  $G_1$  has a Hamilton circuit:  $a, b, c, d, e, a$ . There is no Hamilton circuit in  $G_2$  (this can be seen by noting that any circuit containing every vertex must contain the edge  $\{a, b\}$  twice), but  $G_2$  does have a Hamilton path, namely,  $a, b, c, d$ .  $G_3$  has neither a Hamilton circuit nor a Hamilton path, because any path containing all vertices must contain one of the edges  $\{a, b\}$ ,  $\{e, f\}$ , and  $\{c, d\}$  more than once.



**FIGURE 10** Three Simple Graphs.

**CONDITIONS FOR THE EXISTENCE OF HAMILTON CIRCUITS** Is there a simple way to determine whether a graph has a Hamilton circuit or path? At first, it might seem that there should be an easy way to determine this, because there is a simple way to answer the similar question of whether a graph has an Euler circuit. Surprisingly, there are no known simple necessary and sufficient criteria for the existence of Hamilton circuits. However, many theorems are known that give sufficient conditions for the existence of Hamilton circuits. Also, certain properties can be used to show that a graph has no Hamilton circuit. For instance, a graph with a vertex of degree one cannot have a Hamilton circuit, because in a Hamilton circuit, each vertex is incident with two edges in the circuit. Moreover, if a vertex in the graph has degree two, then both edges that are incident with this vertex must be part of any Hamilton circuit. Also, note that when a Hamilton circuit is being constructed and this circuit has passed through a vertex, then all remaining edges incident with this vertex, other than the two used in the circuit, can be removed from consideration. Furthermore, a Hamilton circuit cannot contain a smaller circuit within it.



**FIGURE 11** Two Graphs That Do Not Have a Hamilton Circuit.

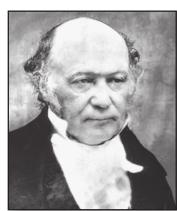
**EXAMPLE 6** Show that neither graph displayed in Figure 11 has a Hamilton circuit.

**Solution:** There is no Hamilton circuit in  $G$  because  $G$  has a vertex of degree one, namely,  $e$ . Now consider  $H$ . Because the degrees of the vertices  $a, b, d$ , and  $e$  are all two, every edge incident with these vertices must be part of any Hamilton circuit. It is now easy to see that no Hamilton circuit can exist in  $H$ , for any Hamilton circuit would have to contain four edges incident with  $c$ , which is impossible.  $\blacktriangleleft$

**EXAMPLE 7** Show that  $K_n$  has a Hamilton circuit whenever  $n \geq 3$ .

**Solution:** We can form a Hamilton circuit in  $K_n$  beginning at any vertex. Such a circuit can be built by visiting vertices in any order we choose, as long as the path begins and ends at the same vertex and visits each other vertex exactly once. This is possible because there are edges in  $K_n$  between any two vertices.  $\blacktriangleleft$

Although no useful necessary and sufficient conditions for the existence of Hamilton circuits are known, quite a few sufficient conditions have been found. Note that the more edges a graph has, the more likely it is to have a Hamilton circuit. Furthermore, adding edges (but not vertices) to a graph with a Hamilton circuit produces a graph with the same Hamilton circuit. So as we add edges to a graph, especially when we make sure to add edges to each vertex, we make it



**WILLIAM ROWAN HAMILTON (1805–1865)** William Rowan Hamilton, the most famous Irish scientist ever to have lived, was born in 1805 in Dublin. His father was a successful lawyer, his mother came from a family noted for their intelligence, and he was a child prodigy. By the age of 3 he was an excellent reader and had mastered advanced arithmetic. Because of his brilliance, he was sent off to live with his uncle James, a noted linguist. By age 8 Hamilton had learned Latin, Greek, and Hebrew; by 10 he had also learned Italian and French and he began his study of oriental languages, including Arabic, Sanskrit, and Persian. During this period he took pride in knowing as many languages as his age. At 17, no longer devoted to learning new languages and having mastered calculus and much mathematical astronomy, he began original work in optics, and he also found an important mistake in Laplace's work on celestial mechanics.

Before entering Trinity College, Dublin, at 18, Hamilton had not attended school; rather, he received private tutoring. At Trinity, he was a superior student in both the sciences and the classics. Prior to receiving his degree, because of his brilliance he was appointed the Astronomer Royal of Ireland, beating out several famous astronomers for the post. He held this position until his death, living and working at Dunsink Observatory outside of Dublin. Hamilton made important contributions to optics, abstract algebra, and dynamics. Hamilton invented algebraic objects called quaternions as an example of a noncommutative system. He discovered the appropriate way to multiply quaternions while walking along a canal in Dublin. In his excitement, he carved the formula in the stone of a bridge crossing the canal, a spot marked today by a plaque. Later, Hamilton remained obsessed with quaternions, working to apply them to other areas of mathematics, instead of moving to new areas of research.

In 1857 Hamilton invented “The Icosian Game” based on his work in noncommutative algebra. He sold the idea for 25 pounds to a dealer in games and puzzles. (Because the game never sold well, this turned out to be a bad investment for the dealer.) The “Traveler’s Dodecahedron,” also called “A Voyage Round the World,” the puzzle described in this section, is a variant of that game.

Hamilton married his third love in 1833, but his marriage worked out poorly, because his wife, a semi-invalid, was unable to cope with his household affairs. He suffered from alcoholism and lived reclusively for the last two decades of his life. He died from gout in 1865, leaving masses of papers containing unpublished research. Mixed in with these papers were a large number of dinner plates, many containing the remains of desiccated, uneaten chops.

increasingly likely that a Hamilton circuit exists in this graph. Consequently, we would expect there to be sufficient conditions for the existence of Hamilton circuits that depend on the degrees of vertices being sufficiently large. We state two of the most important sufficient conditions here. These conditions were found by Gabriel A. Dirac in 1952 and Øystein Ore in 1960.

### THEOREM 3

**DIRAC'S THEOREM** If  $G$  is a simple graph with  $n$  vertices with  $n \geq 3$  such that the degree of every vertex in  $G$  is at least  $n/2$ , then  $G$  has a Hamilton circuit.

### THEOREM 4

**ORE'S THEOREM** If  $G$  is a simple graph with  $n$  vertices with  $n \geq 3$  such that  $\deg(u) + \deg(v) \geq n$  for every pair of nonadjacent vertices  $u$  and  $v$  in  $G$ , then  $G$  has a Hamilton circuit.

The proof of Ore's theorem is outlined in Exercise 65. Dirac's theorem can be proved as a corollary to Ore's theorem because the conditions of Dirac's theorem imply those of Ore's theorem.

Both Ore's theorem and Dirac's theorem provide sufficient conditions for a connected simple graph to have a Hamilton circuit. However, these theorems do not provide necessary conditions for the existence of a Hamilton circuit. For example, the graph  $C_5$  has a Hamilton circuit but does not satisfy the hypotheses of either Ore's theorem or Dirac's theorem, as the reader can verify.



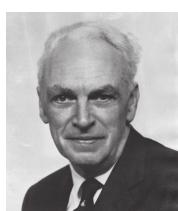
The best algorithms known for finding a Hamilton circuit in a graph or determining that no such circuit exists have exponential worst-case time complexity (in the number of vertices of the graph). Finding an algorithm that solves this problem with polynomial worst-case time



**GABRIEL ANDREW DIRAC (1925–1984)** Gabriel Dirac was born in Budapest. He moved to England in 1937 when his mother married the famous physicist and Nobel Laureate Paul Adrien Maurice Dirac, who adopted him. Gabriel A. Dirac entered Cambridge University in 1942, but his studies were interrupted by wartime service in the aviation industry. He obtained his Ph.D. in mathematics in 1951 from the University of London. He held university positions in England, Canada, Austria, Germany, and Denmark, where he spent his last 14 years. Dirac became interested in graph theory early in his career and helped raise its status as an important topic of research. He made important contributions to many aspects of graph theory, including graph coloring and Hamilton circuits. Dirac attracted many students to graph theory and was noted as an excellent lecturer.



Dirac was noted for his penetrating mind and held unconventional views on many topics, including politics and social life. Dirac was a man with many interests and held a great passion for fine art. He had a happy family life with his wife Rosemarie and his four children.



**ØYSTEIN ORE (1899–1968)** Ore was born in Kristiania (the old name for Oslo, Norway). In 1922 he received his bachelors degree and in 1925 his Ph.D. in mathematics from Kristiania University, after studies in Germany and in Sweden. In 1927 he was recruited to leave his junior position at Kristiania and join Yale University. He was promoted rapidly at Yale, becoming full professor in 1929 and Sterling Professor in 1931, a position he held until 1968.

Ore made many contributions to number theory, ring theory, lattice theory, graph theory, and probability theory. He was a prolific author of papers and books. His interest in the history of mathematics is reflected in his biographies of Abel and Cardano, and in his popular textbook *Number Theory and its History*. He wrote four books on graph theory in the 1960s.

During and after World War II Ore played a major role supporting his native Norway. In 1947 King Haakon VII of Norway gave him the Knight Order of St. Olaf to recognize these efforts. Ore possessed deep knowledge of painting and sculpture and was an ardent collector of ancient maps. He was married and had two children.

complexity would be a major accomplishment because it has been shown that this problem is NP-complete (see Section 3.3). Consequently, the existence of such an algorithm would imply that many other seemingly intractable problems could be solved using algorithms with polynomial worst-case time complexity.

## Applications of Hamilton Circuits

Hamilton paths and circuits can be used to solve practical problems. For example, many applications ask for a path or circuit that visits each road intersection in a city, each place pipelines intersect in a utility grid, or each node in a communications network exactly once. Finding a Hamilton path or circuit in the appropriate graph model can solve such problems. The famous **traveling salesperson problem** or **TSP** (also known in older literature as the **traveling salesman problem**) asks for the shortest route a traveling salesperson should take to visit a set of cities. This problem reduces to finding a Hamilton circuit in a complete graph such that the total weight of its edges is as small as possible. We will return to this question in Section 10.6.

We now describe a less obvious application of Hamilton circuits to coding.

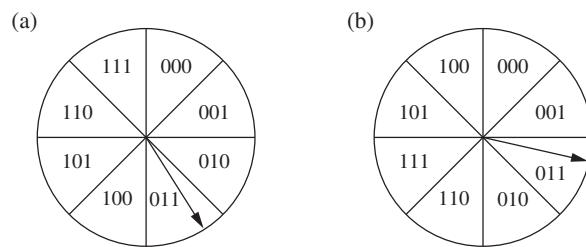
**EXAMPLE 8 Gray Codes** The position of a rotating pointer can be represented in digital form. One way to do this is to split the circle into  $2^n$  arcs of equal length and to assign a bit string of length  $n$  to each arc. Two ways to do this using bit strings of length three are shown in Figure 12.

The digital representation of the position of the pointer can be determined using a set of  $n$  contacts. Each contact is used to read one bit in the digital representation of the position. This is illustrated in Figure 13 for the two assignments from Figure 12.

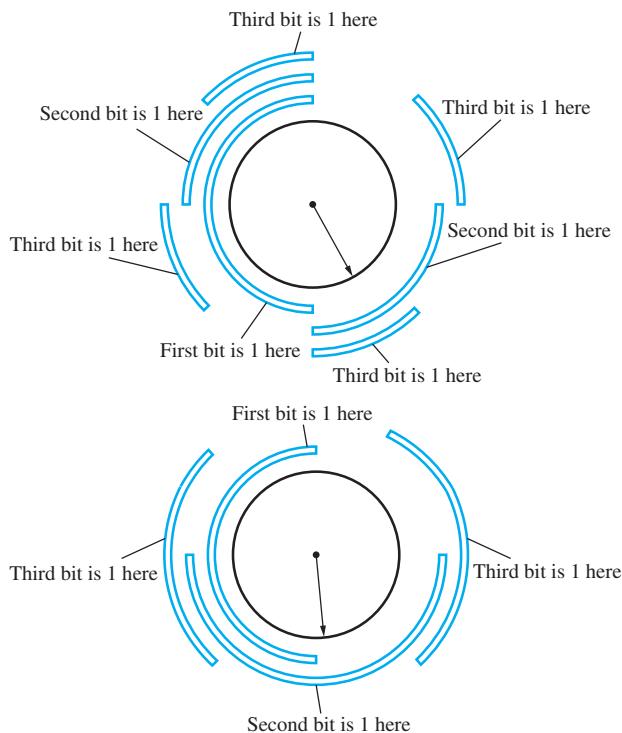
When the pointer is near the boundary of two arcs, a mistake may be made in reading its position. This may result in a major error in the bit string read. For instance, in the coding scheme in Figure 12(a), if a small error is made in determining the position of the pointer, the bit string 100 is read instead of 011. All three bits are incorrect! To minimize the effect of an error in determining the position of the pointer, the assignment of the bit strings to the  $2^n$  arcs should be made so that only one bit is different in the bit strings represented by adjacent arcs. This is exactly the situation in the coding scheme in Figure 12(b). An error in determining the position of the pointer gives the bit string 010 instead of 011. Only one bit is wrong.



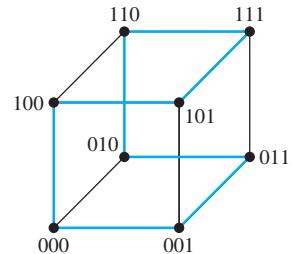
A **Gray code** is a labeling of the arcs of the circle such that adjacent arcs are labeled with bit strings that differ in exactly one bit. The assignment in Figure 12(b) is a Gray code. We can find a Gray code by listing all bit strings of length  $n$  in such a way that each string differs in exactly one position from the preceding bit string, and the last string differs from the first in exactly one position. We can model this problem using the  $n$ -cube  $Q_n$ . What is needed to solve this problem is a Hamilton circuit in  $Q_n$ . Such Hamilton circuits are easily found. For instance, a Hamilton



**FIGURE 12** Converting the Position of a Pointer into Digital Form.



**FIGURE 13** The Digital Representation of the Position of the Pointer.



**FIGURE 14** A Hamilton Circuit for  $Q_3$ .

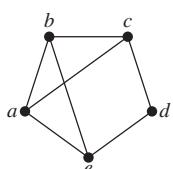
circuit for  $Q_3$  is displayed in Figure 14. The sequence of bit strings differing in exactly one bit produced by this Hamilton circuit is 000, 001, 011, 010, 110, 111, 101, 100.

Gray codes are named after Frank Gray, who invented them in the 1940s at AT&T Bell Laboratories to minimize the effect of errors in transmitting digital signals. 

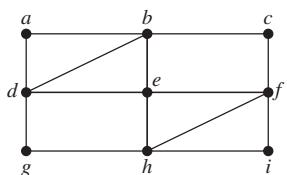
## Exercises

In Exercises 1–8 determine whether the given graph has an Euler circuit. Construct such a circuit when one exists. If no Euler circuit exists, determine whether the graph has an Euler path and construct such a path if one exists.

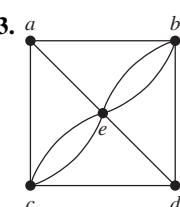
1.



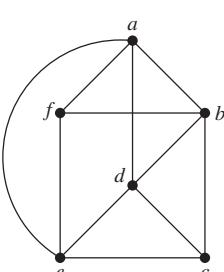
2.



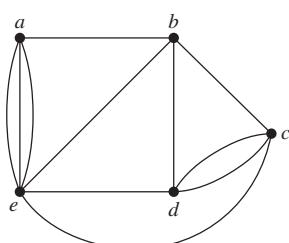
3.



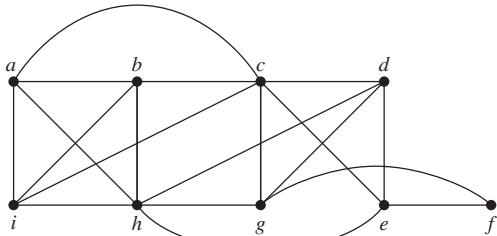
4.



5.



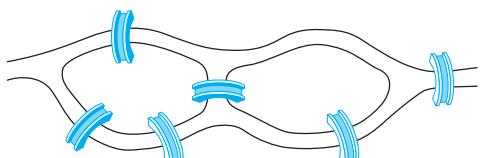
- The graph consists of 10 nodes labeled  $a$  through  $j$ . Node  $a$  is at the bottom left, node  $b$  is above it, node  $c$  is to its right, node  $d$  is further to the right, and node  $e$  is at the far right. Node  $i$  is located in the center. Node  $h$  is below node  $b$ , node  $g$  is below node  $c$ , and node  $j$  is below node  $d$ . Edges connect  $a$  to  $b$ ,  $a$  to  $c$ ,  $a$  to  $i$ ,  $b$  to  $c$ ,  $b$  to  $i$ ,  $c$  to  $d$ ,  $c$  to  $i$ ,  $c$  to  $g$ ,  $d$  to  $i$ ,  $d$  to  $j$ ,  $g$  to  $j$ , and  $i$  to  $j$ .



8.

9. Suppose that in addition to the seven bridges of Königsberg (shown in Figure 1) there were two additional bridges, connecting regions  $B$  and  $C$  and regions  $B$  and  $D$ , respectively. Could someone cross all nine of these bridges exactly once and return to the starting point?

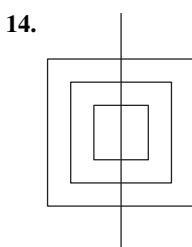
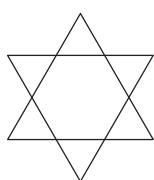
**10.** Can someone cross all the bridges shown in this map exactly once and return to the starting point?



- When can the centerlines of the streets in a city be painted without traveling a street more than once? (Assume that all the streets are two-way streets.)
  - Devise a procedure, similar to Algorithm 1, for constructing Euler paths in multigraphs.

In Exercises 13–15 determine whether the picture shown can be drawn with a pencil in a continuous motion without lifting the pencil or retracing part of the picture.

13. ▲



15.

- \*16. Show that a directed multigraph having no isolated vertices has an Euler circuit if and only if the graph is weakly connected and the in-degree and out-degree of each vertex are equal.

- \* 17. Show that a directed multigraph having no isolated vertices has an Euler path but not an Euler circuit if and only if the graph is weakly connected and the in-degree and out-degree of each vertex are equal for all but two vertices, one that has in-degree one larger than its out-degree and the other that has out-degree one larger than its in-degree.

In Exercises 18–23 determine whether the directed graph shown has an Euler circuit. Construct an Euler circuit if one exists. If no Euler circuit exists, determine whether the directed graph has an Euler path. Construct an Euler path if one exists.

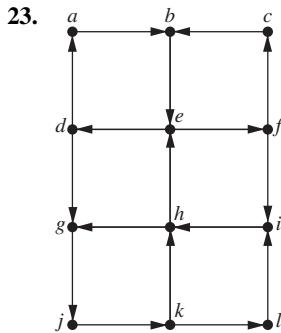
18.  A directed graph with four nodes labeled  $a$ ,  $b$ ,  $c$ , and  $d$ . The edges are directed as follows:  $a \rightarrow b$ ,  $a \rightarrow c$ ,  $a \rightarrow d$ ,  $b \rightarrow c$ ,  $b \rightarrow d$ ,  $c \rightarrow d$ , and  $d \rightarrow c$ . There are two self-loops on node  $b$ .

19.  A directed graph with four nodes labeled  $a$ ,  $b$ ,  $c$ , and  $d$ . The edges are directed as follows:  $a \rightarrow b$ ,  $a \rightarrow c$ ,  $a \rightarrow d$ ,  $b \rightarrow c$ ,  $b \rightarrow d$ ,  $c \rightarrow d$ , and  $d \rightarrow c$ . There are also two self-loops on node  $a$ .

20.

21.

22.



- \* 24. Devise an algorithm for constructing Euler circuits in directed graphs.

25. Devise an algorithm for constructing Euler paths in directed graphs.

26. For which values of  $n$  do these graphs have an Euler circuit?

a)  $K_n$       b)  $C_n$       c)  $W_n$       d)  $Q_n$

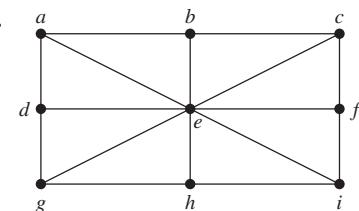
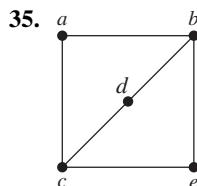
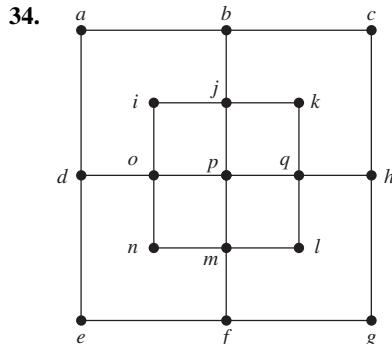
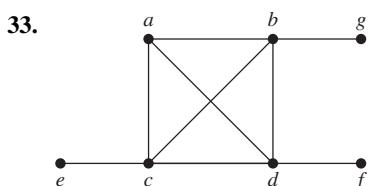
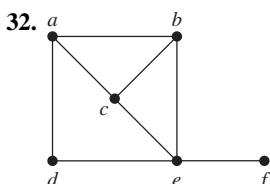
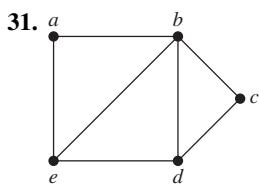
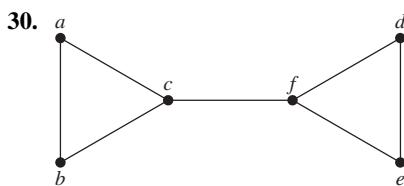
27. For which values of  $n$  do the graphs in Exercise 26 have an Euler path but no Euler circuit?

28. For which values of  $m$  and  $n$  does the complete bipartite graph  $K_{m,n}$  have an

a) Euler circuit?  
b) Euler path?

29. Find the least number of times it is necessary to lift a pencil from the paper when drawing each of the graphs in Exercises 1–7 without retracing any part of the graph.

In Exercises 30–36 determine whether the given graph has a Hamilton circuit. If it does, find such a circuit. If it does not, give an argument to show why no such circuit exists.



37. Does the graph in Exercise 30 have a Hamilton path? If so, find such a path. If it does not, give an argument to show why no such path exists.

38. Does the graph in Exercise 31 have a Hamilton path? If so, find such a path. If it does not, give an argument to show why no such path exists.

39. Does the graph in Exercise 32 have a Hamilton path? If so, find such a path. If it does not, give an argument to show why no such path exists.

40. Does the graph in Exercise 33 have a Hamilton path? If so, find such a path. If it does not, give an argument to show why no such path exists.

\*41. Does the graph in Exercise 34 have a Hamilton path? If so, find such a path. If it does not, give an argument to show why no such path exists.

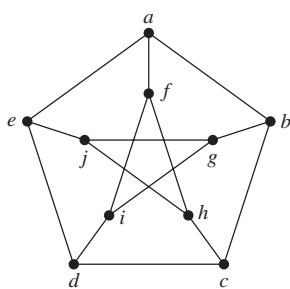
42. Does the graph in Exercise 35 have a Hamilton path? If so, find such a path. If it does not, give an argument to show why no such path exists.

43. Does the graph in Exercise 36 have a Hamilton path? If so, find such a path. If it does not, give an argument to show why no such path exists.

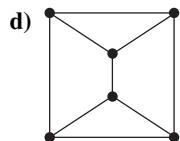
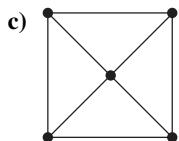
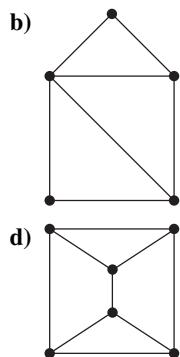
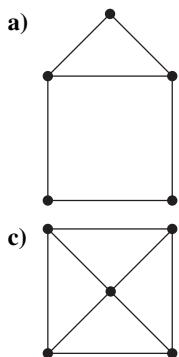
44. For which values of  $n$  do the graphs in Exercise 26 have a Hamilton circuit?

45. For which values of  $m$  and  $n$  does the complete bipartite graph  $K_{m,n}$  have a Hamilton circuit?

- \*46. Show that the **Petersen graph**, shown here, does not have a Hamilton circuit, but that the subgraph obtained by deleting a vertex  $v$ , and all edges incident with  $v$ , does have a Hamilton circuit.



47. For each of these graphs, determine (i) whether Dirac's theorem can be used to show that the graph has a Hamilton circuit, (ii) whether Ore's theorem can be used to show that the graph has a Hamilton circuit, and (iii) whether the graph has a Hamilton circuit.



**JULIUS PETER CHRISTIAN PETERSEN (1839–1910)** Julius Petersen was born in the Danish town of Sorø. His father was a dyer. In 1854 his parents were no longer able to pay for his schooling, so he became an apprentice in an uncle's grocery store. When this uncle died, he left Petersen enough money to return to school. After graduating, he began studying engineering at the Polytechnical School in Copenhagen, later deciding to concentrate on mathematics. He published his first textbook, a book on logarithms, in 1858. When his inheritance ran out, he had to teach to make a living. From 1859 until 1871 Petersen taught at a prestigious private high school in Copenhagen. While teaching high school he continued his studies, entering Copenhagen University in 1862. He married Laura Bertelsen in 1862; they had three children, two sons and a daughter.

Petersen obtained a mathematics degree from Copenhagen University in 1866 and finally obtained his doctorate in 1871 from that school. After receiving his doctorate, he taught at a polytechnic and military academy. In 1887 he was appointed to a professorship at the University of Copenhagen. Petersen was well known in Denmark as the author of a large series of textbooks for high schools and universities. One of his books, *Methods and Theories for the Solution of Problems of Geometrical Construction*, was translated into eight languages, with the English language version last reprinted in 1960 and the French version reprinted as recently as 1990, more than a century after the original publication date.

Petersen worked in a wide range of areas, including algebra, analysis, cryptography, geometry, mechanics, mathematical economics, and number theory. His contributions to graph theory, including results on regular graphs, are his best-known work. He was noted for his clarity of exposition, problem-solving skills, originality, sense of humor, vigor, and teaching. One interesting fact about Petersen was that he preferred not to read the writings of other mathematicians. This led him often to rediscover results already proved by others, often with embarrassing consequences. However, he was often angry when other mathematicians did not read his writings!

Petersen's death was front-page news in Copenhagen. A newspaper of the time described him as the Hans Christian Andersen of science—a child of the people who made good in the academic world.

48. Can you find a simple graph with  $n$  vertices with  $n \geq 3$  that does not have a Hamilton circuit, yet the degree of every vertex in the graph is at least  $(n - 1)/2$ ?

- \*49. Show that there is a Gray code of order  $n$  whenever  $n$  is a positive integer, or equivalently, show that the  $n$ -cube  $Q_n$ ,  $n > 1$ , always has a Hamilton circuit. [Hint: Use mathematical induction. Show how to produce a Gray code of order  $n$  from one of order  $n - 1$ .]

**Fleury's algorithm**, published in 1883, constructs Euler circuits by first choosing an arbitrary vertex of a connected multigraph, and then forming a circuit by choosing edges successively. Once an edge is chosen, it is removed. Edges are chosen successively so that each edge begins where the last edge ends, and so that this edge is not a cut edge unless there is no alternative.

50. Use Fleury's algorithm to find an Euler circuit in the graph  $G$  in Figure 5.

- \*51. Express Fleury's algorithm in pseudocode.

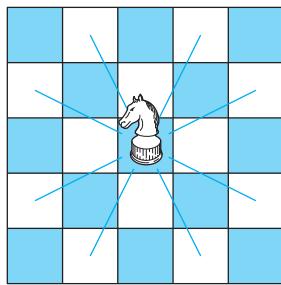
- \*\*52. Prove that Fleury's algorithm always produces an Euler circuit.

- \*53. Give a variant of Fleury's algorithm to produce Euler paths.

54. A diagnostic message can be sent out over a computer network to perform tests over all links and in all devices. What sort of paths should be used to test all links? To test all devices?

55. Show that a bipartite graph with an odd number of vertices does not have a Hamilton circuit.

A **knight** is a chess piece that can move either two spaces horizontally and one space vertically or one space horizontally and two spaces vertically. That is, a knight on square  $(x, y)$  can move to any of the eight squares  $(x \pm 2, y \pm 1)$ ,  $(x \pm 1, y \pm 2)$ , if these squares are on the chessboard, as illustrated here.



A **knight's tour** is a sequence of legal moves by a knight starting at some square and visiting each square exactly once. A knight's tour is called **reentrant** if there is a legal move that takes the knight from the last square of the tour back to where the tour began. We can model knight's tours using the graph that has a vertex for each square on the board, with an edge connecting two vertices if a knight can legally move between the squares represented by these vertices.

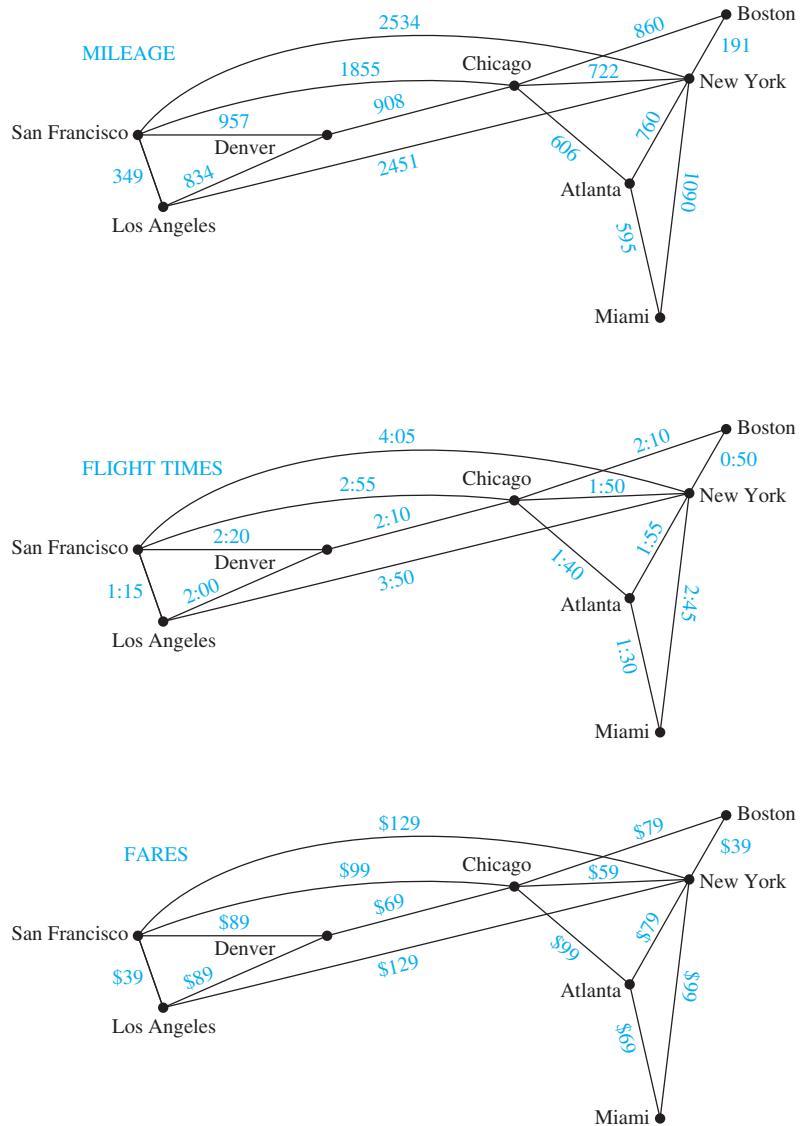
- 56. Draw the graph that represents the legal moves of a knight on a  $3 \times 3$  chessboard.
- 57. Draw the graph that represents the legal moves of a knight on a  $3 \times 4$  chessboard.
- 58. a) Show that finding a knight's tour on an  $m \times n$  chessboard is equivalent to finding a Hamilton path on the graph representing the legal moves of a knight on that board.  
b) Show that finding a reentrant knight's tour on an  $m \times n$  chessboard is equivalent to finding a Hamilton circuit on the corresponding graph.
- \*59. Show that there is a knight's tour on a  $3 \times 4$  chessboard.
- \*60. Show that there is no knight's tour on a  $3 \times 3$  chessboard.
- \*61. Show that there is no knight's tour on a  $4 \times 4$  chessboard.

- 62. Show that the graph representing the legal moves of a knight on an  $m \times n$  chessboard, whenever  $m$  and  $n$  are positive integers, is bipartite.
- 63. Show that there is no reentrant knight's tour on an  $m \times n$  chessboard when  $m$  and  $n$  are both odd. [Hint: Use Exercises 55, 58b, and 62.]
- \*64. Show that there is a knight's tour on an  $8 \times 8$  chessboard. [Hint: You can construct a knight's tour using a method invented by H. C. Wrensdorff in 1823: Start in any square, and then always move to a square connected to the fewest number of unused squares. Although this method may not always produce a knight's tour, it often does.]
- 65. The parts of this exercise outline a proof of Ore's theorem. Suppose that  $G$  is a simple graph with  $n$  vertices,  $n \geq 3$ , and  $\deg(x) + \deg(y) \geq n$  whenever  $x$  and  $y$  are nonadjacent vertices in  $G$ . Ore's theorem states that under these conditions,  $G$  has a Hamilton circuit.
  - a) Show that if  $G$  does not have a Hamilton circuit, then there exists another graph  $H$  with the same vertices as  $G$ , which can be constructed by adding edges to  $G$  such that the addition of a single edge would produce a Hamilton circuit in  $H$ . [Hint: Add as many edges as possible at each successive vertex of  $G$  without producing a Hamilton circuit.]
  - b) Show that there is a Hamilton path in  $H$ .
  - c) Let  $v_1, v_2, \dots, v_n$  be a Hamilton path in  $H$ . Show that  $\deg(v_1) + \deg(v_n) \geq n$  and that there are at most  $\deg(v_1)$  vertices not adjacent to  $v_n$  (including  $v_n$  itself).
  - d) Let  $S$  be the set of vertices preceding each vertex adjacent to  $v_1$  in the Hamilton path. Show that  $S$  contains  $\deg(v_1)$  vertices and  $v_n \notin S$ .
  - e) Show that  $S$  contains a vertex  $v_k$ , which is adjacent to  $v_n$ , implying that there are edges connecting  $v_1$  and  $v_{k+1}$  and  $v_k$  and  $v_n$ .
  - f) Show that part (e) implies that  $v_1, v_2, \dots, v_{k-1}, v_k, v_n, v_{n-1}, \dots, v_{k+1}, v_1$  is a Hamilton circuit in  $G$ . Conclude from this contradiction that Ore's theorem holds.
- \*66. Show that the worst case computational complexity of Algorithm 1 for finding Euler circuits in a connected graph with all vertices of even degree is  $O(m)$ , where  $m$  is the number of edges of  $G$ .

## 10.6 Shortest-Path Problems

### Introduction

Many problems can be modeled using graphs with weights assigned to their edges. As an illustration, consider how an airline system can be modeled. We set up the basic graph model by representing cities by vertices and flights by edges. Problems involving distances can be modeled by assigning distances between cities to the edges. Problems involving flight time can be modeled by assigning flight times to edges. Problems involving fares can be modeled by

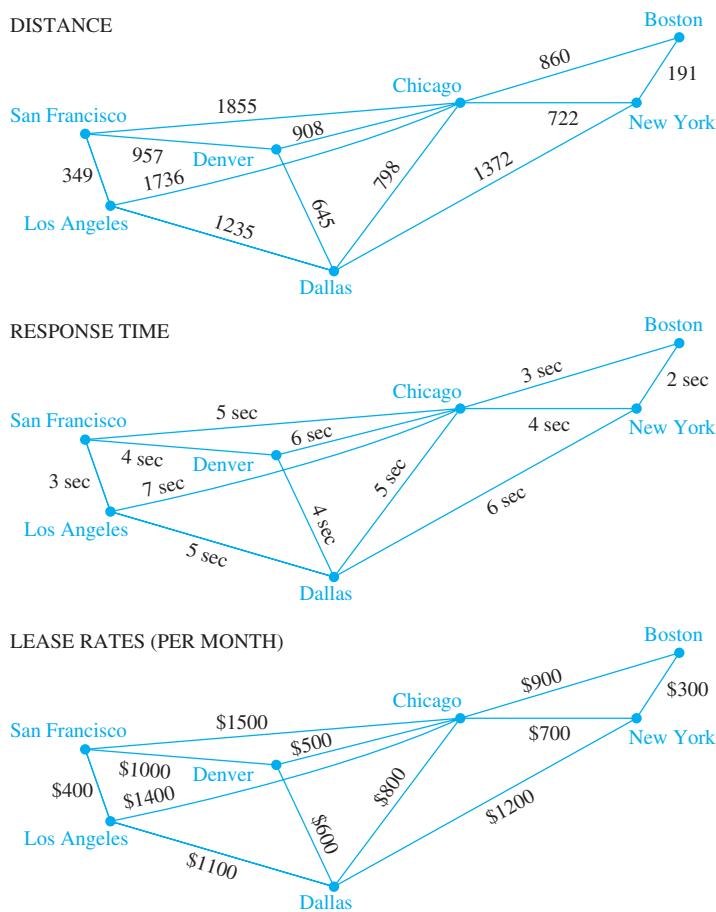


**FIGURE 1** Weighted Graphs Modeling an Airline System.

assigning fares to the edges. Figure 1 displays three different assignments of weights to the edges of a graph representing distances, flight times, and fares, respectively.

Graphs that have a number assigned to each edge are called **weighted graphs**. Weighted graphs are used to model computer networks. Communications costs (such as the monthly cost of leasing a telephone line), the response times of the computers over these lines, or the distance between computers, can all be studied using weighted graphs. Figure 2 displays weighted graphs that represent three ways to assign weights to the edges of a graph of a computer network, corresponding to distance, response time, and cost.

Several types of problems involving weighted graphs arise frequently. Determining a path of least length between two vertices in a network is one such problem. To be more specific, let the **length** of a path in a weighted graph be the sum of the weights of the edges of this path. (The reader should note that this use of the term *length* is different from the use of *length* to denote the number of edges in a path in a graph without weights.) The question is: What is a shortest path, that is, a path of least length, between two given vertices? For instance, in the airline system



**FIGURE 2** Weighted Graphs Modeling a Computer Network.

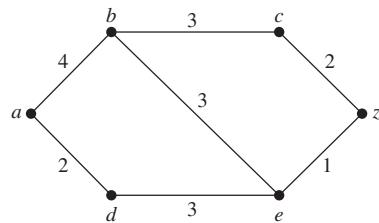
represented by the weighted graph shown in Figure 1, what is a shortest path in air distance between Boston and Los Angeles? What combinations of flights has the smallest total flight time (that is, total time in the air, not including time between flights) between Boston and Los Angeles? What is the cheapest fare between these two cities? In the computer network shown in Figure 2, what is a least expensive set of telephone lines needed to connect the computers in San Francisco with those in New York? Which set of telephone lines gives a fastest response time for communications between San Francisco and New York? Which set of lines has a shortest overall distance?

Another important problem involving weighted graphs asks for a circuit of shortest total length that visits every vertex of a complete graph exactly once. This is the famous *traveling salesperson problem*, which asks for an order in which a salesperson should visit each of the cities on his route exactly once so that he travels the minimum total distance. We will discuss the traveling salesperson problem later in this section.

## A Shortest-Path Algorithm



There are several different algorithms that find a shortest path between two vertices in a weighted graph. We will present a greedy algorithm discovered by the Dutch mathematician Edsger Di-



**FIGURE 3** A Weighted Simple Graph.

Jkstra in 1959. The version we will describe solves this problem in undirected weighted graphs where all the weights are positive. It is easy to adapt it to solve shortest-path problems in directed graphs.

Before giving a formal presentation of the algorithm, we will give an illustrative example.

**EXAMPLE 1** What is the length of a shortest path between  $a$  and  $z$  in the weighted graph shown in Figure 3?

**Solution:** Although a shortest path is easily found by inspection, we will develop some ideas useful in understanding Dijkstra's algorithm. We will solve this problem by finding the length of a shortest path from  $a$  to successive vertices, until  $z$  is reached.

The only paths starting at  $a$  that contain no vertex other than  $a$  are formed by adding an edge that has  $a$  as one endpoint. These paths have only one edge. They are  $a, b$  of length 4 and  $a, d$  of length 2. It follows that  $d$  is the closest vertex to  $a$ , and the shortest path from  $a$  to  $d$  has length 2.

We can find the second closest vertex by examining all paths that begin with the shortest path from  $a$  to a vertex in the set  $\{a, d\}$ , followed by an edge that has one endpoint in  $\{a, d\}$  and its other endpoint not in this set. There are two such paths to consider,  $a, d, e$  of length 7 and  $a, b$  of length 4. Hence, the second closest vertex to  $a$  is  $b$  and the shortest path from  $a$  to  $b$  has length 4.

To find the third closest vertex to  $a$ , we need examine only the paths that begin with the shortest path from  $a$  to a vertex in the set  $\{a, d, b\}$ , followed by an edge that has one endpoint in the set  $\{a, d, b\}$  and its other endpoint not in this set. There are three such paths,  $a, b, c$  of length 7,  $a, b, e$  of length 7, and  $a, d, e$  of length 5. Because the shortest of these paths is  $a, d, e$ , the third closest vertex to  $a$  is  $e$  and the length of the shortest path from  $a$  to  $e$  is 5.



**EDSGER WYBE DIJKSTRA (1930–2002)** Edsger Dijkstra, born in the Netherlands, began programming computers in the early 1950s while studying theoretical physics at the University of Leiden. In 1952, realizing that he was more interested in programming than in physics, he quickly completed the requirements for his physics degree and began his career as a programmer, even though programming was not a recognized profession. (In 1957, the authorities in Amsterdam refused to accept “programming” as his profession on his marriage license. However, they did accept “theoretical physicist” when he changed his entry to this.)

Dijkstra was one of the most forceful proponents of programming as a scientific discipline. He has made fundamental contributions to the areas of operating systems, including deadlock avoidance; programming languages, including the notion of structured programming; and algorithms. In 1972 Dijkstra received the Turing Award from the Association for Computing Machinery, one of the most prestigious awards in computer science. Dijkstra became a Burroughs Research Fellow in 1973, and in 1984 he was appointed to a chair in Computer Science at the University of Texas, Austin.

To find the fourth closest vertex to  $a$ , we need examine only the paths that begin with the shortest path from  $a$  to a vertex in the set  $\{a, d, b, e\}$ , followed by an edge that has one endpoint in the set  $\{a, d, b, e\}$  and its other endpoint not in this set. There are two such paths,  $a, b, c$  of length 7 and  $a, d, e, z$  of length 6. Because the shorter of these paths is  $a, d, e, z$ , the fourth closest vertex to  $a$  is  $z$  and the length of the shortest path from  $a$  to  $z$  is 6. 

Example 1 illustrates the general principles used in Dijkstra's algorithm. Note that a shortest path from  $a$  to  $z$  could have been found by a brute force approach by examining the length of every path from  $a$  to  $z$ . However, this brute force approach is impractical for humans and even for computers for graphs with a large number of edges.

We will now consider the general problem of finding the length of a shortest path between  $a$  and  $z$  in an undirected connected simple weighted graph. Dijkstra's algorithm proceeds by finding the length of a shortest path from  $a$  to a first vertex, the length of a shortest path from  $a$  to a second vertex, and so on, until the length of a shortest path from  $a$  to  $z$  is found. As a side benefit, this algorithm is easily extended to find the length of the shortest path from  $a$  to all other vertices of the graph, and not just to  $z$ .

The algorithm relies on a series of iterations. A distinguished set of vertices is constructed by adding one vertex at each iteration. A labeling procedure is carried out at each iteration. In this labeling procedure, a vertex  $w$  is labeled with the length of a shortest path from  $a$  to  $w$  that contains only vertices already in the distinguished set. The vertex added to the distinguished set is one with a minimal label among those vertices not already in the set.

We now give the details of Dijkstra's algorithm. It begins by labeling  $a$  with 0 and the other vertices with  $\infty$ . We use the notation  $L_0(a) = 0$  and  $L_0(v) = \infty$  for these labels before any iterations have taken place (the subscript 0 stands for the “0th” iteration). These labels are the lengths of shortest paths from  $a$  to the vertices, where the paths contain only the vertex  $a$ . (Because no path from  $a$  to a vertex different from  $a$  exists,  $\infty$  is the length of a shortest path between  $a$  and this vertex.)

Dijkstra's algorithm proceeds by forming a distinguished set of vertices. Let  $S_k$  denote this set after  $k$  iterations of the labeling procedure. We begin with  $S_0 = \emptyset$ . The set  $S_k$  is formed from  $S_{k-1}$  by adding a vertex  $u$  not in  $S_{k-1}$  with the smallest label.

Once  $u$  is added to  $S_k$ , we update the labels of all vertices not in  $S_k$ , so that  $L_k(v)$ , the label of the vertex  $v$  at the  $k$ th stage, is the length of a shortest path from  $a$  to  $v$  that contains vertices only in  $S_k$  (that is, vertices that were already in the distinguished set together with  $u$ ). Note that the way we choose the vertex  $u$  to add to  $S_k$  at each step is an optimal choice at each step, making this a greedy algorithm. (We will prove shortly that this greedy algorithm always produces an optimal solution.)

Let  $v$  be a vertex not in  $S_k$ . To update the label of  $v$ , note that  $L_k(v)$  is the length of a shortest path from  $a$  to  $v$  containing only vertices in  $S_k$ . The updating can be carried out efficiently when this observation is used: A shortest path from  $a$  to  $v$  containing only elements of  $S_k$  is either a shortest path from  $a$  to  $v$  that contains only elements of  $S_{k-1}$  (that is, the distinguished vertices not including  $u$ ), or it is a shortest path from  $a$  to  $u$  at the  $(k-1)$ st stage with the edge  $\{u, v\}$  added. In other words,

$$L_k(a, v) = \min\{L_{k-1}(a, v), L_{k-1}(a, u) + w(u, v)\},$$

where  $w(u, v)$  is the length of the edge with  $u$  and  $v$  as endpoints. This procedure is iterated by successively adding vertices to the distinguished set until  $z$  is added. When  $z$  is added to the distinguished set, its label is the length of a shortest path from  $a$  to  $z$ .

Dijkstra's algorithm is given in Algorithm 1. Later we will give a proof that this algorithm is correct. Note that we can find the length of the shortest path from  $a$  to all other vertices of the graph if we continue this procedure until all vertices are added to the distinguished set.



**ALGORITHM 1 Dijkstra's Algorithm.**

```

procedure Dijkstra(G: weighted connected simple graph, with
 all weights positive)
{G has vertices $a = v_0, v_1, \dots, v_n = z$ and lengths $w(v_i, v_j)$
 where $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge in G}
for $i := 1$ to n
 $L(v_i) := \infty$
 $L(a) := 0$
 $S := \emptyset$
{the labels are now initialized so that the label of a is 0 and all
 other labels are ∞ , and S is the empty set}
while $z \notin S$
 $u :=$ a vertex not in S with $L(u)$ minimal
 $S := S \cup \{u\}$
 for all vertices v not in S
 if $L(u) + w(u, v) < L(v)$ then $L(v) := L(u) + w(u, v)$
 {this adds a vertex to S with minimal label and updates the
 labels of vertices not in S }
return $L(z)$ { $L(z)$ = length of a shortest path from a to z }

```

Example 2 illustrates how Dijkstra's algorithm works. Afterward, we will show that this algorithm always produces the length of a shortest path between two vertices in a weighted graph.

**EXAMPLE 2** Use Dijkstra's algorithm to find the length of a shortest path between the vertices  $a$  and  $z$  in the weighted graph displayed in Figure 4(a).

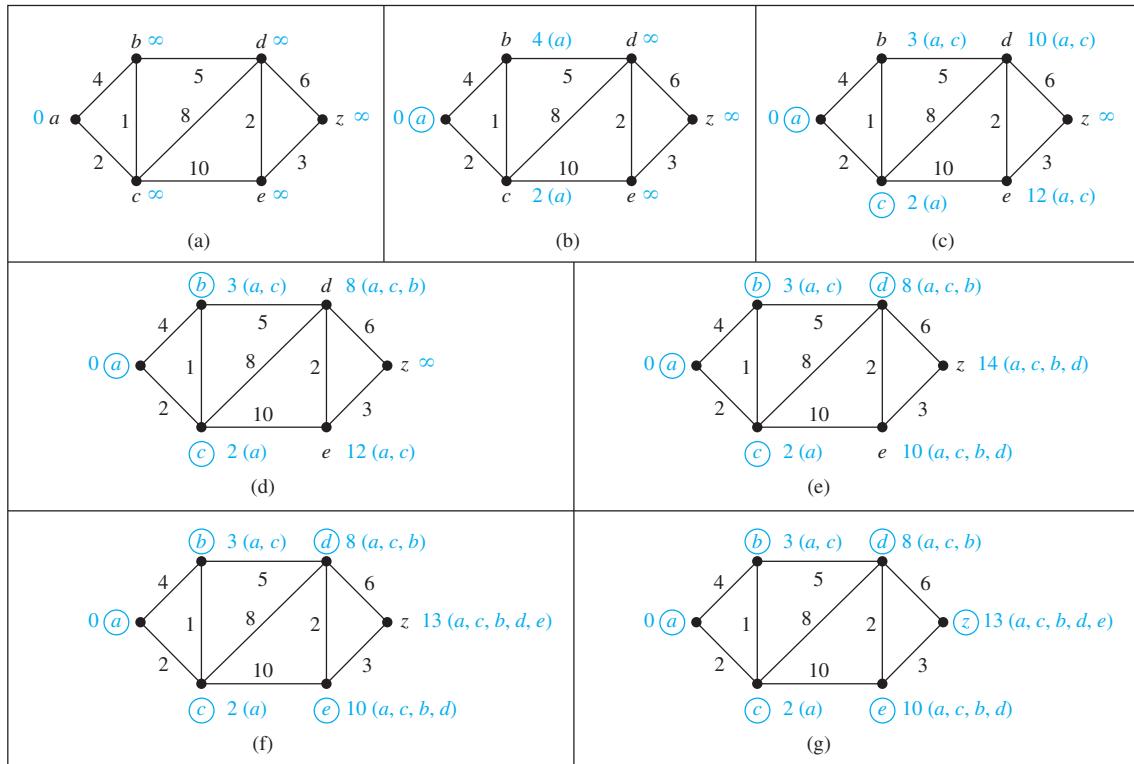
**Solution:** The steps used by Dijkstra's algorithm to find a shortest path between  $a$  and  $z$  are shown in Figure 4. At each iteration of the algorithm the vertices of the set  $S_k$  are circled. A shortest path from  $a$  to each vertex containing only vertices in  $S_k$  is indicated for each iteration. The algorithm terminates when  $z$  is circled. We find that a shortest path from  $a$  to  $z$  is  $a, c, b, d, e, z$ , with length 13. 

**Remark:** In performing Dijkstra's algorithm it is sometimes more convenient to keep track of labels of vertices in each step using a table instead of redrawing the graph for each step.

Next, we use an inductive argument to show that Dijkstra's algorithm produces the length of a shortest path between two vertices  $a$  and  $z$  in an undirected connected weighted graph. Take as the inductive hypothesis the following assertion: At the  $k$ th iteration

- (i) the label of every vertex  $v$  in  $S$  is the length of a shortest path from  $a$  to this vertex, and
- (ii) the label of every vertex not in  $S$  is the length of a shortest path from  $a$  to this vertex that contains only (besides the vertex itself) vertices in  $S$ .

When  $k = 0$ , before any iterations are carried out,  $S = \emptyset$ , so the length of a shortest path from  $a$  to a vertex other than  $a$  is  $\infty$ . Hence, the basis case is true.



**FIGURE 4** Using Dijkstra’s Algorithm to Find a Shortest Path from  $a$  to  $z$ .



Assume that the inductive hypothesis holds for the  $k$ th iteration. Let  $v$  be the vertex added to  $S$  at the  $(k+1)$ st iteration, so  $v$  is a vertex not in  $S$  at the end of the  $k$ th iteration with the smallest label (in the case of ties, any vertex with smallest label may be used).

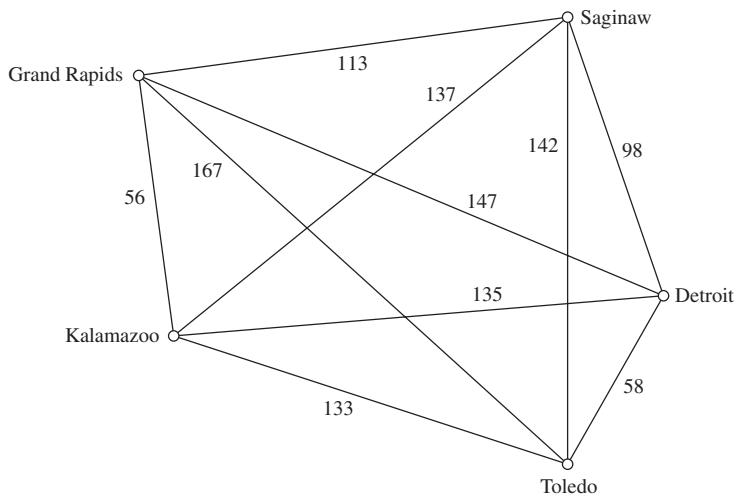
From the inductive hypothesis we see that the vertices in  $S$  before the  $(k+1)$ st iteration are labeled with the length of a shortest path from  $a$ . Also,  $v$  must be labeled with the length of a shortest path to it from  $a$ . If this were not the case, at the end of the  $k$ th iteration there would be a path of length less than  $L_k(v)$  containing a vertex not in  $S$  [because  $L_k(v)$  is the length of a shortest path from  $a$  to  $v$  containing only vertices in  $S$  after the  $k$ th iteration]. Let  $u$  be the first vertex not in  $S$  in such a path. There is a path with length less than  $L_k(v)$  from  $a$  to  $u$  containing only vertices of  $S$ . This contradicts the choice of  $v$ . Hence, (i) holds at the end of the  $(k+1)$ st iteration.

Let  $u$  be a vertex not in  $S$  after  $k+1$  iterations. A shortest path from  $a$  to  $u$  containing only elements of  $S$  either contains  $v$  or it does not. If it does not contain  $v$ , then by the inductive hypothesis its length is  $L_k(u)$ . If it does contain  $v$ , then it must be made up of a path from  $a$  to  $v$  of shortest possible length containing elements of  $S$  other than  $v$ , followed by the edge from  $v$  to  $u$ . In this case, its length would be  $L_k(v) + w(v, u)$ . This shows that (ii) is true, because  $L_{k+1}(u) = \min\{L_k(u), L_k(v) + w(v, u)\}$ .

We now state the theorem that we have proved.

### THEOREM 1

Dijkstra’s algorithm finds the length of a shortest path between two vertices in a connected simple undirected weighted graph.



**FIGURE 5** The Graph Showing the Distances between Five Cities.

We can now estimate the computational complexity of Dijkstra's algorithm (in terms of additions and comparisons). The algorithm uses no more than  $n - 1$  iterations where  $n$  is the number of vertices in the graph, because one vertex is added to the distinguished set at each iteration. We are done if we can estimate the number of operations used for each iteration. We can identify the vertex not in  $S_k$  with the smallest label using no more than  $n - 1$  comparisons. Then we use an addition and a comparison to update the label of each vertex not in  $S_k$ . It follows that no more than  $2(n - 1)$  operations are used at each iteration, because there are no more than  $n - 1$  labels to update at each iteration. Because we use no more than  $n - 1$  iterations, each using no more than  $2(n - 1)$  operations, we have Theorem 2.

### THEOREM 2

Dijkstra's algorithm uses  $O(n^2)$  operations (additions and comparisons) to find the length of a shortest path between two vertices in a connected simple undirected weighted graph with  $n$  vertices.

## The Traveling Salesperson Problem



We now discuss an important problem involving weighted graphs. Consider the following problem: A traveling salesperson wants to visit each of  $n$  cities exactly once and return to his starting point. For example, suppose that the salesperson wants to visit Detroit, Toledo, Saginaw, Grand Rapids, and Kalamazoo (see Figure 5). In which order should he visit these cities to travel the minimum total distance? To solve this problem we can assume the salesperson starts in Detroit (because this must be part of the circuit) and examine all possible ways for him to visit the other four cities and then return to Detroit (starting elsewhere will produce the same circuits). There are a total of 24 such circuits, but because we travel the same distance when we travel a circuit in reverse order, we need only consider 12 different circuits to find the minimum total distance he must travel. We list these 12 different circuits and the total distance traveled for each circuit. As can be seen from the list, the minimum total distance of 458 miles is traveled using the circuit Detroit–Toledo–Kalamazoo–Grand Rapids–Saginaw–Detroit (or its reverse).

| <i>Route</i>                                          | <i>Total Distance (miles)</i> |
|-------------------------------------------------------|-------------------------------|
| Detroit–Toledo–Grand Rapids–Saginaw–Kalamazoo–Detroit | 610                           |
| Detroit–Toledo–Grand Rapids–Kalamazoo–Saginaw–Detroit | 516                           |
| Detroit–Toledo–Kalamazoo–Saginaw–Grand Rapids–Detroit | 588                           |
| Detroit–Toledo–Kalamazoo–Grand Rapids–Saginaw–Detroit | 458                           |
| Detroit–Toledo–Saginaw–Kalamazoo–Grand Rapids–Detroit | 540                           |
| Detroit–Toledo–Saginaw–Grand Rapids–Kalamazoo–Detroit | 504                           |
| Detroit–Saginaw–Toledo–Grand Rapids–Kalamazoo–Detroit | 598                           |
| Detroit–Saginaw–Toledo–Kalamazoo–Grand Rapids–Detroit | 576                           |
| Detroit–Saginaw–Kalamazoo–Toledo–Grand Rapids–Detroit | 682                           |
| Detroit–Saginaw–Grand Rapids–Toledo–Kalamazoo–Detroit | 646                           |
| Detroit–Grand Rapids–Saginaw–Toledo–Kalamazoo–Detroit | 670                           |
| Detroit–Grand Rapids–Toledo–Saginaw–Kalamazoo–Detroit | 728                           |

We just described an instance of the **traveling salesperson problem**. The traveling salesperson problem asks for the circuit of minimum total weight in a weighted, complete, undirected graph that visits each vertex exactly once and returns to its starting point. This is equivalent to asking for a Hamilton circuit with minimum total weight in the complete graph, because each vertex is visited exactly once in the circuit.

The most straightforward way to solve an instance of the traveling salesperson problem is to examine all possible Hamilton circuits and select one of minimum total length. How many circuits do we have to examine to solve the problem if there are  $n$  vertices in the graph? Once a starting point is chosen, there are  $(n - 1)!$  different Hamilton circuits to examine, because there are  $n - 1$  choices for the second vertex,  $n - 2$  choices for the third vertex, and so on. Because a Hamilton circuit can be traveled in reverse order, we need only examine  $(n - 1)!/2$  circuits to find our answer. Note that  $(n - 1)!/2$  grows extremely rapidly. Trying to solve a traveling salesperson problem in this way when there are only a few dozen vertices is impractical. For example, with 25 vertices, a total of  $24!/2$  (approximately  $3.1 \times 10^{23}$ ) different Hamilton circuits would have to be considered. If it took just one nanosecond ( $10^{-9}$  second) to examine each Hamilton circuit, a total of approximately ten million years would be required to find a minimum-length Hamilton circuit in this graph by exhaustive search techniques.

Because the traveling salesperson problem has both practical and theoretical importance, a great deal of effort has been devoted to devising efficient algorithms that solve it. However, no algorithm with polynomial worst-case time complexity is known for solving this problem. Furthermore, if a polynomial worst-case time complexity algorithm were discovered for the traveling salesperson problem, many other difficult problems would also be solvable using polynomial worst-case time complexity algorithms (such as determining whether a proposition in  $n$  variables is a tautology, discussed in Chapter 1). This follows from the theory of NP-completeness. (For more information about this, consult [GaJo79].)

A practical approach to the traveling salesperson problem when there are many vertices to visit is to use an **approximation algorithm**. These are algorithms that do not necessarily produce the exact solution to the problem but instead are guaranteed to produce a solution that is close to an exact solution. (Also, see the preamble to Exercise 46 in the Supplementary Exercises of Chapter 3.) That is, they may produce a Hamilton circuit with total weight  $W'$  such that  $W \leq W' \leq cW$ , where  $W$  is the total length of an exact solution and  $c$  is a constant. For example, there is an algorithm with polynomial worst-case time complexity that works if the weighted graph satisfies the triangle inequality such that  $c = 3/2$ . For general weighted graphs for every positive real number  $k$  no algorithm is known that will always produce a solution at most  $k$  times a best solution. If such an algorithm existed, this would show that the class P would be the same as the class NP, perhaps the most famous open question about the complexity of algorithms (see Section 3.3).

An 1832 handbook *Der Handlungsreisende* (The Traveling Salesman) mentions the traveling salesman problem, with sample tours through Germany and Switzerland.

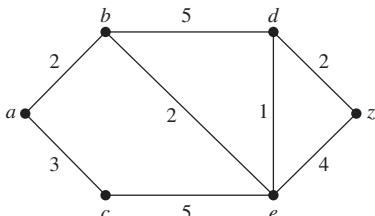
In practice, algorithms have been developed that can solve traveling salesperson problems with as many as 1000 vertices within 2% of an exact solution using only a few minutes of computer time. For more information about the traveling salesperson problem, including history, applications, and algorithms, see the chapter on this topic in *Applications of Discrete Mathematics* [MiRo91] also available on the website for this book.

## Exercises

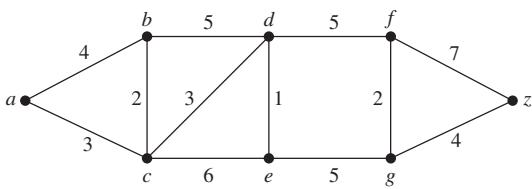
1. For each of these problems about a subway system, describe a weighted graph model that can be used to solve the problem.
  - a) What is the least amount of time required to travel between two stops?
  - b) What is the minimum distance that can be traveled to reach a stop from another stop?
  - c) What is the least fare required to travel between two stops if fares between stops are added to give the total fare?

In Exercises 2–4 find the length of a shortest path between  $a$  and  $z$  in the given weighted graph.

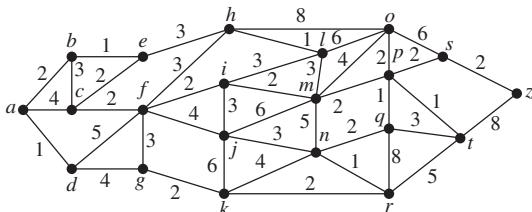
2.



3.



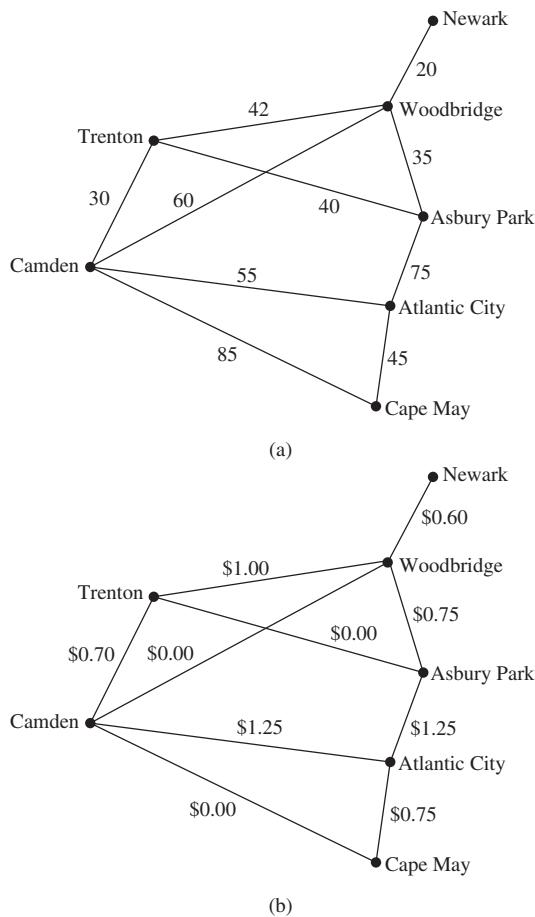
4.



5. Find a shortest path between  $a$  and  $z$  in each of the weighted graphs in Exercises 2–4.
6. Find the length of a shortest path between these pairs of vertices in the weighted graph in Exercise 3.
  - a)  $a$  and  $d$
  - b)  $a$  and  $f$
  - c)  $c$  and  $f$
  - d)  $b$  and  $z$

7. Find shortest paths in the weighted graph in Exercise 3 between the pairs of vertices in Exercise 6.
8. Find a shortest path (in mileage) between each of the following pairs of cities in the airline system shown in Figure 1.
  - a) New York and Los Angeles
  - b) Boston and San Francisco
  - c) Miami and Denver
  - d) Miami and Los Angeles
9. Find a combination of flights with the least total air time between the pairs of cities in Exercise 8, using the flight times shown in Figure 1.
10. Find a least expensive combination of flights connecting the pairs of cities in Exercise 8, using the fares shown in Figure 1.
11. Find a shortest route (in distance) between computer centers in each of these pairs of cities in the communications network shown in Figure 2.
  - a) Boston and Los Angeles
  - b) New York and San Francisco
  - c) Dallas and San Francisco
  - d) Denver and New York
12. Find a route with the shortest response time between the pairs of computer centers in Exercise 11 using the response times given in Figure 2.
13. Find a least expensive route, in monthly lease charges, between the pairs of computer centers in Exercise 11 using the lease charges given in Figure 2.
14. Explain how to find a path with the least number of edges between two vertices in an undirected graph by considering it as a shortest path problem in a weighted graph.
15. Extend Dijkstra's algorithm for finding the length of a shortest path between two vertices in a weighted simple connected graph so that the length of a shortest path between the vertex  $a$  and every other vertex of the graph is found.
16. Extend Dijkstra's algorithm for finding the length of a shortest path between two vertices in a weighted simple connected graph so that a shortest path between these vertices is constructed.

17. The weighted graphs in the figures here show some major roads in New Jersey. Part (a) shows the distances between cities on these roads; part (b) shows the tolls.



21. Use Floyd's algorithm to find the distance between all pairs of vertices in the weighted graph in Figure 4(a).
- \*22. Prove that Floyd's algorithm determines the shortest distance between all pairs of vertices in a weighted simple graph.
- \*23. Give a big- $O$  estimate of the number of operations (comparisons and additions) used by Floyd's algorithm to determine the shortest distance between every pair of vertices in a weighted simple graph with  $n$  vertices.
- \*24. Show that Dijkstra's algorithm may not work if edges can have negative weights.

#### ALGORITHM 2 Floyd's Algorithm.

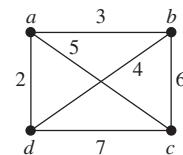
```

procedure Floyd(G : weighted simple graph)
{ G has vertices v_1, v_2, \dots, v_n and weights $w(v_i, v_j)$
 with $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge}
for $i := 1$ to n
 for $j := 1$ to n
 $d(v_i, v_j) := w(v_i, v_j)$
for $i := 1$ to n
 for $j := 1$ to n
 for $k := 1$ to n
 if $d(v_j, v_i) + d(v_i, v_k) < d(v_j, v_k)$
 then $d(v_j, v_k) := d(v_j, v_i) + d(v_i, v_k)$
return [$d(v_i, v_j)$] { $d(v_i, v_j)$ is the length of a shortest
path between v_i and v_j for $1 \leq i \leq n, 1 \leq j \leq n$ }
```

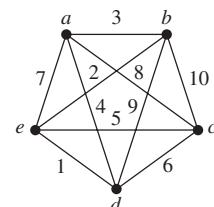
- a) Find a shortest route in distance between Newark and Camden, and between Newark and Cape May, using these roads.
- b) Find a least expensive route in terms of total tolls using the roads in the graph between the pairs of cities in part (a) of this exercise.
18. Is a shortest path between two vertices in a weighted graph unique if the weights of edges are distinct?
19. What are some applications where it is necessary to find the length of a longest simple path between two vertices in a weighted graph?
20. What is the length of a longest simple path in the weighted graph in Figure 4 between  $a$  and  $z$ ? Between  $c$  and  $z$ ?

 **Floyd's algorithm**, displayed as Algorithm 2, can be used to find the length of a shortest path between all pairs of vertices in a weighted connected simple graph. However, this algorithm cannot be used to construct shortest paths. (We assign an infinite weight to any pair of vertices not connected by an edge in the graph.)

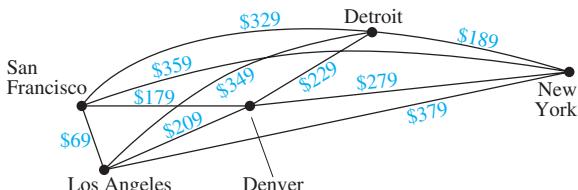
25. Solve the traveling salesperson problem for this graph by finding the total weight of all Hamilton circuits and determining a circuit with minimum total weight.



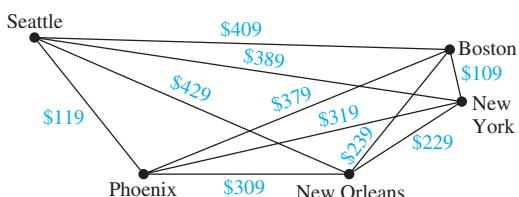
26. Solve the traveling salesperson problem for this graph by finding the total weight of all Hamilton circuits and determining a circuit with minimum total weight.



27. Find a route with the least total airfare that visits each of the cities in this graph, where the weight on an edge is the least price available for a flight between the two cities.



28. Find a route with the least total airfare that visits each of the cities in this graph, where the weight on an edge is the least price available for a flight between the two cities.



29. Construct a weighted undirected graph such that the total weight of a circuit that visits every vertex at least once is minimized for a circuit that visits some vertices more than once. [Hint: There are examples with three vertices.]

30. Show that the problem of finding a circuit of minimum total weight that visits every vertex of a weighted graph at least once can be reduced to the problem of finding a circuit of minimum total weight that visits each vertex of a weighted graph exactly once. Do so by constructing a new weighted graph with the same vertices and edges as the original graph but whose weight of the edge connecting the vertices  $u$  and  $v$  is equal to the minimum total weight of a path from  $u$  to  $v$  in the original graph.

- \*31. The **longest path problem** in a weighted directed graph with no simple circuits asks for a path in this graph such that the sum of its edge weights is a maximum. Devise an algorithm for solving the longest path problem. [Hint: First find a topological ordering of the vertices of the graph.]

## 10.7 Planar Graphs

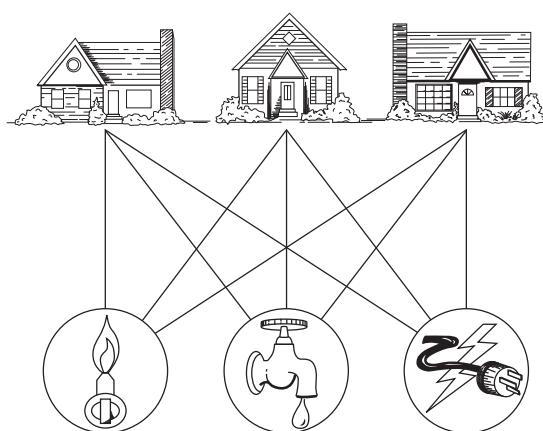
### Introduction



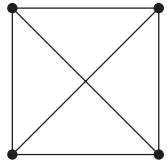
Consider the problem of joining three houses to each of three separate utilities, as shown in Figure 1. Is it possible to join these houses and utilities so that none of the connections cross? This problem can be modeled using the complete bipartite graph  $K_{3,3}$ . The original question can be rephrased as: Can  $K_{3,3}$  be drawn in the plane so that no two of its edges cross?

In this section we will study the question of whether a graph can be drawn in the plane without edges crossing. In particular, we will answer the houses-and-utilities problem.

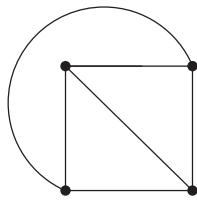
There are always many ways to represent a graph. When is it possible to find at least one way to represent this graph in a plane without any edges crossing?



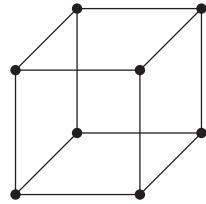
**FIGURE 1** Three Houses and Three Utilities.



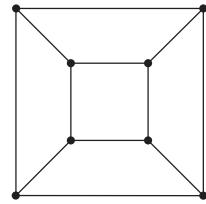
**FIGURE 2** The Graph  $K_4$ .



**FIGURE 3**  $K_4$  Drawn with No Crossings.



**FIGURE 4** The Graph  $Q_3$ .



**FIGURE 5** A Planar Representation of  $Q_3$ .

### DEFINITION 1

A graph is called *planar* if it can be drawn in the plane without any edges crossing (where a crossing of edges is the intersection of the lines or arcs representing them at a point other than their common endpoint). Such a drawing is called a *planar representation* of the graph.

A graph may be planar even if it is usually drawn with crossings, because it may be possible to draw it in a different way without crossings.

**EXAMPLE 1** Is  $K_4$  (shown in Figure 2 with two edges crossing) planar?

*Solution:*  $K_4$  is planar because it can be drawn without crossings, as shown in Figure 3.

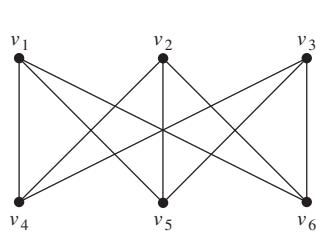
**EXAMPLE 2** Is  $Q_3$ , shown in Figure 4, planar?

*Solution:*  $Q_3$  is planar, because it can be drawn without any edges crossing, as shown in Figure 5.

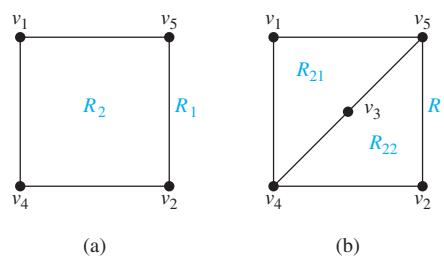
We can show that a graph is planar by displaying a planar representation. It is harder to show that a graph is nonplanar. We will give an example to show how this can be done in an ad hoc fashion. Later we will develop some general results that can be used to do this.

**EXAMPLE 3** Is  $K_{3,3}$ , shown in Figure 6, planar?

*Solution:* Any attempt to draw  $K_{3,3}$  in the plane with no edges crossing is doomed. We now show why. In any planar representation of  $K_{3,3}$ , the vertices  $v_1$  and  $v_2$  must be connected to both  $v_4$  and  $v_5$ . These four edges form a closed curve that splits the plane into two regions,  $R_1$  and  $R_2$ , as shown in Figure 7(a). The vertex  $v_3$  is in either  $R_1$  or  $R_2$ . When  $v_3$  is in  $R_2$ , the inside of the closed curve, the edges between  $v_3$  and  $v_4$  and between  $v_3$  and  $v_5$  separate  $R_2$  into two subregions,  $R_{21}$  and  $R_{22}$ , as shown in Figure 7(b).



**FIGURE 6** The Graph  $K_{3,3}$ .



**FIGURE 7** Showing that  $K_{3,3}$  Is Nonplanar.

Next, note that there is no way to place the final vertex  $v_6$  without forcing a crossing. For if  $v_6$  is in  $R_1$ , then the edge between  $v_6$  and  $v_3$  cannot be drawn without a crossing. If  $v_6$  is in  $R_{21}$ , then the edge between  $v_2$  and  $v_6$  cannot be drawn without a crossing. If  $v_6$  is in  $R_{22}$ , then the edge between  $v_1$  and  $v_6$  cannot be drawn without a crossing.

A similar argument can be used when  $v_3$  is in  $R_1$ . The completion of this argument is left for the reader (see Exercise 10). It follows that  $K_{3,3}$  is not planar.  $\blacktriangleleft$

Example 3 solves the utilities-and-houses problem that was described at the beginning of this section. The three houses and three utilities cannot be connected in the plane without a crossing. A similar argument can be used to show that  $K_5$  is nonplanar. (See Exercise 11.)

**APPLICATIONS OF PLANAR GRAPHS** Planarity of graphs plays an important role in the design of electronic circuits. We can model a circuit with a graph by representing components of the circuit by vertices and connections between them by edges. We can print a circuit on a single board with no connections crossing if the graph representing the circuit is planar. When this graph is not planar, we must turn to more expensive options. For example, we can partition the vertices in the graph representing the circuit into planar subgraphs. We then construct the circuit using multiple layers. (See the preamble to Exercise 30 to learn about the thickness of a graph.) We can construct the circuit using insulated wires whenever connections cross. In this case, drawing the graph with the fewest possible crossings is important. (See the preamble to Exercise 26 to learn about the crossing number of a graph.)

The planarity of graphs is also useful in the design of road networks. Suppose we want to connect a group of cities by roads. We can model a road network connecting these cities using a simple graph with vertices representing the cities and edges representing the highways connecting them. We can built this road network without using underpasses or overpasses if the resulting graph is planar.

## Euler's Formula

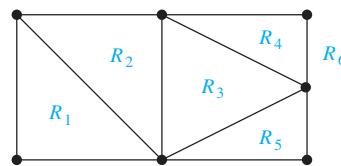
A planar representation of a graph splits the plane into **regions**, including an unbounded region. For instance, the planar representation of the graph shown in Figure 8 splits the plane into six regions. These are labeled in the figure. Euler showed that all planar representations of a graph split the plane into the same number of regions. He accomplished this by finding a relationship among the number of regions, the number of vertices, and the number of edges of a planar graph.

### THEOREM 1

**EULER'S FORMULA** Let  $G$  be a connected planar simple graph with  $e$  edges and  $v$  vertices. Let  $r$  be the number of regions in a planar representation of  $G$ . Then  $r = e - v + 2$ .



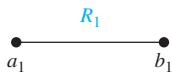
**Proof:** First, we specify a planar representation of  $G$ . We will prove the theorem by constructing a sequence of subgraphs  $G_1, G_2, \dots, G_e = G$ , successively adding an edge at each stage. This is done using the following inductive definition. Arbitrarily pick one edge of  $G$  to obtain  $G_1$ . Obtain  $G_n$  from  $G_{n-1}$  by arbitrarily adding an edge that is incident with a vertex already in  $G_{n-1}$ ,



**FIGURE 8** The Regions of the Planar Representation of a Graph.

adding the other vertex incident with this edge if it is not already in  $G_{n-1}$ . This construction is possible because  $G$  is connected.  $G$  is obtained after  $e$  edges are added. Let  $r_n$ ,  $e_n$ , and  $v_n$  represent the number of regions, edges, and vertices of the planar representation of  $G_n$  induced by the planar representation of  $G$ , respectively.

The proof will now proceed by induction. The relationship  $r_1 = e_1 - v_1 + 2$  is true for  $G_1$ , because  $e_1 = 1$ ,  $v_1 = 2$ , and  $r_1 = 1$ . This is shown in Figure 9.



**FIGURE 9** The Basis Case of the Proof of Euler's Formula.

Now assume that  $r_k = e_k - v_k + 2$ . Let  $\{a_{k+1}, b_{k+1}\}$  be the edge that is added to  $G_k$  to obtain  $G_{k+1}$ . There are two possibilities to consider. In the first case, both  $a_{k+1}$  and  $b_{k+1}$  are already in  $G_k$ . These two vertices must be on the boundary of a common region  $R$ , or else it would be impossible to add the edge  $\{a_{k+1}, b_{k+1}\}$  to  $G_k$  without two edges crossing (and  $G_{k+1}$  is planar). The addition of this new edge splits  $R$  into two regions. Consequently, in this case,  $r_{k+1} = r_k + 1$ ,  $e_{k+1} = e_k + 1$ , and  $v_{k+1} = v_k$ . Thus, each side of the formula relating the number of regions, edges, and vertices increases by exactly one, so this formula is still true. In other words,  $r_{k+1} = e_{k+1} - v_{k+1} + 2$ . This case is illustrated in Figure 10(a).

In the second case, one of the two vertices of the new edge is not already in  $G_k$ . Suppose that  $a_{k+1}$  is in  $G_k$  but that  $b_{k+1}$  is not. Adding this new edge does not produce any new regions, because  $b_{k+1}$  must be in a region that has  $a_{k+1}$  on its boundary. Consequently,  $r_{k+1} = r_k$ . Moreover,  $e_{k+1} = e_k + 1$  and  $v_{k+1} = v_k + 1$ . Each side of the formula relating the number of regions, edges, and vertices remains the same, so the formula is still true. In other words,  $r_{k+1} = e_{k+1} - v_{k+1} + 2$ . This case is illustrated in Figure 10(b).

We have completed the induction argument. Hence,  $r_n = e_n - v_n + 2$  for all  $n$ . Because the original graph is the graph  $G_e$ , obtained after  $e$  edges have been added, the theorem is true.  $\triangleleft$

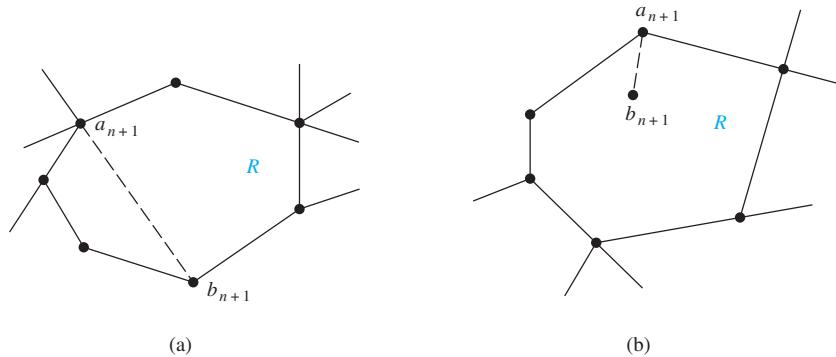
Euler's formula is illustrated in Example 4.

**EXAMPLE 4** Suppose that a connected planar simple graph has 20 vertices, each of degree 3. Into how many regions does a representation of this planar graph split the plane?

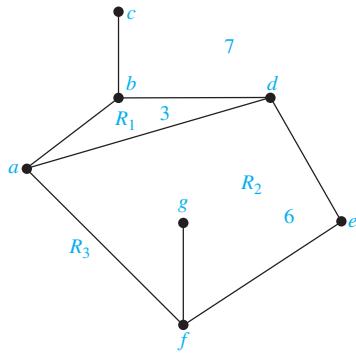
**Solution:** This graph has 20 vertices, each of degree 3, so  $v = 20$ . Because the sum of the degrees of the vertices,  $3v = 3 \cdot 20 = 60$ , is equal to twice the number of edges,  $2e$ , we have  $2e = 60$ , or  $e = 30$ . Consequently, from Euler's formula, the number of regions is

$$r = e - v + 2 = 30 - 20 + 2 = 12.$$

Euler's formula can be used to establish some inequalities that must be satisfied by planar graphs. One such inequality is given in Corollary 1.



**FIGURE 10** Adding an Edge to  $G_n$  to Produce  $G_{n+1}$ .



**FIGURE 11** The Degrees of Regions.

### COROLLARY 1

If  $G$  is a connected planar simple graph with  $e$  edges and  $v$  vertices, where  $v \geq 3$ , then  $e \leq 3v - 6$ .

Before we prove Corollary 1 we will use it to prove the following useful result.

### COROLLARY 2

If  $G$  is a connected planar simple graph, then  $G$  has a vertex of degree not exceeding five.

**Proof:** If  $G$  has one or two vertices, the result is true. If  $G$  has at least three vertices, by Corollary 1 we know that  $e \leq 3v - 6$ , so  $2e \leq 6v - 12$ . If the degree of every vertex were at least six, then because  $2e = \sum_{v \in V} \deg(v)$  (by the handshaking theorem), we would have  $2e \geq 6v$ . But this contradicts the inequality  $2e \leq 6v - 12$ . It follows that there must be a vertex with degree no greater than five.  $\triangleleft$

The proof of Corollary 1 is based on the concept of the **degree** of a region, which is defined to be the number of edges on the boundary of this region. When an edge occurs twice on the boundary (so that it is traced out twice when the boundary is traced out), it contributes two to the degree. We denote the degree of a region  $R$  by  $\deg(R)$ . The degrees of the regions of the graph shown in Figure 11 are displayed in the figure.

The proof of Corollary 1 can now be given.

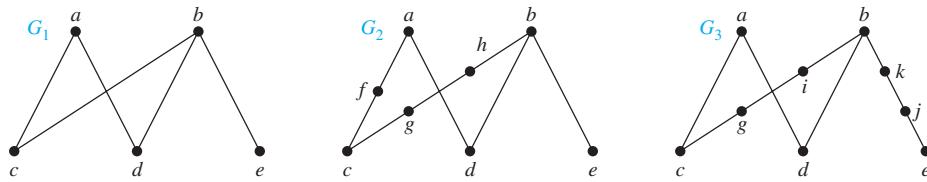
**Proof:** A connected planar simple graph drawn in the plane divides the plane into regions, say  $r$  of them. The degree of each region is at least three. (Because the graphs discussed here are simple graphs, no multiple edges that could produce regions of degree two, or loops that could produce regions of degree one, are permitted.) In particular, note that the degree of the unbounded region is at least three because there are at least three vertices in the graph.

Note that the sum of the degrees of the regions is exactly twice the number of edges in the graph, because each edge occurs on the boundary of a region exactly twice (either in two different regions, or twice in the same region). Because each region has degree greater than or equal to three, it follows that

$$2e = \sum_{\text{all regions } R} \deg(R) \geq 3r.$$

Hence,

$$(2/3)e \geq r.$$



**FIGURE 12 Homeomorphic Graphs.**

Using  $r = e - v + 2$  (Euler's formula), we obtain

$$e - v + 2 \leq (2/3)e.$$

It follows that  $e/3 \leq v - 2$ . This shows that  $e \leq 3v - 6$ .  $\triangleleft$

This corollary can be used to demonstrate that  $K_5$  is nonplanar.

**EXAMPLE 5** Show that  $K_5$  is nonplanar using Corollary 1.

**Solution:** The graph  $K_5$  has five vertices and 10 edges. However, the inequality  $e \leq 3v - 6$  is not satisfied for this graph because  $e = 10$  and  $3v - 6 = 9$ . Therefore,  $K_5$  is not planar.  $\triangleleft$

It was previously shown that  $K_{3,3}$  is not planar. Note, however, that this graph has six vertices and nine edges. This means that the inequality  $e = 9 \leq 12 = 3 \cdot 6 - 6$  is satisfied. Consequently, the fact that the inequality  $e \leq 3v - 6$  is satisfied does *not* imply that a graph is planar. However, the following corollary of Theorem 1 can be used to show that  $K_{3,3}$  is nonplanar.

### COROLLARY 3

If a connected planar simple graph has  $e$  edges and  $v$  vertices with  $v \geq 3$  and no circuits of length three, then  $e \leq 2v - 4$ .

The proof of Corollary 3 is similar to that of Corollary 1, except that in this case the fact that there are no circuits of length three implies that the degree of a region must be at least four. The details of this proof are left for the reader (see Exercise 15).

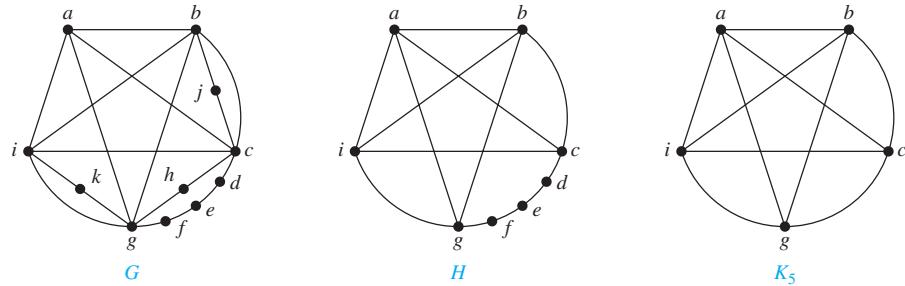
**EXAMPLE 6** Use Corollary 3 to show that  $K_{3,3}$  is nonplanar.

**Solution:** Because  $K_{3,3}$  has no circuits of length three (this is easy to see because it is bipartite), Corollary 3 can be used.  $K_{3,3}$  has six vertices and nine edges. Because  $e = 9$  and  $2v - 4 = 8$ , Corollary 3 shows that  $K_{3,3}$  is nonplanar.  $\triangleleft$



**KAZIMIERZ KURATOWSKI (1896–1980)** Kazimierz Kuratowski, the son of a famous Warsaw lawyer, attended secondary school in Warsaw. He studied in Glasgow, Scotland, from 1913 to 1914 but could not return there after the outbreak of World War I. In 1915 he entered Warsaw University, where he was active in the Polish patriotic student movement. He published his first paper in 1919 and received his Ph.D. in 1921. He was an active member of the group known as the Warsaw School of Mathematics, working in the areas of the foundations of set theory and topology. He was appointed associate professor at the Lwów Polytechnical University, where he stayed for seven years, collaborating with the important Polish mathematicians Banach and Ulam. In 1930, while at Lwów, Kuratowski completed his work characterizing planar graphs.

In 1934 he returned to Warsaw University as a full professor. Until the start of World War II, he was active in research and teaching. During the war, because of the persecution of educated Poles, Kuratowski went into hiding under an assumed name and taught at the clandestine Warsaw University. After the war he helped revive Polish mathematics, serving as director of the Polish National Mathematics Institute. He wrote over 180 papers and three widely used textbooks.



**FIGURE 13** The Undirected Graph  $G$ , a Subgraph  $H$  Homeomorphic to  $K_5$ , and  $K_5$ .

### Kuratowski's Theorem

We have seen that  $K_{3,3}$  and  $K_5$  are not planar. Clearly, a graph is not planar if it contains either of these two graphs as a subgraph. Surprisingly, all nonplanar graphs must contain a subgraph that can be obtained from  $K_{3,3}$  or  $K_5$  using certain permitted operations.

If a graph is planar, so will be any graph obtained by removing an edge  $\{u, v\}$  and adding a new vertex  $w$  together with edges  $\{u, w\}$  and  $\{w, v\}$ . Such an operation is called an **elementary subdivision**. The graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  are called **homeomorphic** if they can be obtained from the same graph by a sequence of elementary subdivisions.

**EXAMPLE 7** Show that the graphs  $G_1$ ,  $G_2$ , and  $G_3$  displayed in Figure 12 are all homeomorphic.

**Solution:** These three graphs are homeomorphic because all three can be obtained from  $G_1$  by elementary subdivisions.  $G_1$  can be obtained from itself by an empty sequence of elementary subdivisions. To obtain  $G_2$  from  $G_1$  we can use this sequence of elementary subdivisions: (i) remove the edge  $\{a, c\}$ , add the vertex  $f$ , and add the edges  $\{a, f\}$  and  $\{f, c\}$ ; (ii) remove the edge  $\{b, c\}$ , add the vertex  $g$ , and add the edges  $\{b, g\}$  and  $\{g, c\}$ ; and (iii) remove the edge  $\{b, g\}$ , add the vertex  $h$ , and add the edges  $\{g, h\}$  and  $\{b, h\}$ . We leave it to the reader to determine the sequence of elementary subdivisions needed to obtain  $G_3$  from  $G_1$ .  $\blacktriangleleft$

The Polish mathematician Kazimierz Kuratowski established Theorem 2 in 1930, which characterizes planar graphs using the concept of graph homeomorphism.

### THEOREM 2

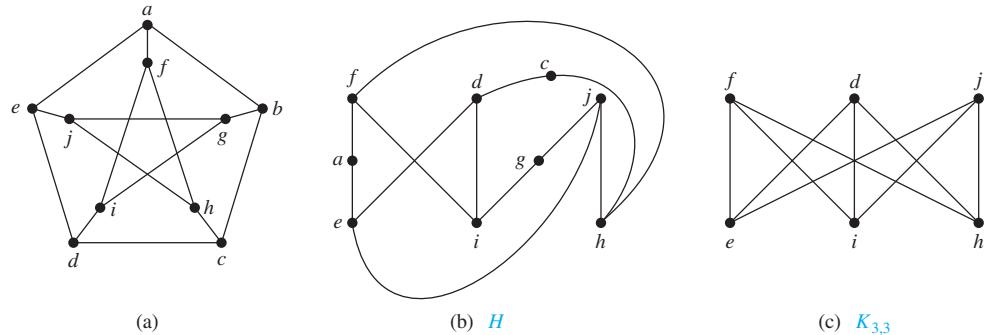
A graph is nonplanar if and only if it contains a subgraph homeomorphic to  $K_{3,3}$  or  $K_5$ .

It is clear that a graph containing a subgraph homeomorphic to  $K_{3,3}$  or  $K_5$  is nonplanar. However, the proof of the converse, namely that every nonplanar graph contains a subgraph homeomorphic to  $K_{3,3}$  or  $K_5$ , is complicated and will not be given here. Examples 8 and 9 illustrate how Kuratowski's theorem is used.

**EXAMPLE 8** Determine whether the graph  $G$  shown in Figure 13 is planar.



**Solution:**  $G$  has a subgraph  $H$  homeomorphic to  $K_5$ .  $H$  is obtained by deleting  $h$ ,  $j$ , and  $k$  and all edges incident with these vertices.  $H$  is homeomorphic to  $K_5$  because it can be obtained from  $K_5$  (with vertices  $a$ ,  $b$ ,  $c$ ,  $d$ , and  $e$ ) by a sequence of elementary subdivisions, adding the vertices  $d$ ,  $e$ , and  $f$ . (The reader should construct such a sequence of elementary subdivisions.) Hence,  $G$  is nonplanar.  $\blacktriangleleft$



**FIGURE 14** (a) The Petersen Graph, (b) a Subgraph  $H$  Homeomorphic to  $K_{3,3}$ , and (c)  $K_{3,3}$ .

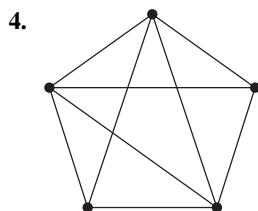
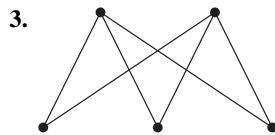
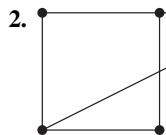
**EXAMPLE 9** Is the Petersen graph, shown in Figure 14(a), planar? (The Danish mathematician Julius Petersen studied this graph in 1891; it is often used to illustrate various theoretical properties of graphs.)

**Solution:** The subgraph  $H$  of the Petersen graph obtained by deleting  $b$  and the three edges that have  $b$  as an endpoint, shown in Figure 14(b), is homeomorphic to  $K_{3,3}$ , with vertex sets  $\{f, d, j\}$  and  $\{e, i, h\}$ , because it can be obtained by a sequence of elementary subdivisions, deleting  $\{d, h\}$  and adding  $\{c, h\}$  and  $\{c, d\}$ , deleting  $\{e, f\}$  and adding  $\{a, e\}$  and  $\{a, f\}$ , and deleting  $\{i, j\}$  and adding  $\{g, i\}$  and  $\{g, j\}$ . Hence, the Petersen graph is not planar.  $\blacktriangleleft$

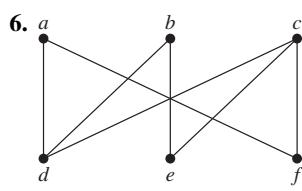
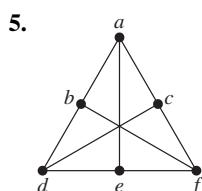
## Exercises

1. Can five houses be connected to two utilities without connections crossing?

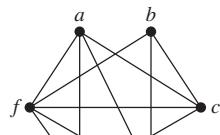
In Exercises 2–4 draw the given planar graph without any crossings.



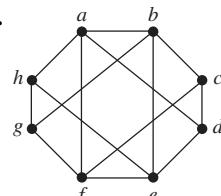
In Exercises 5–9 determine whether the given graph is planar. If so, draw it so that no edges cross.



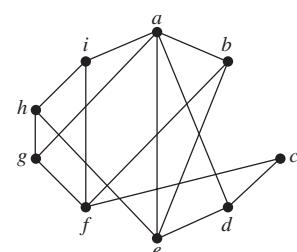
- 7.



- 8.



- 9.



10. Complete the argument in Example 3.  
 11. Show that  $K_5$  is nonplanar using an argument similar to that given in Example 3.  
 12. Suppose that a connected planar graph has eight vertices, each of degree three. Into how many regions is the plane divided by a planar representation of this graph?  
 13. Suppose that a connected planar graph has six vertices, each of degree four. Into how many regions is the plane divided by a planar representation of this graph?  
 14. Suppose that a connected planar graph has 30 edges. If a planar representation of this graph divides the plane into 20 regions, how many vertices does this graph have?

**15.** Prove Corollary 3.

**16.** Suppose that a connected bipartite planar simple graph has  $e$  edges and  $v$  vertices. Show that  $e \leq 2v - 4$  if  $v \geq 3$ .

**\*17.** Suppose that a connected planar simple graph with  $e$  edges and  $v$  vertices contains no simple circuits of length 4 or less. Show that  $e \leq (5/3)v - (10/3)$  if  $v \geq 4$ .

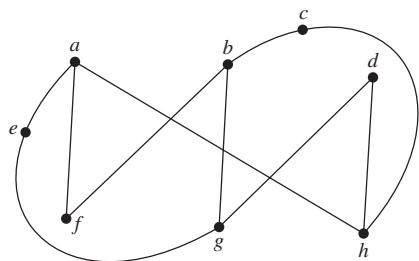
**18.** Suppose that a planar graph has  $k$  connected components,  $e$  edges, and  $v$  vertices. Also suppose that the plane is divided into  $r$  regions by a planar representation of the graph. Find a formula for  $r$  in terms of  $e$ ,  $v$ , and  $k$ .

**19.** Which of these nonplanar graphs have the property that the removal of any vertex and all edges incident with that vertex produces a planar graph?

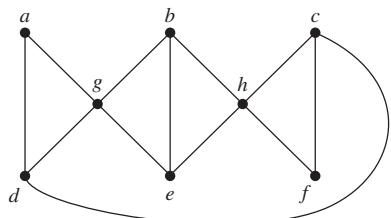
- a)  $K_5$     b)  $K_6$     c)  $K_{3,3}$     d)  $K_{3,4}$

In Exercises 20–22 determine whether the given graph is homeomorphic to  $K_{3,3}$ .

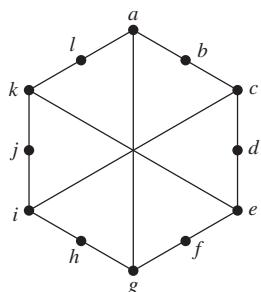
**20.**



**21.**

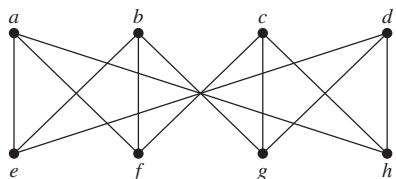


**22.**

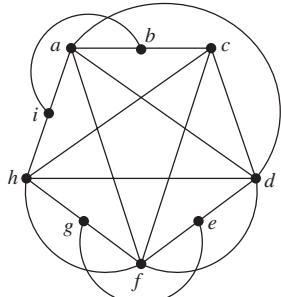


In Exercises 23–25 use Kuratowski's theorem to determine whether the given graph is planar.

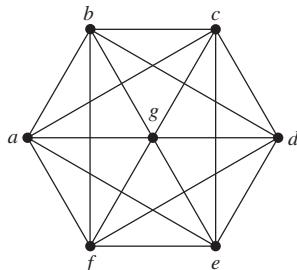
**23.**



**24.**



**25.**



The **crossing number** of a simple graph is the minimum number of crossings that can occur when this graph is drawn in the plane where no three arcs representing edges are permitted to cross at the same point.

**26.** Show that  $K_{3,3}$  has 1 as its crossing number.

**\*27.** Find the crossing numbers of each of these nonplanar graphs.

- a)  $K_5$     b)  $K_6$     c)  $K_7$   
d)  $K_{3,4}$     e)  $K_{4,4}$     f)  $K_{5,5}$

**\*28.** Find the crossing number of the Petersen graph.

**\*29.** Show that if  $m$  and  $n$  are even positive integers, the crossing number of  $K_{m,n}$  is less than or equal to  $mn(m-2)(n-2)/16$ . [Hint: Place  $m$  vertices along the  $x$ -axis so that they are equally spaced and symmetric about the origin and place  $n$  vertices along the  $y$ -axis so that they are equally spaced and symmetric about the origin. Now connect each of the  $m$  vertices on the  $x$ -axis to each of the vertices on the  $y$ -axis and count the crossings.]

The **thickness** of a simple graph  $G$  is the smallest number of planar subgraphs of  $G$  that have  $G$  as their union.

**30.** Show that  $K_{3,3}$  has 2 as its thickness.

**\*31.** Find the thickness of the graphs in Exercise 27.

**32.** Show that if  $G$  is a connected simple graph with  $v$  vertices and  $e$  edges, where  $v \geq 3$ , then the thickness of  $G$  is at least  $\lceil e/(3v-6) \rceil$ .

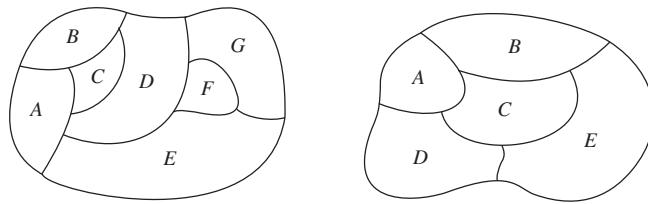
**\*33.** Use Exercise 32 to show that the thickness of  $K_n$  is at least  $\lfloor (n+7)/6 \rfloor$  whenever  $n$  is a positive integer.

**34.** Show that if  $G$  is a connected simple graph with  $v$  vertices and  $e$  edges, where  $v \geq 3$ , and no circuits of length three, then the thickness of  $G$  is at least  $\lceil e/(2v-4) \rceil$ .

**35.** Use Exercise 34 to show that the thickness of  $K_{m,n}$ , where  $m$  and  $n$  are not both 1, is at least  $\lceil mn/(2m+2n-4) \rceil$  whenever  $m$  and  $n$  are positive integers.

**\*36.** Draw  $K_5$  on the surface of a torus (a doughnut-shaped solid) so that no edges cross.

**\*37.** Draw  $K_{3,3}$  on the surface of a torus so that no edges cross.



**FIGURE 1** Two Maps.

## 10.8 Graph Coloring

### Introduction



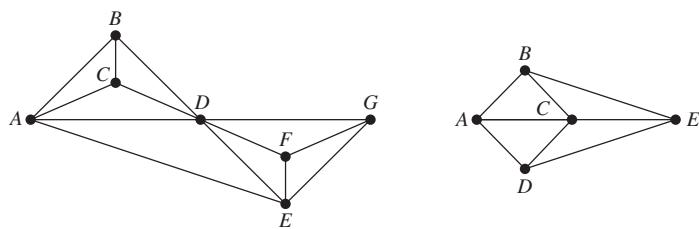
Problems related to the coloring of maps of regions, such as maps of parts of the world, have generated many results in graph theory. When a map\* is colored, two regions with a common border are customarily assigned different colors. One way to ensure that two adjacent regions never have the same color is to use a different color for each region. However, this is inefficient, and on maps with many regions it would be hard to distinguish similar colors. Instead, a small number of colors should be used whenever possible. Consider the problem of determining the least number of colors that can be used to color a map so that adjacent regions never have the same color. For instance, for the map shown on the left in Figure 1, four colors suffice, but three colors are not enough. (The reader should check this.) In the map on the right in Figure 1, three colors are sufficient (but two are not).

Each map in the plane can be represented by a graph. To set up this correspondence, each region of the map is represented by a vertex. Edges connect two vertices if the regions represented by these vertices have a common border. Two regions that touch at only one point are not considered adjacent. The resulting graph is called the **dual graph** of the map. By the way in which dual graphs of maps are constructed, it is clear that any map in the plane has a planar dual graph. Figure 2 displays the dual graphs that correspond to the maps shown in Figure 1.

The problem of coloring the regions of a map is equivalent to the problem of coloring the vertices of the dual graph so that no two adjacent vertices in this graph have the same color. We now define a graph coloring.

#### DEFINITION 1

A *coloring* of a simple graph is the assignment of a color to each vertex of the graph so that no two adjacent vertices are assigned the same color.



**FIGURE 2** Dual Graphs of the Maps in Figure 1.

\*We will assume that all regions in a map are connected. This eliminates any problems presented by such geographical entities as Michigan.

A graph can be colored by assigning a different color to each of its vertices. However, for most graphs a coloring can be found that uses fewer colors than the number of vertices in the graph. What is the least number of colors necessary?

## DEFINITION 2

The *chromatic number* of a graph is the least number of colors needed for a coloring of this graph. The chromatic number of a graph  $G$  is denoted by  $\chi(G)$ . (Here  $\chi$  is the Greek letter *chi*.)

Note that asking for the chromatic number of a planar graph is the same as asking for the minimum number of colors required to color a planar map so that no two adjacent regions are assigned the same color. This question has been studied for more than 100 years. The answer is provided by one of the most famous theorems in mathematics.

## THEOREM 1

**THE FOUR COLOR THEOREM** The chromatic number of a planar graph is no greater than four.

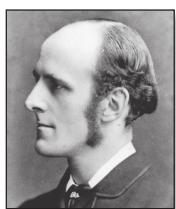


The four color theorem was originally posed as a conjecture in the 1850s. It was finally proved by the American mathematicians Kenneth Appel and Wolfgang Haken in 1976. Prior to 1976, many incorrect proofs were published, often with hard-to-find errors. In addition, many futile attempts were made to construct counterexamples by drawing maps that require more than four colors. (Proving the five color theorem is not that difficult; see Exercise 36.)

Perhaps the most notorious fallacious proof in all of mathematics is the incorrect proof of the four color theorem published in 1879 by a London barrister and amateur mathematician, Alfred Kempe. Mathematicians accepted his proof as correct until 1890, when Percy Heawood found an error that made Kempe's argument incomplete. However, Kempe's line of reasoning turned out to be the basis of the successful proof given by Appel and Haken. Their proof relies on a careful case-by-case analysis carried out by computer. They showed that if the four color theorem were false, there would have to be a counterexample of one of approximately 2000 different types, and they then showed that none of these types exists. They used over 1000 hours of computer time in their proof. This proof generated a large amount of controversy, because computers played such an important role in it. For example, could there be an error in a computer program that led to incorrect results? Was their argument really a proof if it depended on what could be unreliable computer output? Since their proof appeared, simpler proofs that rely on checking fewer types of possible counterexamples have been found and a proof using an automated proof system has been created. However, no proof not relying on a computer has yet been found.

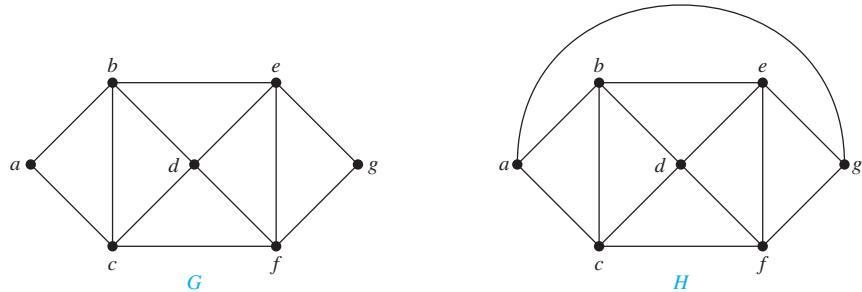
Note that the four color theorem applies only to planar graphs. Nonplanar graphs can have arbitrarily large chromatic numbers, as will be shown in Example 2.

Two things are required to show that the chromatic number of a graph is  $k$ . First, we must show that the graph can be colored with  $k$  colors. This can be done by constructing such a coloring. Second, we must show that the graph cannot be colored using fewer than  $k$  colors. Examples 1–4 illustrate how chromatic numbers can be found.




---

**ALFRED BRAY KEMPE (1849–1922)** Kempe was a barrister and a leading authority on ecclesiastical law. However, having studied mathematics at Cambridge University, he retained his interest in it, and later in life he devoted considerable time to mathematical research. Kempe made contributions to kinematics, the branch of mathematics dealing with motion, and to mathematical logic. However, Kempe is best remembered for his fallacious proof of the four color theorem.



**FIGURE 3** The Simple Graphs  $G$  and  $H$ .

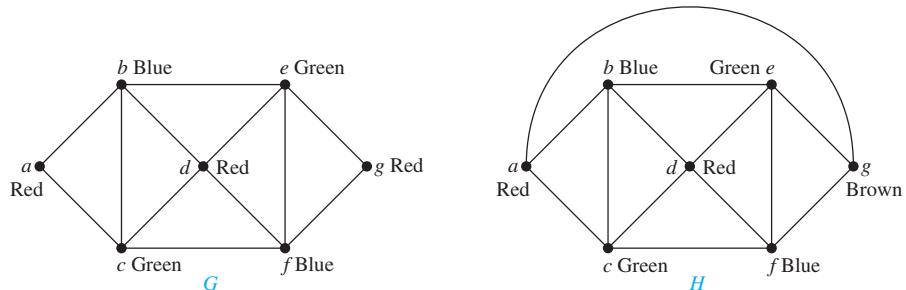
### EXAMPLE 1

What are the chromatic numbers of the graphs  $G$  and  $H$  shown in Figure 3?

#### Extra Examples

**Solution:** The chromatic number of  $G$  is at least three, because the vertices  $a$ ,  $b$ , and  $c$  must be assigned different colors. To see if  $G$  can be colored with three colors, assign red to  $a$ , blue to  $b$ , and green to  $c$ . Then,  $d$  can (and must) be colored red because it is adjacent to  $b$  and  $c$ . Furthermore,  $e$  can (and must) be colored green because it is adjacent only to vertices colored red and blue, and  $f$  can (and must) be colored blue because it is adjacent only to vertices colored red and green. Finally,  $g$  can (and must) be colored red because it is adjacent only to vertices colored blue and green. This produces a coloring of  $G$  using exactly three colors. Figure 4 displays such a coloring.

The graph  $H$  is made up of the graph  $G$  with an edge connecting  $a$  and  $g$ . Any attempt to color  $H$  using three colors must follow the same reasoning as that used to color  $G$ , except at the last stage, when all vertices other than  $g$  have been colored. Then, because  $g$  is adjacent (in  $H$ ) to vertices colored red, blue, and green, a fourth color, say brown, needs to be used. Hence,  $H$  has a chromatic number equal to 4. A coloring of  $H$  is shown in Figure 4. 



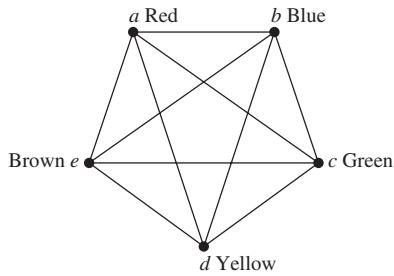
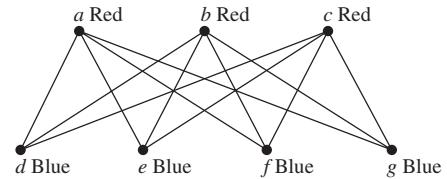
**FIGURE 4** Colorings of the Graphs  $G$  and  $H$ .

---

**HISTORICAL NOTE** In 1852, an ex-student of Augustus De Morgan, Francis Guthrie, noticed that the counties in England could be colored using four colors so that no adjacent counties were assigned the same color. On this evidence, he conjectured that the four color theorem was true. Francis told his brother Frederick, at that time a student of De Morgan, about this problem. Frederick in turn asked his teacher De Morgan about his brother's conjecture. De Morgan was extremely interested in this problem and publicized it throughout the mathematical community. In fact, the first written reference to the conjecture can be found in a letter from De Morgan to Sir William Rowan Hamilton. Although De Morgan thought Hamilton would be interested in this problem, Hamilton apparently was not interested in it, because it had nothing to do with quaternions.

#### Links

**HISTORICAL NOTE** Although a simpler proof of the four color theorem was found by Robertson, Sanders, Seymour, and Thomas in 1996, reducing the computational part of the proof to examining 633 configurations, no proof that does not rely on extensive computation has yet been found.

**FIGURE 5** A Coloring of  $K_5$ .**FIGURE 6** A Coloring of  $K_{3,4}$ .

**EXAMPLE 2** What is the chromatic number of  $K_n$ ?

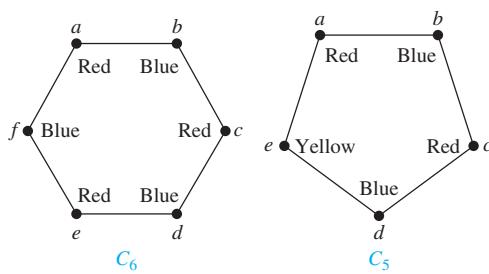
**Solution:** A coloring of  $K_n$  can be constructed using  $n$  colors by assigning a different color to each vertex. Is there a coloring using fewer colors? The answer is no. No two vertices can be assigned the same color, because every two vertices of this graph are adjacent. Hence, the chromatic number of  $K_n$  is  $n$ . That is,  $\chi(K_n) = n$ . (Recall that  $K_n$  is not planar when  $n \geq 5$ , so this result does not contradict the four color theorem.) A coloring of  $K_5$  using five colors is shown in Figure 5.  $\blacktriangleleft$

**EXAMPLE 3** What is the chromatic number of the complete bipartite graph  $K_{m,n}$ , where  $m$  and  $n$  are positive integers?

**Solution:** The number of colors needed may seem to depend on  $m$  and  $n$ . However, as Theorem 4 in Section 10.2 tells us, only two colors are needed, because  $K_{m,n}$  is a bipartite graph. Hence,  $\chi(K_{m,n}) = 2$ . This means that we can color the set of  $m$  vertices with one color and the set of  $n$  vertices with a second color. Because edges connect only a vertex from the set of  $m$  vertices and a vertex from the set of  $n$  vertices, no two adjacent vertices have the same color. A coloring of  $K_{3,4}$  with two colors is displayed in Figure 6.  $\blacktriangleleft$

**EXAMPLE 4** What is the chromatic number of the graph  $C_n$ , where  $n \geq 3$ ? (Recall that  $C_n$  is the cycle with  $n$  vertices.)

**Solution:** We will first consider some individual cases. To begin, let  $n = 6$ . Pick a vertex and color it red. Proceed clockwise in the planar depiction of  $C_6$  shown in Figure 7. It is necessary to assign a second color, say blue, to the next vertex reached. Continue in the clockwise direction; the third vertex can be colored red, the fourth vertex blue, and the fifth vertex red. Finally, the sixth vertex, which is adjacent to the first, can be colored blue. Hence, the chromatic number of  $C_6$  is 2. Figure 7 displays the coloring constructed here.

**FIGURE 7** Colorings of  $C_5$  and  $C_6$ .

Next, let  $n = 5$  and consider  $C_5$ . Pick a vertex and color it red. Proceeding clockwise, it is necessary to assign a second color, say blue, to the next vertex reached. Continuing in the clockwise direction, the third vertex can be colored red, and the fourth vertex can be colored blue. The fifth vertex cannot be colored either red or blue, because it is adjacent to the fourth vertex and the first vertex. Consequently, a third color is required for this vertex. Note that we would have also needed three colors if we had colored vertices in the counterclockwise direction. Thus, the chromatic number of  $C_5$  is 3. A coloring of  $C_5$  using three colors is displayed in Figure 7.

In general, two colors are needed to color  $C_n$  when  $n$  is even. To construct such a coloring, simply pick a vertex and color it red. Proceed around the graph in a clockwise direction (using a planar representation of the graph) coloring the second vertex blue, the third vertex red, and so on. The  $n$ th vertex can be colored blue, because the two vertices adjacent to it, namely the  $(n - 1)$ st and the first vertices, are both colored red.

When  $n$  is odd and  $n > 1$ , the chromatic number of  $C_n$  is 3. To see this, pick an initial vertex. To use only two colors, it is necessary to alternate colors as the graph is traversed in a clockwise direction. However, the  $n$ th vertex reached is adjacent to two vertices of different colors, namely, the first and  $(n - 1)$ st. Hence, a third color must be used.

We have shown that  $\chi(C_n) = 2$  if  $n$  is an even positive integer with  $n \geq 4$  and  $\chi(C_n) = 3$  if  $n$  is an odd positive integer with  $n \geq 3$ . 



The best algorithms known for finding the chromatic number of a graph have exponential worst-case time complexity (in the number of vertices of the graph). Even the problem of finding an approximation to the chromatic number of a graph is difficult. It has been shown that if there were an algorithm with polynomial worst-case time complexity that could approximate the chromatic number of a graph up to a factor of 2 (that is, construct a bound that was no more than double the chromatic number of the graph), then an algorithm with polynomial worst-case time complexity for finding the chromatic number of the graph would also exist.

## Applications of Graph Colorings

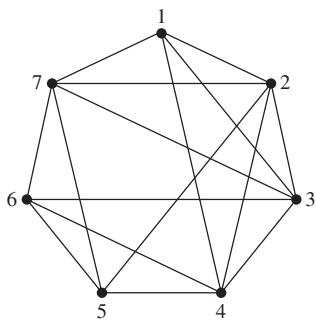
Graph coloring has a variety of applications to problems involving scheduling and assignments. (Note that because no efficient algorithm is known for graph coloring, this does not lead to efficient algorithms for scheduling and assignments.) Examples of such applications will be given here. The first application deals with the scheduling of final exams.

**EXAMPLE 5 Scheduling Final Exams** How can the final exams at a university be scheduled so that no student has two exams at the same time?

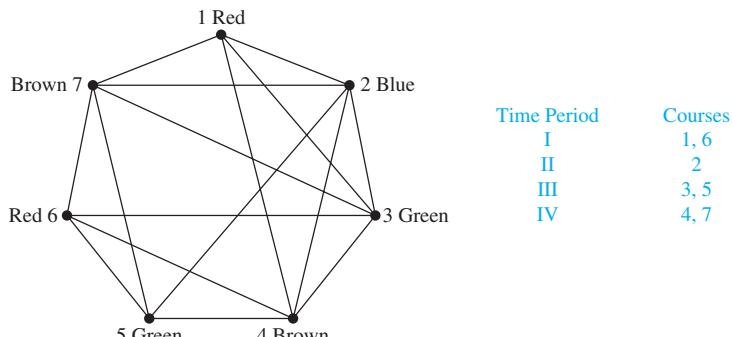
**Solution:** This scheduling problem can be solved using a graph model, with vertices representing courses and with an edge between two vertices if there is a common student in the courses they represent. Each time slot for a final exam is represented by a different color. A scheduling of the exams corresponds to a coloring of the associated graph.

For instance, suppose there are seven finals to be scheduled. Suppose the courses are numbered 1 through 7. Suppose that the following pairs of courses have common students: 1 and 2, 1 and 3, 1 and 4, 1 and 7, 2 and 3, 2 and 4, 2 and 5, 2 and 7, 3 and 4, 3 and 6, 3 and 7, 4 and 5, 4 and 6, 5 and 6, 5 and 7, and 6 and 7. In Figure 8 the graph associated with this set of classes is shown. A scheduling consists of a coloring of this graph.

Because the chromatic number of this graph is 4 (the reader should verify this), four time slots are needed. A coloring of the graph using four colors and the associated schedule are shown in Figure 9. 



**FIGURE 8** The Graph Representing the Scheduling of Final Exams.



**FIGURE 9** Using a Coloring to Schedule Final Exams.

Now consider an application to the assignment of television channels.

**EXAMPLE 6 Frequency Assignments** Television channels 2 through 13 are assigned to stations in North America so that no two stations within 150 miles can operate on the same channel. How can the assignment of channels be modeled by graph coloring?

**Solution:** Construct a graph by assigning a vertex to each station. Two vertices are connected by an edge if they are located within 150 miles of each other. An assignment of channels corresponds to a coloring of the graph, where each color represents a different channel. 

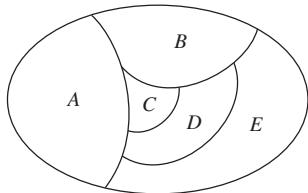
An application of graph coloring to compilers is considered in Example 7.

**EXAMPLE 7 Index Registers** In efficient compilers the execution of loops is speeded up when frequently used variables are stored temporarily in index registers in the central processing unit, instead of in regular memory. For a given loop, how many index registers are needed? This problem can be addressed using a graph coloring model. To set up the model, let each vertex of a graph represent a variable in the loop. There is an edge between two vertices if the variables they represent must be stored in index registers at the same time during the execution of the loop. Thus, the chromatic number of the graph gives the number of index registers needed, because different registers must be assigned to variables when the vertices representing these variables are adjacent in the graph. 

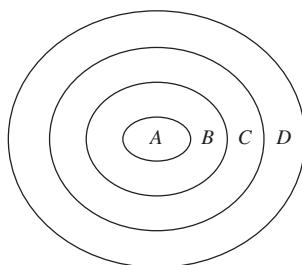
## Exercises

In Exercises 1–4 construct the dual graph for the map shown. Then find the number of colors needed to color the map so that no two adjacent regions have the same color.

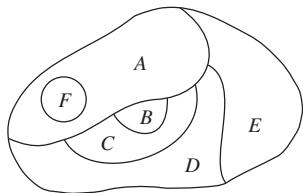
1.



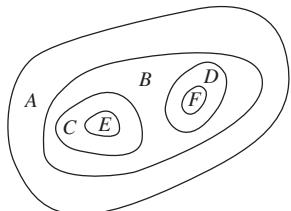
2.



3.

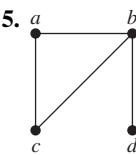


4.

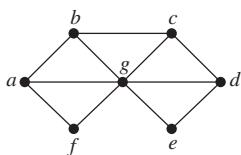


In Exercises 5–11 find the chromatic number of the given graph.

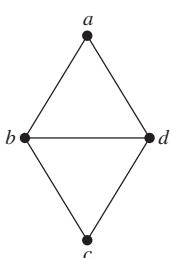
5.



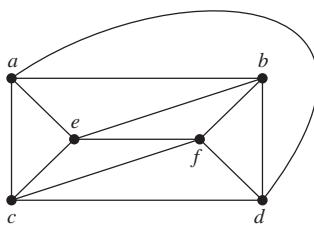
6.



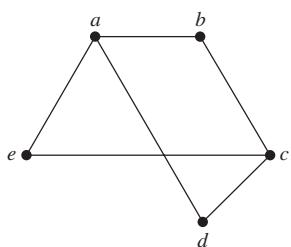
7.



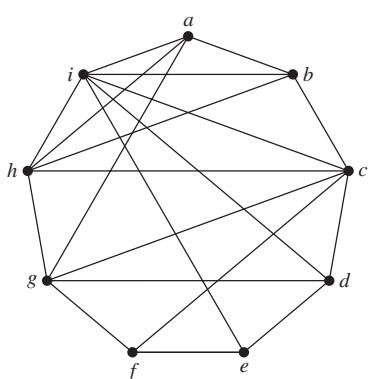
8.



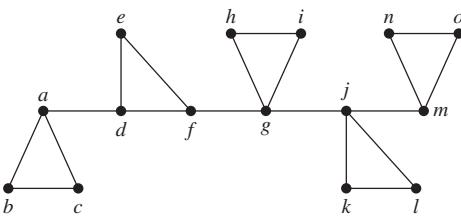
9.



10.



11.



12. For the graphs in Exercises 5–11, decide whether it is possible to decrease the chromatic number by removing a single vertex and all edges incident with it.

13. Which graphs have a chromatic number of 1?

14. What is the least number of colors needed to color a map of the United States? Do not consider adjacent states that meet only at a corner. Suppose that Michigan is one region. Consider the vertices representing Alaska and Hawaii as isolated vertices.

15. What is the chromatic number of  $W_n$ ?

16. Show that a simple graph that has a circuit with an odd number of vertices in it cannot be colored using two colors.

17. Schedule the final exams for Math 115, Math 116, Math 185, Math 195, CS 101, CS 102, CS 273, and CS 473, using the fewest number of different time slots, if there are no students taking both Math 115 and CS 473, both Math 116 and CS 473, both Math 195 and CS 101, both Math 195 and CS 102, both Math 115 and Math 116, both Math 115 and Math 185, and both Math 185 and Math 195, but there are students in every other pair of courses.

18. How many different channels are needed for six stations located at the distances shown in the table, if two stations cannot use the same channel when they are within 150 miles of each other?

|   | 1   | 2   | 3   | 4   | 5   | 6   |
|---|-----|-----|-----|-----|-----|-----|
| 1 | —   | 85  | 175 | 200 | 50  | 100 |
| 2 | 85  | —   | 125 | 175 | 100 | 160 |
| 3 | 175 | 125 | —   | 100 | 200 | 250 |
| 4 | 200 | 175 | 100 | —   | 210 | 220 |
| 5 | 50  | 100 | 200 | 210 | —   | 100 |
| 6 | 100 | 160 | 250 | 220 | 100 | —   |

19. The mathematics department has six committees, each meeting once a month. How many different meeting times must be used to ensure that no member is scheduled to attend two meetings at the same time if the committees are  $C_1 = \{\text{Arlinghaus, Brand, Zaslavsky}\}$ ,  $C_2 = \{\text{Brand, Lee, Rosen}\}$ ,  $C_3 = \{\text{Arlinghaus, Rosen, Zaslavsky}\}$ ,  $C_4 = \{\text{Lee, Rosen, Zaslavsky}\}$ ,  $C_5 = \{\text{Arlinghaus, Brand}\}$ , and  $C_6 = \{\text{Brand, Rosen, Zaslavsky}\}$ ?

- 20.** A zoo wants to set up natural habitats in which to exhibit its animals. Unfortunately, some animals will eat some of the others when given the opportunity. How can a graph model and a coloring be used to determine the number of different habitats needed and the placement of the animals in these habitats?

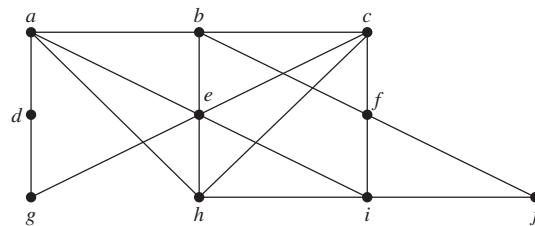


An **edge coloring** of a graph is an assignment of colors to edges so that edges incident with a common vertex are assigned different colors. The **edge chromatic number** of a graph is the smallest number of colors that can be used in an edge coloring of the graph. The edge chromatic number of a graph  $G$  is denoted by  $\chi'(G)$ .

- 21.** Find the edge chromatic number of each of the graphs in Exercises 5–11.
- 22.** Suppose that  $n$  devices are on a circuit board and that these devices are connected by colored wires. Express the number of colors needed for the wires, in terms of the edge chromatic number of the graph representing this circuit board, under the requirement that the wires leaving a particular device must be different colors. Explain your answer.
- 23.** Find the edge chromatic numbers of
- $C_n$ , where  $n \geq 3$ .
  - $W_n$ , where  $n \geq 3$ .
- 24.** Show that the edge chromatic number of a graph must be at least as large as the maximum degree of a vertex of the graph.
- 25.** Show that if  $G$  is a graph with  $n$  vertices, then no more than  $n/2$  edges can be colored the same in an edge coloring of  $G$ .
- \*26.** Find the edge chromatic number of  $K_n$  when  $n$  is a positive integer.
- 27.** Seven variables occur in a loop of a computer program. The variables and the steps during which they must be stored are  $t$ : steps 1 through 6;  $u$ : step 2;  $v$ : steps 2 through 4;  $w$ : steps 1, 3, and 5;  $x$ : steps 1 and 6;  $y$ : steps 3 through 6; and  $z$ : steps 4 and 5. How many different index registers are needed to store these variables during execution?
- 28.** What can be said about the chromatic number of a graph that has  $K_n$  as a subgraph?

This algorithm can be used to color a simple graph: First, list the vertices  $v_1, v_2, v_3, \dots, v_n$  in order of decreasing degree so that  $\deg(v_1) \geq \deg(v_2) \geq \dots \geq \deg(v_n)$ . Assign color 1 to  $v_1$  and to the next vertex in the list not adjacent to  $v_1$  (if one exists), and successively to each vertex in the list not adjacent to a vertex already assigned color 1. Then assign color 2 to the first vertex in the list not already colored. Successively assign color 2 to vertices in the list that have not already been colored and are not adjacent to vertices assigned color 2. If uncolored vertices remain, assign color 3 to the first vertex in the list not yet colored, and use color 3 to successively color those vertices not already colored and not adjacent to vertices assigned color 3. Continue this process until all vertices are colored.

- 29.** Construct a coloring of the graph shown using this algorithm.

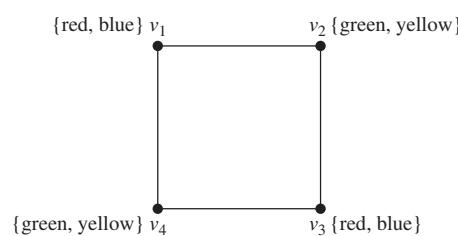


- \*30.** Use pseudocode to describe this coloring algorithm.  
**\*31.** Show that the coloring produced by this algorithm may use more colors than are necessary to color a graph.

A connected graph  $G$  is called **chromatically  $k$ -critical** if the chromatic number of  $G$  is  $k$ , but for every edge of  $G$ , the chromatic number of the graph obtained by deleting this edge from  $G$  is  $k - 1$ .

- 32.** Show that  $C_n$  is chromatically 3-critical whenever  $n$  is an odd positive integer,  $n \geq 3$ .
- 33.** Show that  $W_n$  is chromatically 4-critical whenever  $n$  is an odd integer,  $n \geq 3$ .
- 34.** Show that  $W_4$  is not chromatically 3-critical.
- 35.** Show that if  $G$  is a chromatically  $k$ -critical graph, then the degree of every vertex of  $G$  is at least  $k - 1$ .

A  **$k$ -tuple coloring** of a graph  $G$  is an assignment of a set of  $k$  different colors to each of the vertices of  $G$  such that no two adjacent vertices are assigned a common color. We denote by  $\chi_k(G)$  the smallest positive integer  $n$  such that  $G$  has a  $k$ -tuple coloring using  $n$  colors. For example,  $\chi_2(C_4) = 4$ . To see this, note that using only four colors we can assign two colors to each vertex of  $C_4$ , as illustrated, so that no two adjacent vertices are assigned the same color. Furthermore, no fewer than four colors suffice because the vertices  $v_1$  and  $v_2$  each must be assigned two colors, and a common color cannot be assigned to both  $v_1$  and  $v_2$ . (For more information about  $k$ -tuple coloring, see [MiRo91].)



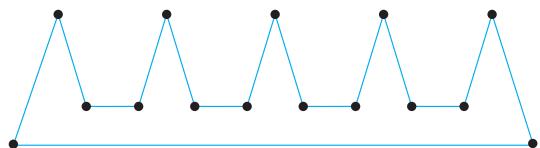
- 36.** Find these values:
- |                          |                             |                         |
|--------------------------|-----------------------------|-------------------------|
| <b>a)</b> $\chi_2(K_3)$  | <b>b)</b> $\chi_2(K_4)$     | <b>c)</b> $\chi_2(W_4)$ |
| <b>d)</b> $\chi_2(C_5)$  | <b>e)</b> $\chi_2(K_{3,4})$ | <b>f)</b> $\chi_3(K_5)$ |
| <b>*g)</b> $\chi_3(C_5)$ | <b>h)</b> $\chi_3(K_{4,5})$ |                         |

- \*37. Let  $G$  and  $H$  be the graphs displayed in Figure 3. Find
- $\chi_2(G)$ .
  - $\chi_2(H)$ .
  - $\chi_3(G)$ .
  - $\chi_3(H)$ .
38. What is  $\chi_k(G)$  if  $G$  is a bipartite graph and  $k$  is a positive integer?
39. Frequencies for mobile radio (or cellular) telephones are assigned by zones. Each zone is assigned a set of frequencies to be used by vehicles in that zone. The same frequency cannot be used in different zones when interference can occur between telephones in these zones. Explain how a  $k$ -tuple coloring can be used to assign  $k$  frequencies to each mobile radio zone in a region.
- \*40. Show that every planar graph  $G$  can be colored using six or fewer colors. [Hint: Use mathematical induction on the number of vertices of the graph. Apply Corollary 2 of Section 10.7 to find a vertex  $v$  with  $\deg(v) \leq 5$ . Consider the subgraph of  $G$  obtained by deleting  $v$  and all edges incident with it.]
- \*\*41. Show that every planar graph  $G$  can be colored using five or fewer colors. [Hint: Use the hint provided for Exercise 40.]

The famous Art Gallery Problem asks how many guards are needed to see all parts of an art gallery, where the gallery is the interior and boundary of a polygon with  $n$  sides. To state this problem more precisely, we need some terminology. A point  $x$  inside or on the boundary of a simple polygon  $P$  **covers** or **sees** a point  $y$  inside or on  $P$  if all points on the line segment  $xy$  are in the interior or on the boundary of  $P$ . We say that a set of points is a **guarding set** of a simple polygon  $P$  if for every point  $y$  inside  $P$  or on the boundary of  $P$  there is a point  $x$  in this guarding set that sees  $y$ . Denote by  $G(P)$  the minimum number of points needed to guard the simple polygon  $P$ . The **art gallery problem** asks for the function  $g(n)$ , which is the maximum value of  $G(P)$  over all simple polygons with  $n$  vertices. That is,  $g(n)$  is the minimum positive integer for which

it is guaranteed that a simple polygon with  $n$  vertices can be guarded with  $g(n)$  or fewer guards.

42. Show that  $g(3) = 1$  and  $g(4) = 1$  by showing that all triangles and quadrilaterals can be guarded using one point.
- \*43. Show that  $g(5) = 1$ . That is, show that all pentagons can be guarded using one point. [Hint: Show that there are either 0, 1, or 2 vertices with an interior angle greater than 180 degrees and that in each case, one guard suffices.]
- \*44. Show that  $g(6) = 2$  by first using Exercises 42 and 43 as well as Lemma 1 in Section 5.2 to show that  $g(6) \leq 2$  and then find a simple hexagon for which two guards are needed.
- \*45. Show that  $g(n) \geq \lfloor n/3 \rfloor$ . [Hint: Consider the polygon with  $3k$  vertices that resembles a comb with  $k$  prongs, such as the polygon with 15 sides shown here.]



- \*46. Solve the art gallery problem by proving the **art gallery theorem**, which states that at most  $\lfloor n/3 \rfloor$  guards are needed to guard the interior and boundary of a simple polygon with  $n$  vertices. [Hint: Use Theorem 1 in Section 5.2 to triangulate the simple polygon into  $n - 2$  triangles. Then show that it is possible to color the vertices of the triangulated polygon using three colors so that no two adjacent vertices have the same color. Use induction and Exercise 23 in Section 5.2. Finally, put guards at all vertices that are colored red, where red is the color used least in the coloring of the vertices. Show that placing guards at these points is all that is needed.]

## Key Terms and Results

### TERMS

- undirected edge:** an edge associated to a set  $\{u, v\}$ , where  $u$  and  $v$  are vertices
- directed edge:** an edge associated to an ordered pair  $(u, v)$ , where  $u$  and  $v$  are vertices
- multiple edges:** distinct edges connecting the same vertices
- multiple directed edges:** distinct directed edges associated with the same ordered pair  $(u, v)$ , where  $u$  and  $v$  are vertices
- loop:** an edge connecting a vertex with itself
- undirected graph:** a set of vertices and a set of undirected edges each of which is associated with a set of one or two of these vertices
- simple graph:** an undirected graph with no multiple edges or loops
- multigraph:** an undirected graph that may contain multiple edges but no loops

**pseudograph:** an undirected graph that may contain multiple edges and loops

**directed graph:** a set of vertices together with a set of directed edges each of which is associated with an ordered pair of vertices

**directed multigraph:** a graph with directed edges that may contain multiple directed edges

**simple directed graph:** a directed graph without loops or multiple directed edges

**adjacent:** two vertices are adjacent if there is an edge between them

**incident:** an edge is incident with a vertex if the vertex is an endpoint of that edge

**deg  $v$  (degree of the vertex  $v$  in an undirected graph):** the number of edges incident with  $v$  with loops counted twice

- deg<sup>-</sup>(v) (the in-degree of the vertex v in a graph with directed edges):** the number of edges with v as their terminal vertex
- deg<sup>+</sup>(v) (the out-degree of the vertex v in a graph with directed edges):** the number of edges with v as their initial vertex
- underlying undirected graph of a graph with directed edges:** the undirected graph obtained by ignoring the directions of the edges
- $K_n$  (complete graph on n vertices):** the undirected graph with  $n$  vertices where each pair of vertices is connected by an edge
- bipartite graph:** a graph with vertex set that can be partitioned into subsets  $V_1$  and  $V_2$  so that each edge connects a vertex in  $V_1$  and a vertex in  $V_2$ . The pair  $(V_1, V_2)$  is called a **bipartition** of  $V$ .
- $K_{m,n}$  (complete bipartite graph):** the graph with vertex set partitioned into a subset of  $m$  elements and a subset of  $n$  elements with two vertices connected by an edge if and only if one is in the first subset and the other is in the second subset
- $C_n$  (cycle of size n),  $n \geq 3$ :** the graph with  $n$  vertices  $v_1, v_2, \dots, v_n$  and edges  $\{v_1, v_2\}, \{v_2, v_3\}, \dots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$
- $W_n$  (wheel of size n),  $n \geq 3$ :** the graph obtained from  $C_n$  by adding a vertex and edges from this vertex to the original vertices in  $C_n$
- $Q_n$  (n-cube),  $n \geq 1$ :** the graph that has the  $2^n$  bit strings of length  $n$  as its vertices and edges connecting every pair of bit strings that differ by exactly one bit
- matching in a graph G:** a set of edges such that no two edges have a common endpoint
- complete matching M from  $V_1$  to  $V_2$ :** a matching such that every vertex in  $V_1$  is an endpoint of an edge in  $M$
- maximum matching:** a matching containing the most edges among all matchings in a graph
- isolated vertex:** a vertex of degree zero
- pendant vertex:** a vertex of degree one
- regular graph:** a graph where all vertices have the same degree
- subgraph of a graph  $G = (V, E)$ :** a graph  $(W, F)$ , where  $W$  is a subset of  $V$  and  $F$  is a subset of  $E$
- $G_1 \cup G_2$  (union of  $G_1$  and  $G_2$ ):** the graph  $(V_1 \cup V_2, E_1 \cup E_2)$ , where  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$
- adjacency matrix:** a matrix representing a graph using the adjacency of vertices
- incidence matrix:** a matrix representing a graph using the incidence of edges and vertices
- isomorphic simple graphs:** the simple graphs  $G_1 = (V_1, E_1)$  and  $G_2 = (V_2, E_2)$  are isomorphic if there exists a one-to-one correspondence  $f$  from  $V_1$  to  $V_2$  such that  $\{f(v_1), f(v_2)\} \in E_2$  if and only if  $\{v_1, v_2\} \in E_1$  for all  $v_1$  and  $v_2$  in  $V_1$
- invariant for graph isomorphism:** a property that isomorphic graphs either both have or both do not have
- path from  $u$  to  $v$  in an undirected graph:** a sequence of edges  $e_1, e_2, \dots, e_n$ , where  $e_i$  is associated to  $\{x_i, x_{i+1}\}$  for  $i = 0, 1, \dots, n$ , where  $x_0 = u$  and  $x_{n+1} = v$
- path from  $u$  to  $v$  in a graph with directed edges:** a sequence of edges  $e_1, e_2, \dots, e_n$ , where  $e_i$  is associated to  $(x_i, x_{i+1})$  for  $i = 0, 1, \dots, n$ , where  $x_0 = u$  and  $x_{n+1} = v$
- simple path:** a path that does not contain an edge more than once
- circuit:** a path of length  $n \geq 1$  that begins and ends at the same vertex
- connected graph:** an undirected graph with the property that there is a path between every pair of vertices
- cut vertex of  $G$ :** a vertex  $v$  such that  $G - v$  is disconnected
- cut edge of  $G$ :** an edge  $e$  such that  $G - e$  is disconnected
- nonseparable graph:** a graph without a cut vertex
- vertex cut of  $G$ :** a subset  $V'$  of the set of vertices of  $G$  such that  $G - V'$  is disconnected
- $\kappa(G)$  (the vertex connectivity of  $G$ ):** the size of a smallest vertex cut of  $G$
- k-connected graph:** a graph that has a vertex connectivity no smaller than  $k$
- edge cut of  $G$ :** a set of edges  $E'$  of  $G$  such that  $G - E'$  is disconnected
- $\lambda(G)$  (the edge connectivity of  $G$ ):** the size of a smallest edge cut of  $G$
- connected component of a graph  $G$ :** a maximal connected subgraph of  $G$
- strongly connected directed graph:** a directed graph with the property that there is a directed path from every vertex to every vertex
- strongly connected component of a directed graph  $G$ :** a maximal strongly connected subgraph of  $G$
- Euler path:** a path that contains every edge of a graph exactly once
- Euler circuit:** a circuit that contains every edge of a graph exactly once
- Hamilton path:** a path in a graph that passes through each vertex exactly once
- Hamilton circuit:** a circuit in a graph that passes through each vertex exactly once
- weighted graph:** a graph with numbers assigned to its edges
- shortest-path problem:** the problem of determining the path in a weighted graph such that the sum of the weights of the edges in this path is a minimum over all paths between specified vertices
- traveling salesperson problem:** the problem that asks for the circuit of shortest total length that visits every vertex of a weighted graph exactly once
- planar graph:** a graph that can be drawn in the plane with no crossings
- regions of a representation of a planar graph:** the regions the plane is divided into by the planar representation of the graph
- elementary subdivision:** the removal of an edge  $\{u, v\}$  of an undirected graph and the addition of a new vertex  $w$  together with edges  $\{u, w\}$  and  $\{w, v\}$
- homeomorphic:** two undirected graphs are homeomorphic if they can be obtained from the same graph by a sequence of elementary subdivisions
- graph coloring:** an assignment of colors to the vertices of a graph so that no two adjacent vertices have the same color

**chromatic number:** the minimum number of colors needed in a coloring of a graph

## RESULTS

**The handshaking theorem:** If  $G = (V, E)$  be an undirected graph with  $m$  edges, then  $2m = \sum_{v \in V} \deg(v)$ .

**Hall's marriage theorem:** The bipartite graph  $G = (V, E)$  with bipartition  $(V_1, V_2)$  has a complete matching from  $V_1$  to  $V_2$  if and only if  $|N(A)| \geq |A|$  for all subsets  $A$  of  $V_1$ .

There is an Euler circuit in a connected multigraph if and only if every vertex has even degree.

There is an Euler path in a connected multigraph if and only if at most two vertices have odd degree.

**Dijkstra's algorithm:** a procedure for finding a shortest path between two vertices in a weighted graph (see Section 10.6).

**Euler's formula:**  $r = e - v + 2$  where  $r$ ,  $e$ , and  $v$  are the number of regions of a planar representation, the number of edges, and the number of vertices, respectively, of a connected planar graph.

**Kuratowski's theorem:** A graph is nonplanar if and only if it contains a subgraph homeomorphic to  $K_{3,3}$  or  $K_5$ . (Proof beyond scope of this book.)

**The four color theorem:** Every planar graph can be colored using no more than four colors. (Proof far beyond the scope of this book!)

## Review Questions

---

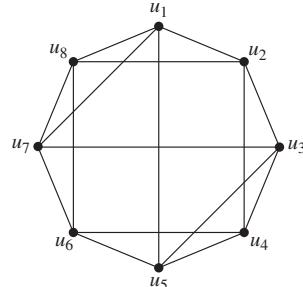
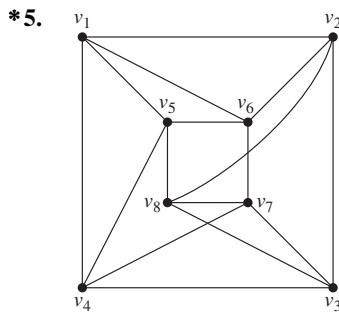
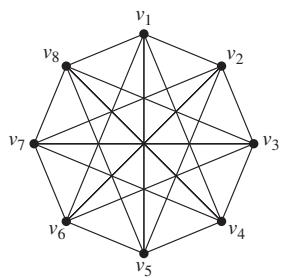
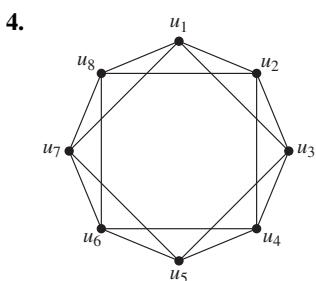
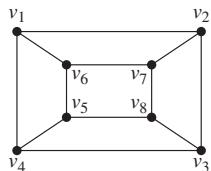
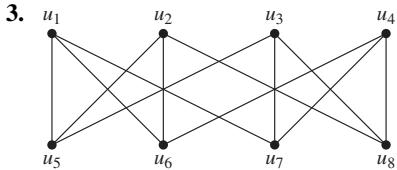
1. a) Define a simple graph, a multigraph, a pseudograph, a directed graph, and a directed multigraph.  
b) Use an example to show how each of the types of graph in part (a) can be used in modeling. For example, explain how to model different aspects of a computer network or airline routes.
2. Give at least four examples of how graphs are used in modeling.
3. What is the relationship between the sum of the degrees of the vertices in an undirected graph and the number of edges in this graph? Explain why this relationship holds.
4. Why must there be an even number of vertices of odd degree in an undirected graph?
5. What is the relationship between the sum of the in-degrees and the sum of the out-degrees of the vertices in a directed graph? Explain why this relationship holds.
6. Describe the following families of graphs.
  - a)  $K_n$ , the complete graph on  $n$  vertices
  - b)  $K_{m,n}$ , the complete bipartite graph on  $m$  and  $n$  vertices
  - c)  $C_n$ , the cycle with  $n$  vertices
  - d)  $W_n$ , the wheel of size  $n$
  - e)  $Q_n$ , the  $n$ -cube
7. How many vertices and how many edges are there in each of the graphs in the families in Question 6?
8. a) What is a bipartite graph?  
b) Which of the graphs  $K_n$ ,  $C_n$ , and  $W_n$  are bipartite?  
c) How can you determine whether an undirected graph is bipartite?
9. a) Describe three different methods that can be used to represent a graph.  
b) Draw a simple graph with at least five vertices and eight edges. Illustrate how it can be represented using the methods you described in part (a).
10. a) What does it mean for two simple graphs to be isomorphic?
- b) What is meant by an invariant with respect to isomorphism for simple graphs? Give at least five examples of such invariants.
- c) Give an example of two graphs that have the same numbers of vertices, edges, and degrees of vertices, but that are not isomorphic.
- d) Is a set of invariants known that can be used to efficiently determine whether two simple graphs are isomorphic?
11. a) What does it mean for a graph to be connected?  
b) What are the connected components of a graph?
12. a) Explain how an adjacency matrix can be used to represent a graph.  
b) How can adjacency matrices be used to determine whether a function from the vertex set of a graph  $G$  to the vertex set of a graph  $H$  is an isomorphism?  
c) How can the adjacency matrix of a graph be used to determine the number of paths of length  $r$ , where  $r$  is a positive integer, between two vertices of a graph?
13. a) Define an Euler circuit and an Euler path in an undirected graph.  
b) Describe the famous Königsberg bridge problem and explain how to rephrase it in terms of an Euler circuit.  
c) How can it be determined whether an undirected graph has an Euler path?  
d) How can it be determined whether an undirected graph has an Euler circuit?
14. a) Define a Hamilton circuit in a simple graph.  
b) Give some properties of a simple graph that imply that it does not have a Hamilton circuit.
15. Give examples of at least two problems that can be solved by finding the shortest path in a weighted graph.
16. a) Describe Dijkstra's algorithm for finding the shortest path in a weighted graph between two vertices.  
b) Draw a weighted graph with at least 10 vertices and 20 edges. Use Dijkstra's algorithm to find the shortest path between two vertices of your choice in the graph.

- 17.** a) What does it mean for a graph to be planar?  
 b) Give an example of a nonplanar graph.
- 18.** a) What is Euler's formula for connected planar graphs?  
 b) How can Euler's formula for planar graphs be used to show that a simple graph is nonplanar?
- 19.** State Kuratowski's theorem on the planarity of graphs and explain how it characterizes which graphs are planar.
- 20.** a) Define the chromatic number of a graph.
- b)** What is the chromatic number of the graph  $K_n$  when  $n$  is a positive integer?  
 c) What is the chromatic number of the graph  $C_n$  when  $n$  is an integer greater than 2?  
 d) What is the chromatic number of the graph  $K_{m,n}$  when  $m$  and  $n$  are positive integers?
- 21.** State the four color theorem. Are there graphs that cannot be colored with four colors?
- 22.** Explain how graph coloring can be used in modeling. Use at least two different examples.

## Supplementary Exercises

- 1.** How many edges does a 50-regular graph with 100 vertices have?
- 2.** How many nonisomorphic subgraphs does  $K_3$  have?

In Exercises 3–5 determine whether two given graphs are isomorphic.



The **complete  $m$ -partite graph**  $K_{n_1, n_2, \dots, n_m}$  has vertices partitioned into  $m$  subsets of  $n_1, n_2, \dots, n_m$  elements each, and vertices are adjacent if and only if they are in different subsets in the partition.

- 6.** Draw these graphs.  
 a)  $K_{1,2,3}$       b)  $K_{2,2,2}$       c)  $K_{1,2,2,3}$
- \*7.** How many vertices and how many edges does the complete  $m$ -partite graph  $K_{n_1, n_2, \dots, n_m}$  have?
- 8.** Prove or disprove that there are always two vertices of the same degree in a finite multigraph having at least two vertices.
- 9.** Let  $G = (V, E)$  be an undirected graph and let  $A \subseteq V$  and  $B \subseteq V$ . Show that  
 a)  $N(A \cup B) = N(A) \cup N(B)$ .  
 b)  $N(A \cap B) \subseteq N(A) \cap N(B)$ , and give an example where  $N(A \cap B) \neq N(A) \cap N(B)$ .

10. Let  $G = (V, E)$  be an undirected graph. Show that

- a)  $|N(v)| \leq \deg(v)$  for all  $v \in V$ .
- b)  $|N(v)| = \deg v$  for all  $v \in V$  if and only if  $G$  is a simple graph.

Suppose that  $S_1, S_2, \dots, S_n$  is a collection of subsets of a set  $S$  where  $n$  is a positive integer. A **system of distinct representatives (SDR)** for this family is an ordered  $n$ -tuple  $(a_1, a_2, \dots, a_n)$  with the property that  $a_i \in S_i$  for  $i = 1, 2, \dots, n$  and  $a_i \neq a_j$  for all  $i \neq j$ .

11. Find a SDR for the sets  $S_1 = \{a, c, m, e\}$ ,  $S_2 = \{m, a, c, e\}$ ,  $S_3 = \{a, p, e, x\}$ ,  $S_4 = \{x, e, n, a\}$ ,  $S_5 = \{n, a, m, e\}$ , and  $S_6 = \{e, x, a, m\}$ .

12. Use Hall's marriage theorem to show that a collection of finite subsets  $S_1, S_2, \dots, S_n$  of a set  $S$  has a SDR  $(a_1, a_2, \dots, a_n)$  if and only if  $|\bigcup_{i \in I} S_i| \geq |I|$  for all subsets  $I$  of  $\{1, 2, \dots, n\}$ .

13. a) Use Exercise 12 to show that the collection of sets  $S_1 = \{a, b, c\}$ ,  $S_2 = \{b, c, d\}$ ,  $S_3 = \{a, b, d\}$ ,  $S_4 = \{b, c, d\}$  has a SDR without finding one explicitly.

- b) Find a SDR for the family of four sets in part (a).

14. Use Exercise 12 to show that collection of sets  $S_1 = \{a, b, c\}$ ,  $S_2 = \{a, c\}$ ,  $S_3 = \{c, d, e\}$ ,  $S_4 = \{b, c\}$ ,  $S_5 = \{d, e, f\}$ ,  $S_6 = \{a, c, e\}$ , and  $S_7 = \{a, b\}$  does not have a SDR.

The **clustering coefficient**  $C(G)$  of a simple graph  $G$  is the probability that if  $u$  and  $v$  are neighbors and  $v$  and  $w$  are neighbors, then  $u$  and  $w$  are neighbors, where  $u$ ,  $v$ , and  $w$  are distinct vertices of  $G$ .

15. We say that three vertices  $u$ ,  $v$ , and  $w$  of a simple graph  $G$  form a triangle if there are edges connecting all three pairs of these vertices. Find a formula for  $C(G)$  in terms of the number of triangles in  $G$  and the number of paths of length two in the graph. [Hint: Count each triangle in the graph once for each order of three vertices that form it.]

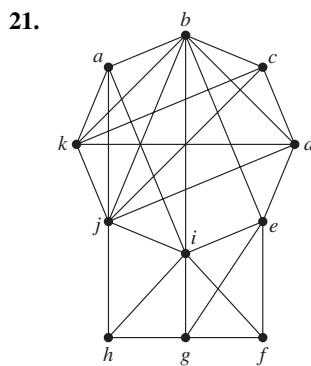
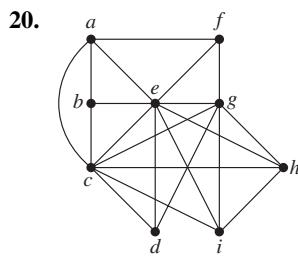
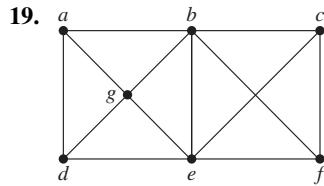
16. Find the clustering coefficient of each of the graphs in Exercise 20 of Section 10.2

17. Explain what the clustering coefficient measures in each of these graphs.

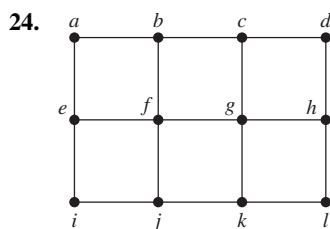
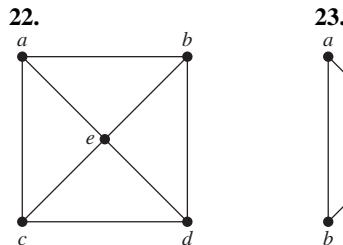
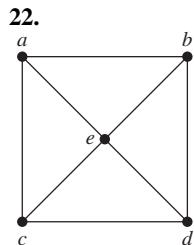
- a) the Hollywood graph
- b) the graph of Facebook friends
- c) the academic collaboration graph for researchers in graph theory
- d) the protein interaction graph for a human cell
- e) the graph representing the routers and communications links that make up the worldwide Internet

18. For each of the graphs in Exercise 17, explain whether you would expect its clustering coefficient to be closer to 0.01 or to 0.10 and why you expect this.

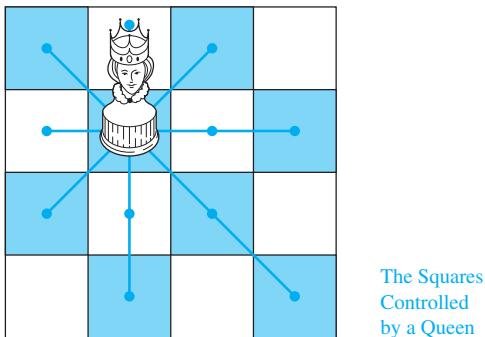
 A **clique** in a simple undirected graph is a complete subgraph that is not contained in any larger complete subgraph. In Exercises 19–21 find all cliques in the graph shown.



A **dominating set** of vertices in a simple graph is a set of vertices such that every other vertex is adjacent to at least one vertex of this set. A dominating set with the least number of vertices is called a **minimum dominating set**. In Exercises 22–24 find a minimum dominating set for the given graph.



A simple graph can be used to determine the minimum number of queens on a chessboard that control the entire chessboard. An  $n \times n$  chessboard has  $n^2$  squares in an  $n \times n$  configuration. A queen in a given position controls all squares in the same row, the same column, and on the two diagonals containing this square, as illustrated. The appropriate simple graph has  $n^2$  vertices, one for each square, and two vertices are adjacent if a queen in the square represented by one of the vertices controls the square represented by the other vertex.



25. Construct the simple graph representing the  $n \times n$  chessboard with edges representing the control of squares by queens for

a)  $n = 3$ .      b)  $n = 4$ .

26. Explain how the concept of a minimum dominating set applies to the problem of determining the minimum number of queens controlling an  $n \times n$  chessboard.

- \*\*27. Find the minimum number of queens controlling an  $n \times n$  chessboard for

a)  $n = 3$ .      b)  $n = 4$ .      c)  $n = 5$ .

28. Suppose that  $G_1$  and  $H_1$  are isomorphic and that  $G_2$  and  $H_2$  are isomorphic. Prove or disprove that  $G_1 \cup G_2$  and  $H_1 \cup H_2$  are isomorphic.

29. Show that each of these properties is an invariant that isomorphic simple graphs either both have or both do not have.

- a) connectedness
- b) the existence of a Hamilton circuit
- c) the existence of an Euler circuit
- d) having crossing number  $C$
- e) having  $n$  isolated vertices
- f) being bipartite

30. How can the adjacency matrix of  $\overline{G}$  be found from the adjacency matrix of  $G$ , where  $G$  is a simple graph?

31. How many nonisomorphic connected bipartite simple graphs are there with four vertices?

- \*32. How many nonisomorphic simple connected graphs with five vertices are there

- a) with no vertex of degree more than two?

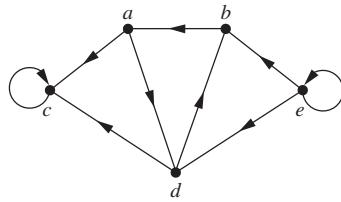
- b) with chromatic number equal to four?

- c) that are nonplanar?

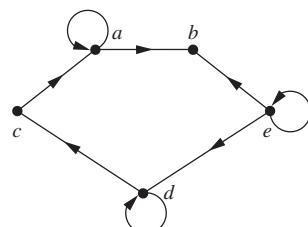
A directed graph is **self-converse** if it is isomorphic to its converse.

33. Determine whether the following graphs are self-converse.

a)



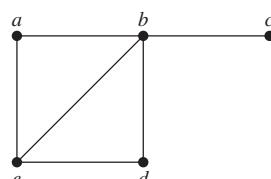
b)



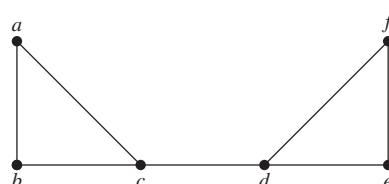
34. Show that if the directed graph  $G$  is self-converse and  $H$  is a directed graph isomorphic to  $G$ , then  $H$  is also self-converse.

An **orientation** of an undirected simple graph is an assignment of directions to its edges such that the resulting directed graph is strongly connected. When an orientation of an undirected graph exists, this graph is called **orientable**. In Exercises 35–37 determine whether the given simple graph is orientable.

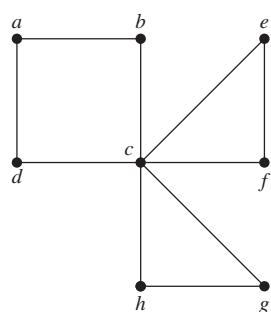
35.



36.



37.



- 38.** Because traffic is growing heavy in the central part of a city, traffic engineers are planning to change all the streets, which are currently two-way, into one-way streets. Explain how to model this problem.
- \*39.** Show that a graph is not orientable if it has a cut edge.
- A **tournament** is a simple directed graph such that if  $u$  and  $v$  are distinct vertices in the graph, exactly one of  $(u, v)$  and  $(v, u)$  is an edge of the graph.
- 40.** How many different tournaments are there with  $n$  vertices?
- 41.** What is the sum of the in-degree and out-degree of a vertex in a tournament?
- \*42.** Show that every tournament has a Hamilton path.
- 43.** Given two chickens in a flock, one of them is dominant. This defines the **pecking order** of the flock. How can a tournament be used to model pecking order?
- 44.** Suppose that a connected graph  $G$  has  $n$  vertices and vertex connectivity  $\kappa(G) = k$ . Show that  $G$  must have at least  $\lceil kn/2 \rceil$  edges.

A connected graph  $G = (V, E)$  with  $n$  vertices and  $m$  edges is said to have **optimal connectivity** if  $\kappa(G) = \lambda(G) = \min_{v \in V} \deg v = 2m/n$ .

- 45.** Show that a connected graph with optimal connectivity must be regular.
- 46.** Show these graphs have optimal connectivity.

- a)  $C_n$  for  $n \geq 3$
- b)  $K_n$  for  $n \geq 3$
- c)  $K_{r,r}$  for  $r \geq 2$

- \*47.** Find the two nonisomorphic simple graphs with six vertices and nine edges that have optimal connectivity.
- 48.** Suppose that  $G$  is a connected multigraph with  $2k$  vertices of odd degree. Show that there exist  $k$  subgraphs that have  $G$  as their union, where each of these subgraphs has an Euler path and where no two of these subgraphs have an edge in common. [Hint: Add  $k$  edges to the graph connecting pairs of vertices of odd degree and use an Euler circuit in this larger graph.]

In Exercises 49 and 50 we consider a puzzle posed by Petković in [Pe09] (based on a problem in [AvCh80]). Suppose that King Arthur has gathered his  $2n$  knights of the Round Table for an important council. Every two knights are either friends or enemies, and each knight has no more than  $n - 1$  enemies among the other  $2n - 1$  knights. The puzzle asks whether King Arthur can seat his knights around the Round Table so that each knight has two friends for his neighbors.

- 49. a)** Show that the puzzle can be reduced to determining whether there is a Hamilton circuit in the graph in which each knight is represented by a vertex and two knights are connected in the graph if they are friends.
- b)** Answer the question posed in the puzzle. [Hint: Use Dirac's theorem.]
- 50.** Suppose that are eight knights Alynore, Bedivere, De-gore, Gareth, Kay, Lancelot, Perceval, and Tristan. Their

lists of enemies are A (D, G, P), B (K, P, T), D (A, G, L), G (A, D, T), K (B, L, P), L (D, K, T), P (A, B, K), T (B, G, L), where we have represented each knight by the first letter of his name and shown the list of enemies of that knight following this first letter. Draw the graph representing these eight knight and their friends and find a seating arrangement where each knight sits next to two friends.

- \*51.** Let  $G$  be a simple graph with  $n$  vertices. The **bandwidth** of  $G$ , denoted by  $B(G)$ , is the minimum, over all permutations  $a_1, a_2, \dots, a_n$  of the vertices of  $G$ , of  $\max\{|i - j| \mid a_i$  and  $a_j$  are adjacent}. That is, the bandwidth is the minimum over all listings of the vertices of the maximum difference in the indices assigned to adjacent vertices. Find the bandwidths of these graphs.

- a)  $K_5$
- b)  $K_{1,3}$
- c)  $K_{2,3}$
- d)  $K_{3,3}$
- e)  $Q_3$
- f)  $C_5$

- \*52.** The **distance** between two distinct vertices  $v_1$  and  $v_2$  of a connected simple graph is the length (number of edges) of the shortest path between  $v_1$  and  $v_2$ . The **radius** of a graph is the minimum over all vertices  $v$  of the maximum distance from  $v$  to another vertex. The **diameter** of a graph is the maximum distance between two distinct vertices. Find the radius and diameter of

- a)  $K_6$ .
- b)  $K_{4,5}$ .
- c)  $Q_3$ .
- d)  $C_6$ .

- \*53. a)** Show that if the diameter of the simple graph  $G$  is at least four, then the diameter of its complement  $\overline{G}$  is no more than two.
- b)** Show that if the diameter of the simple graph  $G$  is at least three, then the diameter of its complement  $\overline{G}$  is no more than three.

- \*54.** Suppose that a multigraph has  $2m$  vertices of odd degree. Show that any circuit that contains every edge of the graph must contain at least  $m$  edges more than once.

- 55.** Find the second shortest path between the vertices  $a$  and  $z$  in Figure 3 of Section 10.6.

- 56.** Devise an algorithm for finding the second shortest path between two vertices in a simple connected weighted graph.

- 57.** Find the shortest path between the vertices  $a$  and  $z$  that passes through the vertex  $f$  in the weighted graph in Exercise 3 in Section 10.6.

- 58.** Devise an algorithm for finding the shortest path between two vertices in a simple connected weighted graph that passes through a specified third vertex.

- \*59.** Show that if  $G$  is a simple graph with at least 11 vertices, then either  $G$  or  $\overline{G}$ , the complement of  $G$ , is nonplanar.

 A set of vertices in a graph is called **independent** if no two vertices in the set are adjacent. The **independence number** of a graph is the maximum number of vertices in an independent set of vertices for the graph.

- \*60.** What is the independence number of

- a)  $K_n$ ?
- b)  $C_n$ ?
- c)  $Q_n$ ?
- d)  $K_{m,n}$ ?

61. Show that the number of vertices in a simple graph is less than or equal to the product of the independence number and the chromatic number of the graph.
62. Show that the chromatic number of a graph is less than or equal to  $n - i + 1$ , where  $n$  is the number of vertices in the graph and  $i$  is the independence number of this graph.
63. Suppose that to generate a random simple graph with  $n$  vertices we first choose a real number  $p$  with  $0 \leq p \leq 1$ . For each of the  $C(n, 2)$  pairs of distinct vertices we generate a random number  $x$  between 0 and 1. If  $0 \leq x \leq p$ , we connect these two vertices with an edge; otherwise these vertices are not connected.
  - a) What is the probability that a graph with  $m$  edges where  $0 \leq m \leq C(n, 2)$  is generated?
  - b) What is the expected number of edges in a randomly generated graph with  $n$  vertices if each edge is included with probability  $p$ ?
  - c) Show that if  $p = 1/2$  then every simple graph with  $n$  vertices is equally likely to be generated.

A property retained whenever additional edges are added to a simple graph (without adding vertices) is called **monotone increasing**, and a property that is retained whenever edges are

removed from a simple graph (without removing vertices) is called **monotone decreasing**.

64. For each of these properties, determine whether it is monotone increasing and determine whether it is monotone decreasing.
  - a) The graph  $G$  is connected.
  - b) The graph  $G$  is not connected.
  - c) The graph  $G$  has an Euler circuit.
  - d) The graph  $G$  has a Hamilton circuit.
  - e) The graph  $G$  is planar.
  - f) The graph  $G$  has chromatic number four.
  - g) The graph  $G$  has radius three.
  - h) The graph  $G$  has diameter three.
65. Show that the graph property  $P$  is monotone increasing if and only if the graph property  $Q$  is monotone decreasing where  $Q$  is the property of not having property  $P$ .
- \*\*66. Suppose that  $P$  is a monotone increasing property of simple graphs. Show that the probability a random graph with  $n$  vertices has property  $P$  is a monotonic nondecreasing function of  $p$ , the probability an edge is chosen to be in the graph.

## Computer Projects

---

**Write programs with these input and output.**

1. Given the vertex pairs associated to the edges of an undirected graph, find the degree of each vertex.
2. Given the ordered pairs of vertices associated to the edges of a directed graph, determine the in-degree and out-degree of each vertex.
3. Given the list of edges of a simple graph, determine whether the graph is bipartite.
4. Given the vertex pairs associated to the edges of a graph, construct an adjacency matrix for the graph. (Produce a version that works when loops, multiple edges, or directed edges are present.)
5. Given an adjacency matrix of a graph, list the edges of this graph and give the number of times each edge appears.
6. Given the vertex pairs associated to the edges of an undirected graph and the number of times each edge appears, construct an incidence matrix for the graph.
7. Given an incidence matrix of an undirected graph, list its edges and give the number of times each edge appears.
8. Given a positive integer  $n$ , generate a simple graph with  $n$  vertices by producing an adjacency matrix for the graph so that all simple graphs with  $n$  vertices are equally likely to be generated.
9. Given a positive integer  $n$ , generate a simple directed graph with  $n$  vertices by producing an adjacency matrix for the graph so that all simple directed graphs with  $n$  vertices are equally likely to be generated.
10. Given the lists of edges of two simple graphs with no more than six vertices, determine whether the graphs are isomorphic.
11. Given an adjacency matrix of a graph and a positive integer  $n$ , find the number of paths of length  $n$  between two vertices. (Produce a version that works for directed and undirected graphs.)
- \*12. Given the list of edges of a simple graph, determine whether it is connected and find the number of connected components if it is not connected.
13. Given the vertex pairs associated to the edges of a multigraph, determine whether it has an Euler circuit and, if not, whether it has an Euler path. Construct an Euler path or circuit if it exists.
- \*14. Given the ordered pairs of vertices associated to the edges of a directed multigraph, construct an Euler path or Euler circuit, if such a path or circuit exists.
- \*\*15. Given the list of edges of a simple graph, produce a Hamilton circuit, or determine that the graph does not have such a circuit.
- \*\*16. Given the list of edges of a simple graph, produce a Hamilton path, or determine that the graph does not have such a path.
17. Given the list of edges and weights of these edges of a weighted connected simple graph and two vertices in this graph, find the length of a shortest path between them using Dijkstra's algorithm. Also, find a shortest path.

- 18.** Given the list of edges of an undirected graph, find a coloring of this graph using the algorithm given in the exercise set of Section 10.8.
- 19.** Given a list of students and the courses that they are enrolled in, construct a schedule of final exams.
- 20.** Given the distances between pairs of television stations and the minimum allowable distance between stations, assign frequencies to these stations.

## Computations and Explorations

Use a computational program or programs you have written to do these exercises.

1. Display all simple graphs with four vertices.
2. Display a full set of nonisomorphic simple graphs with six vertices.
3. Display a full set of nonisomorphic directed graphs with four vertices.
4. Generate at random 10 different simple graphs each with 20 vertices so that each such graph is equally likely to be generated.
5. Construct a Gray code where the code words are bit strings of length six.
6. Construct knight's tours on chessboards of various sizes.
7. Determine whether each of the graphs you generated in Exercise 4 of this set is planar. If you can, determine the thickness of each of the graphs that are not planar.
8. Determine whether each of the graphs you generated in Exercise 4 of this set is connected. If a graph is not connected, determine the number of connected components of the graph.
9. Generate at random simple graphs with 10 vertices. Stop when you have constructed one with an Euler circuit. Display an Euler circuit in this graph.
10. Generate at random simple graphs with 10 vertices. Stop when you have constructed one with a Hamilton circuit. Display a Hamilton circuit in this graph.
11. Find the chromatic number of each of the graphs you generated in Exercise 4 of this set.
- \*\*12. Find the shortest path a traveling salesperson can take to visit each of the capitals of the 50 states in the United States, traveling by air between cities in a straight line.
- \*13. Estimate the probability that a randomly generated simple graph with  $n$  vertices is connected for each positive integer  $n$  not exceeding ten by generating a set of random simple graphs and determining whether each is connected.
- \*\*14. Work on the problem of determining whether the crossing number of  $K_{7,7}$  is 77, 79, or 81. It is known that it equals one of these three values.

## Writing Projects

Respond to these with essays using outside sources.

1. Describe the origins and development of graph theory prior to the year 1900.
2. Discuss the applications of graph theory to the study of ecosystems.
3. Discuss the applications of graph theory to sociology and psychology.
4. Discuss what can be learned by investigating the properties of the Web graph.
5. Explain what community structure is in a graph representing a network, such as a social network, a computer network, an information network, or a biological network. Define what a community in such a graph is, and explain what communities represent in graphs representing the types of networks listed.
6. Describe some of the algorithms used to detect communities in graphs representing networks of the types listed in Question 5.
7. Describe algorithms for drawing a graph on paper or on a display given the vertices and edges of the graph. What considerations arise in drawing a graph so that it has the best appearance for understanding its properties?
8. Explain how graph theory can help uncover networks of criminals or terrorists by studying relevant social and communication networks.
9. What are some of the capabilities that a software tool for inputting, displaying, and manipulating graphs should have? Which of these capabilities do available tools have?
10. Describe some of the algorithms available for determining whether two graphs are isomorphic and the computational complexity of these algorithms. What is the most efficient such algorithm currently known?
11. What is the subgraph isomorphism problem and what are some of its important applications, including those to chemistry, bioinformatics, electronic circuit design, and computer vision?
12. Explain what the area of graph mining, an important area of data mining, is and describe some of the basic techniques used in graph mining.

13. Describe how Euler paths can be used to help determine DNA sequences.
14. Define *de Bruijn sequences* and discuss how they arise in applications. Explain how de Bruijn sequences can be constructed using Euler circuits.
15. Describe the *Chinese postman problem* and explain how to solve this problem.
16. Describe some of the different conditions that imply that a graph has a Hamilton circuit.
17. Describe some of the strategies and algorithms used to solve the traveling salesperson problem.
18. Describe several different algorithms for determining whether a graph is planar. What is the computational complexity of each of these algorithms?
19. In modeling, very large scale integration (VLSI) graphs are sometimes embedded in a book, with the vertices on the spine and the edges on pages. Define the *book number* of a graph and find the book number of various graphs including  $K_n$  for  $n = 3, 4, 5$ , and 6.
20. Discuss the history of the four color theorem.
21. Describe the role computers played in the proof of the four color theorem. How can we be sure that a proof that relies on a computer is correct?
22. Describe and compare several different algorithms for coloring a graph, in terms of whether they produce a coloring with the least number of colors possible and in terms of their complexity.
23. Explain how graph multicolorings can be used in a variety of different models.
24. Describe some of the applications of edge colorings.
25. Explain how the theory of random graphs can be used in nonconstructive existence proofs of graphs with certain properties.

# 11

# Trees

## 11.1 Introduction to Trees

## 11.2 Applications of Trees

## 11.3 Tree Traversal

## 11.4 Spanning Trees

## 11.5 Minimum Spanning Trees

A connected graph that contains no simple circuits is called a tree. Trees were used as long ago as 1857, when the English mathematician Arthur Cayley used them to count certain types of chemical compounds. Since that time, trees have been employed to solve problems in a wide variety of disciplines, as the examples in this chapter will show.

Trees are particularly useful in computer science, where they are employed in a wide range of algorithms. For instance, trees are used to construct efficient algorithms for locating items in a list. They can be used in algorithms, such as Huffman coding, that construct efficient codes saving costs in data transmission and storage. Trees can be used to study games such as checkers and chess and can help determine winning strategies for playing these games. Trees can be used to model procedures carried out using a sequence of decisions. Constructing these models can help determine the computational complexity of algorithms based on a sequence of decisions, such as sorting algorithms.

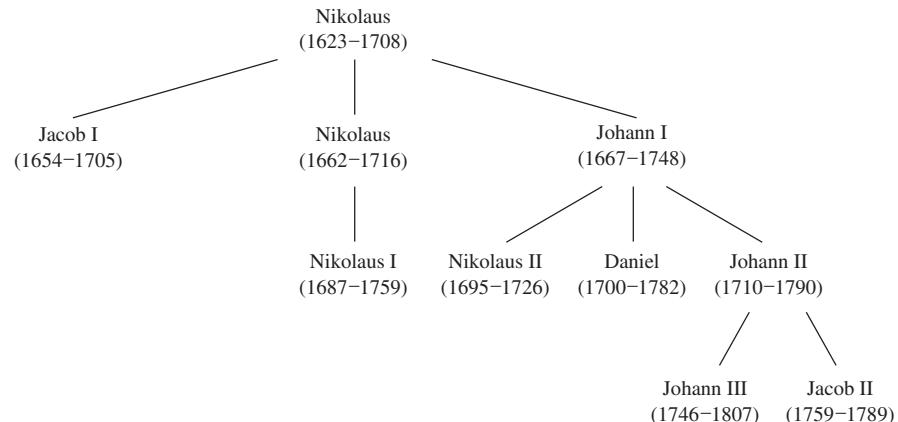
Procedures for building trees containing every vertex of a graph, including depth-first search and breadth-first search, can be used to systematically explore the vertices of a graph. Exploring the vertices of a graph via depth-first search, also known as backtracking, allows for the systematic search for solutions to a wide variety of problems, such as determining how eight queens can be placed on a chessboard so that no queen can attack another.

We can assign weights to the edges of a tree to model many problems. For example, using weighted trees we can develop algorithms to construct networks containing the least expensive set of telephone lines linking different network nodes.

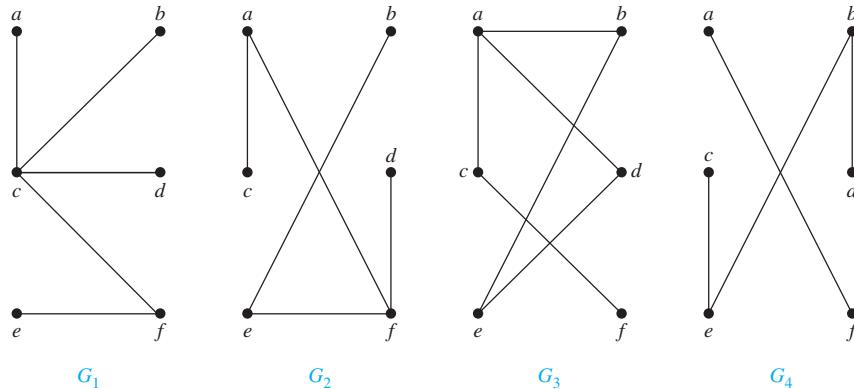
## 11.1 Introduction to Trees



In Chapter 10 we showed how graphs can be used to model and solve many problems. In this chapter we will focus on a particular type of graph called a **tree**, so named because such graphs resemble trees. For example, *family trees* are graphs that represent genealogical charts. Family trees use vertices to represent the members of a family and edges to represent parent-child relationships. The family tree of the male members of the Bernoulli family of Swiss mathematicians is shown in Figure 1. The undirected graph representing a family tree (restricted to people of just one gender and with no inbreeding) is an example of a tree.



**FIGURE 1** The Bernoulli Family of Mathematicians.



**FIGURE 2 Examples of Trees and Graphs That Are Not Trees.**

### DEFINITION 1

A **tree** is a connected undirected graph with no simple circuits.

Because a tree cannot have a simple circuit, a tree cannot contain multiple edges or loops. Therefore any tree must be a simple graph.

**EXAMPLE 1** Which of the graphs shown in Figure 2 are trees?

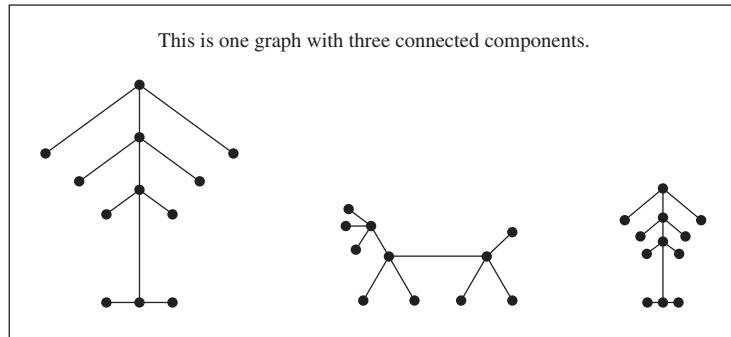
**Solution:**  $G_1$  and  $G_2$  are trees, because both are connected graphs with no simple circuits.  $G_3$  is not a tree because  $e, b, a, d, e$  is a simple circuit in this graph. Finally,  $G_4$  is not a tree because it is not connected. ◀

Any connected graph that contains no simple circuits is a tree. What about graphs containing no simple circuits that are not necessarily connected? These graphs are called **forests** and have the property that each of their connected components is a tree. Figure 3 displays a forest.

Trees are often defined as undirected graphs with the property that there is a unique simple path between every pair of vertices. Theorem 1 shows that this alternative definition is equivalent to our definition.

### THEOREM 1

An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices.



**FIGURE 3 Example of a Forest.**

**Proof:** First assume that  $T$  is a tree. Then  $T$  is a connected graph with no simple circuits. Let  $x$  and  $y$  be two vertices of  $T$ . Because  $T$  is connected, by Theorem 1 of Section 10.4 there is a simple path between  $x$  and  $y$ . Moreover, this path must be unique, for if there were a second such path, the path formed by combining the first path from  $x$  to  $y$  followed by the path from  $y$  to  $x$  obtained by reversing the order of the second path from  $x$  to  $y$  would form a circuit. This implies, using Exercise 59 of Section 10.4, that there is a simple circuit in  $T$ . Hence, there is a unique simple path between any two vertices of a tree.

Now assume that there is a unique simple path between any two vertices of a graph  $T$ . Then  $T$  is connected, because there is a path between any two of its vertices. Furthermore,  $T$  can have no simple circuits. To see that this is true, suppose  $T$  had a simple circuit that contained the vertices  $x$  and  $y$ . Then there would be two simple paths between  $x$  and  $y$ , because the simple circuit is made up of a simple path from  $x$  to  $y$  and a second simple path from  $y$  to  $x$ . Hence, a graph with a unique simple path between any two vertices is a tree.  $\triangle$

## Rooted Trees

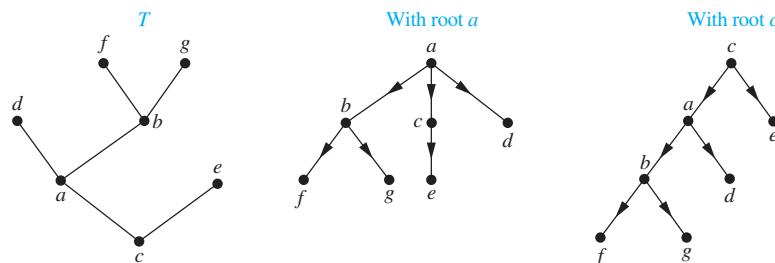
In many applications of trees, a particular vertex of a tree is designated as the **root**. Once we specify a root, we can assign a direction to each edge as follows. Because there is a unique path from the root to each vertex of the graph (by Theorem 1), we direct each edge away from the root. Thus, a tree together with its root produces a directed graph called a **rooted tree**.

### DEFINITION 2

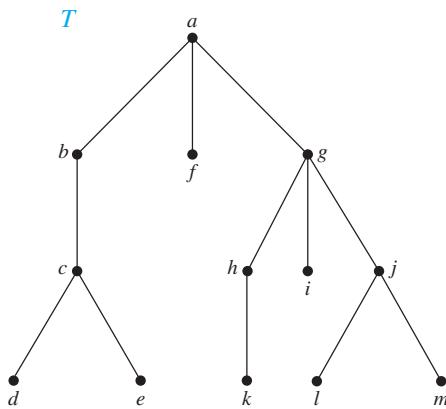
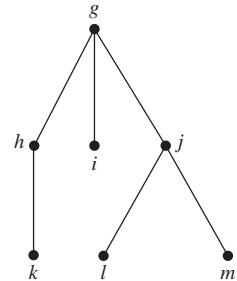
A *rooted tree* is a tree in which one vertex has been designated as the root and every edge is directed away from the root.

Rooted trees can also be defined recursively. Refer to Section 5.3 to see how this can be done. We can change an unrooted tree into a rooted tree by choosing any vertex as the root. Note that different choices of the root produce different rooted trees. For instance, Figure 4 displays the rooted trees formed by designating  $a$  to be the root and  $c$  to be the root, respectively, in the tree  $T$ . We usually draw a rooted tree with its root at the top of the graph. The arrows indicating the directions of the edges in a rooted tree can be omitted, because the choice of root determines the directions of the edges.

The terminology for trees has botanical and genealogical origins. Suppose that  $T$  is a rooted tree. If  $v$  is a vertex in  $T$  other than the root, the **parent** of  $v$  is the unique vertex  $u$  such that there is a directed edge from  $u$  to  $v$  (the reader should show that such a vertex is unique). When  $u$  is the parent of  $v$ ,  $v$  is called a **child** of  $u$ . Vertices with the same parent are called **siblings**. The **ancestors** of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root (that is, its parent, its parent's parent, and so on, until the root is reached). The **descendants** of a vertex  $v$  are those vertices that have  $v$  as



**FIGURE 4** A Tree and Rooted Trees Formed by Designating Two Different Roots.

FIGURE 5 A Rooted Tree  $T$ .FIGURE 6 The Subtree Rooted at  $g$ .

an ancestor. A vertex of a rooted tree is called a **leaf** if it has no children. Vertices that have children are called **internal vertices**. The root is an internal vertex unless it is the only vertex in the graph, in which case it is a leaf.

If  $a$  is a vertex in a tree, the **subtree** with  $a$  as its root is the subgraph of the tree consisting of  $a$  and its descendants and all edges incident to these descendants.

**EXAMPLE 2**

In the rooted tree  $T$  (with root  $a$ ) shown in Figure 5, find the parent of  $c$ , the children of  $g$ , the siblings of  $h$ , all ancestors of  $e$ , all descendants of  $b$ , all internal vertices, and all leaves. What is the subtree rooted at  $g$ ?



**Solution:** The parent of  $c$  is  $b$ . The children of  $g$  are  $h$ ,  $i$ , and  $j$ . The siblings of  $h$  are  $i$  and  $j$ . The ancestors of  $e$  are  $c$ ,  $b$ , and  $a$ . The descendants of  $b$  are  $c$ ,  $d$ , and  $e$ . The internal vertices are  $a$ ,  $b$ ,  $c$ ,  $g$ ,  $h$ , and  $j$ . The leaves are  $d$ ,  $e$ ,  $f$ ,  $i$ ,  $k$ ,  $l$ , and  $m$ . The subtree rooted at  $g$  is shown in Figure 6.

Rooted trees with the property that all of their internal vertices have the same number of children are used in many different applications. Later in this chapter we will use such trees to study problems involving searching, sorting, and coding.

**DEFINITION 3**

A rooted tree is called an  *$m$ -ary tree* if every internal vertex has no more than  $m$  children. The tree is called a *full  $m$ -ary tree* if every internal vertex has exactly  $m$  children. An  $m$ -ary tree with  $m = 2$  is called a *binary tree*.

**EXAMPLE 3**

Are the rooted trees in Figure 7 full  $m$ -ary trees for some positive integer  $m$ ?

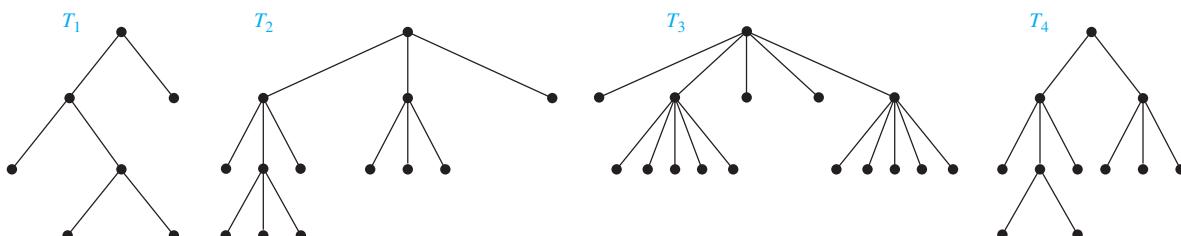


FIGURE 7 Four Rooted Trees.

**Solution:**  $T_1$  is a full binary tree because each of its internal vertices has two children.  $T_2$  is a full 3-ary tree because each of its internal vertices has three children. In  $T_3$  each internal vertex has five children, so  $T_3$  is a full 5-ary tree.  $T_4$  is not a full  $m$ -ary tree for any  $m$  because some of its internal vertices have two children and others have three children.  $\blacktriangleleft$

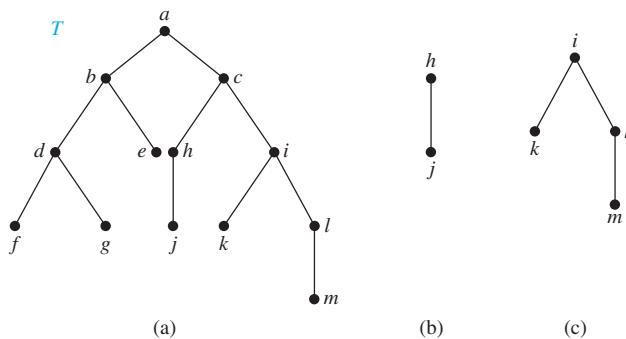
**ORDERED ROOTED TREES** An **ordered rooted tree** is a rooted tree where the children of each internal vertex are ordered. Ordered rooted trees are drawn so that the children of each internal vertex are shown in order from left to right. Note that a representation of a rooted tree in the conventional way determines an ordering for its edges. We will use such orderings of edges in drawings without explicitly mentioning that we are considering a rooted tree to be ordered.

In an ordered binary tree (usually called just a **binary tree**), if an internal vertex has two children, the first child is called the **left child** and the second child is called the **right child**. The tree rooted at the left child of a vertex is called the **left subtree** of this vertex, and the tree rooted at the right child of a vertex is called the **right subtree** of the vertex. The reader should note that for some applications every vertex of a binary tree, other than the root, is designated as a right or a left child of its parent. This is done even when some vertices have only one child. We will make such designations whenever it is necessary, but not otherwise.

Ordered rooted trees can be defined recursively. Binary trees, a type of ordered rooted trees, were defined this way in Section 5.3.

**EXAMPLE 4** What are the left and right children of  $d$  in the binary tree  $T$  shown in Figure 8(a) (where the order is that implied by the drawing)? What are the left and right subtrees of  $c$ ?

**Solution:** The left child of  $d$  is  $f$  and the right child is  $g$ . We show the left and right subtrees of  $c$  in Figures 8(b) and 8(c), respectively.  $\blacktriangleleft$

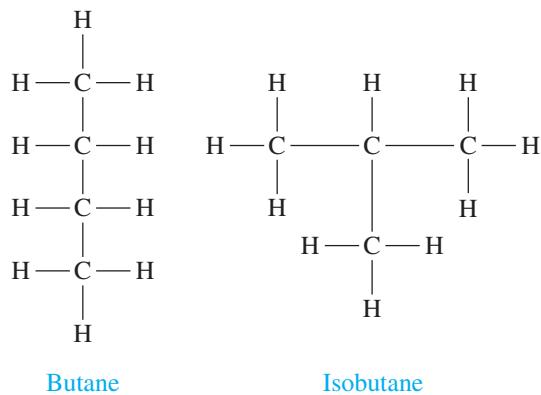


**FIGURE 8** A Binary Tree  $T$  and Left and Right Subtrees of the Vertex  $c$ .

Just as in the case of graphs, there is no standard terminology used to describe trees, rooted trees, ordered rooted trees, and binary trees. This nonstandard terminology occurs because trees are used extensively throughout computer science, which is a relatively young field. The reader should carefully check meanings given to terms dealing with trees whenever they occur.

## Trees as Models

Trees are used as models in such diverse areas as computer science, chemistry, geology, botany, and psychology. We will describe a variety of such models based on trees.

**FIGURE 9** The Two Isomers of Butane.**EXAMPLE 5**

**Saturated Hydrocarbons and Trees** Graphs can be used to represent molecules, where atoms are represented by vertices and bonds between them by edges. The English mathematician Arthur Cayley discovered trees in 1857 when he was trying to enumerate the isomers of compounds of the form  $C_nH_{2n+2}$ , which are called *saturated hydrocarbons*.

In graph models of saturated hydrocarbons, each carbon atom is represented by a vertex of degree 4, and each hydrogen atom is represented by a vertex of degree 1. There are  $3n + 2$  vertices in a graph representing a compound of the form  $C_nH_{2n+2}$ . The number of edges in such a graph is half the sum of the degrees of the vertices. Hence, there are  $(4n + 2n + 2)/2 = 3n + 1$  edges in this graph. Because the graph is connected and the number of edges is one less than the number of vertices, it must be a tree (see Exercise 15).

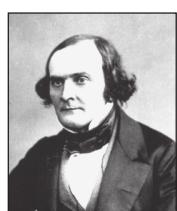
The nonisomorphic trees with  $n$  vertices of degree 4 and  $2n + 2$  of degree 1 represent the different isomers of  $C_nH_{2n+2}$ . For instance, when  $n = 4$ , there are exactly two nonisomorphic trees of this type (the reader should verify this). Hence, there are exactly two different isomers of  $C_4H_{10}$ . Their structures are displayed in Figure 9. These two isomers are called butane and isobutane. ▲

**EXAMPLE 6**

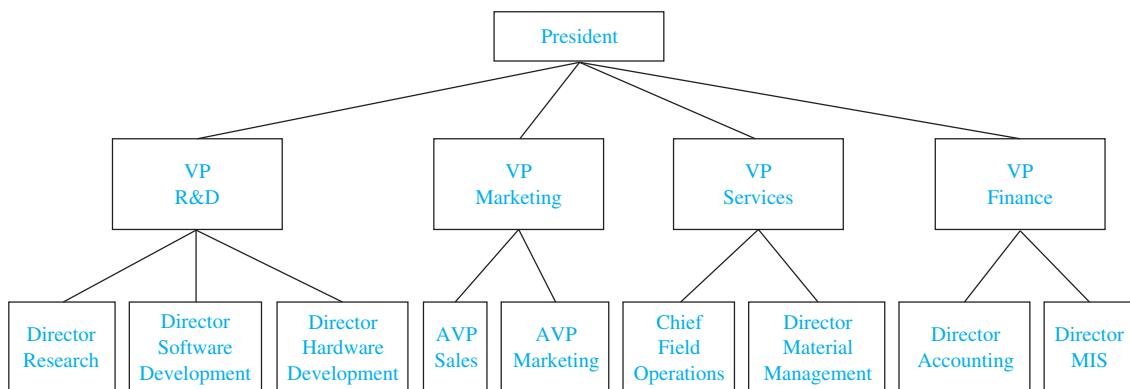
**Representing Organizations** The structure of a large organization can be modeled using a rooted tree. Each vertex in this tree represents a position in the organization. An edge from one vertex to another indicates that the person represented by the initial vertex is the (direct) boss of the person represented by the terminal vertex. The graph shown in Figure 10 displays such a tree. In the organization represented by this tree, the Director of Hardware Development works directly for the Vice President of R&D. ▲

**EXAMPLE 7**

**Computer File Systems** Files in computer memory can be organized into directories. A directory can contain both files and subdirectories. The root directory contains the entire file

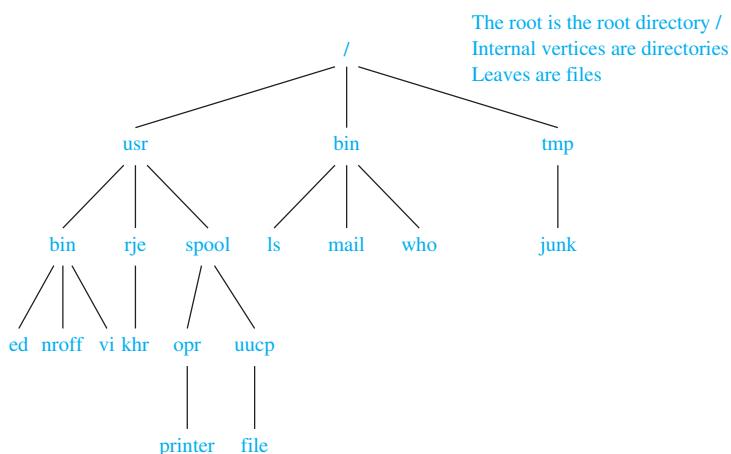


**ARTHUR CAYLEY (1821–1895)** Arthur Cayley, the son of a merchant, displayed his mathematical talents at an early age with amazing skill in numerical calculations. Cayley entered Trinity College, Cambridge, when he was 17. While in college he developed a passion for reading novels. Cayley excelled at Cambridge and was elected to a 3-year appointment as Fellow of Trinity and assistant tutor. During this time Cayley began his study of  $n$ -dimensional geometry and made a variety of contributions to geometry and to analysis. He also developed an interest in mountaineering, which he enjoyed during vacations in Switzerland. Because no position as a mathematician was available to him, Cayley left Cambridge, entering the legal profession and gaining admittance to the bar in 1849. Although Cayley limited his legal work to be able to continue his mathematics research, he developed a reputation as a legal specialist. During his legal career he was able to write more than 300 mathematical papers. In 1863 Cambridge University established a new post in mathematics and offered it to Cayley. He took this job, even though it paid less money than he made as a lawyer.



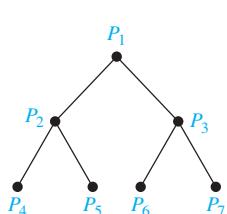
**FIGURE 10** An Organizational Tree for a Computer Company.

system. Thus, a file system may be represented by a rooted tree, where the root represents the root directory, internal vertices represent subdirectories, and leaves represent ordinary files or empty directories. One such file system is shown in Figure 11. In this system, the file khr is in the directory rje. (Note that links to files where the same file may have more than one pathname can lead to circuits in computer file systems.)



**FIGURE 11** A Computer File System.

### EXAMPLE 8



**FIGURE 12** A Tree-Connected Network of Seven Processors.

**Tree-Connected Parallel Processors** In Example 17 of Section 10.2 we described several interconnection networks for parallel processing. A **tree-connected network** is another important way to interconnect processors. The graph representing such a network is a complete binary tree, that is, a full binary tree where every root is at the same level. Such a network interconnects  $n = 2^k - 1$  processors, where  $k$  is a positive integer. A processor represented by the vertex  $v$  that is not a root or a leaf has three two-way connections—one to the processor represented by the parent of  $v$  and two to the processors represented by the two children of  $v$ . The processor represented by the root has two two-way connections to the processors represented by its two children. A processor represented by a leaf  $v$  has a single two-way connection to the parent of  $v$ . We display a tree-connected network with seven processors in Figure 12.

We now illustrate how a tree-connected network can be used for parallel computation. In particular, we show how the processors in Figure 12 can be used to add eight numbers, using three steps. In the first step, we add  $x_1$  and  $x_2$  using  $P_4$ ,  $x_3$  and  $x_4$  using  $P_5$ ,  $x_5$  and  $x_6$  using  $P_6$ ,

and  $x_7$  and  $x_8$  using  $P_7$ . In the second step, we add  $x_1 + x_2$  and  $x_3 + x_4$  using  $P_2$  and  $x_5 + x_6$  and  $x_7 + x_8$  using  $P_3$ . Finally, in the third step, we add  $x_1 + x_2 + x_3 + x_4$  and  $x_5 + x_6 + x_7 + x_8$  using  $P_1$ . The three steps used to add eight numbers compares favorably to the seven steps required to add eight numbers serially, where the steps are the addition of one number to the sum of the previous numbers in the list.  $\blacktriangleleft$

## Properties of Trees

We will often need results relating the numbers of edges and vertices of various types in trees.

### THEOREM 2

A tree with  $n$  vertices has  $n - 1$  edges.



**Proof:** We will use mathematical induction to prove this theorem. Note that for all the trees here we can choose a root and consider the tree rooted.

**BASIS STEP:** When  $n = 1$ , a tree with  $n = 1$  vertex has no edges. It follows that the theorem is true for  $n = 1$ .

**INDUCTIVE STEP:** The inductive hypothesis states that every tree with  $k$  vertices has  $k - 1$  edges, where  $k$  is a positive integer. Suppose that a tree  $T$  has  $k + 1$  vertices and that  $v$  is a leaf of  $T$  (which must exist because the tree is finite), and let  $w$  be the parent of  $v$ . Removing from  $T$  the vertex  $v$  and the edge connecting  $w$  to  $v$  produces a tree  $T'$  with  $k$  vertices, because the resulting graph is still connected and has no simple circuits. By the inductive hypothesis,  $T'$  has  $k - 1$  edges. It follows that  $T$  has  $k$  edges because it has one more edge than  $T'$ , the edge connecting  $v$  and  $w$ . This completes the inductive step.  $\blacktriangleleft$

Recall that a tree is a connected undirected graph with no simple circuits. So, when  $G$  is an undirected graph with  $n$  vertices, Theorem 2 tells us that the two conditions (i)  $G$  is connected and (ii)  $G$  has no simple circuits, imply (iii)  $G$  has  $n - 1$  edges. Also, when (i) and (iii) hold, then (ii) must also hold, and when (ii) and (iii) hold, (i) must also hold. That is, if  $G$  is connected and  $G$  has  $n - 1$  edges, then  $G$  has no simple circuits, so that  $G$  is a tree (see Exercise 15(a)), and if  $G$  has no simple circuits and  $G$  has  $n - 1$  edges, then  $G$  is connected, and so is a tree (see Exercise 15(b)). Consequently, when two of (i), (ii), and (iii) hold, the third condition must also hold, and  $G$  must be a tree.

**COUNTING VERTICES IN FULL  $m$ -ARY TREES** The number of vertices in a full  $m$ -ary tree with a specified number of internal vertices is determined, as Theorem 3 shows. As in Theorem 2, we will use  $n$  to denote the number of vertices in a tree.

### THEOREM 3

A full  $m$ -ary tree with  $i$  internal vertices contains  $n = mi + 1$  vertices.

**Proof:** Every vertex, except the root, is the child of an internal vertex. Because each of the  $i$  internal vertices has  $m$  children, there are  $mi$  vertices in the tree other than the root. Therefore, the tree contains  $n = mi + 1$  vertices.  $\blacktriangleleft$

Suppose that  $T$  is a full  $m$ -ary tree. Let  $i$  be the number of internal vertices and  $l$  the number of leaves in this tree. Once one of  $n$ ,  $i$ , and  $l$  is known, the other two quantities are determined. Theorem 4 explains how to find the other two quantities from the one that is known.

**THEOREM 4**

A full  $m$ -ary tree with

- (i)  $n$  vertices has  $i = (n - 1)/m$  internal vertices and  $l = [(m - 1)n + 1]/m$  leaves,
- (ii)  $i$  internal vertices has  $n = mi + 1$  vertices and  $l = (m - 1)i + 1$  leaves,
- (iii)  $l$  leaves has  $n = (ml - 1)/(m - 1)$  vertices and  $i = (l - 1)/(m - 1)$  internal vertices.

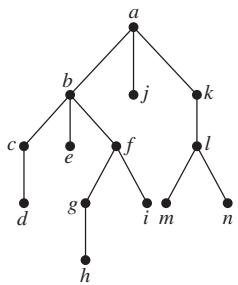
**Proof:** Let  $n$  represent the number of vertices,  $i$  the number of internal vertices, and  $l$  the number of leaves. The three parts of the theorem can all be proved using the equality given in Theorem 3, that is,  $n = mi + 1$ , together with the equality  $n = l + i$ , which is true because each vertex is either a leaf or an internal vertex. We will prove part (i) here. The proofs of parts (ii) and (iii) are left as exercises for the reader.

Solving for  $i$  in  $n = mi + 1$  gives  $i = (n - 1)/m$ . Then inserting this expression for  $i$  into the equation  $n = l + i$  shows that  $l = n - i = n - (n - 1)/m = [(m - 1)n + 1]/m$ .  $\triangleleft$

Example 9 illustrates how Theorem 4 can be used.

**EXAMPLE 9**

Suppose that someone starts a chain letter. Each person who receives the letter is asked to send it on to four other people. Some people do this, but others do not send any letters. How many people have seen the letter, including the first person, if no one receives more than one letter and if the chain letter ends after there have been 100 people who read it but did not send it out? How many people sent out the letter?



**FIGURE 13** A Rooted Tree.

**Solution:** The chain letter can be represented using a 4-ary tree. The internal vertices correspond to people who sent out the letter, and the leaves correspond to people who did not send it out. Because 100 people did not send out the letter, the number of leaves in this rooted tree is  $l = 100$ . Hence, part (iii) of Theorem 4 shows that the number of people who have seen the letter is  $n = (4 \cdot 100 - 1)/(4 - 1) = 133$ . Also, the number of internal vertices is  $133 - 100 = 33$ , so 33 people sent out the letter.  $\triangleleft$

**BALANCED  $m$ -ARY TREES** It is often desirable to use rooted trees that are “balanced” so that the subtrees at each vertex contain paths of approximately the same length. Some definitions will make this concept clear. The **level** of a vertex  $v$  in a rooted tree is the length of the unique path from the root to this vertex. The level of the root is defined to be zero. The **height** of a rooted tree is the maximum of the levels of vertices. In other words, the height of a rooted tree is the length of the longest path from the root to any vertex.

**EXAMPLE 10**

Find the level of each vertex in the rooted tree shown in Figure 13. What is the height of this tree?

**Solution:** The root  $a$  is at level 0. Vertices  $b$ ,  $j$ , and  $k$  are at level 1. Vertices  $c$ ,  $e$ ,  $f$ , and  $l$  are at level 2. Vertices  $d$ ,  $g$ ,  $i$ ,  $m$ , and  $n$  are at level 3. Finally, vertex  $h$  is at level 4. Because the largest level of any vertex is 4, this tree has height 4.  $\triangleleft$

A rooted  $m$ -ary tree of height  $h$  is **balanced** if all leaves are at levels  $h$  or  $h - 1$ .

**EXAMPLE 11**

Which of the rooted trees shown in Figure 14 are balanced?

**Solution:**  $T_1$  is balanced, because all its leaves are at levels 3 and 4. However,  $T_2$  is not balanced, because it has leaves at levels 2, 3, and 4. Finally,  $T_3$  is balanced, because all its leaves are at level 3.  $\triangleleft$

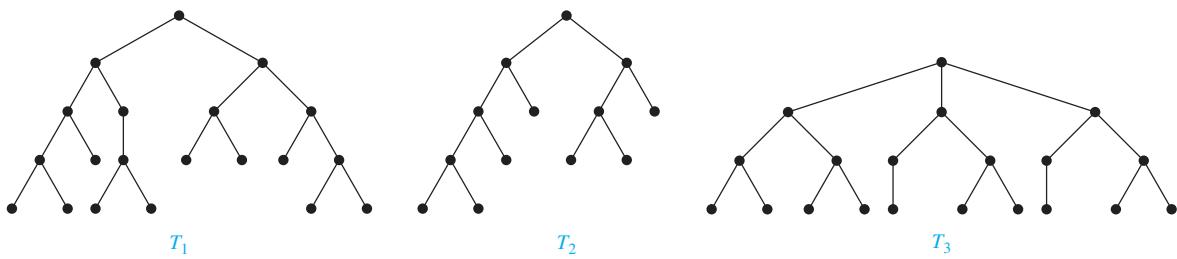


FIGURE 14 Some Rooted Trees.

**A BOUND FOR THE NUMBER OF LEAVES IN AN  $m$ -ARY TREE** It is often useful to have an upper bound for the number of leaves in an  $m$ -ary tree. Theorem 5 provides such a bound in terms of the height of the  $m$ -ary tree.

**THEOREM 5**

There are at most  $m^h$  leaves in an  $m$ -ary tree of height  $h$ .

**Proof:** The proof uses mathematical induction on the height. First, consider  $m$ -ary trees of height 1. These trees consist of a root with no more than  $m$  children, each of which is a leaf. Hence, there are no more than  $m^1 = m$  leaves in an  $m$ -ary tree of height 1. This is the basis step of the inductive argument.

Now assume that the result is true for all  $m$ -ary trees of height less than  $h$ ; this is the inductive hypothesis. Let  $T$  be an  $m$ -ary tree of height  $h$ . The leaves of  $T$  are the leaves of the subtrees of  $T$  obtained by deleting the edges from the root to each of the vertices at level 1, as shown in Figure 15.

Each of these subtrees has height less than or equal to  $h - 1$ . So by the inductive hypothesis, each of these rooted trees has at most  $m^{h-1}$  leaves. Because there are at most  $m$  such subtrees, each with a maximum of  $m^{h-1}$  leaves, there are at most  $m \cdot m^{h-1} = m^h$  leaves in the rooted tree. This finishes the inductive argument.  $\triangleleft$

**COROLLARY 1**

If an  $m$ -ary tree of height  $h$  has  $l$  leaves, then  $h \geq \lceil \log_m l \rceil$ . If the  $m$ -ary tree is full and balanced, then  $h = \lceil \log_m l \rceil$ . (We are using the ceiling function here. Recall that  $\lceil x \rceil$  is the smallest integer greater than or equal to  $x$ .)

**Proof:** We know that  $l \leq m^h$  from Theorem 5. Taking logarithms to the base  $m$  shows that  $\log_m l \leq h$ . Because  $h$  is an integer, we have  $h \geq \lceil \log_m l \rceil$ . Now suppose that the tree is balanced.

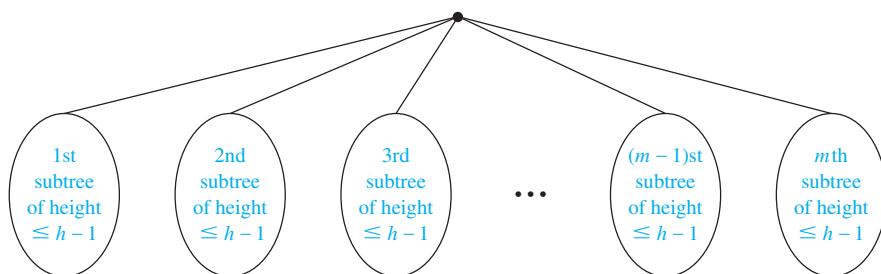
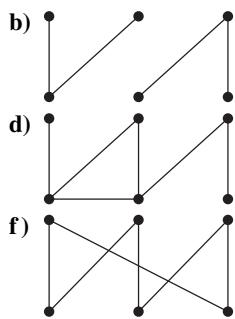
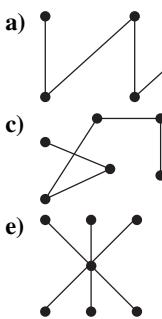


FIGURE 15 The Inductive Step of the Proof.

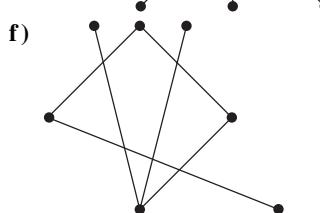
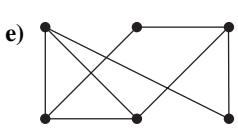
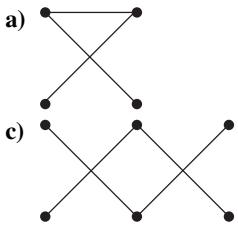
Then each leaf is at level  $h$  or  $h - 1$ , and because the height is  $h$ , there is at least one leaf at level  $h$ . It follows that there must be more than  $m^{h-1}$  leaves (see Exercise 30). Because  $l \leq m^h$ , we have  $m^{h-1} < l \leq m^h$ . Taking logarithms to the base  $m$  in this inequality gives  $h - 1 < \log_m l \leq h$ . Hence,  $h = \lceil \log_m l \rceil$ .  $\triangleleft$

## Exercises

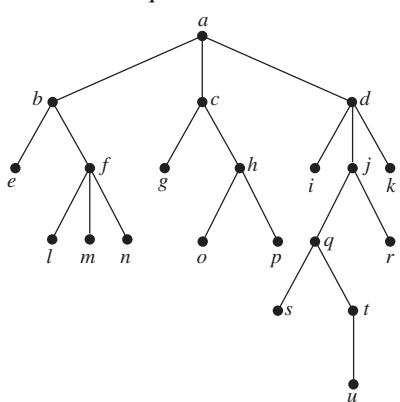
1. Which of these graphs are trees?



2. Which of these graphs are trees?



3. Answer these questions about the rooted tree illustrated.



- a) Which vertex is the root?

- b) Which vertices are internal?

- c) Which vertices are leaves?

- d) Which vertices are children of  $j$ ?

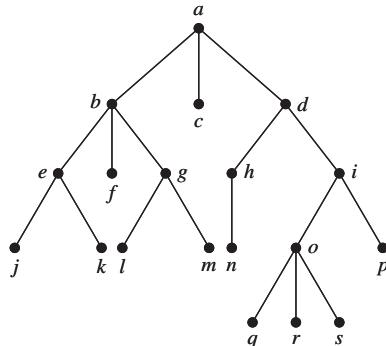
- e) Which vertex is the parent of  $h$ ?

- f) Which vertices are siblings of  $o$ ?

- g) Which vertices are ancestors of  $m$ ?

- h) Which vertices are descendants of  $b$ ?

4. Answer the same questions as listed in Exercise 3 for the rooted tree illustrated.



5. Is the rooted tree in Exercise 3 a full  $m$ -ary tree for some positive integer  $m$ ?

6. Is the rooted tree in Exercise 4 a full  $m$ -ary tree for some positive integer  $m$ ?

7. What is the level of each vertex of the rooted tree in Exercise 3?

8. What is the level of each vertex of the rooted tree in Exercise 4?

9. Draw the subtree of the tree in Exercise 3 that is rooted at

- a)  $a$ .      b)  $c$ .      c)  $e$ .

10. Draw the subtree of the tree in Exercise 4 that is rooted at

- a)  $a$ .      b)  $c$ .      c)  $e$ .

11. a) How many nonisomorphic unrooted trees are there with three vertices?

- b) How many nonisomorphic rooted trees are there with three vertices (using isomorphism for directed graphs)?

- \*12. a) How many nonisomorphic unrooted trees are there with four vertices?

- b) How many nonisomorphic rooted trees are there with four vertices (using isomorphism for directed graphs)?

- \*13. a) How many nonisomorphic unrooted trees are there with five vertices?  
 b) How many nonisomorphic rooted trees are there with five vertices (using isomorphism for directed graphs)?
- \*14. Show that a simple graph is a tree if and only if it is connected but the deletion of any of its edges produces a graph that is not connected.
- \*15. Let  $G$  be a simple graph with  $n$  vertices. Show that  
 a)  $G$  is a tree if and only if it is connected and has  $n - 1$  edges.  
 b)  $G$  is a tree if and only if  $G$  has no simple circuits and has  $n - 1$  edges. [Hint: To show that  $G$  is connected if it has no simple circuits and  $n - 1$  edges, show that  $G$  cannot have more than one connected component.]
16. Which complete bipartite graphs  $K_{m,n}$ , where  $m$  and  $n$  are positive integers, are trees?
17. How many edges does a tree with 10,000 vertices have?
18. How many vertices does a full 5-ary tree with 100 internal vertices have?
19. How many edges does a full binary tree with 1000 internal vertices have?
20. How many leaves does a full 3-ary tree with 100 vertices have?
21. Suppose 1000 people enter a chess tournament. Use a rooted tree model of the tournament to determine how many games must be played to determine a champion, if a player is eliminated after one loss and games are played until only one entrant has not lost. (Assume there are no ties.)
22. A chain letter starts when a person sends a letter to five others. Each person who receives the letter either sends it to five other people who have never received it or does not send it to anyone. Suppose that 10,000 people send out the letter before the chain ends and that no one receives more than one letter. How many people receive the letter, and how many do not send it out?
23. A chain letter starts with a person sending a letter out to 10 others. Each person is asked to send the letter out to 10 others, and each letter contains a list of the previous six people in the chain. Unless there are fewer than six names in the list, each person sends one dollar to the first person in this list, removes the name of this person from the list, moves up each of the other five names one position, and inserts his or her name at the end of this list. If no person breaks the chain and no one receives more than one letter, how much money will a person in the chain ultimately receive?
- \*24. Either draw a full  $m$ -ary tree with 76 leaves and height 3, where  $m$  is a positive integer, or show that no such tree exists.
- \*25. Either draw a full  $m$ -ary tree with 84 leaves and height 3, where  $m$  is a positive integer, or show that no such tree exists.

- \*26. A full  $m$ -ary tree  $T$  has 81 leaves and height 4.

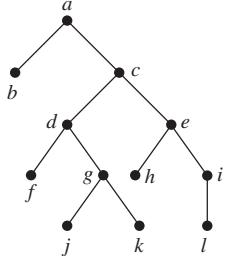
- a) Give the upper and lower bounds for  $m$ .  
 b) What is  $m$  if  $T$  is also balanced?

A **complete  $m$ -ary tree** is a full  $m$ -ary tree in which every leaf is at the same level.

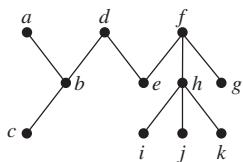
27. Construct a complete binary tree of height 4 and a complete 3-ary tree of height 3.
28. How many vertices and how many leaves does a complete  $m$ -ary tree of height  $h$  have?
29. Prove  
 a) part (ii) of Theorem 4.  
 b) part (iii) of Theorem 4.
30. Show that a full  $m$ -ary balanced tree of height  $h$  has more than  $m^{h-1}$  leaves.
31. How many edges are there in a forest of  $t$  trees containing a total of  $n$  vertices?
32. Explain how a tree can be used to represent the table of contents of a book organized into chapters, where each chapter is organized into sections, and each section is organized into subsections.
33. How many different isomers do these saturated hydrocarbons have?  
 a)  $C_3H_8$       b)  $C_5H_{12}$       c)  $C_6H_{14}$
34. What does each of these represent in an organizational tree?  
 a) the parent of a vertex  
 b) a child of a vertex  
 c) a sibling of a vertex  
 d) the ancestors of a vertex  
 e) the descendants of a vertex  
 f) the level of a vertex  
 g) the height of the tree
35. Answer the same questions as those given in Exercise 34 for a rooted tree representing a computer file system.
36. a) Draw the complete binary tree with 15 vertices that represents a tree-connected network of 15 processors.  
 b) Show how 16 numbers can be added using the 15 processors in part (a) using four steps.
37. Let  $n$  be a power of 2. Show that  $n$  numbers can be added in  $\log n$  steps using a tree-connected network of  $n - 1$  processors.
- \*38. A **labeled tree** is a tree where each vertex is assigned a label. Two labeled trees are considered isomorphic when there is an isomorphism between them that preserves the labels of vertices. How many nonisomorphic trees are there with three vertices labeled with different integers from the set  $\{1, 2, 3\}$ ? How many nonisomorphic trees are there with four vertices labeled with different integers from the set  $\{1, 2, 3, 4\}$ ?

The **eccentricity** of a vertex in an unrooted tree is the length of the longest simple path beginning at this vertex. A vertex is called a **center** if no vertex in the tree has smaller eccentricity than this vertex. In Exercises 39–41 find every vertex that is a center in the given tree.

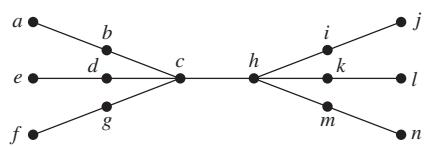
39.



40.



41.



42. Show that a center should be chosen as the root to produce a rooted tree of minimal height from an unrooted tree.

- \*43. Show that a tree has either one center or two centers that are adjacent.

44. Show that every tree can be colored using two colors.

The **rooted Fibonacci trees**  $T_n$  are defined recursively in the following way.  $T_1$  and  $T_2$  are both the rooted tree consisting of a single vertex, and for  $n = 3, 4, \dots$ , the rooted tree  $T_n$  is constructed from a root with  $T_{n-1}$  as its left subtree and  $T_{n-2}$  as its right subtree.

45. Draw the first seven rooted Fibonacci trees.

- \*46. How many vertices, leaves, and internal vertices does the rooted Fibonacci tree  $T_n$  have, where  $n$  is a positive integer? What is its height?

47. What is wrong with the following “proof” using mathematical induction of the statement that every tree with  $n$  vertices has a path of length  $n - 1$ . *Basis step:* Every tree with one vertex clearly has a path of length 0. *Inductive step:* Assume that a tree with  $n$  vertices has a path of length  $n - 1$ , which has  $u$  as its terminal vertex. Add a vertex  $v$  and the edge from  $u$  to  $v$ . The resulting tree has  $n + 1$  vertices and has a path of length  $n$ . This completes the inductive step.

48. Show that the average depth of a leaf in a binary tree with  $n$  vertices is  $\Omega(\log n)$ .

## 11.2 Applications of Trees

### Introduction

We will discuss three problems that can be studied using trees. The first problem is: How should items in a list be stored so that an item can be easily located? The second problem is: What series of decisions should be made to find an object with a certain property in a collection of objects of a certain type? The third problem is: How should a set of characters be efficiently coded by bit strings?

### Binary Search Trees



Searching for items in a list is one of the most important tasks that arises in computer science. Our primary goal is to implement a searching algorithm that finds items efficiently when the items are totally ordered. This can be accomplished through the use of a **binary search tree**, which is a binary tree in which each child of a vertex is designated as a right or left child, no vertex has more than one right child or left child, and each vertex is labeled with a key, which is one of the items. Furthermore, vertices are assigned keys so that the key of a vertex is both larger than the keys of all vertices in its left subtree and smaller than the keys of all vertices in its right subtree.

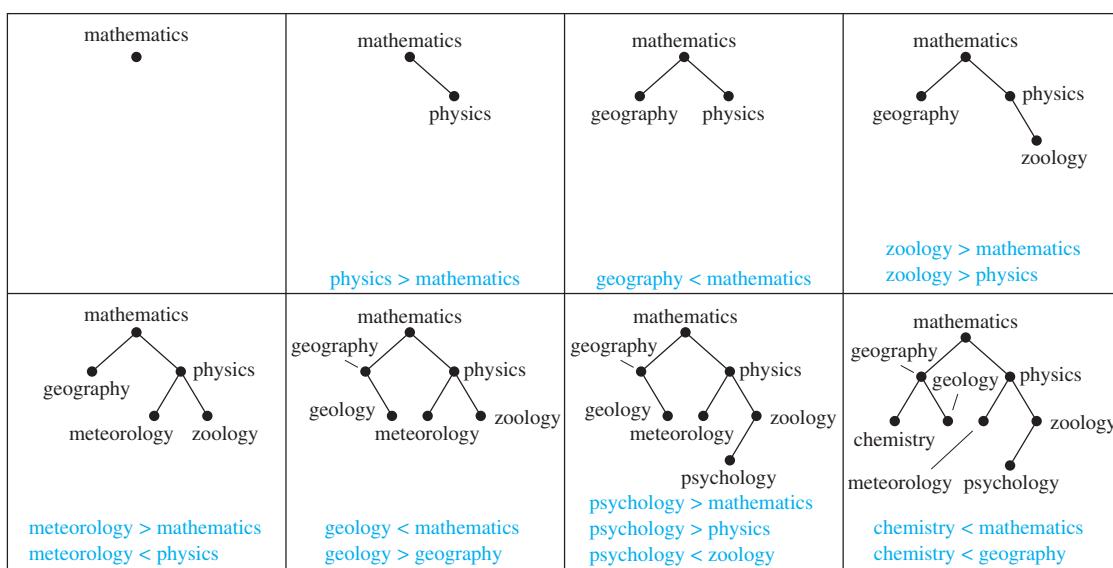
This recursive procedure is used to form the binary search tree for a list of items. Start with a tree containing just one vertex, namely, the root. The first item in the list is assigned as the key of the root. To add a new item, first compare it with the keys of vertices already in the tree, starting at the root and moving to the left if the item is less than the key of the respective vertex if this vertex has a left child, or moving to the right if the item is greater than the key of the

respective vertex if this vertex has a right child. When the item is less than the respective vertex and this vertex has no left child, then a new vertex with this item as its key is inserted as a new left child. Similarly, when the item is greater than the respective vertex and this vertex has no right child, then a new vertex with this item as its key is inserted as a new right child. We illustrate this procedure with Example 1.

**EXAMPLE 1** Form a binary search tree for the words *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology*, and *chemistry* (using alphabetical order).

**Solution:** Figure 1 displays the steps used to construct this binary search tree. The word *mathematics* is the key of the root. Because *physics* comes after *mathematics* (in alphabetical order), add a right child of the root with key *physics*. Because *geography* comes before *mathematics*, add a left child of the root with key *geography*. Next, add a right child of the vertex with key *physics*, and assign it the key *zoology*, because *zoology* comes after *mathematics* and after *physics*. Similarly, add a left child of the vertex with key *physics* and assign this new vertex the key *meteorology*. Add a right child of the vertex with key *geography* and assign this new vertex the key *geology*. Add a left child of the vertex with key *zoology* and assign it the key *psychology*. Add a left child of the vertex with key *geology* and assign it the key *chemistry*. (The reader should work through all the comparisons needed at each step.) 

Once we have a binary search tree, we need a way to locate items in the binary search tree, as well as a way to add new items. Algorithm 1, an insertion algorithm, actually does both of these tasks, even though it may appear that it is only designed to add vertices to a binary search tree. That is, Algorithm 1 is a procedure that locates an item  $x$  in a binary search tree if it is present, and adds a new vertex with  $x$  as its key if  $x$  is not present. In the pseudocode,  $v$  is the vertex currently under examination and  $\text{label}(v)$  represents the key of this vertex. The algorithm begins by examining the root. If  $x$  equals the key of  $v$ , then the algorithm has found the location of  $x$  and terminates; if  $x$  is less than the key of  $v$ , we move to the left child of  $v$  and repeat the procedure; and if  $x$  is greater than the key of  $v$ , we move to the right child of  $v$  and repeat the procedure. If at any step we attempt to move to a child that is not present, we know that  $x$  is not present in the tree, and we add a new vertex as this child with  $x$  as its key.



**FIGURE 1** Constructing a Binary Search Tree.

**ALGORITHM 1 Locating an Item in or Adding an Item to a Binary Search Tree.**

```

procedure insertion(T : binary search tree, x : item)
 $v :=$ root of T
{a vertex not present in T has the value null}
while $v \neq null$ and $label(v) \neq x$
 if $x < label(v)$ then
 if left child of $v \neq null$ then $v :=$ left child of v
 else add new vertex as a left child of v and set $v := null$
 else
 if right child of $v \neq null$ then $v :=$ right child of v
 else add new vertex as a right child of v and set $v := null$
 if root of $T = null$ then add a vertex v to the tree and label it with x
 else if v is null or $label(v) \neq x$ then label new vertex with x and let v be this new vertex
return v { v = location of x }

```

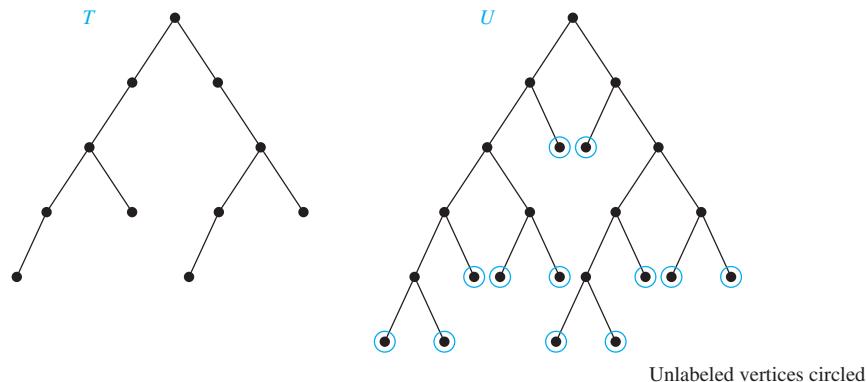
Example 2 illustrates the use of Algorithm 1 to insert a new item into a binary search tree.

**EXAMPLE 2** Use Algorithm 1 to insert the word *oceanography* into the binary search tree in Example 1.

**Solution:** Algorithm 1 begins with  $v$ , the vertex under examination, equal to the root of  $T$ , so  $label(v) = mathematics$ . Because  $v \neq null$  and  $label(v) = mathematics < oceanography$ , we next examine the right child of the root. This right child exists, so we set  $v$ , the vertex under examination, to be this right child. At this step we have  $v \neq null$  and  $label(v) = physics > oceanography$ , so we examine the left child of  $v$ . This left child exists, so we set  $v$ , the vertex under examination, to this left child. At this step, we also have  $v \neq null$  and  $label(v) = metereology < oceanography$ , so we try to examine the right child of  $v$ . However, this right child does not exist, so we add a new vertex as the right child of  $v$  (which at this point is the vertex with the key *metereology*) and we set  $v := null$ . We now exit the **while** loop because  $v = null$ . Because the root of  $T$  is not *null* and  $v = null$ , we use the **else if** statement at the end of the algorithm to label our new vertex with the key *oceanography*. 

We will now determine the computational complexity of this procedure. Suppose we have a binary search tree  $T$  for a list of  $n$  items. We can form a full binary tree  $U$  from  $T$  by adding unlabeled vertices whenever necessary so that every vertex with a key has two children. This is illustrated in Figure 2. Once we have done this, we can easily locate or add a new item as a key without adding a vertex.

The most comparisons needed to add a new item is the length of the longest path in  $U$  from the root to a leaf. The internal vertices of  $U$  are the vertices of  $T$ . It follows that  $U$  has  $n$  internal vertices. We can now use part (ii) of Theorem 4 in Section 11.1 to conclude that  $U$  has  $n + 1$  leaves. Using Corollary 1 of Section 11.1, we see that the height of  $U$  is greater than or equal to  $h = \lceil \log(n + 1) \rceil$ . Consequently, it is necessary to perform at least  $\lceil \log(n + 1) \rceil$  comparisons to add some item. Note that if  $U$  is balanced, its height is  $\lceil \log(n + 1) \rceil$  (by Corollary 1 of Section 11.1). Thus, if a binary search tree is balanced, locating or adding an item requires no more than  $\lceil \log(n + 1) \rceil$  comparisons. A binary search tree can become unbalanced as items are added to it. Because balanced binary search trees give optimal worst-case complexity for binary searching, algorithms have been devised that rebalance binary search trees as items are added. The interested reader can consult references on data structures for the description of such algorithms.



**FIGURE 2** Adding Unlabeled Vertices to Make a Binary Search Tree Full.

## Decision Trees



Rooted trees can be used to model problems in which a series of decisions leads to a solution. For instance, a binary search tree can be used to locate items based on a series of comparisons, where each comparison tells us whether we have located the item, or whether we should go right or left in a subtree. A rooted tree in which each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision, is called a **decision tree**. The possible solutions of the problem correspond to the paths to the leaves of this rooted tree. Example 3 illustrates an application of decision trees.

### EXAMPLE 3

Suppose there are seven coins, all with the same weight, and a counterfeit coin that weighs less than the others. How many weighings are necessary using a balance scale to determine which of the eight coins is the counterfeit one? Give an algorithm for finding this counterfeit coin.

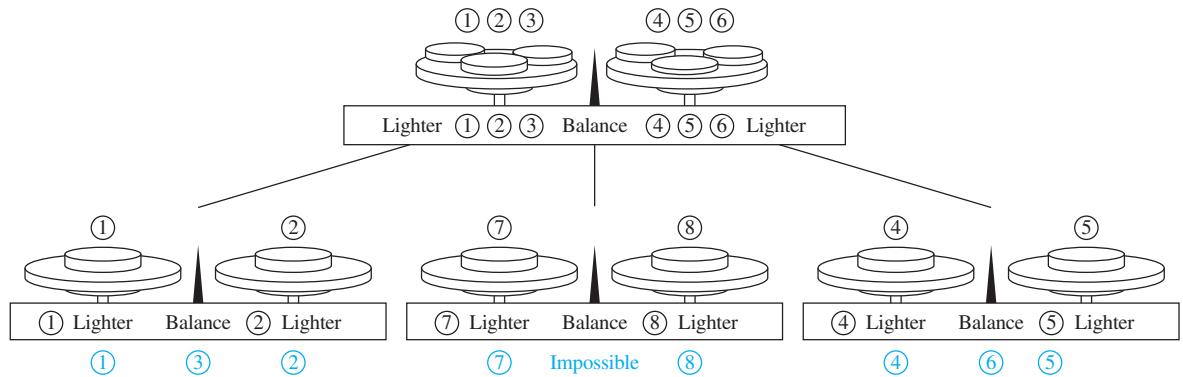


**Solution:** There are three possibilities for each weighing on a balance scale. The two pans can have equal weight, the first pan can be heavier, or the second pan can be heavier. Consequently, the decision tree for the sequence of weighings is a 3-ary tree. There are at least eight leaves in the decision tree because there are eight possible outcomes (because each of the eight coins can be the counterfeit lighter coin), and each possible outcome must be represented by at least one leaf. The largest number of weighings needed to determine the counterfeit coin is the height of the decision tree. From Corollary 1 of Section 11.1 it follows that the height of the decision tree is at least  $\lceil \log_3 8 \rceil = 2$ . Hence, at least two weighings are needed.

It is possible to determine the counterfeit coin using two weighings. The decision tree that illustrates how this is done is shown in Figure 3.

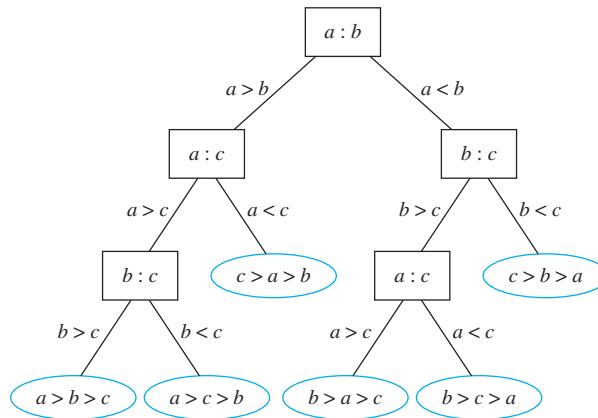
**THE COMPLEXITY OF COMPARISON-BASED SORTING ALGORITHMS** Many different sorting algorithms have been developed. To decide whether a particular sorting algorithm is efficient, its complexity is determined. Using decision trees as models, a lower bound for the worst-case complexity of sorting algorithms that are based on binary comparisons can be found.

We can use decision trees to model sorting algorithms and to determine an estimate for the worst-case complexity of these algorithms. Note that given  $n$  elements, there are  $n!$  possible orderings of these elements, because each of the  $n!$  permutations of these elements can be the correct order. The sorting algorithms studied in this book, and most commonly used sorting algorithms, are based on binary comparisons, that is, the comparison of two elements at a time. The result of each such comparison narrows down the set of possible orderings. Thus, a sorting algorithm based on binary comparisons can be represented by a binary decision tree in which each internal vertex represents a comparison of two elements. Each leaf represents one of the  $n!$  permutations of  $n$  elements.



**FIGURE 3** A Decision Tree for Locating a Counterfeit Coin. The counterfeit coin is shown in color below each final weighing.

**EXAMPLE 4** We display in Figure 4 a decision tree that orders the elements of the list  $a, b, c$ . ◀



**FIGURE 4** A Decision Tree for Sorting Three Distinct Elements.

The complexity of a sort based on binary comparisons is measured in terms of the number of such comparisons used. The largest number of binary comparisons ever needed to sort a list with  $n$  elements gives the worst-case performance of the algorithm. The most comparisons used equals the longest path length in the decision tree representing the sorting procedure. In other words, the largest number of comparisons ever needed is equal to the height of the decision tree. Because the height of a binary tree with  $n!$  leaves is at least  $\lceil \log n! \rceil$  (using Corollary 1 in Section 11.1), at least  $\lceil \log n! \rceil$  comparisons are needed, as stated in Theorem 1.

### THEOREM 1

A sorting algorithm based on binary comparisons requires at least  $\lceil \log n! \rceil$  comparisons.

We can use Theorem 1 to provide a big-Omega estimate for the number of comparisons used by a sorting algorithm based on binary comparison. We need only note that by Exercise 72 in Section 3.2 we know that  $\lceil \log n! \rceil$  is  $\Theta(n \log n)$ , one of the commonly used reference functions for the computational complexity of algorithms. Corollary 1 is a consequence of this estimate.

**COROLLARY 1**

The number of comparisons used by a sorting algorithm to sort  $n$  elements based on binary comparisons is  $\Omega(n \log n)$ .

A consequence of Corollary 1 is that a sorting algorithm based on binary comparisons that uses  $\Theta(n \log n)$  comparisons, in the worst case, to sort  $n$  elements is optimal, in the sense that no other such algorithm has better worst-case complexity. Note that by Theorem 1 in Section 5.4 we see that the merge sort algorithm is optimal in this sense.

We can also establish a similar result for the average-case complexity of sorting algorithms. The average number of comparisons used by a sorting algorithm based on binary comparisons is the average depth of a leaf in the decision tree representing the sorting algorithm. By Exercise 48 in Section 11.1 we know that the average depth of a leaf in a binary tree with  $N$  vertices is  $\Omega(\log N)$ . We obtain the following estimate when we let  $N = n!$  and note that a function that is  $\Omega(\log n!)$  is also  $\Omega(n \log n)$  because  $\log n!$  is  $\Theta(n \log n)$ .

**THEOREM 2**

The average number of comparisons used by a sorting algorithm to sort  $n$  elements based on binary comparisons is  $\Omega(n \log n)$ .

**Prefix Codes**

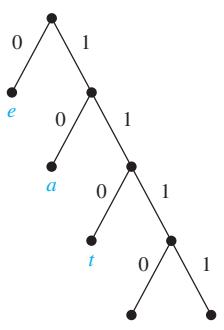
Consider the problem of using bit strings to encode the letters of the English alphabet (where no distinction is made between lowercase and uppercase letters). We can represent each letter with a bit string of length five, because there are only 26 letters and there are 32 bit strings of length five. The total number of bits used to encode data is five times the number of characters in the text when each character is encoded with five bits. Is it possible to find a coding scheme of these letters such that, when data are coded, fewer bits are used? We can save memory and reduce transmittal time if this can be done.

Consider using bit strings of different lengths to encode letters. Letters that occur more frequently should be encoded using short bit strings, and longer bit strings should be used to encode rarely occurring letters. When letters are encoded using varying numbers of bits, some method must be used to determine where the bits for each character start and end. For instance, if *e* were encoded with 0, *a* with 1, and *t* with 01, then the bit string 0101 could correspond to *eat*, *tea*, *eaea*, or *tt*.

One way to ensure that no bit string corresponds to more than one sequence of letters is to encode letters so that the bit string for a letter never occurs as the first part of the bit string for another letter. Codes with this property are called **prefix codes**. For instance, the encoding of *e* as 0, *a* as 10, and *t* as 11 is a prefix code. A word can be recovered from the unique bit string that encodes its letters. For example, the string 10110 is the encoding of *ate*. To see this, note that the initial 1 does not represent a character, but 10 does represent *a* (and could not be the first part of the bit string of another letter). Then, the next 1 does not represent a character, but 11 does represent *t*. The final bit, 0, represents *e*.

A prefix code can be represented using a binary tree, where the characters are the labels of the leaves in the tree. The edges of the tree are labeled so that an edge leading to a left child is assigned a 0 and an edge leading to a right child is assigned a 1. The bit string used to encode a character is the sequence of labels of the edges in the unique path from the root to the leaf that has this character as its label. For instance, the tree in Figure 5 represents the encoding of *e* by 0, *a* by 10, *t* by 110, *n* by 1110, and *s* by 1111.

The tree representing a code can be used to decode a bit string. For instance, consider the word encoded by 111110111100 using the code in Figure 5. This bit string can be decoded by starting at the root, using the sequence of bits to form a path that stops when a leaf is reached.



**FIGURE 5** A  
Binary Tree with a  
Prefix Code.

Each 0 bit takes the path down the edge leading to the left child of the last vertex in the path, and each 1 bit corresponds to the right child of this vertex. Consequently, the initial 1111 corresponds to the path starting at the root, going right four times, leading to a leaf in the graph that has  $s$  as its label, because the string 1111 is the code for  $s$ . Continuing with the fifth bit, we reach a leaf next after going right then left, when the vertex labeled with  $a$ , which is encoded by 10, is visited. Starting with the seventh bit, we reach a leaf next after going right three times and then left, when the vertex labeled with  $n$ , which is encoded by 1110, is visited. Finally, the last bit, 0, leads to the leaf that is labeled with  $e$ . Therefore, the original word is *sane*.

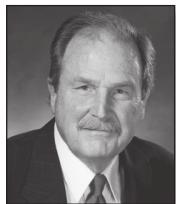
We can construct a prefix code from any binary tree where the left edge at each internal vertex is labeled by 0 and the right edge by a 1 and where the leaves are labeled by characters. Characters are encoded with the bit string constructed using the labels of the edges in the unique path from the root to the leaves.



**HUFFMAN CODING** We now introduce an algorithm that takes as input the frequencies (which are the probabilities of occurrences) of symbols in a string and produces as output a prefix code that encodes the string using the fewest possible bits, among all possible binary prefix codes for these symbols. This algorithm, known as **Huffman coding**, was developed by David Huffman in a term paper he wrote in 1951 while a graduate student at MIT. (Note that this algorithm assumes that we already know how many times each symbol occurs in the string, so we can compute the frequency of each symbol by dividing the number of times this symbol occurs by the length of the string.) Huffman coding is a fundamental algorithm in *data compression*, the subject devoted to reducing the number of bits required to represent information. Huffman coding is extensively used to compress bit strings representing text and it also plays an important role in compressing audio and image files.



Algorithm 2 presents the Huffman coding algorithm. Given symbols and their frequencies, our goal is to construct a rooted binary tree where the symbols are the labels of the leaves. The algorithm begins with a forest of trees each consisting of one vertex, where each vertex has a symbol as its label and where the weight of this vertex equals the frequency of the symbol that is its label. At each step, we combine two trees having the least total weight into a single tree by introducing a new root and placing the tree with larger weight as its left subtree and the tree with smaller weight as its right subtree. Furthermore, we assign the sum of the weights of the two subtrees of this tree as the total weight of the tree. (Although procedures for breaking ties by choosing between trees with equal weights can be specified, we will not specify such procedures here.) The algorithm is finished when it has constructed a tree, that is, when the forest is reduced to a single tree.



**DAVID A. HUFFMAN (1925–1999)** David Huffman grew up in Ohio. At the age of 18 he received his B.S. in electrical engineering from The Ohio State University. Afterward he served in the U.S. Navy as a radar maintenance officer on a destroyer that had the mission of clearing mines in Asian waters after World War II. Later, he earned his M.S. from Ohio State and his Ph.D. in electrical engineering from MIT. Huffman joined the MIT faculty in 1953, where he remained until 1967 when he became the founding member of the computer science department at the University of California at Santa Cruz. He played an important role in developing this department and spent the remainder of his career there, retiring in 1994.

Huffman is noted for his contributions to information theory and coding, signal designs for radar and for communications, and design procedures for asynchronous logical circuits. His work on surfaces with zero curvature led him to develop original techniques for folding paper and vinyl into unusual shapes considered works of art by many and publicly displayed in several exhibits. However, Huffman is best known for his development of what is now called Huffman coding, a result of a term paper he wrote during his graduate work at MIT.

Huffman enjoyed exploring the outdoors, hiking, and traveling extensively. He became certified as a scuba diver when he was in his late 60s. He kept poisonous snakes as pets.

**ALGORITHM 2 Huffman Coding.**

```

procedure Huffman(C: symbols a_i with frequencies w_i , $i = 1, \dots, n$)
F := forest of n rooted trees, each consisting of the single vertex a_i and assigned weight w_i
while F is not a tree
 Replace the rooted trees T and T' of least weights from F with $w(T) \geq w(T')$ with a tree
 having a new root that has T as its left subtree and T' as its right subtree. Label the new
 edge to T with 0 and the new edge to T' with 1.
 Assign $w(T) + w(T')$ as the weight of the new tree.
{the Huffman coding for the symbol a_i is the concatenation of the labels of the edges in the
unique path from the root to the vertex a_i }

```

Example 5 illustrates how Algorithm 2 is used to encode a set of five symbols.

**EXAMPLE 5**

Use Huffman coding to encode the following symbols with the frequencies listed: A: 0.08, B: 0.10, C: 0.12, D: 0.15, E: 0.20, F: 0.35. What is the average number of bits used to encode a character?

**Solution:** Figure 6 displays the steps used to encode these symbols. The encoding produced encodes A by 111, B by 110, C by 011, D by 010, E by 10, and F by 00. The average number of bits used to encode a symbol using this encoding is

$$3 \cdot 0.08 + 3 \cdot 0.10 + 3 \cdot 0.12 + 3 \cdot 0.15 + 2 \cdot 0.20 + 2 \cdot 0.35 = 2.45.$$

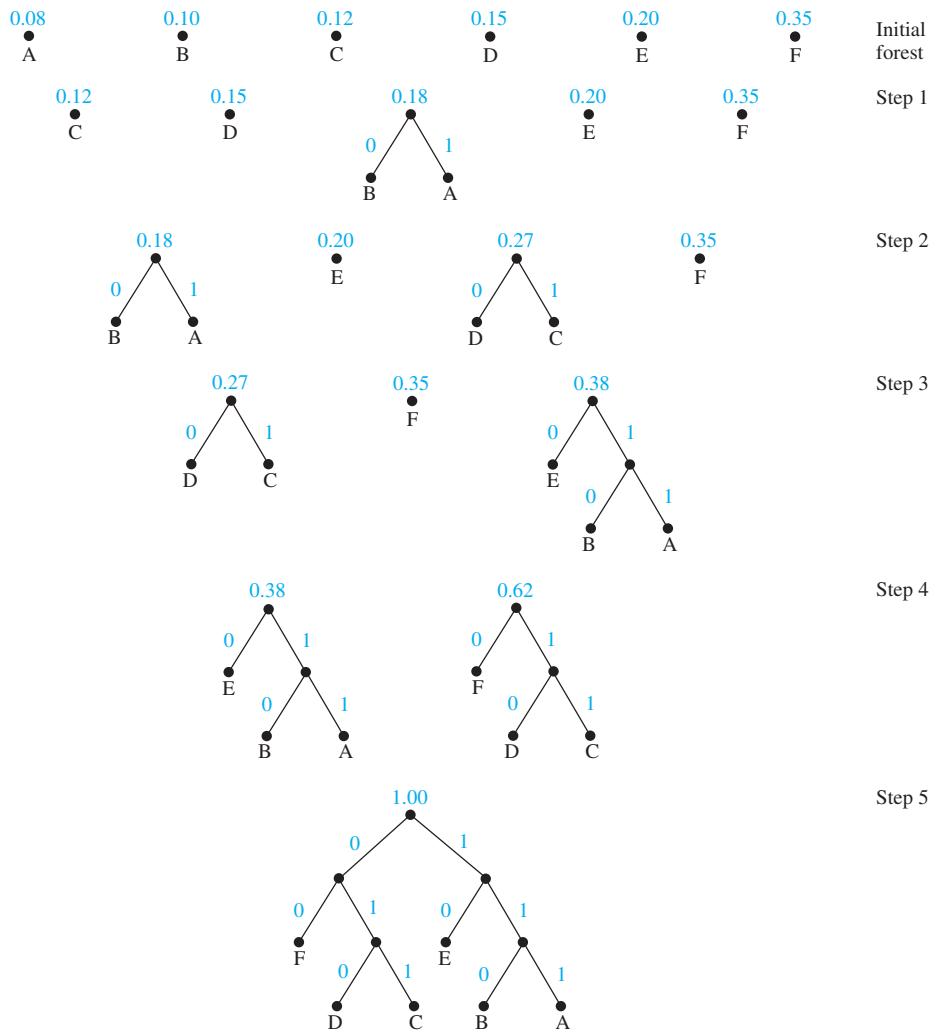
Note that Huffman coding is a greedy algorithm. Replacing the two subtrees with the smallest weight at each step leads to an optimal code in the sense that no binary prefix code for these symbols can encode these symbols using fewer bits. We leave the proof that Huffman codes are optimal as Exercise 32.

There are many variations of Huffman coding. For example, instead of encoding single symbols, we can encode blocks of symbols of a specified length, such as blocks of two symbols. Doing so may reduce the number of bits required to encode the string (see Exercise 30). We can also use more than two symbols to encode the original symbols in the string (see the preamble to Exercise 28). Furthermore, a variation known as adaptive Huffman coding (see [Sa00]) can be used when the frequency of each symbol in a string is not known in advance, so that encoding is done at the same time the string is being read.

Huffman coding is used  
in JPEG image coding

**Game Trees**

Trees can be used to analyze certain types of games such as tic-tac-toe, nim, checkers, and chess. In each of these games, two players take turns making moves. Each player knows the moves made by the other player and no element of chance enters into the game. We model such games using **game trees**; the vertices of these trees represent the positions that a game can be in as it progresses; the edges represent legal moves between these positions. Because game trees are usually large, we simplify game trees by representing all symmetric positions of a game by the same vertex. However, the same position of a game may be represented by different vertices



**FIGURE 6** Huffman Coding of Symbols in Example 4.

if different sequences of moves lead to this position. The root represents the starting position. The usual convention is to represent vertices at even levels by boxes and vertices at odd levels by circles. When the game is in a position represented by a vertex at an even level, it is the first player's move; when the game is in a position represented by a vertex at an odd level, it is the second player's move. Game trees may be infinite when the games they represent never end, such as games that can enter infinite loops, but for most games there are rules that lead to finite game trees.

The leaves of a game tree represent the final positions of a game. We assign a value to each leaf indicating the payoff to the first player if the game terminates in the position represented by this leaf. For games that are win–lose, we label a terminal vertex represented by a circle with a 1 to indicate a win by the first player and we label a terminal vertex represented by a box with a  $-1$  to indicate a win by the second player. For games where draws are allowed, we label a terminal vertex corresponding to a draw position with a 0. Note that for win–lose games, we have assigned values to terminal vertices so that the larger the value, the better the outcome for the first player.

In Example 6 we display a game tree for a well-known and well-studied game.

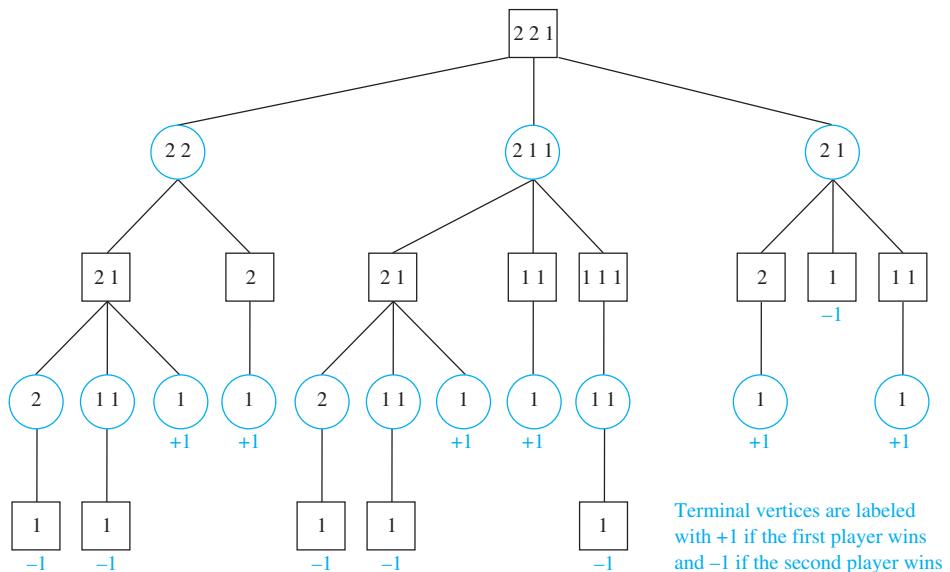


FIGURE 7 The Game Tree for a Game of Nim.

**EXAMPLE 6**

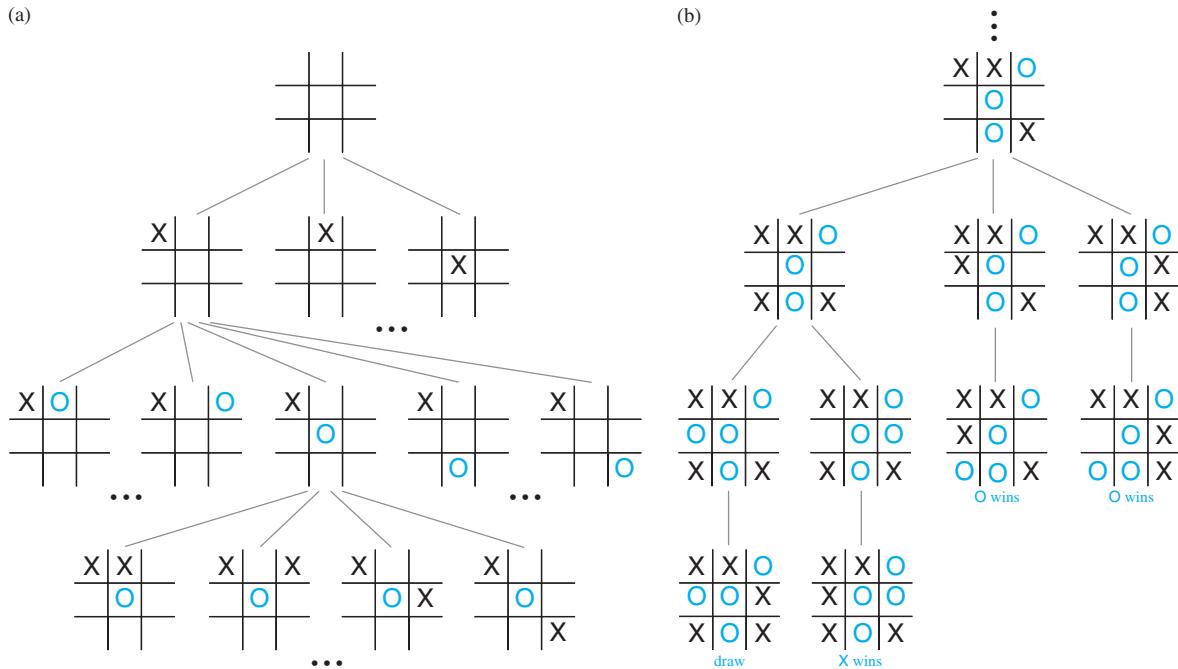
Although nim is an ancient game, Charles Bouton coined its modern name in 1901 after an archaic English word meaning “to steal.”

**Nim** In a version of the game of **nim**, at the start of a game there are a number of piles of stones. Two players take turns making moves; a legal move consists of removing one or more stones from one of the piles, without removing all the stones left. A player without a legal move loses. (Another way to look at this is that the player removing the last stone loses because the position with no piles of stones is not allowed.) The game tree shown in Figure 7 represents this version of nim given the starting position where there are three piles of stones containing two, two, and one stone each, respectively. We represent each position with an unordered list of the number of stones in the different piles (the order of the piles does not matter). The initial move by the first player can lead to three possible positions because this player can remove one stone from a pile with two stones (leaving three piles containing one, one, and two stones); two stones from a pile containing two stones (leaving two piles containing two stones and one stone); or one stone from the pile containing one stone (leaving two piles of two stones). When only one pile with one stone is left, no legal moves are possible, so such positions are terminal positions. Because nim is a win–lose game, we label the terminal vertices with +1 when they represent wins for the first player and –1 when they represent wins for the second player.  $\blacktriangleleft$

**EXAMPLE 7**

**Tic-tac-toe** The game tree for tic-tac-toe is extremely large and cannot be drawn here, although a computer could easily build such a tree. We show a portion of the game tic-tac-toe in Figure 8(a). Note that by considering symmetric positions equivalent, we need only consider three possible initial moves, as shown in Figure 8(a). We also show a subtree of this game tree leading to terminal positions in Figure 8(b), where a player who can win makes a winning move.  $\blacktriangleleft$

We can recursively define the values of all vertices in a game tree in a way that enables us to determine the outcome of this game when both players follow optimal strategies. By a **strategy** we mean a set of rules that tells a player how to select moves to win the game. An optimal strategy for the first player is a strategy that maximizes the payoff to this player and for the second player is a strategy that minimizes this payoff. We now recursively define the value of a vertex.



**FIGURE 8** Some of the Game Tree for Tic-Tac-Toe.

### DEFINITION 1

The *value* of a vertex in a game tree is defined recursively as:

- (i) the value of a leaf is the payoff to the first player when the game terminates in the position represented by this leaf.
- (ii) the value of an internal vertex at an even level is the maximum of the values of its children, and the value of an internal vertex at an odd level is the minimum of the values of its children.

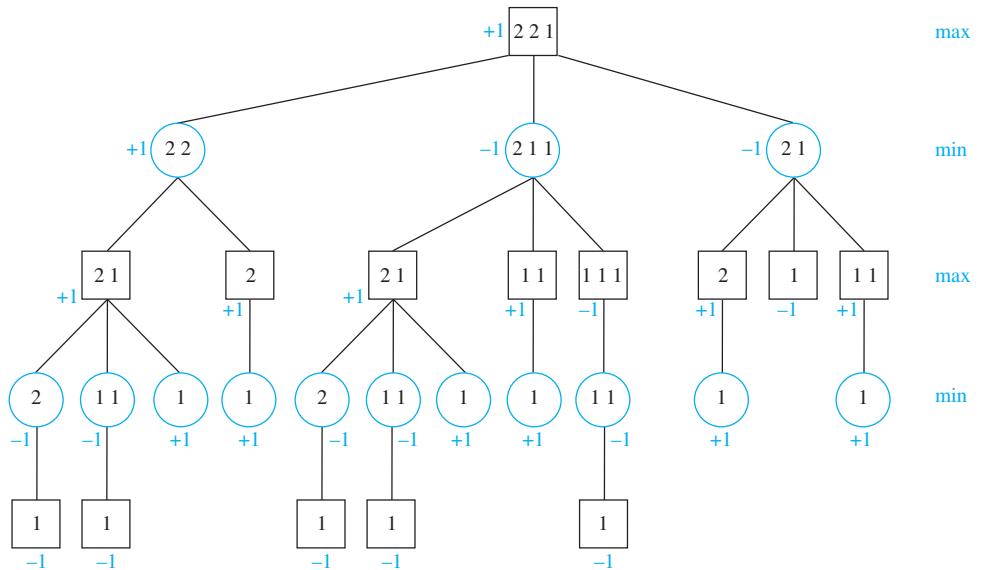
The strategy where the first player moves to a position represented by a child with maximum value and the second player moves to a position of a child with minimum value is called the **minmax strategy**. We can determine who will win the game when both players follow the minmax strategy by calculating the value of the root of the tree; this value is called the **value** of the tree. This is a consequence of Theorem 3.

### THEOREM 3

The value of a vertex of a game tree tells us the payoff to the first player if both players follow the minmax strategy and play starts from the position represented by this vertex.

**Proof:** We will use induction to prove this theorem.

**BASIS STEP:** If the vertex is a leaf, by definition the value assigned to this vertex is the payoff to the first player.



**FIGURE 9** Showing the Values of Vertices in the Game of Nim.

**INDUCTIVE STEP:** The inductive hypothesis is the assumption that the values of the children of a vertex are the payoffs to the first player, assuming that play starts at each of the positions represented by these vertices. We need to consider two cases, when it is the first player's turn and when it is the second player's turn.

When it is the first player's turn, this player follows the minmax strategy and moves to the position represented by the child with the largest value. By the inductive hypothesis, this value is the payoff to the first player when play starts at the position represented by this child and follows the minmax strategy. By the recursive step in the definition of the value of an internal vertex at an even level (as the maximum value of its children), the value of this vertex is the payoff when play begins at the position represented by this vertex.

When it is the second player's turn, this player follows the minmax strategy and moves to the position represented by the child with the least value. By the inductive hypothesis, this value is the payoff to the first player when play starts at the position represented by this child and both players follow the minmax strategy. By the recursive definition of the value of an internal vertex at an odd level as the minimum value of its children, the value of this vertex is the payoff when play begins at the position represented by this vertex. 

**Remark:** By extending the proof of Theorem 3, it can be shown that the minmax strategy is the optimal strategy for both players.

Example 8 illustrates how the minmax procedure works. It displays the values assigned to the internal vertices in the game tree from Example 6. Note that we can shorten the computation required by noting that for win–lose games, once a child of a square vertex with value +1 is found, the value of the square vertex is also +1 because +1 is the largest possible payoff. Similarly, once a child of a circle vertex with value –1 is found, this is the value of the circle vertex also.

**EXAMPLE 8** In Example 6 we constructed the game tree for nim with a starting position where there are three piles containing two, two, and one stones. In Figure 9 we show the values of the vertices of this game tree. The values of the vertices are computed using the values of the leaves and working one level up at a time. In the right margin of this figure we indicate whether we use the maximum or minimum of the values of the children to find the value of an internal vertex at

each level. For example, once we have found the values of the three children of the root, which are 1,  $-1$ , and  $-1$ , we find the value of the root by computing  $\max(1, -1, -1) = 1$ . Because the value of the root is 1, it follows that the first player wins when both players follow a minmax strategy. 

Game trees for some well-known games can be extraordinarily large, because these games have many different possible moves. For example, the game tree for chess has been estimated to have as many as  $10^{100}$  vertices! It may be impossible to use Theorem 3 directly to study a game because of the size of the game tree. Therefore, various approaches have been devised to help determine good strategies and to determine the outcome of such games. One useful technique, called *alpha-beta pruning*, eliminates much computation by pruning portions of the game tree that cannot affect the values of ancestor vertices. (For information about alpha-beta pruning, consult [Gr90].) Another useful approach is to use *evaluation functions*, which estimate the value of internal vertices in the game tree when it is not feasible to compute these values exactly. For example, in the game of tic-tac-toe, as an evaluation function for a position, we may use the number of files (rows, columns, and diagonals) containing no Os (used to indicate moves of the second player) minus the number of files containing no Xs (used to indicate moves of the first player). This evaluation function provides some indication of which player has the advantage in the game. Once the values of an evaluation function are inserted, the value of the game can be computed following the rules used for the minmax strategy. Computer programs created to play chess, such as the famous Deep Blue program, are based on sophisticated evaluation functions. For more information about how computers play chess see [Le91].

Chess programs on smartphones can now play at the grandmaster level.



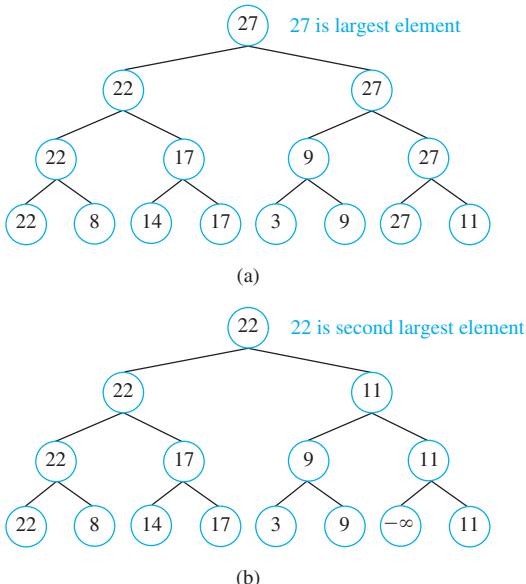
## Exercises

1. Build a binary search tree for the words *banana*, *peach*, *apple*, *pear*, *coconut*, *mango*, and *papaya* using alphabetical order.
2. Build a binary search tree for the words *oenology*, *phrenology*, *campanology*, *ornithology*, *ichthyology*, *limnology*, *alchemy*, and *astrology* using alphabetical order.
3. How many comparisons are needed to locate or to add each of these words in the search tree for Exercise 1, starting fresh each time?
  - a) *pear*
  - b) *banana*
  - c) *kumquat*
  - d) *orange*
4. How many comparisons are needed to locate or to add each of the words in the search tree for Exercise 2, starting fresh each time?
  - a) *palmistry*
  - b) *etymology*
  - c) *paleontology*
  - d) *glaciology*
5. Using alphabetical order, construct a binary search tree for the words in the sentence “*The quick brown fox jumps over the lazy dog.*”
6. How many weighings of a balance scale are needed to find a lighter counterfeit coin among four coins? Describe an algorithm to find the lighter coin using this number of weighings.
7. How many weighings of a balance scale are needed to find a counterfeit coin among four coins if the counterfeit coin may be either heavier or lighter than the others?

Describe an algorithm to find the counterfeit coin using this number of weighings.

- \*8. How many weighings of a balance scale are needed to find a counterfeit coin among eight coins if the counterfeit coin is either heavier or lighter than the others? Describe an algorithm to find the counterfeit coin using this number of weighings.
- \*9. How many weighings of a balance scale are needed to find a counterfeit coin among 12 coins if the counterfeit coin is lighter than the others? Describe an algorithm to find the lighter coin using this number of weighings.
- \*10. One of four coins may be counterfeit. If it is counterfeit, it may be lighter or heavier than the others. How many weighings are needed, using a balance scale, to determine whether there is a counterfeit coin, and if there is, whether it is lighter or heavier than the others? Describe an algorithm to find the counterfeit coin and determine whether it is lighter or heavier using this number of weighings.
11. Find the least number of comparisons needed to sort four elements and devise an algorithm that sorts these elements using this number of comparisons.
- \*12. Find the least number of comparisons needed to sort five elements and devise an algorithm that sorts these elements using this number of comparisons.

The **tournament sort** is a sorting algorithm that works by building an ordered binary tree. We represent the elements to be sorted by vertices that will become the leaves. We build up the tree one level at a time as we would construct the tree representing the winners of matches in a tournament. Working left to right, we compare pairs of consecutive elements, adding a parent vertex labeled with the larger of the two elements under comparison. We make similar comparisons between labels of vertices at each level until we reach the root of the tree that is labeled with the largest element. The tree constructed by the tournament sort of 22, 8, 14, 17, 3, 9, 27, 11 is illustrated in part (a) of the figure. Once the largest element has been determined, the leaf with this label is relabeled by  $-\infty$ , which is defined to be less than every element. The labels of all vertices on the path from this vertex up to the root of the tree are recalculated, as shown in part (b) of the figure. This produces the second largest element. This process continues until the entire list has been sorted.



13. Complete the tournament sort of the list 22, 8, 14, 17, 3, 9, 27, 11. Show the labels of the vertices at each step.
14. Use the tournament sort to sort the list 17, 4, 1, 5, 13, 10, 14, 6.
15. Describe the tournament sort using pseudocode.
16. Assuming that  $n$ , the number of elements to be sorted, equals  $2^k$  for some positive integer  $k$ , determine the number of comparisons used by the tournament sort to find the largest element of the list using the tournament sort.
17. How many comparisons does the tournament sort use to find the second largest, the third largest, and so on, up to the  $(n - 1)$ st largest (or second smallest) element?
18. Show that the tournament sort requires  $\Theta(n \log n)$  comparisons to sort a list of  $n$  elements. [Hint: By inserting the appropriate number of dummy elements defined to be smaller than all integers, such as  $-\infty$ , assume that  $n = 2^k$  for some positive integer  $k$ .]

19. Which of these codes are prefix codes?
  - a)  $a: 11, e: 00, t: 10, s: 01$
  - b)  $a: 0, e: 1, t: 01, s: 001$
  - c)  $a: 101, e: 11, t: 001, s: 011, n: 010$
  - d)  $a: 010, e: 11, t: 011, s: 1011, n: 1001, i: 10101$
20. Construct the binary tree with prefix codes representing these coding schemes.
  - a)  $a: 11, e: 0, t: 101, s: 100$
  - b)  $a: 1, e: 01, t: 001, s: 0001, n: 00001$
  - c)  $a: 1010, e: 0, t: 11, s: 1011, n: 1001, i: 10001$
21. What are the codes for  $a, e, i, k, o, p$ , and  $u$  if the coding scheme is represented by this tree?
22. Given the coding scheme  $a: 001, b: 0001, e: 1, r: 0000, s: 0100, t: 011, x: 01010$ , find the word represented by
  - a) 01110100011.
  - b) 0001110000.
  - c) 0100101010.
  - d) 01100101010.
23. Use Huffman coding to encode these symbols with given frequencies:  $a: 0.20, b: 0.10, c: 0.15, d: 0.25, e: 0.30$ . What is the average number of bits required to encode a character?
24. Use Huffman coding to encode these symbols with given frequencies:  $A: 0.10, B: 0.25, C: 0.05, D: 0.15, E: 0.30, F: 0.07, G: 0.08$ . What is the average number of bits required to encode a symbol?
25. Construct two different Huffman codes for these symbols and frequencies:  $t: 0.2, u: 0.3, v: 0.2, w: 0.3$ .
26. a) Use Huffman coding to encode these symbols with frequencies  $a: 0.4, b: 0.2, c: 0.2, d: 0.1, e: 0.1$  in two different ways by breaking ties in the algorithm differently. First, among the trees of minimum weight select two trees with the largest number of vertices to combine at each stage of the algorithm. Second, among the trees of minimum weight select two trees with the smallest number of vertices at each stage.
   
b) Compute the average number of bits required to encode a symbol with each code and compute the variances of this number of bits for each code. Which tie-breaking procedure produced the smaller variance in the number of bits required to encode a symbol?

27. Construct a Huffman code for the letters of the English alphabet where the frequencies of letters in typical English text are as shown in this table.

| <i>Letter</i> | <i>Frequency</i> | <i>Letter</i> | <i>Frequency</i> |
|---------------|------------------|---------------|------------------|
| A             | 0.0817           | N             | 0.0662           |
| B             | 0.0145           | O             | 0.0781           |
| C             | 0.0248           | P             | 0.0156           |
| D             | 0.0431           | Q             | 0.0009           |
| E             | 0.1232           | R             | 0.0572           |
| F             | 0.0209           | S             | 0.0628           |
| G             | 0.0182           | T             | 0.0905           |
| H             | 0.0668           | U             | 0.0304           |
| I             | 0.0689           | V             | 0.0102           |
| J             | 0.0010           | W             | 0.0264           |
| K             | 0.0080           | X             | 0.0015           |
| L             | 0.0397           | Y             | 0.0211           |
| M             | 0.0277           | Z             | 0.0005           |

Suppose that  $m$  is a positive integer with  $m \geq 2$ . An  $m$ -ary Huffman code for a set of  $N$  symbols can be constructed analogously to the construction of a binary Huffman code. At the initial step,  $((N - 1) \bmod (m - 1)) + 1$  trees consisting of a single vertex with least weights are combined into a rooted tree with these vertices as leaves. At each subsequent step, the  $m$  trees of least weight are combined into an  $m$ -ary tree.

28. Describe the  $m$ -ary Huffman coding algorithm in pseudocode.
29. Using the symbols 0, 1, and 2 use ternary ( $m = 3$ ) Huffman coding to encode these letters with the given frequencies: A: 0.25, E: 0.30, N: 0.10, R: 0.05, T: 0.12, Z: 0.18.
30. Consider the three symbols A, B, and C with frequencies A: 0.80, B: 0.19, C: 0.01.
- Construct a Huffman code for these three symbols.
  - Form a new set of nine symbols by grouping together blocks of two symbols, AA, AB, AC, BA, BB, BC, CA, CB, and CC. Construct a Huffman code for these nine symbols, assuming that the occurrences of symbols in the original text are independent.
  - Compare the average number of bits required to encode text using the Huffman code for the three symbols in part (a) and the Huffman code for the nine blocks of two symbols constructed in part (b). Which is more efficient?
31. Given  $n + 1$  symbols  $x_1, x_2, \dots, x_n, x_{n+1}$  appearing  $1, f_1, f_2, \dots, f_n$  times in a symbol string, respectively, where  $f_j$  is the  $j$ th Fibonacci number, what is the maximum number of bits used to encode a symbol when all possible tie-breaking selections are considered at each stage of the Huffman coding algorithm?
- \*32. Show that Huffman codes are optimal in the sense that they represent a string of symbols using the fewest bits among all binary prefix codes.

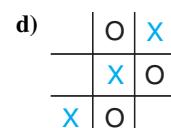
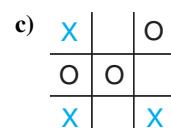
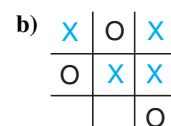
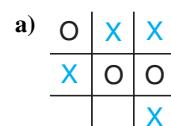
33. Draw a game tree for nim if the starting position consists of two piles with two and three stones, respectively. When drawing the tree represent by the same vertex symmetric positions that result from the same move. Find the value of each vertex of the game tree. Who wins the game if both players follow an optimal strategy?

34. Draw a game tree for nim if the starting position consists of three piles with one, two, and three stones, respectively. When drawing the tree represent by the same vertex symmetric positions that result from the same move. Find the value of each vertex of the game tree. Who wins the game if both players follow an optimal strategy?

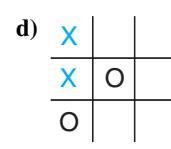
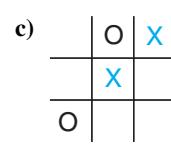
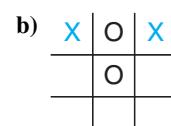
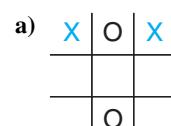
35. Suppose that we vary the payoff to the winning player in the game of nim so that the payoff is  $n$  dollars when  $n$  is the number of legal moves made before a terminal position is reached. Find the payoff to the first player if the initial position consists of
- two piles with one and three stones, respectively.
  - two piles with two and four stones, respectively.
  - three piles with one, two, and three stones, respectively.

36. Suppose that in a variation of the game of nim we allow a player to either remove one or more stones from a pile or merge the stones from two piles into one pile as long as at least one stone remains. Draw the game tree for this variation of nim if the starting position consists of three piles containing two, two, and one stone, respectively. Find the values of each vertex in the game tree and determine the winner if both players follow an optimal strategy.

37. Draw the subtree of the game tree for tic-tac-toe beginning at each of these positions. Determine the value of each of these subtrees.



38. Suppose that the first four moves of a tic-tac-toe game are as shown. Does the first player (whose moves are marked with Xs) have a strategy that will always win?



- 39.** Show that if a game of nim begins with two piles containing the same number of stones, as long as this number is at least two, then the second player wins when both players follow optimal strategies.
- 40.** Show that if a game of nim begins with two piles containing different numbers of stones, the first player wins when both players follow optimal strategies.
- 41.** How many children does the root of the game tree for checkers have? How many grandchildren does it have?
- 42.** How many children does the root of the game tree for nim have and how many grandchildren does it have if the starting position is
- piles with four and five stones, respectively.
  - piles with two, three, and four stones, respectively.
- 43.** Draw the game tree for the game of tic-tac-toe for the levels corresponding to the first two moves. Assign the value of the evaluation function mentioned in the text that assigns to a position the number of files containing no Os minus the number of files containing no Xs as the value of each vertex at this level and compute the value of the tree for vertices as if the evaluation function gave the correct values for these vertices.
- 44.** Use pseudocode to describe an algorithm for determining the value of a game tree when both players follow a minmax strategy.

## 11.3 Tree Traversal

### Introduction



Ordered rooted trees are often used to store information. We need procedures for visiting each vertex of an ordered rooted tree to access data. We will describe several important algorithms for visiting all the vertices of an ordered rooted tree. Ordered rooted trees can also be used to represent various types of expressions, such as arithmetic expressions involving numbers, variables, and operations. The different listings of the vertices of ordered rooted trees used to represent expressions are useful in the evaluation of these expressions.

### Universal Address Systems

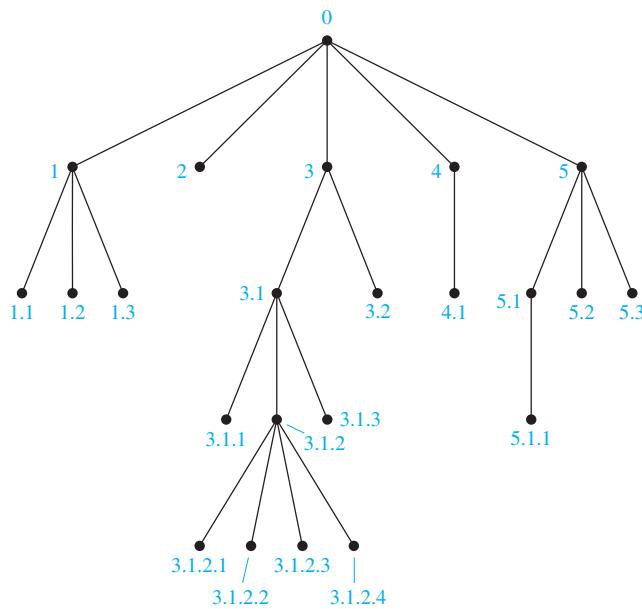
Procedures for traversing all vertices of an ordered rooted tree rely on the orderings of children. In ordered rooted trees, the children of an internal vertex are shown from left to right in the drawings representing these directed graphs.

We will describe one way we can totally order the vertices of an ordered rooted tree. To produce this ordering, we must first label all the vertices. We do this recursively:

1. Label the root with the integer 0. Then label its  $k$  children (at level 1) from left to right with  $1, 2, 3, \dots, k$ .
2. For each vertex  $v$  at level  $n$  with label  $A$ , label its  $k_v$  children, as they are drawn from left to right, with  $A.1, A.2, \dots, A.k_v$ .

Following this procedure, a vertex  $v$  at level  $n$ , for  $n \geq 1$ , is labeled  $x_1.x_2.\dots.x_n$ , where the unique path from the root to  $v$  goes through the  $x_1$ st vertex at level 1, the  $x_2$ nd vertex at level 2, and so on. This labeling is called the **universal address system** of the ordered rooted tree.

We can totally order the vertices using the lexicographic ordering of their labels in the universal address system. The vertex labeled  $x_1.x_2.\dots.x_n$  is less than the vertex labeled  $y_1.y_2.\dots.y_m$  if there is an  $i$ ,  $0 \leq i \leq n$ , with  $x_1 = y_1, x_2 = y_2, \dots, x_{i-1} = y_{i-1}$ , and  $x_i < y_i$ ; or if  $n < m$  and  $x_i = y_i$  for  $i = 1, 2, \dots, n$ .



**FIGURE 1** The Universal Address System of an Ordered Rooted Tree.

### EXAMPLE 1



We display the labelings of the universal address system next to the vertices in the ordered rooted tree shown in Figure 1. The lexicographic ordering of the labelings is

$$\begin{aligned} & 0 < 1 < 1.1 < 1.2 < 1.3 < 2 < 3 < 3.1 < 3.1.1 < 3.1.2 < 3.1.2.1 < 3.1.2.2 \\ & < 3.1.2.3 < 3.1.2.4 < 3.1.3 < 3.2 < 4 < 4.1 < 5 < 5.1 < 5.2 < 5.3 \end{aligned}$$



## Traversal Algorithms

Procedures for systematically visiting every vertex of an ordered rooted tree are called **traversal algorithms**. We will describe three of the most commonly used such algorithms, **preorder traversal**, **inorder traversal**, and **postorder traversal**. Each of these algorithms can be defined recursively. We first define preorder traversal.

### DEFINITION 1

Let  $T$  be an ordered rooted tree with root  $r$ . If  $T$  consists only of  $r$ , then  $r$  is the *preorder traversal* of  $T$ . Otherwise, suppose that  $T_1, T_2, \dots, T_n$  are the subtrees at  $r$  from left to right in  $T$ . The *preorder traversal* begins by visiting  $r$ . It continues by traversing  $T_1$  in preorder, then  $T_2$  in preorder, and so on, until  $T_n$  is traversed in preorder.

The reader should verify that the preorder traversal of an ordered rooted tree gives the same ordering of the vertices as the ordering obtained using a universal address system. Figure 2 indicates how a preorder traversal is carried out.

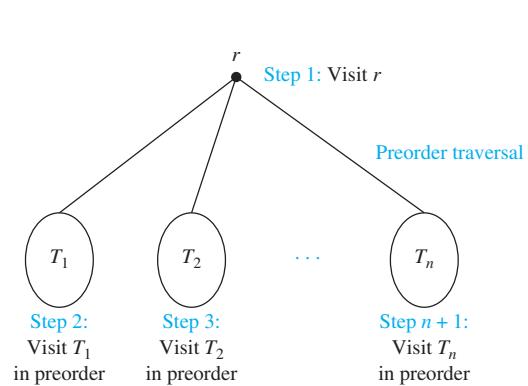
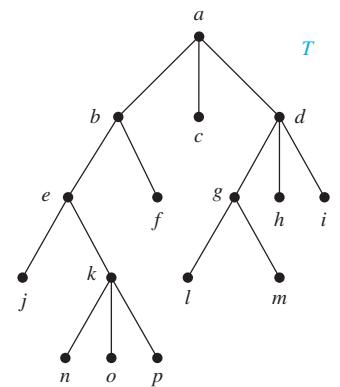
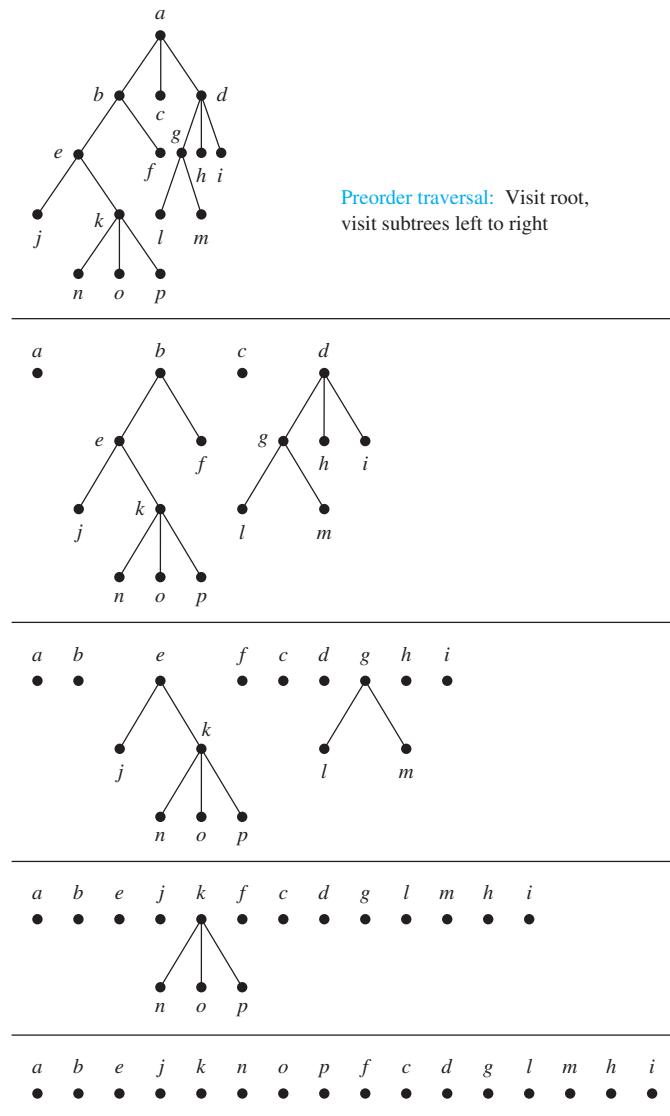
Example 2 illustrates preorder traversal.

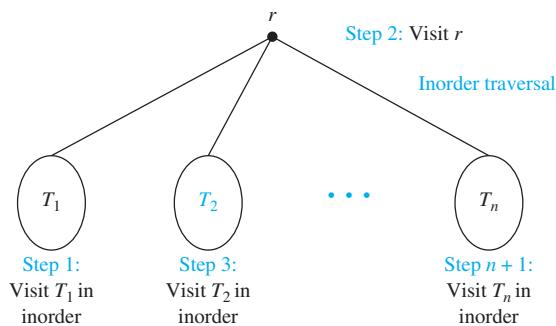
### EXAMPLE 2

In which order does a preorder traversal visit the vertices in the ordered rooted tree  $T$  shown in Figure 3?



**Solution:** The steps of the preorder traversal of  $T$  are shown in Figure 4. We traverse  $T$  in preorder by first listing the root  $a$ , followed by the preorder list of the subtree with root  $b$ , the preorder list of the subtree with root  $c$  (which is just  $c$ ) and the preorder list of the subtree with root  $d$ .

**FIGURE 2** Preorder Traversal.**FIGURE 3** The Ordered Rooted Tree  $T$ .**FIGURE 4** The Preorder Traversal of  $T$ .

**FIGURE 5 Inorder Traversal.**

The preorder list of the subtree with root  $b$  begins by listing  $b$ , then the vertices of the subtree with root  $e$  in preorder, and then the subtree with root  $f$  in preorder (which is just  $f$ ). The preorder list of the subtree with root  $d$  begins by listing  $d$ , followed by the preorder list of the subtree with root  $g$ , followed by the subtree with root  $h$  (which is just  $h$ ), followed by the subtree with root  $i$  (which is just  $i$ ).

The preorder list of the subtree with root  $e$  begins by listing  $e$ , followed by the preorder listing of the subtree with root  $j$  (which is just  $j$ ), followed by the preorder listing of the subtree with root  $k$ . The preorder listing of the subtree with root  $g$  is  $g$  followed by  $l$ , followed by  $m$ . The preorder listing of the subtree with root  $k$  is  $k, n, o, p$ . Consequently, the preorder traversal of  $T$  is  $a, b, e, j, k, n, o, p, f, c, d, g, l, m, h, i$ .  $\blacktriangleleft$

We will now define inorder traversal.

### DEFINITION 2

Let  $T$  be an ordered rooted tree with root  $r$ . If  $T$  consists only of  $r$ , then  $r$  is the *inorder traversal* of  $T$ . Otherwise, suppose that  $T_1, T_2, \dots, T_n$  are the subtrees at  $r$  from left to right. The *inorder traversal* begins by traversing  $T_1$  in inorder, then visiting  $r$ . It continues by traversing  $T_2$  in inorder, then  $T_3$  in inorder,  $\dots$ , and finally  $T_n$  in inorder.

Figure 5 indicates how inorder traversal is carried out. Example 3 illustrates how inorder traversal is carried out for a particular tree.

### EXAMPLE 3

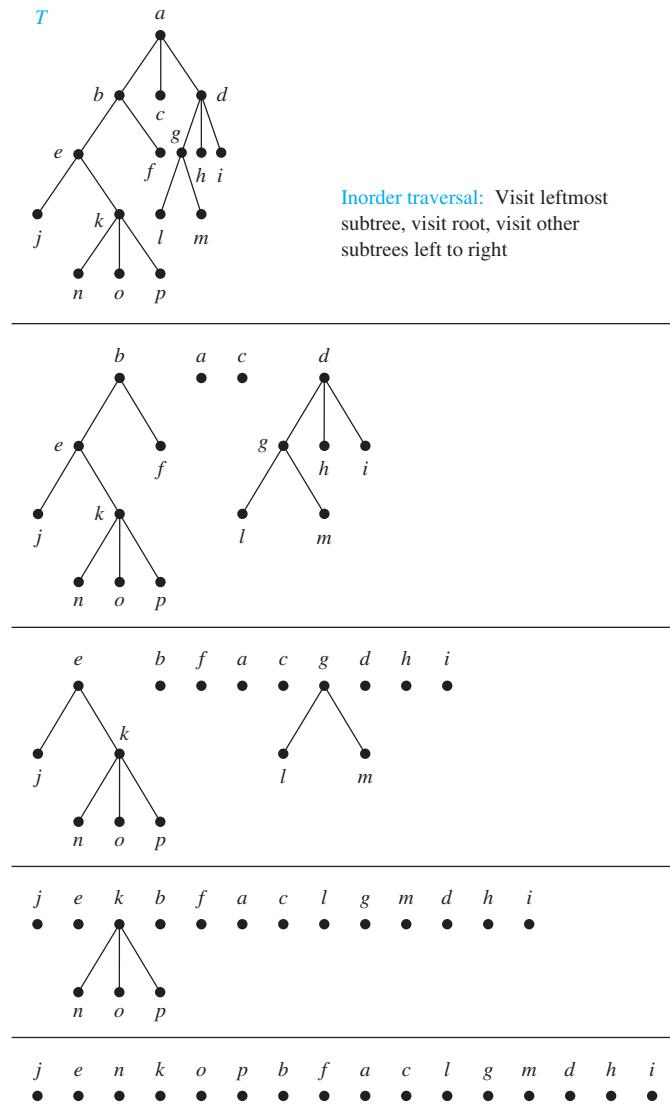
In which order does an inorder traversal visit the vertices of the ordered rooted tree  $T$  in Figure 3?



**Solution:** The steps of the inorder traversal of the ordered rooted tree  $T$  are shown in Figure 6. The inorder traversal begins with an inorder traversal of the subtree with root  $b$ , the root  $a$ , the inorder listing of the subtree with root  $c$ , which is just  $c$ , and the inorder listing of the subtree with root  $d$ .

The inorder listing of the subtree with root  $b$  begins with the inorder listing of the subtree with root  $e$ , the root  $b$ , and  $f$ . The inorder listing of the subtree with root  $d$  begins with the inorder listing of the subtree with root  $g$ , followed by the root  $d$ , followed by  $h$ , followed by  $i$ .

The inorder listing of the subtree with root  $e$  is  $j$ , followed by the root  $e$ , followed by the inorder listing of the subtree with root  $k$ . The inorder listing of the subtree with root  $g$  is  $l, g, m$ . The inorder listing of the subtree with root  $k$  is  $n, k, o, p$ . Consequently, the inorder listing of the ordered rooted tree is  $j, e, n, k, o, p, b, f, a, c, l, g, m, d, h, i$ .  $\blacktriangleleft$

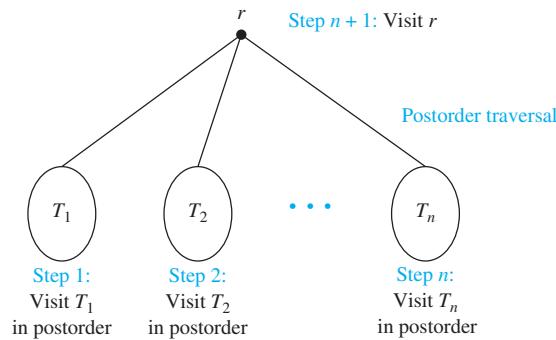
FIGURE 6 The Inorder Traversal of  $T$ .

We now define postorder traversal.

### DEFINITION 3

Let  $T$  be an ordered rooted tree with root  $r$ . If  $T$  consists only of  $r$ , then  $r$  is the *postorder traversal* of  $T$ . Otherwise, suppose that  $T_1, T_2, \dots, T_n$  are the subtrees at  $r$  from left to right. The *postorder traversal* begins by traversing  $T_1$  in postorder, then  $T_2$  in postorder,  $\dots$ , then  $T_n$  in postorder, and ends by visiting  $r$ .

Figure 7 illustrates how postorder traversal is done. Example 4 illustrates how postorder traversal works.



**FIGURE 7 Postorder Traversal.**

**EXAMPLE 4** In which order does a postorder traversal visit the vertices of the ordered rooted tree  $T$  shown in Figure 3?



**Solution:** The steps of the postorder traversal of the ordered rooted tree  $T$  are shown in Figure 8. The postorder traversal begins with the postorder traversal of the subtree with root  $b$ , the postorder traversal of the subtree with root  $c$ , which is just  $c$ , the postorder traversal of the subtree with root  $d$ , followed by the root  $a$ .

The postorder traversal of the subtree with root  $b$  begins with the postorder traversal of the subtree with root  $e$ , followed by  $f$ , followed by the root  $b$ . The postorder traversal of the rooted tree with root  $d$  begins with the postorder traversal of the subtree with root  $g$ , followed by  $h$ , followed by  $i$ , followed by the root  $d$ .

The postorder traversal of the subtree with root  $e$  begins with  $j$ , followed by the postorder traversal of the subtree with root  $k$ , followed by the root  $e$ . The postorder traversal of the subtree with root  $g$  is  $l, m, g$ . The postorder traversal of the subtree with root  $k$  is  $n, o, p, k$ . Therefore, the postorder traversal of  $T$  is  $j, n, o, p, k, e, f, b, c, l, m, g, h, i, d, a$ .

There are easy ways to list the vertices of an ordered rooted tree in preorder, inorder, and postorder. To do this, first draw a curve around the ordered rooted tree starting at the root, moving along the edges, as shown in the example in Figure 9. We can list the vertices in preorder by listing each vertex the first time this curve passes it. We can list the vertices in inorder by listing a leaf the first time the curve passes it and listing each internal vertex the second time the curve passes it. We can list the vertices in postorder by listing a vertex the last time it is passed on the way back up to its parent. When this is done in the rooted tree in Figure 9, it follows that the preorder traversal gives  $a, b, d, h, e, i, j, c, f, g, k$ , the inorder traversal gives  $h, d, b, i, e, j, a, f, c, k, g$ ; and the postorder traversal gives  $h, d, i, j, e, b, f, k, g, c, a$ .

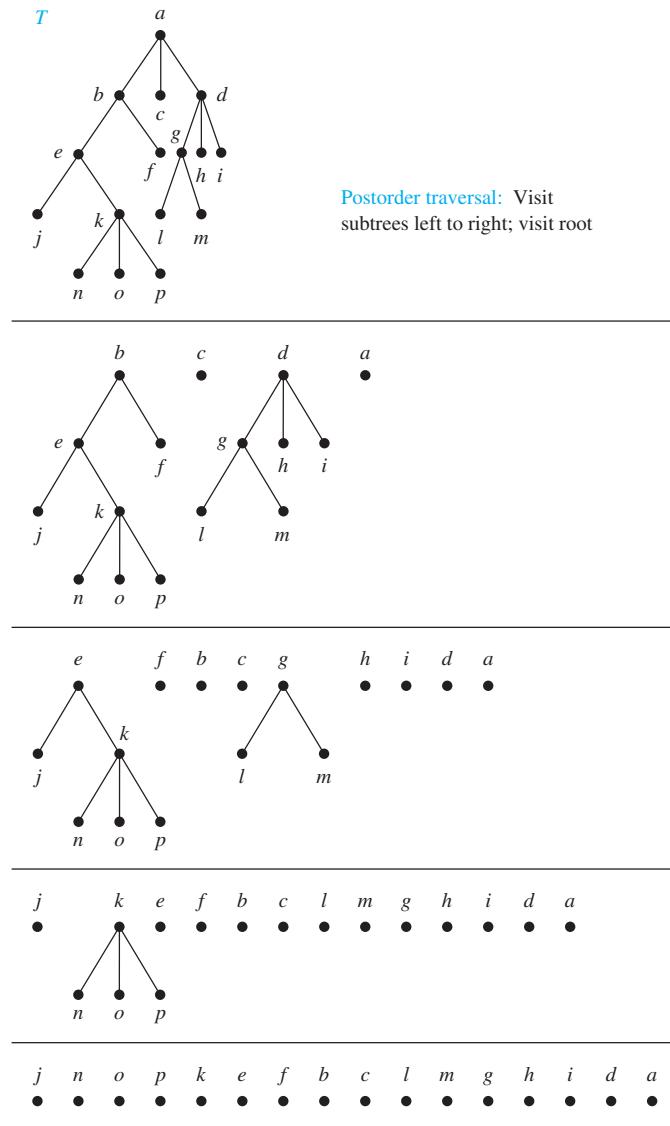
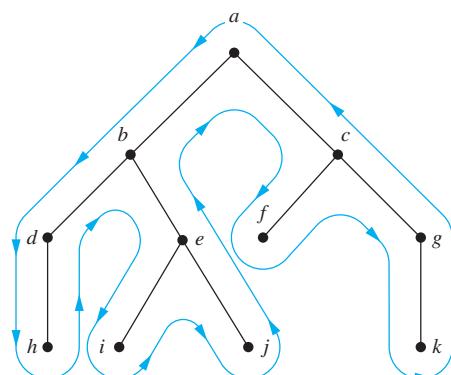
Algorithms for traversing ordered rooted trees in preorder, inorder, or postorder are most easily expressed recursively.

**ALGORITHM 1 Preorder Traversal.**

```

procedure preorder(T : ordered rooted tree)
 $r :=$ root of T
list r
for each child c of r from left to right
 $T(c) :=$ subtree with c as its root
 preorder($T(c)$)

```

**FIGURE 8** The Postorder Traversal of  $T$ .**FIGURE 9** A Shortcut for Traversing an Ordered Rooted Tree in Preorder, Inorder, and Postorder.

**ALGORITHM 2 Inorder Traversal.**

```

procedure inorder(T : ordered rooted tree)
 $r :=$ root of T
if r is a leaf then list r
else
 $l :=$ first child of r from left to right
 $T(l) :=$ subtree with l as its root
 inorder($T(l)$)
 list r
 for each child c of r except for l from left to right
 $T(c) :=$ subtree with c as its root
 inorder($T(c)$)

```

**ALGORITHM 3 Postorder Traversal.**

```

procedure postorder(T : ordered rooted tree)
 $r :=$ root of T
for each child c of r from left to right
 $T(c) :=$ subtree with c as its root
 postorder($T(c)$)
list r

```

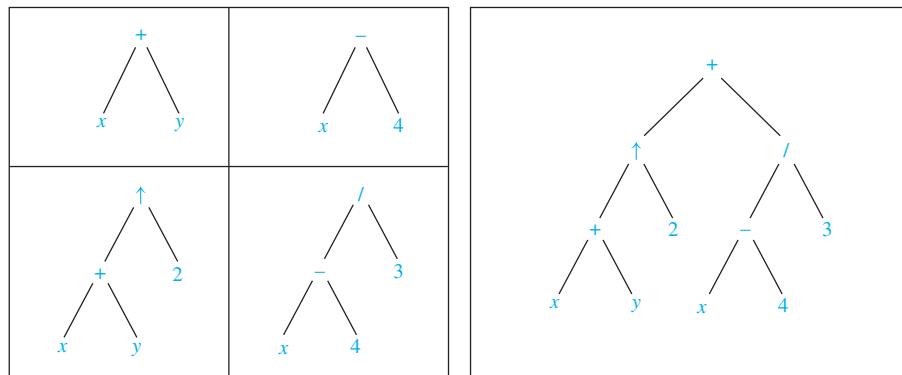
Note that both the preorder traversal and the postorder traversal encode the structure of an ordered rooted tree when the number of children of each vertex is specified. That is, an ordered rooted tree is uniquely determined when we specify a list of vertices generated by a preorder traversal or by a postorder traversal of the tree, together with the number of children of each vertex (see Exercises 26 and 27). In particular, both a preorder traversal and a postorder traversal encode the structure of a full ordered  $m$ -ary tree. However, when the number of children of vertices is not specified, neither a preorder traversal nor a postorder traversal encodes the structure of an ordered rooted tree (see Exercises 28 and 29).

## Infix, Prefix, and Postfix Notation

We can represent complicated expressions, such as compound propositions, combinations of sets, and arithmetic expressions using ordered rooted trees. For instance, consider the representation of an arithmetic expression involving the operators  $+$  (addition),  $-$  (subtraction),  $*$  (multiplication),  $/$  (division), and  $\uparrow$  (exponentiation). We will use parentheses to indicate the order of the operations. An ordered rooted tree can be used to represent such expressions, where the internal vertices represent operations, and the leaves represent the variables or numbers. Each operation operates on its left and right subtrees (in that order).

**EXAMPLE 5** What is the ordered rooted tree that represents the expression  $((x + y) \uparrow 2) + ((x - 4)/3)$ ?

**Solution:** The binary tree for this expression can be built from the bottom up. First, a subtree for the expression  $x + y$  is constructed. Then this is incorporated as part of the larger subtree representing  $(x + y) \uparrow 2$ . Also, a subtree for  $x - 4$  is constructed, and then this is incorporated into a subtree representing  $(x - 4)/3$ . Finally the subtrees representing  $(x + y) \uparrow 2$



**FIGURE 10** A Binary Tree Representing  $((x + y) \uparrow 2) + ((x - 4)/3)$ .

and  $(x - 4)/3$  are combined to form the ordered rooted tree representing  $((x + y) \uparrow 2) + ((x - 4)/3)$ . These steps are shown in Figure 10.  $\blacktriangleleft$

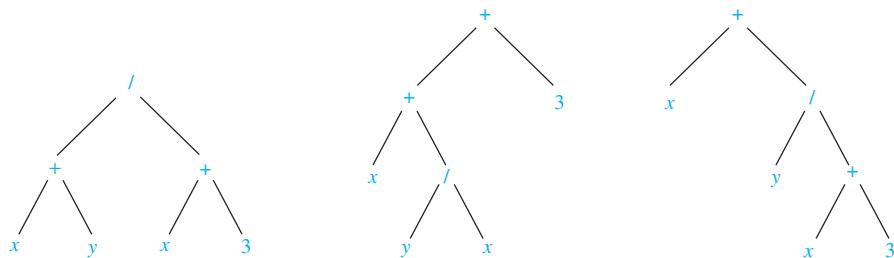
An inorder traversal of the binary tree representing an expression produces the original expression with the elements and operations in the same order as they originally occurred, except for unary operations, which instead immediately follow their operands. For instance, inorder traversals of the binary trees in Figure 11, which represent the expressions  $(x + y)/(x + 3)$ ,  $(x + (y/x)) + 3$ , and  $x + (y/(x + 3))$ , all lead to the infix expression  $x + y/x + 3$ . To make such expressions unambiguous it is necessary to include parentheses in the inorder traversal whenever we encounter an operation. The fully parenthesized expression obtained in this way is said to be in **infix form**.

We obtain the **prefix form** of an expression when we traverse its rooted tree in preorder. Expressions written in prefix form are said to be in **Polish notation**, which is named after the Polish logician Jan Łukasiewicz. An expression in prefix notation (where each operation has a specified number of operands), is unambiguous, so no parentheses are needed in such an expression. The verification of this is left as an exercise for the reader.

**EXAMPLE 6** What is the prefix form for  $((x + y) \uparrow 2) + ((x - 4)/3)$ ?

*Solution:* We obtain the prefix form for this expression by traversing the binary tree that represents it in preorder, shown in Figure 10. This produces  $+ \uparrow + x y 2 / - x 4 3$ .  $\blacktriangleleft$

In the prefix form of an expression, a binary operator, such as  $+$ , precedes its two operands. Hence, we can evaluate an expression in prefix form by working from right to left. When we encounter an operator, we perform the corresponding operation with the two operands



**FIGURE 11** Rooted Trees Representing  $(x + y)/(x + 3)$ ,  $(x + (y/x)) + 3$ , and  $x + (y/(x + 3))$ .

$$\begin{array}{ccccccccc}
 + & - & * & 2 & 3 & 5 & / & \uparrow & 2 & 3 & 4 \\
 & & & & & & & \text{2} & \uparrow & 3 = 8 \\
 + & - & * & 2 & 3 & 5 & / & 8 & 4 & \\
 & & & & & & & \text{8} & / & 4 = 2 \\
 + & - & * & 2 & 3 & 5 & 2 & & \\
 & & & \text{2} & * & 3 = 6 & & \\
 + & - & 6 & 5 & 2 & & & \\
 & & \text{6} & - & 5 = 1 & & & \\
 + & 1 & 2 & & & & & \\
 & \text{1} & + & 2 = 3 & & & & \\
 \end{array}$$

Value of expression: 3

**FIGURE 12 Evaluating a Prefix Expression.**

$$\begin{array}{ccccccccc}
 7 & 2 & 3 & * & - & 4 & \uparrow & 9 & 3 & / & + \\
 & \text{2} & * & 3 = 6 & & & & \\
 7 & 6 & - & 4 & \uparrow & 9 & 3 & / & + \\
 & \text{7} & - & 6 = 1 & & & & \\
 1 & 4 & \uparrow & 9 & 3 & / & + & \\
 & \text{1} & ^4 = 1 & & & & & \\
 1 & 9 & 3 & / & + & & & \\
 & \text{9} & / & 3 = 3 & & & & \\
 1 & 3 & + & & & & & \\
 & \text{1} & + & 3 = 4 & & & & \\
 \end{array}$$

Value of expression: 4

**FIGURE 13 Evaluating a Postfix Expression.**

immediately to the right of this operand. Also, whenever an operation is performed, we consider the result a new operand.

**EXAMPLE 7** What is the value of the prefix expression  $+ - * 2 3 5 / \uparrow 2 3 4$ ?

**Solution:** The steps used to evaluate this expression by working right to left, and performing operations using the operands on the right, are shown in Figure 12. The value of this expression is 3.



Reverse polish notation was first proposed in 1954 by Burks, Warren, and Wright.

We obtain the **postfix form** of an expression by traversing its binary tree in postorder. Expressions written in postfix form are said to be in **reverse Polish notation**. Expressions in reverse Polish notation are unambiguous, so parentheses are not needed. The verification of this is left to the reader. Reverse polish notation was extensively used in electronic calculators in the 1970s and 1980s.

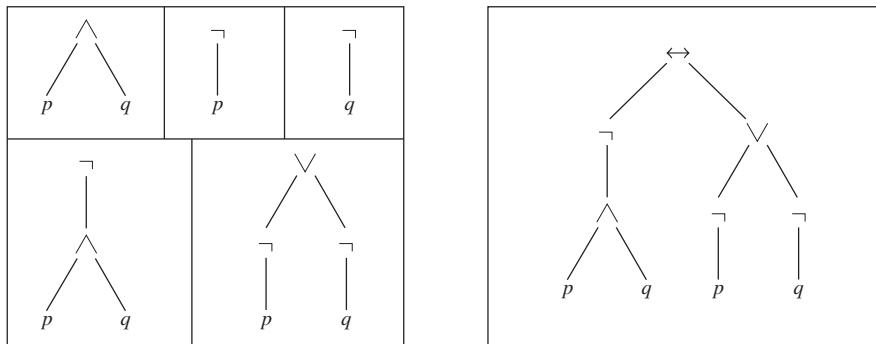
**EXAMPLE 8** What is the postfix form of the expression  $((x + y) \uparrow 2) + ((x - 4)/3)$ ?

**Solution:** The postfix form of the expression is obtained by carrying out a postorder traversal of the binary tree for this expression, shown in Figure 10. This produces the postfix expression:  $x y + 2 \uparrow x 4 - 3 / +$ .

In the postfix form of an expression, a binary operator follows its two operands. So, to evaluate an expression from its postfix form, work from left to right, carrying out operations whenever an operator follows two operands. After an operation is carried out, the result of this operation becomes a new operand.

**EXAMPLE 9** What is the value of the postfix expression  $7 2 3 * - 4 \uparrow 9 3 / +$ ?

**Solution:** The steps used to evaluate this expression by starting at the left and carrying out operations when two operands are followed by an operator are shown in Figure 13. The value of this expression is 4.



**FIGURE 14** Constructing the Rooted Tree for a Compound Proposition.

Rooted trees can be used to represent other types of expressions, such as those representing compound propositions and combinations of sets. In these examples unary operators, such as the negation of a proposition, occur. To represent such operators and their operands, a vertex representing the operator and a child of this vertex representing the operand are used.

**EXAMPLE 10** Find the ordered rooted tree representing the compound proposition  $(\neg(p \wedge q)) \leftrightarrow (\neg p \vee \neg q)$ . Then use this rooted tree to find the prefix, postfix, and infix forms of this expression.



**Solution:** The rooted tree for this compound proposition is constructed from the bottom up. First, subtrees for  $\neg p$  and  $\neg q$  are formed (where  $\neg$  is considered a unary operator). Also, a subtree for  $p \wedge q$  is formed. Then subtrees for  $\neg(p \wedge q)$  and  $(\neg p) \vee (\neg q)$  are constructed. Finally, these two subtrees are used to form the final rooted tree. The steps of this procedure are shown in Figure 14.

The prefix, postfix, and infix forms of this expression are found by traversing this rooted tree in preorder, postorder, and inorder (including parentheses), respectively. These traversals give  $\leftrightarrow \neg \wedge p q \vee \neg p \neg q$ ,  $p q \wedge \neg p \neg q \neg \vee \leftrightarrow$ , and  $(\neg(p \wedge q)) \leftrightarrow ((\neg p) \vee (\neg q))$ , respectively.

Because prefix and postfix expressions are unambiguous and because they can be evaluated easily without scanning back and forth, they are used extensively in computer science. Such expressions are especially useful in the construction of compilers.



**JAN ŁUKASIEWICZ (1878–1956)** Jan Łukasiewicz was born into a Polish-speaking family in Lvov. At that time Lvov was part of Austria, but it is now in the Ukraine. His father was a captain in the Austrian army. Łukasiewicz became interested in mathematics while in high school. He studied mathematics and philosophy at the University of Lvov at both the undergraduate and graduate levels. After completing his doctoral work he became a lecturer there, and in 1911 he was appointed to a professorship. When the University of Warsaw was reopened as a Polish university in 1915, Łukasiewicz accepted an invitation to join the faculty. In 1919 he served as the Polish Minister of Education. He returned to the position of professor at Warsaw University where he remained from 1920 to 1939, serving as rector of the university twice.

Łukasiewicz was one of the cofounders of the famous Warsaw School of Logic. He published his famous text, *Elements of Mathematical Logic*, in 1928. With his influence, mathematical logic was made a required course for mathematics and science undergraduates in Poland. His lectures were considered excellent, even attracting students of the humanities.

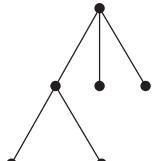
Łukasiewicz and his wife experienced great suffering during World War II, which he documented in a posthumously published autobiography. After the war they lived in exile in Belgium. Fortunately, in 1949 he was offered a position at the Royal Irish Academy in Dublin.

Łukasiewicz worked on mathematical logic throughout his career. His work on a three-valued logic was an important contribution to the subject. Nevertheless, he is best known in the mathematical and computer science communities for his introduction of parenthesis-free notation, now called Polish notation.

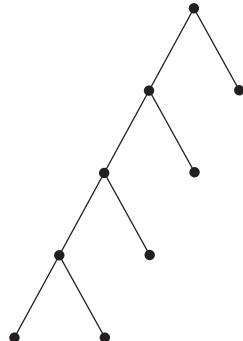
## Exercises

In Exercises 1–3 construct the universal address system for the given ordered rooted tree. Then use this to order its vertices using the lexicographic order of their labels.

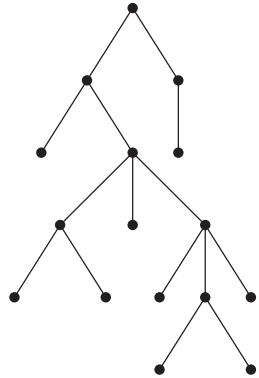
1.



2.



3.



4. Suppose that the address of the vertex  $v$  in the ordered rooted tree  $T$  is 3.4.5.2.4.

- a) At what level is  $v$ ?
- b) What is the address of the parent of  $v$ ?
- c) What is the least number of siblings  $v$  can have?
- d) What is the smallest possible number of vertices in  $T$  if  $v$  has this address?
- e) Find the other addresses that must occur.

5. Suppose that the vertex with the largest address in an ordered rooted tree  $T$  has address 2.3.4.3.1. Is it possible to determine the number of vertices in  $T$ ?

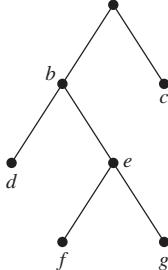
6. Can the leaves of an ordered rooted tree have the following list of universal addresses? If so, construct such an ordered rooted tree.

- a) 1.1.1, 1.1.2, 1.2, 2.1.1.1, 2.1.2, 2.1.3, 2.2, 3.1.1, 3.1.2.1, 3.1.2.2, 3.2

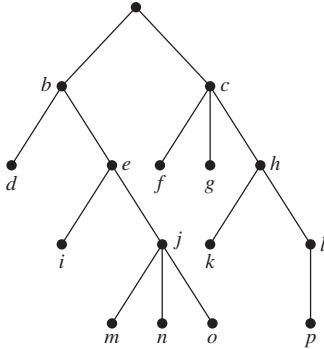
- b) 1.1, 1.2.1, 1.2.2, 1.2.3, 2.1, 2.2.1, 2.3.1, 2.3.2, 2.4.2.1, 2.4.2.2, 3.1, 3.2.1, 3.2.2
- c) 1.1, 1.2.1, 1.2.2, 1.2.2.1, 1.3, 1.4, 2, 3.1, 3.2, 4.1.1.1

In Exercises 7–9 determine the order in which a preorder traversal visits the vertices of the given ordered rooted tree.

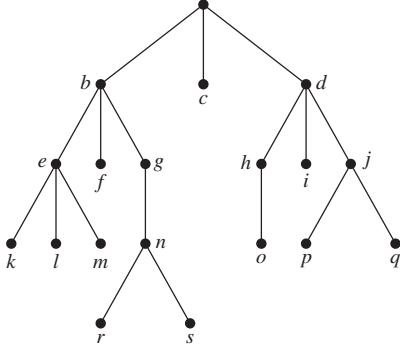
7.



8.



9.



- 10. In which order are the vertices of the ordered rooted tree in Exercise 7 visited using an inorder traversal?
- 11. In which order are the vertices of the ordered rooted tree in Exercise 8 visited using an inorder traversal?
- 12. In which order are the vertices of the ordered rooted tree in Exercise 9 visited using an inorder traversal?
- 13. In which order are the vertices of the ordered rooted tree in Exercise 7 visited using a postorder traversal?
- 14. In which order are the vertices of the ordered rooted tree in Exercise 8 visited using a postorder traversal?
- 15. In which order are the vertices of the ordered rooted tree in Exercise 9 visited using a postorder traversal?

- 16.** a) Represent the expression  $((x+2) \uparrow 3) * (y - (3+x)) - 5$  using a binary tree.

Write this expression in

- b) prefix notation.  
c) postfix notation.  
d) infix notation.

- 17.** a) Represent the expressions  $(x+xy)+(x/y)$  and  $x+((xy+x)/y)$  using binary trees.

Write these expressions in

- b) prefix notation.  
c) postfix notation.  
d) infix notation.

- 18.** a) Represent the compound propositions  $\neg(p \wedge q) \leftrightarrow (\neg p \vee \neg q)$  and  $(\neg p \wedge (q \leftrightarrow \neg p)) \vee \neg q$  using ordered rooted trees.

Write these expressions in

- b) prefix notation.  
c) postfix notation.  
d) infix notation.

- 19.** a) Represent  $(A \cap B) - (A \cup (B - A))$  using an ordered rooted tree.

Write this expression in

- b) prefix notation.  
c) postfix notation.  
d) infix notation.

- \***20.** In how many ways can the string  $\neg p \wedge q \leftrightarrow \neg p \vee \neg q$  be fully parenthesized to yield an infix expression?

- \***21.** In how many ways can the string  $A \cap B - A \cap B - A$  be fully parenthesized to yield an infix expression?

- 22.** Draw the ordered rooted tree corresponding to each of these arithmetic expressions written in prefix notation. Then write each expression using infix notation.

- a)  $+ * + - 5 3 2 1 4$   
b)  $\uparrow + 2 3 - 5 1$   
c)  $* / 9 3 + * 2 4 - 7 6$

- 23.** What is the value of each of these prefix expressions?

- a)  $- * 2 / 8 4 3$   
b)  $\uparrow - * 3 3 * 4 2 5$   
c)  $+ - \uparrow 3 2 \uparrow 2 3 / 6 - 4 2$   
d)  $* + 3 + 3 \uparrow 3 + 3 3$

- 24.** What is the value of each of these postfix expressions?

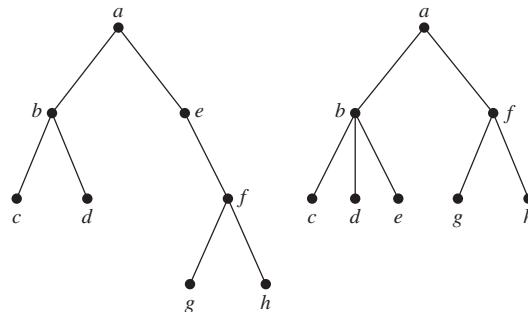
- a)  $5 2 1 - - 3 1 4 ++ *$   
b)  $9 3 / 5 + 7 2 - *$   
c)  $3 2 * 2 \uparrow 5 3 - 8 4 / * -$

- 25.** Construct the ordered rooted tree whose preorder traversal is  $a, b, f, c, g, h, i, d, e, j, k, l$ , where  $a$  has four children,  $c$  has three children,  $j$  has two children,  $b$  and  $e$  have one child each, and all other vertices are leaves.

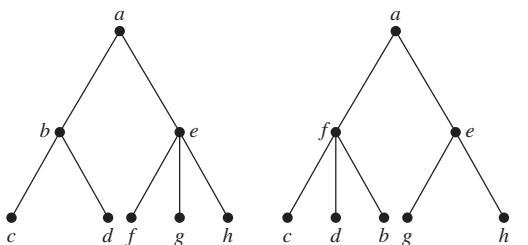
- \***26.** Show that an ordered rooted tree is uniquely determined when a list of vertices generated by a preorder traversal of the tree and the number of children of each vertex are specified.

- \***27.** Show that an ordered rooted tree is uniquely determined when a list of vertices generated by a postorder traversal of the tree and the number of children of each vertex are specified.

- 28.** Show that preorder traversals of the two ordered rooted trees displayed below produce the same list of vertices. Note that this does not contradict the statement in Exercise 26, because the numbers of children of internal vertices in the two ordered rooted trees differ.



- 29.** Show that postorder traversals of these two ordered rooted trees produce the same list of vertices. Note that this does not contradict the statement in Exercise 27, because the numbers of children of internal vertices in the two ordered rooted trees differ.



**Well-formed formulae** in prefix notation over a set of symbols and a set of binary operators are defined recursively by these rules:

- (i) if  $x$  is a symbol, then  $x$  is a well-formed formula in prefix notation;  
(ii) if  $X$  and  $Y$  are well-formed formulae and  $*$  is an operator, then  $*XY$  is a well-formed formula.

- 30.** Which of these are well-formed formulae over the symbols  $\{x, y, z\}$  and the set of binary operators  $\{\times, +, \circ\}$ ?

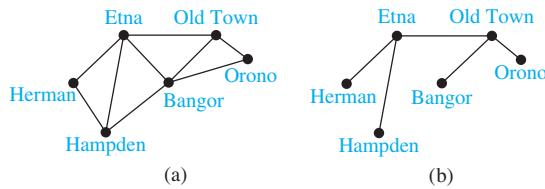
- a)  $\times + + x y x$   
b)  $\circ x y \times x z$   
c)  $\times \circ x z \times \times x y$   
d)  $\times + \circ x x \circ x x x$

- \***31.** Show that any well-formed formula in prefix notation over a set of symbols and a set of binary operators contains exactly one more symbol than the number of operators.

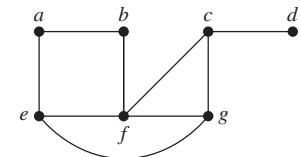
- 32.** Give a definition of well-formed formulae in postfix notation over a set of symbols and a set of binary operators.

- 33.** Give six examples of well-formed formulae with three or more operators in postfix notation over the set of symbols  $\{x, y, z\}$  and the set of operators  $\{+, \times, \circ\}$ .

- 34.** Extend the definition of well-formed formulae in prefix notation to sets of symbols and operators where the operators may not be binary.



**FIGURE 1** (a) A Road System and (b) a Set of Roads to Plow.



## FIGURE 2 The Simple Graph $G$ .

## 11.4 Spanning Trees

## Introduction

Consider the system of roads in Maine represented by the simple graph shown in Figure 1(a). The only way the roads can be kept open in the winter is by frequently plowing them. The highway department wants to plow the fewest roads so that there will always be cleared roads connecting any two towns. How can this be done?

At least five roads must be plowed to ensure that there is a path between any two towns. Figure 1(b) shows one such set of roads. Note that the subgraph representing these roads is a tree, because it is connected and contains six vertices and five edges.

This problem was solved with a connected subgraph with the minimum number of edges containing all vertices of the original simple graph. Such a graph must be a tree.

## DEFINITION 1

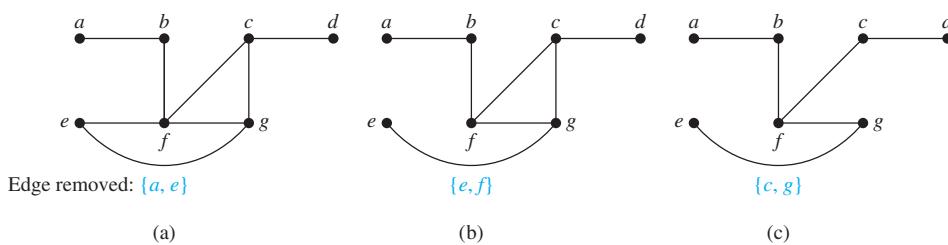
Let  $G$  be a simple graph. A *spanning tree* of  $G$  is a subgraph of  $G$  that is a tree containing every vertex of  $G$ .

A simple graph with a spanning tree must be connected, because there is a path in the spanning tree between any two vertices. The converse is also true; that is, every connected simple graph has a spanning tree. We will give an example before proving this result.

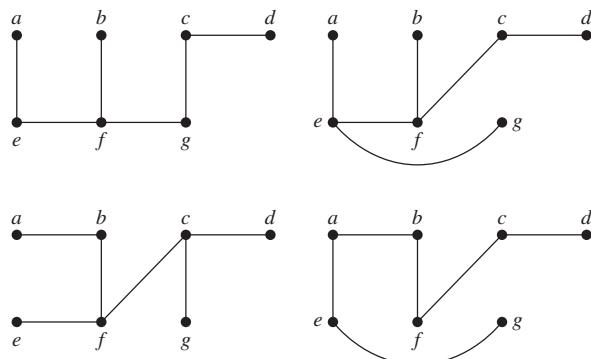
## EXAMPLE 1

Find a spanning tree of the simple graph  $G$  shown in Figure 2.

**Solution:** The graph  $G$  is connected, but it is not a tree because it contains simple circuits. Remove the edge  $\{a, e\}$ . This eliminates one simple circuit, and the resulting subgraph is still connected and still contains every vertex of  $G$ . Next remove the edge  $\{e, f\}$  to eliminate a second simple circuit. Finally, remove edge  $\{c, g\}$  to produce a simple graph with no simple circuits. This subgraph is a spanning tree, because it is a tree that contains every vertex of  $G$ . The sequence of edge removals used to produce the spanning tree is illustrated in Figure 3.



**FIGURE 3** Producing a Spanning Tree for  $G$  by Removing Edges That Form Simple Circuits.

**FIGURE 4** Spanning Trees of  $G$ .

The tree shown in Figure 3 is not the only spanning tree of  $G$ . For instance, each of the trees shown in Figure 4 is a spanning tree of  $G$ .  $\blacktriangleleft$

**THEOREM 1**

A simple graph is connected if and only if it has a spanning tree.

**Proof:** First, suppose that a simple graph  $G$  has a spanning tree  $T$ .  $T$  contains every vertex of  $G$ . Furthermore, there is a path in  $T$  between any two of its vertices. Because  $T$  is a subgraph of  $G$ , there is a path in  $G$  between any two of its vertices. Hence,  $G$  is connected.

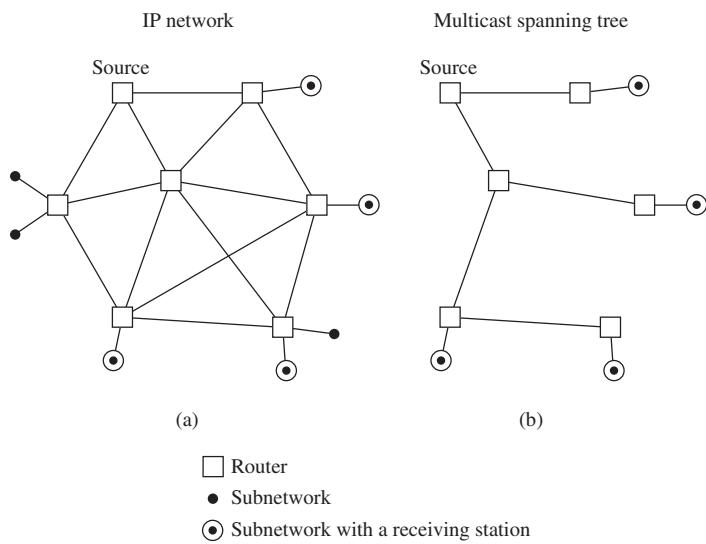
Now suppose that  $G$  is connected. If  $G$  is not a tree, it must contain a simple circuit. Remove an edge from one of these simple circuits. The resulting subgraph has one fewer edge but still contains all the vertices of  $G$  and is connected. This subgraph is still connected because when two vertices are connected by a path containing the removed edge, they are connected by a path not containing this edge. We can construct such a path by inserting into the original path, at the point where the removed edge once was, the simple circuit with this edge removed. If this subgraph is not a tree, it has a simple circuit; so as before, remove an edge that is in a simple circuit. Repeat this process until no simple circuits remain. This is possible because there are only a finite number of edges in the graph. The process terminates when no simple circuits remain. A tree is produced because the graph stays connected as edges are removed. This tree is a spanning tree because it contains every vertex of  $G$ .  $\blacktriangleleft$

Spanning trees are important in data networking, as Example 2 shows.

**EXAMPLE 2**

**IP Multicasting** Spanning trees play an important role in multicasting over Internet Protocol (IP) networks. To send data from a source computer to multiple receiving computers, each of which is a subnetwork, data could be sent separately to each computer. This type of networking, called unicasting, is inefficient, because many copies of the same data are transmitted over the network. To make the transmission of data to multiple receiving computers more efficient, IP multicasting is used. With IP multicasting, a computer sends a single copy of data over the network, and as data reaches intermediate routers, the data are forwarded to one or more other routers so that ultimately all receiving computers in their various subnetworks receive these data. (Routers are computers that are dedicated to forwarding IP datagrams between subnetworks in a network. In multicasting, routers use Class D addresses, each representing a session that receiving computers may join; see Example 17 in Section 6.1.)

For data to reach receiving computers as quickly as possible, there should be no loops (which in graph theory terminology are circuits or cycles) in the path that data take through the network. That is, once data have reached a particular router, data should never return to this



**FIGURE 5** A Multicast Spanning Tree.

router. To avoid loops, the multicast routers use network algorithms to construct a spanning tree in the graph that has the multicast source, the routers, and the subnetworks containing receiving computers as vertices, with edges representing the links between computers and/or routers. The root of this spanning tree is the multicast source. The subnetworks containing receiving computers are leaves of the tree. (Note that subnetworks not containing receiving stations are not included in the graph.) This is illustrated in Figure 5. ◀

## Depth-First Search

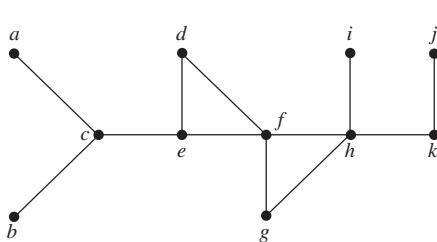
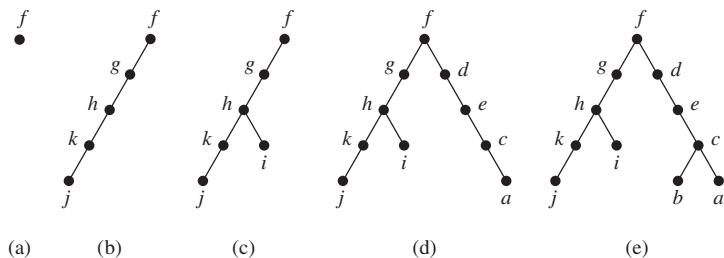


The proof of Theorem 1 gives an algorithm for finding spanning trees by removing edges from simple circuits. This algorithm is inefficient, because it requires that simple circuits be identified. Instead of constructing spanning trees by removing edges, spanning trees can be built up by successively adding edges. Two algorithms based on this principle will be presented here.



We can build a spanning tree for a connected simple graph using **depth-first search**. We will form a rooted tree, and the spanning tree will be the underlying undirected graph of this rooted tree. Arbitrarily choose a vertex of the graph as the root. Form a path starting at this vertex by successively adding vertices and edges, where each new edge is incident with the last vertex in the path and a vertex not already in the path. Continue adding vertices and edges to this path as long as possible. If the path goes through all vertices of the graph, the tree consisting of this path is a spanning tree. However, if the path does not go through all vertices, more vertices and edges must be added. Move back to the next to last vertex in the path, and, if possible, form a new path starting at this vertex passing through vertices that were not already visited. If this cannot be done, move back another vertex in the path, that is, two vertices back in the path, and try again.

Repeat this procedure, beginning at the last vertex visited, moving back up the path one vertex at a time, forming new paths that are as long as possible until no more edges can be added. Because the graph has a finite number of edges and is connected, this process ends with the production of a spanning tree. Each vertex that ends a path at a stage of the algorithm will be a leaf in the rooted tree, and each vertex where a path is constructed starting at this vertex will be an internal vertex.

FIGURE 6 The Graph  $G$ .FIGURE 7 Depth-First Search of  $G$ .

The reader should note the recursive nature of this procedure. Also, note that if the vertices in the graph are ordered, the choices of edges at each stage of the procedure are all determined when we always choose the first vertex in the ordering that is available. However, we will not always explicitly order the vertices of a graph.

Depth-first search is also called **backtracking**, because the algorithm returns to vertices previously visited to add paths. Example 3 illustrates backtracking.

**EXAMPLE 3** Use depth-first search to find a spanning tree for the graph  $G$  shown in Figure 6.



**Solution:** The steps used by depth-first search to produce a spanning tree of  $G$  are shown in Figure 7. We arbitrarily start with the vertex  $f$ . A path is built by successively adding edges incident with vertices not already in the path, as long as this is possible. This produces a path  $f, g, h, k, j$  (note that other paths could have been built). Next, backtrack to  $k$ . There is no path beginning at  $k$  containing vertices not already visited. So we backtrack to  $h$ . Form the path  $h, i$ . Then backtrack to  $h$ , and then to  $f$ . From  $f$  build the path  $f, d, e, c, a$ . Then backtrack to  $c$  and form the path  $c, b$ . This produces the spanning tree.

The edges selected by depth-first search of a graph are called **tree edges**. All other edges of the graph must connect a vertex to an ancestor or descendant of this vertex in the tree. These edges are called **back edges**. (Exercise 43 asks for a proof of this fact.)

**EXAMPLE 4** In Figure 8 we highlight the tree edges found by depth-first search starting at vertex  $f$  by showing them with heavy colored lines. The back edges  $(e, f)$  and  $(f, h)$  are shown with thinner black lines.

We have explained how to find a spanning tree of a graph using depth-first search. However, our discussion so far has not brought out the recursive nature of depth-first search. To help make the recursive nature of the algorithm clear, we need a little terminology. We say that we

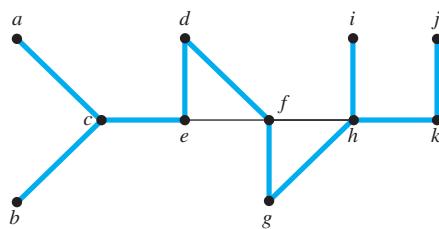


FIGURE 8 The Tree Edges and Back Edges of the Depth-First Search in Example 4.

*explore* from a vertex  $v$  when we carry out the steps of depth-first search beginning when  $v$  is added to the tree and ending when we have backtracked back to  $v$  for the last time. The key observation needed to understand the recursive nature of the algorithm is that when we add an edge connecting a vertex  $v$  to a vertex  $w$ , we finish exploring from  $w$  before we return to  $v$  to complete exploring from  $v$ .

In Algorithm 1 we construct the spanning tree of a graph  $G$  with vertices  $v_1, \dots, v_n$  by first selecting the vertex  $v_1$  to be the root. We initially set  $T$  to be the tree with just this one vertex. At each step we add a new vertex to the tree  $T$  together with an edge from a vertex already in  $T$  to this new vertex and we explore from this new vertex. Note that at the completion of the algorithm,  $T$  contains no simple circuits because no edge is ever added that connects two vertices in the tree. Moreover,  $T$  remains connected as it is built. (These last two observations can be easily proved via mathematical induction.) Because  $G$  is connected, every vertex in  $G$  is visited by the algorithm and is added to the tree (as the reader should verify). It follows that  $T$  is a spanning tree of  $G$ .

#### ALGORITHM 1 Depth-First Search.

```

procedure DFS(G : connected graph with vertices v_1, v_2, \dots, v_n)
 $T :=$ tree consisting only of the vertex v_1
 visit(v_1)

 procedure visit(v : vertex of G)
 for each vertex w adjacent to v and not yet in T
 add vertex w and edge $\{v, w\}$ to T
 visit(w)

```

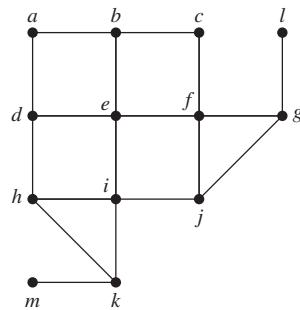
We now analyze the computational complexity of the depth-first search algorithm. The key observation is that for each vertex  $v$ , the procedure *visit*( $v$ ) is called when the vertex  $v$  is first encountered in the search and it is not called again. Assuming that the adjacency lists for  $G$  are available (see Section 10.3), no computations are required to find the vertices adjacent to  $v$ . As we follow the steps of the algorithm, we examine each edge at most twice to determine whether to add this edge and one of its endpoints to the tree. Consequently, the procedure *DFS* constructs a spanning tree using  $O(e)$ , or  $O(n^2)$ , steps where  $e$  and  $n$  are the number of edges and vertices in  $G$ , respectively. [Note that a step involves examining a vertex to see whether it is already in the spanning tree as it is being built and adding this vertex and the corresponding edge if the vertex is not already in the tree. We have also made use of the inequality  $e \leq n(n - 1)/2$ , which holds for any simple graph.]

Depth-first search can be used as the basis for algorithms that solve many different problems. For example, it can be used to find paths and circuits in a graph, it can be used to determine the connected components of a graph, and it can be used to find the cut vertices of a connected graph. As we will see, depth-first search is the basis of backtracking techniques used to search for solutions of computationally difficult problems. (See [GrYe05], [Ma89], and [CoLeRiSt09] for a discussion of algorithms based on depth-first search.)

## Breadth-First Search



We can also produce a spanning tree of a simple graph by the use of **breadth-first search**. Again, a rooted tree will be constructed, and the underlying undirected graph of this rooted tree forms the spanning tree. Arbitrarily choose a root from the vertices of the graph. Then add all

FIGURE 9 A Graph  $G$ .

edges incident to this vertex. The new vertices added at this stage become the vertices at level 1 in the spanning tree. Arbitrarily order them. Next, for each vertex at level 1, visited in order, add each edge incident to this vertex to the tree as long as it does not produce a simple circuit. Arbitrarily order the children of each vertex at level 1. This produces the vertices at level 2 in the tree. Follow the same procedure until all the vertices in the tree have been added. The procedure ends because there are only a finite number of edges in the graph. A spanning tree is produced because we have produced a tree containing every vertex of the graph. An example of breadth-first search is given in Example 5.

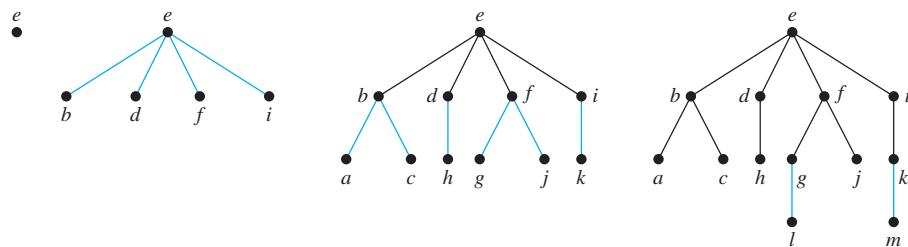
**EXAMPLE 5**

Use breadth-first search to find a spanning tree for the graph shown in Figure 9.



**Solution:** The steps of the breadth-first search procedure are shown in Figure 10. We choose the vertex  $e$  to be the root. Then we add edges incident with all vertices adjacent to  $e$ , so edges from  $e$  to  $b$ ,  $d$ ,  $f$ , and  $i$  are added. These four vertices are at level 1 in the tree. Next, add the edges from these vertices at level 1 to adjacent vertices not already in the tree. Hence, the edges from  $b$  to  $a$  and  $c$  are added, as are edges from  $d$  to  $h$ , from  $f$  to  $j$  and  $g$ , and from  $i$  to  $k$ . The new vertices  $a$ ,  $c$ ,  $h$ ,  $j$ ,  $g$ , and  $k$  are at level 2. Next, add edges from these vertices to adjacent vertices not already in the graph. This adds edges from  $g$  to  $l$  and from  $k$  to  $m$ .

We describe breadth-first search in pseudocode as Algorithm 2. In this algorithm, we assume the vertices of the connected graph  $G$  are ordered as  $v_1, v_2, \dots, v_n$ . In the algorithm we use the term “process” to describe the procedure of adding new vertices, and corresponding edges, to the tree adjacent to the current vertex being processed as long as a simple circuit is not produced.

FIGURE 10 Breadth-First Search of  $G$ .

**ALGORITHM 2 Breadth-First Search.**

```

procedure BFS (G: connected graph with vertices v_1, v_2, \dots, v_n)
 T := tree consisting only of vertex v_1
 L := empty list
 put v_1 in the list L of unprocessed vertices
 while L is not empty
 remove the first vertex, v, from L
 for each neighbor w of v
 if w is not in L and not in T then
 add w to the end of the list L
 add w and edge $\{v, w\}$ to T

```

We now analyze the computational complexity of breadth-first search. For each vertex  $v$  in the graph we examine all vertices adjacent to  $v$  and we add each vertex not yet visited to the tree  $T$ . Assuming we have the adjacency lists for the graph available, no computation is required to determine which vertices are adjacent to a given vertex. As in the analysis of the depth-first search algorithm, we see that we examine each edge at most twice to determine whether we should add this edge and its endpoint not already in the tree. It follows that the breadth-first search algorithm uses  $O(e)$  or  $O(n^2)$  steps.

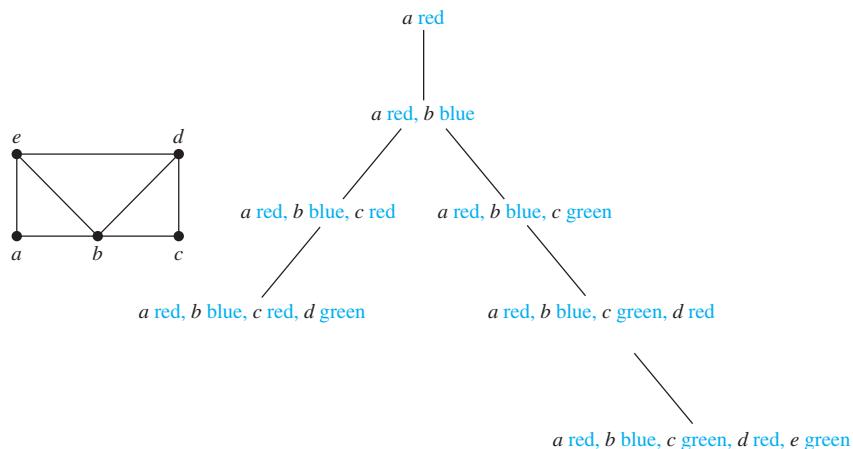
Breadth-first search is one of the most useful algorithms in graph theory. In particular, it can serve as the basis for algorithms that solve a wide variety of problems. For example, algorithms that find the connected components of a graph, that determine whether a graph is bipartite, and that find the path with the fewest edges between two vertices in a graph can all be built using breadth-first search.

## Backtracking Applications

There are problems that can be solved only by performing an exhaustive search of all possible solutions. One way to search systematically for a solution is to use a decision tree, where each internal vertex represents a decision and each leaf a possible solution. To find a solution via backtracking, first make a sequence of decisions in an attempt to reach a solution as long as this is possible. The sequence of decisions can be represented by a path in the decision tree. Once it is known that no solution can result from any further sequence of decisions, backtrack to the parent of the current vertex and work toward a solution with another series of decisions, if this is possible. The procedure continues until a solution is found, or it is established that no solution exists. Examples 6 to 8 illustrate the usefulness of backtracking.

**EXAMPLE 6 Graph Colorings** How can backtracking be used to decide whether a graph can be colored using  $n$  colors?

**Solution:** We can solve this problem using backtracking in the following way. First pick some vertex  $a$  and assign it color 1. Then pick a second vertex  $b$ , and if  $b$  is not adjacent to  $a$ , assign it color 1. Otherwise, assign color 2 to  $b$ . Then go on to a third vertex  $c$ . Use color 1, if possible, for  $c$ . Otherwise use color 2, if this is possible. Only if neither color 1 nor color 2 can be used should color 3 be used. Continue this process as long as it is possible to assign one of the  $n$  colors to each additional vertex, always using the first allowable color in the list. If a vertex is reached that cannot be colored by any of the  $n$  colors, backtrack to the last assignment made and change the coloring of the last vertex colored, if possible, using the next allowable color in the list. If it is not possible to change this coloring, backtrack farther to previous assignments, one step back at a time, until it is possible to change a coloring of a vertex. Then continue assigning



**FIGURE 11** Coloring a Graph Using Backtracking.

colors of additional vertices as long as possible. If a coloring using  $n$  colors exists, backtracking will produce it. (Unfortunately this procedure can be extremely inefficient.)

In particular, consider the problem of coloring the graph shown in Figure 11 with three colors. The tree shown in Figure 11 illustrates how backtracking can be used to construct a 3-coloring. In this procedure, red is used first, then blue, and finally green. This simple example can obviously be done without backtracking, but it is a good illustration of the technique.

In this tree, the initial path from the root, which represents the assignment of red to  $a$ , leads to a coloring with  $a$  red,  $b$  blue,  $c$  red, and  $d$  green. It is impossible to color  $e$  using any of the three colors when  $a$ ,  $b$ ,  $c$ , and  $d$  are colored in this way. So, backtrack to the parent of the vertex representing this coloring. Because no other color can be used for  $d$ , backtrack one more level. Then change the color of  $c$  to green. We obtain a coloring of the graph by then assigning red to  $d$  and green to  $e$ .  $\blacktriangleleft$

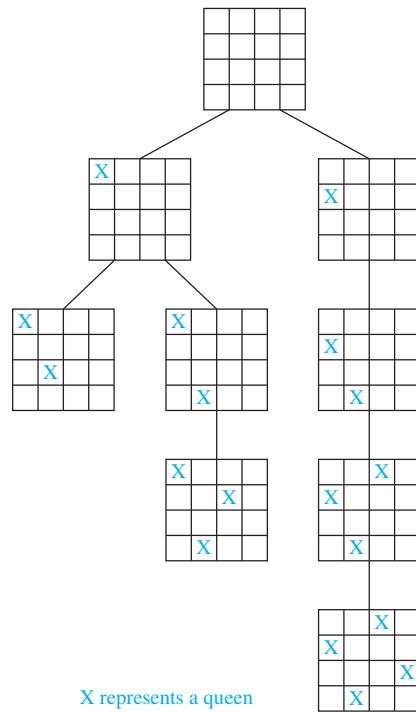
### EXAMPLE 7



**The  $n$ -Queens Problem** The  $n$ -queens problem asks how  $n$  queens can be placed on an  $n \times n$  chessboard so that no two queens can attack one another. How can backtracking be used to solve the  $n$ -queens problem?

**Solution:** To solve this problem we must find  $n$  positions on an  $n \times n$  chessboard so that no two of these positions are in the same row, same column, or in the same diagonal [a diagonal consists of all positions  $(i, j)$  with  $i + j = m$  for some  $m$ , or  $i - j = m$  for some  $m$ ]. We will use backtracking to solve the  $n$ -queens problem. We start with an empty chessboard. At stage  $k + 1$  we attempt putting an additional queen on the board in the  $(k + 1)$ st column, where there are already queens in the first  $k$  columns. We examine squares in the  $(k + 1)$ st column starting with the square in the first row, looking for a position to place this queen so that it is not in the same row or on the same diagonal as a queen already on the board. (We already know it is not in the same column.) If it is impossible to find a position to place the queen in the  $(k + 1)$ st column, backtrack to the placement of the queen in the  $k$ th column, and place this queen in the next allowable row in this column, if such a row exists. If no such row exists, backtrack further.

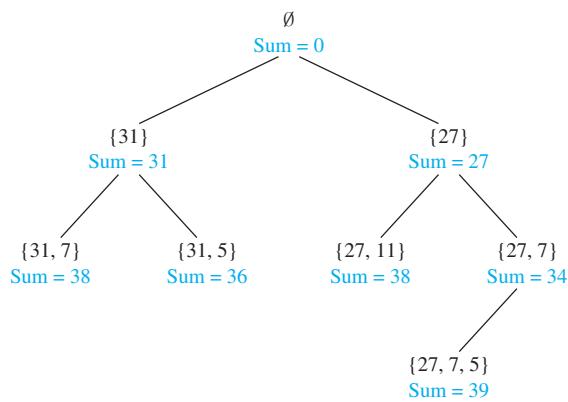
In particular, Figure 12 displays a backtracking solution to the four-queens problem. In this solution, we place a queen in the first row and column. Then we put a queen in the third row of the second column. However, this makes it impossible to place a queen in the third column. So we backtrack and put a queen in the fourth row of the second column. When we do this, we can place a queen in the second row of the third column. But there is no way to add a queen to the fourth column. This shows that no solution results when a queen is placed in the first row and column. We backtrack to the empty chessboard, and place a queen in the second row of the first column. This leads to a solution as shown in Figure 12.  $\blacktriangleleft$

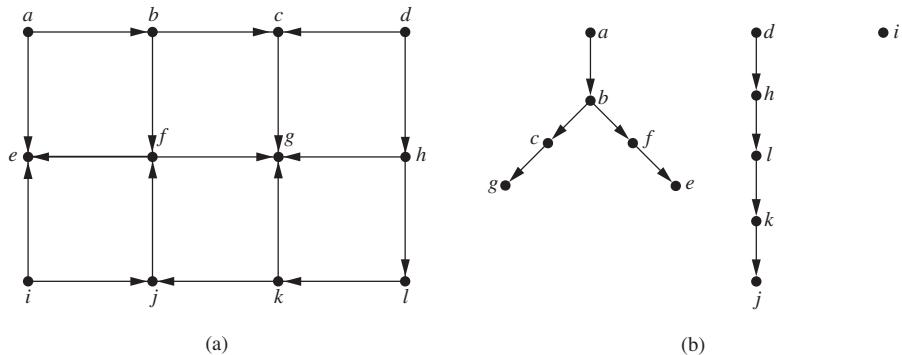
**FIGURE 12** A Backtracking Solution of the Four-Queens Problem.

**EXAMPLE 8 Sums of Subsets** Consider this problem. Given a set of positive integers  $x_1, x_2, \dots, x_n$ , find a subset of this set of integers that has  $M$  as its sum. How can backtracking be used to solve this problem?

**Solution:** We start with a sum with no terms. We build up the sum by successively adding terms. An integer in the sequence is included if the sum remains less than  $M$  when this integer is added to the sum. If a sum is reached such that the addition of any term is greater than  $M$ , backtrack by dropping the last term of the sum.

Figure 13 displays a backtracking solution to the problem of finding a subset of  $\{31, 27, 15, 11, 7, 5\}$  with the sum equal to 39. ◀

**FIGURE 13** Find a Sum Equal to 39 Using Backtracking.



## FIGURE 14 Depth-First Search of a Directed Graph.

## Depth-First Search in Directed Graphs

We can easily modify both depth-first search and breadth-first search so that they can run given a directed graph as input. However, the output will not necessarily be a spanning tree, but rather a spanning forest. In both algorithms we can add an edge only when it is directed away from the vertex that is being visited and to a vertex not yet added. If at a stage of either algorithm we find that no edge exists starting at a vertex already added to one not yet added, the next vertex added by the algorithm becomes the root of a new tree in the spanning forest. This is illustrated in Example 9.

**EXAMPLE 9** What is the output of depth-first search given the graph  $G$  shown in Figure 14(a) as input?

**Solution:** We begin the depth-first search at vertex  $a$  and add vertices  $b$ ,  $c$ , and  $g$  and the corresponding edges where we are blocked. We backtrack to  $c$  but we are still blocked, and then backtrack to  $b$ , where we add vertices  $f$  and  $e$  and the corresponding edges. Backtracking takes us all the way back to  $a$ . We then start a new tree at  $d$  and add vertices  $h$ ,  $l$ ,  $k$ , and  $j$  and the corresponding edges. We backtrack to  $k$ , then  $l$ , then  $h$ , and back to  $d$ . Finally, we start a new tree at  $i$ , completing the depth-first search. The output is shown in Figure 14(b). 

Depth-first search in directed graphs is the basis of many algorithms (see [GrYe05], [Ma89], and [CoLeRiSt09]). It can be used to determine whether a directed graph has a circuit, it can be used to carry out a topological sort of a graph, and it can also be used to find the strongly connected components of a directed graph.

We conclude this section with an application of depth-first search and breadth-first search to search engines on the Web.

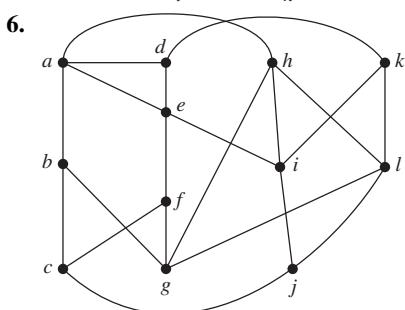
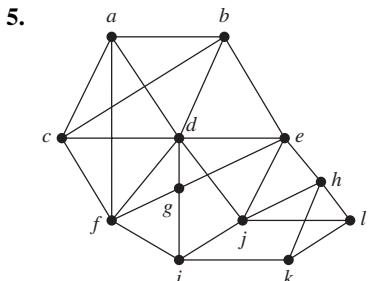
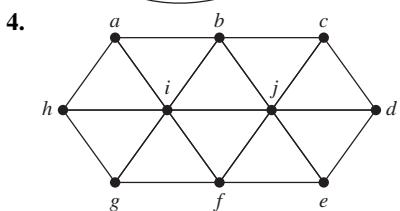
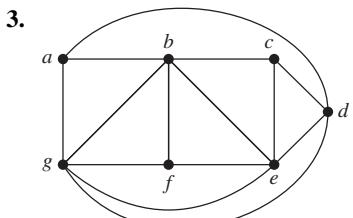
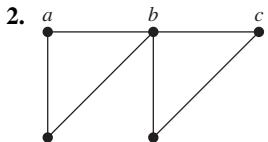
**EXAMPLE 10** **Web Spiders** To index websites, search engines such as Google and Yahoo systematically explore the Web starting at known sites. These search engines use programs called Web spiders (or crawlers or bots) to visit websites and analyze their contents. Web spiders use both depth-first searching and breadth-first searching to create indices. As described in Example 5 in Section 10.1, Web pages and links between them can be modeled by a directed graph called the Web graph. Web pages are represented by vertices and links are represented by directed edges. Using depth-first search, an initial Web page is selected, a link is followed to a second Web page (if there is such a link), a link on the second Web page is followed to a third Web page, if there is such a link, and so on, until a page with no new links is found. Backtracking is then used to examine

links at the previous level to look for new links, and so on. (Because of practical limitations, Web spiders have limits to the depth they search in depth-first search.) Using breadth-first search, an initial Web page is selected and a link on this page is followed to a second Web page, then a second link on the initial page is followed (if it exists), and so on, until all links of the initial page have been followed. Then links on the pages one level down are followed, page by page, and so on.

## Exercises

1. How many edges must be removed from a connected graph with  $n$  vertices and  $m$  edges to produce a spanning tree?

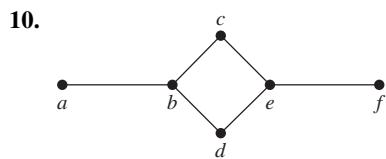
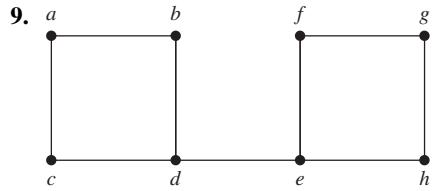
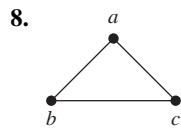
In Exercises 2–6 find a spanning tree for the graph shown by removing edges in simple circuits.



7. Find a spanning tree for each of these graphs.

- a)  $K_5$       b)  $K_{4,4}$       c)  $K_{1,6}$   
d)  $Q_3$       e)  $C_5$       f)  $W_5$

In Exercises 8–10 draw all the spanning trees of the given simple graphs.



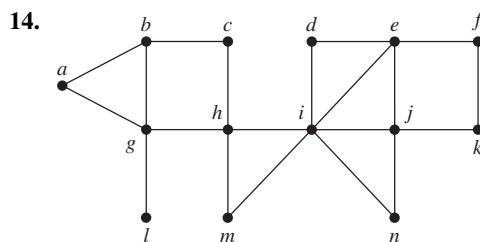
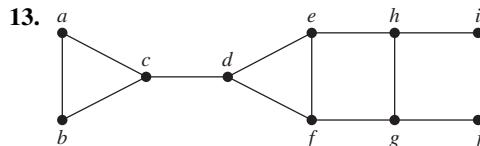
- \*11. How many different spanning trees does each of these simple graphs have?

- a)  $K_3$       b)  $K_4$       c)  $K_{2,2}$       d)  $C_5$

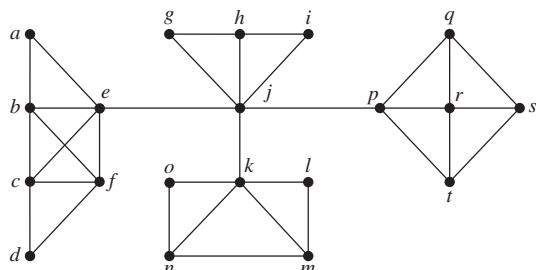
- \*12. How many nonisomorphic spanning trees does each of these simple graphs have?

- a)  $K_3$       b)  $K_4$       c)  $K_5$

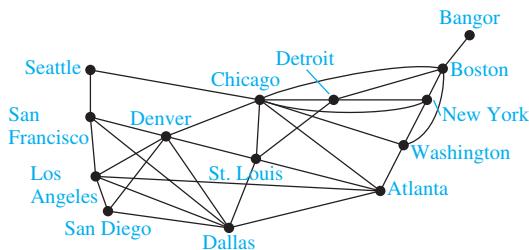
In Exercises 13–15 use depth-first search to produce a spanning tree for the given simple graph. Choose  $a$  as the root of this spanning tree and assume that the vertices are ordered alphabetically.



15.



16. Use breadth-first search to produce a spanning tree for each of the simple graphs in Exercises 13–15. Choose  $a$  as the root of each spanning tree.
17. Use depth-first search to find a spanning tree of each of these graphs.
- $W_6$  (see Example 7 of Section 10.2), starting at the vertex of degree 6
  - $K_5$
  - $K_{3,4}$ , starting at a vertex of degree 3
  - $Q_3$
18. Use breadth-first search to find a spanning tree of each of the graphs in Exercise 17.
19. Describe the trees produced by breadth-first search and depth-first search of the wheel graph  $W_n$ , starting at the vertex of degree  $n$ , where  $n$  is an integer with  $n \geq 3$ . (See Example 7 of Section 10.2.) Justify your answers.
20. Describe the trees produced by breadth-first search and depth-first search of the complete graph  $K_n$ , where  $n$  is a positive integer. Justify your answers.
21. Describe the trees produced by breadth-first search and depth-first search of the complete bipartite graph  $K_{m,n}$ , starting at a vertex of degree  $m$ , where  $m$  and  $n$  are positive integers. Justify your answers.
22. Describe the tree produced by breadth-first search and depth-first search for the  $n$ -cube graph  $Q_n$ , where  $n$  is a positive integer.
23. Suppose that an airline must reduce its flight schedule to save money. If its original routes are as illustrated here, which flights can be discontinued to retain service between all pairs of cities (where it may be necessary to combine flights to fly from one city to another)?



24. Explain how breadth-first search or depth-first search can be used to order the vertices of a connected graph.
- \*25. Show that the length of the shortest path between vertices  $v$  and  $u$  in a connected simple graph equals the level number of  $u$  in the breadth-first spanning tree of  $G$  with root  $v$ .

26. Use backtracking to try to find a coloring of each of the graphs in Exercises 7–9 of Section 10.8 using three colors.

27. Use backtracking to solve the  $n$ -queens problem for these values of  $n$ .

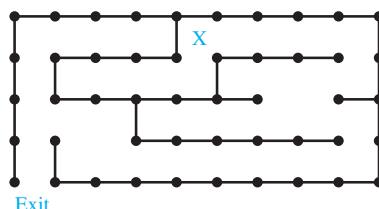
- $n = 3$
- $n = 5$
- $n = 6$

28. Use backtracking to find a subset, if it exists, of the set  $\{27, 24, 19, 14, 11, 8\}$  with sum

- 20.
- 41.
- 60.

29. Explain how backtracking can be used to find a Hamilton path or circuit in a graph.

30. a) Explain how backtracking can be used to find the way out of a maze, given a starting position and the exit position. Consider the maze divided into positions, where at each position the set of available moves includes one to four possibilities (up, down, right, left).  
b) Find a path from the starting position marked by X to the exit in this maze.



A **spanning forest** of a graph  $G$  is a forest that contains every vertex of  $G$  such that two vertices are in the same tree of the forest when there is a path in  $G$  between these two vertices.

- Show that every finite simple graph has a spanning forest.
- How many trees are in the spanning forest of a graph?
- How many edges must be removed to produce the spanning forest of a graph with  $n$  vertices,  $m$  edges, and  $c$  connected components?
- Let  $G$  be a connected graph. Show that if  $T$  is a spanning tree of  $G$  constructed using breadth-first search, then an edge of  $G$  not in  $T$  must connect vertices at the same level or at levels that differ by 1 in this spanning tree.
- Explain how to use breadth-first search to find the length of a shortest path between two vertices in an undirected graph.
- Devise an algorithm based on breadth-first search that determines whether a graph has a simple circuit, and if so, finds one.
- Devise an algorithm based on breadth-first search for finding the connected components of a graph.
- Explain how breadth-first search and how depth-first search can be used to determine whether a graph is bipartite.
- Which connected simple graphs have exactly one spanning tree?
- Devise an algorithm for constructing the spanning forest of a graph based on deleting edges that form simple circuits.

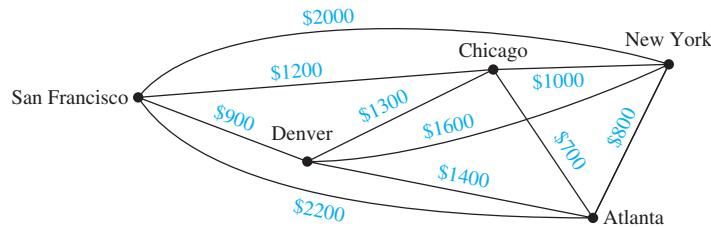
41. Devise an algorithm for constructing the spanning forest of a graph based on depth-first searching.
42. Devise an algorithm for constructing the spanning forest of a graph based on breadth-first searching.
43. Let  $G$  be a connected graph. Show that if  $T$  is a spanning tree of  $G$  constructed using depth-first search, then an edge of  $G$  not in  $T$  must be a back edge, that is, it must connect a vertex to one of its ancestors or one of its descendants in  $T$ .
44. When must an edge of a connected simple graph be in every spanning tree for this graph?
45. For which graphs do depth-first search and breadth-first search produce identical spanning trees no matter which vertex is selected as the root of the tree? Justify your answer.
46. Use Exercise 43 to prove that if  $G$  is a connected, simple graph with  $n$  vertices and  $G$  does not contain a simple path of length  $k$  then it contains at most  $(k - 1)n$  edges.
47. Use mathematical induction to prove that breadth-first search visits vertices in order of their level in the resulting spanning tree.
48. Use pseudocode to describe a variation of depth-first search that assigns the integer  $n$  to the  $n$ th vertex visited in the search. Show that this numbering corresponds to the numbering of the vertices created by a preorder traversal of the spanning tree.
49. Use pseudocode to describe a variation of breadth-first search that assigns the integer  $m$  to the  $m$ th vertex visited in the search.
- \*50. Suppose that  $G$  is a directed graph and  $T$  is a spanning tree constructed using breadth-first search. Show that every edge of  $G$  has endpoints that are at the same level or one level higher or lower.
51. Show that if  $G$  is a directed graph and  $T$  is a spanning tree constructed using depth-first search, then every edge not in the spanning tree is a **forward edge** connecting an ancestor to a descendant, a **back edge** connecting a descendant to an ancestor, or a **cross edge** connecting a vertex to a vertex in a previously visited subtree.
- \*52. Describe a variation of depth-first search that assigns the smallest available positive integer to a vertex when the algorithm is totally finished with this vertex. Show that in this numbering, each vertex has a larger number than its children and that the children have increasing numbers from left to right.
- Let  $T_1$  and  $T_2$  be spanning trees of a graph. The **distance** between  $T_1$  and  $T_2$  is the number of edges in  $T_1$  and  $T_2$  that are not common to  $T_1$  and  $T_2$ .
53. Find the distance between each pair of spanning trees shown in Figures 3(c) and 4 of the graph  $G$  shown in Figure 2.
- \*54. Suppose that  $T_1$ ,  $T_2$ , and  $T_3$  are spanning trees of the simple graph  $G$ . Show that the distance between  $T_1$  and  $T_3$  does not exceed the sum of the distance between  $T_1$  and  $T_2$  and the distance between  $T_2$  and  $T_3$ .
- \*\*55. Suppose that  $T_1$  and  $T_2$  are spanning trees of a simple graph  $G$ . Moreover, suppose that  $e_1$  is an edge in  $T_1$  that is not in  $T_2$ . Show that there is an edge  $e_2$  in  $T_2$  that is not in  $T_1$  such that  $T_1$  remains a spanning tree if  $e_1$  is removed from it and  $e_2$  is added to it, and  $T_2$  remains a spanning tree if  $e_2$  is removed from it and  $e_1$  is added to it.
- \*56. Show that it is possible to find a sequence of spanning trees leading from any spanning tree to any other by successively removing one edge and adding another.
- A **rooted spanning tree** of a directed graph is a rooted tree containing edges of the graph such that every vertex of the graph is an endpoint of one of the edges in the tree.
57. For each of the directed graphs in Exercises 18–23 of Section 10.5 either find a rooted spanning tree of the graph or determine that no such tree exists.
- \*58. Show that a connected directed graph in which each vertex has the same in-degree and out-degree has a rooted spanning tree. [Hint: Use an Euler circuit.]
- \*59. Give an algorithm to build a rooted spanning tree for connected directed graphs in which each vertex has the same in-degree and out-degree.
- \*60. Show that if  $G$  is a directed graph and  $T$  is a spanning tree constructed using depth-first search, then  $G$  contains a circuit if and only if  $G$  contains a back edge (see Exercise 51) relative to the spanning tree  $T$ .
- \*61. Use Exercise 60 to construct an algorithm for determining whether a directed graph contains a circuit.

## 11.5 Minimum Spanning Trees

### Introduction



A company plans to build a communications network connecting its five computer centers. Any pair of these centers can be linked with a leased telephone line. Which links should be made to ensure that there is a path between any two computer centers so that the total cost of the network is minimized? We can model this problem using the weighted graph shown in Figure 1, where vertices represent computer centers, edges represent possible leased lines, and the weights on edges are the monthly lease rates of the lines represented by the edges. We can solve this problem



**FIGURE 1** A Weighted Graph Showing Monthly Lease Costs for Lines in a Computer Network.

by finding a spanning tree so that the sum of the weights of the edges of the tree is minimized. Such a spanning tree is called a **minimum spanning tree**.

## Algorithms for Minimum Spanning Trees

A wide variety of problems are solved by finding a spanning tree in a weighted graph such that the sum of the weights of the edges in the tree is a minimum.

### DEFINITION 1

A *minimum spanning tree* in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.



We will present two algorithms for constructing minimum spanning trees. Both proceed by successively adding edges of smallest weight from those edges with a specified property that have not already been used. Both are greedy algorithms. Recall from Section 3.1 that a greedy algorithm is a procedure that makes an optimal choice at each of its steps. Optimizing at each step does not guarantee that the optimal overall solution is produced. However, the two algorithms presented in this section for constructing minimum spanning trees are greedy algorithms that do produce optimal solutions.

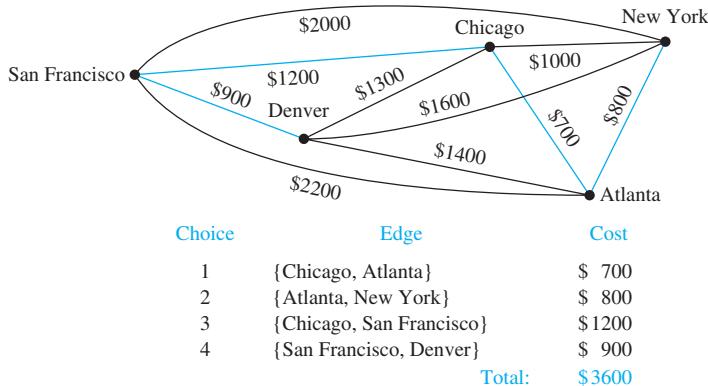


The first algorithm that we will discuss was originally discovered by the Czech mathematician Vojtěch Jarník in 1930, who described it in a paper in an obscure Czech journal. The algorithm became well known when it was rediscovered in 1957 by Robert Prim. Because of this, it is known as **Prim's algorithm** (and sometimes as the **Prim-Jarník algorithm**). Begin by choosing any edge with smallest weight, putting it into the spanning tree. Successively add to the tree edges of minimum weight that are incident to a vertex already in the tree, never forming a simple circuit with those edges already in the tree. Stop when  $n - 1$  edges have been added.

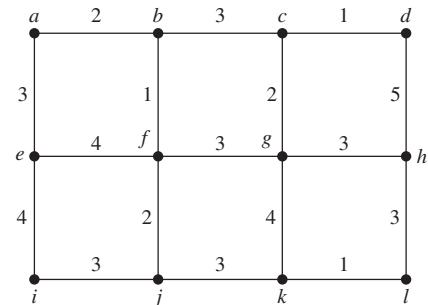
Later in this section, we will prove that this algorithm produces a minimum spanning tree for any connected weighted graph. Algorithm 1 gives a pseudocode description of Prim's algorithm.



**ROBERT CLAY PRIM (BORN 1921)** Robert Prim, born in Sweetwater, Texas, received his B.S. in electrical engineering in 1941 and his Ph.D. in mathematics from Princeton University in 1949. He was an engineer at the General Electric Company from 1941 until 1944, an engineer and mathematician at the United States Naval Ordnance Lab from 1944 until 1949, and a research associate at Princeton University from 1948 until 1949. Among the other positions he has held are director of mathematics and mechanics research at Bell Telephone Laboratories from 1958 until 1961 and vice president of research at Sandia Corporation. He is currently retired.



**FIGURE 2** A Minimum Spanning Tree for the Weighted Graph in Figure 1.



**FIGURE 3** A Weighted Graph.

#### ALGORITHM 1 Prim's Algorithm.

```

procedure Prim(G: weighted connected undirected graph with n vertices)
 T := a minimum-weight edge
 for $i := 1$ to $n - 2$
 e := an edge of minimum weight incident to a vertex in T and not forming a
 simple circuit in T if added to T
 T := T with e added
 return T {T is a minimum spanning tree of G}

```

Note that the choice of an edge to add at a stage of the algorithm is not determined when there is more than one edge with the same weight that satisfies the appropriate criteria. We need to order the edges to make the choices deterministic. We will not worry about this in the remainder of the section. Also note that there may be more than one minimum spanning tree for a given connected weighted simple graph. (See Exercise 9.) Examples 1 and 2 illustrate how Prim's algorithm is used.

**EXAMPLE 1** Use Prim's algorithm to design a minimum-cost communications network connecting all the computers represented by the graph in Figure 1.

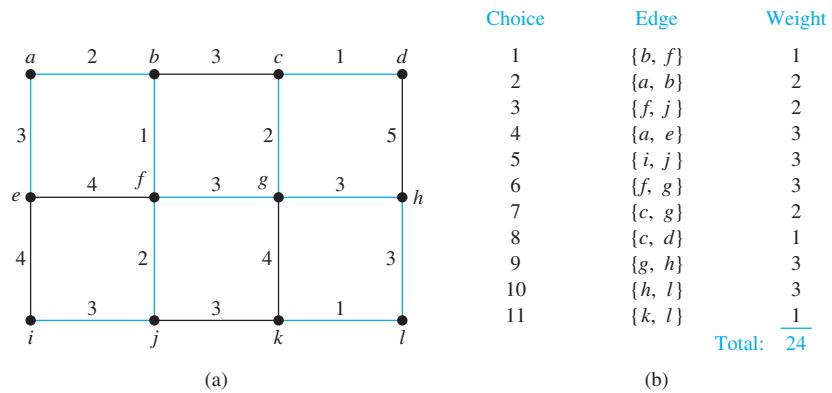
**Solution:** We solve this problem by finding a minimum spanning tree in the graph in Figure 1. Prim's algorithm is carried out by choosing an initial edge of minimum weight and successively adding edges of minimum weight that are incident to a vertex in the tree and that do not form simple circuits. The edges in color in Figure 2 show a minimum spanning tree produced by Prim's algorithm, with the choice made at each step displayed. 

**EXAMPLE 2** Use Prim's algorithm to find a minimum spanning tree in the graph shown in Figure 3.

**Solution:** A minimum spanning tree constructed using Prim's algorithm is shown in Figure 4. The successive edges chosen are displayed. 



The second algorithm we will discuss was discovered by Joseph Kruskal in 1956, although the basic ideas it uses were described much earlier. To carry out **Kruskal's algorithm**, choose an edge in the graph with minimum weight.



**FIGURE 4** A Minimum Spanning Tree Produced Using Prim's Algorithm.

Successively add edges with minimum weight that do not form a simple circuit with those edges already chosen. Stop after  $n - 1$  edges have been selected.

The proof that Kruskal's algorithm produces a minimum spanning tree for every connected weighted graph is left as an exercise. Pseudocode for Kruskal's algorithm is given in Algorithm 2.

#### **ALGORITHM 2 Kruskal's Algorithm.**

```

procedure Kruskal(G : weighted connected undirected graph with n vertices)
 $T :=$ empty graph
 for $i := 1$ to $n - 1$
 $e :=$ any edge in G with smallest weight that does not form a simple circuit
 when added to T
 $T := T$ with e added
 return T { T is a minimum spanning tree of G }

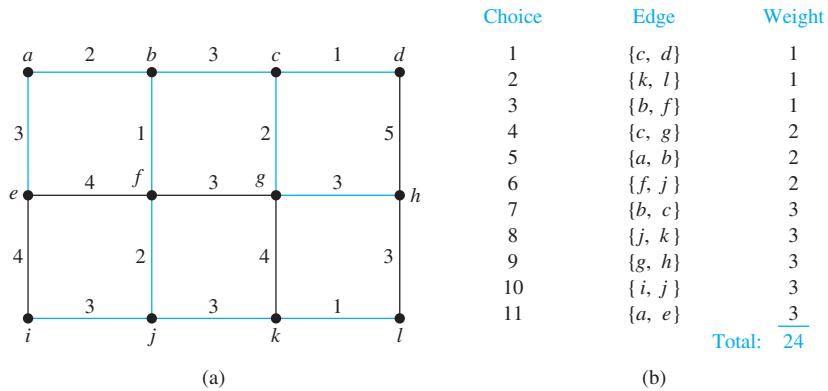
```



**JOSEPH BERNARD KRUSKAL (1928–2010)** Joseph Kruskal was born in New York City, where his father was a fur dealer and his mother promoted the art of origami on early television. Kruskal attended the University of Chicago and received his Ph.D. from Princeton University in 1954. He was an instructor in mathematics at Princeton and at the University of Wisconsin, and later he was an assistant professor at the University of Michigan. In 1959 he became a member of the technical staff at Bell Laboratories, where he worked until his retirement in the late 1990s. Kruskal discovered his algorithm for producing minimum spanning trees when he was a second-year graduate student. He was not sure his  $2\frac{1}{2}$ -page paper on this subject was worthy of publication, but was convinced by others to submit it. His research interests included statistical linguistics and psychometrics. Besides his work on minimum spanning trees, Kruskal is also known for contributions to multidimensional scaling. It is noteworthy that Joseph Kruskal's two brothers, Martin and William, also were well known mathematicians.



**HISTORICAL NOTE** Joseph Kruskal and Robert Prim developed their algorithms for constructing minimum spanning trees in the mid-1950s. However, they were not the first people to discover such algorithms. For example, the work of the anthropologist Jan Czekanowski, in 1909, contains many of the ideas required to find minimum spanning trees. In 1926, Otakar Boruvka described methods for constructing minimum spanning trees in work relating to the construction of electric power networks, and as mentioned in the text what is now called Prim's algorithm was discovered by Vojtěch Jarník in 1930.

**FIGURE 5 A Minimum Spanning Tree Produced by Kruskal's Algorithm.**

The reader should note the difference between Prim's and Kruskal's algorithms. In Prim's algorithm edges of minimum weight that are incident to a vertex already in the tree, and not forming a circuit, are chosen; whereas in Kruskal's algorithm edges of minimum weight that are not necessarily incident to a vertex already in the tree, and that do not form a circuit, are chosen. Note that as in Prim's algorithm, if the edges are not ordered, there may be more than one choice for the edge to add at a stage of this procedure. Consequently, the edges need to be ordered for the procedure to be deterministic. Example 3 illustrates how Kruskal's algorithm is used.

**EXAMPLE 3** Use Kruskal's algorithm to find a minimum spanning tree in the weighted graph shown in Figure 3.



**Solution:** A minimum spanning tree and the choices of edges at each stage of Kruskal's algorithm are shown in Figure 5.

We will now prove that Prim's algorithm produces a minimum spanning tree of a connected weighted graph.



**Proof:** Let  $G$  be a connected weighted graph. Suppose that the successive edges chosen by Prim's algorithm are  $e_1, e_2, \dots, e_{n-1}$ . Let  $S$  be the tree with  $e_1, e_2, \dots, e_{n-1}$  as its edges, and let  $S_k$  be the tree with  $e_1, e_2, \dots, e_k$  as its edges. Let  $T$  be a minimum spanning tree of  $G$  containing the edges  $e_1, e_2, \dots, e_k$ , where  $k$  is the maximum integer with the property that a minimum spanning tree exists containing the first  $k$  edges chosen by Prim's algorithm. The theorem follows if we can show that  $S = T$ .

Suppose that  $S \neq T$ , so that  $k < n - 1$ . Consequently,  $T$  contains  $e_1, e_2, \dots, e_k$ , but not  $e_{k+1}$ . Consider the graph made up of  $T$  together with  $e_{k+1}$ . Because this graph is connected and has  $n$  edges, too many edges to be a tree, it must contain a simple circuit. This simple circuit must contain  $e_{k+1}$  because there was no simple circuit in  $T$ . Furthermore, there must be an edge in the simple circuit that does not belong to  $S_{k+1}$  because  $S_{k+1}$  is a tree. By starting at an endpoint of  $e_{k+1}$  that is also an endpoint of one of the edges  $e_1, \dots, e_k$ , and following the circuit until it reaches an edge not in  $S_{k+1}$ , we can find an edge  $e$  not in  $S_{k+1}$  that has an endpoint that is also an endpoint of one of the edges  $e_1, e_2, \dots, e_k$ .

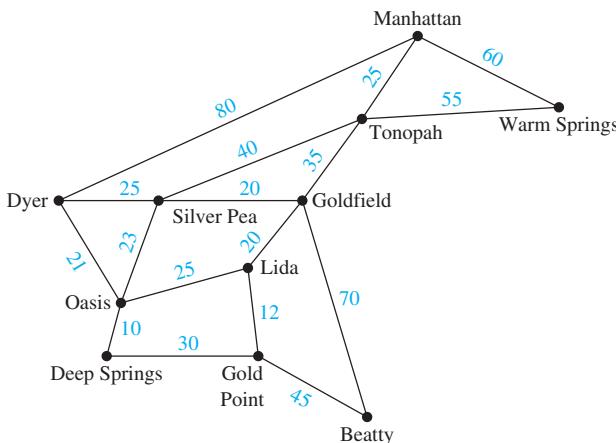
By deleting  $e$  from  $T$  and adding  $e_{k+1}$ , we obtain a tree  $T'$  with  $n - 1$  edges (it is a tree because it has no simple circuits). Note that the tree  $T'$  contains  $e_1, e_2, \dots, e_k, e_{k+1}$ . Furthermore, because  $e_{k+1}$  was chosen by Prim's algorithm at the  $(k + 1)$ st step, and  $e$  was also available at that step, the weight of  $e_{k+1}$  is less than or equal to the weight of  $e$ . From this observation, it follows that  $T'$  is also a minimum spanning tree, because the sum of the weights of its edges

does not exceed the sum of the weights of the edges of  $T$ . This contradicts the choice of  $k$  as the maximum integer such that a minimum spanning tree exists containing  $e_1, \dots, e_k$ . Hence,  $k = n - 1$ , and  $S = T$ . It follows that Prim's algorithm produces a minimum spanning tree.  $\triangleleft$

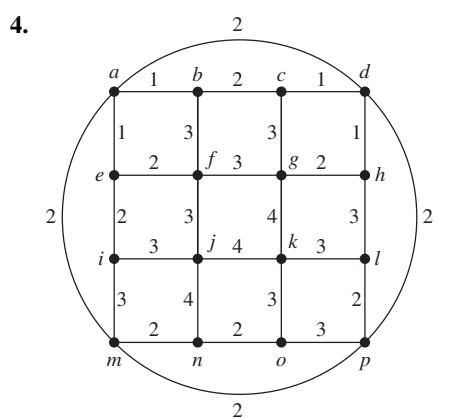
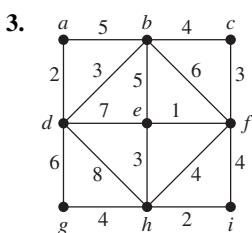
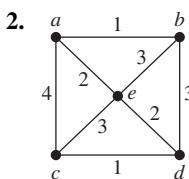
It can be shown (see [CoLeRiSt09]) that to find a minimum spanning tree of a graph with  $m$  edges and  $n$  vertices, Kruskal's algorithm can be carried out using  $O(m \log m)$  operations and Prim's algorithm can be carried out using  $O(m \log n)$  operations. Consequently, it is preferable to use Kruskal's algorithm for graphs that are **sparse**, that is, where  $m$  is very small compared to  $C(n, 2) = n(n - 1)/2$ , the total number of possible edges in an undirected graph with  $n$  vertices. Otherwise, there is little difference in the complexity of these two algorithms.

## Exercises

1. The roads represented by this graph are all unpaved. The lengths of the roads between pairs of towns are represented by edge weights. Which roads should be paved so that there is a path of paved roads between each pair of towns so that a minimum road length is paved? (Note: These towns are in Nevada.)



In Exercises 2–4 use Prim's algorithm to find a minimum spanning tree for the given weighted graph.



5. Use Kruskal's algorithm to design the communications network described at the beginning of the section.
6. Use Kruskal's algorithm to find a minimum spanning tree for the weighted graph in Exercise 2.
7. Use Kruskal's algorithm to find a minimum spanning tree for the weighted graph in Exercise 3.
8. Use Kruskal's algorithm to find a minimum spanning tree for the weighted graph in Exercise 4.
9. Find a connected weighted simple graph with the fewest edges possible that has more than one minimum spanning tree.
10. A **minimum spanning forest** in a weighted graph is a spanning forest with minimal weight. Explain how Prim's and Kruskal's algorithms can be adapted to construct minimum spanning forests.
- A **maximum spanning tree** of a connected weighted undirected graph is a spanning tree with the largest possible weight.
11. Devise an algorithm similar to Prim's algorithm for constructing a maximum spanning tree of a connected weighted graph.
12. Devise an algorithm similar to Kruskal's algorithm for constructing a maximum spanning tree of a connected weighted graph.
13. Find a maximum spanning tree for the weighted graph in Exercise 2.
14. Find a maximum spanning tree for the weighted graph in Exercise 3.

- 15.** Find a maximum spanning tree for the weighted graph in Exercise 4.
- 16.** Find the second least expensive communications network connecting the five computer centers in the problem posed at the beginning of the section.
- \*17.** Devise an algorithm for finding the second shortest spanning tree in a connected weighted graph.
- \*18.** Show that an edge with smallest weight in a connected weighted graph must be part of any minimum spanning tree.
- 19.** Show that there is a unique minimum spanning tree in a connected weighted graph if the weights of the edges are all different.
- 20.** Suppose that the computer network connecting the cities in Figure 1 must contain a direct link between New York and Denver. What other links should be included so that there is a link between every two computer centers and the cost is minimized?
- 21.** Find a spanning tree with minimal total weight containing the edges  $\{e, i\}$  and  $\{g, k\}$  in the weighted graph in Figure 3.
- 22.** Describe an algorithm for finding a spanning tree with minimal weight containing a specified set of edges in a connected weighted undirected simple graph.
- 23.** Express the algorithm devised in Exercise 22 in pseudocode.
- Sollin's algorithm** produces a minimum spanning tree from a connected weighted simple graph  $G = (V, E)$  by successively adding groups of edges. Suppose that the vertices in  $V$  are ordered. This produces an ordering of the edges where  $\{u_0, v_0\}$  precedes  $\{u_1, v_1\}$  if  $u_0$  precedes  $u_1$  or if  $u_0 = u_1$  and  $v_0$  precedes  $v_1$ . The algorithm begins by simultaneously choosing the edge of least weight incident to each vertex. The first edge in the ordering is taken in the case of ties. This produces a graph with no simple circuits, that is, a forest of trees (Exercise 24 asks for a proof of this fact). Next, simultaneously choose for each tree in the forest the shortest edge between a vertex in this tree and a vertex in a different tree. Again the first edge in the ordering is chosen in the case of ties. (This produces a graph with no simple circuits containing fewer trees than were present before this step; see Exercise 24.) Continue the process of simultaneously adding edges connecting trees until  $n - 1$  edges have been chosen. At this stage a minimum spanning tree has been constructed.
- \*24.** Show that the addition of edges at each stage of Sollin's algorithm produces a forest.
- 25.** Use Sollin's algorithm to produce a minimum spanning tree for the weighted graph shown in
- Figure 1.
  - Figure 3.
- \*26.** Express Sollin's algorithm in pseudocode.
- \*\*27.** Prove that Sollin's algorithm produces a minimum spanning tree in a connected undirected weighted graph.
- \*28.** Show that the first step of Sollin's algorithm produces a forest containing at least  $\lceil n/2 \rceil$  edges when the input is an undirected graph with  $n$  vertices.
- \*29.** Show that if there are  $r$  trees in the forest at some intermediate step of Sollin's algorithm, then at least  $\lceil r/2 \rceil$  edges are added by the next iteration of the algorithm.
- \*30.** Show that when given as input an undirected graph with  $n$  vertices, no more than  $\lfloor n/2^k \rfloor$  trees remain after the first step of Sollin's algorithm has been carried out and the second step of the algorithm has been carried out  $k - 1$  times.
- \*31.** Show that Sollin's algorithm requires at most  $\log n$  iterations to produce a minimum spanning tree from a connected undirected graph with  $n$  vertices.
- 32.** Prove that Kruskal's algorithm produces minimum spanning trees.
- 33.** Show that if  $G$  is a weighted graph with distinct edge weights, then for every simple circuit of  $G$ , the edge of maximum weight in this circuit does not belong to any minimum spanning tree of  $G$ .
- When Kruskal invented the algorithm that finds minimum spanning trees by adding edges in order of increasing weight as long as they do not form a simple circuit, he also invented another algorithm sometimes called the **reverse-delete algorithm**. This algorithm proceeds by successively deleting edges of maximum weight from a connected graph as long as doing so does not disconnect the graph.
- 34.** Express the reverse-delete algorithm in pseudocode.
- 35.** Prove that the reverse-delete algorithm always produces a minimum spanning tree when given as input a weighted graph with distinct edge weights. [Hint: Use Exercise 33.]

## Key Terms and Results

---

### TERMS

- tree:** a connected undirected graph with no simple circuits
- forest:** an undirected graph with no simple circuits
- rooted tree:** a directed graph with a specified vertex, called the root, such that there is a unique path to every other vertex from this root
- subtree:** a subgraph of a tree that is also a tree

**parent of  $v$  in a rooted tree:** the vertex  $u$  such that  $(u, v)$  is an edge of the rooted tree

**child of a vertex  $v$  in a rooted tree:** any vertex with  $v$  as its parent

**sibling of a vertex  $v$  in a rooted tree:** a vertex with the same parent as  $v$

**ancestor of a vertex  $v$  in a rooted tree:** any vertex on the path from the root to  $v$

**descendant of a vertex  $v$  in a rooted tree:** any vertex that has  $v$  as an ancestor

**internal vertex:** a vertex that has children

**leaf:** a vertex with no children

**level of a vertex:** the length of the path from the root to this vertex

**height of a tree:** the largest level of the vertices of a tree

**$m$ -ary tree:** a tree with the property that every internal vertex has no more than  $m$  children

**full  $m$ -ary tree:** a tree with the property that every internal vertex has exactly  $m$  children

**binary tree:** an  $m$ -ary tree with  $m = 2$  (each child may be designated as a left or a right child of its parent)

**ordered tree:** a tree in which the children of each internal vertex are linearly ordered

**balanced tree:** a tree in which every leaf is at level  $h$  or  $h - 1$ , where  $h$  is the height of the tree

**binary search tree:** a binary tree in which the vertices are labeled with items so that a label of a vertex is greater than the labels of all vertices in the left subtree of this vertex and is less than the labels of all vertices in the right subtree of this vertex

**decision tree:** a rooted tree where each vertex represents a possible outcome of a decision and the leaves represent the possible solutions of a problem

**game tree:** a rooted tree where vertices represent the possible positions of a game as it progresses and edges represent legal moves between these positions

**prefix code:** a code that has the property that the code of a character is never a prefix of the code of another character

**minmax strategy:** the strategy where the first player and second player move to positions represented by a child with maximum and minimum value, respectively

**value of a vertex in a game tree:** for a leaf, the payoff to the first player when the game terminates in the position represented by this leaf; for an internal vertex, the maximum or minimum of the values of its children, for an internal vertex at an even or odd level, respectively

**tree traversal:** a listing of the vertices of a tree

**preorder traversal:** a listing of the vertices of an ordered rooted tree defined recursively—the root is listed, followed by the first subtree, followed by the other subtrees in the order they occur from left to right

**inorder traversal:** a listing of the vertices of an ordered rooted tree defined recursively—the first subtree is listed, followed by the root, followed by the other subtrees in the order they occur from left to right

**postorder traversal:** a listing of the vertices of an ordered rooted tree defined recursively—the subtrees are listed in the order they occur from left to right, followed by the root

**infix notation:** the form of an expression (including a full set of parentheses) obtained from an inorder traversal of the binary tree representing this expression

**prefix (or Polish) notation:** the form of an expression obtained from a preorder traversal of the tree representing this expression

**postfix (or reverse Polish) notation:** the form of an expression obtained from a postorder traversal of the tree representing this expression

**spanning tree:** a tree containing all vertices of a graph

**minimum spanning tree:** a spanning tree with smallest possible sum of weights of its edges

## RESULTS

A graph is a tree if and only if there is a unique simple path between every pair of its vertices.

A tree with  $n$  vertices has  $n - 1$  edges.

A full  $m$ -ary tree with  $i$  internal vertices has  $mi + 1$  vertices.

The relationships among the numbers of vertices, leaves, and internal vertices in a full  $m$ -ary tree (see Theorem 4 in Section 11.1)

There are at most  $m^h$  leaves in an  $m$ -ary tree of height  $h$ .

If an  $m$ -ary tree has  $l$  leaves, its height  $h$  is at least  $\lceil \log_m l \rceil$ . If the tree is also full and balanced, then its height is  $\lceil \log_m l \rceil$ .

**Huffman coding:** a procedure for constructing an optimal binary code for a set of symbols, given the frequencies of these symbols

**depth-first search, or backtracking:** a procedure for constructing a spanning tree by adding edges that form a path until this is not possible, and then moving back up the path until a vertex is found where a new path can be formed

**breadth-first search:** a procedure for constructing a spanning tree that successively adds all edges incident to the last set of edges added, unless a simple circuit is formed

**Prim's algorithm:** a procedure for producing a minimum spanning tree in a weighted graph that successively adds edges with minimal weight among all edges incident to a vertex already in the tree so that no edge produces a simple circuit when it is added

**Kruskal's algorithm:** a procedure for producing a minimum spanning tree in a weighted graph that successively adds edges of least weight that are not already in the tree such that no edge produces a simple circuit when it is added

## Review Questions

1. a) Define a tree. b) Define a forest.
2. Can there be two different simple paths between the vertices of a tree?
3. Give at least three examples of how trees are used in modeling.
4. a) Define a rooted tree and the root of such a tree.  
b) Define the parent of a vertex and a child of a vertex in a rooted tree.  
c) What are an internal vertex, a leaf, and a subtree in a rooted tree?  
d) Draw a rooted tree with at least 10 vertices, where the degree of each vertex does not exceed 3. Identify the

- root, the parent of each vertex, the children of each vertex, the internal vertices, and the leaves.
5. a) How many edges does a tree with  $n$  vertices have?  
b) What do you need to know to determine the number of edges in a forest with  $n$  vertices?
  6. a) Define a full  $m$ -ary tree.  
b) How many vertices does a full  $m$ -ary tree have if it has  $i$  internal vertices? How many leaves does the tree have?
  7. a) What is the height of a rooted tree?  
b) What is a balanced tree?  
c) How many leaves can an  $m$ -ary tree of height  $h$  have?
  8. a) What is a binary search tree?  
b) Describe an algorithm for constructing a binary search tree.  
c) Form a binary search tree for the words *vireo*, *warbler*, *egret*, *grosbeak*, *nuthatch*, and *kingfisher*.
  9. a) What is a prefix code?  
b) How can a prefix code be represented by a binary tree?
  10. a) Define preorder, inorder, and postorder tree traversal.  
b) Give an example of preorder, postorder, and inorder traversal of a binary tree of your choice with at least 12 vertices.
  11. a) Explain how to use preorder, inorder, and postorder traversals to find the prefix, infix, and postfix forms of an arithmetic expression.  
b) Draw the ordered rooted tree that represents  $((x - 3) + ((x/4) + (x - y) \uparrow 3))$ .  
c) Find the prefix and postfix forms of the expression in part (b).
  12. Show that the number of comparisons used by a sorting algorithm to sort a list of  $n$  elements is at least  $\lceil \log n! \rceil$ .
  13. a) Describe the Huffman coding algorithm for constructing an optimal code for a set of symbols, given the frequency of these symbols.

- b) Use Huffman coding to find an optimal code for these symbols and frequencies: A: 0.2, B: 0.1, C: 0.3, D: 0.4.

14. Draw the game tree for nim if the starting position consists of two piles with one and four stones, respectively. Who wins the game if both players follow an optimal strategy?
15. a) What is a spanning tree of a simple graph?  
b) Which simple graphs have spanning trees?  
c) Describe at least two different applications that require that a spanning tree of a simple graph be found.
16. a) Describe two different algorithms for finding a spanning tree in a simple graph.  
b) Illustrate how the two algorithms you described in part (a) can be used to find the spanning tree of a simple graph, using a graph of your choice with at least eight vertices and 15 edges.
17. a) Explain how backtracking can be used to determine whether a simple graph can be colored using  $n$  colors.  
b) Show, with an example, how backtracking can be used to show that a graph with a chromatic number equal to 4 cannot be colored with three colors, but can be colored with four colors.
18. a) What is a minimum spanning tree of a connected weighted graph?  
b) Describe at least two different applications that require that a minimum spanning tree of a connected weighted graph be found.
19. a) Describe Kruskal's algorithm and Prim's algorithm for finding minimum spanning trees.  
b) Illustrate how Kruskal's algorithm and Prim's algorithm are used to find a minimum spanning tree, using a weighted graph with at least eight vertices and 15 edges.

## Supplementary Exercises

- \*1. Show that a simple graph is a tree if and only if it contains no simple circuits and the addition of an edge connecting two nonadjacent vertices produces a new graph that has exactly one simple circuit (where circuits that contain the same edges are not considered different).
- \*2. How many nonisomorphic rooted trees are there with six vertices?
3. Show that every tree with at least one edge must have at least two pendant vertices.
4. Show that a tree with  $n$  vertices that has  $n - 1$  pendant vertices must be isomorphic to  $K_{1,n-1}$ .
5. What is the sum of the degrees of the vertices of a tree with  $n$  vertices?
- \*6. Suppose that  $d_1, d_2, \dots, d_n$  are  $n$  positive integers with sum  $2n - 2$ . Show that there is a tree that has  $n$  vertices such that the degrees of these vertices are  $d_1, d_2, \dots, d_n$ .

7. Show that every tree is a planar graph.

8. Show that every tree is bipartite.

9. Show that every forest can be colored using two colors.

 A **B-tree of degree  $k$**  is a rooted tree such that all its leaves are at the same level, its root has at least two and at most  $k$  children unless it is a leaf, and every internal vertex other than the root has at least  $\lceil k/2 \rceil$ , but no more than  $k$ , children. Computer files can be accessed efficiently when B-trees are used to represent them.

10. Draw three different B-trees of degree 3 with height 4.

- \*11. Give an upper bound and a lower bound for the number of leaves in a B-tree of degree  $k$  with height  $h$ .

- \*12. Give an upper bound and a lower bound for the height of a B-tree of degree  $k$  with  $n$  leaves.

The **binomial trees**  $B_i$ ,  $i = 0, 1, 2, \dots$ , are ordered rooted trees defined recursively:

**Basis step:** The binomial tree  $B_0$  is the tree with a single vertex.

**Recursive step:** Let  $k$  be a nonnegative integer. To construct the binomial tree  $B_{k+1}$ , add a copy of  $B_k$  to a second copy of  $B_k$  by adding an edge that makes the root of the first copy of  $B_k$  the leftmost child of the root of the second copy of  $B_k$ .

13. Draw  $B_k$  for  $k = 0, 1, 2, 3, 4$ .
14. How many vertices does  $B_k$  have? Prove that your answer is correct.
15. Find the height of  $B_k$ . Prove that your answer is correct.
16. How many vertices are there in  $B_k$  at depth  $j$ , where  $0 \leq j \leq k$ ? Justify your answer.
17. What is the degree of the root of  $B_k$ ? Prove that your answer is correct.

18. Show that the vertex of largest degree in  $B_k$  is the root. A rooted tree  $T$  is called an  **$S_k$ -tree** if it satisfies this recursive definition. It is an  $S_0$ -tree if it has one vertex. For  $k > 0$ ,  $T$  is an  $S_k$ -tree if it can be built from two  $S_{k-1}$ -trees by making the root of one the root of the  $S_k$ -tree and making the root of the other the child of the root of the first  $S_{k-1}$ -tree.

19. Draw an  $S_k$ -tree for  $k = 0, 1, 2, 3, 4$ .
20. Show that an  $S_k$ -tree has  $2^k$  vertices and a unique vertex at level  $k$ . This vertex at level  $k$  is called the **handle**.
- \*21. Suppose that  $T$  is an  $S_k$ -tree with handle  $v$ . Show that  $T$  can be obtained from disjoint trees  $T_0, T_1, \dots, T_{k-1}$ , with roots  $r_0, r_1, \dots, r_{k-1}$ , respectively, where  $v$  is not in any of these trees, where  $T_i$  is an  $S_i$ -tree for  $i = 0, 1, \dots, k-1$ , by connecting  $v$  to  $r_0$  and  $r_i$  to  $r_{i+1}$  for  $i = 0, 1, \dots, k-2$ .

The listing of the vertices of an ordered rooted tree in **level order** begins with the root, followed by the vertices at level 1 from left to right, followed by the vertices at level 2 from left to right, and so on.

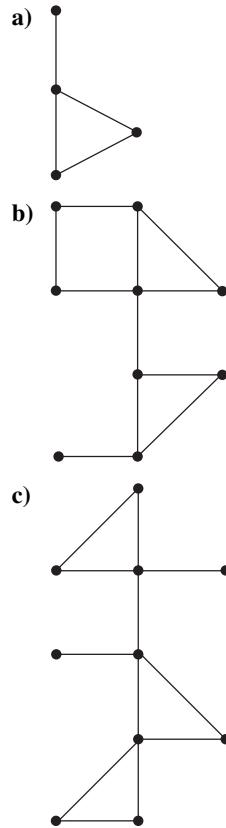
22. List the vertices of the ordered rooted trees in Figures 3 and 9 of Section 11.3 in level order.
23. Devise an algorithm for listing the vertices of an ordered rooted tree in level order.
- \*24. Devise an algorithm for determining if a set of universal addresses can be the addresses of the leaves of a rooted tree.
25. Devise an algorithm for constructing a rooted tree from the universal addresses of its leaves.

A **cut set** of a graph is a set of edges such that the removal of these edges produces a subgraph with more connected components than in the original graph, but no proper subset of this set of edges has this property.

26. Show that a cut set of a graph must have at least one edge in common with any spanning tree of this graph.

A **cactus** is a connected graph in which no edge is in more than one simple circuit not passing through any vertex other than its initial vertex more than once or its initial vertex other than at its terminal vertex (where two circuits that contain the same edges are not considered different).

27. Which of these graphs are cacti?

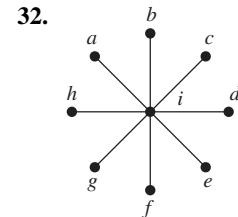
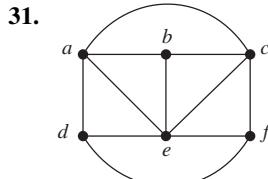


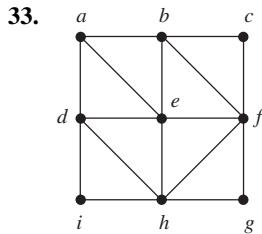
28. Is a tree necessarily a cactus?

29. Show that a cactus is formed if we add a circuit containing new edges beginning and ending at a vertex of a tree.
- \*30. Show that if every circuit not passing through any vertex other than its initial vertex more than once in a connected graph contains an odd number of edges, then this graph must be a cactus.

A **degree-constrained spanning tree** of a simple graph  $G$  is a spanning tree with the property that the degree of a vertex in this tree cannot exceed some specified bound. Degree-constrained spanning trees are useful in models of transportation systems where the number of roads at an intersection is limited, models of communications networks where the number of links entering a node is limited, and so on.

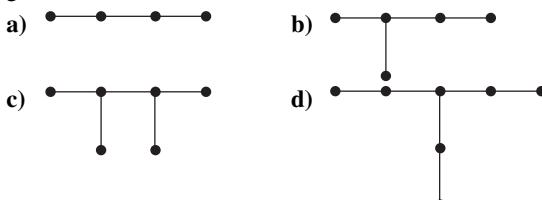
In Exercises 31–33 find a degree-constrained spanning tree of the given graph where each vertex has degree less than or equal to 3, or show that such a spanning tree does not exist.





34. Show that a degree-constrained spanning tree of a simple graph in which each vertex has degree not exceeding 2 consists of a single Hamilton path in the graph.

35. A tree with  $n$  vertices is called **graceful** if its vertices can be labeled with the integers  $1, 2, \dots, n$  such that the absolute values of the difference of the labels of adjacent vertices are all different. Show that these trees are graceful.

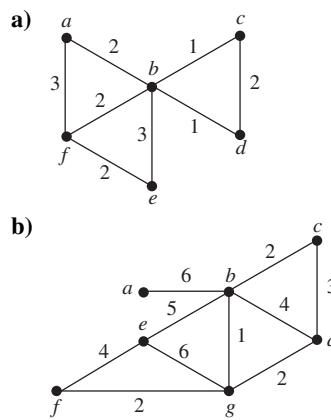


A **caterpillar** is a tree that contains a simple path such that every vertex not contained in this path is adjacent to a vertex in the path.

36. Which of the graphs in Exercise 35 are caterpillars?
37. How many nonisomorphic caterpillars are there with six vertices?
- \*\*38. a) Prove or disprove that all trees whose edges form a single path are graceful.  
 b) Prove or disprove that all caterpillars are graceful.
39. Suppose that in a long bit string the frequency of occurrence of a 0 bit is 0.9 and the frequency of a 1 bit is 0.1 and bits occur independently.
- a) Construct a Huffman code for the four blocks of two bits, 00, 01, 10, and 11. What is the average number of bits required to encode a bit string using this code?  
 b) Construct a Huffman code for the eight blocks of three bits. What is the average number of bits required to encode a bit string using this code?
40. Suppose that  $G$  is a directed graph with no circuits. Describe how depth-first search can be used to carry out a topological sort of the vertices of  $G$ .
- \*41. Suppose that  $e$  is an edge in a weighted graph that is incident to a vertex  $v$  such that the weight of  $e$  does not exceed the weight of any other edge incident to  $v$ . Show that there exists a minimum spanning tree containing this edge.
42. Three couples arrive at the bank of a river. Each of the wives is jealous and does not trust her husband when he is with one of the other wives (and perhaps with other

people), but not with her. How can six people cross to the other side of the river using a boat that can hold no more than two people so that no husband is alone with a woman other than his wife? Use a graph theory model.

- \*43. Show that if no two edges in a weighted graph have the same weight, then the edge with least weight incident to a vertex  $v$  is included in every minimum spanning tree.
44. Find a minimum spanning tree of each of these graphs where the degree of each vertex in the spanning tree does not exceed 2.



Let  $G = (V, E)$  be a directed graph and let  $r$  be a vertex in  $G$ . An **arborescence** of  $G$  rooted at  $r$  is a subgraph  $T = (V, F)$  of  $G$  such that the underlying undirected graph of  $T$  is a spanning tree of the underlying undirected graph of  $G$  and for every vertex  $v \in V$  there is a path from  $r$  to  $v$  in  $T$  (with directions taken into account).

45. Show that a subgraph  $T = (V, F)$  of the graph  $G = (V, E)$  is an arborescence of  $G$  rooted at  $r$  if and only if  $T$  contains  $r$ ,  $T$  has no simple circuits, and for every vertex  $v \in V$  other than  $r$ ,  $\deg^-(v) = 1$  in  $T$ .
46. Show that a directed graph  $G = (V, E)$  has an arborescence rooted at the vertex  $r$  if and only if for every vertex  $v \in V$ , there is a directed path from  $r$  to  $v$ .
47. In this exercise we will develop an algorithm to find the strong components of a directed graph  $G = (V, E)$ . Recall that a vertex  $w \in V$  is **reachable** from a vertex  $v \in V$  if there is a directed path from  $v$  to  $w$ .
- a) Explain how to use breadth-first search in the directed graph  $G$  to find all the vertices reachable from a vertex  $v \in G$ .  
 b) Explain how to use breadth-first search in  $G^{conv}$  to find all the vertices from which a vertex  $v \in G$  is reachable. (Recall that  $G^{conv}$  is the directed graph obtained from  $G$  by reversing the direction of all its edges.)  
 c) Explain how to use parts (a) and (b) to construct an algorithm that finds the strong components of a directed graph  $G$ , and explain why your algorithm is correct.

## Computer Projects

---

**Write programs with these input and output.**

1. Given the adjacency matrix of an undirected simple graph, determine whether the graph is a tree.
2. Given the adjacency matrix of a rooted tree and a vertex in the tree, find the parent, children, ancestors, descendants, and level of this vertex.
3. Given the list of edges of a rooted tree and a vertex in the tree, find the parent, children, ancestors, descendants, and level of this vertex.
4. Given a list of items, construct a binary search tree containing these items.
5. Given a binary search tree and an item, locate or add this item to the binary search tree.
6. Given the ordered list of edges of an ordered rooted tree, find the universal addresses of its vertices.
7. Given the ordered list of edges of an ordered rooted tree, list its vertices in preorder, inorder, and postorder.
8. Given an arithmetic expression in prefix form, find its value.
9. Given an arithmetic expression in postfix form, find its value.
10. Given the frequency of symbols, use Huffman coding to find an optimal code for these symbols.
11. Given an initial position in the game of nim, determine an optimal strategy for the first player.
12. Given the adjacency matrix of a connected undirected simple graph, find a spanning tree for this graph using depth-first search.
13. Given the adjacency matrix of a connected undirected simple graph, find a spanning tree for this graph using breadth-first search.
14. Given a set of positive integers and a positive integer  $N$ , use backtracking to find a subset of these integers that have  $N$  as their sum.
- \*15. Given the adjacency matrix of an undirected simple graph, use backtracking to color the graph with three colors, if this is possible.
- \*16. Given a positive integer  $n$ , solve the  $n$ -queens problem using backtracking.
17. Given the list of edges and their weights of a weighted undirected connected graph, use Prim's algorithm to find a minimum spanning tree of this graph.
18. Given the list of edges and their weights of a weighted undirected connected graph, use Kruskal's algorithm to find a minimum spanning tree of this graph.

## Computations and Explorations

---

**Use a computational program or programs you have written to do these exercises.**

1. Display all trees with six vertices.
2. Display a full set of nonisomorphic trees with seven vertices.
- \*3. Construct a Huffman code for the symbols with ASCII codes given the frequency of their occurrence in representative input.
4. Compute the number of different spanning trees of  $K_n$  for  $n = 1, 2, 3, 4, 5, 6$ . Conjecture a formula for the number of such spanning trees whenever  $n$  is a positive integer.
5. Compare the number of comparisons needed to sort lists of  $n$  elements for  $n = 100, 1000$ , and  $10,000$  from the set of positive integers less than  $1,000,000$ , where the elements

are randomly selected positive integers, using the selection sort, the insertion sort, the merge sort, and the quick sort.

6. Compute the number of different ways  $n$  queens can be arranged on an  $n \times n$  chessboard so that no two queens can attack each other for all positive integers  $n$  not exceeding 10.
- \*7. Find a minimum spanning tree of the graph that connects the capital cities of the 50 states in the United States to each other where the weight of each edge is the distance between the cities.
8. Draw the complete game tree for a game of checkers on a  $4 \times 4$  board.

## Writing Projects

---

**Respond to these with essays using outside sources.**

1. Explain how Cayley used trees to enumerate the number of certain types of hydrocarbons.
2. Explain how trees are used to represent ancestral relations in the study of evolution.
3. Discuss hierarchical cluster trees and how they are used.
4. Define *AVL-trees* (sometimes also known as *height-balanced trees*). Describe how and why AVL-trees are used in a variety of different algorithms.

5. Define *quad trees* and explain how images can be represented using them. Describe how images can be rotated, scaled, and translated by manipulating the corresponding quad tree.
6. Define a *heap* and explain how trees can be turned into heaps. Why are heaps useful in sorting?
7. Describe dynamic algorithms for data compression based on letter frequencies as they change as characters are successively read, such as adaptive Huffman coding.
8. Explain how *alpha-beta pruning* can be used to simplify the computation of the value of a game tree.
9. Describe the techniques used by chess-playing programs such as Deep Blue.
10. Define the type of graph known as a *mesh of trees*. Explain how this graph is used in applications to very large system integration and parallel computing.
11. Discuss the algorithms used in IP multicasting to avoid loops between routers.
12. Describe an algorithm based on depth-first search for finding the articulation points of a graph.
13. Describe an algorithm based on depth-first search to find the strongly connected components of a directed graph.
14. Describe the search techniques used by the crawlers and spiders in different search engines on the Web.
15. Describe an algorithm for finding the minimum spanning tree of a graph such that the maximum degree of any vertex in the spanning tree does not exceed a fixed constant  $k$ .
16. Compare and contrast some of the most important sorting algorithms in terms of their complexity and when they are used.
17. Discuss the history and origins of algorithms for constructing minimum spanning trees.
18. Describe algorithms for producing random trees.



# 12

# Boolean Algebra

- 12.1** Boolean Functions
- 12.2** Representing Boolean Functions
- 12.3** Logic Gates
- 12.4** Minimization of Circuits

The circuits in computers and other electronic devices have inputs, each of which is either a 0 or a 1, and produce outputs that are also 0s and 1s. Circuits can be constructed using any basic element that has two different states. Such elements include switches that can be in either the on or the off position and optical devices that can be either lit or unlit. In 1938 Claude Shannon showed how the basic rules of logic, first given by George Boole in 1854 in his *The Laws of Thought*, could be used to design circuits. These rules form the basis for Boolean algebra. In this chapter we develop the basic properties of Boolean algebra. The operation of a circuit is defined by a Boolean function that specifies the value of an output for each set of inputs. The first step in constructing a circuit is to represent its Boolean function by an expression built up using the basic operations of Boolean algebra. We will provide an algorithm for producing such expressions. The expression that we obtain may contain many more operations than are necessary to represent the function. Later in the chapter we will describe methods for finding an expression with the minimum number of sums and products that represents a Boolean function. The procedures that we will develop, Karnaugh maps and the Quine–McCluskey method, are important in the design of efficient circuits.

## 12.1 Boolean Functions

### Introduction

Boolean algebra provides the operations and the rules for working with the set  $\{0, 1\}$ . Electronic and optical switches can be studied using this set and the rules of Boolean algebra. The three operations in Boolean algebra that we will use most are complementation, the Boolean sum, and the Boolean product. The **complement** of an element, denoted with a bar, is defined by  $\bar{0} = 1$  and  $\bar{1} = 0$ . The Boolean sum, denoted by  $+$  or by *OR*, has the following values:

$$1 + 1 = 1, \quad 1 + 0 = 1, \quad 0 + 1 = 1, \quad 0 + 0 = 0.$$

The Boolean product, denoted by  $\cdot$  or by *AND*, has the following values:

$$1 \cdot 1 = 1, \quad 1 \cdot 0 = 0, \quad 0 \cdot 1 = 0, \quad 0 \cdot 0 = 0.$$

When there is no danger of confusion, the symbol  $\cdot$  can be deleted, just as in writing algebraic products. Unless parentheses are used, the rules of precedence for Boolean operators are: first, all complements are computed, followed by all Boolean products, followed by all Boolean sums. This is illustrated in Example 1.

**EXAMPLE 1** Find the value of  $1 \cdot 0 + \overline{(0 + 1)}$ .

**Solution:** Using the definitions of complementation, the Boolean sum, and the Boolean product, it follows that

$$\begin{aligned} 1 \cdot 0 + \overline{(0 + 1)} &= 0 + \bar{1} \\ &= 0 + 0 \\ &= 0. \end{aligned}$$

The complement, Boolean sum, and Boolean product correspond to the logical operators,  $\neg$ ,  $\vee$ , and  $\wedge$ , respectively, where 0 corresponds to **F** (false) and 1 corresponds to **T** (true). Equalities in Boolean algebra can be directly translated into equivalences of compound propositions. Conversely, equivalences of compound propositions can be translated into equalities in Boolean algebra. We will see later in this section why these translations yield valid logical equivalences and identities in Boolean algebra. Example 2 illustrates the translation from Boolean algebra to propositional logic.

**EXAMPLE 2** Translate  $1 \cdot 0 + \overline{(0+1)} = 0$ , the equality found in Example 1, into a logical equivalence.

**Solution:** We obtain a logical equivalence when we translate each 1 into a **T**, each 0 into an **F**, each Boolean sum into a disjunction, each Boolean product into a conjunction, and each complementation into a negation. We obtain

$$(T \wedge F) \vee \neg(T \vee F) \equiv F.$$

Example 3 illustrates the translation from propositional logic to Boolean algebra.

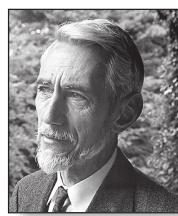
**EXAMPLE 3** Translate the logical equivalence  $(T \wedge T) \vee \neg F \equiv T$  into an identity in Boolean algebra.

**Solution:** We obtain an identity in Boolean algebra when we translate each **T** into a 1, each **F** into a 0, each disjunction into a Boolean sum, each conjunction into a Boolean product, and each negation into a complementation. We obtain

$$(1 \cdot 1) + \overline{0} = 1.$$

## Boolean Expressions and Boolean Functions

Let  $B = \{0, 1\}$ . Then  $B^n = \{(x_1, x_2, \dots, x_n) \mid x_i \in B \text{ for } 1 \leq i \leq n\}$  is the set of all possible  $n$ -tuples of 0s and 1s. The variable  $x$  is called a **Boolean variable** if it assumes values only from  $B$ , that is, if its only possible values are 0 and 1. A function from  $B^n$  to  $B$  is called a **Boolean function of degree  $n$** .



**CLAUDE ELWOOD SHANNON (1916–2001)** Claude Shannon was born in Petoskey, Michigan, and grew up in Gaylord, Michigan. His father was a businessman and a probate judge, and his mother was a language teacher and a high school principal. Shannon attended the University of Michigan, graduating in 1936. He continued his studies at M.I.T., where he took the job of maintaining the differential analyzer, a mechanical computing device consisting of shafts and gears built by his professor, Vannevar Bush. Shannon's master's thesis, written in 1936, studied the logical aspects of the differential analyzer. This master's thesis presents the first application of Boolean algebra to the design of switching circuits; it is perhaps the most famous master's thesis of the twentieth century. He received his Ph.D. from M.I.T. in 1940. Shannon joined Bell Laboratories in 1940, where he worked on transmitting data efficiently. He was one of the first people to use bits to represent information. At Bell Laboratories he worked on determining the amount of traffic that telephone lines can carry. Shannon made many fundamental contributions to information theory. In the early 1950s he was one of the founders of the study of artificial intelligence. He joined the M.I.T. faculty in 1956, where he continued his study of information theory.

Shannon had an unconventional side. He is credited with inventing the rocket-powered Frisbee. He is also famous for riding a unicycle down the hallways of Bell Laboratories while juggling four balls. Shannon retired when he was 50 years old, publishing papers sporadically over the following 10 years. In his later years he concentrated on some pet projects, such as building a motorized pogo stick. One interesting quote from Shannon, published in *Omni Magazine* in 1987, is “I visualize a time when we will be to robots what dogs are to humans. And I am rooting for the machines.”

**EXAMPLE 4**

The function  $F(x, y) = x\bar{y}$  from the set of ordered pairs of Boolean variables to the set  $\{0, 1\}$  is a Boolean function of degree 2 with  $F(1, 1) = 0$ ,  $F(1, 0) = 1$ ,  $F(0, 1) = 0$ , and  $F(0, 0) = 0$ . We display these values of  $F$  in Table 1.

| TABLE 1 |     |           |
|---------|-----|-----------|
| $x$     | $y$ | $F(x, y)$ |
| 1       | 1   | 0         |
| 1       | 0   | 1         |
| 0       | 1   | 0         |
| 0       | 0   | 0         |

Boolean functions can be represented using expressions made up from variables and Boolean operations. The **Boolean expressions** in the variables  $x_1, x_2, \dots, x_n$  are defined recursively as

$0, 1, x_1, x_2, \dots, x_n$  are Boolean expressions;  
if  $E_1$  and  $E_2$  are Boolean expressions, then  $\bar{E}_1$ ,  $(E_1 E_2)$ , and  $(E_1 + E_2)$  are Boolean expressions.

Each Boolean expression represents a Boolean function. The values of this function are obtained by substituting 0 and 1 for the variables in the expression. In Section 12.2 we will show that every Boolean function can be represented by a Boolean expression.

**EXAMPLE 5**

Find the values of the Boolean function represented by  $F(x, y, z) = xy + \bar{z}$ .

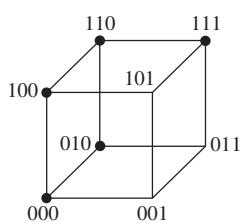
*Solution:* The values of this function are displayed in Table 2.

| TABLE 2 |     |     |      |           |                             |
|---------|-----|-----|------|-----------|-----------------------------|
| $x$     | $y$ | $z$ | $xy$ | $\bar{z}$ | $F(x, y, z) = xy + \bar{z}$ |
| 1       | 1   | 1   | 1    | 0         | 1                           |
| 1       | 1   | 0   | 1    | 1         | 1                           |
| 1       | 0   | 1   | 0    | 0         | 0                           |
| 1       | 0   | 0   | 0    | 1         | 1                           |
| 0       | 1   | 1   | 0    | 0         | 0                           |
| 0       | 1   | 0   | 0    | 1         | 1                           |
| 0       | 0   | 1   | 0    | 0         | 0                           |
| 0       | 0   | 0   | 0    | 1         | 1                           |

Note that we can represent a Boolean function graphically by distinguishing the vertices of the  $n$ -cube that correspond to the  $n$ -tuples of bits where the function has value 1.

**EXAMPLE 6**

The function  $F(x, y, z) = xy + \bar{z}$  from  $B^3$  to  $B$  from Example 5 can be represented by distinguishing the vertices that correspond to the five 3-tuples  $(1, 1, 1)$ ,  $(1, 1, 0)$ ,  $(1, 0, 0)$ ,  $(0, 1, 0)$ , and  $(0, 0, 0)$ , where  $F(x, y, z) = 1$ , as shown in Figure 1. These vertices are displayed using solid black circles.

**FIGURE 1**

Boolean functions  $F$  and  $G$  of  $n$  variables are equal if and only if  $F(b_1, b_2, \dots, b_n) = G(b_1, b_2, \dots, b_n)$  whenever  $b_1, b_2, \dots, b_n$  belong to  $B$ . Two different Boolean expressions that represent the same function are called **equivalent**. For instance, the Boolean expressions  $xy$ ,  $xy + 0$ , and  $xy \cdot 1$  are equivalent. The **complement** of the Boolean function  $F$  is the function  $\bar{F}$ , where  $\bar{F}(x_1, \dots, x_n) = F(\bar{x}_1, \dots, \bar{x}_n)$ . Let  $F$  and  $G$  be Boolean functions of degree  $n$ . The **Boolean sum**  $F + G$  and the **Boolean product**  $FG$  are defined by

$$(F + G)(x_1, \dots, x_n) = F(x_1, \dots, x_n) + G(x_1, \dots, x_n),$$

$$(FG)(x_1, \dots, x_n) = F(x_1, \dots, x_n)G(x_1, \dots, x_n).$$

A Boolean function of degree two is a function from a set with four elements, namely, pairs of elements from  $B = \{0, 1\}$ , to  $B$ , a set with two elements. Hence, there are 16 different Boolean functions of degree two. In Table 3 we display the values of the 16 different Boolean functions of degree two, labeled  $F_1, F_2, \dots, F_{16}$ .

**TABLE 3** The 16 Boolean Functions of Degree Two.

| $x$ | $y$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ | $F_{10}$ | $F_{11}$ | $F_{12}$ | $F_{13}$ | $F_{14}$ | $F_{15}$ | $F_{16}$ |
|-----|-----|-------|-------|-------|-------|-------|-------|-------|-------|-------|----------|----------|----------|----------|----------|----------|----------|
| 1   | 1   | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 1     | 0     | 0        | 0        | 0        | 0        | 0        | 0        | 0        |
| 1   | 0   | 1     | 1     | 1     | 1     | 0     | 0     | 0     | 0     | 1     | 1        | 1        | 0        | 0        | 0        | 0        | 0        |
| 0   | 1   | 1     | 1     | 0     | 0     | 1     | 1     | 0     | 0     | 1     | 1        | 0        | 0        | 1        | 1        | 0        | 0        |
| 0   | 0   | 1     | 0     | 1     | 0     | 1     | 0     | 1     | 0     | 1     | 0        | 1        | 0        | 1        | 0        | 1        | 0        |

**EXAMPLE 7** How many different Boolean functions of degree  $n$  are there?

*Solution:* From the product rule for counting, it follows that there are  $2^n$  different  $n$ -tuples of 0s and 1s. Because a Boolean function is an assignment of 0 or 1 to each of these  $2^n$  different  $n$ -tuples, the product rule shows that there are  $2^{2^n}$  different Boolean functions of degree  $n$ .  $\blacktriangleleft$

Table 4 displays the number of different Boolean functions of degrees one through six. The number of such functions grows extremely rapidly.

**TABLE 4** The Number of Boolean Functions of Degree  $n$ .

| Degree | Number                     |
|--------|----------------------------|
| 1      | 4                          |
| 2      | 16                         |
| 3      | 256                        |
| 4      | 65,536                     |
| 5      | 4,294,967,296              |
| 6      | 18,446,744,073,709,551,616 |

## Identities of Boolean Algebra

There are many identities in Boolean algebra. The most important of these are displayed in Table 5. These identities are particularly useful in simplifying the design of circuits. Each of the identities in Table 5 can be proved using a table. We will prove one of the distributive laws in this way in Example 8. The proofs of the remaining properties are left as exercises for the reader.

**EXAMPLE 8** Show that the distributive law  $x(y + z) = xy + xz$  is valid.

*Solution:* The verification of this identity is shown in Table 6. The identity holds because the last two columns of the table agree.  $\blacktriangleleft$

The reader should compare the Boolean identities in Table 5 to the logical equivalences in Table 6 of Section 1.3 and the set identities in Table 1 in Section 2.2. All are special cases of the same set of identities in a more abstract structure. Each collection of identities can be obtained by making the appropriate translations. For example, we can transform each of the identities in Table 5 into a logical equivalence by changing each Boolean variable into a propositional variable, each 0 into a **F**, each 1 into a **T**, each Boolean sum into a disjunction, each Boolean product into a conjunction, and each complementation into a negation, as we illustrate in Example 9.

**TABLE 5** Boolean Identities.

| <i>Identity</i>                                                                 | <i>Name</i>                  |
|---------------------------------------------------------------------------------|------------------------------|
| $\bar{\bar{x}} = x$                                                             | Law of the double complement |
| $x + x = x$<br>$x \cdot x = x$                                                  | Idempotent laws              |
| $x + 0 = x$<br>$x \cdot 1 = x$                                                  | Identity laws                |
| $x + 1 = 1$<br>$x \cdot 0 = 0$                                                  | Domination laws              |
| $x + y = y + x$<br>$xy = yx$                                                    | Commutative laws             |
| $x + (y + z) = (x + y) + z$<br>$x(yz) = (xy)z$                                  | Associative laws             |
| $x + yz = (x + y)(x + z)$<br>$x(y + z) = xy + xz$                               | Distributive laws            |
| $\overline{(xy)} = \bar{x} + \bar{y}$<br>$\overline{(x + y)} = \bar{x} \bar{y}$ | De Morgan's laws             |
| $x + xy = x$<br>$x(x + y) = x$                                                  | Absorption laws              |
| $x + \bar{x} = 1$                                                               | Unit property                |
| $x\bar{x} = 0$                                                                  | Zero property                |

Compare these Boolean identities with the logical equivalences in Section 1.3 and the set identities in Section 2.2.

## EXAMPLE 9

Translate the distributive law  $x + yz = (x + y)(x + z)$  in Table 5 into a logical equivalence.

**Solution:** To translate a Boolean identity into a logical equivalence, we change each Boolean variable into a propositional variable. Here we will change the Boolean variables  $x$ ,  $y$ , and  $z$  into the propositional variables  $p$ ,  $q$ , and  $r$ . Next, we change each Boolean sum into a disjunction and each Boolean product into a conjunction. (Note that 0 and 1 do not appear in this identity and

**TABLE 6** Verifying One of the Distributive Laws.

complementation also does not appear.) This transforms the Boolean identity into the logical equivalence

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r).$$

This logical equivalence is one of the distributive laws for propositional logic in Table 6 in Section 1.3. 

Identities in Boolean algebra can be used to prove further identities. We demonstrate this in Example 10.

**EXAMPLE 10** Prove the **absorption law**  $x(x + y) = x$  using the other identities of Boolean algebra shown in Table 5. (This is called an absorption law because absorbing  $x + y$  into  $x$  leaves  $x$  unchanged.)



*Solution:* We display steps used to derive this identity and the law used in each step:

$$\begin{aligned} x(x + y) &= (x + 0)(x + y) && \text{Identity law for the Boolean sum} \\ &= x + 0 \cdot y && \text{Distributive law of the Boolean sum over the} \\ & & & \text{Boolean product} \\ &= x + y \cdot 0 && \text{Commutative law for the Boolean product} \\ &= x + 0 && \text{Domination law for the Boolean product} \\ &= x && \text{Identity law for the Boolean sum.} \end{aligned}$$



## Duality

The identities in Table 5 come in pairs (except for the law of the double complement and the unit and zero properties). To explain the relationship between the two identities in each pair we use the concept of a dual. The **dual** of a Boolean expression is obtained by interchanging Boolean sums and Boolean products and interchanging 0s and 1s.

**EXAMPLE 11** Find the duals of  $x(y + 0)$  and  $\bar{x} \cdot 1 + (\bar{y} + z)$ .

*Solution:* Interchanging  $\cdot$  signs and  $+$  signs and interchanging 0s and 1s in these expressions produces their duals. The duals are  $x + (y \cdot 1)$  and  $(\bar{x} + 0)(\bar{y}z)$ , respectively. 

The dual of a Boolean function  $F$  represented by a Boolean expression is the function represented by the dual of this expression. This dual function, denoted by  $F^d$ , does not depend on the particular Boolean expression used to represent  $F$ . An identity between functions represented by Boolean expressions remains valid when the duals of both sides of the identity are taken. (See Exercise 30 for the reason why this is true.) This result, called the **duality principle**, is useful for obtaining new identities.

**EXAMPLE 12** Construct an identity from the absorption law  $x(x + y) = x$  by taking duals.

*Solution:* Taking the duals of both sides of this identity produces the identity  $x + xy = x$ , which is also called an absorption law and is shown in Table 5. 

## The Abstract Definition of a Boolean Algebra

In this section we have focused on Boolean functions and expressions. However, the results we have established can be translated into results about propositions or results about sets. Because of this, it is useful to define Boolean algebras abstractly. Once it is shown that a particular structure is a Boolean algebra, then all results established about Boolean algebras in general apply to this particular structure.

Boolean algebras can be defined in several ways. The most common way is to specify the properties that operations must satisfy, as is done in Definition 1.

### DEFINITION 1

A *Boolean algebra* is a set  $B$  with two binary operations  $\vee$  and  $\wedge$ , elements 0 and 1, and a unary operation  $\bar{\phantom{x}}$  such that these properties hold for all  $x$ ,  $y$ , and  $z$  in  $B$ :

$$\begin{aligned} & \left. \begin{array}{l} x \vee 0 = x \\ x \wedge 1 = x \end{array} \right\} \quad \text{Identity laws} \\ & \left. \begin{array}{l} x \vee \bar{x} = 1 \\ x \wedge \bar{x} = 0 \end{array} \right\} \quad \text{Complement laws} \\ & \left. \begin{array}{l} (x \vee y) \vee z = x \vee (y \vee z) \\ (x \wedge y) \wedge z = x \wedge (y \wedge z) \end{array} \right\} \quad \text{Associative laws} \\ & \left. \begin{array}{l} x \vee y = y \vee x \\ x \wedge y = y \wedge x \end{array} \right\} \quad \text{Commutative laws} \\ & \left. \begin{array}{l} x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \\ x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) \end{array} \right\} \quad \text{Distributive laws} \end{aligned}$$

Using the laws given in Definition 1, it is possible to prove many other laws that hold for every Boolean algebra, such as idempotent and domination laws. (See Exercises 35–42.)

From our previous discussion,  $B = \{0, 1\}$  with the *OR* and *AND* operations and the complement operator, satisfies all these properties. The set of propositions in  $n$  variables, with the  $\vee$  and  $\wedge$  operators, **F** and **T**, and the negation operator, also satisfies all the properties of a Boolean algebra, as can be seen from Table 6 in Section 1.3. Similarly, the set of subsets of a universal set  $U$  with the union and intersection operations, the empty set and the universal set, and the set complementation operator, is a Boolean algebra as can be seen by consulting Table 1 in Section 2.2. So, to establish results about each of Boolean expressions, propositions, and sets, we need only prove results about abstract Boolean algebras.

Boolean algebras may also be defined using the notion of a lattice, discussed in Chapter 9. Recall that a lattice  $L$  is a partially ordered set in which every pair of elements  $x$ ,  $y$  has a least upper bound, denoted by  $\text{lub}(x, y)$  and a greatest lower bound denoted by  $\text{glb}(x, y)$ . Given two elements  $x$  and  $y$  of  $L$ , we can define two operations  $\vee$  and  $\wedge$  on pairs of elements of  $L$  by  $x \vee y = \text{lub}(x, y)$  and  $x \wedge y = \text{glb}(x, y)$ .

For a lattice  $L$  to be a Boolean algebra as specified in Definition 1, it must have two properties. First, it must be **complemented**. For a lattice to be complemented it must have a least element 0 and a greatest element 1 and for every element  $x$  of the lattice there must exist an element  $\bar{x}$  such that  $x \vee \bar{x} = 1$  and  $x \wedge \bar{x} = 0$ . Second, it must be **distributive**. This means that for every  $x$ ,  $y$ , and  $z$  in  $L$ ,  $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$  and  $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ . Showing that a complemented, distributive lattice is a Boolean algebra has been left as Supplementary Exercise 39 in Chapter 9.

## Exercises

---

1. Find the values of these expressions.

a)  $1 \cdot \bar{0}$     b)  $1 + \bar{1}$     c)  $\bar{0} \cdot 0$     d)  $\overline{(1 + 0)}$

2. Find the values, if any, of the Boolean variable  $x$  that satisfy these equations.

a)  $x \cdot 1 = 0$     b)  $x + x = 0$   
c)  $x \cdot 1 = x$     d)  $x \cdot \bar{x} = 1$

3. a) Show that  $(1 \cdot 1) + (\overline{0 \cdot 1} + 0) = 1$ .

b) Translate the equation in part (a) into a propositional equivalence by changing each 0 into an **F**, each 1 into a **T**, each Boolean sum into a disjunction, each Boolean product into a conjunction, each complementation into a negation, and the equals sign into a propositional equivalence sign.

4. a) Show that  $(\bar{1} \cdot \bar{0}) + (1 \cdot \bar{0}) = 1$ .

b) Translate the equation in part (a) into a propositional equivalence by changing each 0 into an **F**, each 1 into a **T**, each Boolean sum into a disjunction, each Boolean product into a conjunction, each complementation into a negation, and the equals sign into a propositional equivalence sign.

5. Use a table to express the values of each of these Boolean functions.

a)  $F(x, y, z) = \bar{x}y$   
b)  $F(x, y, z) = x + yz$   
c)  $F(x, y, z) = x\bar{y} + \overline{(xyz)}$   
d)  $F(x, y, z) = x(yz + \bar{y}\bar{z})$

6. Use a table to express the values of each of these Boolean functions.

a)  $F(x, y, z) = \bar{z}$   
b)  $F(x, y, z) = \bar{x}y + \bar{y}z$   
c)  $F(x, y, z) = x\bar{y}z + \overline{(xyz)}$   
d)  $F(x, y, z) = \bar{y}(xz + \bar{x}\bar{z})$

7. Use a 3-cube  $Q_3$  to represent each of the Boolean functions in Exercise 5 by displaying a black circle at each vertex that corresponds to a 3-tuple where this function has the value 1.

8. Use a 3-cube  $Q_3$  to represent each of the Boolean functions in Exercise 6 by displaying a black circle at each vertex that corresponds to a 3-tuple where this function has the value 1.

9. What values of the Boolean variables  $x$  and  $y$  satisfy  $xy = x + y$ ?

10. How many different Boolean functions are there of degree 7?

11. Prove the absorption law  $x + xy = x$  using the other laws in Table 5.

12. Show that  $F(x, y, z) = xy + xz + yz$  has the value 1 if and only if at least two of the variables  $x$ ,  $y$ , and  $z$  have the value 1.

13. Show that  $x\bar{y} + y\bar{z} + \bar{x}z = \bar{x}y + \bar{y}z + x\bar{z}$ .

Exercises 14–23 deal with the Boolean algebra  $\{0, 1\}$  with addition, multiplication, and complement defined at the beginning of this section. In each case, use a table as in Example 8.

14. Verify the law of the double complement.

15. Verify the idempotent laws.

16. Verify the identity laws.

17. Verify the domination laws.

18. Verify the commutative laws.

19. Verify the associative laws.

20. Verify the first distributive law in Table 5.

21. Verify De Morgan's laws.

22. Verify the unit property.

23. Verify the zero property.

The Boolean operator  $\oplus$ , called the *XOR* operator, is defined by  $1 \oplus 1 = 0$ ,  $1 \oplus 0 = 1$ ,  $0 \oplus 1 = 1$ , and  $0 \oplus 0 = 0$ .

24. Simplify these expressions.

a)  $x \oplus 0$     b)  $x \oplus 1$   
c)  $x \oplus x$     d)  $x \oplus \bar{x}$

25. Show that these identities hold.

a)  $x \oplus y = (x + y)\overline{(xy)}$   
b)  $x \oplus y = (x\bar{y}) + (\bar{x}y)$

26. Show that  $x \oplus y = y \oplus x$ .

27. Prove or disprove these equalities.

a)  $x \oplus (y \oplus z) = (x \oplus y) \oplus z$   
b)  $x + (y \oplus z) = (x + y) \oplus (x + z)$   
c)  $x \oplus (y + z) = (x \oplus y) + (x \oplus z)$

28. Find the duals of these Boolean expressions.

a)  $x + y$     b)  $\bar{x}\bar{y}$   
c)  $xyz + \bar{x}\bar{y}\bar{z}$     d)  $x\bar{z} + x \cdot 0 + \bar{x} \cdot 1$

- \*29. Suppose that  $F$  is a Boolean function represented by a Boolean expression in the variables  $x_1, \dots, x_n$ . Show that  $F^d(x_1, \dots, x_n) = \overline{F(\bar{x}_1, \dots, \bar{x}_n)}$ .

- \*30. Show that if  $F$  and  $G$  are Boolean functions represented by Boolean expressions in  $n$  variables and  $F = G$ , then  $F^d = G^d$ , where  $F^d$  and  $G^d$  are the Boolean functions represented by the duals of the Boolean expressions representing  $F$  and  $G$ , respectively. [Hint: Use the result of Exercise 29.]

- \*31. How many different Boolean functions  $F(x, y, z)$  are there such that  $F(\bar{x}, \bar{y}, \bar{z}) = F(x, y, z)$  for all values of the Boolean variables  $x$ ,  $y$ , and  $z$ ?

- \*32. How many different Boolean functions  $F(x, y, z)$  are there such that  $F(\bar{x}, y, z) = F(x, \bar{y}, z) = F(x, y, \bar{z})$  for all values of the Boolean variables  $x$ ,  $y$ , and  $z$ ?

33. Show that you obtain De Morgan's laws for propositions (in Table 6 in Section 1.3) when you transform De Morgan's laws for Boolean algebra in Table 6 into logical equivalences.

34. Show that you obtain the absorption laws for propositions (in Table 6 in Section 1.3) when you transform the absorption laws for Boolean algebra in Table 6 into logical equivalences.

In Exercises 35–42, use the laws in Definition 1 to show that the stated properties hold in every Boolean algebra.

35. Show that in a Boolean algebra, the **idempotent laws**  $x \vee x = x$  and  $x \wedge x = x$  hold for every element  $x$ .
36. Show that in a Boolean algebra, every element  $x$  has a unique complement  $\bar{x}$  such that  $x \vee \bar{x} = 1$  and  $x \wedge \bar{x} = 0$ .
37. Show that in a Boolean algebra, the complement of the element 0 is the element 1 and vice versa.
38. Prove that in a Boolean algebra, the **law of the double complement** holds; that is,  $\bar{\bar{x}} = x$  for every element  $x$ .
39. Show that **De Morgan's laws** hold in a Boolean algebra.

That is, show that for all  $x$  and  $y$ ,  $\overline{(x \vee y)} = \bar{x} \wedge \bar{y}$  and  $(x \wedge y) = \bar{x} \vee \bar{y}$ .

40. Show that in a Boolean algebra, the **modular properties** hold. That is, show that  $x \wedge (y \vee (x \wedge z)) = (x \wedge y) \vee (x \wedge z)$  and  $x \vee (y \wedge (x \vee z)) = (x \vee y) \wedge (x \vee z)$ .
41. Show that in a Boolean algebra, if  $x \vee y = 0$ , then  $x = 0$  and  $y = 0$ , and that if  $x \wedge y = 1$ , then  $x = 1$  and  $y = 1$ .
42. Show that in a Boolean algebra, the **dual** of an identity, obtained by interchanging the  $\vee$  and  $\wedge$  operators and interchanging the elements 0 and 1, is also a valid identity.
43. Show that a complemented, distributive lattice is a Boolean algebra.

## 12.2 Representing Boolean Functions

Two important problems of Boolean algebra will be studied in this section. The first problem is: Given the values of a Boolean function, how can a Boolean expression that represents this function be found? This problem will be solved by showing that any Boolean function can be represented by a Boolean sum of Boolean products of the variables and their complements. The solution of this problem shows that every Boolean function can be represented using the three Boolean operators  $\cdot$ ,  $+$ , and  $\bar{\phantom{x}}$ . The second problem is: Is there a smaller set of operators that can be used to represent all Boolean functions? We will answer this question by showing that all Boolean functions can be represented using only one operator. Both of these problems have practical importance in circuit design.

### Sum-of-Products Expansions

We will use examples to illustrate one important way to find a Boolean expression that represents a Boolean function.

**EXAMPLE 1** Find Boolean expressions that represent the functions  $F(x, y, z)$  and  $G(x, y, z)$ , which are given in Table 1.

**Solution:** An expression that has the value 1 when  $x = z = 1$  and  $y = 0$ , and the value 0 otherwise, is needed to represent  $F$ . Such an expression can be formed by taking the Boolean product of  $x$ ,  $\bar{y}$ , and  $z$ . This product,  $x\bar{y}z$ , has the value 1 if and only if  $x = \bar{y} = z = 1$ , which holds if and only if  $x = z = 1$  and  $y = 0$ .

To represent  $G$ , we need an expression that equals 1 when  $x = y = 1$  and  $z = 0$ , or  $x = z = 0$  and  $y = 1$ . We can form an expression with these values by taking the Boolean sum of two different Boolean products. The Boolean product  $xy\bar{z}$  has the value 1 if and only if  $x = y = 1$  and  $z = 0$ . Similarly, the product  $\bar{x}y\bar{z}$  has the value 1 if and only if  $x = z = 0$  and  $y = 1$ . The Boolean sum of these two products,  $xy\bar{z} + \bar{x}y\bar{z}$ , represents  $G$ , because it has the value 1 if and only if  $x = y = 1$  and  $z = 0$ , or  $x = z = 0$  and  $y = 1$ . 

Example 1 illustrates a procedure for constructing a Boolean expression representing a function with given values. Each combination of values of the variables for which the function has the value 1 leads to a Boolean product of the variables or their complements.

| <b>TABLE 1</b>        |                       |                       |                       |                       |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| <b><math>x</math></b> | <b><math>y</math></b> | <b><math>z</math></b> | <b><math>F</math></b> | <b><math>G</math></b> |
| 1                     | 1                     | 1                     | 0                     | 0                     |
| 1                     | 1                     | 0                     | 0                     | 1                     |
| 1                     | 0                     | 1                     | 1                     | 0                     |
| 1                     | 0                     | 0                     | 0                     | 0                     |
| 0                     | 1                     | 1                     | 0                     | 0                     |
| 0                     | 1                     | 0                     | 0                     | 1                     |
| 0                     | 0                     | 1                     | 0                     | 0                     |
| 0                     | 0                     | 0                     | 0                     | 0                     |

**DEFINITION 1**

A *literal* is a Boolean variable or its complement. A *minterm* of the Boolean variables  $x_1, x_2, \dots, x_n$  is a Boolean product  $y_1 y_2 \dots y_n$ , where  $y_i = x_i$  or  $y_i = \bar{x}_i$ . Hence, a minterm is a product of  $n$  literals, with one literal for each variable.

A minterm has the value 1 for one and only one combination of values of its variables. More precisely, the minterm  $y_1 y_2 \dots y_n$  is 1 if and only if each  $y_i$  is 1, and this occurs if and only if  $x_i = 1$  when  $y_i = x_i$  and  $x_i = 0$  when  $y_i = \bar{x}_i$ .

**EXAMPLE 2** Find a minterm that equals 1 if  $x_1 = x_3 = 0$  and  $x_2 = x_4 = x_5 = 1$ , and equals 0 otherwise.

*Solution:* The minterm  $\bar{x}_1 x_2 \bar{x}_3 x_4 x_5$  has the correct set of values. 

By taking Boolean sums of distinct minterms we can build up a Boolean expression with a specified set of values. In particular, a Boolean sum of minterms has the value 1 when exactly one of the minterms in the sum has the value 1. It has the value 0 for all other combinations of values of the variables. Consequently, given a Boolean function, a Boolean sum of minterms can be formed that has the value 1 when this Boolean function has the value 1, and has the value 0 when the function has the value 0. The minterms in this Boolean sum correspond to those combinations of values for which the function has the value 1. The sum of minterms that represents the function is called the **sum-of-products expansion** or the **disjunctive normal form** of the Boolean function.



(See Exercise 42 in Section 1.3 for the development of disjunctive normal form in propositional calculus.)

**EXAMPLE 3** Find the sum-of-products expansion for the function  $F(x, y, z) = (x + y)\bar{z}$ .



*Solution:* We will find the sum-of-products expansion of  $F(x, y, z)$  in two ways. First, we will use Boolean identities to expand the product and simplify. We find that

$$\begin{aligned}
 F(x, y, z) &= (x + y)\bar{z} \\
 &= x\bar{z} + y\bar{z} && \text{Distributive law} \\
 &= x1\bar{z} + 1y\bar{z} && \text{Identity law} \\
 &= x(y + \bar{y})\bar{z} + (x + \bar{x})y\bar{z} && \text{Unit property} \\
 &= xy\bar{z} + x\bar{y}\bar{z} + xy\bar{z} + \bar{x}y\bar{z} && \text{Distributive law} \\
 &= xy\bar{z} + x\bar{y}\bar{z} + \bar{x}y\bar{z}. && \text{Idempotent law}
 \end{aligned}$$

Second, we can construct the sum-of-products expansion by determining the values of  $F$  for all possible values of the variables  $x$ ,  $y$ , and  $z$ . These values are found in Table 2. The sum-of-products expansion of  $F$  is the Boolean sum of three minterms corresponding to the three rows of this table that give the value 1 for the function. This gives

$$F(x, y, z) = xy\bar{z} + x\bar{y}\bar{z} + \bar{x}y\bar{z}.$$



It is also possible to find a Boolean expression that represents a Boolean function by taking a Boolean product of Boolean sums. The resulting expansion is called the **conjunctive normal form** or **product-of-sums expansion** of the function. These expansions can be found from sum-of-products expansions by taking duals. How to find such expansions directly is described in Exercise 10.

**TABLE 2**

| $x$ | $y$ | $z$ | $x + y$ | $\bar{z}$ | $(x + y)\bar{z}$ |
|-----|-----|-----|---------|-----------|------------------|
| 1   | 1   | 1   | 1       | 0         | 0                |
| 1   | 1   | 0   | 1       | 1         | 1                |
| 1   | 0   | 1   | 1       | 0         | 0                |
| 1   | 0   | 0   | 1       | 1         | 1                |
| 0   | 1   | 1   | 1       | 0         | 0                |
| 0   | 1   | 0   | 1       | 1         | 1                |
| 0   | 0   | 1   | 0       | 0         | 0                |
| 0   | 0   | 0   | 0       | 1         | 0                |

## Functional Completeness

Every Boolean function can be expressed as a Boolean sum of minterms. Each minterm is the Boolean product of Boolean variables or their complements. This shows that every Boolean function can be represented using the Boolean operators  $\cdot$ ,  $+$ , and  $\bar{\phantom{x}}$ . Because every Boolean function can be represented using these operators we say that the set  $\{\cdot, +, \bar{\phantom{x}}\}$  is **functionally complete**. Can we find a smaller set of functionally complete operators? We can do so if one of the three operators of this set can be expressed in terms of the other two. This can be done using one of De Morgan's laws. We can eliminate all Boolean sums using the identity

$$x + y = \bar{x}\bar{y},$$

which is obtained by taking complements of both sides in the second De Morgan law, given in Table 5 in Section 12.1, and then applying the double complementation law. This means that the set  $\{\cdot, \bar{\phantom{x}}\}$  is functionally complete. Similarly, we could eliminate all Boolean products using the identity

$$xy = \bar{\bar{x}} + \bar{\bar{y}},$$

which is obtained by taking complements of both sides in the first De Morgan law, given in Table 5 in Section 12.1, and then applying the double complementation law. Consequently  $\{+, \bar{\phantom{x}}\}$  is functionally complete. Note that the set  $\{+, \cdot\}$  is not functionally complete, because it is impossible to express the Boolean function  $F(x) = \bar{x}$  using these operators (see Exercise 19).

We have found sets containing two operators that are functionally complete. Can we find a smaller set of functionally complete operators, namely, a set containing just one operator? Such sets exist. Define two operators, the  $|$  or **NAND** operator, defined by  $1 | 1 = 0$  and  $1 | 0 = 0 | 1 = 0 | 0 = 1$ ; and the  $\downarrow$  or **NOR** operator, defined by  $1 \downarrow 1 = 1 \downarrow 0 = 0 \downarrow 1 = 0$  and  $0 \downarrow 0 = 1$ . Both of the sets  $\{| \}$  and  $\{\downarrow\}$  are functionally complete. To see that  $\{| \}$  is functionally complete, because  $\{\cdot, \bar{\phantom{x}}\}$  is functionally complete, all that we have to do is show that both of the operators  $\cdot$  and  $\bar{\phantom{x}}$  can be expressed using just the  $|$  operator. This can be done as

$$\begin{aligned} \bar{x} &= x | x, \\ xy &= (x | y) | (x | y). \end{aligned}$$

The reader should verify these identities (see Exercise 14). We leave the demonstration that  $\{\downarrow\}$  is functionally complete for the reader (see Exercises 15 and 16).

## Exercises

---

1. Find a Boolean product of the Boolean variables  $x$ ,  $y$ , and  $z$ , or their complements, that has the value 1 if and only if
    - a)  $x = y = 0, z = 1$ .
    - b)  $x = 0, y = 1, z = 0$ .
    - c)  $x = 0, y = z = 1$ .
    - d)  $x = y = z = 0$ .
  2. Find the sum-of-products expansions of these Boolean functions.
    - a)  $F(x, y) = \bar{x} + y$
    - b)  $F(x, y) = x\bar{y}$
    - c)  $F(x, y) = 1$
    - d)  $F(x, y) = \bar{y}$
  3. Find the sum-of-products expansions of these Boolean functions.
    - a)  $F(x, y, z) = x + y + z$
    - b)  $F(x, y, z) = (x + z)y$
    - c)  $F(x, y, z) = x$
    - d)  $F(x, y, z) = x\bar{y}$
  4. Find the sum-of-products expansions of the Boolean function  $F(x, y, z)$  that equals 1 if and only if
    - a)  $x = 0$ .
    - b)  $xy = 0$ .
    - c)  $x + y = 0$ .
    - d)  $xyz = 0$ .
  5. Find the sum-of-products expansion of the Boolean function  $F(w, x, y, z)$  that has the value 1 if and only if an odd number of  $w$ ,  $x$ ,  $y$ , and  $z$  have the value 1.
  6. Find the sum-of-products expansion of the Boolean function  $F(x_1, x_2, x_3, x_4, x_5)$  that has the value 1 if and only if three or more of the variables  $x_1, x_2, x_3, x_4$ , and  $x_5$  have the value 1.
- Another way to find a Boolean expression that represents a Boolean function is to form a Boolean product of Boolean sums of literals. Exercises 7–11 are concerned with representations of this kind.
7. Find a Boolean sum containing either  $x$  or  $\bar{x}$ , either  $y$  or  $\bar{y}$ , and either  $z$  or  $\bar{z}$  that has the value 0 if and only if
    - a)  $x = y = 1, z = 0$ .
    - b)  $x = y = z = 0$ .
    - c)  $x = z = 0, y = 1$ .
  8. Find a Boolean product of Boolean sums of literals that has the value 0 if and only if  $x = y = 1$  and  $z = 0$ ,  $x = z = 0$  and  $y = 1$ , or  $x = y = z = 0$ . [Hint: Take the Boolean product of the Boolean sums found in parts (a), (b), and (c) in Exercise 7.]
9. Show that the Boolean sum  $y_1 + y_2 + \dots + y_n$ , where  $y_i = x_i$  or  $y_i = \bar{x}_i$ , has the value 0 for exactly one combination of the values of the variables, namely, when  $x_i = 0$  if  $y_i = x_i$  and  $x_i = 1$  if  $y_i = \bar{x}_i$ . This Boolean sum is called a **maxterm**.
  10. Show that a Boolean function can be represented as a Boolean product of maxterms. This representation is called the **product-of-sums expansion** or **conjunctive normal form** of the function. [Hint: Include one maxterm in this product for each combination of the variables where the function has the value 0.]
  11. Find the product-of-sums expansion of each of the Boolean functions in Exercise 3.
  12. Express each of these Boolean functions using the operators  $\cdot$  and  $\bar{\phantom{x}}$ .
    - a)  $x + y + z$
    - b)  $x + \bar{y}(\bar{x} + z)$
    - c)  $\bar{x} + \bar{y}$
    - d)  $\bar{x}(x + \bar{y} + \bar{z})$
  13. Express each of the Boolean functions in Exercise 12 using the operators  $+$  and  $\bar{\phantom{x}}$ .
  14. Show that
    - a)  $\bar{x} = x \downarrow x$ .
    - b)  $xy = (x \mid y) \mid (x \mid y)$ .
    - c)  $x + y = (x \downarrow x) \downarrow (y \downarrow y)$ .
  15. Show that
    - a)  $\bar{x} = x \downarrow x$ .
    - b)  $xy = (x \downarrow x) \downarrow (y \downarrow y)$ .
    - c)  $x + y = (x \downarrow y) \downarrow (x \downarrow y)$ .
  16. Show that  $\{\downarrow\}$  is functionally complete using Exercise 15.
  17. Express each of the Boolean functions in Exercise 3 using the operator  $\mid$ .
  18. Express each of the Boolean functions in Exercise 3 using the operator  $\downarrow$ .
  19. Show that the set of operators  $\{+, \cdot\}$  is not functionally complete.
  20. Are these sets of operators functionally complete?
    - a)  $\{+, \oplus\}$
    - b)  $\{\bar{\phantom{x}}, \oplus\}$
    - c)  $\{\cdot, \oplus\}$

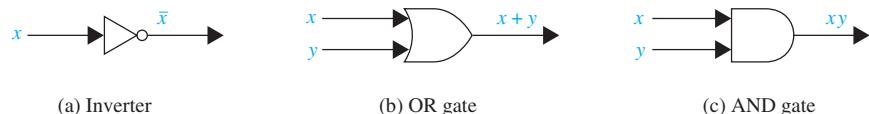
## 12.3 Logic Gates

---

### Introduction



Boolean algebra is used to model the circuitry of electronic devices. Each input and each output of such a device can be thought of as a member of the set  $\{0, 1\}$ . A computer, or other electronic device, is made up of a number of circuits. Each circuit can be designed using the rules of Boolean algebra that were studied in Sections 12.1 and 12.2. The basic elements of circuits



## **FIGURE 1 Basic Types of Gates.**

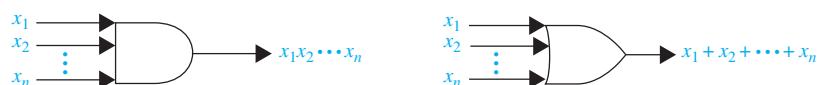
are called **gates**, and were introduced in Section 1.2. Each type of gate implements a Boolean operation. In this section we define several types of gates. Using these gates, we will apply the rules of Boolean algebra to design circuits that perform a variety of tasks. The circuits that we will study in this chapter give output that depends only on the input, and not on the current state of the circuit. In other words, these circuits have no memory capabilities. Such circuits are called **combinational circuits** or **gating networks**.

We will construct combinational circuits using three types of elements. The first is an **inverter**, which accepts the value of one Boolean variable as input and produces the complement of this value as its output. The symbol used for an inverter is shown in Figure 1(a). The input to the inverter is shown on the left side entering the element, and the output is shown on the right side leaving the element.

The next type of element we will use is the **OR gate**. The inputs to this gate are the values of two or more Boolean variables. The output is the Boolean sum of their values. The symbol used for an OR gate is shown in Figure 1(b). The inputs to the OR gate are shown on the left side entering the element, and the output is shown on the right side leaving the element.

The third type of element we will use is the **AND gate**. The inputs to this gate are the values of two or more Boolean variables. The output is the Boolean product of their values. The symbol used for an AND gate is shown in Figure 1(c). The inputs to the AND gate are shown on the left side entering the element, and the output is shown on the right side leaving the element.

We will permit multiple inputs to AND and OR gates. The inputs to each of these gates are shown on the left side entering the element, and the output is shown on the right side. Examples of AND and OR gates with  $n$  inputs are shown in Figure 2.



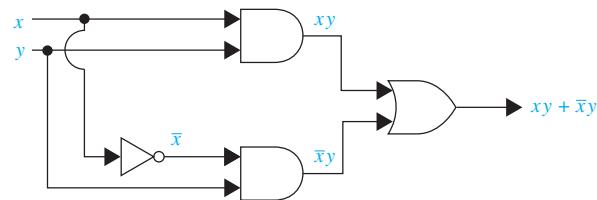
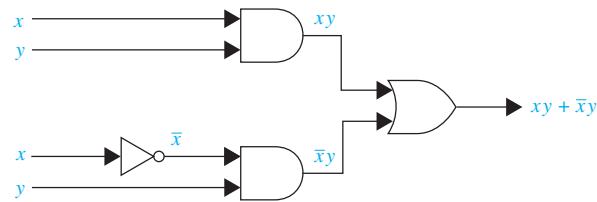
**FIGURE 2** Gates with  $n$  Inputs.

## Combinations of Gates

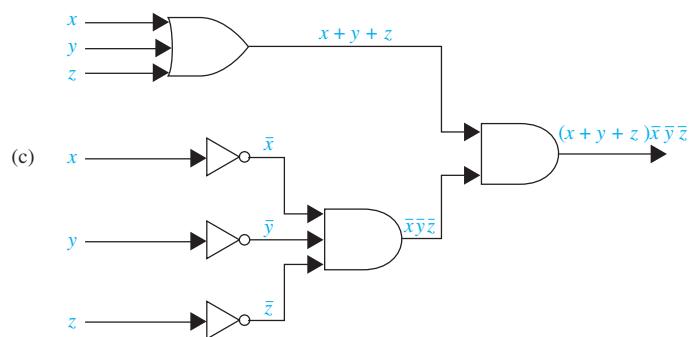
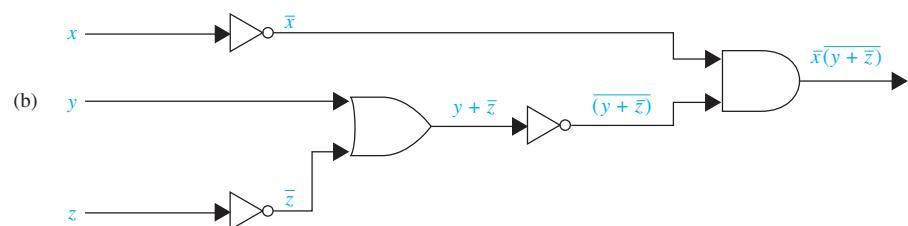
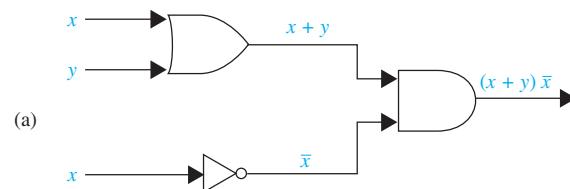
Combinational circuits can be constructed using a combination of inverters, OR gates, and AND gates. When combinations of circuits are formed, some gates may share inputs. This is shown in one of two ways in depictions of circuits. One method is to use branchings that indicate all the gates that use a given input. The other method is to indicate this input separately for each gate. Figure 3 illustrates the two ways of showing gates with the same input values. Note also that output from a gate may be used as input by one or more other elements, as shown in Figure 3. Both drawings in Figure 3 depict the circuit that produces the output  $xy + \bar{x}y$ .

**EXAMPLE 1** Construct circuits that produce the following outputs: (a)  $(x + y)\bar{x}$ , (b)  $\bar{x}\bar{(y + \bar{z})}$ , and (c)  $(x + y + z)(\bar{x}\bar{y}\bar{z})$ .

**Solution:** Circuits that produce these outputs are shown in Figure 4.



**FIGURE 3** Two Ways to Draw the Same Circuit.



**FIGURE 4** Circuits that Produce the Outputs Specified in Example 1.

## Examples of Circuits

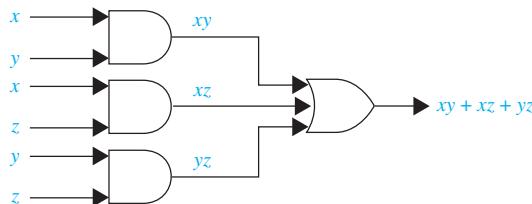
We will give some examples of circuits that perform some useful functions.

### EXAMPLE 2

A committee of three individuals decides issues for an organization. Each individual votes either yes or no for each proposal that arises. A proposal is passed if it receives at least two yes votes. Design a circuit that determines whether a proposal passes.



**Solution:** Let  $x = 1$  if the first individual votes yes, and  $x = 0$  if this individual votes no; let  $y = 1$  if the second individual votes yes, and  $y = 0$  if this individual votes no; let  $z = 1$  if the third individual votes yes, and  $z = 0$  if this individual votes no. Then a circuit must be designed that produces the output 1 from the inputs  $x$ ,  $y$ , and  $z$  when two or more of  $x$ ,  $y$ , and  $z$  are 1. One representation of the Boolean function that has these output values is  $xy + xz + yz$  (see Exercise 12 in Section 12.1). The circuit that implements this function is shown in Figure 5.



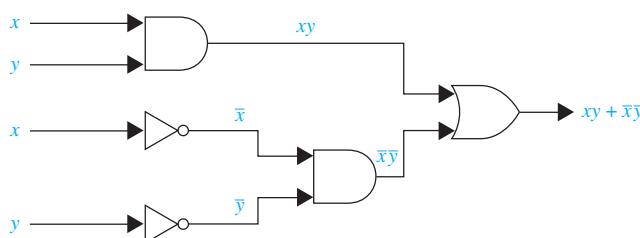
**FIGURE 5** A Circuit for Majority Voting.

### EXAMPLE 3

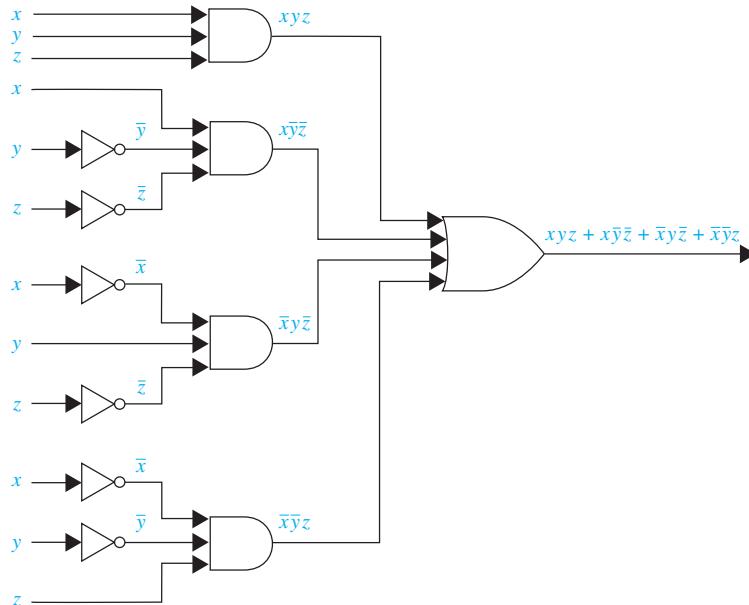
Sometimes light fixtures are controlled by more than one switch. Circuits need to be designed so that flipping any one of the switches for the fixture turns the light on when it is off and turns the light off when it is on. Design circuits that accomplish this when there are two switches and when there are three switches.

| TABLE 1 |     |           |
|---------|-----|-----------|
| $x$     | $y$ | $F(x, y)$ |
| 1       | 1   | 1         |
| 1       | 0   | 0         |
| 0       | 1   | 0         |
| 0       | 0   | 1         |

**Solution:** We will begin by designing the circuit that controls the light fixture when two different switches are used. Let  $x = 1$  when the first switch is closed and  $x = 0$  when it is open, and let  $y = 1$  when the second switch is closed and  $y = 0$  when it is open. Let  $F(x, y) = 1$  when the light is on and  $F(x, y) = 0$  when it is off. We can arbitrarily decide that the light will be on when both switches are closed, so that  $F(1, 1) = 1$ . This determines all the other values of  $F$ . When one of the two switches is opened, the light goes off, so  $F(1, 0) = F(0, 1) = 0$ . When the other switch is also opened, the light goes on, so  $F(0, 0) = 1$ . Table 1 displays these values. Note that  $F(x, y) = xy + \bar{x}\bar{y}$ . This function is implemented by the circuit shown in Figure 6.



**FIGURE 6** A Circuit for a Light Controlled by Two Switches.

**FIGURE 7** A Circuit for a Fixture Controlled by Three Switches.

| <b>TABLE 2</b> |   |   |            |
|----------------|---|---|------------|
| x              | y | z | F(x, y, z) |
| 1              | 1 | 1 | 1          |
| 1              | 1 | 0 | 0          |
| 1              | 0 | 1 | 0          |
| 1              | 0 | 0 | 1          |
| 0              | 1 | 1 | 0          |
| 0              | 1 | 0 | 1          |
| 0              | 0 | 1 | 1          |
| 0              | 0 | 0 | 0          |

We will now design a circuit for three switches. Let  $x$ ,  $y$ , and  $z$  be the Boolean variables that indicate whether each of the three switches is closed. We let  $x = 1$  when the first switch is closed, and  $x = 0$  when it is open;  $y = 1$  when the second switch is closed, and  $y = 0$  when it is open; and  $z = 1$  when the third switch is closed, and  $z = 0$  when it is open. Let  $F(x, y, z) = 1$  when the light is on and  $F(x, y, z) = 0$  when the light is off. We can arbitrarily specify that the light be on when all three switches are closed, so that  $F(1, 1, 1) = 1$ . This determines all other values of  $F$ . When one switch is opened, the light goes off, so  $F(1, 1, 0) = F(1, 0, 1) = F(0, 1, 1) = 0$ . When a second switch is opened, the light goes on, so  $F(1, 0, 0) = F(0, 1, 0) = F(0, 0, 1) = 1$ . Finally, when the third switch is opened, the light goes off again, so  $F(0, 0, 0) = 0$ . Table 2 shows the values of this function.

The function  $F$  can be represented by its sum-of-products expansion as  $F(x, y, z) = xyz + x\bar{y}\bar{z} + \bar{x}y\bar{z} + \bar{x}\bar{y}z$ . The circuit shown in Figure 7 implements this function.  $\blacktriangleleft$

## Adders



| <b>TABLE 3</b><br>Input and Output for the Half Adder. |   |        |   |
|--------------------------------------------------------|---|--------|---|
| Input                                                  |   | Output |   |
| x                                                      | y | s      | c |
| 1                                                      | 1 | 0      | 1 |
| 1                                                      | 0 | 1      | 0 |
| 0                                                      | 1 | 1      | 0 |
| 0                                                      | 0 | 0      | 0 |

We will illustrate how logic circuits can be used to carry out addition of two positive integers from their binary expansions. We will build up the circuitry to do this addition from some component circuits. First, we will build a circuit that can be used to find  $x + y$ , where  $x$  and  $y$  are two bits. The input to our circuit will be  $x$  and  $y$ , because these each have the value 0 or the value 1. The output will consist of two bits, namely,  $s$  and  $c$ , where  $s$  is the sum bit and  $c$  is the carry bit. This circuit is called a **multiple output circuit** because it has more than one output. The circuit that we are designing is called the **half adder**, because it adds two bits, without considering a carry from a previous addition. We show the input and output for the half adder in Table 3. From Table 3 we see that  $c = xy$  and that  $s = x\bar{y} + \bar{x}y = (x + y)\overline{(xy)}$ . Hence, the circuit shown in Figure 8 computes the sum bit  $s$  and the carry bit  $c$  from the bits  $x$  and  $y$ .

We use the **full adder** to compute the sum bit and the carry bit when two bits and a carry are added. The inputs to the full adder are the bits  $x$  and  $y$  and the carry  $c_i$ . The outputs are the sum bit  $s$  and the new carry  $c_{i+1}$ . The inputs and outputs for the full adder are shown in Table 4.

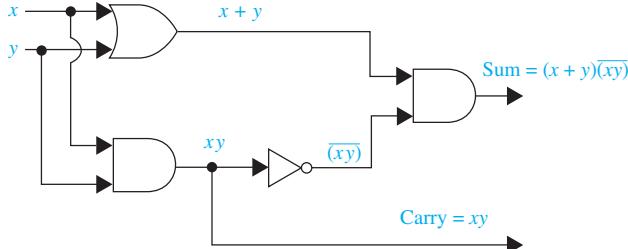


FIGURE 8 The Half Adder.

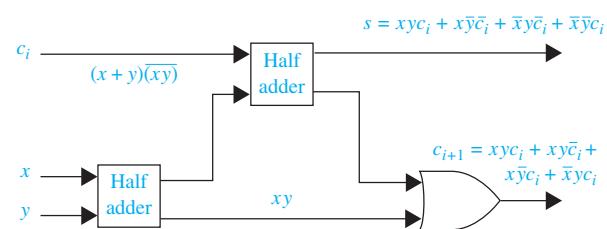


FIGURE 9 A Full Adder.

**TABLE 4**  
Input and Output for the Full Adder.

| Input |   |    | Output |      |
|-------|---|----|--------|------|
| x     | y | ci | s      | ci+1 |
| 1     | 1 | 1  | 1      | 1    |
| 1     | 1 | 0  | 0      | 1    |
| 1     | 0 | 1  | 0      | 1    |
| 1     | 0 | 0  | 1      | 0    |
| 0     | 1 | 1  | 0      | 1    |
| 0     | 1 | 0  | 1      | 0    |
| 0     | 0 | 1  | 1      | 0    |
| 0     | 0 | 0  | 0      | 0    |

The two outputs of the full adder, the sum bit  $s$  and the carry  $c_{i+1}$ , are given by the sum-of-products expansions  $xyc_i + x\bar{y}c_i + \bar{x}yc_i + \bar{x}\bar{y}c_i$  and  $xyc_i + xy\bar{c}_i + x\bar{y}c_i + \bar{x}yc_i$ , respectively. However, instead of designing the full adder from scratch, we will use half adders to produce the desired output. A full adder circuit using half adders is shown in Figure 9.

Finally, in Figure 10 we show how full and half adders can be used to add the two three-bit integers  $(x_2x_1x_0)_2$  and  $(y_2y_1y_0)_2$  to produce the sum  $(s_3s_2s_1s_0)_2$ . Note that  $s_3$ , the highest-order bit in the sum, is given by the carry  $c_2$ .

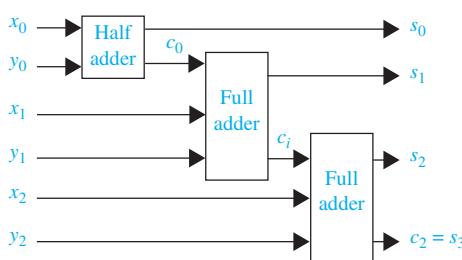
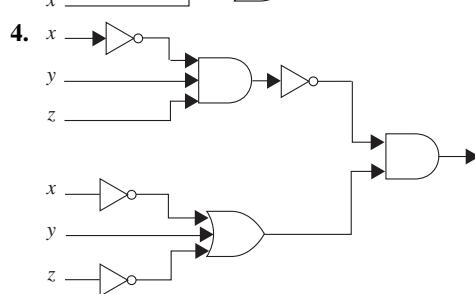
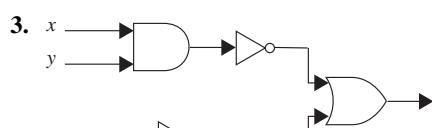
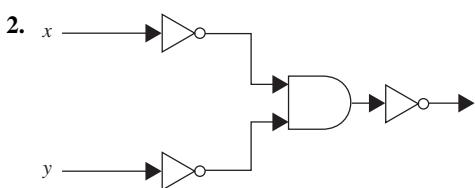
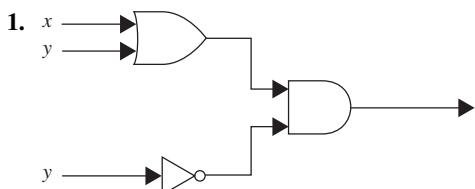
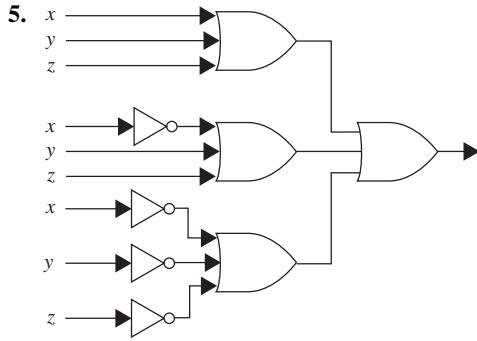


FIGURE 10 Adding Two Three-Bit Integers with Full and Half Adders.

## Exercises

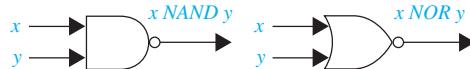
In Exercises 1–5 find the output of the given circuit.





6. Construct circuits from inverters, AND gates, and OR gates to produce these outputs.
- $\bar{x} + y$
  - $\overline{(x+y)x}$
  - $xyz + \bar{x}\bar{y}\bar{z}$
  - $\overline{(\bar{x}+z)(y+\bar{z})}$
7. Design a circuit that implements majority voting for five individuals.
8. Design a circuit for a light fixture controlled by four switches, where flipping one of the switches turns the light on when it is off and turns it off when it is on.
9. Show how the sum of two five-bit integers can be found using full and half adders.
10. Construct a circuit for a half subtractor using AND gates, OR gates, and inverters. A **half subtractor** has two bits as input and produces as output a difference bit and a borrow.
11. Construct a circuit for a full subtractor using AND gates, OR gates, and inverters. A **full subtractor** has two bits and a borrow as input, and produces as output a difference bit and a borrow.
12. Use the circuits from Exercises 10 and 11 to find the difference of two four-bit integers, where the first integer is greater than the second integer.
- \*13. Construct a circuit that compares the two-bit integers  $(x_1x_0)_2$  and  $(y_1y_0)_2$ , returning an output of 1 when the first of these numbers is larger and an output of 0 otherwise.
- \*14. Construct a circuit that computes the product of the two-bit integers  $(x_1x_0)_2$  and  $(y_1y_0)_2$ . The circuit should have four output bits for the bits in the product.

Two gates that are often used in circuits are NAND and NOR gates. When NAND or NOR gates are used to represent circuits, no other types of gates are needed. The notation for these gates is as follows:



- \*15. Use NAND gates to construct circuits with these outputs.

- $\bar{x}$
- $x + y$
- $xy$
- $x \oplus y$

- \*16. Use NOR gates to construct circuits for the outputs given in Exercise 15.

- \*17. Construct a half adder using NAND gates.

- \*18. Construct a half adder using NOR gates.

A **multiplexer** is a switching circuit that produces as output one of a set of input bits based on the value of control bits.

19. Construct a multiplexer using AND gates, OR gates, and inverters that has as input the four bits  $x_0, x_1, x_2$ , and  $x_3$  and the two control bits  $c_0$  and  $c_1$ . Set up the circuit so that  $x_i$  is the output, where  $i$  is the value of the two-bit integer  $(c_1c_0)_2$ .

The **depth** of a combinatorial circuit can be defined by specifying that the depth of the initial input is 0 and if a gate has  $n$  different inputs at depths  $d_1, d_2, \dots, d_n$ , respectively, then its outputs have depth equal to  $\max(d_1, d_2, \dots, d_n) + 1$ ; this value is also defined to be the depth of the gate. The depth of a combinatorial circuit is the maximum depth of the gates in the circuit.

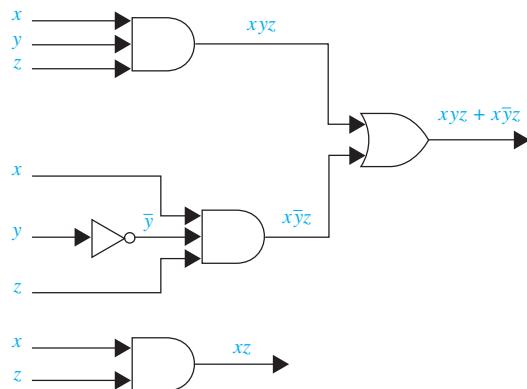
20. Find the depth of

- the circuit constructed in Example 2 for majority voting among three people.
- the circuit constructed in Example 3 for a light controlled by two switches.
- the half adder shown in Figure 8.
- the full adder shown in Figure 9.

## 12.4 Minimization of Circuits

### Introduction

The efficiency of a combinational circuit depends on the number and arrangement of its gates. The process of designing a combinational circuit begins with the table specifying the output for each combination of input values. We can always use the sum-of-products expansion of a circuit to find a set of logic gates that will implement this circuit. However, the sum-of-products expansion



**FIGURE 1** Two Circuits with the Same Output.

may contain many more terms than are necessary. Terms in a sum-of-products expansion that differ in just one variable, so that in one term this variable occurs and in the other term the complement of this variable occurs, can be combined. For instance, consider the circuit that has output 1 if and only if  $x = y = z = 1$  or  $x = z = 1$  and  $y = 0$ . The sum-of-products expansion of this circuit is  $xyz + x\bar{y}z$ . The two products in this expansion differ in exactly one variable, namely,  $y$ . They can be combined as

$$\begin{aligned} xyz + x\bar{y}z &= (y + \bar{y})(xz) \\ &= 1 \cdot (xz) \\ &= xz. \end{aligned}$$

Hence,  $xz$  is a Boolean expression with fewer operators that represents the circuit. We show two different implementations of this circuit in Figure 1. The second circuit uses only one gate, whereas the first circuit uses three gates and an inverter.

This example shows that combining terms in the sum-of-products expansion of a circuit leads to a simpler expression for the circuit. We will describe two procedures that simplify sum-of-products expansions.

The goal of both procedures is to produce Boolean sums of Boolean products that represent a Boolean function with the fewest products of literals such that these products contain the fewest literals possible among all sums of products that represent a Boolean function. Finding such a sum of products is called **minimization of the Boolean function**. Minimizing a Boolean function makes it possible to construct a circuit for this function that uses the fewest gates and fewest inputs to the *AND* gates and *OR* gates in the circuit, among all circuits for the Boolean expression we are minimizing.

Until the early 1960s logic gates were individual components. To reduce costs it was important to use the fewest gates to produce a desired output. However, in the mid-1960s, integrated circuit technology was developed that made it possible to combine gates on a single chip. Even though it is now possible to build increasingly complex integrated circuits on chips at low cost, minimization of Boolean functions remains important.

Reducing the number of gates on a chip can lead to a more reliable circuit and can reduce the cost to produce the chip. Also, minimization makes it possible to fit more circuits on the same chip. Furthermore, minimization reduces the number of inputs to gates in a circuit. This reduces the time used by a circuit to compute its output. Moreover, the number of inputs to a gate may be limited because of the particular technology used to build logic gates.

The first procedure we will introduce, known as Karnaugh maps (or K-maps), was designed in the 1950s to help minimize circuits by hand. K-maps are useful in minimizing circuits with up to six variables, although they become rather complex even for five or six variables. The

second procedure we will describe, the Quine–McCluskey method, was invented in the 1960s. It automates the process of minimizing combinatorial circuits and can be implemented as a computer program.

**COMPLEXITY OF BOOLEAN FUNCTION MINIMIZATION** Unfortunately, minimizing Boolean functions with many variables is a computationally intensive problem. It has been shown that this problem is an NP-complete problem (see Section 3.3 and [Ka93]), so the existence of a polynomial-time algorithm for minimizing Boolean circuits is unlikely. The Quine–McCluskey method has exponential complexity. In practice, it can be used only when the number of literals does not exceed ten. Since the 1970s a number of newer algorithms have been developed for minimizing combinatorial circuits (see [Ha93] and [KaBe04]). However, with the best algorithms yet devised, only circuits with no more than 25 variables can be minimized. Also, heuristic (or rule-of-thumb) methods can be used to substantially simplify, but not necessarily minimize, Boolean expressions with a larger number of literals.

## Karnaugh Maps



To reduce the number of terms in a Boolean expression representing a circuit, it is necessary to find terms to combine. There is a graphical method, called a **Karnaugh map** or **K-map**, for finding terms to combine for Boolean functions involving a relatively small number of variables. The method we will describe was introduced by Maurice Karnaugh in 1953. His method is based on earlier work by E. W. Veitch. (This method is usually applied only when the function involves six or fewer variables.) K-maps give us a visual method for simplifying sum-of-products expansions; they are not suited for mechanizing this process. We will first illustrate how K-maps are used to simplify expansions of Boolean functions in two variables. We will continue by showing how K-maps can be used to minimize Boolean functions in three variables and then in four variables. Then we will describe the concepts that can be used to extend K-maps to minimize Boolean functions in more than four variables.

|           | $y$        | $\bar{y}$        |
|-----------|------------|------------------|
| $x$       | $xy$       | $x\bar{y}$       |
| $\bar{x}$ | $\bar{x}y$ | $\bar{x}\bar{y}$ |

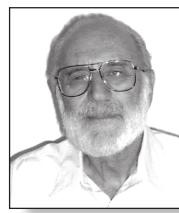
**FIGURE 2**  
K-maps in Two Variables.

There are four possible minterms in the sum-of-products expansion of a Boolean function in the two variables  $x$  and  $y$ . A K-map for a Boolean function in these two variables consists of four cells, where a 1 is placed in the cell representing a minterm if this minterm is present in the expansion. Cells are said to be **adjacent** if the minterms that they represent differ in exactly one literal. For instance, the cell representing  $\bar{x}y$  is adjacent to the cells representing  $xy$  and  $\bar{x}\bar{y}$ . The four cells and the terms that they represent are shown in Figure 2.

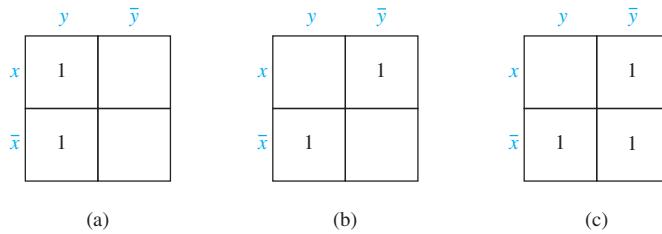
**EXAMPLE 1** Find the K-maps for (a)  $xy + \bar{x}y$ , (b)  $x\bar{y} + \bar{x}y$ , and (c)  $x\bar{y} + \bar{x}y + \bar{x}\bar{y}$ .

**Solution:** We include a 1 in a cell when the minterm represented by this cell is present in the sum-of-products expansion. The three K-maps are shown in Figure 3.

We can identify minterms that can be combined from the K-map. Whenever there are 1s in two adjacent cells in the K-map, the minterms represented by these cells can be combined into a product involving just one of the variables. For instance,  $x\bar{y}$  and  $\bar{x}\bar{y}$  are represented by adjacent cells and can be combined into  $\bar{y}$ , because  $x\bar{y} + \bar{x}\bar{y} = (x + \bar{x})\bar{y} = \bar{y}$ . Moreover, if 1s



**MAURICE KARNAUGH (BORN 1924)** Maurice Karnaugh, born in New York City, received his B.S. from the City College of New York and his M.S. and Ph.D. from Yale University. He was a member of the technical staff at Bell Laboratories from 1952 until 1966 and Manager of Research and Development at the Federal Systems Division of AT&T from 1966 to 1970. In 1970 he joined IBM as a member of the research staff. Karnaugh has made fundamental contributions to the application of digital techniques in both computing and telecommunications. His current interests include knowledge-based systems in computers and heuristic search methods.

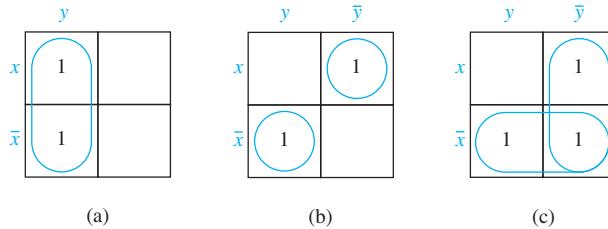


**FIGURE 3** K-maps for the Sum-of-Products Expansions in Example 1.

are in all four cells, the four minterms can be combined into one term, namely, the Boolean expression 1 that involves none of the variables. We circle blocks of cells in the K-map that represent minterms that can be combined and then find the corresponding sum of products. The goal is to identify the largest possible blocks, and to cover all the 1s with the fewest blocks using the largest blocks first and always using the largest possible blocks.

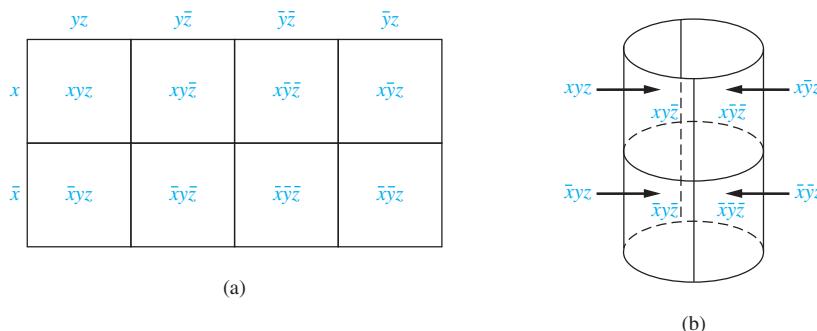
**EXAMPLE 2** Simplify the sum-of-products expansions given in Example 1.

**Solution:** The grouping of minterms is shown in Figure 4 using the K-maps for these expansions. Minimal expansions for these sums-of-products are (a)  $y$ , (b)  $x\bar{y} + \bar{x}y$ , and (c)  $\bar{x} + \bar{y}$ . ◀



**FIGURE 4** Simplifying the Sum-of-Products Expansions from Example 2.

A K-map in three variables is a rectangle divided into eight cells. The cells represent the eight possible minterms in three variables. Two cells are said to be adjacent if the minterms that they represent differ in exactly one literal. One of the ways to form a K-map in three variables is shown in Figure 5(a). This K-map can be thought of as lying on a cylinder, as shown in Figure 5(b). On the cylinder, two cells have a common border if and only if they are adjacent.



**FIGURE 5** K-maps in Three Variables.

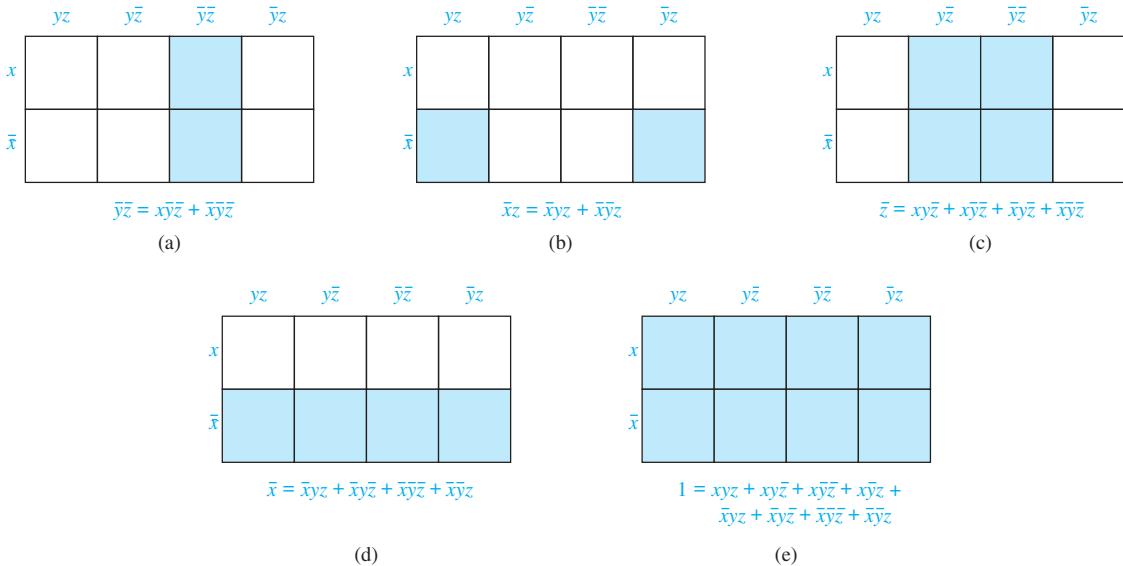


FIGURE 6 Blocks in K-maps in Three Variables.

To simplify a sum-of-products expansion in three variables, we use the K-map to identify blocks of minterms that can be combined. Blocks of two adjacent cells represent pairs of minterms that can be combined into a product of two literals;  $2 \times 2$  and  $4 \times 1$  blocks of cells represent minterms that can be combined into a single literal; and the block of all eight cells represents a product of no literals, namely, the function 1. In Figure 6,  $1 \times 2$ ,  $2 \times 1$ ,  $2 \times 2$ ,  $4 \times 1$ , and  $4 \times 2$  blocks and the products they represent are shown.

The product of literals corresponding to a block of all 1s in the K-map is called an **implicant** of the function being minimized. It is called a **prime implicant** if this block of 1s is not contained in a larger block of 1s representing the product of fewer literals than in this product.

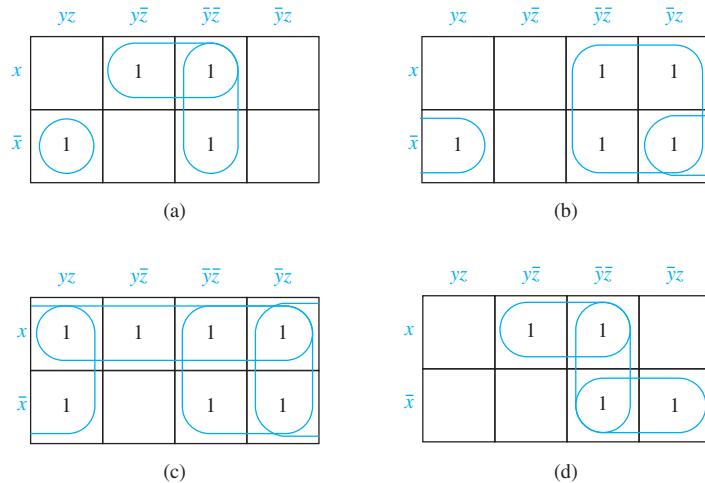
The goal is to identify the largest possible blocks in the map and cover all the 1s in the map with the least number of blocks, using the largest blocks first. The largest possible blocks are always chosen, but we must always choose a block if it is the only block of 1s covering a 1 in the K-map. Such a block represents an **essential prime implicant**. By covering all the 1s in the map with blocks corresponding to prime implicants we can express the sum of products as a sum of prime implicants. Note that there may be more than one way to cover all the 1s using the least number of blocks.

Example 3 illustrates how K-maps in three variables are used.

**EXAMPLE 3** Use K-maps to minimize these sum-of-products expansions.

- $xy\bar{z} + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}\bar{z}$
- $x\bar{y}z + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z}$
- $xyz + xy\bar{z} + x\bar{y}z + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z}$
- $xy\bar{z} + x\bar{y}\bar{z} + \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z}$

**Solution:** The K-maps for these sum-of-products expansions are shown in Figure 7. The grouping of blocks shows that minimal expansions into Boolean sums of Boolean products are (a)  $x\bar{z} + \bar{y}\bar{z} + \bar{x}yz$ , (b)  $\bar{y} + \bar{x}z$ , (c)  $x + \bar{y} + z$ , and (d)  $x\bar{z} + \bar{x}\bar{y}$ . In part (d) note that the prime implicants  $x\bar{z}$  and  $\bar{x}\bar{y}$  are essential prime implicants, but the prime implicant  $\bar{y}\bar{z}$  is a prime implicant that is not essential, because the cells it covers are covered by the other two prime implicants. 



## FIGURE 7 Using K-maps in Three Variables.

A K-map in four variables is a square that is divided into 16 cells. The cells represent the 16 possible minterms in four variables. One of the ways to form a K-map in four variables is shown in Figure 8.

Two cells are adjacent if and only if the minterms they represent differ in one literal. Consequently, each cell is adjacent to four other cells. The K-map of a sum-of-products expansion in four variables can be thought of as lying on a torus, so that adjacent cells have a common boundary (see Exercise 28). The simplification of a sum-of-products expansion in four variables is carried out by identifying those blocks of 2, 4, 8, or 16 cells that represent minterms that can be combined. Each cell representing a minterm must either be used to form a product using fewer literals, or be included in the expansion. In Figure 9 some examples of blocks that represent products of three literals, products of two literals, and a single literal are illustrated.

As is the case in K-maps in two and three variables, the goal is to identify the largest blocks of 1s in the map that correspond to the prime implicants and to cover all the 1s using the fewest blocks needed, using the largest blocks first. The largest possible blocks are always used. Example 4 illustrates how K-maps in four variables are used.

|                  | $yz$               | $y\bar{z}$               | $\bar{y}z$               | $\bar{y}\bar{z}$               |
|------------------|--------------------|--------------------------|--------------------------|--------------------------------|
| $wx$             | $wxyz$             | $wxy\bar{z}$             | $wx\bar{y}z$             | $wx\bar{y}\bar{z}$             |
| $w\bar{x}$       | $w\bar{x}yz$       | $w\bar{x}y\bar{z}$       | $w\bar{x}\bar{y}z$       | $w\bar{x}\bar{y}\bar{z}$       |
| $\bar{w}\bar{x}$ | $\bar{w}\bar{x}yz$ | $\bar{w}\bar{x}y\bar{z}$ | $\bar{w}\bar{x}\bar{y}z$ | $\bar{w}\bar{x}\bar{y}\bar{z}$ |
| $\bar{w}x$       | $\bar{w}xyz$       | $\bar{w}xy\bar{z}$       | $\bar{w}x\bar{y}z$       | $\bar{w}x\bar{y}\bar{z}$       |

**FIGURE 8 K-maps in Four Variables.**

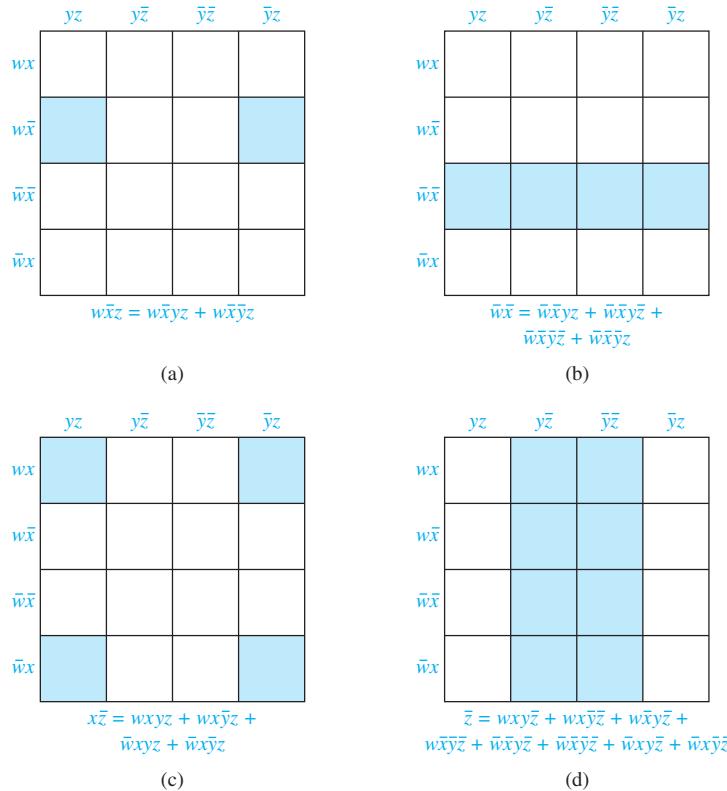


FIGURE 9 Blocks in K-maps in Four Variables.

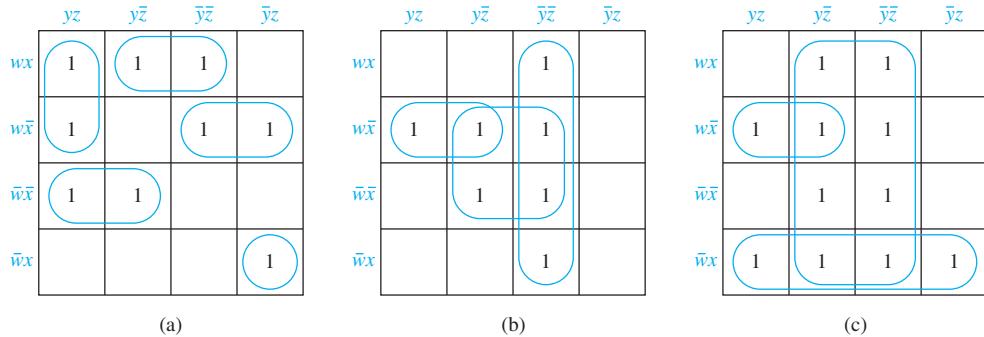
**EXAMPLE 4** Use K-maps to simplify these sum-of-products expansions.

- $wxyz + wxy\bar{z} + wx\bar{y}\bar{z} + w\bar{x}yz + w\bar{x}\bar{y}z + w\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}yz + \bar{w}\bar{x}\bar{y}z$
- $wx\bar{y}\bar{z} + w\bar{x}yz + w\bar{x}y\bar{z} + w\bar{x}\bar{y}\bar{z} + \bar{w}xy\bar{z} + \bar{w}\bar{x}\bar{y}\bar{z}$
- $wxy\bar{z} + w\bar{x}\bar{y}\bar{z} + w\bar{x}yz + w\bar{x}\bar{y}z + w\bar{x}\bar{y}\bar{z} + \bar{w}xyz + \bar{w}xy\bar{z} + \bar{w}x\bar{y}\bar{z} + \bar{w}\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}y\bar{z} + \bar{w}\bar{x}\bar{y}\bar{z}$

**Solution:** The K-maps for these expansions are shown in Figure 10. Using the blocks shown leads to the sum of products (a)  $wyz + wx\bar{z} + w\bar{x}\bar{y} + \bar{w}\bar{x}y + \bar{w}\bar{x}\bar{y}z$ , (b)  $\bar{y}\bar{z} + w\bar{x}y + \bar{x}\bar{z}$ , and (c)  $\bar{z} + \bar{w}x + w\bar{x}y$ . The reader should determine whether there are other choices of blocks in each part that lead to different sums of products representing these Boolean functions. 

K-maps can realistically be used to minimize Boolean functions with five or six variables, but beyond that, they are rarely used because they become extremely complicated. However, the concepts used in K-maps play an important role in newer algorithms. Furthermore, mastering these concepts helps you understand these newer algorithms and the computer-aided design (CAD) programs that implement them. As we develop these concepts, we will be able to illustrate them by referring back to our discussion of minimization of Boolean functions in three and in four variables.

The K-maps we used to minimize Boolean functions in two, three, and four variables are built using  $2 \times 2$ ,  $2 \times 4$ , and  $4 \times 4$  rectangles, respectively. Furthermore, corresponding cells in the top row and bottom row and in the leftmost column and rightmost column in each of these

**FIGURE 10** Using K-maps in Four Variables.

cases are considered adjacent because they represent minterms differing in only one literal. We can build K-maps for minimizing Boolean functions in more than four variables in a similar way. We use a rectangle containing  $2^{\lceil n/2 \rceil}$  rows and  $2^{\lceil n/2 \rceil}$  columns. (These K-maps contain  $2^n$  cells because  $\lceil n/2 \rceil + \lfloor n/2 \rfloor = n$ .) The rows and columns need to be positioned so that the cells representing minterms differing in just one literal are adjacent or are considered adjacent by specifying additional adjacencies of rows and columns. To help (but not entirely) achieve this, the rows and columns of a K-map are arranged using Gray codes (see Section 10.5), where we associate bit strings and products by specifying that a 1 corresponds to the appearance of a variable and a 0 with the appearance of its complement. For example, in a 10-dimensional K-map, the Gray code 01110 used to label a row corresponds to the product  $\bar{x}_1x_2x_3x_4\bar{x}_5$ .

**EXAMPLE 5** The K-maps we used to minimize Boolean functions with four variables have four rows and four columns. Both the rows and the columns are arranged using the Gray code 11,10,00,01. The rows represent products  $wx$ ,  $w\bar{x}$ ,  $\bar{w}\bar{x}$ , and  $\bar{w}x$ , respectively, and the columns correspond to the products  $yz$ ,  $y\bar{z}$ ,  $\bar{y}\bar{z}$ , and  $\bar{y}z$ , respectively. Using Gray codes and considering cells adjacent in the first and last rows and in the first and last columns, we ensured that minterms that differ in only one variable are always adjacent.  $\blacktriangleleft$

**EXAMPLE 6** To minimize Boolean functions in five variables we use K-maps with  $2^3 = 8$  columns and  $2^2 = 4$  rows. We label the four rows using the Gray code 11,10,00,01, corresponding to  $x_1x_2$ ,  $x_1\bar{x}_2$ ,  $\bar{x}_1\bar{x}_2$ , and  $\bar{x}_1x_2$ , respectively. We label the eight columns using the Gray code 111,110,100,101,001,000,010,011 corresponding to the terms  $x_3x_4x_5$ ,  $x_3x_4\bar{x}_5$ ,  $x_3\bar{x}_4x_5$ ,  $\bar{x}_3x_4x_5$ ,  $\bar{x}_3\bar{x}_4\bar{x}_5$ ,  $\bar{x}_3x_4\bar{x}_5$ , and  $\bar{x}_3\bar{x}_4x_5$ , respectively. Using Gray codes to label columns and rows ensures that the minterms represented by adjacent cells differ in only one variable. However, to make sure all cells representing products that differ in only one variable are considered adjacent, we consider cells in the top and bottom rows to be adjacent, as well as cells in the first and eighth columns, the first and fourth columns, the second and seventh columns, the third and sixth columns, and the fifth and eighth columns (as the reader should verify).  $\blacktriangleleft$

To use a K-map to minimize a Boolean function in  $n$  variables, we first draw a K-map of the appropriate size. We place 1s in all cells corresponding to minterms in the sum-of-products expansion of this function. We then identify all prime implicants of the Boolean function. To do this we look for the blocks consisting of  $2^k$  clustered cells all containing a 1, where  $1 \leq k \leq n$ . These blocks correspond to the product of  $n - k$  literals. (Exercise 33 asks the reader to verify this.) Furthermore, a block of  $2^k$  cells each containing a 1 not contained in a block of  $2^{k+1}$  cells each containing a 1 represents a prime implicant. The reason that this implicant is a prime implicant is that no product obtained by deleting a literal is also represented by a block of cells all containing 1s.

**EXAMPLE 7** A block of eight cells representing a product of two literals in a K-map for minimizing Boolean functions in five variables all containing 1s is a prime implicant if it is not contained in a larger block of 16 cells all containing 1s representing a single literal. 

Once all prime implicants have been identified, the goal is to find the smallest possible subset of these prime implicants with the property that the cells representing these prime implicants cover all the cells containing a 1 in the K-map. We begin by selecting the essential prime implicants because each of these is represented by a block that covers a cell containing a 1 that is not covered by any other prime implicant. We add additional prime implicants to ensure that all 1s in the K-map are covered. When the number of variables is large, this last step can become exceedingly complicated.

## Don't Care Conditions



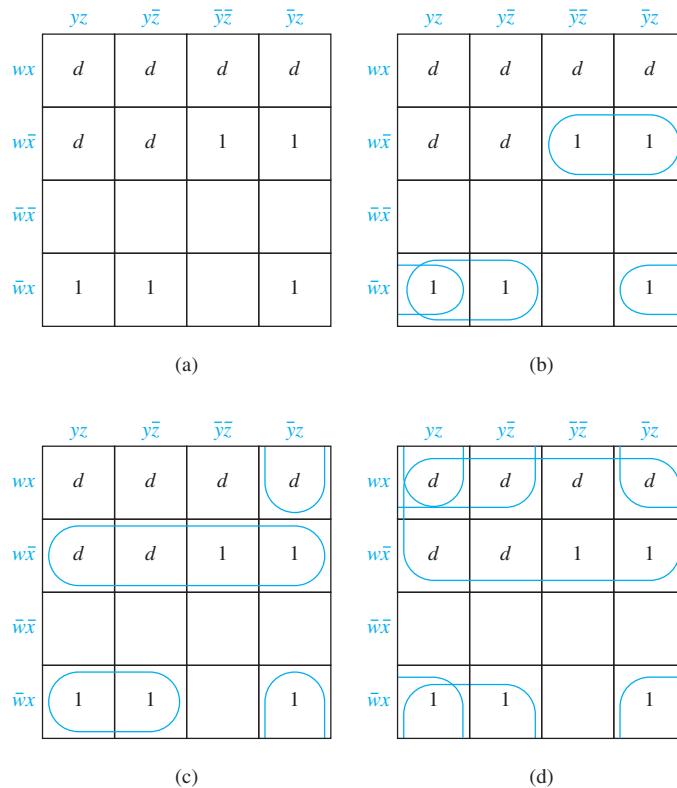
In some circuits we care only about the output for some combinations of input values, because other combinations of input values are not possible or never occur. This gives us freedom in producing a simple circuit with the desired output because the output values for all those combinations that never occur can be arbitrarily chosen. The values of the function for these combinations are called **don't care conditions**. A *d* is used in a K-map to mark those combinations of values of the variables for which the function can be arbitrarily assigned. In the minimization process we can assign 1s as values to those combinations of the input values that lead to the largest blocks in the K-map. This is illustrated in Example 8.

**EXAMPLE 8** One way to code decimal expansions using bits is to use the four bits of the binary expansion of each digit in the decimal expansion. For instance, 873 is encoded as 100001110011. This encoding of a decimal expansion is called a **binary coded decimal expansion**. Because there are 16 blocks of four bits and only 10 decimal digits, there are six combinations of four bits that are not used to encode digits. Suppose that a circuit is to be built that produces an output of 1 if the decimal digit is 5 or greater and an output of 0 if the decimal digit is less than 5. How can this circuit be simply built using OR gates, AND gates, and inverters?

**Solution:** Let  $F(w, x, y, z)$  denote the output of the circuit, where  $wxyz$  is a binary expansion of a decimal digit. The values of  $F$  are shown in Table 1. The K-map for  $F$ , with *ds* in the *don't care* positions, is shown in Figure 11(a). We can either include or exclude squares with *ds* from blocks. This gives us many possible choices for the blocks. For example, excluding all squares with *ds* and forming blocks, as shown in Figure 11(b), produces the expression  $w\bar{x}y + \bar{w}xy + \bar{w}xz$ . Including some of the *ds* and excluding others and forming blocks, as

TABLE 1

| Digit | w | x | y | z | F |
|-------|---|---|---|---|---|
| 0     | 0 | 0 | 0 | 0 | 0 |
| 1     | 0 | 0 | 0 | 1 | 0 |
| 2     | 0 | 0 | 1 | 0 | 0 |
| 3     | 0 | 0 | 1 | 1 | 0 |
| 4     | 0 | 1 | 0 | 0 | 0 |
| 5     | 0 | 1 | 0 | 1 | 1 |
| 6     | 0 | 1 | 1 | 0 | 1 |
| 7     | 0 | 1 | 1 | 1 | 1 |
| 8     | 1 | 0 | 0 | 0 | 1 |
| 9     | 1 | 0 | 0 | 1 | 1 |



**FIGURE 11** The K-map for  $F$  Showing Its *Don't Care* Positions.

shown in Figure 11(c), produces the expression  $w\bar{x} + \bar{w}xy + x\bar{y}z$ . Finally, including all the  $ds$  and using the blocks shown in Figure 11(d) produces the simplest sum-of-products expansion possible, namely,  $F(x, y, z) = w + xy + xz$ .

## The Quine–McCluskey Method



We have seen that K-maps can be used to produce minimal expansions of Boolean functions as Boolean sums of Boolean products. However, K-maps are awkward to use when there are more than four variables. Furthermore, the use of K-maps relies on visual inspection to identify terms to group. For these reasons there is a need for a procedure for simplifying sum-of-products expansions that can be mechanized. The Quine–McCluskey method is such a procedure. It can be used for Boolean functions in any number of variables. It was developed in the 1950s by W. V. Quine and E. J. McCluskey, Jr. Basically, the Quine–McCluskey method consists of two



**EDWARD J. MCCLUSKEY (BORN 1929)** Edward McCluskey attended Bowdoin College and M.I.T., where he received his doctorate in electrical engineering in 1956. He joined Bell Telephone Laboratories in 1955, remaining there until 1959. McCluskey was professor of electrical engineering at Princeton University from 1959 until 1966, also serving as director of the Computer Center at Princeton from 1961 to 1966. In 1967 he took a position as professor of computer science and electrical engineering at Stanford University, where he also served as director of the Digital Systems Laboratory from 1969 to 1978. McCluskey has worked in a variety of areas in computer science, including fault-tolerant computing, computer architecture, testing, and logic design. He is the director of the Center for Reliable Computing at Stanford University where he is now an emeritus professor. McCluskey is also an ACM Fellow.

**TABLE 2**

| <i>Minterm</i>          | <i>Bit String</i> | <i>Number of 1s</i> |
|-------------------------|-------------------|---------------------|
| $xyz$                   | 111               | 3                   |
| $x\bar{y}z$             | 101               | 2                   |
| $\bar{x}yz$             | 011               | 2                   |
| $\bar{x}\bar{y}z$       | 001               | 1                   |
| $\bar{x}\bar{y}\bar{z}$ | 000               | 0                   |

parts. The first part finds those terms that are candidates for inclusion in a minimal expansion as a Boolean sum of Boolean products. The second part determines which of these terms to actually use. We will use Example 9 to illustrate how, by successively combining implicants into implicants with one fewer literal, this procedure works.

**EXAMPLE 9** We will show how the Quine–McCluskey method can be used to find a minimal expansion equivalent to

$$xyz + x\bar{y}z + \bar{x}yz + \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z}.$$

We will represent the minterms in this expansion by bit strings. The first bit will be 1 if  $x$  occurs and 0 if  $\bar{x}$  occurs. The second bit will be 1 if  $y$  occurs and 0 if  $\bar{y}$  occurs. The third bit will be 1 if  $z$  occurs and 0 if  $\bar{z}$  occurs. We then group these terms according to the number of 1s in the corresponding bit strings. This information is shown in Table 2.

Minterms that can be combined are those that differ in exactly one literal. Hence, two terms that can be combined differ by exactly one in the number of 1s in the bit strings that represent them. When two minterms are combined into a product, this product contains two literals. A product in two literals is represented using a dash to denote the variable that does not occur. For instance, the minterms  $x\bar{y}z$  and  $\bar{x}\bar{y}z$ , represented by bit strings 101 and 001, can be combined into  $\bar{y}z$ , represented by the string –01. All pairs of minterms that can be combined and the product formed from these combinations are shown in Table 3.

Next, all pairs of products of two literals that can be combined are combined into one literal. Two such products can be combined if they contain literals for the same two variables, and literals for only one of the two variables differ. In terms of the strings representing the products, these strings must have a dash in the same position and must differ in exactly one of the other two slots. We can combine the products  $yz$  and  $\bar{y}z$ , represented by the strings –11 and –01, into  $z$ , represented by the string ––1. We show all the combinations of terms that can be formed in this way in Table 3.

**TABLE 3**

|                           |                   | <i>Step 1</i> |                  | <i>Step 2</i> |               |
|---------------------------|-------------------|---------------|------------------|---------------|---------------|
| <i>Term</i>               | <i>Bit String</i> | <i>Term</i>   | <i>String</i>    | <i>Term</i>   | <i>String</i> |
| 1 $xyz$                   | 111               | (1,2)         | $xz$             | 1–1           | (1,2,3,4) $z$ |
| 2 $x\bar{y}z$             | 101               | (1,3)         | $yz$             | –11           |               |
| 3 $\bar{x}yz$             | 011               | (2,4)         | $\bar{y}z$       | –01           |               |
| 4 $\bar{x}\bar{y}z$       | 001               | (3,4)         | $\bar{x}z$       | 0–1           |               |
| 5 $\bar{x}\bar{y}\bar{z}$ | 000               | (4,5)         | $\bar{x}\bar{y}$ | 00–           |               |

**TABLE 4**

|                  | $xyz$ | $\bar{x}yz$ | $\bar{x}\bar{y}z$ | $\bar{x}\bar{y}\bar{z}$ |   |
|------------------|-------|-------------|-------------------|-------------------------|---|
| $z$              | X     | X           | X                 | X                       |   |
| $\bar{x}\bar{y}$ |       |             |                   | X                       | X |

In Table 3 we also indicate which terms have been used to form products with fewer literals; these terms will not be needed in a minimal expansion. The next step is to identify a minimal set of products needed to represent the Boolean function. We begin with all those products that were not used to construct products with fewer literals. Next, we form Table 4, which has a row for each candidate product formed by combining original terms, and a column for each original term; and we put an X in a position if the original term in the sum-of-products expansion was used to form this candidate product. In this case, we say that the candidate product **covers** the original minterm. We need to include at least one product that covers each of the original minterms. Consequently, whenever there is only one X in a column in the table, the product corresponding to the row this X is in must be used. From Table 4 we see that both  $z$  and  $\bar{x}\bar{y}$  are needed. Hence, the final answer is  $z + \bar{x}\bar{y}$ .  $\blacktriangleleft$

As was illustrated in Example 9, the Quine–McCluskey method uses this sequence of steps to simplify a sum-of-products expression.



1. Express each minterm in  $n$  variables by a bit string of length  $n$  with a 1 in the  $i$ th position if  $x_i$  occurs and a 0 in this position if  $\bar{x}_i$  occurs.
2. Group the bit strings according to the number of 1s in them.
3. Determine all products in  $n - 1$  variables that can be formed by taking the Boolean sum of minterms in the expansion. Minterms that can be combined are represented by bit strings that differ in exactly one position. Represent these products in  $n - 1$  variables with strings that have a 1 in the  $i$ th position if  $x_i$  occurs in the product, a 0 in this position if  $\bar{x}_i$  occurs, and a dash in this position if there is no literal involving  $x_i$  in the product.



**WILLARD VAN ORMAN QUINE (1908–2000)** Willard Quine, born in Akron, Ohio, attended Oberlin College and later Harvard University, where he received his Ph.D. in philosophy in 1932. He became a Junior Fellow at Harvard in 1933 and was appointed to a position on the faculty there in 1936. He remained at Harvard his entire professional life, except for World War II, when he worked for the U.S. Navy decrypting messages from German submarines. Quine was always interested in algorithms, but not in hardware. He arrived at his discovery of what is now called the Quine–McCluskey method as a device for teaching mathematical logic, rather than as a method for simplifying switching circuits. Quine was one of the most famous philosophers of the twentieth century. He made fundamental contributions to the theory of knowledge, mathematical logic and set theory, and the philosophies of logic and language. His books, including *New Foundations of Mathematical Logic* published in 1937 and *Word and Object* published in 1960, have had a profound impact. Quine retired from Harvard in 1978 but continued to commute from his home in Beacon Hill to his office there. He used the 1927 Remington typewriter on which he prepared his doctoral thesis for his entire life. He even had an operation performed on this machine to add a few special symbols, removing the second period, the second comma, and the question mark. When asked whether he missed the question mark, he replied, “Well, you see, I deal in certainties.” There is even a word *quine*, defined in the *New Hacker’s Dictionary* as a program that generates a copy of its own source code as its complete output. Producing the shortest possible quine in a given programming language is a popular puzzle for hackers.

4. Determine all products in  $n - 2$  variables that can be formed by taking the Boolean sum of the products in  $n - 1$  variables found in the previous step. Products in  $n - 1$  variables that can be combined are represented by bit strings that have a dash in the same position and differ in exactly one position.
5. Continue combining Boolean products into products in fewer variables as long as possible.
6. Find all the Boolean products that arose that were not used to form a Boolean product in one fewer literal.
7. Find the smallest set of these Boolean products such that the sum of these products represents the Boolean function. This is done by forming a table showing which minterms are covered by which products. Every minterm must be covered by at least one product. The first step in using this table is to find all essential prime implicants. Each essential prime implicant must be included because it is the only prime implicant that covers one of the minterms. Once we have found essential prime implicants, we can simplify the table by eliminating the rows for minterms covered by these prime implicants. Furthermore, we can eliminate any prime implicants that cover a subset of minterms covered by another prime implicant (as the reader should verify). Moreover, we can eliminate from the table the row for a minterm if there is another minterm that is covered by a subset of the prime implicants that cover this minterm. This process of identifying essential prime implicants that must be included, followed by eliminating redundant prime implicants and identifying minterms that can be ignored, is iterated until the table does not change. At this point we use a backtracking procedure to find the optimal solution where we add prime implicants to the cover to find possible solutions, which we compare to the best solution found so far at each step.

A final example will illustrate how this procedure is used to simplify a sum-of-products expansion in four variables.

**EXAMPLE 10** Use the Quine–McCluskey method to simplify the sum-of-products expansion  $wxy\bar{z} + w\bar{x}yz + \bar{w}\bar{x}y\bar{z} + \bar{w}xyz + \bar{w}x\bar{y}z + \bar{w}\bar{x}\bar{y}z$ .

**Solution:** We first represent the minterms by bit strings and then group these terms together according to the number of 1s in the bit strings. This is shown in Table 5. All the Boolean products that can be formed by taking Boolean sums of these products are shown in Table 6.

The only products that were not used to form products in fewer variables are  $\bar{w}z$ ,  $w\bar{y}\bar{z}$ ,  $w\bar{x}y$ , and  $\bar{x}yz$ . In Table 7 we show the minterms covered by each of these products. To cover these minterms we must include  $\bar{w}z$  and  $w\bar{y}\bar{z}$ , because these products are the only products that cover  $\bar{w}xyz$  and  $wxy\bar{z}$ , respectively. Once these two products are included, we see that only one of the two products left is needed. Consequently, we can take either  $\bar{w}z + w\bar{y}\bar{z} + w\bar{x}y$  or  $\bar{w}z + w\bar{y}\bar{z} + \bar{x}yz$  as the final answer. 

TABLE 5

| Term                     | Bit String | Number of 1s |
|--------------------------|------------|--------------|
| $wxy\bar{z}$             | 1110       | 3            |
| $w\bar{x}yz$             | 1011       | 3            |
| $\bar{w}xyz$             | 0111       | 3            |
| $w\bar{x}y\bar{z}$       | 1010       | 2            |
| $\bar{w}x\bar{y}z$       | 0101       | 2            |
| $\bar{w}\bar{x}yz$       | 0011       | 2            |
| $\bar{w}\bar{x}\bar{y}z$ | 0001       | 1            |

**TABLE 6**

|                            |            | Step 1 |                   | Step 2 |                      |        |
|----------------------------|------------|--------|-------------------|--------|----------------------|--------|
| Term                       | Bit String | Term   | String            | Term   | String               |        |
| 1 $wxy\bar{z}$             | 1110       | (1,4)  | $wy\bar{z}$       | 1–10   | (3,5,6,7) $\bar{w}z$ | 0 – –1 |
| 2 $w\bar{x}yz$             | 1011       | (2,4)  | $w\bar{x}y$       | 101–   |                      |        |
| 3 $\bar{w}xyz$             | 0111       | (2,6)  | $\bar{x}yz$       | –011   |                      |        |
| 4 $w\bar{x}y\bar{z}$       | 1010       | (3,5)  | $\bar{w}xz$       | 01–1   |                      |        |
| 5 $\bar{w}x\bar{y}z$       | 0101       | (3,6)  | $\bar{w}yz$       | 0–11   |                      |        |
| 6 $\bar{w}\bar{x}yz$       | 0011       | (5,7)  | $\bar{w}\bar{y}z$ | 0–01   |                      |        |
| 7 $\bar{w}\bar{x}\bar{y}z$ | 0001       | (6,7)  | $\bar{w}\bar{x}z$ | 00–1   |                      |        |

**TABLE 7**

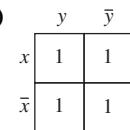
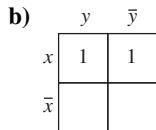
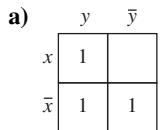
|             | $wxy\bar{z}$ | $w\bar{x}yz$ | $\bar{w}xyz$ | $w\bar{x}y\bar{z}$ | $\bar{w}\bar{x}yz$ | $\bar{w}\bar{x}y$ | $\bar{w}\bar{x}\bar{y}z$ |
|-------------|--------------|--------------|--------------|--------------------|--------------------|-------------------|--------------------------|
| $\bar{w}z$  |              |              | X            |                    | X                  | X                 | X                        |
| $wy\bar{z}$ | X            |              |              | X                  |                    |                   |                          |
| $w\bar{x}y$ |              | X            |              | X                  |                    |                   |                          |
| $\bar{x}yz$ |              | X            |              |                    |                    | X                 |                          |

## Exercises

1. a) Draw a K-map for a function in two variables and put a 1 in the cell representing  $\bar{x}y$ .

- b) What are the minterms represented by cells adjacent to this cell?

2. Find the sum-of-products expansions represented by each of these K-maps.



3. Draw the K-maps of these sum-of-products expansions in two variables.

a)  $x\bar{y}$

b)  $xy + \bar{x}\bar{y}$

c)  $xy + x\bar{y} + \bar{x}y + \bar{x}\bar{y}$

4. Use a K-map to find a minimal expansion as a Boolean sum of Boolean products of each of these functions of the Boolean variables  $x$  and  $y$ .

a)  $\bar{x}y + \bar{x}\bar{y}$

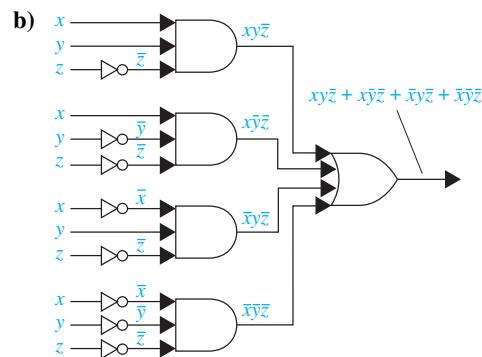
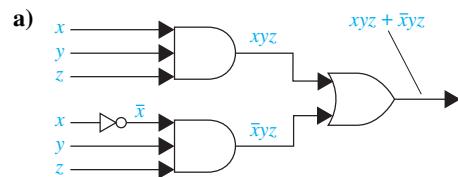
b)  $xy + x\bar{y}$

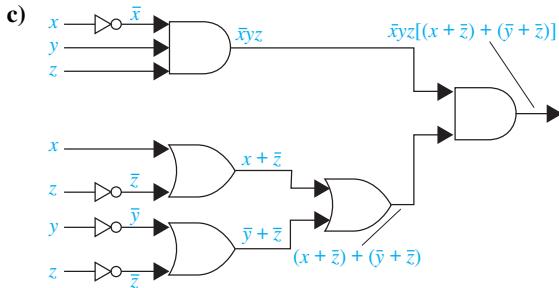
c)  $xy + x\bar{y} + \bar{x}y + \bar{x}\bar{y}$

5. a) Draw a K-map for a function in three variables. Put a 1 in the cell that represents  $\bar{x}y\bar{z}$ .

- b) Which minterms are represented by cells adjacent to this cell?

6. Use K-maps to find simpler circuits with the same output as each of the circuits shown.





7. Draw the K-maps of these sum-of-products expansions in three variables.
  - $x\bar{y}\bar{z}$
  - $\bar{x}yz + \bar{x}\bar{y}\bar{z}$
  - $xyz + xy\bar{z} + \bar{x}y\bar{z} + \bar{x}\bar{y}z$
8. Construct a K-map for  $F(x, y, z) = xz + yz + xy\bar{z}$ . Use this K-map to find the implicants, prime implicants, and essential prime implicants of  $F(x, y, z)$ .
9. Construct a K-map for  $F(x, y, z) = x\bar{z} + xyz + y\bar{z}$ . Use this K-map to find the implicants, prime implicants, and essential prime implicants of  $F(x, y, z)$ .
10. Draw the 3-cube  $Q_3$  and label each vertex with the minterm in the Boolean variables  $x$ ,  $y$ , and  $z$  associated with the bit string represented by this vertex. For each literal in these variables indicate the 2-cube  $Q_2$  that is a subgraph of  $Q_3$  and represents this literal.
11. Draw the 4-cube  $Q_4$  and label each vertex with the minterm in the Boolean variables  $w$ ,  $x$ ,  $y$ , and  $z$  associated with the bit string represented by this vertex. For each literal in these variables, indicate which 3-cube  $Q_3$  that is a subgraph of  $Q_4$  represents this literal. Indicate which 2-cube  $Q_2$  that is a subgraph of  $Q_4$  represents the products  $wz$ ,  $\bar{x}y$ , and  $\bar{y}\bar{z}$ .
12. Use a K-map to find a minimal expansion as a Boolean sum of Boolean products of each of these functions in the variables  $x$ ,  $y$ , and  $z$ .
  - $\bar{x}yz + \bar{x}\bar{y}z$
  - $xyz + xy\bar{z} + \bar{x}yz + \bar{x}\bar{y}z$
  - $xy\bar{z} + x\bar{y}z + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}z$
  - $xyz + x\bar{y}z + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z}$
13. a) Draw a K-map for a function in four variables. Put a 1 in the cell that represents  $\bar{w}xy\bar{z}$ .
  - Which minterms are represented by cells adjacent to this cell?
14. Use a K-map to find a minimal expansion as a Boolean sum of Boolean products of each of these functions in the variables  $w$ ,  $x$ ,  $y$ , and  $z$ .
  - $wxyz + wx\bar{y}\bar{z} + w\bar{x}\bar{y}\bar{z} + w\bar{x}y\bar{z} + w\bar{x}\bar{y}z$
  - $wxy\bar{z} + wx\bar{y}\bar{z} + w\bar{x}yz + \bar{w}x\bar{y}z + \bar{w}\bar{x}\bar{y}z + \bar{w}\bar{x}\bar{y}\bar{z}$
  - $wxyz + wx\bar{y}\bar{z} + w\bar{x}\bar{y}\bar{z} + w\bar{x}y\bar{z} + w\bar{x}\bar{y}z + \bar{w}x\bar{y}z + \bar{w}\bar{x}yz + \bar{w}\bar{x}\bar{y}\bar{z}$
  - $wxyz + wx\bar{y}\bar{z} + wx\bar{y}\bar{z} + w\bar{x}yz + w\bar{x}\bar{y}z + wxyz + \bar{w}x\bar{y}z + \bar{w}\bar{x}yz + \bar{w}\bar{x}\bar{y}\bar{z}$

15. Find the cells in a K-map for Boolean functions with five variables that correspond to each of these products.
  - $x_1x_2x_3x_4$
  - $\bar{x}_1x_3x_5$
  - $x_2x_4$
  - $\bar{x}_3\bar{x}_4$
  - $x_3$
  - $\bar{x}_5$
16. How many cells in a K-map for Boolean functions with six variables are needed to represent  $x_1$ ,  $\bar{x}_1x_6$ ,  $\bar{x}_1x_2\bar{x}_6$ ,  $x_2x_3x_4x_5$ , and  $x_1\bar{x}_2x_4\bar{x}_5$ , respectively?
17. a) How many cells does a K-map in six variables have?  
b) How many cells are adjacent to a given cell in a K-map in six variables?
18. Show that cells in a K-map for Boolean functions in five variables represent minterms that differ in exactly one literal if and only if they are adjacent or are in cells that become adjacent when the top and bottom rows and cells in the first and eighth columns, the first and fourth columns, the second and seventh columns, the third and sixth columns, and the fifth and eighth columns are considered adjacent.
19. Which rows and which columns of a  $4 \times 16$  map for Boolean functions in six variables using the Gray codes 1111, 1110, 1010, 1011, 1001, 1000, 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101 to label the columns and 11, 10, 00, 01 to label the rows need to be considered adjacent so that cells that represent minterms that differ in exactly one literal are considered adjacent?
- \*20. Use K-maps to find a minimal expansion as a Boolean sum of Boolean products of Boolean functions that have as input the binary code for each decimal digit and produce as output a 1 if and only if the digit corresponding to the input is
  - odd.
  - not divisible by 3.
  - not 4, 5, or 6.
- \*21. Suppose that there are five members on a committee, but that Smith and Jones always vote the opposite of Marcus. Design a circuit that implements majority voting of the committee using this relationship between votes.
  22. Use the Quine–McCluskey method to simplify the sum-of-products expansions in Example 3.
  23. Use the Quine–McCluskey method to simplify the sum-of-products expansions in Exercise 12.
  24. Use the Quine–McCluskey method to simplify the sum-of-products expansions in Example 4.
  25. Use the Quine–McCluskey method to simplify the sum-of-products expansions in Exercise 14.
  - \*26. Explain how K-maps can be used to simplify product-of-sums expansions in three variables. [Hint: Mark with a 0 all the maxterms in an expansion and combine blocks of maxterms.]
  27. Use the method from Exercise 26 to simplify the product-of-sums expansion  $(x + y + z)(x + y + \bar{z})(x + \bar{y} + \bar{z})(x + \bar{y} + z)(\bar{x} + y + z)$ .
  - \*28. Draw a K-map for the 16 minterms in four Boolean variables on the surface of a torus.

- 29.** Build a circuit using OR gates, AND gates, and inverters that produces an output of 1 if a decimal digit, encoded using a binary coded decimal expansion, is divisible by 3, and an output of 0 otherwise.

In Exercises 30–32 find a minimal sum-of-products expansion, given the K-map shown with *don't care* conditions indicated with *ds*.

**30.**

|                  | $yz$ | $y\bar{z}$ | $\bar{y}z$ | $\bar{y}\bar{z}$ |
|------------------|------|------------|------------|------------------|
| $wx$             | $d$  | 1          | $d$        | 1                |
| $w\bar{x}$       |      | $d$        | $d$        |                  |
| $\bar{w}\bar{x}$ |      | $d$        |            |                  |
| $\bar{w}x$       | 1    | $d$        |            |                  |

**31.**

|                  | $yz$ | $y\bar{z}$ | $\bar{y}z$ | $\bar{y}\bar{z}$ |
|------------------|------|------------|------------|------------------|
| $wx$             | 1    |            |            | 1                |
| $w\bar{x}$       |      | $d$        | 1          |                  |
| $\bar{w}\bar{x}$ |      | 1          | $d$        |                  |
| $\bar{w}x$       | $d$  |            |            | $d$              |

**32.**

|                  | $yz$ | $y\bar{z}$ | $\bar{y}z$ | $\bar{y}\bar{z}$ |
|------------------|------|------------|------------|------------------|
| $wx$             |      | $d$        | $d$        | 1                |
| $w\bar{x}$       | $d$  | $d$        | 1          | $d$              |
| $\bar{w}\bar{x}$ |      |            |            |                  |
| $\bar{w}x$       | 1    | 1          | 1          | $d$              |

- 33.** Show that products of  $k$  literals correspond to  $2^{n-k}$ -dimensional subcubes of the  $n$ -cube  $Q_n$ , where the vertices of the cube correspond to the minterms represented by the bit strings labeling the vertices, as described in Example 8 of Section 10.2.

## Key Terms and Results

### TERMS

**Boolean variable:** a variable that assumes only the values 0 and 1

**$\bar{x}$  (complement of  $x$ ):** an expression with the value 1 when  $x$  has the value 0 and the value 0 when  $x$  has the value 1

**$x \cdot y$  (or  $xy$ ) (Boolean product or conjunction of  $x$  and  $y$ ):** an expression with the value 1 when both  $x$  and  $y$  have the value 1 and the value 0 otherwise

**$x + y$  (Boolean sum or disjunction of  $x$  and  $y$ ):** an expression with the value 1 when either  $x$  or  $y$ , or both, has the value 1, and 0 otherwise

**Boolean expressions:** the expressions obtained recursively by specifying that 0, 1,  $x_1, \dots, x_n$  are Boolean expressions and  $\bar{E}_1$ ,  $(E_1 + E_2)$ , and  $(E_1 \cdot E_2)$  are Boolean expressions if  $E_1$  and  $E_2$  are

**dual of a Boolean expression:** the expression obtained by interchanging + signs and · signs and interchanging 0s and 1s

**Boolean function of degree  $n$ :** a function from  $B^n$  to  $B$  where  $B = \{0, 1\}$

**Boolean algebra:** a set  $B$  with two binary operations  $\vee$  and  $\wedge$ , elements 0 and 1, and a complementation operator  $\neg$  that satisfies the identity, complement, associative, commutative, and distributive laws

**literal of the Boolean variable  $x$ :** either  $x$  or  $\bar{x}$

**minterm of  $x_1, x_2, \dots, x_n$ :** a Boolean product  $y_1 y_2 \dots y_n$ , where each  $y_i$  is either  $x_i$  or  $\bar{x}_i$

**sum-of-products expansion (or disjunctive normal form):** the representation of a Boolean function as a disjunction of minterms

**functionally complete:** a set of Boolean operators is called functionally complete if every Boolean function can be represented using these operators

**$x \mid y$  (or  $x \text{ NAND } y$ ):** the expression that has the value 0 when both  $x$  and  $y$  have the value 1 and the value 1 otherwise

**$x \downarrow y$  (or  $x \text{ NOR } y$ ):** the expression that has the value 0 when either  $x$  or  $y$  or both have the value 1 and the value 0 otherwise

**inverter:** a device that accepts the value of a Boolean variable as input and produces the complement of the input

**OR gate:** a device that accepts the values of two or more Boolean variables as input and produces their Boolean sum as output

**AND gate:** a device that accepts the values of two or more Boolean variables as input and produces their Boolean product as output

**half adder:** a circuit that adds two bits, producing a sum bit and a carry bit

**full adder:** a circuit that adds two bits and a carry, producing a sum bit and a carry bit

**K-map for  $n$  variables:** a rectangle divided into  $2^n$  cells where each cell represents a minterm in the variables

**minimization of a Boolean function:** representing a Boolean function as the sum of the fewest products of literals such that these products contain the fewest literals possible among all sums of products that represent this Boolean function

**implicant of a Boolean function:** a product of literals with the property that if this product has the value 1, then the value of this Boolean function is 1

**prime implicant of a Boolean function:** a product of literals that is an implicant of the Boolean function and no product obtained by deleting a literal is also an implicant of this function

**essential prime implicant of a Boolean function:** a prime implicant of the Boolean function that must be included in a minimization of this function

**don't care condition:** a combination of input values for a circuit that is not possible or never occurs

### RESULTS

The identities for Boolean algebra (see Table 5 in Section 12.1).

An identity between Boolean functions represented by Boolean expressions remains valid when the duals of both sides of the identity are taken.

Every Boolean function can be represented by a sum-of-products expansion.

Each of the sets  $\{+, -\}$  and  $\{\cdot, \bar{\cdot}\}$  is functionally complete.

Each of the sets  $\{\downarrow\}$  and  $\{\mid\}$  is functionally complete.

The use of K-maps to minimize Boolean expressions.

The Quine–McCluskey method for minimizing Boolean expressions.

## Review Questions

1. Define a Boolean function of degree  $n$ .
2. How many Boolean functions of degree two are there?
3. Give a recursive definition of the set of Boolean expressions.
4. a) What is the dual of a Boolean expression?  
b) What is the duality principle? How can it be used to find new identities involving Boolean expressions?
5. Explain how to construct the sum-of-products expansion of a Boolean function.
6. a) What does it mean for a set of operators to be functionally complete?  
b) Is the set  $\{+, \cdot\}$  functionally complete?  
c) Are there sets of a single operator that are functionally complete?
7. Explain how to build a circuit for a light controlled by two switches using OR gates, AND gates, and inverters.
8. Construct a half adder using OR gates, AND gates, and inverters.
9. Is there a single type of logic gate that can be used to build all circuits that can be built using OR gates, AND gates, and inverters?
10. a) Explain how K-maps can be used to simplify sum-of-products expansions in three Boolean variables.  
b) Use a K-map to simplify the sum-of-products expansion  $xyz + x\bar{y}z + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}z$ .
11. a) Explain how K-maps can be used to simplify sum-of-products expansions in four Boolean variables.  
b) Use a K-map to simplify the sum-of-products expansion  $wxyz + wxy\bar{z} + wx\bar{y}z + wx\bar{y}\bar{z} + w\bar{x}yz + w\bar{x}y\bar{z} + \bar{w}xyz + \bar{w}\bar{x}yz + \bar{w}\bar{x}\bar{y}z$ .
12. a) What is a *don't care* condition?  
b) Explain how *don't care* conditions can be used to build a circuit using OR gates, AND gates, and inverters that produces an output of 1 if a decimal digit is 6 or greater, and an output of 0 if this digit is less than 6.
13. a) Explain how to use the Quine–McCluskey method to simplify sum-of-products expansions.  
b) Use this method to simplify  $xy\bar{z} + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}z + \bar{x}\bar{y}\bar{z}$ .

## Supplementary Exercises

1. For which values of the Boolean variables  $x$ ,  $y$ , and  $z$  does
  - a)  $x + y + z = xyz$ ?
  - b)  $x(y + z) = x + yz$ ?
  - c)  $\bar{x}\bar{y}\bar{z} = x + y + z$ ?
2. Let  $x$  and  $y$  belong to  $\{0, 1\}$ . Does it necessarily follow that  $x = y$  if there exists a value  $z$  in  $\{0, 1\}$  such that
  - a)  $xz = yz$
  - b)  $x + z = y + z$ ?
  - c)  $x \oplus z = y \oplus z$ ?
  - d)  $x \downarrow z = y \downarrow z$ ?
  - e)  $x \mid z = y \mid z$ ?
- A Boolean function  $F$  is called **self-dual** if and only if  $F(x_1, \dots, x_n) = \overline{F(\bar{x}_1, \dots, \bar{x}_n)}$ .
3. Which of these functions are self-dual?
  - a)  $F(x, y) = x$
  - b)  $F(x, y) = xy + \bar{x}\bar{y}$
  - c)  $F(x, y) = x + y$
  - d)  $F(x, y) = xy + \bar{x}y$
4. Give an example of a self-dual Boolean function of three variables.
- \*5. How many Boolean functions of degree  $n$  are self-dual?  
We define the relation  $\leq$  on the set of Boolean functions of degree  $n$  so that  $F \leq G$ , where  $F$  and  $G$  are Boolean functions if and only if  $G(x_1, x_2, \dots, x_n) = 1$  whenever  $F(x_1, x_2, \dots, x_n) = 1$ .
6. Determine whether  $F \leq G$  or  $G \leq F$  for the following pairs of functions.
  - a)  $F(x, y) = x$ ,  $G(x, y) = x + y$
  - b)  $F(x, y) = x + y$ ,  $G(x, y) = xy$
  - c)  $F(x, y) = \bar{x}$ ,  $G(x, y) = x + y$
7. Show that if  $F$  and  $G$  are Boolean functions of degree  $n$ , then
  - a)  $F \leq F + G$ .
  - b)  $FG \leq F$ .
8. Show that if  $F$ ,  $G$ , and  $H$  are Boolean functions of degree  $n$ , then  $F + G \leq H$  if and only if  $F \leq H$  and  $G \leq H$ .
- \*9. Show that the relation  $\leq$  is a partial ordering on the set of Boolean functions of degree  $n$ .
- \*10. Draw the Hasse diagram for the poset consisting of the set of the 16 Boolean functions of degree two (shown in Table 3 of Section 12.1) with the partial ordering  $\leq$ .
- \*11. For each of these equalities either prove it is an identity or find a set of values of the variables for which it does not hold.
  - a)  $x \mid (y \mid z) = (x \mid y) \mid z$
  - b)  $x \downarrow (y \downarrow z) = (x \downarrow y) \downarrow (x \downarrow z)$
  - c)  $x \downarrow (y \mid z) = (x \downarrow y) \mid (x \downarrow z)$

Define the Boolean operator  $\odot$  as follows:  $1 \odot 1 = 1$ ,  $1 \odot 0 = 0$ ,  $0 \odot 1 = 0$ , and  $0 \odot 0 = 1$ .

12. Show that  $x \odot y = xy + \bar{x}\bar{y}$ .

13. Show that  $x \odot y = \overline{(x \oplus y)}$ .

14. Show that each of these identities holds.

a)  $x \odot x = 1$       b)  $x \odot \bar{x} = 0$   
c)  $x \odot y = y \odot x$

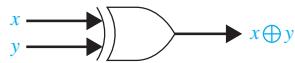
15. Is it always true that  $(x \odot y) \odot z = x \odot (y \odot z)$ ?

\*16. Determine whether the set  $\{\odot\}$  is functionally complete.

\*17. How many of the 16 Boolean functions in two variables  $x$  and  $y$  can be represented using only the given set of operators, variables  $x$  and  $y$ , and values 0 and 1?

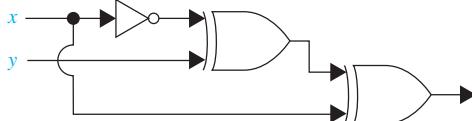
- a)  $\{\square\}$     b)  $\{\cdot\}$     c)  $\{+\}$     d)  $\{\cdot, +\}$

The notation for an **XOR gate**, which produces the output  $x \oplus y$  from  $x$  and  $y$ , is as follows:

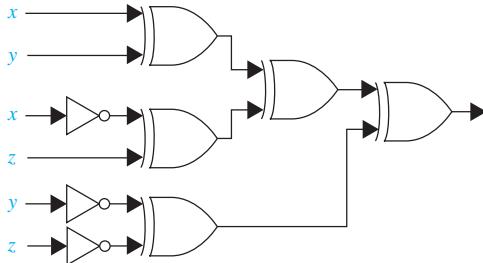


18. Determine the output of each of these circuits.

a)



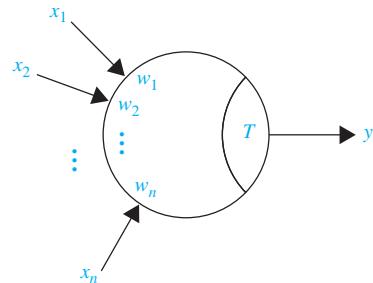
b)



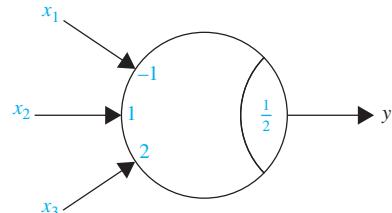
19. Show how a half adder can be constructed using fewer gates than are used in Figure 8 of Section 12.3 when XOR gates can be used in addition to OR gates, AND gates, and inverters.

20. Design a circuit that determines whether three or more of four individuals on a committee vote yes on an issue, where each individual uses a switch for the voting.

A **threshold gate** produces an output  $y$  that is either 0 or 1 given a set of input values for the Boolean variables  $x_1, x_2, \dots, x_n$ . A threshold gate has a **threshold value**  $T$ , which is a real number, and **weights**  $w_1, w_2, \dots, w_n$ , each of which is a real number. The output  $y$  of the threshold gate is 1 if and only if  $w_1x_1 + w_2x_2 + \dots + w_nx_n \geq T$ . The threshold gate with threshold value  $T$  and weights  $w_1, w_2, \dots, w_n$  is represented by the following diagram. Threshold gates are useful in modeling in neurophysiology and in artificial intelligence.



21. A threshold gate represents a Boolean function. Find a Boolean expression for the Boolean function represented by this threshold gate.



22. A Boolean function that can be represented by a threshold gate is called a **threshold function**. Show that each of these functions is a threshold function.

- a)  $F(x) = \bar{x}$       b)  $F(x, y) = x + y$   
c)  $F(x, y) = xy$       d)  $F(x, y) = x \mid y$   
e)  $F(x, y) = x \downarrow y$       f)  $F(x, y, z) = x + yz$   
g)  $F(w, x, y, z) = w + xy + z$   
h)  $F(w, x, y, z) = wxz + x\bar{y}z$

\*23. Show that  $F(x, y) = x \oplus y$  is not a threshold function.

\*24. Show that  $F(w, x, y, z) = wx + yz$  is not a threshold function.

## Computer Projects

Write programs with these input and output.

1. Given the values of two Boolean variables  $x$  and  $y$ , find the values of  $x + y$ ,  $xy$ ,  $x \oplus y$ ,  $x \mid y$ , and  $x \downarrow y$ .
2. Construct a table listing the set of values of all 256 Boolean functions of degree three.
3. Given the values of a Boolean function in  $n$  variables, where  $n$  is a positive integer, construct the sum-of-products expansion of this function.
4. Given the table of values of a Boolean function, express this function using only the operators  $\cdot$  and  $\bar{\phantom{x}}$ .
5. Given the table of values of a Boolean function, express this function using only the operators  $+$  and  $\mid$ .
- \*6. Given the table of values of a Boolean function, express this function using only the operator  $|$ .

- \*7. Given the table of values of a Boolean function, express this function using only the operator  $\downarrow$ .
- 8. Given the table of values of a Boolean function of degree three, construct its K-map.
- 9. Given the table of values of a Boolean function of degree four, construct its K-map.
- \*\*10. Given the table of values of a Boolean function, use the Quine–McCluskey method to find a minimal sum-of-products representation of this function.
- 11. Given a threshold value and a set of weights for a threshold gate and the values of the  $n$  Boolean variables in the input, determine the output of this gate.
- 12. Given a positive integer  $n$ , construct a random Boolean expression in  $n$  variables in disjunctive normal form.

## Computations and Explorations

---

Use a computational program or programs you have written to do these exercises.

1. Compute the number of Boolean functions of degrees seven, eight, nine, and ten.
2. Construct a table of the Boolean functions of degree three.
3. Construct a table of the Boolean functions of degree four.
4. Express each of the different Boolean expressions in three variables in disjunctive normal form with just the *NAND* operator, using as few *NAND* operators as possible. What is the largest number of *NAND* operators required?
5. Express each of the different Boolean expressions in disjunctive normal form in four variables using just the *NOR*

operator, with as few *NOR* operators as possible. What is the largest number of *NOR* operators required?

6. Randomly generate 10 different Boolean expressions in four variables and determine the average number of steps required to minimize them using the Quine–McCluskey method.
7. Randomly generate 10 different Boolean expressions in five variables and determine the average number of steps required to minimize them using the Quine–McCluskey method.

## Writing Projects

---

Respond to these with essays using outside sources.

1. Describe some of the early machines devised to solve problems in logic, such as the Stanhope Demonstrator, Jevons's Logic Machine, and the Marquand Machine.
2. Explain the difference between combinational circuits and sequential circuits. Then explain how *flip-flops* are used to build sequential circuits.
3. Define a *shift register* and discuss how shift registers are used. Show how to build shift registers using flip-flops and logic gates.
4. Show how *multipliers* can be built using logic gates.
5. Find out how logic gates are physically constructed. Discuss whether *NAND* and *NOR* gates are used in building circuits.
6. Explain how *dependency notation* can be used to describe complicated switching circuits.
7. Describe how multiplexers are used to build switching circuits.
8. Explain the advantages of using threshold gates to construct switching circuits. Illustrate this by using threshold gates to construct half and full adders.
9. Describe the concept of *hazard-free switching circuits* and give some of the principles used in designing such circuits.
10. Explain how to use K-maps to minimize functions of six variables.
11. Discuss the ideas used by newer methods for minimizing Boolean functions, such as Espresso. Explain how these methods can help solve minimization problems in as many as 25 variables.
12. Describe what is meant by the *functional decomposition* of a Boolean function of  $n$  variables and discuss procedures for decomposing Boolean functions into a composition of Boolean functions with fewer variables.

# 13

# ~~A~~ Modeling Computation

**13.1** Languages and Grammars

**13.2** Finite-State Machines with Output

**13.3** Finite-State Machines with No Output

**13.4** Language Recognition

**13.5** Turing Machines

**C**omputers can perform many tasks. Given a task, two questions arise. The first is: Can it be carried out using a computer? Once we know that this first question has an affirmative answer, we can ask the second question: How can the task be carried out? Models of computation are used to help answer these questions.

We will study three types of structures used in models of computation, namely, grammars, finite-state machines, and Turing machines. Grammars are used to generate the words of a language and to determine whether a word is in a language. Formal languages, which are generated by grammars, provide models both for natural languages, such as English, and for programming languages, such as Pascal, Fortran, Prolog, C, and Java. In particular, grammars are extremely important in the construction and theory of compilers. The grammars that we will discuss were first used by the American linguist Noam Chomsky in the 1950s.

Various types of finite-state machines are used in modeling. All finite-state machines have a set of states, including a starting state, an input alphabet, and a transition function that assigns a next state to every pair of a state and an input. The states of a finite-state machine give it limited memory capabilities. Some finite-state machines produce an output symbol for each transition; these machines can be used to model many kinds of machines, including vending machines, delay machines, binary adders, and language recognizers. We will also study finite-state machines that have no output but do have final states. Such machines are extensively used in language recognition. The strings that are recognized are those that take the starting state to a final state. The concepts of grammars and finite-state machines can be tied together. We will characterize those sets that are recognized by a finite-state machine and show that these are precisely the sets that are generated by a certain type of grammar.

Finally, we will introduce the concept of a Turing machine. We will show how Turing machines can be used to recognize sets. We will also show how Turing machines can be used to compute number-theoretic functions. We will discuss the Church–Turing thesis, which states that every effective computation can be carried out using a Turing machine. We will explain how Turing machines can be used to study the difficulty of solving certain classes of problems. In particular, we will describe how Turing machines are used to classify problems as tractable versus intractable and solvable versus unsolvable.

## 13.1 Languages and Grammars

### Introduction

Words in the English language can be combined in various ways. The grammar of English tells us whether a combination of words is a valid sentence. For instance, *the frog writes neatly* is a valid sentence, because it is formed from a noun phrase, *the frog*, made up of the article *the* and the noun *frog*, followed by a verb phrase, *writes neatly*, made up of the verb *writes* and the adverb *neatly*. We do not care that this is a nonsensical statement, because we are concerned only with the **syntax**, or form, of the sentence, and not its **semantics**, or meaning. We also note that the combination of words *swims quickly mathematics* is not a valid sentence because it does not follow the rules of English grammar.

The syntax of a **natural language**, that is, a spoken language, such as English, French, German, or Spanish, is extremely complicated. In fact, it does not seem possible to specify all the rules of syntax for a natural language. Research in the automatic translation of one language

to another has led to the concept of a **formal language**, which, unlike a natural language, is specified by a well-defined set of rules of syntax. Rules of syntax are important not only in linguistics, the study of natural languages, but also in the study of programming languages.

We will describe the sentences of a formal language using a grammar. The use of grammars helps when we consider the two classes of problems that arise most frequently in applications to programming languages: (1) How can we determine whether a combination of words is a valid sentence in a formal language? (2) How can we generate the valid sentences of a formal language? Before giving a technical definition of a grammar, we will describe an example of a grammar that generates a subset of English. This subset of English is defined using a list of rules that describe how a valid sentence can be produced. We specify that

1. a **sentence** is made up of a **noun phrase** followed by a **verb phrase**;
2. a **noun phrase** is made up of an **article** followed by an **adjective** followed by a **noun**, or
3. a **noun phrase** is made up of an **article** followed by a **noun**;
4. a **verb phrase** is made up of a **verb** followed by an **adverb**, or
5. a **verb phrase** is made up of a **verb**;
6. an **article** is *a*, or
7. an **article** is *the*;
8. an **adjective** is *large*, or
9. an **adjective** is *hungry*;
10. a **noun** is *rabbit*, or
11. a **noun** is *mathematician*;
12. a **verb** is *eats*, or
13. a **verb** is *hops*;
14. an **adverb** is *quickly*, or
15. an **adverb** is *wildly*.

From these rules we can form valid sentences using a series of replacements until no more rules can be used. For instance, we can follow the sequence of replacements:

```

sentence
noun phrase verb phrase
article adjective noun verb phrase
article adjective noun verb adverb
the adjective noun verb adverb
the large noun verb adverb
the large rabbit verb adverb
the large rabbit hops adverb
the large rabbit hops quickly

```

to obtain a valid sentence. It is also easy to see that some other valid sentences are: *a hungry mathematician eats wildly*, *a large mathematician hops*, *the rabbit eats quickly*, and so on. Also, we can see that *the quickly eats mathematician* is not a valid sentence.

## Phrase-Structure Grammars

Before we give a formal definition of a grammar, we introduce a little terminology.

**DEFINITION 1**

A *vocabulary* (or *alphabet*)  $V$  is a finite, nonempty set of elements called *symbols*. A *word* (or *sentence*) over  $V$  is a string of finite length of elements of  $V$ . The *empty string* or *null string*, denoted by  $\lambda$ , is the string containing no symbols. The set of all words over  $V$  is denoted by  $V^*$ . A *language over*  $V$  is a subset of  $V^*$ .

Note that  $\lambda$ , the empty string, is the string containing no symbols. It is different from  $\emptyset$ , the empty set. It follows that  $\{\lambda\}$  is the set containing exactly one string, namely, the empty string.

Languages can be specified in various ways. One way is to list all the words in the language. Another is to give some criteria that a word must satisfy to be in the language. In this section, we describe another important way to specify a language, namely, through the use of a grammar, such as the set of rules we gave in the introduction to this section. A grammar provides a set of symbols of various types and a set of rules for producing words. More precisely, a grammar has a **vocabulary**  $V$ , which is a set of symbols used to derive members of the language. Some of the elements of the vocabulary cannot be replaced by other symbols. These are called **terminals**, and the other members of the vocabulary, which can be replaced by other symbols, are called **nonterminals**. The sets of terminals and nonterminals are usually denoted by  $T$  and  $N$ , respectively. In the example given in the introduction of the section, the set of terminals is  $\{a, \text{the}, \text{rabbit}, \text{mathematician}, \text{hops}, \text{eats}, \text{quickly}, \text{wildly}\}$ , and the set of nonterminals is  $\{\text{sentence, noun phrase, verb phrase, adjective, article, noun, verb, adverb}\}$ . There is a special member of the vocabulary called the **start symbol**, denoted by  $S$ , which is the element of the vocabulary that we always begin with. In the example in the introduction, the start symbol is **sentence**. The rules that specify when we can replace a string from  $V^*$ , the set of all strings of elements in the vocabulary, with another string are called the **productions** of the grammar. We denote by  $z_0 \rightarrow z_1$  the production that specifies that  $z_0$  can be replaced by  $z_1$  within a string. The productions in the grammar given in the introduction of this section were listed. The first production, written using this notation, is **sentence  $\rightarrow$  noun phrase verb phrase**. We summarize this terminology in Definition 2.

The notion of a phrase-structure grammar extends the concept of a rewrite system devised by Axel Thue in the early 20th century.

**DEFINITION 2**

A *phrase-structure grammar*  $G = (V, T, S, P)$  consists of a vocabulary  $V$ , a subset  $T$  of  $V$  consisting of terminal symbols, a start symbol  $S$  from  $V$ , and a finite set of productions  $P$ . The set  $V - T$  is denoted by  $N$ . Elements of  $N$  are called *nonterminal symbols*. Every production in  $P$  must contain at least one nonterminal on its left side.

**EXAMPLE 1**

Let  $G = (V, T, S, P)$ , where  $V = \{a, b, A, B, S\}$ ,  $T = \{a, b\}$ ,  $S$  is the start symbol, and  $P = \{S \rightarrow ABa, A \rightarrow BB, B \rightarrow ab, AB \rightarrow b\}$ .  $G$  is an example of a phrase-structure grammar. 

We will be interested in the words that can be generated by the productions of a phrase-structure grammar.

**DEFINITION 3**

Let  $G = (V, T, S, P)$  be a phrase-structure grammar. Let  $w_0 = lz_0r$  (that is, the concatenation of  $l$ ,  $z_0$ , and  $r$ ) and  $w_1 = lz_1r$  be strings over  $V$ . If  $z_0 \rightarrow z_1$  is a production of  $G$ , we say that  $w_1$  is *directly derivable* from  $w_0$  and we write  $w_0 \Rightarrow w_1$ . If  $w_0, w_1, \dots, w_n$  are strings over  $V$  such that  $w_0 \Rightarrow w_1, w_1 \Rightarrow w_2, \dots, w_{n-1} \Rightarrow w_n$ , then we say that  $w_n$  is *derivable* from  $w_0$ , and we write  $w_0 \xrightarrow{*} w_n$ . The sequence of steps used to obtain  $w_n$  from  $w_0$  is called a *derivation*.

**EXAMPLE 2** The string  $Aaba$  is directly derivable from  $ABa$  in the grammar in Example 1 because  $B \rightarrow ab$  is a production in the grammar. The string  $abababa$  is derivable from  $ABa$  because  $ABa \Rightarrow Aaba \Rightarrow BBaba \Rightarrow Bababa \Rightarrow abababa$ , using the productions  $B \rightarrow ab$ ,  $A \rightarrow BB$ ,  $B \rightarrow ab$ , and  $B \rightarrow ab$  in succession. 

#### DEFINITION 4

Let  $G = (V, T, S, P)$  be a phrase-structure grammar. The *language generated by G* (or the *language of G*), denoted by  $L(G)$ , is the set of all strings of terminals that are derivable from the starting state  $S$ . In other words,

$$L(G) = \{w \in T^* \mid S \xrightarrow{*} w\}.$$

In Examples 3 and 4 we find the language generated by a phrase-structure grammar.

**EXAMPLE 3** Let  $G$  be the grammar with vocabulary  $V = \{S, A, a, b\}$ , set of terminals  $T = \{a, b\}$ , starting symbol  $S$ , and productions  $P = \{S \rightarrow aA, S \rightarrow b, A \rightarrow aa\}$ . What is  $L(G)$ , the language of this grammar?

*Solution:* From the start state  $S$  we can derive  $aA$  using the production  $S \rightarrow aA$ . We can also use the production  $S \rightarrow b$  to derive  $b$ . From  $aA$  the production  $A \rightarrow aa$  can be used to derive  $aaa$ . No additional words can be derived. Hence,  $L(G) = \{b, aaa\}$ . 

**EXAMPLE 4** Let  $G$  be the grammar with vocabulary  $V = \{S, 0, 1\}$ , set of terminals  $T = \{0, 1\}$ , starting symbol  $S$ , and productions  $P = \{S \rightarrow 11S, S \rightarrow 0\}$ . What is  $L(G)$ , the language of this grammar?

*Solution:* From  $S$  we can derive 0 using  $S \rightarrow 0$ , or  $11S$  using  $S \rightarrow 11S$ . From  $11S$  we can derive either  $110$  or  $1111S$ . From  $1111S$  we can derive  $11110$  and  $111111S$ . At any stage of a derivation we can either add two 1s at the end of the string or terminate the derivation by adding a 0 at the end of the string. We surmise that  $L(G) = \{0, 110, 11110, 1111110, \dots\}$ , the set of all strings that begin with an even number of 1s and end with a 0. This can be proved using an inductive argument that shows that after  $n$  productions have been used, the only strings of terminals generated are those consisting of  $n - 1$  concatenations of 11 followed by 0. (This is left as an exercise for the reader.) 

The problem of constructing a grammar that generates a given language often arises. Examples 5, 6, and 7 describe problems of this kind.

**EXAMPLE 5** Give a phrase-structure grammar that generates the set  $\{0^n 1^n \mid n = 0, 1, 2, \dots\}$ .

*Solution:* Two productions can be used to generate all strings consisting of a string of 0s followed by a string of the same number of 1s, including the null string. The first builds up successively longer strings in the language by adding a 0 at the start of the string and a 1 at the end. The second production replaces  $S$  with the empty string. The solution is the grammar  $G = (V, T, S, P)$ , where  $V = \{0, 1, S\}$ ,  $T = \{0, 1\}$ ,  $S$  is the starting symbol, and the productions are

$$\begin{aligned} S &\rightarrow 0S1 \\ S &\rightarrow \lambda. \end{aligned}$$

The verification that this grammar generates the correct set is left as an exercise for the reader. 

Example 5 involved the set of strings made up of 0s followed by 1s, where the number of 0s and 1s are the same. Example 6 considers the set of strings consisting of 0s followed by 1s, where the number of 0s and 1s may differ.

**EXAMPLE 6** Find a phrase-structure grammar to generate the set  $\{0^m 1^n \mid m \text{ and } n \text{ are nonnegative integers}\}$ .

**Solution:** We will give two grammars  $G_1$  and  $G_2$  that generate this set. This will illustrate that two grammars can generate the same language.

The grammar  $G_1$  has alphabet  $V = \{S, 0, 1\}$ ; terminals  $T = \{0, 1\}$ ; and productions  $S \rightarrow 0S$ ,  $S \rightarrow S1$ , and  $S \rightarrow \lambda$ .  $G_1$  generates the correct set, because using the first production  $m$  times puts  $m$  0s at the beginning of the string, and using the second production  $n$  times puts  $n$  1s at the end of the string. The details of this verification are left to the reader.

The grammar  $G_2$  has alphabet  $V = \{S, A, 0, 1\}$ ; terminals  $T = \{0, 1\}$ ; and productions  $S \rightarrow 0S$ ,  $S \rightarrow 1A$ ,  $S \rightarrow 1$ ,  $A \rightarrow 1A$ ,  $A \rightarrow 1$ , and  $S \rightarrow \lambda$ . The details that this grammar generates the correct set are left as an exercise for the reader. 

Sometimes a set that is easy to describe can be generated only by a complicated grammar. Example 7 illustrates this.

**EXAMPLE 7** One grammar that generates the set  $\{0^n 1^n 2^n \mid n = 0, 1, 2, 3, \dots\}$  is  $G = (V, T, S, P)$  with  $V = \{0, 1, 2, S, A, B, C\}$ ;  $T = \{0, 1, 2\}$ ; starting state  $S$ ; and productions  $S \rightarrow C$ ,  $C \rightarrow 0CAB$ ,  $S \rightarrow \lambda$ ,  $BA \rightarrow AB$ ,  $0A \rightarrow 01$ ,  $1A \rightarrow 11$ ,  $1B \rightarrow 12$ , and  $2B \rightarrow 22$ . We leave it as an exercise for the reader (Exercise 12) to show that this statement is correct. The grammar given is the simplest type of grammar that generates this set, in a sense that will be made clear later in this section. 

## Types of Phrase-Structure Grammars



Phrase-structure grammars can be classified according to the types of productions that are allowed. We will describe the classification scheme introduced by Noam Chomsky. In Section 13.4 we will see that the different types of languages defined in this scheme correspond to the classes of languages that can be recognized using different models of computing machines.

A **type 0** grammar has no restrictions on its productions. A **type 1** grammar can have productions of the form  $w_1 \rightarrow w_2$ , where  $w_1 = lAr$  and  $w_2 = lwr$ , where  $A$  is a nonterminal symbol,  $l$  and  $r$  are strings of zero or more terminal or nonterminal symbols, and  $w$  is a nonempty string of terminal or nonterminal symbols. It can also have the production  $S \rightarrow \lambda$  as long as  $S$  does not appear on the right-hand side of any other production. A **type 2** grammar can have productions only of the form  $w_1 \rightarrow w_2$ , where  $w_1$  is a single symbol that is not a terminal symbol. A **type 3** grammar can have productions only of the form  $w_1 \rightarrow w_2$  with  $w_1 = A$  and either  $w_2 = aB$  or  $w_2 = a$ , where  $A$  and  $B$  are nonterminal symbols and  $a$  is a terminal symbol, or with  $w_1 = S$  and  $w_2 = \lambda$ .

Type 2 grammars are called **context-free grammars** because a nonterminal symbol that is the left side of a production can be replaced in a string whenever it occurs, no matter what else is in the string. A language generated by a type 2 grammar is called a **context-free language**. When there is a production of the form  $lw_1r \rightarrow lw_2r$  (but not of the form  $w_1 \rightarrow w_2$ ), the grammar is called type 1 or **context-sensitive** because  $w_1$  can be replaced by  $w_2$  only when it is surrounded by the strings  $l$  and  $r$ . A language generated by a type 1 grammar is called a **context-sensitive language**. Type 3 grammars are also called **regular grammars**. A language generated by a regular grammar is called **regular**. Section 13.4 deals with the relationship between regular languages and finite-state machines.

Of the four types of grammars we have defined, context-sensitive grammars have the most complicated definition. Sometimes, these grammars are defined in a different way. A production of the form  $w_1 \rightarrow w_2$  is called **noncontracting** if the length of  $w_1$  is less than or equal to the

length of  $w_2$ . According to our characterization of context-sensitive languages, every production in a type 1 grammar, other than the production  $S \rightarrow \lambda$ , if it is present, is noncontracting. It follows that the lengths of the strings in a derivation in a context-sensitive language are nondecreasing unless the production  $S \rightarrow \lambda$  is used. This means that the only way for the empty string to belong to the language generated by a context-sensitive grammar is for the production  $S \rightarrow \lambda$  to be part of the grammar. The other way that context-sensitive grammars are defined is by specifying that all productions are noncontracting. A grammar with this property is called **noncontracting** or **monotonic**. The class of noncontracting grammars is not the same as the class of context-sensitive grammars. However, these two classes are closely related; it can be shown that they define the same set of languages except that noncontracting grammars cannot generate any language containing the empty string  $\lambda$ .

**EXAMPLE 8**

From Example 6 we know that  $\{0^m 1^n \mid m, n = 0, 1, 2, \dots\}$  is a regular language, because it can be generated by a regular grammar, namely, the grammar  $G_2$  in Example 6.

Context-free and regular grammars play an important role in programming languages. Context-free grammars are used to define the syntax of almost all programming languages. These grammars are strong enough to define a wide range of languages. Furthermore, efficient algorithms can be devised to determine whether and how a string can be generated. Regular grammars are used to search text for certain patterns and in lexical analysis, which is the process of transforming an input stream into a stream of tokens for use by a parser.

**EXAMPLE 9**

It follows from Example 5 that  $\{0^n 1^n \mid n = 0, 1, 2, \dots\}$  is a context-free language, because the productions in this grammar are  $S \rightarrow 0S1$  and  $S \rightarrow \lambda$ . However, it is not a regular language. This will be shown in Section 13.4.

**EXAMPLE 10**

The set  $\{0^n 1^n 2^n \mid n = 0, 1, 2, \dots\}$  is a context-sensitive language, because it can be generated by a type 1 grammar, as Example 7 shows, but not by any type 2 language. (This is shown in Exercise 28 in the supplementary exercises at the end of the chapter.)

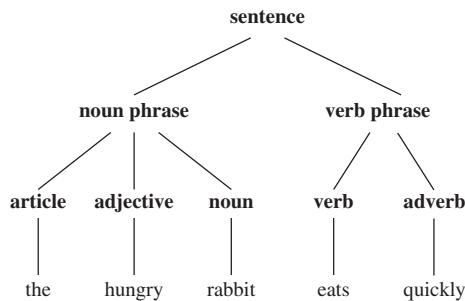
Table 1 summarizes the terminology used to classify phrase-structure grammars.

## Derivation Trees

A derivation in the language generated by a context-free grammar can be represented graphically using an ordered rooted tree, called a **derivation**, or **parse tree**. The root of this tree represents the starting symbol. The internal vertices of the tree represent the nonterminal symbols that arise in the derivation. The leaves of the tree represent the terminal symbols that arise. If the production  $A \rightarrow w$  arises in the derivation, where  $w$  is a word, the vertex that represents  $A$  has as children vertices that represent each symbol in  $w$ , in order from left to right.

**TABLE 1** Types of Grammars.

| Type | Restrictions on Productions $w_1 \rightarrow w_2$                                                                                                                                                        |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0    | No restrictions                                                                                                                                                                                          |
| 1    | $w_1 = lAr$ and $w_2 = lwr$ , where $A \in N$ , $l, r, w \in (N \cup T)^*$ and $w \neq \lambda$ ;<br>or $w_1 = S$ and $w_2 = \lambda$ as long as $S$ is not on the right-hand side of another production |
| 2    | $w_1 = A$ , where $A$ is a nonterminal symbol                                                                                                                                                            |
| 3    | $w_1 = A$ and $w_2 = aB$ or $w_2 = a$ , where $A \in N$ , $B \in N$ , and $a \in T$ ; or $w_1 = S$ and $w_2 = \lambda$                                                                                   |



**FIGURE 1 A Derivation Tree.**

**EXAMPLE 11** Construct a derivation tree for the derivation of *the hungry rabbit eats quickly*, given in the introduction of this section.

*Solution:* The derivation tree is shown in Figure 1. 

The problem of determining whether a string is in the language generated by a context-free grammar arises in many applications, such as in the construction of compilers. Two approaches to this problem are indicated in Example 12.

**EXAMPLE 12** Determine whether the word *cbab* belongs to the language generated by the grammar  $G = (V, T, S, P)$ , where  $V = \{a, b, c, A, B, C, S\}$ ,  $T = \{a, b, c\}$ ,  $S$  is the starting symbol, and the productions are

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow Ca \\ B &\rightarrow Ba \\ B &\rightarrow Cb \\ B &\rightarrow b \\ C &\rightarrow cb \\ C &\rightarrow b. \end{aligned}$$

*Solution:* One way to approach this problem is to begin with  $S$  and attempt to derive *cbab* using a series of productions. Because there is only one production with  $S$  on its left-hand side, we must start with  $S \Rightarrow AB$ . Next we use the only production that has  $A$  on its left-hand side, namely,  $A \Rightarrow Ca$ , to obtain  $S \Rightarrow AB \Rightarrow CaB$ . Because *cbab* begins with the symbols *cb*, we use the production  $C \Rightarrow cb$ . This gives us  $S \Rightarrow AB \Rightarrow CaB \Rightarrow cbaB$ . We finish by using the production  $B \Rightarrow b$ , to obtain  $S \Rightarrow AB \Rightarrow CaB \Rightarrow cbaB \Rightarrow cbab$ . The approach that we have used is called **top-down parsing**, because it begins with the starting symbol and proceeds by successively applying productions.

There is another approach to this problem, called **bottom-up parsing**. In this approach, we work backward. Because *cbab* is the string to be derived, we can use the production  $C \Rightarrow cb$ , so



**AVRAM NOAM CHOMSKY (BORN 1928)** Noam Chomsky, born in Philadelphia, is the son of a Hebrew scholar. He received his B.A., M.A., and Ph.D. in linguistics, all from the University of Pennsylvania. He was on the staff of the University of Pennsylvania from 1950 until 1951. In 1955 he joined the faculty at M.I.T., beginning his M.I.T. career teaching engineers French and German. Chomsky is currently the Ferrari P. Ward Professor of foreign languages and linguistics at M.I.T. He is known for his many fundamental contributions to linguistics, including the study of grammars. Chomsky is also widely known for his outspoken political activism.

that  $Cab \Rightarrow cbab$ . Then, we can use the production  $A \rightarrow Ca$ , so that  $Ab \Rightarrow Cab \Rightarrow cbab$ . Using the production  $B \rightarrow b$  gives  $AB \Rightarrow Ab \Rightarrow Cab \Rightarrow cbab$ . Finally, using  $S \rightarrow AB$  shows that a complete derivation for  $cbab$  is  $S \Rightarrow AB \Rightarrow Ab \Rightarrow Cab \Rightarrow cbab$ . 

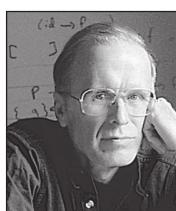
## Backus–Naur Form



The ancient Indian grammarian Pāṇini specified Sanskrit using 3959 rules; Backus–Naur form is sometimes called Backus–Pāṇini form.

There is another notation that is sometimes used to specify a type 2 grammar, called the **Backus–Naur form (BNF)**, after John Backus, who invented it, and Peter Naur, who refined it for use in the specification of the programming language ALGOL. (Surprisingly, a notation quite similar to the Backus–Naur form was used approximately 2500 years ago to describe the grammar of Sanskrit.) The Backus–Naur form is used to specify the syntactic rules of many computer languages, including Java. The productions in a type 2 grammar have a single nonterminal symbol as their left-hand side. Instead of listing all the productions separately, we can combine all those with the same nonterminal symbol on the left-hand side into one statement. Instead of using the symbol  $\rightarrow$  in a production, we use the symbol  $::=$ . We enclose all nonterminal symbols in brackets,  $\langle \rangle$ , and we list all the right-hand sides of productions in the same statement, separating them by bars. For instance, the productions  $A \rightarrow Aa$ ,  $A \rightarrow a$ , and  $A \rightarrow AB$  can be combined into  $\langle A \rangle ::= \langle A \rangle a \mid a \mid \langle A \rangle \langle B \rangle$ .

Example 13 illustrates how the Backus–Naur form is used to describe the syntax of programming languages. Our example comes from the original use of Backus–Naur form in the description of ALGOL 60.



**JOHN BACKUS (1924–2007)** John Backus was born in Philadelphia and grew up in Wilmington, Delaware. He attended the Hill School in Pottstown, Pennsylvania. He needed to attend summer school every year because he disliked studying and was not a serious student. But he enjoyed spending his summers in New Hampshire where he attended summer school and amused himself with summer activities, including sailing. He obliged his father by enrolling at the University of Virginia to study chemistry. But he quickly decided chemistry was not for him, and in 1943 he entered the army, where he received medical training and worked in a neurosurgery ward in an army hospital. Ironically, Backus was soon diagnosed with a bone tumor in his skull and was fitted with a metal plate. His medical work in the army convinced him to try medical school, but he abandoned this after nine months because he disliked the rote memorization required. After dropping out of medical school,

he entered a school for radio technicians because he wanted to build his own high fidelity set. A teacher in this school recognized his potential and asked him to help with some mathematical calculations needed for an article in a magazine. Finally, Backus found what he was interested in: mathematics and its applications. He enrolled at Columbia University, from which he received both bachelor's and master's degrees in mathematics. Backus joined IBM as a programmer in 1950. He participated in the design and development of two of IBM's early computers. From 1954 to 1958 he led the IBM group that developed FORTRAN. Backus became a staff member at the IBM Watson Research Center in 1958. He was part of the committees that designed the programming language ALGOL, using what is now called the Backus–Naur form for the description of the syntax of this language. Later, Backus worked on the mathematics of families of sets and on a functional style of programming. Backus became an IBM Fellow in 1963, and he received the National Medal of Science in 1974 and the prestigious Turing Award from the Association of Computing Machinery in 1977.



**PETER NAUR (BORN 1928)** Peter Naur was born in Frederiksberg, near Copenhagen. As a boy he became interested in astronomy. Not only did he observe heavenly bodies, but he also computed the orbits of comets and asteroids. Naur attended Copenhagen University, receiving his degree in 1949. He spent 1950 and 1951 in Cambridge, where he used an early computer to calculate the motions of comets and planets. After returning to Denmark he continued working in astronomy but kept his ties to computing. In 1955 he served as a consultant to the building of the first Danish computer. In 1959 Naur made the switch from astronomy to computing as a full-time activity. His first job as a full-time computer scientist was participating in the development of the programming language ALGOL. From 1960 to 1967 he worked on the development of compilers for ALGOL and COBOL. In 1969 he became professor of computer science at Copenhagen University, where he has worked in the area of programming methodology. His research interests include the design, structure, and performance of computer programs. Naur has been a pioneer in both the areas of software architecture and software engineering. He rejects the view that computer programming is a branch of mathematics and prefers that computer science be called *datalogy*.

**EXAMPLE 13**

In ALGOL 60 an identifier (which is the name of an entity such as a variable) consists of a string of alphanumeric characters (that is, letters and digits) and must begin with a letter. We can use these rules in Backus–Naur to describe the set of allowable identifiers:

$$\begin{aligned}\langle \text{identifier} \rangle &::= \langle \text{letter} \rangle \mid \langle \text{identifier} \rangle \langle \text{letter} \rangle \mid \langle \text{identifier} \rangle \langle \text{digit} \rangle \\ \langle \text{letter} \rangle &::= a \mid b \mid \dots \mid y \mid z \quad \text{the ellipsis indicates that all 26 letters are included} \\ \langle \text{digit} \rangle &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\end{aligned}$$

For example, we can produce the valid identifier  $x99a$  by using the first rule to replace  $\langle \text{identifier} \rangle$  by  $\langle \text{identifier} \rangle \langle \text{letter} \rangle$ , the second rule to obtain  $\langle \text{identifier} \rangle a$ , the first rule twice to obtain  $\langle \text{identifier} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle a$ , the third rule twice to obtain  $\langle \text{identifier} \rangle 99a$ , the first rule to obtain  $\langle \text{letter} \rangle 99a$ , and finally the second rule to obtain  $x99a$ . 

**EXAMPLE 14**

What is the Backus–Naur form of the grammar for the subset of English described in the introduction to this section?

*Solution:* The Backus–Naur form of this grammar is

$$\begin{aligned}\langle \text{sentence} \rangle &::= \langle \text{noun phrase} \rangle \langle \text{verb phrase} \rangle \\ \langle \text{noun phrase} \rangle &::= \langle \text{article} \rangle \langle \text{adjective} \rangle \langle \text{noun} \rangle \mid \langle \text{article} \rangle \langle \text{noun} \rangle \\ \langle \text{verb phrase} \rangle &::= \langle \text{verb} \rangle \langle \text{adverb} \rangle \mid \langle \text{verb} \rangle \\ \langle \text{article} \rangle &::= a \mid \text{the} \\ \langle \text{adjective} \rangle &::= \text{large} \mid \text{hungry} \\ \langle \text{noun} \rangle &::= \text{rabbit} \mid \text{mathematician} \\ \langle \text{verb} \rangle &::= \text{eats} \mid \text{hops} \\ \langle \text{adverb} \rangle &::= \text{quickly} \mid \text{wildly}\end{aligned}$$
**EXAMPLE 15**

Give the Backus–Naur form for the production of signed integers in decimal notation. (A **signed integer** is a nonnegative integer preceded by a plus sign or a minus sign.)

*Solution:* The Backus–Naur form for a grammar that produces signed integers is

$$\begin{aligned}\langle \text{signed integer} \rangle &::= \langle \text{sign} \rangle \langle \text{integer} \rangle \\ \langle \text{sign} \rangle &::= + \mid - \\ \langle \text{integer} \rangle &::= \langle \text{digit} \rangle \mid \langle \text{digit} \rangle \langle \text{integer} \rangle \\ \langle \text{digit} \rangle &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\end{aligned}$$

The Backus–Naur form, with a variety of extensions, is used extensively to specify the syntax of programming languages, such as Java and LISP; database languages, such as SQL; and markup languages, such as XML. Some extensions of the Backus–Naur form that are commonly used in the description of programming languages are introduced in the preamble to Exercise 34.

## Exercises

---

Exercises 1–3 refer to the grammar with start symbol **sentence**, set of terminals  $T = \{\text{the}, \text{sleepy}, \text{happy}, \text{tortoise}, \text{hare}, \text{passes}, \text{runs}, \text{quickly}, \text{slowly}\}$ , set of nonterminals  $N = \{\text{noun phrase}, \text{transitive verb phrase}, \text{intransitive verb phrase}, \text{article}, \text{adjective}, \text{noun}, \text{verb}, \text{adverb}\}$ , and productions:

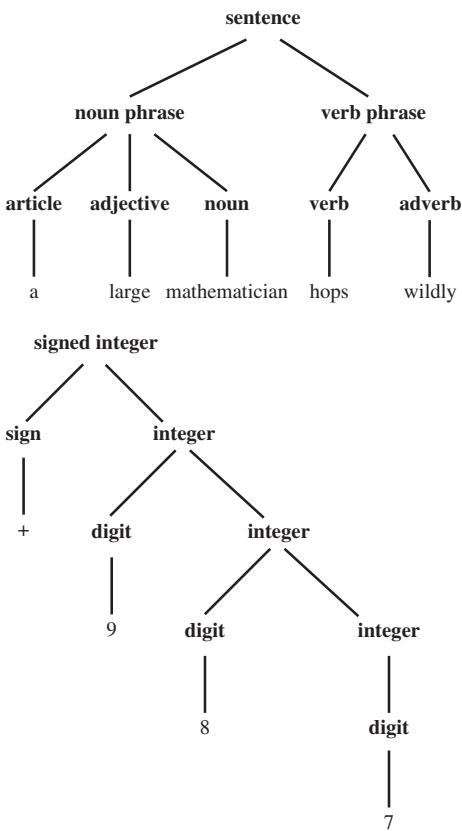
$$\begin{aligned}\text{sentence} &\rightarrow \text{noun phrase} \quad \text{transitive verb phrase} \\ \text{noun phrase} &\end{aligned}$$

$$\begin{aligned}\text{sentence} &\rightarrow \text{noun phrase} \quad \text{intransitive verb phrase} \\ \text{noun phrase} &\rightarrow \text{article} \quad \text{adjective} \quad \text{noun} \\ \text{noun phrase} &\rightarrow \text{article} \quad \text{noun} \\ \text{transitive verb phrase} &\rightarrow \text{transitive verb} \\ \text{intransitive verb phrase} &\rightarrow \text{intransitive verb} \quad \text{adverb} \\ \text{intransitive verb phrase} &\rightarrow \text{intransitive verb} \\ \text{article} &\rightarrow \text{the}\end{aligned}$$

**adjective** → *sleepy*  
**adjective** → *happy*  
**noun** → *tortoise*  
**noun** → *hare*  
**transitive verb** → *passes*  
**intransitive verb** → *runs*  
**adverb** → *quickly*  
**adverb** → *slowly*

1. Use the set of productions to show that each of these sentences is a valid sentence.
    - a) *the happy hare runs*
    - b) *the sleepy tortoise runs quickly*
    - c) *the tortoise passes the hare*
    - d) *the sleepy hare passes the happy tortoise*
  2. Find five other valid sentences, besides those given in Exercise 1.
  3. Show that *the hare runs the sleepy tortoise* is not a valid sentence.
  4. Let  $G = (V, T, S, P)$  be the phrase-structure grammar with  $V = \{0, 1, A, S\}$ ,  $T = \{0, 1\}$ , and set of productions  $P$  consisting of  $S \rightarrow 1S$ ,  $S \rightarrow 00A$ ,  $A \rightarrow 0A$ , and  $A \rightarrow 0$ .
    - a) Show that 111000 belongs to the language generated by  $G$ .
    - b) Show that 11001 does not belong to the language generated by  $G$ .
    - c) What is the language generated by  $G$ ?
  5. Let  $G = (V, T, S, P)$  be the phrase-structure grammar with  $V = \{0, 1, A, B, S\}$ ,  $T = \{0, 1\}$ , and set of productions  $P$  consisting of  $S \rightarrow 0A$ ,  $S \rightarrow 1A$ ,  $A \rightarrow 0B$ ,  $B \rightarrow 1A$ ,  $B \rightarrow 1$ .
    - a) Show that 10101 belongs to the language generated by  $G$ .
    - b) Show that 10110 does not belong to the language generated by  $G$ .
    - c) What is the language generated by  $G$ ?
  - \*6. Let  $V = \{S, A, B, a, b\}$  and  $T = \{a, b\}$ . Find the language generated by the grammar  $(V, T, S, P)$  when the set  $P$  of productions consists of
    - a)  $S \rightarrow AB$ ,  $A \rightarrow ab$ ,  $B \rightarrow bb$ .
    - b)  $S \rightarrow AB$ ,  $S \rightarrow aA$ ,  $A \rightarrow a$ ,  $B \rightarrow ba$ .
    - c)  $S \rightarrow AB$ ,  $S \rightarrow AA$ ,  $A \rightarrow aB$ ,  $A \rightarrow ab$ ,  $B \rightarrow b$ .
    - d)  $S \rightarrow AA$ ,  $S \rightarrow B$ ,  $A \rightarrow aaA$ ,  $A \rightarrow aa$ ,  $B \rightarrow bB$ ,  $B \rightarrow b$ .
    - e)  $S \rightarrow AB$ ,  $A \rightarrow aAb$ ,  $B \rightarrow bBa$ ,  $A \rightarrow \lambda$ ,  $B \rightarrow \lambda$ .
  7. Construct a derivation of  $0^31^3$  using the grammar given in Example 5.
  8. Show that the grammar given in Example 5 generates the set  $\{0^n1^n \mid n = 0, 1, 2, \dots\}$ .
  9. a) Construct a derivation of  $0^21^4$  using the grammar  $G_1$  in Example 6.  
 b) Construct a derivation of  $0^21^4$  using the grammar  $G_2$  in Example 6.
  10. a) Show that the grammar  $G_1$  given in Example 6 generates the set  $\{0^m1^n \mid m, n = 0, 1, 2, \dots\}$ .
- b) Show that the grammar  $G_2$  in Example 6 generates the same set.
  11. Construct a derivation of  $0^21^22^2$  in the grammar given in Example 7.
  - \*12. Show that the grammar given in Example 7 generates the set  $\{0^n1^n2^n \mid n = 0, 1, 2, \dots\}$ .
  13. Find a phrase-structure grammar for each of these languages.
    - a) the set consisting of the bit strings 0, 1, and 11
    - b) the set of bit strings containing only 1s
    - c) the set of bit strings that start with 0 and end with 1
    - d) the set of bit strings that consist of a 0 followed by an even number of 1s
  14. Find a phrase-structure grammar for each of these languages.
    - a) the set consisting of the bit strings 10, 01, and 101
    - b) the set of bit strings that start with 00 and end with one or more 1s
    - c) the set of bit strings consisting of an even number of 1s followed by a final 0
    - d) the set of bit strings that have neither two consecutive 0s nor two consecutive 1s
  - \*15. Find a phrase-structure grammar for each of these languages.
    - a) the set of all bit strings containing an even number of 0s and no 1s
    - b) the set of all bit strings made up of a 1 followed by an odd number of 0s
    - c) the set of all bit strings containing an even number of 0s and an even number of 1s
    - d) the set of all strings containing 10 or more 0s and no 1s
    - e) the set of all strings containing more 0s than 1s
    - f) the set of all strings containing an equal number of 0s and 1s
    - g) the set of all strings containing an unequal number of 0s and 1s
  16. Construct phrase-structure grammars to generate each of these sets.
    - a)  $\{1^n \mid n \geq 0\}$
    - b)  $\{10^n \mid n \geq 0\}$
    - c)  $\{(11)^n \mid n \geq 0\}$
  17. Construct phrase-structure grammars to generate each of these sets.
    - a)  $\{0^n \mid n \geq 0\}$
    - b)  $\{1^n0 \mid n \geq 0\}$
    - c)  $\{(000)^n \mid n \geq 0\}$
  18. Construct phrase-structure grammars to generate each of these sets.
    - a)  $\{01^{2n} \mid n \geq 0\}$
    - b)  $\{0^n1^{2n} \mid n \geq 0\}$
    - c)  $\{0^n1^m0^n \mid m \geq 0 \text{ and } n \geq 0\}$
  19. Let  $V = \{S, A, B, a, b\}$  and  $T = \{a, b\}$ . Determine whether  $G = (V, T, S, P)$  is a type 0 grammar but not a type 1 grammar, a type 1 grammar but not a type 2 grammar, or a type 2 grammar but not a type 3 grammar if  $P$ , the set of productions, is
    - a)  $S \rightarrow aAB$ ,  $A \rightarrow Bb$ ,  $B \rightarrow \lambda$ .

- b)  $S \rightarrow aA, A \rightarrow a, A \rightarrow b.$   
 c)  $S \rightarrow ABa, AB \rightarrow a.$   
 d)  $S \rightarrow ABA, A \rightarrow aB, B \rightarrow ab.$   
 e)  $S \rightarrow ba, A \rightarrow B, B \rightarrow a.$   
 f)  $S \rightarrow aA, aA \rightarrow B, B \rightarrow aA, A \rightarrow b.$   
 g)  $S \rightarrow bA, A \rightarrow b, S \rightarrow \lambda.$   
 h)  $S \rightarrow AB, B \rightarrow aAb, aAb \rightarrow b.$   
 i)  $S \rightarrow aA, A \rightarrow bB, B \rightarrow b, B \rightarrow \lambda.$   
 j)  $S \rightarrow A, A \rightarrow B, B \rightarrow \lambda.$
20. A **palindrome** is a string that reads the same backward as it does forward, that is, a string  $w$ , where  $w = w^R$ , where  $w^R$  is the reversal of the string  $w$ . Find a context-free grammar that generates the set of all palindromes over the alphabet  $\{0, 1\}$ .
- \*21. Let  $G_1$  and  $G_2$  be context-free grammars, generating the languages  $L(G_1)$  and  $L(G_2)$ , respectively. Show that there is a context-free grammar generating each of these sets.
- a)  $L(G_1) \cup L(G_2)$       b)  $L(G_1)L(G_2)$   
 c)  $L(G_1)^*$
22. Find the strings constructed using the derivation trees shown here.



23. Construct derivation trees for the sentences in Exercise 1.
24. Let  $G$  be the grammar with  $V = \{a, b, c, S\}$ ;  $T = \{a, b, c\}$ ; starting symbol  $S$ ; and productions  $S \rightarrow abS$ ,  $S \rightarrow bcS$ ,  $S \rightarrow bbS$ ,  $S \rightarrow a$ , and  $S \rightarrow cb$ . Construct derivation trees for
- a)  $bcbba.$   
 b)  $bbbcbba.$   
 c)  $bcabbbbcbb.$

- \*25. Use top-down parsing to determine whether each of the following strings belongs to the language generated by the grammar in Example 12.
- a)  $baba$       b)  $abab$   
 c)  $cbaba$       d)  $bbbcba$
- \*26. Use bottom-up parsing to determine whether the strings in Exercise 25 belong to the language generated by the grammar in Example 12.
27. Construct a derivation tree for  $-109$  using the grammar given in Example 15.
28. a) Explain what the productions are in a grammar if the Backus–Naur form for productions is as follows:
- ```

<expression> ::= ((expression)) |  

                  <expression> + <expression> |  

                  <expression> * <expression> |  

                  <variable>  

<variable> ::= x | y
  
```
- b) Find a derivation tree for $(x * y) + x$ in this grammar.
29. a) Construct a phrase-structure grammar that generates all signed decimal numbers, consisting of a sign, either $+$ or $-$; a nonnegative integer; and a decimal fraction that is either the empty string or a decimal point followed by a positive integer, where initial zeros in an integer are allowed.
 b) Give the Backus–Naur form of this grammar.
 c) Construct a derivation tree for -31.4 in this grammar.
30. a) Construct a phrase-structure grammar for the set of all fractions of the form a/b , where a is a signed integer in decimal notation and b is a positive integer.
 b) What is the Backus–Naur form for this grammar?
 c) Construct a derivation tree for $+311/17$ in this grammar.
31. Give production rules in Backus–Naur form for an identifier if it can consist of
- a) one or more lowercase letters.
 b) at least three but no more than six lowercase letters.
 c) one to six uppercase or lowercase letters beginning with an uppercase letter.
 d) a lowercase letter, followed by a digit or an underscore, followed by three or four alphanumeric characters (lower or uppercase letters and digits).
32. Give production rules in Backus–Naur form for the name of a person if this name consists of a first name, which is a string of letters, where only the first letter is uppercase; a middle initial; and a last name, which can be any string of letters.
33. Give production rules in Backus–Naur form that generate all identifiers in the C programming language. In C an identifier starts with a letter or an underscore ($_$) that is followed by one or more lowercase letters, uppercase letters, underscores, and digits.

 Several extensions to Backus–Naur form are commonly used to define phrase-structure grammars. In one such extension, a question mark (?) indicates that the symbol, or group of symbols inside parentheses, to its left can appear zero or once (that is, it is optional), an asterisk (*) indicates that the symbol to its left can appear zero or more times, and a plus (+) indicates that the symbol to its left can appear one or more times. These extensions are part of **extended Backus–Naur form (EBNF)**, and the symbols ?, *, and + are called **metacharacters**. In EBNF the brackets used to denote nonterminals are usually not shown.

34. Describe the set of strings defined by each of these sets of productions in EBNF.

a) $\text{string} ::= L + D?L +$

$$L ::= a \mid b \mid c$$

$$D ::= 0 \mid 1$$

b) $\text{string} ::= \text{sign } D + \mid D +$

$$\text{sign} ::= + \mid -$$

$$D ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

c) $\text{string} ::= L*(D+)?L*$

$$L ::= x \mid y$$

$$D ::= 0 \mid 1$$

35. Give production rules in extended Backus–Naur form that generate all decimal numerals consisting of an optional sign, a nonnegative integer, and a decimal fraction that is either the empty string or a decimal point followed by an optional positive integer optionally preceded by some number of zeros.

36. Give production rules in extended Backus–Naur form that generate a sandwich if a sandwich consists of a lower slice of bread; mustard or mayonnaise; optional lettuce; an optional slice of tomato; one or more slices of either turkey, chicken, or roast beef (in any combination); optionally some number of slices of cheese; and a top slice of bread.

37. Give production rules in extended Backus–Naur form for identifiers in the C programming language (see Exercise 33).

38. Describe how productions for a grammar in extended Backus–Naur form can be translated into a set of productions for the grammar in Backus–Naur form.

This is the Backus–Naur form that describes the syntax of expressions in postfix (or reverse Polish) notation.

$$\langle \text{expression} \rangle ::= \langle \text{term} \rangle \mid \langle \text{term} \rangle \langle \text{term} \rangle \langle \text{addOperator} \rangle$$

$$\langle \text{addOperator} \rangle ::= + \mid -$$

$$\langle \text{term} \rangle ::= \langle \text{factor} \rangle \mid \langle \text{factor} \rangle \langle \text{factor} \rangle \langle \text{mulOperator} \rangle$$

$$\langle \text{mulOperator} \rangle ::= * \mid /$$

$$\langle \text{factor} \rangle ::= \langle \text{identifier} \rangle \mid \langle \text{expression} \rangle$$

$$\langle \text{identifier} \rangle ::= a \mid b \mid \dots \mid z$$

39. For each of these strings, determine whether it is generated by the grammar given for postfix notation. If it is, find the steps used to generate the string

a) $abc*+$

b) $xy++$

c) $xy-z*$

d) $wxyz-* /$

e) $ade-*$

40. Use Backus–Naur form to describe the syntax of expressions in infix notation, where the set of operators and identifiers is the same as in the BNF for postfix expressions given in the preamble to Exercise 39, but parentheses must surround expressions being used as factors.

41. For each of these strings, determine whether it is generated by the grammar for infix expressions from Exercise 40. If it is, find the steps used to generate the string.

a) $x + y + z$

b) $a/b + c/d$

c) $m * (n + p)$

d) $+m - n + p - q$

e) $(m + n) * (p - q)$

42. Let G be a grammar and let R be the relation containing the ordered pair (w_0, w_1) if and only if w_1 is directly derivable from w_0 in G . What is the reflexive transitive closure of R ?

13.2 Finite-State Machines with Output

Introduction



Many kinds of machines, including components in computers, can be modeled using a structure called a finite-state machine. Several types of finite-state machines are commonly used in models. All these versions of finite-state machines include a finite set of states, with a designated starting state, an input alphabet, and a transition function that assigns a next state to every state and input pair. Finite-state machines are used extensively in applications in computer science and data networking. For example, finite-state machines are the basis for programs for spell checking, grammar checking, indexing or searching large bodies of text, recognizing speech, transforming text using markup languages such as XML and HTML, and network protocols that specify how computers communicate.

In this section, we will study those finite-state machines that produce output. We will show how finite-state machines can be used to model a vending machine, a machine that delays input, a machine that adds integers, and a machine that determines whether a bit string contains a specified pattern.

TABLE 1 State Table for a Vending Machine.

State	Next State					Output				
	Input					Input				
	5	10	25	O	R	5	10	25	O	R
s_0	s_1	s_2	s_5	s_0	s_0	n	n	n	n	n
s_1	s_2	s_3	s_6	s_1	s_1	n	n	n	n	n
s_2	s_3	s_4	s_6	s_2	s_2	n	n	5	n	n
s_3	s_4	s_5	s_6	s_3	s_3	n	n	10	n	n
s_4	s_5	s_6	s_6	s_4	s_4	n	n	15	n	n
s_5	s_6	s_6	s_6	s_5	s_5	n	5	20	n	n
s_6	s_6	s_6	s_6	s_0	s_0	5	10	25	OJ	AJ

Before giving formal definitions, we will show how a vending machine can be modeled. A vending machine accepts nickels (5 cents), dimes (10 cents), and quarters (25 cents). When a total of 30 cents or more has been deposited, the machine immediately returns the amount in excess of 30 cents. When 30 cents has been deposited and any excess refunded, the customer can push an orange button and receive an orange juice or push a red button and receive an apple juice. We can describe how the machine works by specifying its states, how it changes states when input is received, and the output that is produced for every combination of input and current state.

The machine can be in any of seven different states s_i , $i = 0, 1, 2, \dots, 6$, where s_i is the state where the machine has collected $5i$ cents. The machine starts in state s_0 , with 0 cents received. The possible inputs are 5 cents, 10 cents, 25 cents, the orange button (O), and the red button (R). The possible outputs are nothing (n), 5 cents, 10 cents, 15 cents, 20 cents, 25 cents, an orange juice, and an apple juice.

We illustrate how this model of the machine works with this example. Suppose that a student puts in a dime followed by a quarter, receives 5 cents back, and then pushes the orange button for an orange juice. The machine starts in state s_0 . The first input is 10 cents, which changes the state of the machine to s_2 and gives no output. The second input is 25 cents. This changes the state from s_2 to s_6 , and gives 5 cents as output. The next input is the orange button, which changes the state from s_6 back to s_0 (because the machine returns to the start state) and gives an orange juice as its output.

We can display all the state changes and output of this machine in a table. To do this we need to specify for each combination of state and input the next state and the output obtained. Table 1 shows the transitions and outputs for each pair of a state and an input.

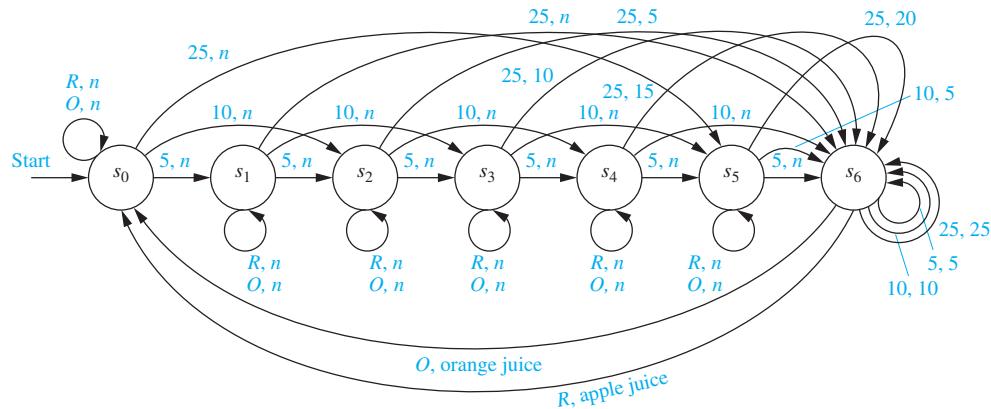
Another way to show the actions of a machine is to use a directed graph with labeled edges, where each state is represented by a circle, edges represent the transitions, and edges are labeled with the input and the output for that transition. Figure 1 shows such a directed graph for the vending machine.

Finite-State Machines with Outputs

We will now give the formal definition of a finite-state machine with output.

DEFINITION 1

A *finite-state machine* $M = (S, I, O, f, g, s_0)$ consists of a finite set S of *states*, a finite *input alphabet* I , a finite *output alphabet* O , a *transition function* f that assigns to each state and input pair a new state, an *output function* g that assigns to each state and input pair an output, and an *initial state* s_0 .

**FIGURE 1** A Vending Machine.

Let $M = (S, I, O, f, g, s_0)$ be a finite-state machine. We can use a **state table** to represent the values of the transition function f and the output function g for all pairs of states and input. We previously constructed a state table for the vending machine discussed in the introduction to this section.

EXAMPLE 1 The state table shown in Table 2 describes a finite-state machine with $S = \{s_0, s_1, s_2, s_3\}$, $I = \{0, 1\}$, and $O = \{0, 1\}$. The values of the transition function f are displayed in the first two columns, and the values of the output function g are displayed in the last two columns. \blacktriangleleft

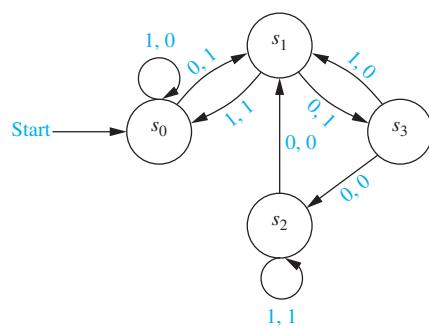
Another way to represent a finite-state machine is to use a **state diagram**, which is a directed graph with labeled edges. In this diagram, each state is represented by a circle. Arrows labeled with the input and output pair are shown for each transition.

EXAMPLE 2 Construct the state diagram for the finite-state machine with the state table shown in Table 2.

Solution: The state diagram for this machine is shown in Figure 2. \blacktriangleleft

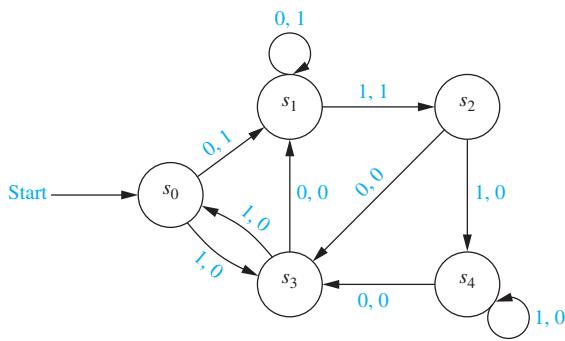
EXAMPLE 3 Construct the state table for the finite-state machine with the state diagram shown in Figure 3.

Solution: The state table for this machine is shown in Table 3. \blacktriangleleft



State	f		g	
	Input		Input	
	0	1	0	1
s_0	s_1	s_0	1	0
s_1	s_3	s_0	1	1
s_2	s_1	s_2	0	1
s_3	s_2	s_1	0	0

FIGURE 2 The State Diagram for the Finite-State Machine Shown in Table 2.

**TABLE 3**

State	f		g	
	Input		Input	
	0	1	0	1
s_0	s_1	s_3	1	0
s_1	s_1	s_2	1	1
s_2	s_3	s_4	0	0
s_3	s_1	s_0	0	0
s_4	s_3	s_4	0	0

FIGURE 3 A Finite-State Machine.

An input string takes the starting state through a sequence of states, as determined by the transition function. As we read the input string symbol by symbol (from left to right), each input symbol takes the machine from one state to another. Because each transition produces an output, an input string also produces an output string.

Suppose that the input string is $x = x_1x_2 \dots x_k$. Then, reading this input takes the machine from state s_0 to state s_1 , where $s_1 = f(s_0, x_1)$, then to state s_2 , where $s_2 = f(s_1, x_2)$, and so on, with $s_j = f(s_{j-1}, x_j)$ for $j = 1, 2, \dots, k$, ending at state $s_k = f(s_{k-1}, x_k)$. This sequence of transitions produces an output string $y_1y_2 \dots y_k$, where $y_1 = g(s_0, x_1)$ is the output corresponding to the transition from s_0 to s_1 , $y_2 = g(s_1, x_2)$ is the output corresponding to the transition from s_1 to s_2 , and so on. In general, $y_j = g(s_{j-1}, x_j)$ for $j = 1, 2, \dots, k$. Hence, we can extend the definition of the output function g to input strings so that $g(x) = y$, where y is the output corresponding to the input string x . This notation is useful in many applications.

EXAMPLE 4 Find the output string generated by the finite-state machine in Figure 3 if the input string is 101011.

Solution: The output obtained is 001000. The successive states and outputs are shown in Table 4. ◀

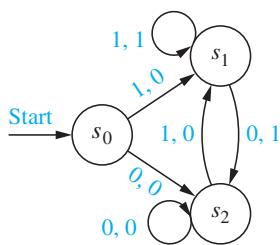
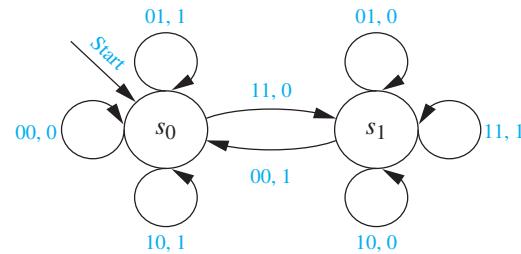
We can now look at some examples of useful finite-state machines. Examples 5, 6, and 7 illustrate that the states of a finite-state machine give it limited memory capabilities. The states can be used to remember the properties of the symbols that have been read by the machine. However, because there are only finitely many different states, finite-state machines cannot be used for some important purposes. This will be illustrated in Section 13.4.

EXAMPLE 5 An important element in many electronic devices is a *unit-delay machine*, which produces as output the input string delayed by a specified amount of time. How can a finite-state machine be constructed that delays an input string by one unit of time, that is, produces as output the bit string $0x_1x_2 \dots x_{k-1}$ given the input bit string $x_1x_2 \dots x_k$?

Solution: A delay machine can be constructed that has two possible inputs, namely, 0 and 1. The machine must have a start state s_0 . Because the machine has to remember whether the previous

TABLE 4

Input	1	0	1	0	1	1	—
State	s_0	s_3	s_1	s_2	s_3	s_0	s_3
Output	0	0	1	0	0	0	—

**FIGURE 4** A Unit-Delay Machine.**FIGURE 5** A Finite-State Machine for Addition.

input was a 0 or a 1, two other states s_1 and s_2 are needed, where the machine is in state s_1 if the previous input was 1 and in state s_2 if the previous input was 0. An output of 0 is produced for the initial transition from s_0 . Each transition from s_1 gives an output of 1, and each transition from s_2 gives an output of 0. The output corresponding to the input of a string $x_1 \dots x_k$ is the string that begins with 0, followed by x_1 , followed by x_2, \dots , ending with x_{k-1} . The state diagram for this machine is shown in Figure 4. \blacktriangleleft

EXAMPLE 6 Produce a finite-state machine that adds two positive integers using their binary expansions.



Solution: When $(x_n \dots x_1 x_0)_2$ and $(y_n \dots y_1 y_0)_2$ are added, the following procedure (as described in Section 4.2) is followed. First, the bits x_0 and y_0 are added, producing a sum bit z_0 and a carry bit c_0 . This carry bit is either 0 or 1. Then, the bits x_1 and y_1 are added, together with the carry c_0 . This gives a sum bit z_1 and a carry bit c_1 . This procedure is continued until the n th stage, where x_n , y_n , and the previous carry c_{n-1} are added to produce the sum bit z_n and the carry bit c_n , which is equal to the sum bit z_{n+1} .

A finite-state machine to carry out this addition can be constructed using just two states. For simplicity we assume that both the initial bits x_n and y_n are 0 (otherwise we have to make special arrangements concerning the sum bit z_{n+1}). The start state s_0 is used to remember that the previous carry is 0 (or for the addition of the rightmost bits). The other state, s_1 , is used to remember that the previous carry is 1.

Because the inputs to the machine are pairs of bits, there are four possible inputs. We represent these possibilities by 00 (when both bits are 0), 01 (when the first bit is 0 and the second is 1), 10 (when the first bit is 1 and the second is 0), and 11 (when both bits are 1). The transitions and the outputs are constructed from the sum of the two bits represented by the input and the carry represented by the state. For instance, when the machine is in state s_1 and receives 01 as input, the next state is s_1 and the output is 0, because the sum that arises is $0 + 1 + 1 = (10)_2$. The state diagram for this machine is shown in Figure 5. \blacktriangleleft

EXAMPLE 7 In a certain coding scheme, when three consecutive 1s appear in a message, the receiver of the message knows that there has been a transmission error. Construct a finite-state machine that gives a 1 as its current output bit if and only if the last three bits received are all 1s.

Solution: Three states are needed in this machine. The start state s_0 remembers that the previous input value, if it exists, was not a 1. The state s_1 remembers that the previous input was a 1, but the input before the previous input, if it exists, was not a 1. The state s_2 remembers that the previous two inputs were 1s.

An input of 1 takes s_0 to s_1 , because now a 1, and not two consecutive 1s, has been read; it takes s_1 to s_2 , because now two consecutive 1s have been read; and it takes s_2 to itself, because at least two consecutive 1s have been read. An input of 0 takes every state to s_0 , because this breaks up any string of consecutive 1s. The output for the transition from s_2 to itself when a 1

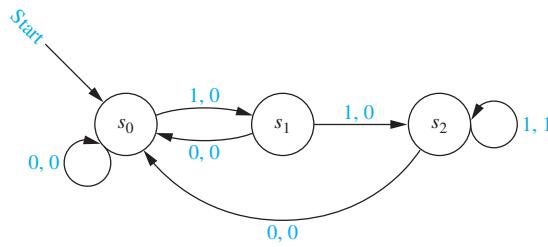


FIGURE 6 A Finite-State Machine That Gives an Output of 1 If and Only If the Input String Read So Far Ends with 111.

is read is 1, because this combination of input and state shows that three consecutive 1s have been read. All other outputs are 0. The state diagram of this machine is shown in Figure 6. \blacktriangleleft

The final output bit of the finite-state machine we constructed in Example 7 is 1 if and only if the input string ends with 111. Because of this, we say that this finite-state machine **recognizes** the set of bit strings that end with 111. This leads us to Definition 2.

DEFINITION 2

Let $M = (S, I, O, f, g, s_0)$ be a finite-state machine and $L \subseteq I^*$. We say that M *recognizes* (or *accepts*) L if an input string x belongs to L if and only if the last output bit produced by M when given x as input is a 1.

TYPES OF FINITE-STATE MACHINES Many different kinds of finite-state machines have been developed to model computing machines. In this section we have given a definition of one type of finite-state machine. In the type of machine introduced in this section, outputs correspond to transitions between states. Machines of this type are known as **Mealy machines**, because they were first studied by G. H. Mealy in 1955. There is another important type of finite-state machine with output, where the output is determined only by the state. This type of finite-state machine is known as a **Moore machine**, because E. F. Moore introduced this type of machine in 1956. Moore machines are considered in a sequence of exercises.

In Example 7 we showed how a Mealy machine can be used for language recognition. However, another type of finite-state machine, giving no output, is usually used for this purpose. Finite-state machines with no output, also known as finite-state automata, have a set of final states and recognize a string if and only if it takes the start state to a final state. We will study this type of finite-state machine in Section 13.3.

Exercises

1. Draw the state diagrams for the finite-state machines with these state tables.

a)

State	f		g	
	Input		Input	
	0	1	0	1
s_0	s_1	s_0	0	1
s_1	s_0	s_2	0	1
s_2	s_1	s_1	0	0

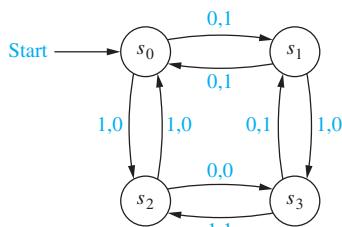
b)

State	f		g	
	Input		Input	
	0	1	0	1
s_0	s_1	s_0	0	0
s_1	s_2	s_0	1	1
s_2	s_0	s_3	0	1
s_3	s_1	s_2	1	0

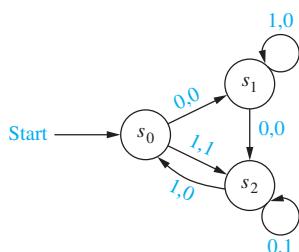
State	<i>f</i>		<i>g</i>	
	Input		Input	
	0	1	0	1
s_0	s_0	s_4	1	1
s_1	s_0	s_3	0	1
s_2	s_0	s_2	0	0
s_3	s_1	s_1	1	1
s_4	s_1	s_0	1	0

2. Give the state tables for the finite-state machines with these state diagrams.

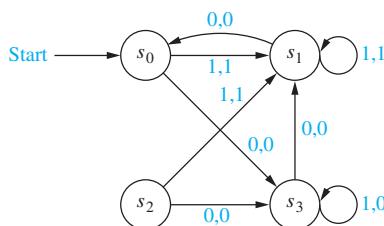
a)



b)



c)



3. Find the output generated from the input string 01110 for the finite-state machine with the state table in
- Exercise 1(a).
 - Exercise 1(b).
 - Exercise 1(c).
4. Find the output generated from the input string 10001 for the finite-state machine with the state diagram in
- Exercise 2(a).
 - Exercise 2(b).
 - Exercise 2(c).
5. Find the output for each of these input strings when given as input to the finite-state machine in Example 2.
- 0111
 - 11011011
 - 01010101010
6. Find the output for each of these input strings when given as input to the finite-state machine in Example 3.
- 0000
 - 101010
 - 11011100010

7. Construct a finite-state machine that models an old-fashioned soda machine that accepts nickels, dimes, and quarters. The soda machine accepts change until 35 cents has been put in. It gives change back for any amount greater than 35 cents. Then the customer can push buttons to receive either a cola, a root beer, or a ginger ale.

8. Construct a finite-state machine that models a newspaper vending machine that has a door that can be opened only after either three dimes (and any number of other coins) or a quarter and a nickel (and any number of other coins) have been inserted. Once the door can be opened, the customer opens it and takes a paper, closing the door. No change is ever returned no matter how much extra money has been inserted. The next customer starts with no credit.

9. Construct a finite-state machine that delays an input string two bits, giving 00 as the first two bits of output.

10. Construct a finite-state machine that changes every other bit, starting with the second bit, of an input string, and leaves the other bits unchanged.

11. Construct a finite-state machine for the log-on procedure for a computer, where the user logs on by entering a user identification number, which is considered to be a single input, and then a password, which is considered to be a single input. If the password is incorrect, the user is asked for the user identification number again.

12. Construct a finite-state machine for a combination lock that contains numbers 1 through 40 and that opens only when the correct combination, 10 right, 8 second left, 37 right, is entered. Each input is a triple consisting of a number, the direction of the turn, and the number of times the lock is turned in that direction.

13. Construct a finite-state machine for a toll machine that opens a gate after 25 cents, in nickels, dimes, or quarters, has been deposited. No change is given for overpayment, and no credit is given to the next driver when more than 25 cents has been deposited.

14. Construct a finite-state machine for entering a security code into an automatic teller machine (ATM) that implements these rules: A user enters a string of four digits, one digit at a time. If the user enters the correct four digits of the password, the ATM displays a welcome screen. When the user enters an incorrect string of four digits, the ATM displays a screen that informs the user that an incorrect password was entered. If a user enters the incorrect password three times, the account is locked.

15. Construct a finite-state machine for a restricted telephone switching system that implements these rules. Only calls to the telephone numbers 0, 911, and the digit 1 followed by 10-digit telephone numbers that begin with 212, 800, 866, 877, and 888 are sent to the network. All other strings of digits are blocked by the system and the user hears an error message.

16. Construct a finite-state machine that gives an output of 1 if the number of input symbols read so far is divisible by 3 and an output of 0 otherwise.

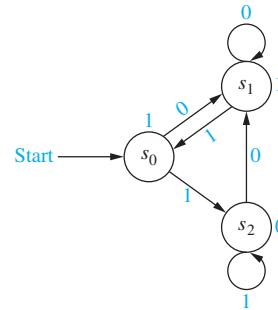
17. Construct a finite-state machine that determines whether the input string has a 1 in the last position and a 0 in the third to the last position read so far.
18. Construct a finite-state machine that determines whether the input string read so far ends in at least five consecutive 1s.
19. Construct a finite-state machine that determines whether the word *computer* has been read as the last eight characters in the input read so far, where the input can be any string of English letters.

A **Moore machine** $M = (S, I, O, f, g, s_0)$ consists of a finite set of states, an input alphabet I , an output alphabet O , a transition function f that assigns a next state to every pair of a state and an input, an output function g that assigns an output to every state, and a starting state s_0 . A Moore machine can be represented either by a table listing the transitions for each pair of state and input and the outputs for each state, or by a state diagram that displays the states, the transitions between states, and the output for each state. In the diagram, transitions are indicated with arrows labeled with the input, and the outputs are shown next to the states.

20. Construct the state diagram for the Moore machine with this state table.

State	f		g
	Input 0	1	
s_0	s_0	s_2	0
s_1	s_3	s_0	1
s_2	s_2	s_1	1
s_3	s_2	s_0	1

21. Construct the state table for the Moore machine with the state diagram shown here. Each input string to a Moore machine M produces an output string. In particular, the output corresponding to the input string $a_1a_2 \dots a_k$ is the string $g(s_0)g(s_1) \dots g(s_k)$, where $s_i = f(s_{i-1}, a_i)$ for $i = 1, 2, \dots, k$.



22. Find the output string generated by the Moore machine in Exercise 20 with each of these input strings.
 - a) 0101
 - b) 111111
 - c) 11101110111
23. Find the output string generated by the Moore machine in Exercise 21 with each of the input strings in Exercise 22.
24. Construct a Moore machine that gives an output of 1 whenever the number of symbols in the input string read so far is divisible by 4 and an output of 0 otherwise.
25. Construct a Moore machine that determines whether an input string contains an even or odd number of 1s. The machine should give 1 as output if an even number of 1s are in the string and 0 as output if an odd number of 1s are in the string.

13.3 Finite-State Machines with No Output

Introduction



One of the most important applications of finite-state machines is in language recognition. This application plays a fundamental role in the design and construction of compilers for programming languages. In Section 13.2 we showed that a finite-state machine with output can be used to recognize a language, by giving an output of 1 when a string from the language has been read and a 0 otherwise. However, there are other types of finite-state machines that are specially designed for recognizing languages. Instead of producing output, these machines have final states. A string is recognized if and only if it takes the starting state to one of these final states.

Set of Strings

Before discussing finite-state machines with no output, we will introduce some important background material on sets of strings. The operations that will be defined here will be used extensively in our discussion of language recognition by finite-state machines.

DEFINITION 1

Suppose that A and B are subsets of V^* , where V is a vocabulary. The *concatenation* of A and B , denoted by AB , is the set of all strings of the form xy , where x is a string in A and y is a string in B .

EXAMPLE 1 Let $A = \{0, 11\}$ and $B = \{1, 10, 110\}$. Find AB and BA .

Solution: The set AB contains every concatenation of a string in A and a string in B . Hence, $AB = \{01, 010, 0110, 111, 1110, 11110\}$. The set BA contains every concatenation of a string in B and a string in A . Hence, $BA = \{10, 111, 100, 1011, 1100, 11011\}$. \blacktriangleleft

Note that it is not necessarily the case that $AB = BA$ when A and B are subsets of V^* , where V is an alphabet, as Example 1 illustrates.

From the definition of the concatenation of two sets of strings, we can define A^n , for $n = 0, 1, 2, \dots$. This is done recursively by specifying that

$$\begin{aligned} A^0 &= \{\lambda\}, \\ A^{n+1} &= A^n A \quad \text{for } n = 0, 1, 2, \dots \end{aligned}$$

EXAMPLE 2 Let $A = \{1, 00\}$. Find A^n for $n = 0, 1, 2$, and 3 .

Solution: We have $A^0 = \{\lambda\}$ and $A^1 = A^0 A = \{\lambda\}A = \{1, 00\}$. To find A^2 we take concatenations of pairs of elements of A . This gives $A^2 = \{11, 100, 001, 0000\}$. To find A^3 we take concatenations of elements in A^2 and A ; this gives $A^3 = \{111, 1100, 1001, 10000, 0011, 00100, 00001, 000000\}$. \blacktriangleleft

DEFINITION 2

Suppose that A is a subset of V^* . Then the *Kleene closure* of A , denoted by A^* , is the set consisting of concatenations of arbitrarily many strings from A . That is, $A^* = \bigcup_{k=0}^{\infty} A^k$.

EXAMPLE 3 What are the Kleene closures of the sets $A = \{0\}$, $B = \{0, 1\}$, and $C = \{11\}$?

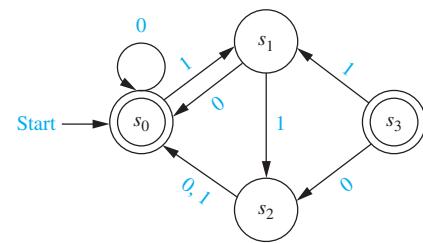
Solution: The Kleene closure of A is the concatenation of the string 0 with itself an arbitrary finite number of times. Hence, $A^* = \{0^n \mid n = 0, 1, 2, \dots\}$. The Kleene closure of B is the concatenation of an arbitrary number of strings, where each string is either 0 or 1. This is the set of all strings over the alphabet $V = \{0, 1\}$. That is, $B^* = V^*$. Finally, the Kleene closure of C is the concatenation of the string 11 with itself an arbitrary number of times. Hence, C^* is the set of strings consisting of an even number of 1s. That is, $C^* = \{1^{2n} \mid n = 0, 1, 2, \dots\}$. \blacktriangleleft

Finite-State Automata

] We will now give a definition of a finite-state machine with no output. Such machines are also called **finite-state automata**, and that is the terminology we will use for them here. (Note: The singular of *automata* is *automaton*.) These machines differ from the finite-state machines studied in Section 13.2 in that they do not produce output, but they do have a set of final states. As we will see, they recognize strings that take the starting state to a final state.

TABLE 1

State	<i>f</i>	
	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_0
s_3	s_2	s_1

**FIGURE 1** The State Diagram for a Finite-State Automaton.**DEFINITION 3**

A *finite-state automaton* $M = (S, I, f, s_0, F)$ consists of a finite set S of *states*, a finite *input alphabet* I , a *transition function* f that assigns a next state to every pair of state and input (so that $f : S \times I \rightarrow S$), an *initial* or *start state* s_0 , and a subset F of S consisting of *final* (or *accepting states*).

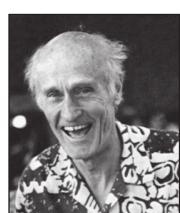
We can represent finite-state automata using either state tables or state diagrams. Final states are indicated in state diagrams by using double circles.

EXAMPLE 4 Construct the state diagram for the finite-state automaton $M = (S, I, f, s_0, F)$, where $S = \{s_0, s_1, s_2, s_3\}$, $I = \{0, 1\}$, $F = \{s_0, s_3\}$, and the transition function f is given in Table 1.

Solution: The state diagram is shown in Figure 1. Note that because both the inputs 0 and 1 take s_2 to s_0 , we write 0,1 over the edge from s_2 to s_0 . 

EXTENDING THE TRANSITION FUNCTION The transition function f of a finite-state machine $M = (S, I, f, s_0, F)$ can be extended so that it is defined for all pairs of states and strings; that is, f can be extended to a function $f : S \times I^* \rightarrow S$. Let $x = x_1x_2 \dots x_k$ be a string in I^* . Then $f(s_1, x)$ is the state obtained by using each successive symbol of x , from left to right, as input, starting with state s_1 . From s_1 we go on to state $s_2 = f(s_1, x_1)$, then to state $s_3 = f(s_2, x_2)$, and so on, with $f(s_1, x) = f(s_k, x_k)$. Formally, we can define this extended transition function f recursively for the deterministic finite-state machine $M = (S, I, f, s_0, F)$ by

- (i) $f(s, \lambda) = s$ for every state $s \in S$; and
- (ii) $f(s, xa) = f(f(s, x), a)$ for all $s \in S$, $x \in I^*$, and $a \in I$.



STEPHEN COLE KLEENE (1909–1994) Stephen Kleene was born in Hartford, Connecticut. His mother, Alice Lena Cole, was a poet, and his father, Gustav Adolph Kleene, was an economics professor. Kleene attended Amherst College and received his Ph.D. from Princeton in 1934, where he studied under the famous logician Alonzo Church. Kleene joined the faculty of the University of Wisconsin in 1935, where he remained except for several leaves, including stays at the Institute for Advanced Study in Princeton. During World War II he was a navigation instructor at the Naval Reserve's Midshipmen's School and later served as the director of the Naval Research Laboratory. Kleene made significant contributions to the theory of recursive functions, investigating questions of computability and decidability, and proved one of the central results of automata theory. He served as the Acting Director of the Mathematics Research Center and as Dean of the College of Letters and Sciences at the University of Wisconsin. Kleene was a student of natural history. He discovered a previously undescribed variety of butterfly that is named after him. He was an avid hiker and climber. Kleene was also noted as a talented teller of anecdotes, using a powerful voice that could be heard several offices away.

We can use structural induction and this recursive definition to prove properties of this extended transition function. For example, in Exercise 15 we ask you to prove that

$$f(s, xy) = f(f(s, x), y)$$

for every state $s \in S$ and strings $x \in I^*$ and $y \in I^*$.

Language Recognition by Finite-State Machines

Next, we define some terms that are used when studying the recognition by finite-state automata of certain sets of strings.

DEFINITION 4

A string x is said to be *recognized* or *accepted* by the machine $M = (S, I, f, s_0, F)$ if it takes the initial state s_0 to a final state, that is, $f(s_0, x)$ is a state in F . The *language recognized* or *accepted* by the machine M , denoted by $L(M)$, is the set of all strings that are recognized by M . Two finite-state automata are called *equivalent* if they recognize the same language.

In Example 5 we will find the languages recognized by several finite-state automata.

EXAMPLE 5 Determine the languages recognized by the finite-state automata M_1 , M_2 , and M_3 in Figure 2.

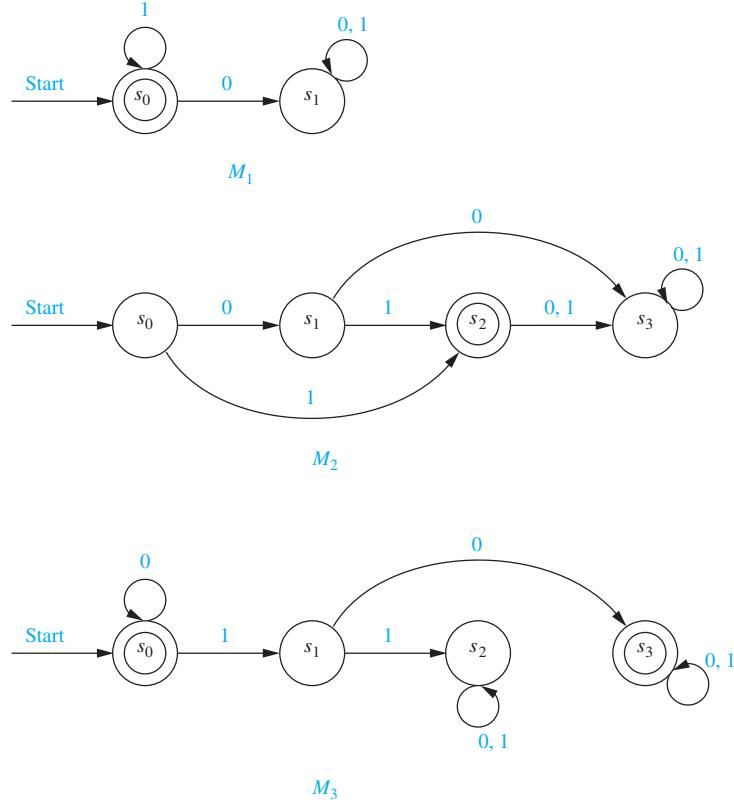


FIGURE 2 Some Finite-State Automata.

Solution: The only final state of M_1 is s_0 . The strings that take s_0 to itself are those consisting of zero or more consecutive 1s. Hence, $L(M_1) = \{1^n \mid n = 0, 1, 2, \dots\}$.

The only final state of M_2 is s_2 . The only strings that take s_0 to s_2 are 1 and 01. Hence, $L(M_2) = \{1, 01\}$.

The final states of M_3 are s_0 and s_3 . The only strings that take s_0 to itself are $\lambda, 0, 00, 000, \dots$, that is, any string of zero or more consecutive 0s. The only strings that take s_0 to s_3 are a string of zero or more consecutive 0s, followed by 10, followed by any string. Hence, $L(M_3) = \{0^n, 0^n 10x \mid n = 0, 1, 2, \dots, \text{and } x \text{ is any string}\}$. \blacktriangleleft

DESIGNING FINITE-STATE AUTOMATA We can often construct a finite-state automaton that recognizes a given set of strings by carefully adding states and transitions and determining which of these states should be final states. When appropriate we include states that can keep track of some of the properties of the input string, providing the finite-state automaton with limited memory. Examples 6 and 7 illustrate some of the techniques that can be used to construct finite-state automata that recognize particular types of sets of strings.

EXAMPLE 6 Construct deterministic finite-state automata that recognize each of these languages.



- the set of bit strings that begin with two 0s
- the set of bit strings that contain two consecutive 0s
- the set of bit strings that do not contain two consecutive 0s
- the set of bit strings that end with two 0s
- the set of bit strings that contain at least two 0s

Solution: (a) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that begin with two 0s. Besides the start state s_0 , we include a nonfinal state s_1 ; we move to s_1 from s_0 if the first bit is a 0. Next, we add a final state s_2 , which we move to from s_1 if the second bit is a 0. When we have reached s_2 we know that the first two input bits are both 0s, so we stay in the state s_2 no matter what the succeeding bits (if any) are. We move to a nonfinal state s_3 from s_0 if the first bit is a 1 and from s_1 if the second bit is a 1. The reader should verify that the finite-state automaton in Figure 3(a) recognizes the set of bit strings that begin with two 0s.

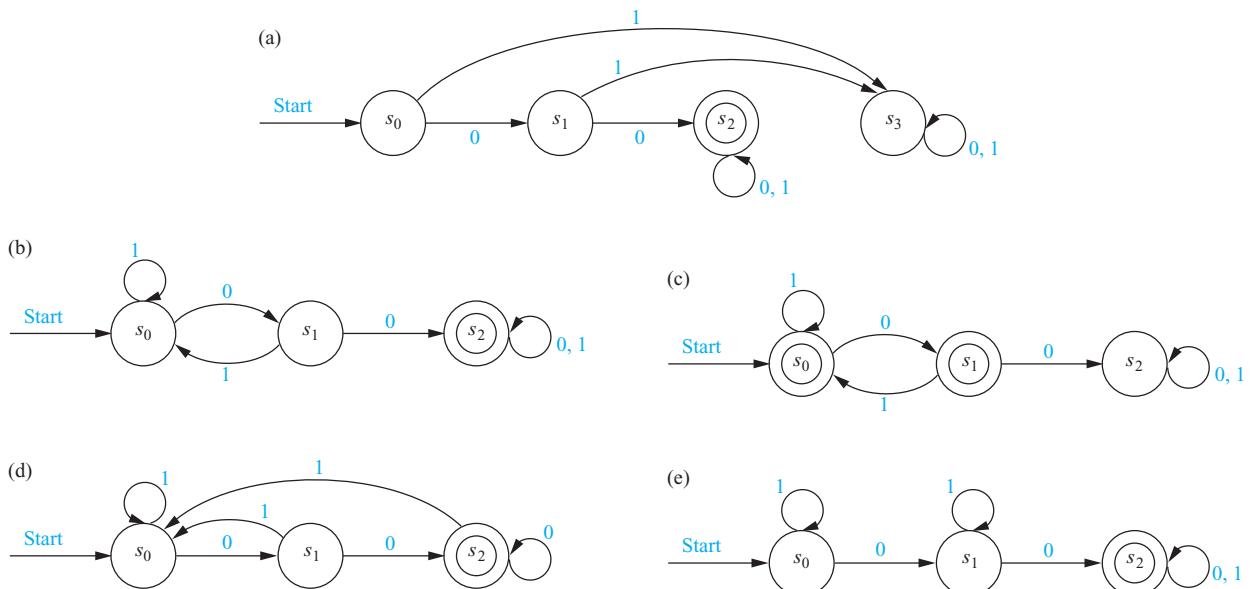


FIGURE 3 Deterministic Finite-State Automata Recognizing the Languages in Example 6.

(b) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that contain two consecutive 0s. Besides the start state s_0 , we include a nonfinal state s_1 , which tells us that the last input bit seen is a 0, but either the bit before it was a 1, or this bit was the initial bit of the string. We include a final state s_2 that we move to from s_1 when the next input bit after a 0 is also a 0. If a 1 follows a 0 in the string (before we encounter two consecutive 0s), we return to s_0 and begin looking for consecutive 0s all over again. The reader should verify that the finite-state automaton in Figure 3(b) recognizes the set of bit strings that contain two consecutive 0s.

(c) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that do not contain two consecutive 0s. Besides the start state s_0 , which should be a final state, we include a final state s_1 , which we move to from s_0 when 0 is the first input bit. When an input bit is a 1, we return to, or stay in, state s_0 . We add a state s_2 , which we move to from s_1 when the input bit is a 0. Reaching s_2 tells us that we have seen two consecutive 0s as input bits. We stay in state s_2 once we have reached it; this state is not final. The reader should verify that the finite-state automaton in Figure 3(c) recognizes the set of bit strings that do not contain two consecutive 0s. [The astute reader will notice the relationship between the finite-state automaton constructed here and the one constructed in part (b). See Exercise 39.]

(d) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that end with two 0s. Besides the start state s_0 , we include a nonfinal state s_1 , which we move to if the first bit is 0. We include a final state s_2 , which we move to from s_1 if the next input bit after a 0 is also a 0. If an input of 0 follows a previous 0, we stay in state s_2 because the last two input bits are still 0s. Once we are in state s_2 , an input bit of 1 sends us back to s_0 , and we begin looking for consecutive 0s all over again. We also return to s_0 if the next input is a 1 when we are in state s_1 . The reader should verify that the finite-state automaton in Figure 3(d) recognizes the set of bit strings that end with two 0s.

(e) Our goal is to construct a deterministic finite-state automaton that recognizes the set of bit strings that contain two 0s. Besides the start state, we include a state s_1 , which is not final; we stay in s_0 until an input bit is a 0 and we move to s_1 when we encounter the first 0 bit in the input. We add a final state s_2 , which we move to from s_1 once we encounter a second 0 bit. Whenever we encounter a 1 as input, we stay in the current state. Once we have reached s_2 , we remain there. Here, s_1 and s_2 are used to tell us that we have already seen one or two 0s in the input string so far, respectively. The reader should verify that the finite-state automaton in Figure 3(e) recognizes the set of bit strings that contain two 0s. 

EXAMPLE 7

Construct a deterministic finite-state automaton that recognizes the set of bit strings that contain an odd number of 1s and that end with at least two consecutive 0s.

Solution: We can build a deterministic finite-state automaton that recognizes the specified set by including states that keep track of both the parity of the number of 1 bits and whether we have seen no, one, or at least two 0s at the end of the input string.

The start state s_0 can be used to tell us that the input read so far contains an even number of 1s and ends with no 0s (that is, is empty or ends with a 1). Besides the start state, we include five more states. We move to states s_1, s_2, s_3, s_4 , and s_5 , respectively, when the input string read so far contains an even number of 1s and ends with one 0; when it contains an even number of 1s and ends with at least two 0s; when it contains an odd number of 1s and ends with no 0s; when it contains an odd number of 1s and ends with one 0; and when it contains an odd number of 1s and ends with two 0s. The state s_5 is a final state.

The reader should verify that the finite-state automaton in Figure 4 recognizes the set of bit strings that contain an odd number of 1s and end with at least two consecutive 0s. 

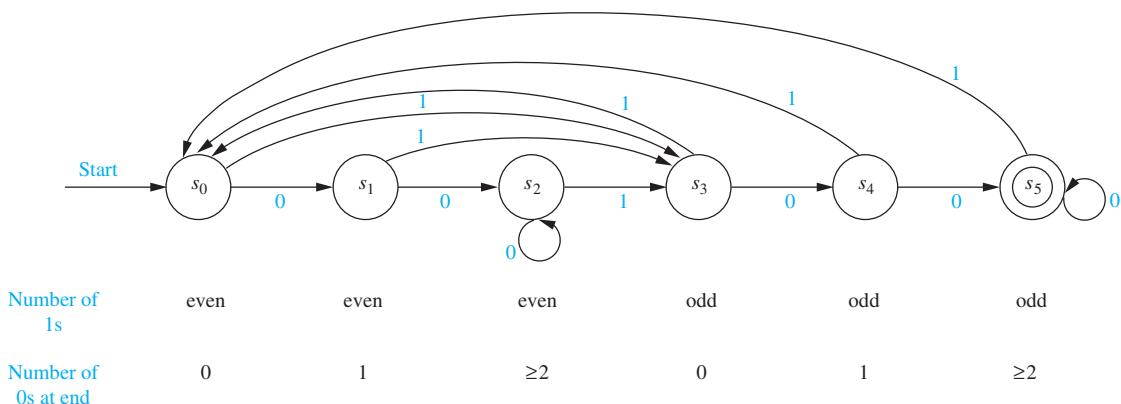


FIGURE 4 A Deterministic Finite-State Automaton Recognizing the Set of Bit Strings Containing an Odd Number of 1s and Ending with at Least Two 0s.

EQUIVALENT FINITE-STATE AUTOMATA In Definition 4 we specified that two finite-state automata are equivalent if they recognize the same language. Example 8 provides an example of two equivalent deterministic finite-state machines.

EXAMPLE 8 Show that the two finite-state automata M_0 and M_1 shown in Figure 5 are equivalent.

Solution: For a string x to be recognized by M_0 , x must take us from s_0 to the final state s_1 or the final state s_4 . The only string that takes us from s_0 to s_1 is the string 1. The strings that take us from s_0 to s_4 are those strings that begin with a 0, which takes us from s_0 to s_2 , followed by zero or more additional 0s, which keep the machine in state s_2 , followed by a 1, which takes us from state s_2 to the final state s_4 . All other strings take us from s_0 to a state that is not final. (We leave it to the reader to fill in the details.) We conclude that $L(M_0)$ is the set of strings of zero or more 0 bits followed by a final 1.

For a string x to be recognized by M_1 , x must take us from s_0 to the final state s_1 . So, for x to be recognized, it must begin with some number of 0s, which leave us in state s_0 , followed by

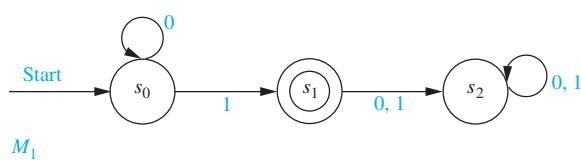
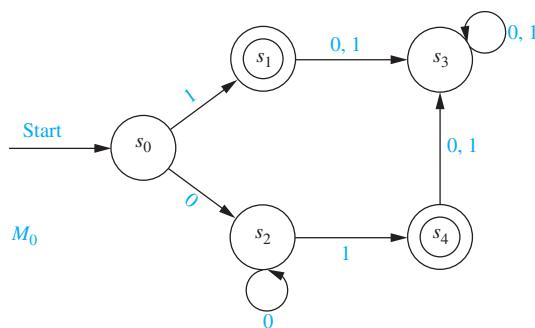


FIGURE 5 M_0 and M_1 Are Equivalent Finite-State Automata.

a 1, which takes us to the final state s_1 . A string of all zeros is not recognized because it leaves us in state s_0 , which is not final. All strings that contain a 0 after 1 are not recognized because they take us to state s_2 , which is not final. It follows that $L(M_1)$ is the same as $L(M_0)$. We conclude that M_0 and M_1 are equivalent.

Note that the finite-state machine M_1 only has three states. No finite state machine with fewer than three states can be used to recognize the set of all strings of zero or more 0 bits followed by a 1 (see Exercise 37). 

As Example 8 shows, a finite-state automaton may have more states than one equivalent to it. In fact, algorithms used to construct finite-state automata to recognize certain languages may have many more states than necessary. Using unnecessarily large finite-state machines to recognize languages can make both hardware and software applications inefficient and costly. This problem arises when finite-state automata are used in compilers, which translate computer programs to a language a computer can understand (object code).

Exercises 58–61 develop a procedure that constructs a finite-state automaton with the fewest states possible among all finite-state automata equivalent to a given finite-state automaton. This procedure is known as **machine minimization**. The minimization procedure described in these exercises reduces the number of states by replacing states with equivalence classes of states with respect to an equivalence relation in which two states are equivalent if every input string either sends both states to a final state or sends both to a state that is not final. Before the minimization procedure begins, all states that cannot be reached from the start state using any input string are first removed; removing these does not change the language recognized.



GRACE BREWSTER MURRAY HOPPER (1906–1992) Grace Hopper, born in New York City, displayed an intense curiosity as a child with how things worked. At the age of seven, she disassembled alarm clocks to discover their mechanisms. She inherited her love of mathematics from her mother, who received special permission to study geometry (but not algebra and trigonometry) at a time when women were actively discouraged from such study. Hopper was inspired by her father, a successful insurance broker, who had lost his legs from circulatory problems. He told his children they could do anything if they put their minds to it. He inspired Hopper to pursue higher education and not conform to the usual roles for females. Her parents made sure that she had an excellent education; she attended private schools for girls in New York. Hopper entered Vassar College in 1924, where she majored in mathematics and physics; she graduated in 1928. She received masters degree in mathematics from Yale University in 1930. In 1930 she also married an English instructor at the New York School of Commerce; she later divorced and did not have children. Hopper was a mathematics professor at Vassar from 1931 until 1943, earning a Ph.D. from Yale in 1934.

After the attack on Pearl Harbor, Hopper, coming from a family with strong military traditions, decided to leave her academic position and join the Navy WAVES. To enlist, she needed special permission to leave her strategic position as a mathematics professor, as well as a waiver for weighing too little. In December 1943, she was sworn into the Navy Reserve and trained at the Midshipman's School for Women. Hopper was assigned to work at the Naval Ordnance Laboratory] at Harvard University. She wrote programs for the world's first large-scale automatically sequenced digital computer, which was used to help aim Navy artillery in varying weather. Hopper has been credited with coining the term "bug" to refer to a hardware glitch, but it was used at Harvard prior to her arrival there. However, it is true that Hopper and her programming team found a moth in one of the relays in the computer hardware that shut the system down. This famous moth was pasted into a lab book. In the 1950s Hopper coined the term "debug" for the process of removing programming errors.

In 1946, when the Navy told her that she was too old for active service, Hopper chose to remain at Harvard as a civilian research fellow. In 1949 she left Harvard to join the Eckert–Mauchly Computer Corporation, where she helped develop the first commercial computer, UNIVAC. Hopper remained with this company when it was taken over by Remington Rand and when Remington Rand merged with the Sperry Corporation. She was a visionary for the potential power of computers; she understood that computers would become widely used if tools that were both programmer-friendly and application-friendly could be developed. In particular, she believed that computer programs could be written in English, rather than using machine instructions. To help achieve this goal, she developed the first compiler. She published the first research paper on compilers in 1952. Hopper is also known as the mother of the computer language COBOL; members of Hopper's staff helped to frame the basic language design for COBOL using their earlier work as a basis.

In 1966, Hopper retired from the Navy Reserve. However, only seven months later, the Navy recalled her from retirement to help standardize high-level naval computer languages. In 1983 she was promoted to the rank of Commodore by special Presidential appointment, and in 1985 she was elevated to the rank of Rear Admiral. Her retirement from the Navy, at the age of 80, was held on the *USS Constitution*.

Nondeterministic Finite-State Automata

The finite-state automata discussed so far are **deterministic**, because for each pair of state and input value there is a unique next state given by the transition function. There is another important type of finite-state automaton in which there may be several possible next states for each pair of input value and state. Such machines are called **nondeterministic**. Nondeterministic finite-state automata are important in determining which languages can be recognized by a finite-state automaton.

DEFINITION 5



A nondeterministic finite-state automaton $M = (S, I, f, s_0, F)$ consists of a set S of states, an input alphabet I , a transition function f that assigns a set of states to each pair of state and input (so that $f : S \times I \rightarrow P(S)$), a starting state s_0 , and a subset F of S consisting of the final states.

We can represent nondeterministic finite-state automata using state tables or state diagrams. When we use a state table, for each pair of state and input value we give a list of possible next states. In the state diagram, we include an edge from each state to all possible next states, labeling edges with the input or inputs that lead to this transition.

EXAMPLE 9 Find the state diagram for the nondeterministic finite-state automaton with the state table shown in Table 2. The final states are s_2 and s_3 .

Solution: The state diagram for this automaton is shown in Figure 6.

EXAMPLE 10 Find the state table for the nondeterministic finite-state automaton with the state diagram shown in Figure 7.

Solution: The state table is given as Table 3.

What does it mean for a nondeterministic finite-state automaton to recognize a string $x = x_1x_2 \dots x_k$? The first input symbol x_1 takes the starting state s_0 to a set S_1 of states. The next input symbol x_2 takes each of the states in S_1 to a set of states. Let S_2 be the union of these sets. We continue this process, including at a stage all states obtained using a state obtained at the previous stage and the current input symbol. We **recognize**, or **accept**, the string x if there is a final state in the set of all states that can be obtained from s_0 using x . The **language recognized** by a nondeterministic finite-state automaton is the set of all strings recognized by this automaton.

TABLE 2

State	f	
	Input	
	0	1
s_0	s_0, s_1	s_3
s_1	s_0	s_1, s_3
s_2		s_0, s_2
s_3	s_0, s_1, s_2	s_1

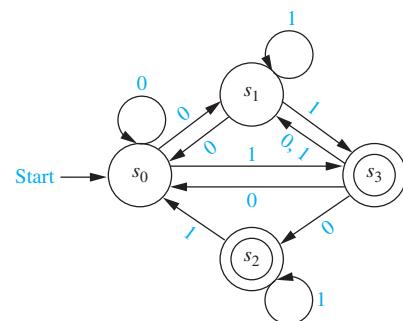


FIGURE 6 The Nondeterministic Finite-State Automaton with State Table Given in Table 2.

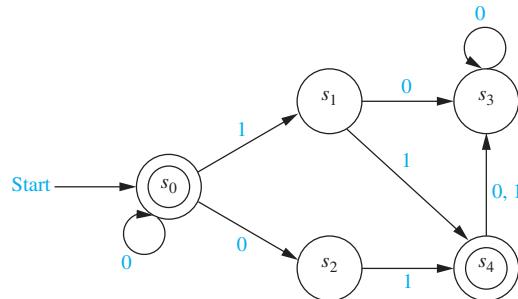


FIGURE 7 A Nondeterministic Finite-State Automaton.

TABLE 3		
State	f	
	0	1
s_0	s_0, s_2	s_1
s_1	s_3	s_4
s_2		s_4
s_3	s_3	
s_4	s_3	s_3

EXAMPLE 11 Find the language recognized by the nondeterministic finite-state automaton shown in Figure 7.

Solution: Because s_0 is a final state, and there is a transition from s_0 to itself when 0 is the input, the machine recognizes all strings consisting of zero or more consecutive 0s. Furthermore, because s_4 is a final state, any string that has s_4 in the set of states that can be reached from s_0 with this input string is recognized. The only such strings are strings consisting of zero or more consecutive 0s followed by 01 or 11. Because s_0 and s_4 are the only final states, the language recognized by the machine is $\{0^n, 0^n01, 0^n11 \mid n \geq 0\}$. \blacktriangleleft

One important fact is that a language recognized by a nondeterministic finite-state automaton is also recognized by a deterministic finite-state automaton. We will take advantage of this fact in Section 13.4 when we will determine which languages are recognized by finite-state automata.

THEOREM 1

If the language L is recognized by a nondeterministic finite-state automaton M_0 , then L is also recognized by a deterministic finite-state automaton M_1 .



Proof: We will describe how to construct the deterministic finite-state automaton M_1 that recognizes L from M_0 , the nondeterministic finite-state automaton that recognizes this language. Each state in M_1 will be made up of a set of states in M_0 . The start symbol of M_1 is $\{s_0\}$, which is the set containing the start state of M_0 . The input set of M_1 is the same as the input set of M_0 .

Given a state $\{s_{i_1}, s_{i_2}, \dots, s_{i_k}\}$ of M_1 , the input symbol x takes this state to the union of the sets of next states for the elements of this set, that is, the union of the sets $f(s_{i_1}, x), f(s_{i_2}, x), \dots, f(s_{i_k}, x)$. The states of M_1 are all the subsets of S , the set of states of M_0 , that are obtained in this way starting with s_0 . (There are as many as 2^n states in the deterministic machine, where n is the number of states in the nondeterministic machine, because all subsets may occur as states, including the empty set, although usually far fewer states occur.) The final states of M_1 are those sets that contain a final state of M_0 .

Suppose that an input string is recognized by M_0 . Then one of the states that can be reached from s_0 using this input string is a final state (the reader should provide an inductive proof of this). This means that in M_1 , this input string leads from $\{s_0\}$ to a set of states of M_0 that contains a final state. This subset is a final state of M_1 , so this string is also recognized by M_1 . Also, an input string not recognized by M_0 does not lead to any final states in M_0 . (The reader should provide the details that prove this statement.) Consequently, this input string does not lead from $\{s_0\}$ to a final state in M_1 . \blacktriangleleft

EXAMPLE 12 Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in Example 10.

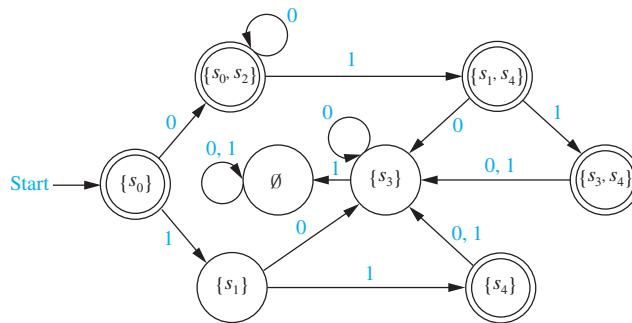


FIGURE 8 A Deterministic Automaton Equivalent to the Nondeterministic Automaton in Example 10.

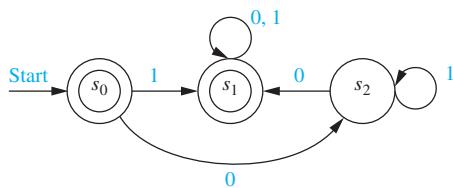
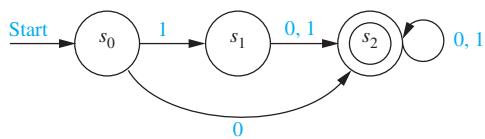
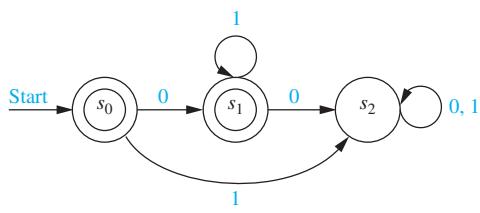
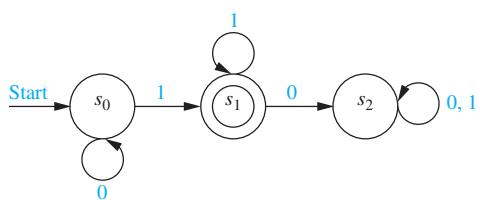
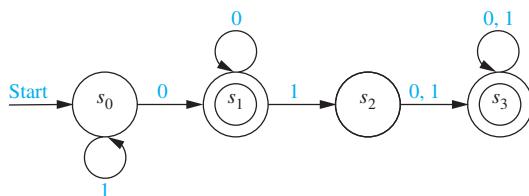
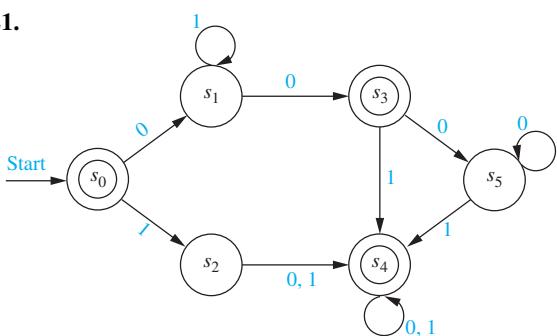
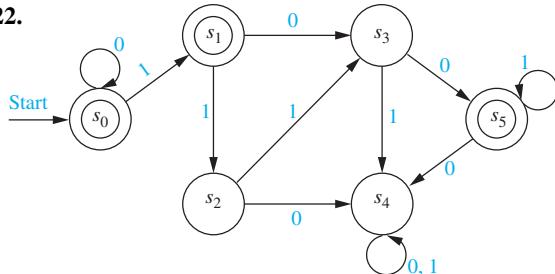
Solution: The deterministic automaton shown in Figure 8 is constructed from the nondeterministic automaton in Example 10. The states of this deterministic automaton are subsets of the set of all states of the nondeterministic machine. The next state of a subset under an input symbol is the subset containing the next states in the nondeterministic machine of all elements in this subset. For instance, on input of 0, $\{s_0\}$ goes to $\{s_0, s_2\}$, because s_0 has transitions to itself and to s_2 in the nondeterministic machine; the set $\{s_0, s_2\}$ goes to $\{s_1, s_4\}$ on input of 1, because s_0 goes just to s_1 and s_2 goes just to s_4 on input of 1 in the nondeterministic machine; and the set $\{s_1, s_4\}$ goes to $\{s_3\}$ on input of 0, because s_1 and s_4 both go to just s_3 on input of 0 in the deterministic machine. All subsets that are obtained in this way are included in the deterministic finite-state machine. Note that the empty set is one of the states of this machine, because it is the subset containing all the next states of $\{s_3\}$ on input of 1. The start state is $\{s_0\}$, and the set of final states are all those that include s_0 or s_4 . \blacktriangleleft

Exercises

1. Let $A = \{0, 11\}$ and $B = \{00, 01\}$. Find each of these sets.
 a) AB b) BA c) A^2 d) B^3
2. Show that if A is a set of strings, then $A\emptyset = \emptyset A = \emptyset$.
3. Find all pairs of sets of strings A and B for which $AB = \{10, 111, 1010, 1000, 10111, 101000\}$.
4. Show that these equalities hold.
 a) $\{\lambda\}^* = \{\lambda\}$
 b) $(A^*)^* = A^*$ for every set of strings A
5. Describe the elements of the set A^* for these values of A .
 a) $\{10\}$ b) $\{111\}$ c) $\{0, 01\}$ d) $\{1, 101\}$
6. Let V be an alphabet, and let A and B be subsets of V^* . Show that $|AB| \leq |A||B|$.
7. Let V be an alphabet, and let A and B be subsets of V^* with $A \subseteq B$. Show that $A^* \subseteq B^*$.
8. Suppose that A is a subset of V^* , where V is an alphabet. Prove or disprove each of these statements.
 a) $A \subseteq A^2$ b) if $A = A^2$, then $\lambda \in A$
 c) $A\{\lambda\} = A$ d) $(A^*)^* = A^*$
 e) $A^*A = A^*$ f) $|A^n| = |A|^n$
9. Determine whether the string 11101 is in each of these sets.
 a) $\{0, 1\}^*$ b) $\{1\}^*\{0\}^*\{1\}^*$
10. Determine whether the string 01001 is in each of these sets.
 c) $\{11\}\{0\}^*\{01\}$ d) $\{11\}^*\{01\}^*$
 e) $\{111\}^*\{0\}^*\{1\}$ f) $\{11, 0\}\{00, 101\}$
11. Determine whether each of these strings is recognized by the deterministic finite-state automaton in Figure 1.
 a) 111 b) 0011 c) 1010111 d) 011011011
12. Determine whether each of these strings is recognized by the deterministic finite-state automaton in Figure 1.
 a) 010 b) 1101 c) 1111110 d) 010101010
13. Determine whether all the strings in each of these sets are recognized by the deterministic finite-state automaton in Figure 1.
 a) $\{0\}^*$ b) $\{0\}\{0\}^*$ c) $\{1\}\{0\}^*$
 d) $\{01\}^*$ e) $\{0\}^*\{1\}^*$ f) $\{1\}\{0, 1\}^*$
14. Show that if $M = (S, I, f, s_0, F)$ is a deterministic finite-state automaton and $f(s, x) = s$ for the state $s \in S$ and the input string $x \in I^*$, then $f(s, x^n) = s$ for every non-negative integer n . (Here x^n is the concatenation of n copies of the string x , defined recursively in Exercise 37 in Section 5.3.)

- 15.** Given a deterministic finite-state automaton $M = (S, I, f, s_0, F)$, use structural induction and the recursive definition of the extended transition function f to prove that $f(s, xy) = f(f(s, x), y)$ for all states $s \in S$ and all strings $x \in I^*$ and $y \in I^*$.

In Exercises 16–22 find the language recognized by the given deterministic finite-state automaton.

16.**17.****18.****19.****20.****21.****22.**

23. Construct a deterministic finite-state automaton that recognizes the set of all bit strings beginning with 01.

24. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that end with 10.

25. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain the string 101.

26. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that do not contain three consecutive 0s.

27. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain exactly three 0s.

28. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain at least three 0s.

29. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain three consecutive 1s.

30. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that begin with 0 or with 11.

31. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that begin and end with 11.

32. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain an even number of 1s.

33. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain an odd number of 0s.

34. Construct a deterministic finite-state automaton that recognizes the set of all bit strings that contain an even number of 0s and an odd number of 1s.

35. Construct a finite-state automaton that recognizes the set of bit strings consisting of a 0 followed by a string with an odd number of 1s.

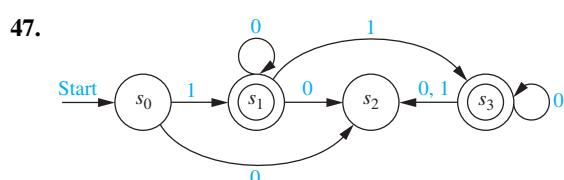
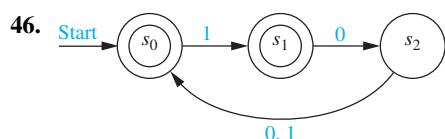
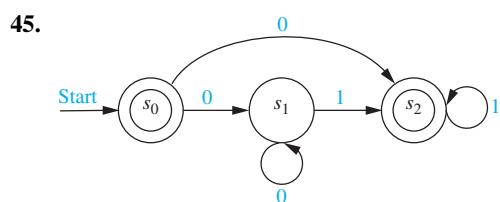
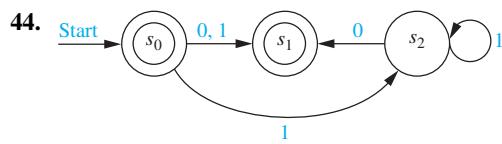
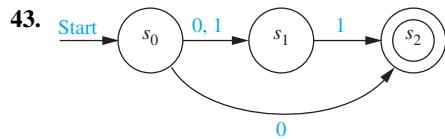
36. Construct a finite-state automaton with four states that recognizes the set of bit strings containing an even number of 1s and an odd number of 0s.

37. Show that there is no finite-state automaton with two states that recognizes the set of all bit strings that have one or more 1 bits and end with a 0.

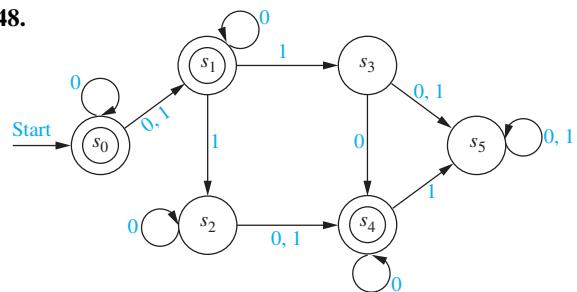
38. Show that there is no finite-state automaton with three states that recognizes the set of bit strings containing an even number of 1s and an even number of 0s.

39. Explain how you can change the deterministic finite-state automaton M so that the changed automaton recognizes the set $I^* - L(M)$.
40. Use Exercise 39 and finite-state automata constructed in Example 6 to find deterministic finite-state automata that recognize each of these sets.
- the set of bit strings that do not begin with two 0s
 - the set of bit strings that do not end with two 0s
 - the set of bit strings that contain at most one 0 (that is, that do not contain at least two 0s)
41. Use the procedure you described in Exercise 39 and the finite-state automaton you constructed in Exercise 25 to find a deterministic finite-state automaton that recognizes the set of all bit strings that do not contain the string 101.
42. Use the procedure you described in Exercise 39 and the finite-state automaton you constructed in Exercise 29 to find a deterministic finite-state automaton that recognizes the set of all bit strings that do not contain three consecutive 1s.

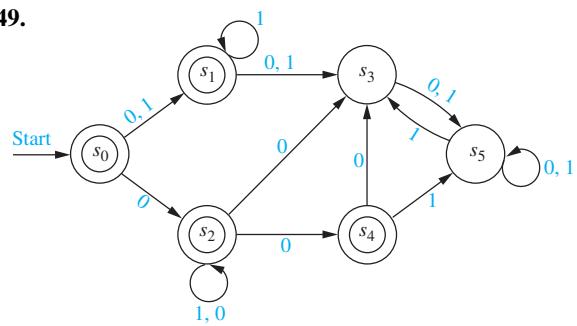
In Exercises 43–49 find the language recognized by the given nondeterministic finite-state automaton.



48.



49.



50. Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in Exercise 43.

51. Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in Exercise 44.

52. Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in Exercise 45.

53. Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in Exercise 46.

54. Find a deterministic finite-state automaton that recognizes the same language as the nondeterministic finite-state automaton in Exercise 47.

55. Find a deterministic finite-state automaton that recognizes each of these sets.

- $\{0\}$
- $\{1, 00\}$
- $\{1^n \mid n = 2, 3, 4, \dots\}$

56. Find a nondeterministic finite-state automaton that recognizes each of the languages in Exercise 55, and has fewer states, if possible, than the deterministic automaton you found in that exercise.

*57. Show that there is no finite-state automaton that recognizes the set of bit strings containing an equal number of 0s and 1s.

In Exercises 58–62 we introduce a technique for constructing a deterministic finite-state machine equivalent to a given deterministic finite-state machine with the least number of states possible. Suppose that $M = (S, I, f, s_0, F)$ is a finite-state automaton and that k is a nonnegative integer. Let R_k be the relation on the set S of states of M such that $s R_k t$ if and only if for every input string x with $l(x) \leq k$ [where $l(x)$ is the length of x , as usual], $f(s, x)$ and $f(t, x)$ are both final states

or both not final states. Furthermore, let R_* be the relation on the set of states of M such that $s R_* t$ if and only if for every input string x , regardless of length, $f(s, x)$ and $f(t, x)$ are both final states or both not final states.

- *58. a) Show that for every nonnegative integer k , R_k is an equivalence relation on S . We say that two states s and t are **k -equivalent** if $s R_k t$.
 - b) Show that R_* is an equivalence relation on S . We say that two states s and t are **$*$ -equivalent** if $s R_* t$.
 - c) Show that if s and t are two k -equivalent states of M , where k is a positive integer, then s and t are also $(k - 1)$ -equivalent.
 - d) Show that the equivalence classes of R_k are a refinement of the equivalence classes of R_{k-1} if k is a positive integer. (The refinement of a partition of a set is defined in the preamble to Exercise 49 in Section 9.5.)
 - e) Show that if s and t are k -equivalent for every nonnegative integer k , then they are $*$ -equivalent.
 - f) Show that all states in a given R_* -equivalence class are final states or all are not final states.
 - g) Show that if s and t are R_* -equivalent, then $f(s, a)$ and $f(t, a)$ are also R_* -equivalent for all $a \in I$.
- *59. Show that there is a nonnegative integer n such that the set of n -equivalence classes of states of M is the same as the set of $(n + 1)$ -equivalence classes of states of M . Then show for this integer n , the set of n -equivalence classes of states of M equals the set of $*$ -equivalence classes of states of M .

The **quotient automaton** \bar{M} of the deterministic finite-state automaton $M = (S, I, f, s_0, F)$ is the finite-state automaton $(\bar{S}, I, \bar{f}, [s_0]_{R_*}, \bar{F})$, where the set of states \bar{S} is the set of $*$ -equivalence classes of S , the transition function \bar{f} is defined by $\bar{f}([s]_{R_*}, a) = [f(s, a)]_{R_*}$ for all states $[s]_{R_*}$ of \bar{M} and input symbols $a \in I$, and \bar{F} is the set consisting of R_* -equivalence classes of final states of M .

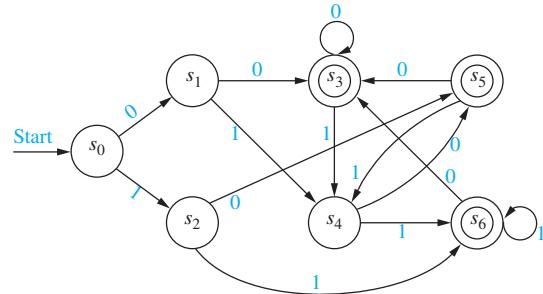
60. a) Show that s and t are 0-equivalent if and only if either both s and t are final states or neither s nor t is a final state. Conclude that each final state of \bar{M} , which is an R_ -equivalence class, contains only final states of M .

- b) Show that if k is a positive integer, then s and t are k -equivalent if and only if s and t are $(k - 1)$ -equivalent and for every input symbol $a \in I$, $f(s, a)$ and $f(t, a)$ are $(k - 1)$ -equivalent. Conclude that the transition function \bar{f} is well-defined.
- c) Describe a procedure that can be used to construct the quotient automaton of a finite-automaton M .

**61. a) Show that if M is a finite-state automaton, then the quotient automaton \bar{M} recognizes the same language as M .

- b) Show that if M is a finite-state automaton with the property that for every state s of M there is a string $x \in I^*$ such that $f(s_0, x) = s$, then the quotient automaton \bar{M} has the minimum number of states of any finite-state automaton equivalent to M .

62. Answer these questions about the finite-state automaton M shown here.



- a) Find the k -equivalence classes of M for $k = 0, 1, 2$, and 3. Also, find the $*$ -equivalence classes of M .
- b) Construct the quotient automaton \bar{M} of M .

13.4 Language Recognition

Introduction

We have seen that finite-state automata can be used as language recognizers. What sets can be recognized by these machines? Although this seems like an extremely difficult problem, there is a simple characterization of the sets that can be recognized by finite state automata. This problem was first solved in 1956 by the American mathematician Stephen Kleene. He showed that there is a finite-state automaton that recognizes a set if and only if this set can be built up from the null set, the empty string, and singleton strings by taking concatenations, unions, and Kleene closures, in arbitrary order. Sets that can be built up in this way are called **regular sets**. Regular grammars were defined in Section 13.1. Because of the terminology used, it is not surprising that there is a connection between regular sets, which are the sets recognized by finite-state automata, and regular grammars. In particular, a set is regular if and only if it is generated by a regular grammar.

Finally, there are sets that cannot be recognized by any finite-state automata. We will give an example of such a set. We will briefly discuss more powerful models of computation, such as pushdown automata and Turing machines, at the end of this section. The regular sets are those

that can be formed using the operations of concatenation, union, and Kleene closure in arbitrary order, starting with the empty set, the set consisting of the empty string, and singleton sets. We will see that the regular sets are those that can be recognized using a finite-state automaton. To define regular sets we first need to define regular expressions.

DEFINITION 1

The *regular expressions* over a set I are defined recursively by:

- the symbol \emptyset is a regular expression;
- the symbol λ is a regular expression;
- the symbol x is a regular expression whenever $x \in I$;
- the symbols (AB) , $(A \cup B)$, and A^* are regular expressions whenever A and B are regular expressions.

Each regular expression represents a set specified by these rules:

- \emptyset represents the empty set, that is, the set with no strings;
- λ represents the set $\{\lambda\}$, which is the set containing the empty string;
- x represents the set $\{x\}$ containing the string with one symbol x ;
- (AB) represents the concatenation of the sets represented by A and by B ;
- $(A \cup B)$ represents the union of the sets represented by A and by B ;
- A^* represents the Kleene closure of the set represented by A .

Sets represented by regular expressions are called **regular sets**. Henceforth regular expressions will be used to describe regular sets, so when we refer to the regular set A , we will mean the regular set represented by the regular expression A . Note that we will leave out outer parentheses from regular expressions when they are not needed.

Example 1 shows how regular expressions are used to specify regular sets.

EXAMPLE 1 What are the strings in the regular sets specified by the regular expressions 10^* , $(10)^*$, $0 \cup 01$, $0(0 \cup 1)^*$, and $(0^*1)^*$?

Solution: The regular sets represented by these expressions are given in Table 1, as the reader should verify. 

Finding a regular expression that specifies a given set can be quite tricky, as Example 2 illustrates.

EXAMPLE 2 Find a regular expression that specifies each of these sets:

- the set of bit strings with even length
- the set of bit strings ending with a 0 and not containing 11
- the set of bit strings containing an odd number of 0s

TABLE 1

Expression	Strings
10^*	a 1 followed by any number of 0s (including no zeros)
$(10)^*$	any number of copies of 10 (including the null string)
$0 \cup 01$	the string 0 or the string 01
$0(0 \cup 1)^*$	any string beginning with 0
$(0^*1)^*$	any string not ending with 0

Solution: (a) To construct a regular expression for the set of bit strings with even length, we use the fact that such a string can be obtained by concatenating zero or more strings each consisting of two bits. The set of strings of two bits is specified by the regular expression $(00 \cup 01 \cup 10 \cup 11)$. Consequently, the set of strings with even length is specified by $(00 \cup 01 \cup 10 \cup 11)^*$.

(b) A bit string ending with a 0 and not containing 11 must be the concatenation of one or more strings where each string is either a 0 or a 10. (To see this, note that such a bit string must consist of 0 bits or 1 bits each followed by a 0; the string cannot end with a single 1 because we know it ends with a 0.) It follows that the regular expression $(0 \cup 10)^*(0 \cup 10)$ specifies the set of bit strings that do not contain 11 and end with a 0. [Note that the set specified by $(0 \cup 10)^*$ includes the empty string, which is not in this set, because the empty string does not end with a 0.]

(c) A bit string containing an odd number of 0s must contain at least one 0, which tells us that it starts with zero or more 1s, followed by a 0, followed by zero or more 1s. That is, each such bit string begins with a string of the form $1^j 0 1^k$ for nonnegative integers j and k . Because the bit string contains an odd number of 0s, additional bits after this initial block can be split into blocks each starting with a 0 and containing one more 0. Each such block is of the form $01^p 0 1^q$, where p and q are nonnegative integers. Consequently, the regular expression $1^* 0 1^* (01^* 0 1^*)^*$ specifies the set of bit strings with an odd number of 0s. 

Kleene's Theorem

In 1956 Kleene proved that regular sets are the sets that are recognized by a finite-state automaton. Consequently, this important result is called Kleene's theorem.

THEOREM 1

KLEENE'S THEOREM A set is regular if and only if it is recognized by a finite-state automaton.



Kleene's theorem is one of the central results in automata theory. We will prove the *only if* part of this theorem, namely, that every regular set is recognized by a finite-state automaton. The proof of the *if* part, that a set recognized by a finite-state automaton is regular, is left as an exercise for the reader.

Proof: Recall that a regular set is defined in terms of regular expressions, which are defined recursively. We can prove that every regular set is recognized by a finite-state automaton if we can do the following things.



1. Show that \emptyset is recognized by a finite-state automaton.
2. Show that $\{\lambda\}$ is recognized by a finite-state automaton.
3. Show that $\{a\}$ is recognized by a finite-state automaton whenever a is a symbol in I .
4. Show that AB is recognized by a finite-state automaton whenever both A and B are.
5. Show that $A \cup B$ is recognized by a finite-state automaton whenever both A and B are.
6. Show that A^* is recognized by a finite-state automaton whenever A is.

We now consider each of these tasks. First, we show that \emptyset is recognized by a nondeterministic finite-state automaton. To do this, all we need is an automaton with no final states. Such an automaton is shown in Figure 1(a).

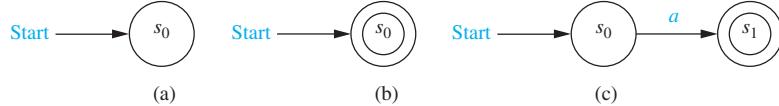


FIGURE 1 Nondeterministic Finite-State Automata That Recognize Some Basic Sets.

Second, we show that $\{\lambda\}$ is recognized by a finite-state automaton. To do this, all we need is an automaton that recognizes λ , the null string, but not any other string. This can be done by making the start state s_0 a final state and having no transitions, so that no other string takes s_0 to a final state. The nondeterministic automaton in Figure 1(b) shows such a machine.

Third, we show that $\{a\}$ is recognized by a nondeterministic finite-state automaton. To do this, we can use a machine with a starting state s_0 and a final state s_1 . We have a transition from s_0 to s_1 when the input is a , and no other transitions. The only string recognized by this machine is a . This machine is shown in Figure 1(c).

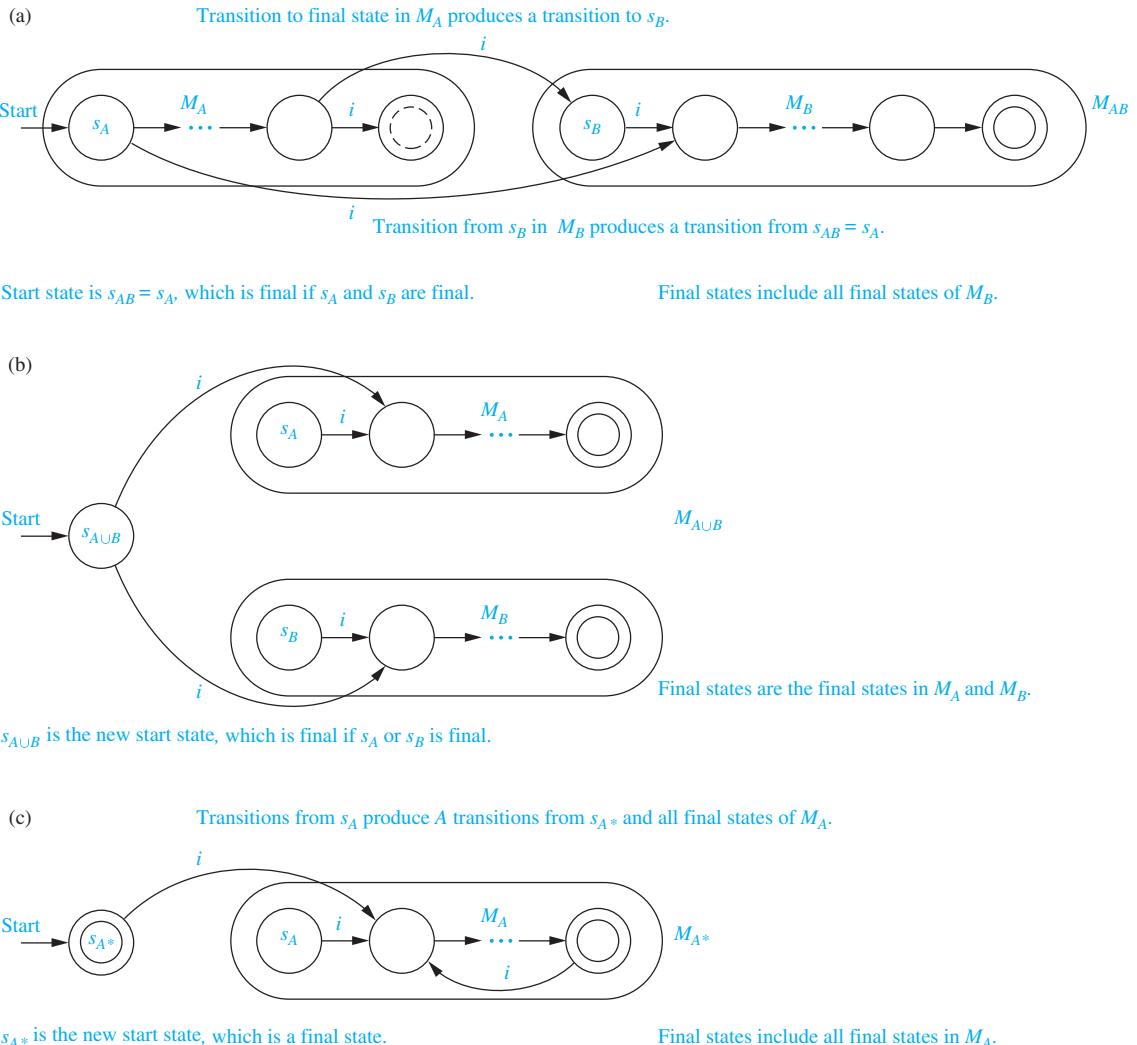
Next, we show that AB and $A \cup B$ can be recognized by finite-state automata if A and B are languages recognized by finite-state automata. Suppose that A is recognized by $M_A = (S_A, I, f_A, s_A, F_A)$ and B is recognized by $M_B = (S_B, I, f_B, s_B, F_B)$.

We begin by constructing a finite-state machine $M_{AB} = (S_{AB}, I, f_{AB}, s_{AB}, F_{AB})$ that recognizes AB , the concatenation of A and B . We build such a machine by combining the machines for A and B in series, so a string in A takes the combined machine from s_A , the start state of M_A , to s_B , the start state of M_B . A string in B should take the combined machine from s_B to a final state of the combined machine. Consequently, we make the following construction. Let S_{AB} be $S_A \cup S_B$. [Note that we can assume that S_A and S_B are disjoint.] The starting state s_{AB} is the same as s_A . The set of final states, F_{AB} , is the set of final states of M_B with s_{AB} included if and only if $\lambda \in A \cap B$. The transitions in M_{AB} include all transitions in M_A and in M_B , as well as some new transitions. For every transition in M_A that leads to a final state, we form a transition in M_{AB} from the same state to s_B , on the same input. In this way, a string in A takes M_{AB} from s_{AB} to s_B , and then a string in B takes s_B to a final state of M_{AB} . Moreover, for every transition from s_B we form a transition in M_{AB} from s_{AB} to the same state. Figure 2(a) contains an illustration of this construction.

We now construct a machine $M_{A \cup B} = (S_{A \cup B}, I, f_{A \cup B}, s_{A \cup B}, F_{A \cup B})$ that recognizes $A \cup B$. This automaton can be constructed by combining M_A and M_B in parallel, using a new start state that has the transitions that both s_A and s_B have. Let $S_{A \cup B} = S_A \cup S_B \cup \{s_{A \cup B}\}$, where $s_{A \cup B}$ is a new state that is the start state of $M_{A \cup B}$. Let the set of final states $F_{A \cup B}$ be $F_A \cup F_B \cup \{s_{A \cup B}\}$ if $\lambda \in A \cup B$, and $F_A \cup F_B$ otherwise. The transitions in $M_{A \cup B}$ include all those in M_A and in M_B . Also, for each transition from s_A to a state s on input i we include a transition from $s_{A \cup B}$ to s on input i , and for each transition from s_B to a state s on input i we include a transition from $s_{A \cup B}$ to s on input i . In this way, a string in A leads from $s_{A \cup B}$ to a final state in the new machine, and a string in B leads from $s_{A \cup B}$ to a final state in the new machine. Figure 2(b) illustrates the construction of $M_{A \cup B}$.

Finally, we construct $M_{A^*} = (S_{A^*}, I, f_{A^*}, s_{A^*}, F_{A^*})$, a machine that recognizes A^* , the Kleene closure of A . Let S_{A^*} include all states in S_A and one additional state s_{A^*} , which is the starting state for the new machine. The set of final states F_{A^*} includes all states in F_A as well as the start state s_{A^*} , because λ must be recognized. To recognize concatenations of arbitrarily many strings from A , we include all the transitions in M_A , as well as transitions from s_{A^*} that match the transitions from s_A , and transitions from each final state that match the transitions from s_A . With this set of transitions, a string made up of concatenations of strings from A will take s_{A^*} to a final state when the first string in A has been read, returning to a final state when the second string in A has been read, and so on. Figure 2(c) illustrates the construction we used. \triangle

A nondeterministic finite-state automaton can be constructed for any regular set using the procedure described in this proof. We illustrate how this is done with Example 3.

**FIGURE 2** Building Automata to Recognize Concatenations, Unions, and Kleene Closures.

EXAMPLE 3 Construct a nondeterministic finite-state automaton that recognizes the regular set $1^* \cup 01$.

Solution: We begin by building a machine that recognizes 1^* . This is done using the machine that recognizes 1 and then using the construction for M_{A^*} described in the proof. Next, we build a machine that recognizes 01 , using machines that recognize 0 and 1 and the construction in the proof for M_{AB} . Finally, using the construction in the proof for $M_{A \cup B}$, we construct the machine for $1^* \cup 01$. The finite-state automata used in this construction are shown in Figure 3. The states in the successive machines have been labeled using different subscripts, even when a state is formed from one previously used in another machine. Note that the construction given here does not produce the simplest machine that recognizes $1^* \cup 01$. A much simpler machine that recognizes this set is shown in Figure 3(b). 

Regular Sets and Regular Grammars

In Section 13.1 we introduced phrase-structure grammars and defined different types of grammars. In particular we defined regular, or type 3, grammars, which are grammars of the

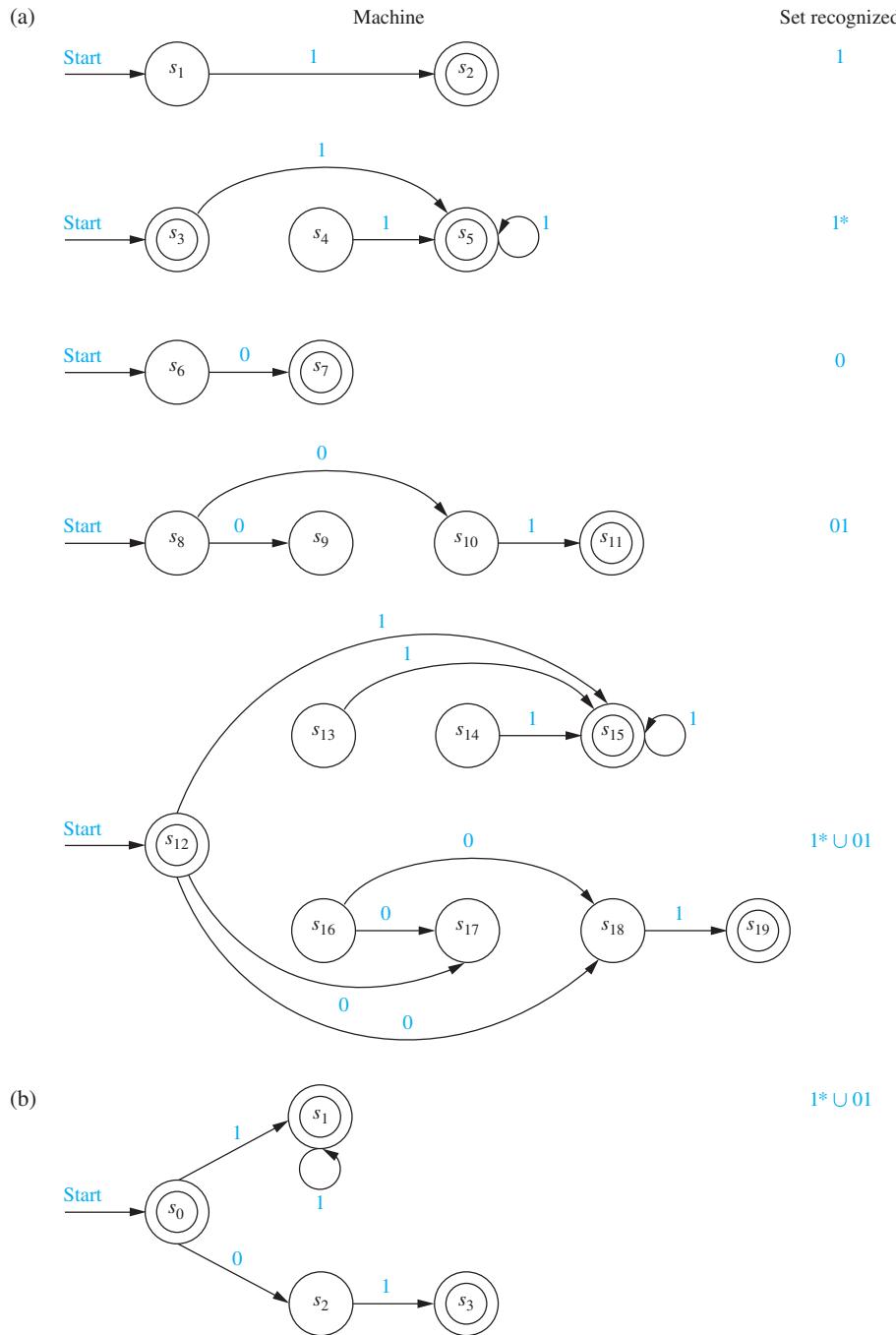


FIGURE 3 Nondeterministic Finite-State Automata Recognizing $1^* \cup 01$.

form $G = (V, T, S, P)$, where each production is of the form $S \rightarrow \lambda$, $A \rightarrow a$, or $A \rightarrow aB$, where a is a terminal symbol, and A and B are nonterminal symbols. As the terminology suggests, there is a close connection between regular grammars and regular sets.

THEOREM 2

A set is generated by a regular grammar if and only if it is a regular set.

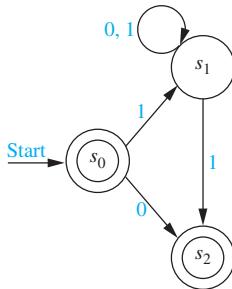


FIGURE 4 A Nondeterministic Finite-State Automaton Recognizing $L(G)$.

Proof: First we show that a set generated by a regular grammar is a regular set. Suppose that $G = (V, T, S, P)$ is a regular grammar generating the set $L(G)$. To show that $L(G)$ is regular we will build a nondeterministic finite-state machine $M = (S, I, f, s_0, F)$ that recognizes $L(G)$. Let S , the set of states, contain a state s_A for each nonterminal symbol A of G and an additional state s_F , which is a final state. The start state s_0 is the state formed from the start symbol S . The transitions of M are formed from the productions of G in the following way. A transition from s_A to s_F on input of a is included if $A \rightarrow a$ is a production, and a transition from s_A to s_B on input of a is included if $A \rightarrow aB$ is a production. The set of final states includes s_F and also includes s_0 if $S \rightarrow \lambda$ is a production in G . It is not hard to show that the language recognized by M equals the language generated by the grammar G , that is, $L(M) = L(G)$. This can be done by determining the words that lead to a final state. The details are left as an exercise for the reader. \triangleleft

Before giving the proof of the converse, we illustrate how a nondeterministic machine is constructed that recognizes the same set as a regular grammar.

EXAMPLE 4 Construct a nondeterministic finite-state automaton that recognizes the language generated by the regular grammar $G = (V, T, S, P)$, where $V = \{0, 1, A, S\}$, $T = \{0, 1\}$, and the productions in P are $S \rightarrow 1A$, $S \rightarrow 0$, $S \rightarrow \lambda$, $A \rightarrow 0A$, $A \rightarrow 1A$, and $A \rightarrow 1$.

Solution: The state diagram for a nondeterministic finite-state automaton that recognizes $L(G)$ is shown in Figure 4. This automaton is constructed following the procedure described in the proof. In this automaton, s_0 is the state corresponding to S , s_1 is the state corresponding to A , and s_2 is the final state. \triangleleft

We now complete the proof of Theorem 2.

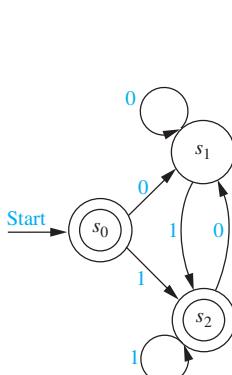


FIGURE 5 A Finite-State Automaton.

Proof: We now show that if a set is regular, then there is a regular grammar that generates it. Suppose that M is a finite-state machine that recognizes this set with the property that s_0 , the starting state of M , is never the next state for a transition. (We can find such a machine by Exercise 20.) The grammar $G = (V, T, S, P)$ is defined as follows. The set V of symbols of G is formed by assigning a symbol to each state of S and each input symbol in I . The set T of terminal symbols of G is the set I . The start symbol S is the symbol formed from the start state s_0 . The set P of productions in G is formed from the transitions in M . In particular, if the state s goes to a final state under input a , then the production $A_s \rightarrow a$ is included in P , where A_s is the nonterminal symbol formed from the state s . If the state s goes to the state t on input a , then the production $A_s \rightarrow aA_t$ is included in P . The production $S \rightarrow \lambda$ is included in P if and only if $\lambda \in L(M)$. Because the productions of G correspond to the transitions of M and the productions leading to terminals correspond to transitions to final states, it is not hard to show that $L(G) = L(M)$. We leave the details as an exercise for the reader. \triangleleft

Example 5 illustrates the construction used to produce a grammar from an automaton that generates the language recognized by this automaton.

EXAMPLE 5 Find a regular grammar that generates the regular set recognized by the finite-state automaton shown in Figure 5.

Solution: The grammar $G = (V, T, S, P)$ generates the set recognized by this automaton where $V = \{S, A, B, 0, 1\}$, the symbols S , A , and B correspond to the states s_0 , s_1 , and s_2 , respectively, $T = \{0, 1\}$, S is the start symbol; and the productions are $S \rightarrow 0A$, $S \rightarrow 1B$, $S \rightarrow 1$, $S \rightarrow \lambda$, $A \rightarrow 0A$, $A \rightarrow 1B$, $A \rightarrow 1$, $B \rightarrow 0A$, $B \rightarrow 1B$, and $B \rightarrow 1$. \triangleleft

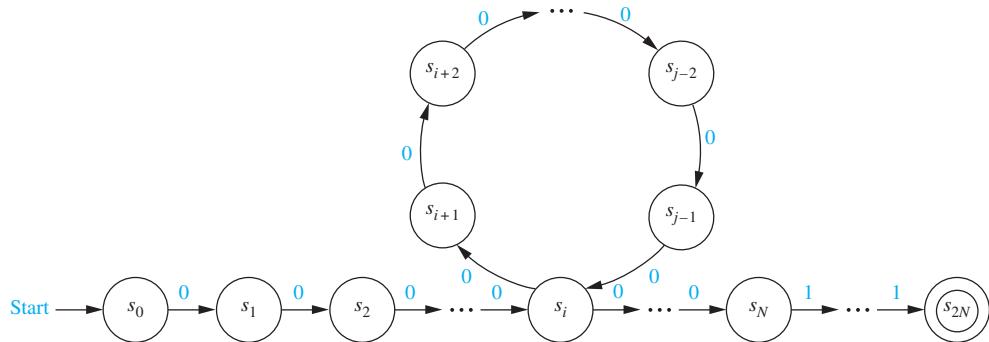


FIGURE 6 The Path Produced by $0^N 1^N$.

A Set Not Recognized by a Finite-State Automaton

We have seen that a set is recognized by a finite state automaton if and only if it is regular. We will now show that there are sets that are not regular by describing one such set. The technique used to show that this set is not regular illustrates an important method for showing that certain sets are not regular.

EXAMPLE 6 Show that the set $\{0^n 1^n \mid n = 0, 1, 2, \dots\}$, made up of all strings consisting of a block of 0s followed by a block of an equal number of 1s, is not regular.

 **Solution:** Suppose that this set were regular. Then there would be a nondeterministic finite-state automaton $M = (S, I, f, s_0, F)$ recognizing it. Let N be the number of states in this machine, that is, $N = |S|$. Because M recognizes all strings made up of a number of 0s followed by an equal number of 1s, M must recognize $0^N 1^N$. Let $s_0, s_1, s_2, \dots, s_{2N}$ be the sequence of states that is obtained starting at s_0 and using the symbols of $0^N 1^N$ as input, so that $s_1 = f(s_0, 0)$, $s_2 = f(s_1, 0), \dots, s_N = f(s_{N-1}, 0), s_{N+1} = f(s_N, 1), \dots, s_{2N} = f(s_{2N-1}, 1)$. Note that s_{2N} is a final state.

Because there are only N states, the pigeonhole principle shows that at least two of the first $N + 1$ of the states, which are s_0, \dots, s_N , must be the same. Say that s_i and s_j are two such identical states, with $0 \leq i < j \leq N$. This means that $f(s_i, 0^t) = s_j$, where $t = j - i$. It follows that there is a loop leading from s_i back to itself, obtained using the input 0 a total of t times, in the state diagram shown in Figure 6.

Now consider the input string $0^N 0^t 1^N = 0^{N+t} 1^N$. There are t more consecutive 0s at the start of this block than there are consecutive 1s that follow it. Because this string is not of the form $0^n 1^n$ (because it has more 0s than 1s), it is not recognized by M . Consequently, $f(s_0, 0^{N+t} 1^N)$ cannot be a final state. However, when we use the string $0^{N+t} 1^N$ as input, we end up in the same state as before, namely, s_{2N} . The reason for this is that the extra t 0s in this string take us around the loop from s_i back to itself an extra time, as shown in Figure 6. Then the rest of the string leads us to exactly the same state as before. This contradiction shows that $\{0^n 1^n \mid n = 0, 1, 2, \dots\}$ is not regular. 

More Powerful Types of Machines



Finite-state automata are unable to carry out many computations. The main limitation of these machines is their finite amount of memory. This prevents them from recognizing languages that are not regular, such as $\{0^n 1^n \mid n = 0, 1, 2, \dots\}$. Because a set is regular if and only if it is the language generated by a regular grammar, Example 6 shows that there is no regular grammar

that generates the set $\{0^n 1^n \mid n = 0, 1, 2, \dots\}$. However, there is a context-free grammar that generates this set. Such a grammar was given in Example 5 in Section 13.1.

Because of the limitations of finite-state machines, it is necessary to use other, more powerful, models of computation. One such model is the **pushdown automaton**. A pushdown automaton includes everything in a finite-state automaton, as well as a stack, which provides unlimited memory. Symbols can be placed on the top or taken off the top of the stack. A set is recognized in one of two ways by a pushdown automaton. First, a set is recognized if the set consists of all the strings that produce an empty stack when they are used as input. Second, a set is recognized if it consists of all the strings that lead to a final state when used as input. It can be shown that a set is recognized by a pushdown automaton if and only if it is the language generated by a context-free grammar.

However, there are sets that cannot be expressed as the language generated by a context-free grammar. One such set is $\{0^n 1^n 2^n \mid n = 0, 1, 2, \dots\}$. We will indicate why this set cannot be recognized by a pushdown automaton, but we will not give a proof, because we have not developed the machinery needed. (However, one method of proof is given in Exercise 28 of the supplementary exercises at the end of this chapter.) The stack can be used to show that a string begins with a sequence of 0s followed by an equal number of 1s by placing a symbol on the stack for each 0 (as long as only 0s are read), and removing one of these symbols for each 1 (as long as only 1s following the 0s are read). But once this is done, the stack is empty, and there is no way to determine that there are the same number of 2s in the string as 0s.

There are other machines called **linear bounded automata**, more powerful than pushdown automata, that can recognize sets such as $\{0^n 1^n 2^n \mid n = 0, 1, 2, \dots\}$. In particular, linear bounded automata can recognize context-sensitive languages. However, these machines cannot recognize all the languages generated by phrase-structure grammars. To avoid the limitations of special types of machines, the model known as a **Turing machine**, named after the British mathematician Alan Turing, is used. A Turing machine is made up of everything included in a finite-state machine together with a tape, which is infinite in both directions. A Turing machine has read and write capabilities on the tape, and it can move back and forth along this tape. Turing machines can recognize all languages generated by phrase-structure grammars. In addition, Turing machines can model all the computations that can be performed on a computing machine. Because of their power, Turing machines are extensively studied in theoretical computer science. We will briefly study them in Section 13.5.

Alan Turing invented
Turning machines before
modern computers
existed!

Links

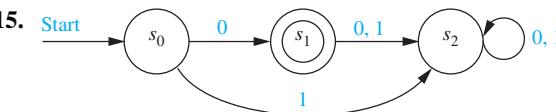
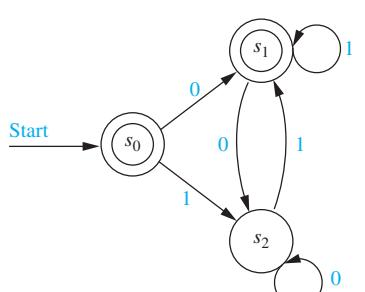
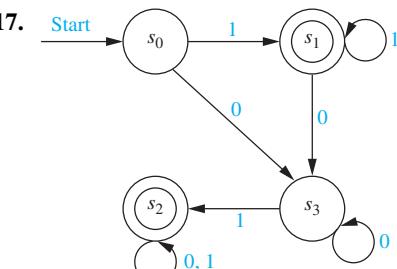


ALAN MATHISON TURING (1912–1954) Alan Turing was born in London, although he was conceived in India, where his father was employed in the Indian Civil Service. As a boy, he was fascinated by chemistry, performing a wide variety of experiments, and by machinery. Turing attended Sherborne, an English boarding school. In 1931 he won a scholarship to King's College, Cambridge. After completing his dissertation, which included a rediscovery of the central limit theorem, a famous theorem in statistics, he was elected a fellow of his college. In 1935 Turing became fascinated with the decision problem, a problem posed by the great German mathematician Hilbert, which asked whether there is a general method that can be applied to any assertion to determine whether the assertion is true. Turing enjoyed running (later in life running as a serious amateur in competitions), and one day, while resting after a run, he discovered the key ideas needed to solve the decision problem. In his solution, he invented what is now called a **Turing machine** as the most general model of a computing machine. Using these machines, he found a problem, involving what he called computable numbers, that could not be decided using a general method.

From 1936 to 1938 Turing visited Princeton University to work with Alonzo Church, who had also solved Hilbert's decision problem. In 1939 Turing returned to King's College. However, at the outbreak of World War II, he joined the Foreign Office, performing cryptanalysis of German ciphers. His contribution to the breaking of the code of the Enigma, a mechanical German cipher machine, played an important role in winning the war.

After the war, Turing worked on the development of early computers. He was interested in the ability of machines to think, proposing that if a computer could not be distinguished from a person based on written replies to questions, it should be considered to be "thinking." He was also interested in biology, having written on morphogenesis, the development of form in organisms. In 1954 Turing committed suicide by taking cyanide, without leaving a clear explanation. Legal troubles related to a homosexual relationship and hormonal treatments mandated by the court to lessen his sex drive may have been factors in his decision to end his life.

Exercises

1. Describe in words the strings in each of these regular sets.
 - a) 1^*0
 - b) 1^*00^*
 - c) $111 \cup 001$
 - d) $(1 \cup 00)^*$
 - e) $(00^*1)^*$
 - f) $(0 \cup 1)(0 \cup 1)^*00$
 2. Describe in words the strings in each of these regular sets.
 - a) 001^*
 - b) $(01)^*$
 - c) $01 \cup 001^*$
 - d) $0(11 \cup 0)^*$
 - e) $(101^*)^*$
 - f) $(0^* \cup 1)11$
 3. Determine whether 0101 belongs to each of these regular sets.
 - a) 01^*0^*
 - b) $0(11)^*(01)^*$
 - c) $0(10)^*1^*$
 - d) $0^*10(0 \cup 1)$
 - e) $(01)^*(11)^*$
 - f) $0^*(10 \cup 11)^*$
 - g) $0^*(10)^*11$
 - h) $01(01 \cup 0)1^*$
 4. Determine whether 1011 belongs to each of these regular sets.
 - a) 10^*1^*
 - b) $0^*(10 \cup 11)^*$
 - c) $1(01)^*1^*$
 - d) $1^*01(0 \cup 1)$
 - e) $(10)^*(11)^*$
 - f) $1(00)^*(11)^*$
 - g) $(10)^*1011$
 - h) $(1 \cup 00)(01 \cup 0)1^*$
 5. Express each of these sets using a regular expression.
 - a) the set consisting of the strings 0, 11, and 010
 - b) the set of strings of three 0s followed by two or more 0s
 - c) the set of strings of odd length
 - d) the set of strings that contain exactly one 1
 - e) the set of strings ending in 1 and not containing 000
 6. Express each of these sets using a regular expression.
 - a) the set containing all strings with zero, one, or two bits
 - b) the set of strings of two 0s, followed by zero or more 1s, and ending with a 0
 - c) the set of strings with every 1 followed by two 0s
 - d) the set of strings ending in 00 and not containing 11
 - e) the set of strings containing an even number of 1s
 7. Express each of these sets using a regular expression.
 - a) the set of strings of one or more 0s followed by a 1
 - b) the set of strings of two or more symbols followed by three or more 0s
 - c) the set of strings with either no 1 preceding a 0 or no 0 preceding a 1
 - d) the set of strings containing a string of 1s such that the number of 1s equals 2 modulo 3, followed by an even number of 0s
 8. Construct deterministic finite-state automata that recognize each of these sets from I^* , where I is an alphabet.
 - a) \emptyset
 - b) $\{\lambda\}$
 - c) $\{a\}$, where $a \in I$
 9. Construct nondeterministic finite-state automata that recognize each of the sets in Exercise 8.
 10. Construct nondeterministic finite-state automata that recognize each of these sets.
 - a) $\{\lambda, 0\}$
 - b) $\{0, 11\}$
 - c) $\{0, 11, 000\}$
 - *11. Show that if A is a regular set, then A^R , the set of all reversals of strings in A , is also regular.
 12. Using the constructions described in the proof of Kleene's theorem, find nondeterministic finite-state automata that recognize each of these sets.
 - a) 01^*
 - b) $(0 \cup 1)1^*$
 - c) $00(1^* \cup 10)$
 13. Using the constructions described in the proof of Kleene's theorem, find nondeterministic finite-state automata that recognize each of these sets.
 - a) 0^*1^*
 - b) $(0 \cup 11)^*$
 - c) $01^* \cup 00^*$
 14. Construct a nondeterministic finite-state automaton that recognizes the language generated by the regular grammar $G = (V, T, S, P)$, where $V = \{0, 1, S, A, B\}$, $T = \{0, 1\}$, S is the start symbol, and the set of productions is
 - a) $S \rightarrow 0A$, $S \rightarrow 1B$, $A \rightarrow 0$, $B \rightarrow 0$.
 - b) $S \rightarrow 1A$, $S \rightarrow 0$, $S \rightarrow \lambda$, $A \rightarrow 0B$, $B \rightarrow 1B$, $B \rightarrow 1$.
 - c) $S \rightarrow 1B$, $S \rightarrow 0$, $A \rightarrow 1A$, $A \rightarrow 0B$, $A \rightarrow 1$, $A \rightarrow 0$, $B \rightarrow 1$.
- In Exercises 15–17 construct a regular grammar $G = (V, T, S, P)$ that generates the language recognized by the given finite-state machine.
15. 
 16. 
 17. 
 18. Show that the finite-state automaton constructed from a regular grammar in the proof of Theorem 2 recognizes the set generated by this grammar.
 19. Show that the regular grammar constructed from a finite-state automaton in the proof of Theorem 2 generates the set recognized by this automaton.

- 20.** Show that every nondeterministic finite-state automaton is equivalent to another such automaton that has the property that its starting state is never revisited.
- *21.** Let $M = (S, I, f, s_0, F)$ be a deterministic finite-state automaton. Show that the language recognized by M , $L(M)$, is infinite if and only if there is a word x recognized by M with $l(x) \geq |S|$.
- *22.** One important technique used to prove that certain sets are not regular is the **pumping lemma**. The pumping lemma states that if $M = (S, I, f, s_0, F)$ is a deterministic finite-state automaton and if x is a string in $L(M)$, the language recognized by M , with $l(x) \geq |S|$, then there are strings u , v , and w in I^* such that $x = uvw$, $l(uv) \leq |S|$ and $l(v) \geq 1$, and $uv^iw \in L(M)$ for $i = 0, 1, 2, \dots$. Prove the pumping lemma. [Hint: Use the same idea as was used in Example 5.]
- *23.** Show that the set $\{0^{2n}1^n \mid n = 0, 1, 2, \dots\}$ is not regular using the pumping lemma given in Exercise 22.
- *24.** Show that the set $\{1^{n^2} \mid n = 0, 1, 2, \dots\}$ is not regular using the pumping lemma from Exercise 22.
- *25.** Show that the set of palindromes over $\{0, 1\}$ is not regular using the pumping lemma given in Exercise 22. [Hint: Consider strings of the form $0^N 1 0^N$.]
- **26.** Show that a set recognized by a finite-state automaton is regular. (This is the *if* part of Kleene's theorem.)
- Suppose that L is a subset of I^* , where I is a nonempty set of symbols. If $x \in I^*$, we let $L/x = \{z \in I^* \mid xz \in L\}$. We say that the strings $x \in I^*$ and $y \in I^*$ are **distinguishable with respect to L** if $L/x \neq L/y$. A string z for which $xz \in L$ but $yz \notin L$, or $xz \notin L$, but $yz \in L$ is said to **distinguish x and y with respect to L** . When $L/x = L/y$, we say that x and y are **indistinguishable** with respect to L .
- 27.** Let L be the set of all bit strings that end with 01. Show that 11 and 10 are distinguishable with respect to L and that the strings 1 and 11 are indistinguishable with respect to L .
- 28.** Suppose that $M = (S, I, f, s_0, F)$ is a deterministic finite-state machine. Show that if x and y are two strings in I^* that are distinguishable with respect to $L(M)$, then $f(s_0, x) \neq f(s_0, y)$.
- *29.** Suppose that L is a subset of I^* and for some positive integer n there are n strings in I^* such that every two of these strings are distinguishable with respect to L . Prove that every deterministic finite-state automaton recognizing L has at least n states.
- *30.** Let L_n be the set of strings with at least n bits in which the n th symbol from the end is a 0. Use Exercise 29 to show that a deterministic finite-state machine recognizing L_n must have at least 2^n states.
- *31.** Use Exercise 29 to show that the language consisting of all bit strings that are palindromes (that is, strings that equal their own reversals) is not regular.

13.5 Turing Machines

Introduction



The finite-state automata studied earlier in this chapter cannot be used as general models of computation. They are limited in what they can do. For example, finite-state automata are able to recognize regular sets, but are not able to recognize many easy-to-describe sets, including $\{0^n 1^n \mid n \geq 0\}$, which computers recognize using memory. We can use finite-state automata to compute relatively simple functions such as the sum of two numbers, but we cannot use them to compute functions that computers can, such as the product of two numbers. To overcome these deficiencies we can use a more powerful type of machine known as a Turing machine, after Alan Turing, the famous mathematician and computer scientist who invented them in the 1930s.

Basically, a Turing machine consists of a control unit, which at any step is in one of finitely many different states, together with a tape divided into cells, which is infinite in both directions. Turing machines have read and write capabilities on the tape as the control unit moves back and forth along this tape, changing states depending on the tape symbol read. Turing machines are more powerful than finite-state machines because they include memory capabilities that finite-state machines lack. We will show how to use Turing machines to recognize sets, including sets that cannot be recognized by finite-state machines. We will also show how to compute functions using Turing machines. Turing machines are the most general models of computation; essentially, they can do whatever a computer can do. Note that Turing machines are much more powerful than real computers, which have finite memory capabilities.

“Machines take me by surprise with great frequency” – Alan Turing

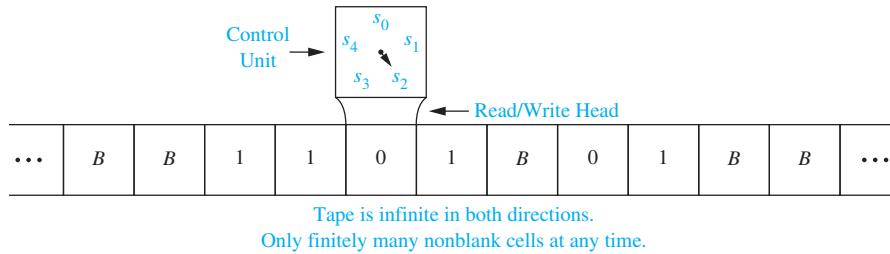


FIGURE 1 A Representation of a Turing Machine.

Definition of Turing Machines

We now give the formal definition of a Turing machine. Afterward we will explain how this formal definition can be interpreted in terms of a control head that can read and write symbols on a tape and move either right or left along the tape.

DEFINITION 1

A *Turing machine* $T = (S, I, f, s_0)$ consists of a finite set S of states, an alphabet I containing the blank symbol B , a partial function f from $S \times I$ to $S \times I \times \{R, L\}$, and a starting state s_0 .

Recall from Section 2.3 that a partial function is defined only for those elements in its domain of definition. This means that for some (state, symbol) pairs the partial function f may be undefined, but for a pair for which it is defined, there is a unique (state, symbol, direction) triple associated to this pair. We call the five-tuples corresponding to the partial function in the definition of a Turing machine the **transition rules** of the machine.

To interpret this definition in terms of a machine, consider a control unit and a tape divided into cells, infinite in both directions, having only a finite number of nonblank symbols on it at any given time, as pictured in Figure 1. The action of the Turing machine at each step of its operation depends on the value of the partial function f for the current state and tape symbol.

At each step, the control unit reads the current tape symbol x . If the control unit is in state s and if the partial function f is defined for the pair (s, x) with $f(s, x) = (s', x', d)$, the control unit

1. enters the state s' ,
2. writes the symbol x' in the current cell, erasing x , and
3. moves right one cell if $d = R$ or moves left one cell if $d = L$.

We write this step as the five-tuple (s, x, s', x', d) . If the partial function f is undefined for the pair (s, x) , then the Turing machine T will *halt*.

A common way to define a Turing machine is to specify a set of five-tuples of the form (s, x, s', x', d) . The set of states and input alphabet is implicitly defined when such a definition is used.

At the beginning of its operation a Turing machine is assumed to be in the initial state s_0 and to be positioned over the leftmost nonblank symbol on the tape. If the tape is all blank, the control head can be positioned over any cell. We will call the positioning of the control head over the leftmost nonblank tape symbol the *initial position* of the machine.

Example 1 illustrates how a Turing machine works.

EXAMPLE 1



What is the final tape when the Turing machine T defined by the seven five-tuples $(s_0, 0, s_0, 0, R)$, $(s_0, 1, s_1, 1, R)$, (s_0, B, s_3, B, R) , $(s_1, 0, s_0, 0, R)$, $(s_1, 1, s_2, 0, L)$, (s_1, B, s_3, B, R) , and $(s_2, 1, s_3, 0, R)$ is run on the tape shown in Figure 2(a)?

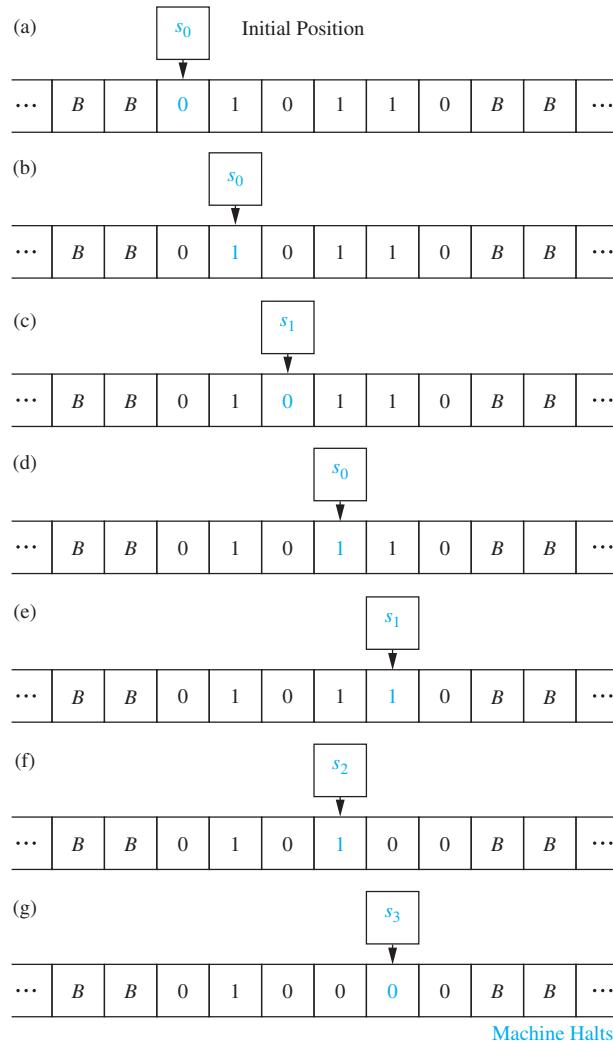


FIGURE 2 The Steps Produced by Running T on the Tape in Figure 1.

Solution: We start the operation with T in state s_0 and with T positioned over the leftmost nonblank symbol on the tape. The first step, using the five-tuple $(s_0, 0, s_0, 0, R)$, reads the 0 in the leftmost nonblank cell, stays in state s_0 , writes a 0 in this cell, and moves one cell right. The second step, using the five-tuple $(s_0, 1, s_1, 1, R)$, reads the 1 in the current cell, enters state s_1 , writes a 1 in this cell, and moves one cell right. The third step, using the five-tuple $(s_1, 0, s_0, 0, R)$, reads the 0 in the current cell, enters state s_0 , writes a 0 in this cell, and moves one cell right. The fourth step, using the five-tuple $(s_0, 1, s_1, 1, R)$, reads the 1 in the current cell, enters state s_1 , writes a 1 in this cell, and moves right one cell. The fifth step, using the five-tuple $(s_1, 1, s_2, 0, L)$, reads the 1 in the current cell, enters state s_2 , writes a 0 in this cell, and moves left one cell. The sixth step, using the five-tuple $(s_2, 1, s_3, 0, R)$, reads the 1 in the current cell, enters the state s_3 , writes a 0 in this cell, and moves right one cell. Finally, in the seventh step, the machine halts because there is no five-tuple beginning with the pair $(s_3, 0)$ in the description of the machine. The steps are shown in Figure 2.

Note that T changes the first pair of consecutive 1s on the tape to 0s and then halts. ◀

Using Turing Machines to Recognize Sets

Turing machines can be used to recognize sets. To do so requires that we define the concept of a final state as follows. A *final state* of a Turing machine T is a state that is not the first state in any five-tuple in the description of T using five-tuples (for example, state s_3 in Example 1).

We can now define what it means for a Turing machine to recognize a string. Given a string, we write consecutive symbols in this string in consecutive cells.

DEFINITION 2

Let V be a subset of an alphabet I . A Turing machine $T = (S, I, f, s_0)$ *recognizes* a string x in V^* if and only if T , starting in the initial position when x is written on the tape, halts in a final state. T is said to recognize a subset A of V^* if x is recognized by T if and only if x belongs to A .

Note that to recognize a subset A of V^* we can use symbols not in V . This means that the input alphabet I may include symbols not in V . These extra symbols are often used as markers (see Example 3).

When does a Turing machine T *not* recognize a string x in V^* ? The answer is that x is not recognized if T does not halt or halts in a state that is not final when it operates on a tape containing the symbols of x in consecutive cells, starting in the initial position. (The reader should understand that this is one of many possible ways to define how to recognize sets using Turing machines.)

We illustrate this concept with Example 2.

EXAMPLE 2 Find a Turing machine that recognizes the set of bit strings that have a 1 as their second bit, that is, the regular set $(0 \cup 1)1(0 \cup 1)^*$.

Solution: We want a Turing machine that, starting at the leftmost nonblank tape cell, moves right, and determines whether the second symbol is a 1. If the second symbol is 1, the machine should move into a final state. If the second symbol is not a 1, the machine should not halt or it should halt in a nonfinal state.

To construct such a machine, we include the five-tuples $(s_0, 0, s_1, 0, R)$ and $(s_0, 1, s_1, 1, R)$ to read in the first symbol and put the Turing machine in state s_1 . Next, we include the five-tuples $(s_1, 0, s_2, 0, R)$ and $(s_1, 1, s_3, 1, R)$ to read in the second symbol and either move to state s_2 if this symbol is a 0, or to state s_3 if this symbol is a 1. We do not want to recognize strings that have a 0 as their second bit, so s_2 should not be a final state. We want s_3 to be a final state. So, we can include the five-tuple $(s_2, 0, s_2, 0, R)$. Because we do not want to recognize the empty string or a string with one bit, we also include the five-tuples $(s_0, B, s_2, 0, R)$ and $(s_1, B, s_2, 0, R)$.

The Turing machine T consisting of the seven five-tuples listed here will terminate in the final state s_3 if and only if the bit string has at least two bits and the second bit of the input string is a 1. If the bit string contains fewer than two bits or if the second bit is not a 1, the machine will terminate in the nonfinal state s_2 . 

Given a regular set, a Turing machine that always moves to the right can be built to recognize this set (as in Example 2). To build the Turing machine, first find a finite-state automaton that recognizes the set and then construct a Turing machine using the transition function of the finite-state machine, always moving to the right.

We will now show how to build a Turing machine that recognizes a nonregular set.

EXAMPLE 3 Find a Turing machine that recognizes the set $\{0^n 1^n \mid n \geq 1\}$.

 **Solution:** To build such a machine, we will use an auxiliary tape symbol M as a marker. We have $V = \{0, 1\}$ and $I = \{0, 1, M\}$. We wish to recognize only a subset of strings in V^* . We will have one final state, s_6 . The Turing machine successively replaces a 0 at the leftmost position of

the string with an M and a 1 at the rightmost position of the string with an M , sweeping back and forth, terminating in a final state if and only if the string consists of a block of 0s followed by a block of the same number of 1s.

Although this is easy to describe and is easily carried out by a Turing machine, the machine we need to use is somewhat complicated. We use the marker M to keep track of the leftmost and rightmost symbols we have already examined. The five-tuples we use are $(s_0, 0, s_1, M, R)$, $(s_1, 0, s_1, 0, R)$, $(s_1, 1, s_1, 1, R)$, (s_1, M, s_2, M, L) , (s_1, B, s_2, B, L) , $(s_2, 1, s_3, M, L)$, $(s_3, 1, s_3, 1, L)$, $(s_3, 0, s_4, 0, L)$, (s_3, M, s_5, M, R) , $(s_4, 0, s_4, 0, L)$, (s_4, M, s_0, M, R) , and (s_5, M, s_6, M, R) . For example, the string 000111 would successively become $M00111$, $M0011M$, $MM011M$, $MM01MM$, $MMM1MM$, $MMMMMM$ as the machine operates until it halts. Only the changes are shown, as most steps leave the string unaltered.

We leave it to the reader (Exercise 13) to explain the actions of this Turing machine and to explain why it recognizes the set $\{0^n 1^n \mid n \geq 1\}$. 

It can be shown that a set can be recognized by a Turing machine if and only if it can be generated by a type 0 grammar, or in other words, if the set is generated by a phrase-structure grammar. The proof will not be presented here.

Computing Functions with Turing Machines

A Turing machine can be thought of as a computer that finds the values of a partial function. To see this, suppose that the Turing machine T , when given the string x as input, halts with the string y on its tape. We can then define $T(x) = y$. The domain of T is the set of strings for which T halts; $T(x)$ is undefined if T does not halt when given x as input. Thinking of a Turing machine as a machine that computes the values of a function on strings is useful, but how can we use Turing machines to compute functions defined on integers, on pairs of integers, on triples of integers, and so on?

To consider a Turing machine as a computer of functions from the set of k -tuples of non-negative integers to the set of nonnegative integers (such functions are called **number-theoretic functions**), we need a way to represent k -tuples of integers on a tape. To do so, we use **unary representations** of integers. We represent the nonnegative integer n by a string of $n + 1$ 1s so that, for instance, 0 is represented by the string 1 and 5 is represented by the string 111111. To represent the k -tuple (n_1, n_2, \dots, n_k) , we use a string of $n_1 + 1$ 1s, followed by an asterisk, followed by a string of $n_2 + 1$ 1s, followed by an asterisk, and so on, ending with a string of $n_k + 1$ 1s. For example, to represent the four-tuple $(2, 0, 1, 3)$ we use the string 111 * 1 * 11 * 1111.

We can now consider a Turing machine T as computing a sequence of number-theoretic functions $T, T^2, \dots, T^k, \dots$. The function T^k is defined by the action of T on k -tuples of integers represented by unary representations of integers separated by asterisks.

EXAMPLE 4 Construct a Turing machine for adding two nonnegative integers.

Solution: We need to build a Turing machine T that computes the function $f(n_1, n_2) = n_1 + n_2$. The pair (n_1, n_2) is represented by a string of $n_1 + 1$ 1s followed by an asterisk followed by $n_2 + 1$ 1s. The machine T should take this as input and produce as output a tape with $n_1 + n_2 + 1$ 1s. One way to do this is as follows. The machine starts at the leftmost 1 of the input string, and carries out steps to erase this 1, halting if $n_1 = 0$ so that there are no more 1s before the asterisk, replaces the asterisk with the leftmost remaining 1, and then halts. We can use these five-tuples to do this: $(s_0, 1, s_1, B, R)$, $(s_1, *, s_3, B, R)$, $(s_1, 1, s_2, B, R)$, $(s_2, 1, s_2, 1, R)$, and $(s_2, *, s_3, 1, R)$. 

Unfortunately, constructing Turing machines to compute relatively simple functions can be extremely demanding. For example, one Turing machine for multiplying two nonnegative integers found in many books has 31 five-tuples and 11 states. If it is challenging to construct Turing machines to compute even relatively simple functions, what hope do we have of building

Turing machines for more complicated functions? One way to simplify this problem is to use a multitape Turing machine that uses more than one tape simultaneously and to build up multitape Turing machines for the composition of functions. It can be shown that for any multitape Turing machine there is a one-tape Turing machine that can do the same thing.

Different Types of Turing Machines

There are many variations on the definition of a Turing machine. We can expand the capabilities of a Turing machine in a wide variety of ways. For example, we can allow a Turing machine to move right, left, or not at all at each step. We can allow a Turing machine to operate on multiple tapes, using $(2 + 3n)$ -tuples to describe the Turing machine when n tapes are used. We can allow the tape to be two-dimensional, where at each step we move up, down, right, or left, not just right or left as we do on a one-dimensional tape. We can allow multiple tape heads that read different cells simultaneously. Furthermore, we can allow a Turing machine to be **nondeterministic**, by allowing a (state, tape symbol) pair to possibly appear as the first elements in more than one five-tuple of the Turing machine. We can also reduce the capabilities of a Turing machine in different ways. For example, we can restrict the tape to be infinite in only one dimension or we can restrict the tape alphabet to have only two symbols. All these variations of Turing machines have been studied in detail.

The crucial point is that no matter which of these variations we use, or even which combination of variations we use, we never increase or decrease the power of the machine. Anything that one of these variations can do can be done by the Turing machine defined in this section, and vice versa. The reason that these variations are useful is that sometimes they make doing some particular job much easier than if the Turing machine defined in Definition 1 were used. They never extend the capability of the machine. Sometimes it is useful to have a wide variety of Turing machines with which to work. For example, one way to show that for every nondeterministic Turing machine, there is a deterministic Turing machine that recognizes the same language is to use a deterministic Turing machine with three tapes. (For details on variations of Turing machines and demonstrations of their equivalence, see [HoMoUl01].)

Besides introducing the notion of a Turing machine, Turing also showed that it is possible to construct a single Turing machine that can simulate the computations of every Turing machine when given an encoding of this target Turing machine and its input. Such a machine is called a **universal Turing machine**. (See a book on the theory of computation, such as [Si06], for more about universal Turing machines.)

The Church–Turing Thesis



Turing machines are relatively simple. They can have only finitely many states and they can read and write only one symbol at a time on a one-dimensional tape. But it turns out that Turing machines are extremely powerful. We have seen that Turing machines can be built to add numbers and to multiply numbers. Although it may be difficult to actually construct a Turing machine to compute a particular function that can be computed with an algorithm, such a Turing machine can always be found. This was the original goal of Turing when he invented his machines. Furthermore, there is a tremendous amount of evidence for the **Church–Turing thesis**, which states that given any problem that can be solved with an effective algorithm, there is a Turing machine that can solve this problem. The reason this is called a *thesis* rather than a theorem is that the concept of solvability by an effective algorithm is informal and imprecise, as opposed to the notion of solvability by a Turing machine, which is formal and precise. Certainly, though, any problem that can be solved using a computer with a program written in any language, perhaps using an unlimited amount of memory, should be considered effectively solvable. (Note that Turing machines have unlimited memory, unlike computers in the real world, which have only a finite amount of memory.)

Many different formal theories have been developed to capture the notion of effective computability. These include Turing's theory and Church's lambda-calculus, as well as theories proposed by Stephen Kleene and by E. L. Post. These theories seem quite different on the surface. The surprising thing is that they can be shown to be equivalent by demonstrating that they define exactly the same class of functions. With this evidence, it seems that Turing's original ideas, formulated before the invention of modern computers, describe the ultimate capabilities of these machines. The interested reader should consult books on the theory of computation, such as [HoMoUl01] and [Si96], for a discussion of these different theories and their equivalence.

For the remainder of this section we will briefly explore some of the consequences of the Church–Turing thesis and we will describe the importance of Turing machines in the study of the complexity of algorithms. Our goal will be to introduce some important ideas from theoretical computer science to entice the interested student to further study. We will cover a lot of ground quickly without providing explicit details. Our discussion will also tie together some of the concepts discussed in previous parts of the book with the theory of computation.

Computational Complexity, Computability, and Decidability

Throughout this book we have discussed the computational complexity of a wide variety of problems. We described the complexity of these problems in terms of the number of operations used by the most efficient algorithms that solve them. The basic operations used by algorithms differ considerably; we have measured the complexity of different algorithms in terms of bit operations, comparisons of integers, arithmetic operations, and so on. In Section 3.3, we defined various classes of problems in terms of their computational complexity. However, these definitions were not precise, because the types of operations used to measure their complexity vary so drastically. Turing machines provide a way to make the concept of computational complexity precise. If the Church–Turing thesis is true, it would then follow that if a problem can be solved using an effective algorithm, then there is a Turing machine that can solve this problem. When a Turing machine is used to solve a problem, the input to the problem is encoded as a string of symbols that is written on the tape of this Turing machine. How we encode input depends on the domain of this input. For example, as we have seen, we can encode a positive integer using a string of 1s. We can also devise ways to express pairs of integers, negative integers, and so on. Similarly, for graph algorithms, we need a way to encode graphs as strings of symbols. This can be done in many ways and can be based on adjacency lists or adjacency matrices. (We omit the details of how this is done.) However, the way input is encoded does not matter as long as it is relatively efficient, as a Turing machine can always change one encoding into another encoding. We will now use this model to make precise some of the notions concerning computational complexity that were informally introduced in Section 3.3.

The kind of problems that are most easily studied by using Turing machines are those problems that can be answered either by a “yes” or by a “no.”

DEFINITION 3

A *decision problem* asks whether statements from a particular class of statements are true. Decision problems are also known as *yes-or-no problems*.

Given a decision problem, we would like to know whether there is an algorithm that can determine whether statements from the class of statements it addresses are true. For example, consider the class of statements each of which asks whether a particular integer n is prime. This is a decision problem because the answer to the question “Is n prime?” is either yes or no. Given this decision problem, we can ask whether there is an algorithm that can decide whether each of the statements in the decision problem is true, that is, given an integer n , deciding whether n is prime. The answer is that there is such an algorithm. In particular, in Section 3.5 we discussed the algorithm that determines whether a positive integer n is prime by checking whether it is divisible by primes not exceeding its square root. (There are many other algorithms for determining whether a positive integer is prime.) The set of inputs for which the answer to

the yes–no problem is “yes” is a subset of the set of possible inputs, that is, it is a subset of the set of strings of the input alphabet. In other words, solving a yes–no problem is the same as recognizing the language consisting of all bit strings that represent input values to the problem leading to the answer “yes.” Consequently, solving a yes–no problem is the same as recognizing the language corresponding to the input values for which the answer to the problem is “yes.”

DECIDABILITY When there is an effective algorithm that decides whether instances of a decision problem are true, we say that this problem is **solvable** or **decidable**. For instance, the problem of determining whether a positive integer is prime is a solvable problem. However, if no effective algorithm exists for solving a problem, then we say the problem is **unsolvable** or **undecidable**. To show that a decision problem is solvable we need only construct an algorithm that can determine whether statements of the particular class are true. On the other hand, to show that a decision problem is unsolvable we need to prove that no such algorithm exists. (The fact that we tried to find such an algorithm but failed, does not prove the problem is unsolvable.)

By studying only decision problems, it may seem that we are studying only a small set of problems of interest. However, most problems can be recast as decision problems. Recasting the types of problems we have studied in this book as decision problems can be quite complicated, so we will not go into the details of this process here. The interested reader can consult references on the theory of computation, such as [Wo87], which, for example, explains how to recast the traveling salesperson problem (described in Section 9.6) as a decision problem. (To recast the traveling salesman problem as a decision problem, we first consider the decision problem that asks whether there is a Hamilton circuit of weight not exceeding k , where k is a positive integer. With some additional effort it is possible to use answers to this question for different values of k to find the smallest possible weight of a Hamilton circuit.)

In Section 3.1 we introduced the halting problem and proved that it is an unsolvable problem. That discussion was somewhat informal because the notion of a procedure was not precisely defined. A precise definition of the halting problem can be made in terms of Turing machines.

DEFINITION 4

The *halting problem* is the decision problem that asks whether a Turing machine T eventually halts when given an input string x .

With this definition of the halting problem, we have Theorem 1.

THEOREM 1

The halting problem is an unsolvable decision problem. That is, no Turing machine exists that, when given an encoding of a Turing machine T and its input string x as input, can determine whether T eventually halts when started with x written on its tape.

The proof of Theorem 1 given in Section 3.1 for the informal definition of the halting problem still applies here.

Other examples of unsolvable problems include:

- (i) the problem of determining whether two context-free grammars generate the same set of strings;
- (ii) the problem of determining whether a given set of tiles can be used with repetition allowed to cover the entire plane without overlap; and
- (iii) *Hilbert’s Tenth Problem*, which asks whether there are integer solutions to a given polynomial equation with integer coefficients. (This question occurs tenth on the famous list of 23 problems Hilbert posed in 1900. Hilbert envisioned that the work done to solve these problems would help further the progress of mathematics in the twentieth century. The unsolvability of Hilbert’s Tenth Problem was established in 1970 by Yuri Matiyasevich.)



COMPUTABILITY A function that can be computed by a Turing machine is called **computable** and a function that cannot be computed by a Turing machine is called **uncomputable**. It is fairly straightforward, using a countability argument, to show that there are number-theoretic functions that are not computable (see Exercise 39 in Section 2.5). However, it is not so easy to actually produce such a function. The **busy beaver function** defined in the preamble to Exercise 31 is an example of an uncomputable function. One way to show that the busy beaver function is not computable is to show that it grows faster than any computable function. (See Exercise 32.)

Note that every decision problem can be reformulated as the problem of computing a function, namely, the function that has the value 1 when the answer to the problem is “yes” and that has the value 0 when the answer to the problem is “no.” A decision problem is solvable if and only if the corresponding function constructed in this way is computable.

THE CLASSES P AND NP In Section 3.3 we informally defined the classes of problems called P and NP. We are now able to define these concepts precisely using the notions of deterministic and nondeterministic Turing machines.

We first elaborate on the difference between a deterministic Turing machine and a nondeterministic Turing machine. The Turing machines we have studied in this section have all been deterministic. In a deterministic Turing machine $T = (S, I, f, s_0)$, transition rules are defined by the partial function f from $S \times I$ to $S \times I \times \{R, L\}$. Consequently, when transition rules of the machine are represented as five-tuples of the form (s, x, s', x', d) , where s is the current state, x is the current tape symbol, s' is the next state, x' is the symbol that replaces x on the tape, and d is the direction the machine moves on the tape, no two transition rules begin with the same pair (s, x) .

In a nondeterministic Turing machine, allowed steps are defined using a relation consisting of five-tuples rather than using a partial function. The restriction that no two transition rules begin with the same pair (s, x) is eliminated; that is, there may be more than one transition rule beginning with each (state, tape symbol) pair. Consequently, in a nondeterministic Turing machine, there is a choice of transitions for some pairs of the current state and the tape symbol being read. At each step of the operation of a nondeterministic Turing machine, the machine picks one of the different choices of the transition rules that begin with the current state and tape symbol pair. This choice can be considered to be a “guess” of which step to use. Just as for deterministic Turing machines, a nondeterministic Turing machine halts when there is no transition rule in its definition that begins with the current state and tape symbol. Given a nondeterministic Turing machine T , we say that a string x is recognized by T if and only if there exists some sequence of transitions of T that ends in a final state when the machine starts in the initial position with x written on the tape. The nondeterministic Turing machine T recognizes the set A if x is recognized by T if and only if $x \in A$. The nondeterministic Turing machine T is said to solve a decision problem if it recognizes the set consisting of all input values for which the answer to the decision problem is yes.

DEFINITION 5

A decision problem is in P, the *class of polynomial-time problems*, if it can be solved by a deterministic Turing machine in polynomial time in terms of the size of its input. That is, a decision problem is in P if there is a deterministic Turing machine T that solves the decision problem and a polynomial $p(n)$ such that for all integers n , T halts in a final state after no more than $p(n)$ transitions whenever the input to T is a string of length n . A decision problem is in NP, the *class of nondeterministic polynomial-time problems*, if it can be solved by a nondeterministic Turing machine in polynomial time in terms of the size of its input. That is, a decision problem is in NP if there is a nondeterministic Turing machine T that solves the problem and a polynomial $p(n)$ such that for all integers n , T halts for every choice of transitions after no more than $p(n)$ transitions whenever the input to T is a string of length n .

Problems in P are called **tractable**, whereas problems not in P are called **intractable**. For a problem to be in P, a deterministic Turing machine must exist that can decide in polynomial time whether a particular statement of the class addressed by the decision problem is true. For example, determining whether an item is in a list of n elements is a tractable problem. (We will not provide details on how this fact can be shown; the basic ideas used in the analyses of algorithms earlier in the text can be adapted when Turing machines are employed.) For a problem to be in NP, it is necessary only that there be a nondeterministic Turing machine that, when given a true statement from the set of statements addressed by the problem, can verify its truth in polynomial time by making the correct guess at each step from the set of allowable steps corresponding to the current state and tape symbol. The problem of determining whether a given graph has a Hamilton circuit is an NP problem, because a nondeterministic Turing machine can easily verify that a simple circuit in a graph passes through each vertex exactly once. It can do this by making a series of correct guesses corresponding to successively adding edges to form the circuit. Because every deterministic Turing machine can also be considered to be a nondeterministic Turing machine where each (state, tape symbol) pair occurs in exactly one transition rule defining the machine, every problem in P is also in NP. In symbols, $P \subseteq NP$.

One of the most perplexing open questions in theoretical computer science is whether every problem in NP is also in P, that is, whether $P = NP$. As mentioned in Section 3.3, there is an important class of problems, the class of NP-complete problems, such that a problem is in this class if it is in the class NP and if it can be shown that if it is also in the class P, then *every* problem in the class NP must also be in the class P. That is, a problem is NP-complete if the existence of a polynomial-time algorithm for solving it implies the existence of a polynomial-time algorithm for every problem in NP. In this book we have discussed several different NP-complete problems, such as determining whether a simple graph has a Hamilton circuit and determining whether a proposition in n -variables is a tautology.

Exercises

1. Let T be the Turing machine defined by the five-tuples: $(s_0, 0, s_1, 1, R)$, $(s_0, 1, s_1, 0, R)$, $(s_0, B, s_1, 0, R)$, $(s_1, 0, s_2, 1, L)$, $(s_1, 1, s_1, 0, R)$, and $(s_1, B, s_2, 0, L)$. For each of these initial tapes, determine the final tape when T halts, assuming that T begins in initial position.

- a) ... | B | B | 0 | 0 | 1 | 1 | B | B | ...
- b) ... | B | B | 1 | 0 | 1 | B | B | B | ...
- c) ... | B | B | 1 | 1 | B | 0 | 1 | B | ...
- d) ... | B | B | B | B | B | B | B | B | ...

2. Let T be the Turing machine defined by the five-tuples: $(s_0, 0, s_1, 0, R)$, $(s_0, 1, s_1, 0, L)$, $(s_0, B, s_1, 1, R)$, $(s_1, 0, s_2, 1, R)$, $(s_1, 1, s_1, 1, R)$, $(s_1, B, s_2, 0, R)$, and $(s_2, B, s_3, 0, R)$. For each of these initial tapes, determine the final tape when T halts, assuming that T begins in initial position.

- a) ... | B | B | 0 | 1 | 0 | 1 | B | B | ...
- b) ... | B | B | 1 | 1 | 1 | B | B | B | ...
- c) ... | B | B | 0 | 0 | B | 0 | 0 | B | ...
- d) ... | B | B | B | B | B | B | B | B | ...



ALONZO CHURCH (1903–1995) Alonzo Church was born in Washington, D.C. He studied at Göttingen under Hilbert and later in Amsterdam. He was a member of the faculty at Princeton University from 1927 until 1967 when he moved to UCLA. Church was one of the founding members of the Association for Symbolic Logic. He made many substantial contributions to the theory of computability, including his solution to the decision problem, his invention of the lambda-calculus, and, of course, his statement of what is now known as the Church-Turing thesis. Among Church's students were Stephen Kleene and Alan Turing. He published articles past his 90th birthday.

- 3.** What does the Turing machine described by the five-tuples $(s_0, 0, s_0, 0, R)$, $(s_0, 1, s_1, 0, R)$, (s_0, B, s_2, B, R) , $(s_1, 0, s_1, 0, R)$, $(s_1, 1, s_0, 1, R)$, and (s_1, B, s_2, B, R) do when given
- 11 as input?
 - an arbitrary bit string as input?
- 4.** What does the Turing machine described by the five-tuples $(s_0, 0, s_0, 1, R)$, $(s_0, 1, s_0, 1, R)$, (s_0, B, s_1, B, L) , $(s_1, 1, s_2, 1, R)$, do when given
- 101 as input?
 - an arbitrary bit string as input?
- 5.** What does the Turing machine described by the five-tuples $(s_0, 1, s_1, 0, R)$, $(s_1, 1, s_1, 1, R)$, $(s_1, 0, s_2, 0, R)$, $(s_2, 0, s_3, 1, L)$, $(s_2, 1, s_2, 1, R)$, $(s_3, 1, s_3, 1, L)$, $(s_3, 0, s_4, 0, L)$, $(s_4, 1, s_4, 1, L)$, and $(s_4, 0, s_0, 1, R)$ do when given
- 11 as input?
 - a bit string consisting entirely of 1s as input?
- 6.** Construct a Turing machine with tape symbols 0, 1, and B that, when given a bit string as input, adds a 1 to the end of the bit string and does not change any of the other symbols on the tape.
- 7.** Construct a Turing machine with tape symbols 0, 1, and B that, when given a bit string as input, replaces the first 0 with a 1 and does not change any of the other symbols on the tape.
- 8.** Construct a Turing machine with tape symbols 0, 1, and B that, given a bit string as input, replaces all 0s on the tape with 1s and does not change any of the 1s on the tape.
- 9.** Construct a Turing machine with tape symbols 0, 1, and B that, given a bit string as input, replaces all but the leftmost 1 on the tape with 0s and does not change any of the other symbols on the tape.
- 10.** Construct a Turing machine with tape symbols 0, 1, and B that, given a bit string as input, replaces the first two consecutive 1s on the tape with 0s and does not change any of the other symbols on the tape.
- 11.** Construct a Turing machine that recognizes the set of all bit strings that end with a 0.
- 12.** Construct a Turing machine that recognizes the set of all bit strings that contain at least two 1s.
- 13.** Construct a Turing machine that recognizes the set of all bit strings that contain an even number of 1s.
- 14.** Show at each step the contents of the tape of the Turing machine in Example 3 starting with each of these strings.
a) 0011 b) 00011 c) 101100 d) 000111
- 15.** Explain why the Turing machine in Example 3 recognizes a bit string if and only if this string is of the form $0^n 1^n$ for some positive integer n .
- *16.** Construct a Turing machine that recognizes the set $\{0^{2n} 1^n \mid n \geq 0\}$.
- *17.** Construct a Turing machine that recognizes the set $\{0^n 1^n 2^n \mid n \geq 0\}$.
- 18.** Construct a Turing machine that computes the function $f(n) = n + 2$ for all nonnegative integers n .
- 19.** Construct a Turing machine that computes the function $f(n) = n - 3$ if $n \geq 3$ and $f(n) = 0$ for $n = 0, 1, 2$ for all nonnegative integers n .
- 20.** Construct a Turing machine that computes the function $f(n) = n \bmod 3$ for every nonnegative integer n .
- 21.** Construct a Turing machine that computes the function $f(n) = 3$ if $n \geq 5$ and $f(n) = 0$ if $n = 0, 1, 2, 3$, or 4.
- 22.** Construct a Turing machine that computes the function $f(n) = 2n$ for all nonnegative integers n .
- 23.** Construct a Turing machine that computes the function $f(n) = 3n$ for all nonnegative integers n .
- 24.** Construct a Turing machine that computes the function $f(n_1, n_2) = n_2 + 2$ for all pairs of nonnegative integers n_1 and n_2 .
- *25.** Construct a Turing machine that computes the function $f(n_1, n_2) = \min(n_1, n_2)$ for all nonnegative integers n_1 and n_2 .
- 26.** Construct a Turing machine that computes the function $f(n_1, n_2) = n_1 + n_2 + 1$ for all nonnegative integers n_1 and n_2 .
- Suppose that T_1 and T_2 are Turing machines with disjoint sets of states S_1 and S_2 and with transition functions f_1 and f_2 , respectively. We can define the Turing machine $T_1 T_2$, the **composite** of T_1 and T_2 , as follows. The set of states of $T_1 T_2$ is $S_1 \cup S_2$. $T_1 T_2$ begins in the start state of T_1 . It first executes the transitions of T_1 using f_1 up to, but not including, the step at which T_1 would halt. Then, for all moves for which T_1 halts, it executes the same transitions of T_1 except that it moves to the start state of T_2 . From this point on, the moves of $T_1 T_2$ are the same as the moves of T_2 .
- 27.** By finding the composite of the Turing machines you constructed in Exercises 18 and 22, construct a Turing machine that computes the function $f(n) = 2n + 2$.
- 28.** By finding the composite of the Turing machines you constructed in Exercises 18 and 23, construct a Turing machine that computes the function $f(n) = 3(n + 2) = 3n + 6$.
- 29.** Which of the following problems is a decision problem?
 - What is the smallest prime greater than n ?
 - Is a graph G bipartite?
 - Given a set of strings, is there a finite-state automaton that recognizes this set of strings?
 - Given a checkerboard and a particular type of polyomino (see Section 1.8), can this checkerboard be tiled using polyominoes of this type?
- 30.** Which of the following problems is a decision problem?
 - Is the sequence a_1, a_2, \dots, a_n of positive integers in increasing order?

- b) Can the vertices of a simple graph G be colored using three colors so that no two adjacent vertices are the same color?
- c) What is the vertex of highest degree in a graph G ?
- d) Given two finite-state machines, do these machines recognize the same language?

 Let $B(n)$ be the maximum number of 1s that a Turing machine with n states with the alphabet $\{1, B\}$ may print on a tape that is initially blank. The problem of determining $B(n)$ for particular values of n is known as the **busy beaver problem**. This problem was first studied by Tibor Rado in 1962. Currently it is known that $B(2) = 4$, $B(3) = 6$, and $B(4) = 13$, but $B(n)$

is not known for $n \geq 5$. $B(n)$ grows rapidly; it is known that $B(5) \geq 4098$ and $B(6) \geq 3.5 \times 10^{18267}$.

- *31. Show that $B(2)$ is at least 4 by finding a Turing machine with two states and alphabet $\{1, B\}$ that halts with four consecutive 1s on the tape.
- **32. Show that the function $B(n)$ cannot be computed by any Turing machine. [Hint: Assume that there is a Turing machine that computes $B(n)$ in binary. Build a Turing machine T that, starting with a blank tape, writes n down in binary, computes $B(n)$ in binary, and converts $B(n)$ from binary to unary. Show that for sufficiently large n , the number of states of T is less than $B(n)$, leading to a contradiction.]

Key Terms and Results

TERMS

- alphabet (or vocabulary):** a set that contains elements used to form strings
- language:** a subset of the set of all strings over an alphabet
- phrase-structure grammar (V, T, S, P):** a description of a language containing an alphabet V , a set of terminal symbols T , a start symbol S , and a set of productions P
- the production $w \rightarrow w_1$:** w can be replaced by w_1 whenever it occurs in a string in the language
- $w_1 \Rightarrow w_2$ (w_2 is directly derivable from w_1):** w_2 can be obtained from w_1 using a production to replace a string in w_1 with another string
- $w_1 \stackrel{*}{\Rightarrow} w_2$ (w_2 is derivable from w_1):** w_2 can be obtained from w_1 using a sequence of productions to replace strings by other strings
- type 0 grammar:** any phrase-structure grammar
- type 1 grammar:** a phrase-structure grammar in which every production is of the form $w_1 \rightarrow w_2$, where $w_1 = lAr$ and $w_2 = lwr$, where $A \in N$, $l, r, w \in (N \cup T)^*$ and $w \neq \lambda$, or $w_1 = S$ and $w_2 = \lambda$ as long as S is not on the right-hand side of another production
- type 2, or context-free, grammar:** a phrase-structure grammar in which every production is of the form $A \rightarrow w_1$, where A is a nonterminal symbol
- type 3, or regular, grammar:** a phrase-structure grammar where every production is of the form $A \rightarrow aB$, $A \rightarrow a$, or $S \rightarrow \lambda$, where A and B are nonterminal symbols, S is the start symbol, and a is a terminal symbol
- derivation (or parse) tree:** an ordered rooted tree where the root represents the starting symbol of a type 2 grammar, internal vertices represent nonterminals, leaves represent terminals, and the children of a vertex are the symbols on the right side of a production, in order from left to right, where the symbol represented by the parent is on the left-hand side
- Backus–Naur form:** a description of a context-free grammar in which all productions having the same nonterminal as

their left-hand side are combined with the different right-hand sides of these productions, each separated by a bar, with nonterminal symbols enclosed in angular brackets and the symbol \rightarrow replaced by $::=$

finite-state machine (S, I, O, f, g, s_0) (or a Mealy machine): a six-tuple containing a set S of states, an input alphabet I , an output alphabet O , a transition function f that assigns a next state to every pair of a state and an input, an output function g that assigns an output to every pair of a state and an input, and a starting state s_0

AB (concatenation of A and B): the set of all strings formed by concatenating a string in A and a string in B in that order

A^* (Kleene closure of A): the set of all strings made up by concatenating arbitrarily many strings from A

deterministic finite-state automaton (S, I, f, s_0, F): a five-tuple containing a set S of states, an input alphabet I , a transition function f that assigns a next state to every pair of a state and an input, a starting state s_0 , and a set of final states F

nondeterministic finite-state automaton (S, I, f, s_0, F): a five-tuple containing a set S of states, an input alphabet I , a transition function f that assigns a set of possible next states to every pair of a state and an input, a starting state s_0 , and a set of final states F

language recognized by an automaton: the set of input strings that take the start state to a final state of the automaton

regular expression: an expression defined recursively by specifying that \emptyset , λ , and x , for all x in the input alphabet, are regular expressions, and that (AB) , $(A \cup B)$, and A^* are regular expressions when A and B are regular expressions

regular set: a set defined by a regular expression (see page 820)

Turing machine $T = (S, I, f, s_0)$: a four-tuple consisting of a finite set S of states, an alphabet I containing the blank symbol B , a partial function f from $S \times I$ to $S \times I \times \{R, L\}$, and a starting state s_0

nondeterministic Turing machine: a Turing machine that may have more than one transition rule corresponding to each (state, tape symbol) pair

decision problem: a problem that asks whether statements from a particular class of statements are true

solvable problem: a problem with the property that there is an effective algorithm that can solve all instances of the problem

unsolvable problem: a problem with the property that no effective algorithm exists that can solve all instances of the problem

computable function: a function whose values can be computed using a Turing machine

uncomputable function: a function whose values cannot be computed using a Turing machine

P, the class of polynomial-time problems: the class of problems that can be solved by a deterministic Turing machine in polynomial time in terms of the size of the input

NP, the class of nondeterministic polynomial-time problems: the class of problems that can be solved by a nondeter-

ministic Turing machine in polynomial time in terms of the size of the input

NP-complete: a subset of the class of NP problems with the property that if any one of them is in the class P, then all problems in NP are in the class P

RESULTS

For every nondeterministic finite-state automaton there is a deterministic finite-state automaton that recognizes the same set.

Kleene's theorem: A set is regular if and only if there is a finite-state automaton that recognizes it.

A set is regular if and only if it is generated by a regular grammar.

The halting problem is unsolvable.

Review Questions

1. a) Define a phrase-structure grammar.
b) What does it mean for a string to be derivable from a string w by a phrase-structure grammar G ?
 2. a) What is the language generated by a phrase-structure grammar G ?
b) What is the language generated by the grammar G with vocabulary $\{S, 0, 1\}$, set of terminals $T = \{0, 1\}$, starting symbol S , and productions $S \rightarrow 000S, S \rightarrow 1$?
c) Give a phrase-structure grammar that generates the set $\{01^n \mid n = 0, 1, 2, \dots\}$.
 3. a) Define a type 1 grammar.
b) Give an example of a grammar that is not a type 1 grammar.
c) Define a type 2 grammar.
d) Give an example of a grammar that is not a type 2 grammar but is a type 1 grammar.
e) Define a type 3 grammar.
f) Give an example of a grammar that is not a type 3 grammar but is a type 2 grammar.
 4. a) Define a regular grammar.
b) Define a regular language.
c) Show that the set $\{0^m 1^n \mid m, n = 0, 1, 2, \dots\}$ is a regular language.
 5. a) What is Backus–Naur form?
b) Give an example of the Backus–Naur form of the grammar for a subset of English of your choice.
 6. a) What is a finite-state machine?
b) Show how a vending machine that accepts only quarters and dispenses a soft drink after 75 cents has been deposited can be modeled using a finite-state machine.
 7. Find the set of strings recognized by the deterministic finite-state automaton shown here.
-
8. Construct a deterministic finite-state automaton that recognizes the set of bit strings that start with 1 and end with 1.
 9. a) What is the Kleene closure of a set of strings?
b) Find the Kleene closure of the set $\{11, 0\}$.
 10. a) Define a finite-state automaton.
b) What does it mean for a string to be recognized by a finite-state automaton?
 11. a) Define a nondeterministic finite-state automaton.
b) Show that given a nondeterministic finite-state automaton, there is a deterministic finite-state automaton that recognizes the same language.
 12. a) Define the set of regular expressions over a set I .
b) Explain how regular expressions are used to represent regular sets.
 13. State Kleene's theorem.
 14. Show that a set is generated by a regular grammar if and only if it is a regular set.
 15. Give an example of a set not recognized by a finite-state automaton. Show that no finite-state automaton recognizes it.
 16. Define a Turing machine.
 17. Describe how Turing machines are used to recognize sets.
 18. Describe how Turing machines are used to compute number-theoretic functions.
 19. What is an unsolvable decision problem? Give an example of such a problem.

Supplementary Exercises

- *1. Find a phrase-structure grammar that generates each of these languages.

- a) the set of bit strings of the form $0^{2n}1^{3n}$, where n is a nonnegative integer
- b) the set of bit strings with twice as many 0s as 1s
- c) the set of bit strings of the form w^2 , where w is a bit string

- *2. Find a phrase-structure grammar that generates the set $\{0^{2^n} \mid n \geq 0\}$.

For Exercises 3 and 4, let $G = (V, T, S, P)$ be the context-free grammar with $V = \{(), S, A, B\}$, $T = \{(), \}$, starting symbol S , and productions $S \rightarrow A$, $A \rightarrow AB$, $A \rightarrow B$, $B \rightarrow (A)$, and $B \rightarrow ()$, $S \rightarrow \lambda$.

3. Construct the derivation trees of these strings.

- a) $(())$
- b) $((())()$
- c) $((((())()$

- *4. Show that $L(G)$ is the set of all balanced strings of parentheses, defined in the preamble to Supplementary Exercise 55 in Chapter 4.

A context-free grammar is **ambiguous** if there is a word in $L(G)$ with two derivations that produce different derivation trees, considered as ordered, rooted trees.

5. Show that the grammar $G = (V, T, S, P)$ with $V = \{0, S\}$, $T = \{0\}$, starting state S , and productions $S \rightarrow 0S$, $S \rightarrow S0$, and $S \rightarrow 0$ is ambiguous by constructing two different derivation trees for 0^3 .

6. Show that the grammar $G = (V, T, S, P)$ with $V = \{0, S\}$, $T = \{0\}$, starting state S , and productions $S \rightarrow 0S$ and $S \rightarrow 0$ is unambiguous.

7. Suppose that A and B are finite subsets of V^* , where V is an alphabet. Is it necessarily true that $|AB| = |BA|$?

8. Prove or disprove each of these statements for subsets A , B , and C of V^* , where V is an alphabet.

- a) $A(B \cup C) = AB \cup AC$
- b) $A(B \cap C) = AB \cap AC$
- c) $(AB)C = A(BC)$
- d) $(A \cup B)^* = A^* \cup B^*$

9. Suppose that A and B are subsets of V^* , where V is an alphabet. Does it follow that $A \subseteq B$ if $A^* \subseteq B^*$?

10. What set of strings with symbols in the set $\{0, 1, 2\}$ is represented by the regular expression $(2^*)(0 \cup (12^*))^*$?

The **star height** $h(E)$ of a regular expression over the set I is defined recursively by

$$\begin{aligned} h(\emptyset) &= 0; \\ h(x) &= 0 \text{ if } x \in I; \\ h(E_1 \cup E_2) &= h(E_1 E_2) = \max(h(E_1), h(E_2)) \\ &\quad \text{if } E_1 \text{ and } E_2 \text{ are regular expressions;} \\ h(E^*) &= h(E) + 1 \text{ if } E \text{ is a regular expression.} \end{aligned}$$

11. Find the star height of each of these regular expressions.

- a) 0^*1
- b) 0^*1^*
- c) $(0^*01)^*$
- d) $((0^*1)^*)^*$

- e) $(010^*)(1^*01^*)^*((01)^*(10)^*)^*$

- f) $((((0^*)1)^*0)^*)1^*$

- *12. For each of these regular expressions find a regular expression that represents the same language with minimum star height.

- a) $(0^*1^*)^*$

- b) $(0(01^*0)^*)^*$

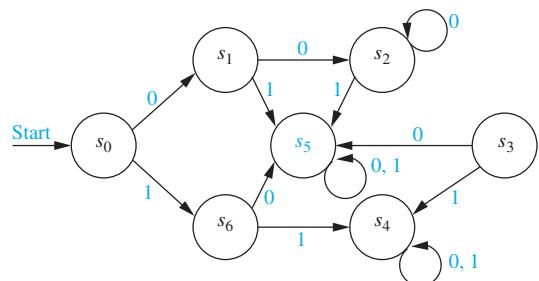
- c) $(0^* \cup (01)^* \cup 1^*)^*$

13. Construct a finite-state machine with output that produces an output of 1 if the bit string read so far as input contains four or more 1s. Then construct a deterministic finite-state automaton that recognizes this set.

14. Construct a finite-state machine with output that produces an output of 1 if the bit string read so far as input contains four or more consecutive 1s. Then construct a deterministic finite-state automaton that recognizes this set.

15. Construct a finite-state machine with output that produces an output of 1 if the bit string read so far as input ends with four or more consecutive 1s. Then construct a deterministic finite-state automaton that recognizes this set.

16. A state s' in a finite-state machine is said to be **reachable** from state s if there is an input string x such that $f(s, x) = s'$. A state s is called **transient** if there is no nonempty input string x with $f(s, x) = s$. A state s is called a **sink** if $f(s, x) = s$ for all input strings x . Answer these questions about the finite-state machine with the state diagram illustrated here.



- a) Which states are reachable from s_0 ?

- b) Which states are reachable from s_2 ?

- c) Which states are transient?

- d) Which states are sinks?

- *17. Suppose that S , I , and O are finite sets such that $|S| = n$, $|I| = k$, and $|O| = m$.

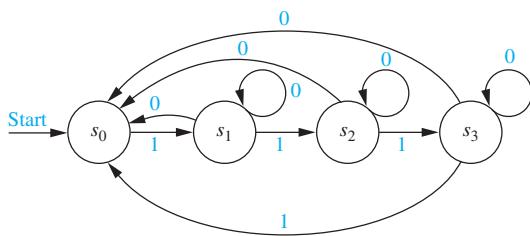
- a) How many different finite-state machines (Mealy machines) $M = (S, I, O, f, g, s_0)$ can be constructed, where the starting state s_0 can be arbitrarily chosen?

- b) How many different Moore machines $M = (S, I, O, f, g, s_0)$ can be constructed, where the starting state s_0 can be arbitrarily chosen?

- *18. Suppose that S and I are finite sets such that $|S| = n$ and $|I| = k$. How many different finite-state automata $M = (S, I, f, s_0, F)$ are there where the starting state s_0

and the subset F of S consisting of final states can be chosen arbitrarily

- a) if the automata are deterministic?
 - b) if the automata may be nondeterministic? (Note: This includes deterministic automata.)
19. Construct a deterministic finite-state automaton that is equivalent to the nondeterministic automaton with the state diagram shown here.



20. What is the language recognized by the automaton in Exercise 19?
21. Construct finite-state automata that recognize these sets.
- a) $0^*(10)^*$
 - b) $(01 \cup 111)^*10^*(0 \cup 1)$
 - c) $(001 \cup (11)^*)^*$
- *22. Find regular expressions that represent the set of all strings of 0s and 1s
- a) made up of blocks of even numbers of 1s interspersed with odd numbers of 0s.
 - b) with at least two consecutive 0s or three consecutive 1s.

- c) with no three consecutive 0s or two consecutive 1s.
- *23. Show that if A is a regular set, then so is \overline{A} .
- *24. Show that if A and B are regular sets, then so is $A \cap B$.
- *25. Find finite-state automata that recognize these sets of strings of 0s and 1s.

 - a) the set of all strings that start with no more than three consecutive 0s and contain at least two consecutive 1s
 - b) the set of all strings with an even number of symbols that do not contain the pattern 101
 - c) the set of all strings with at least three blocks of two or more 1s and at least two 0s

- *26. Show that $\{0^{2^n} \mid n \in \mathbb{N}\}$ is not regular. You may use the pumping lemma given in Exercise 22 of Section 13.4.
- *27. Show that $\{1^p \mid p \text{ is prime}\}$ is not regular. You may use the pumping lemma given in Exercise 22 of Section 13.4.
- *28. There is a result for context-free languages analogous to the pumping lemma for regular sets. Suppose that $L(G)$ is the language recognized by a context-free language G . This result states that there is a constant N such that if z is a word in $L(G)$ with $l(z) \geq N$, then z can be written as $uvwxy$, where $l(vwx) \leq N$, $l(vx) \geq 1$, and $uv^iwx^i y$ belongs to $L(G)$ for $i = 0, 1, 2, 3, \dots$. Use this result to show that there is no context-free grammar G with $L(G) = \{0^n 1^n 2^n \mid n = 0, 1, 2, \dots\}$.
- *29. Construct a Turing machine that computes the function $f(n_1, n_2) = \max(n_1, n_2)$.
- *30. Construct a Turing machine that computes the function $f(n_1, n_2) = n_2 - n_1$ if $n_2 \geq n_1$ and $f(n_1, n_2) = 0$ if $n_2 < n_1$.

Computer Projects

Write programs with these input and output.

1. Given the productions in a phrase-structure grammar, determine which type of grammar this is in the Chomsky classification scheme.
2. Given the productions of a phrase-structure grammar, find all strings that are generated using twenty or fewer applications of its production rules.
3. Given the Backus–Naur form of a type 2 grammar, find all strings that are generated using twenty or fewer applications of the rules defining it.
- *4. Given the productions of a context-free grammar and a string, produce a derivation tree for this string if it is in the language generated by this grammar.
5. Given the state table of a Moore machine and an input string, produce the output string generated by the machine.
6. Given the state table of a Mealy machine and an input string, produce the output string generated by the machine.
7. Given the state table of a deterministic finite-state automaton and a string, decide whether this string is recognized by the automaton.
8. Given the state table of a nondeterministic finite-state automaton and a string, decide whether this string is recognized by the automaton.
- *9. Given the state table of a nondeterministic finite-state automaton, construct the state table of a deterministic finite-state automaton that recognizes the same language.
- **10. Given a regular expression, construct a nondeterministic finite-state automaton that recognizes the set that this expression represents.
11. Given a regular grammar, construct a finite-state automaton that recognizes the language generated by this grammar.
12. Given a finite-state automaton, construct a regular grammar that generates the language recognized by this automaton.
- *13. Given a Turing machine, find the output string produced by a given input string.

Computations and Explorations

Use a computational program or programs you have written to do these exercises.

1. Solve the busy beaver problem for two states by testing all possible Turing machines with two states and alphabet $\{1, B\}$.
- *2. Solve the busy beaver problem for three states by testing all possible Turing machines with three states and alphabet $\{1, B\}$.
- **3. Find a busy beaver machine with four states by testing all possible Turing machines with four states and alphabet $\{1, B\}$.
- **4. Make as much progress as you can toward finding a busy beaver machine with five states.
- **5. Make as much progress as you can toward finding a busy beaver machine with six states.

Writing Projects

Respond to these with essays using outside sources.

1. Describe how the growth of certain types of plants can be modeled using a Lindenmeyer system. Such a system uses a grammar with productions modeling the different ways plants can grow.
2. Describe the Backus–Naur form (and extended Backus–Naur form) rules used to specify the syntax of a programming language, such as Java, LISP, or Ada, or the database language SQL.
3. Explain how finite-state machines are used by spell-checkers.
4. Explain how finite-state machines are used in the study of network protocols.
5. Explain how finite-state machines are used in speech recognition programs.
6. Compare the use of Moore machines versus Mealy machines in the design of hardware systems and computer software.
7. Explain the concept of minimizing finite-state automata. Give an algorithm that carries out this minimization.
8. Give the definition of cellular automata. Explain their applications. Use the Game of Life as an example.
9. Define a pushdown automaton. Explain how pushdown automata are used to recognize sets. Which sets are recognized by pushdown automata? Provide an outline of a proof justifying your answer.
10. Define a linear-bounded automaton. Explain how linear-bounded automata are used to recognize sets. Which sets are recognized by linear-bounded automata? Provide an outline of a proof justifying your answer.
11. Look up Turing’s original definition of what we now call a Turing machine. What was his motivation for defining these machines?
12. Describe the concept of the universal Turing machine. Explain how such a machine can be built.
13. Explain the kinds of applications in which nondeterministic Turing machines are used instead of deterministic Turing machines.
14. Show that a Turing machine can simulate any action of a nondeterministic Turing machine.
15. Show that a set is recognized by a Turing machine if and only if it is generated by a phrase-structure grammar.
16. Describe the basic concepts of the lambda-calculus and explain how it is used to study computability of functions.
17. Show that a Turing machine as defined in this chapter can do anything a Turing machine with n tapes can do.
18. Show that a Turing machine with a tape infinite in one direction can do anything a Turing machine with a tape infinite in both directions can do.

1

Axioms for the Real Numbers and the Positive Integers

In this book we have assumed an explicit set of axioms for the set of real numbers and for the set of positive integers. In this appendix we will list these axioms and we will illustrate how basic facts, also used without proof in the text, can be derived using them.

Axioms for Real Numbers

The standard axioms for real numbers include both the **field** (or **algebraic**) **axioms**, used to specify rules for basic arithmetic operations, and the **order axioms**, used to specify properties of the ordering of real numbers.

THE FIELD AXIOMS We begin with the field axioms. As usual, we denote the sum and product of two real numbers x and y by $x + y$ and $x \cdot y$, respectively. (Note that the product of x and y is often denoted by xy without the use of the dot to indicate multiplication. We will not use this abridged notation in this appendix, but will within the text.) Also, by convention, we perform multiplications before additions unless parentheses are used. Although these statements are axioms, they are commonly called *laws* or *rules*. The first two of these axioms tell us that when we add or multiply two real numbers, the result is again a real number; these are the *closure laws*.

- **Closure law for addition** For all real numbers x and y , $x + y$ is a real number.
- **Closure law for multiplication** For all real numbers x and y , $x \cdot y$ is a real number.

The next two axioms tell us that when we add or multiply three real numbers, we get the same result regardless of the order of operations; these are the *associative laws*.

- **Associative law for addition** For all real numbers x , y , and z , $(x + y) + z = x + (y + z)$.
- **Associative law for multiplication** For all real numbers x , y , and z , $(x \cdot y) \cdot z = x \cdot (y \cdot z)$.

Two additional algebraic axioms tell us that the order in which we add or multiply two numbers does not matter; these are the *commutative laws*.

- **Commutative law for addition** For all real numbers x and y , $x + y = y + x$.
- **Commutative law for multiplication** For all real numbers x and y , $x \cdot y = y \cdot x$.

The next two axioms tell us that 0 and 1 are additive and multiplicative identities for the set of real numbers. That is, when we add 0 to a real number or multiply a real number by 1 we do not change this real number. These laws are called *identity laws*.

- **Additive identity law** For every real number x , $x + 0 = 0 + x = x$.
- **Multiplicative identity law** For every real number x , $x \cdot 1 = 1 \cdot x = x$.

Although it seems obvious, we also need the following axiom.

- **Identity elements axiom** The additive identity 0 and the multiplicative identity 1 are distinct, that is $0 \neq 1$.

Two additional axioms tell us that for every real number, there is a real number that can be added to this number to produce 0, and for every nonzero real number, there is a real number by which it can be multiplied to produce 1. These are the *inverse laws*.

- **Inverse law for addition** For every real number x , there exists a real number $-x$ (called the *additive inverse* of x) such that $x + (-x) = (-x) + x = 0$.
- **Inverse law for multiplication** For every nonzero real number x , there exists a real number $1/x$ (called the *multiplicative inverse* of x) such that $x \cdot (1/x) = (1/x) \cdot x = 1$.

The final algebraic axioms for real numbers are the *distributive laws*, which tell us that multiplication distributes over addition; that is, that we obtain the same result when we first add a pair of real numbers and then multiply by a third real number or when we multiply each of these two real numbers by the third real number and then add the two products.

- **Distributive laws** For all real numbers x , y , and z , $x \cdot (y + z) = x \cdot y + x \cdot z$ and $(x + y) \cdot z = x \cdot z + y \cdot z$.

ORDER AXIOMS Next, we will state the *order axioms* for the real numbers, which specify properties of the “greater than” relation, denoted by $>$, on the set of real numbers. We write $x > y$ (and $y < x$) when x is greater than y , and we write $x \geq y$ (and $y \leq x$) when $x > y$ or $x = y$. The first of these axioms tells us that given two real numbers, exactly one of three possibilities occurs: the two numbers are equal, the first is greater than the second, or the second is greater than the first. This rule is called the *trichotomy law*.

- **Trichotomy law** For all real numbers x and y , exactly one of $x = y$, $x > y$, or $y > x$ is true.

Next, we have an axiom, called the *transitivity law*, that tells us that if one number is greater than a second number and this second number is greater than a third, then the first number is greater than the third.

- **Transitivity law** For all real numbers x , y , and z , if $x > y$ and $y > z$, then $x > z$.

We also have two *compatibility laws*, which tell us that when we add a number to both sides in a greater than relationship, the greater than relationship is preserved and when we multiply both sides of a greater than relationship by a *positive real number* (that is, a real number x with $x > 0$), the greater than relationship is preserved.

- **Additive compatibility law** For all real numbers x , y , and z , if $x > y$, then $x + z > y + z$.
- **Multiplicative compatibility law** For all real numbers x , y , and z , if $x > y$ and $z > 0$, then $x \cdot z > y \cdot z$.

We leave it to the reader (see Exercise 15) to prove that for all real numbers x , y , and z , if $x > y$ and $z < 0$, then $x \cdot z < y \cdot z$. That is, multiplication of an inequality by a negative real number reverses the direction of the inequality.

The final axiom for the set of real numbers is the *completeness property*. Before we state this axiom, we need some definitions. First, given a nonempty set A of real numbers, we say that the real number b is an **upper bound** of A if for every real number a in A , $b \geq a$. A real number s is a **least upper bound** of A if s is an upper bound of A and whenever t is an upper bound of A , then we have $s \leq t$.

- **Completeness property** Every nonempty set of real numbers that is bounded above has a least upper bound.

Using Axioms to Prove Basic Facts

The axioms we have listed can be used to prove many properties that are often used without explicit mention. We give several examples of results we can prove using axioms and leave the proof of a variety of other properties as exercises. Although the results we will prove seem quite obvious, proving them using only the axioms we have stated can be challenging.

THEOREM 1

The additive identity element 0 of the real numbers is unique.

Proof: To show that the additive identity element 0 of the real numbers is unique, suppose that $0'$ is also an additive identity for the real numbers. This means that $0' + x = x + 0' = x$ whenever x is a real number. By the additive identity law, it follows that $0 + 0' = 0'$. Because $0'$ is an additive identity, we know that $0 + 0' = 0$. It follows that $0 = 0'$, because both equal $0 + 0'$. This shows that 0 is the unique additive identity for the real numbers. \triangleleft

THEOREM 2

The additive inverse of a real number x is unique.

Proof: Let x be a real number. Suppose that y and z are both additive inverses of x . Then,

$$\begin{aligned} y &= 0 + y && \text{by the additive identity law} \\ &= (z + x) + y && \text{because } z \text{ is an additive inverse of } x \\ &= z + (x + y) && \text{by the associative law for addition} \\ &= z + 0 && \text{because } y \text{ is an additive inverse of } x \\ &= z && \text{by the additive identity law.} \end{aligned}$$

It follows that $y = z$. \triangleleft

Theorems 1 and 2 tell us that the additive identity and additive inverses are unique. Theorems 3 and 4 tell us that the multiplicative identity and multiplicative inverses of nonzero real numbers are also unique. We leave their proofs as exercises.

THEOREM 3

The multiplicative identity element 1 of the real numbers is unique.

THEOREM 4

The multiplicative inverse of a nonzero real number x is unique.

THEOREM 5

For every real number x , $x \cdot 0 = 0$.

Proof: Suppose that x is a real number. By the additive inverse law, there is a real number y that is the additive inverse of $x \cdot 0$, so we have $x \cdot 0 + y = 0$. By the additive identity law, $0 + 0 = 0$. Using the distributive law, we see that $x \cdot 0 = x \cdot (0 + 0) = x \cdot 0 + x \cdot 0$. It follows that

$$0 = x \cdot 0 + y = (x \cdot 0 + x \cdot 0) + y.$$

A-4 Appendix 1 / Axioms for the Real Numbers and the Positive Integers

Next, note that by the associative law for addition and because $x \cdot 0 + y = 0$, it follows that

$$(x \cdot 0 + x \cdot 0) + y = x \cdot 0 + (x \cdot 0 + y) = x \cdot 0 + 0.$$

Finally, by the additive identity law, we know that $x \cdot 0 + 0 = x \cdot 0$. Consequently, $x \cdot 0 = 0$. \triangleleft

THEOREM 6

For all real numbers x and y , if $x \cdot y = 0$, then $x = 0$ or $y = 0$.

Proof: Suppose that x and y are real numbers and $x \cdot y = 0$. If $x \neq 0$, then, by the multiplicative inverse law, x has a multiplicative inverse $1/x$, such that $x \cdot (1/x) = (1/x) \cdot x = 1$. Because $x \cdot y = 0$, we have $(1/x) \cdot (x \cdot y) = (1/x) \cdot 0 = 0$ by Theorem 5. Using the associate law for multiplication, we have $((1/x) \cdot x) \cdot y = 0$. This means that $1 \cdot y = 0$. By the multiplicative identity rule, we see that $1 \cdot y = y$, so $y = 0$. Consequently, either $x = 0$ or $y = 0$. \triangleleft

THEOREM 7

The multiplicative identity element 1 in the set of real numbers is greater than the additive identity element 0.

Proof: By the trichotomy law, either $0 = 1$, $0 > 1$, or $1 > 0$. We know by the identity elements axiom that $0 \neq 1$.

So, assume that $0 > 1$. We will show that this assumption leads to a contradiction. By the additive inverse law, 1 has an additive inverse -1 with $1 + (-1) = 0$. The additive compatibility law tells us that $0 + (-1) > 1 + (-1) = 0$; the additive identity law tells us that $0 + (-1) = -1$. Consequently, $-1 > 0$, and by the multiplicative compatibility law, $(-1) \cdot (-1) > (-1) \cdot 0$. By Theorem 5 the right-hand side of last inequality is 0. By the distributive law, $(-1) \cdot (-1) + (-1) \cdot 1 = (-1) \cdot (-1 + 1) = (-1) \cdot 0 = 0$. Hence, the left-hand side of this last inequality, $(-1) \cdot (-1)$, is the unique additive inverse of -1 , so this side of the inequality equals 1. Consequently this last inequality becomes $1 > 0$, contradicting the trichotomy law because we had assumed that $0 > 1$.

Because we know that $0 \neq 1$ and that it is impossible for $0 > 1$, by the trichotomy law, we conclude that $1 > 0$. \triangleleft



ARCHIMEDES (287 B.C.E.–212 B.C.E.) Archimedes was one of the greatest scientists and mathematicians of ancient times. He was born in Syracuse, a Greek city-state in Sicily. His father, Phidias, was an astronomer. Archimedes was educated in Alexandria, Egypt. After completing his studies, he returned to Syracuse, where he spent the rest of his life. Little is known about his personal life; we do not know whether he was ever married or had children. Archimedes was killed in 212 B.C.E. by a Roman soldier when the Romans overran Syracuse.

Archimedes made many important discoveries in geometry. His method for computing the area under a curve was described two thousand years before his ideas were re-invented as part of integral calculus. Archimedes also developed a method for expressing large integers inexpressible by the usual Greek method. He discovered a method for computing the volume of a sphere, as well as of other solids, and he calculated an approximation of π . Archimedes was also an accomplished engineer and inventor; his machine for pumping water, now called *Archimedes' screw*, is still in use today. Perhaps his best known discovery is the *principle of buoyancy*, which tells us that an object submerged in liquid becomes lighter by an amount equal to the weight it displaces. Some histories tell us that Archimedes was an early streaker, running naked through the streets of Syracuse shouting “Eureka” (which means “I have found it”) when he made this discovery. He is also known for his clever use of machines that held off Roman forces sieging Syracuse for several years during the Second Punic War.

The next theorem tells us that for every real number there is an integer (where by an *integer*, we mean 0, the sum of any number of 1s, and the additive inverses of these sums) greater than this real number. This result is attributed to the Greek mathematician Archimedes. The result can be found in Book V of Euclid's *Elements*.

THEOREM 8

ARCHIMEDEAN PROPERTY For every real number x there exists an integer n such that $n > x$.

Proof: Suppose that x is a real number such that $n \leq x$ for every integer n . Then x is an upper bound of the set of integers. By the completeness property it follows that the set of integers has a least upper bound M . Because $M - 1 < M$ and M is a least upper bound of the set of integers, $M - 1$ is not an upper bound of the set of integers. This means that there is an integer n with $n > M - 1$. This implies that $n + 1 > M$, contradicting the fact that M is an upper bound of the set of integers. \triangleleft

Axioms for the Set of Positive Integers

The axioms we now list specify the set of positive integers as the subset of the set of integers satisfying four key properties. We assume the truth of these axioms in this textbook.

- **Axiom 1** The number 1 is a positive integer.
- **Axiom 2** If n is a positive integer, then $n + 1$, the *successor* of n , is also a positive integer.
- **Axiom 3** Every positive integer other than 1 is the successor of a positive integer.
- **Axiom 4 The Well-Ordering Property** Every nonempty subset of the set of positive integers has a least element.

In Sections 5.1 and 5.2 it is shown that the well-ordering principle is equivalent to the principle of mathematical induction.

- **Mathematical induction axiom** If S is a set of positive integers such that $1 \in S$ and for all positive integers n if $n \in S$, then $n + 1 \in S$, then S is the set of positive integers.

Most mathematicians take the real number system as already existing, with the real numbers satisfying the axioms we have listed in this appendix. However, mathematicians in the nineteenth century developed techniques to construct the set of real numbers, starting with more basic sets of numbers. (The process of constructing the real numbers is sometimes studied in advanced undergraduate mathematics classes. A treatment of this can be found in [Mo91], for instance.) The first step in the process is the construction of the set of positive integers using axioms 1–3 and either the well-ordering property or the mathematical induction axiom. Then, the operations of addition and multiplication of positive integers are defined. Once this has been done, the set of integers can be constructed using equivalence classes of pairs of positive integers where $(a, b) \sim (c, d)$ if and only if $a + d = b + c$; addition and multiplication of integers can be defined using these pairs (see Exercise 21). (Equivalence relations and equivalence classes are discussed in Chapter 9.) Next, the set of rational numbers can be constructed using the equivalence classes of pairs of integers where the second integer in the pair is not zero, where $(a, b) \approx (c, d)$ if and only if $a \cdot d = b \cdot c$; addition and multiplication of rational numbers can be defined in terms of these pairs (see Exercise 22). Using infinite sequences, the set of real numbers can then be constructed from the set of rational numbers. The interested reader will find it worthwhile to read through the many details of the steps of this construction.

Exercises

Use only the axioms and theorems in this appendix in the proofs in your answers to these exercises.

1. Prove Theorem 3, which states that the multiplicative identity element of the real numbers is unique.
 2. Prove Theorem 4, which states that for every nonzero real number x , the multiplicative inverse of x is unique.
 3. Prove that for all real numbers x and y , $(-x) \cdot y = x \cdot (-y) = -(x \cdot y)$.
 4. Prove that for all real numbers x and y , $-(x + y) = (-x) + (-y)$.
 5. Prove that for all real numbers x and y , $(-x) \cdot (-y) = x \cdot y$.
 6. Prove that for all real numbers x , y , and z , if $x + z = y + z$, then $x = y$.
 7. Prove that for every real number x , $-(-x) = x$.
- Define the **difference** $x - y$ of real numbers x and y by $x - y = x + (-y)$, where $-y$ is the additive inverse of y , and the **quotient** x/y , where $y \neq 0$, by $x/y = x \cdot (1/y)$, where $1/y$ is the multiplicative inverse of y .
8. Prove that for all real numbers x and y , $x = y$ if and only if $x - y = 0$.
 9. Prove that for all real numbers x and y , $-x - y = -(x + y)$.
 10. Prove that for all nonzero real numbers x and y , $1/(x/y) = y/x$, where $1/(x/y)$ is the multiplicative inverse of x/y .
 11. Prove that for all real numbers w , x , y , and z , if $x \neq 0$ and $z \neq 0$, then $(w/x) + (y/z) = (w \cdot z + x \cdot y)/(x \cdot z)$.
 12. Prove that for every positive real number x , $1/x$ is also a positive real number.
 13. Prove that for all positive real numbers x and y , $x \cdot y$ is also a positive real number.

14. Prove that for all real numbers x and y , if $x > 0$ and $y < 0$, then $x \cdot y < 0$.
15. Prove that for all real numbers x , y , and z , if $x > y$ and $z < 0$, then $x \cdot z < y \cdot z$.
16. Prove that for every real number x , $x \neq 0$ if and only if $x^2 > 0$.
17. Prove that for all real numbers w , x , y , and z , if $w < x$ and $y < z$, then $w + y < x + z$.
18. Prove that for all positive real numbers x and y , if $x < y$, then $1/x > 1/y$.
19. Prove that for every positive real number x , there exists a positive integer n such that $n \cdot x > 1$.
- *20. Prove that between every two distinct real numbers there is a rational number (that is, a number of the form x/y , where x and y are integers with $y \neq 0$).

Exercises 21 and 22 involve the notion of an equivalence relation, discussed in Chapter 9 of the text.

- *21. Define a relation \sim on the set of ordered pairs of positive integers by $(w, x) \sim (y, z)$ if and only if $w + z = x + y$. Show that the operations $[(w, x)]_\sim + [(y, z)]_\sim = [(w + y, x + z)]_\sim$ and $[(w, x)]_\sim \cdot [(y, z)]_\sim = [(w \cdot y + x \cdot z, x \cdot y + w \cdot z)]_\sim$ are well-defined, that is, they do not depend on the representative of the equivalence classes chosen for the computation.
- *22. Define a relation \approx on ordered pairs of integers with second entry nonzero by $(w, x) \approx (y, z)$ if and only if $w \cdot z = x \cdot y$. Show that the operations $[(w, x)]_\approx + [(y, z)]_\approx = [(w \cdot z + x \cdot y, x \cdot z)]_\approx$ and $[(w, x)]_\approx \cdot [(y, z)]_\approx = [(w \cdot y, x \cdot z)]_\approx$ are well-defined, that is, they do not depend on the representative of the equivalence classes chosen for the computation.

2

Exponential and Logarithmic Functions

In this appendix we review some of the basic properties of exponential functions and logarithms. These properties are used throughout the text. Students requiring further review of this material should consult precalculus or calculus books, such as those mentioned in the Suggested Readings.

Exponential Functions

Let n be a positive integer, and let b be a fixed positive real number. The function $f_b(n) = b^n$ is defined by

$$f_b(n) = b^n = b \cdot b \cdot b \cdot \dots \cdot b,$$

where there are n factors of b multiplied together on the right-hand side of the equation.

We can define the function $f_b(x) = b^x$ for all real numbers x using techniques from calculus. The function $f_b(x) = b^x$ is called the **exponential function to the base b** . We will not discuss how to find the values of exponential functions to the base b when x is not an integer.

Two of the important properties satisfied by exponential functions are given in Theorem 1. Proofs of these and other related properties can be found in calculus texts.

THEOREM 1

Let b be a positive real number and x and y real numbers. Then

1. $b^{x+y} = b^x b^y$, and
2. $(b^x)^y = b^{xy}$.

We display the graphs of some exponential functions in Figure 1.

Logarithmic Functions

Suppose that b is a real number with $b > 1$. Then the exponential function b^x is strictly increasing (a fact shown in calculus). It is a one-to-one correspondence from the set of real numbers to the set of nonnegative real numbers. Hence, this function has an inverse $\log_b x$, called the **logarithmic function to the base b** . In other words, if b is a real number greater than 1 and x is a positive real number, then

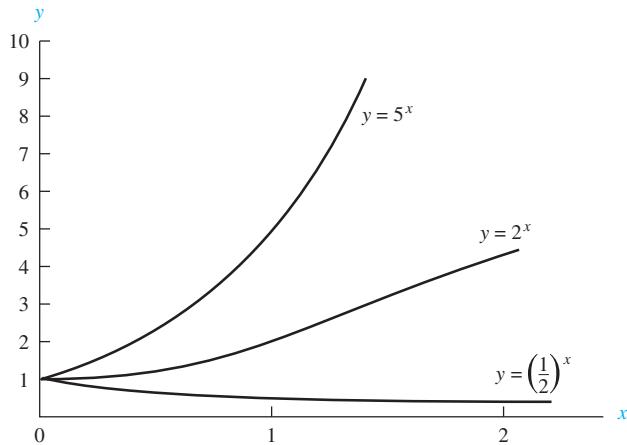
$$b^{\log_b x} = x.$$

The value of this function at x is called the **logarithm of x to the base b** .

From the definition, it follows that

$$\log_b b^x = x.$$

We give several important properties of logarithms in Theorem 2.

**FIGURE 1** Graphs of the Exponential Functions to the Bases $\frac{1}{2}$, 2, and 5.**THEOREM 2**

Let b be a real number greater than 1. Then

1. $\log_b(xy) = \log_b x + \log_b y$ whenever x and y are positive real numbers, and
2. $\log_b(x^y) = y \log_b x$ whenever x is a positive real number and y is a real number.

Proof: Because $\log_b(xy)$ is the unique real number with $b^{\log_b(xy)} = xy$, to prove part 1 it suffices to show that $b^{\log_b x + \log_b y} = xy$. By part 1 of Theorem 1, we have

$$\begin{aligned} b^{\log_b x + \log_b y} &= b^{\log_b x} b^{\log_b y} \\ &= xy. \end{aligned}$$

To prove part 2, it suffices to show that $b^{y \log_b x} = x^y$. By part 2 of Theorem 1, we have

$$\begin{aligned} b^{y \log_b x} &= (b^{\log_b x})^y \\ &= x^y. \end{aligned}$$
△

The following theorem relates logarithms to two different bases.

THEOREM 3

Let a and b be real numbers greater than 1, and let x be a positive real number. Then

$$\log_a x = \log_b x / \log_b a.$$

Proof: To prove this result, it suffices to show that

$$b^{\log_a x \cdot \log_b a} = x.$$

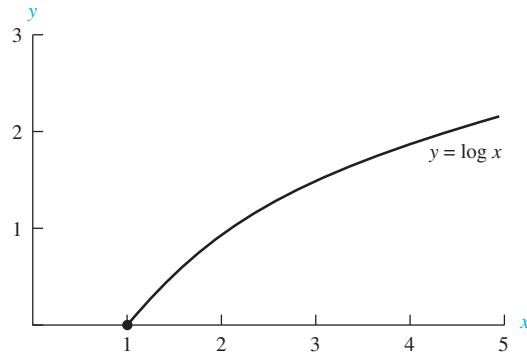


FIGURE 2 The Graph of $f(x) = \log x$.

By part 2 of Theorem 1, we have

$$\begin{aligned} b^{\log_a x \cdot \log_b a} &= (b^{\log_b a})^{\log_a x} \\ &= a^{\log_a x} \\ &= x. \end{aligned}$$

This completes the proof. \triangleleft

Because the base used most often for logarithms in this text is $b = 2$, the notation $\log x$ is used throughout the test to denote $\log_2 x$.

The graph of the function $f(x) = \log x$ is displayed in Figure 2. From Theorem 3, when a base b other than 2 is used, a function that is a constant multiple of the function $\log x$, namely, $(1/\log b) \log x$, is obtained.

Exercises

1. Express each of the following quantities as powers of 2.
 a) $2 \cdot 2^2$ b) $(2^2)^3$ c) $2^{(2^2)}$
2. Find each of the following quantities.
 a) $\log_2 1024$ b) $\log_2 1/4$ c) $\log_4 8$
3. Suppose that $\log_4 x = y$ where x is a positive real number. Find each of the following quantities.
 a) $\log_2 x$ b) $\log_8 x$ c) $\log_{16} x$
4. Let a , b , and c be positive real numbers. Show that $a^{\log_b c} = c^{\log_b a}$.
5. Draw the graph of $f(x) = b^x$ for all real numbers x if b is
 a) 3. b) $1/3$. c) 1.
6. Draw the graph of $f(x) = \log_b x$ for positive real numbers x if b is
 a) 4. b) 100. c) 1000.

3

Pseudocode



The algorithms in this text are described both in English and in **pseudocode**. Pseudocode is an intermediate step between an English language description of the steps of a procedure and a specification of this procedure using an actual programming language. The advantages of using pseudocode include the simplicity with which it can be written and understood and the ease of producing actual computer code (in a variety of programming languages) from the pseudocode. We will describe the particular types of **statements**, or high-level instructions, of the pseudocode that we will use. Each of these statements in pseudocode can be translated into one or more statements in a particular programming language, which in turn can be translated into one or more (possibly many) low-level instructions for a computer.

This appendix describes the format and syntax of the pseudocode used in the text. This pseudocode is designed so that its basic structure resembles that of commonly used programming languages, such as C++ and Java, which are currently the most commonly taught programming languages. However, the pseudocode we use will be a lot looser than a formal programming language because a lot of English language descriptions of steps will be allowed.

This appendix is not meant for formal study. Rather, it should serve as a reference guide for students when they study the descriptions of algorithms given in the text and when they write pseudocode solutions to exercises.

Procedure Statements

The pseudocode for an algorithm begins with a **procedure** statement that gives the name of an algorithm, lists the input variables, and describes what kind of variable each input is. For instance, the statement

```
procedure maximum(L: list of integers)
```

is the first statement in the pseudocode description of the algorithm, which we have named *maximum*, that finds the maximum of a list *L* of integers.

Assignments and Other Types of Statements

An assignment statement is used to assign values to variables. In an assignment statement the left-hand side is the name of the variable and the right-hand side is an expression that involves constants, variables that have been assigned values, or functions defined by procedures. The right-hand side may contain any of the usual arithmetic operations. However, in the pseudocode in this book it may include any well-defined operation, even if this operation can be carried out only by using a large number of statements in an actual programming language.

The symbol `:=` is used for assignments. Thus, an assignment statement has the form

```
variable := expression
```

For example, the statement

```
max := a
```

assigns the value of a to the variable max . A statement such as

```
x := largest integer in the list L
```

can also be used. This sets x equal to the largest integer in the list L . To translate this statement into an actual programming language would require more than one statement. Also, the instruction

```
interchange a and b
```

can be used to interchange a and b . We could also express this one statement with several assignment statements (see Exercise 2), but for simplicity, we will often prefer this abbreviated form of pseudocode.

Comments

In the pseudocode in this book, statements enclosed in curly braces are not executed. Such statements serve as comments or reminders that help explain how the procedure works. For instance, the statement

```
{x is the largest element in L}
```

can be used to remind the reader that at that point in the procedure the variable x equals the largest element in the list L .

Conditional Constructions

The simplest form of the conditional construction that we will use is

```
if condition then statement
```

or

```
if condition then  
    block of statements
```

Here, the condition is checked, and if it is true, then the statement or block of statements given is carried out. In particular, the pseudocode

```
if condition then
    statement 1
    statement 2
    statement 3
    .
    .
    .
    statement n
```

tells us that the statements in the block are executed sequentially if the condition is true.

For example, in Algorithm 1 in Section 3.1, which finds the maximum of a set of integers, we use a conditional statement to check whether $\max < a_i$ for each variable; if it is, we assign the value of a_i to \max .

Often, we require the use of a more general type of construction. This is used when we wish to do one thing when the indicated condition is true, but another when it is false. We use the construction

```
if condition then statement 1
else statement 2
```

Note that either one or both of statement 1 and statement 2 can be replaced with a block of statements.

Sometimes, we require the use of an even more general form of a conditional. The general form of the conditional construction that we will use is

```
if condition 1 then statement 1
else if condition 2 then statement 2
else if condition 3 then statement 3
    .
    .
    .
else if condition n then statement n
else statement n + 1
```

When this construction is used, if condition 1 is true, then statement 1 is carried out, and the program exits this construction. In addition, if condition 1 is false, the program checks whether condition 2 is true; if it is, statement 2 is carried out, and so on. Thus, if none of the first $n - 1$ conditions hold, but condition *n* does, statement *n* is carried out. Finally, if none of condition 1, condition 2, condition 3, . . . , condition *n* is true, then statement *n* + 1 is executed. Note that any of the $n + 1$ statements can be replaced by a block of statements.

Loop Constructions

There are two types of loop construction in the pseudocode in this book. The first is the “for” construction, which has the form

```
for variable := initial value to final value
    statement
```

or

```
for variable := initial value to final value
    block of statements
```

where *initial value* and *final value* are integers. Here, at the start of the loop, *variable* is assigned *initial value* if *initial value* is less than or equal to *final value*, and the statements at the end of this construction are carried out with this value of *variable*. Then *variable* is increased by one, and the statement, or the statements in the block, are carried out with this new value of *variable*. This is repeated until *variable* reaches *final value*. After the instructions are carried out with *variable* equal to *final value*, the algorithm proceeds to the next statement. When *initial value* exceeds *final value*, none of the statements in the loop is executed.

We can use the “for” loop construction to find the sum of the positive integers from 1 to *n* with the following pseudocode.

```
sum := 0
for i := 1 to n
    sum := sum + i
```

Also, the more general “for” statement, of the form

```
for all elements with a certain property
```

is used in this text. This means that the statement or block of statements that follow are carried out successively for the elements with the given property.

The second type of loop construction that we will use is the “while” construction. This has the form

```
while condition
    statement
```

or

```
while condition
    block of statements
```

When this construction is used, the condition given is checked, and if it is true, the statements that follow are carried out, which may change the values of the variables that are part of the condition.

If the condition is still true after these instructions have been carried out, the instructions are carried out again. This is repeated until the condition becomes false. As an example, we can find the sum of the integers from 1 to n using the following block of pseudocode including a “while” construction.

```
sum := 0
while n > 0
    sum := sum + n
    n := n - 1
```

Note that any “for” construction can be turned into a “while” construction (see Exercise 3). However, it is often easier to understand the “for” construction. So, when it makes sense, we will use the “for” construction in preference to the corresponding “while” construction.

Loops within Loops

Loops or conditional statements are often used within other loops or conditional statements. In the pseudocode used in this book, we use successive levels of indentation to indicate nested loops, which are loops within loops, and which blocks of commands correspond to which loops.

Using Procedures in Other Procedures

We can use a procedure from within another procedure (or within itself in a recursive program) simply by writing the name of this procedure followed by the inputs to this procedure. For instance,

```
max(L)
```

will carry out the procedure *max* with the input list L . After all the steps of this procedure have been carried out, execution carries on with the next statement in the procedure.

Return Statements

We use a **return** statement to show where a procedure produces output. A return statement of the form

```
return x
```

produces the current value of x as output. The output x can involve the value of one or more functions, including the same function under evaluation, but at a smaller value. For instance, the statement

```
return f(n - 1)
```

is used to call the algorithm with input of $n - 1$. This means that the algorithm is run again with input equal to $n - 1$.

Exercises

1. What is the difference between the following blocks of two assignment statements?

$a := b$
 $b := c$

and

$b := c$
 $a := b$

2. Give a procedure using assignment statements to interchange the values of the variables x and y . What is the minimum number of assignment statements needed to do this?

3. Show how a loop of the form

for $i :=$ *initial value* **to** *final value*
statement

can be written using the “while” construction.

Suggested Reading

Among the resources available for learning more about the topics covered in this book are printed materials and relevant websites. Printed resources are described in this section of suggested readings. Readings are listed by chapter and keyed to particular topics of interest. Some general references also deserve special mention. A book you may find particularly useful is the *Handbook of Discrete and Combinatorial Mathematics* by Rosen [Ro00], a comprehensive reference book. Additional applications of discrete mathematics can be found in Michaels and Rosen [MiRo91], which is also available online on the companion website for this book. Deeper coverage of many topics in computer science, including those discussed in this book, can be found in Gruska [Gr97]. Biographical information about many of the mathematicians and computer scientists mentioned in this book can be found in Gillispie [Gi70] and on the MacTutor website at <http://www-history.mcs.st-and.ac.uk/>.

To find pertinent websites, consult the links found in the Web Resources Guide on the companion website for this book. Its address is www.mhhe.com/rosen.

CHAPTER 1

An entertaining way to study logic is to read Lewis Carroll's book [Ca78]. General references for logic include M. Huth and M. Ryan [HuRy04], Mendelson [Me09], Stoll [St74], and Suppes [Su87]. A comprehensive treatment of logic in discrete mathematics can be found in Gries and Schneider [GrSc93]. System specifications are discussed in Ince [In93]. Smullyan's knights and knaves puzzles were introduced in [Sm78]. He has written many fascinating books on logic puzzles including [Sm92] and [Sm98]. Prolog is discussed in depth in Nilsson and Maluszynski [NiMa95] and in Clocksin and Mellish [CIMe94]. The basics of proofs are covered in Cuppilliari [Cu05], Morash [Mo91], Solow [So09], Velleman [Ve06], and Wolf [Wo98]. The science and art of constructing proofs is discussed in a delightful way in three books by Pólya: [Po62], [Po71], and [Po90]. Problems involving tiling checkerboards using dominoes and polyominoes are discussed in Golomb [Go94] and Martin [Ma91].

CHAPTER 2

Lin and Lin [LiLi81] is an easily read text on sets and their applications. Axiomatic developments of set theory can be found in Hal-mos [Ha60], Monk [Mo69], and Stoll [St74]. Brualdi [Br09], and Reingold, Nievergelt, and Deo [ReNiDe77] contain introductions to multisets. Fuzzy sets and their application to expert systems and artificial intelligence are treated in Negoita [Ne85] and Zimmerman [Zi91]. Calculus books, such as Apostol [Ap67], Spivak [Sp94], and Thomas and Finney [ThFi96], contain discussions of functions. The best printed source of information about integer sequences is Sloan and Plouffe [SIP195]. Books on proofs, such as [Ve06], often cover countability in some depth. Stanat and McAllister [StMc77] has a thorough section on countability. Chapter 17 of Aigner, Ziegler, and Hoffman [AiZiHo09] provides an excellent discussion of cardinality and the continuum hypothesis.

Discussions of the mathematical foundations needed for computer science can be found in Arbib, Kfoury, and Moll [ArKfMo80], Bobrow and Arbib [BoAr74], Beckman [Be80], and Tremblay and Manohar [TrMa75]. Matrices and their operations are covered in all linear algebra books, such as Curtis [Cu84] and Strang [St09].

CHAPTER 3

The articles by Knuth [Kn77] and Wirth [Wi84] are accessible introductions to the subject of algorithms. Among the best introductions to algorithms are Cormen, Leiserson, Rivest, and Stein [CoLeRiSt09] and Kleinberg and Tardos [KiTa05]. Extensive material on big- O estimates of functions can be found in Knuth [Kn97a]. General references for algorithms and their complexity include Aho, Hopcroft, and Ullman [AhHoU174]; Baase and Van Gelder [BaGe99]; Cormen, Leiserson, Rivest, and Stein [CoLeRiSt09]; Gonnet [Go84]; Goodman and Hedetniemi [GoHe77]; Harel [Ha87]; Horowitz and Sahni [HoSa82]; Kreher and Stinson [KrSt98]; the famous series of books by Knuth on the art of computer programming [Kn97a], [Kn97b], and [Kn98]; Kronsjö [Kr87]; Levitin [Le06]; Manber [Ma89]; Pohl and Shaw [PoSh81]; Purdom and Brown [PuBr85]; Rawlins [Ra92]; Sedgewick [Se03]; Wilf [Wi02]; and Wirth [Wi76]. Sorting and searching algorithms and their complexity are studied in detail in Knuth [Kn98].

CHAPTER 4

References for number theory include Hardy and Wright [HaWr-WiHe08]; LeVeque [Le77]; Rosen [Ro10]; and Stark [St78]. More about the history of number theory can be found in Ore [Or88]. Algorithms for computer arithmetic are discussed in Knuth [Kn97b] and Pohl and Shaw [PoSh81]. More information about algorithms for finding primes and for factorization can be found in Crandall

B-2 Suggested Reading

and Pomerance [CrPo10]. Applications of number theory to cryptography are covered in Denning [De82]; Menezes, van Oorschot, and Vanstone [MeOoVa97]; Rosen [Ro10]; Seberry and Pieprzyk [SePi89]; Sinkov [Si66]; and Stinson [St05]. The RSA public-key system was described by Rivest, Shamir, and Adleman in [RiShAd78]; its discovery by Cocks is described in [Si99], which also provides an appealing account of the history of cryptography.

CHAPTER 5

An accessible introduction to mathematical induction can be found [Gu10] and in Sominskii [So61]. Books that contain thorough treatments of mathematical induction and recursive definitions include Liu [Li85]; Sahni [Sa85]; Stanat and McAllister [StMc77]; and Tremblay and Manohar [TrMa75]. Computational geometry is covered in [DeOr11] and [Or00]. The Ackermann function, introduced in 1928 by W. Ackermann, arises in the theory of recursive function (see Beckman [Be80] and McNaughton [Mc82], for instance) and in the analysis of the complexity of certain set theoretic algorithms (see Tarjan [Ta83]). Recursion is studied in Roberts [Ro86]; Rohl [Ro84]; and Wand [Wa80]. Discussions of program correctness and the logical machinery used to prove that programs are correct can be found in Alagic and Arbib [AlAr78]; Anderson [An79]; Backhouse [Ba86]; Sahni [Sa85]; and Stanat and McAllister [StMc77].

CHAPTER 6

General references for counting techniques and their applications include Allenby and Slomson [AlSl10]; Anderson [An89]; Berman and Fryer [BeFr72]; Bogart [Bo00]; Bona [Bo07]; Bose and Manvel [BoMa86]; Brualdi [Br09]; Cohen [Co78]; Grimaldi [Gr03]; Gross [Gr07]; Liu [Li68]; Pólya, Tarjan, and Woods [PoTaWo83]; Riordan [Ri58]; Roberts and Tesman [RoTe03]; Tucker [Tu06]; and Williamson [Wi85]. Vilenkin [Vi71] contains a selection of combinatorial problems and their solutions. A selection of more difficult combinatorial problems can be found in Lovász [Lo79]. Information about Internet protocol addresses and datagrams can be found in Comer [Co05]. Applications of the pigeonhole principle can be found in Brualdi [Br09]; Liu [Li85]; and Roberts and Tesman [RoTe03]. A wide selection of combinatorial identities can be found in Riordan [Ri68] and in Benjamin and Quinn [BeQu03]. Combinatorial algorithms, including algorithms for generating permutations and combinations, are described by Even [Ev73]; Lehmer [Le64]; and Reingold, Nievergelt, and Deo [ReNiDe77].

CHAPTER 7

Useful references for discrete probability theory include Feller [Fe68], Nabin [Na00], and Ross [Ro09a]. Ross [Ro02], which focuses on the application of probability theory to computer science, provides examples of average case complexity analysis and covers the probabilistic method. Aho and Ullman [AhUi95] includes a discussion of various aspects of probability theory important in

computer science, including programming applications of probability. The probabilistic method is discussed in a chapter in Aigner, Ziegler, and Hoffman [AiZiHo09], a monograph devoted to clever, insightful, and brilliant proofs, that is, proofs that Paul Erdős described as coming from “*The Book*. ” Extensive coverage of the probabilistic method can be found in Alon and Spencer [AlSp00]. Bayes’ theorem is covered in [PaPi01]. Additional material on spam filters can be found in [Zd05].

CHAPTER 8

Many different models using recurrence relations can be found in Roberts and Tesman [RoTe03] and Tucker [Tu06]. Exhaustive treatments of linear homogeneous recurrence relations with constant coefficients, and related inhomogeneous recurrence relations, can be found in Brualdi [Br09], Liu [Li68], and Mattson [Ma93]. Divide-and-conquer algorithms and their complexity are covered in Roberts and Tesman [RoTe03] and Stanat and McAllister [StMc77]. Descriptions of fast multiplication of integers and matrices can be found in Aho, Hopcroft, and Ullman [AhHoUi74] and Knuth [Kn97b]. An excellent introduction to generating functions can be found in Pólya, Tarjan, and Woods [PoTaWo83]. Generating functions are studied in detail in Brualdi [Br09]; Cohen [Co78]; Graham, Knuth, and Patashnik [GrKnPa94]; Grimaldi [Gr03]; and Roberts and Tesman [RoTe03]. Additional applications of the principle of inclusion–exclusion can be found in Liu [Li85] and [Li68]; Roberts and Tesman [RoTe03]; and Ryser [Ry63].

CHAPTER 9

General references for relations, including treatments of equivalence relations and partial orders, include Bobrow and Arbib [BoAr74]; Grimaldi [Gr03]; Sahni [Sa85]; and Tremblay and Manohar [TrMa75]. Discussions of relational models for databases are given in Date [Da82] and Aho and Ullman [AhUi95]. The original papers by Roy and Warshall for finding transitive closures can be found in [Ro59] and [Wa62], respectively. Directed graphs are studied in Chartrand, Lesniak, and Zhang [ChLeZh05]; Gross and Yellen [GrYe05]; Robinson and Foulds [RoFo80]; Roberts and Tesman [RoTe03]; and Tucker [Tu06]. The application of lattices to information flow is treated in Denning [De82].

CHAPTER 10

General references for graph theory include Agnarsson and Greenlaw [AgGr06]; Aldous, Wilson, and Best [AlWiBe00]; Behzad and Chartrand [BeCh71]; Chartrand, Lesniak, and Zhang [ChLeZh05]; Chartrand and Zhang [ChZh04]; Bondy and Murty [BoMu10]; Chartrand and Oellermann [ChOe93]; Graver and Watkins [GrWa77]; Roberts and Tesman [RoTe03]; Tucker [Tu06]; West [We00]; Wilson [Wi85]; and Wilson and Watkins [WiWa90]. A wide variety of applications of graph theory can be found in Chartrand [Ch77]; Deo [De74]; Foulds [Fo92]; Roberts

and Tesman [RoTe03]; Roberts [Ro76]; Wilson and Beineke [WiBe79]; and McHugh [Mc90]. In depth treatments of the use of graph theory to study social networks, and other types of networks, appears in Easley and Kleinberg [EaKl10] and Newman [Ne10]. Applications involving large graphs, including the Web graph, are discussed in Hayes [Ha00a] and [Ha00b].

A comprehensive description of algorithms in graph theory can be found in Gibbons [Gi85] and in Kocay and Kreher [KoKr04]. Other references for algorithms in graph theory include Buckley and Harary [BuHa90]; Chartrand and Oellermann [ChOe93]; Chachra, Ghare, and Moore [ChGhMo79]; Even [Ev73] and [Ev79]; Hu [Hu82]; and Reingold, Nievergelt, and Deo [ReNiDe77]. A translation of Euler's original paper on the Königsberg bridge problem can be found in Euler [Eu53]. Dijkstra's algorithm is studied in Gibbons [Gi85]; Liu [Li85]; and Reingold, Nievergelt, and Deo [ReNiDe77]. Dijkstra's original paper can be found in [Di59]. A proof of Kuratowski's theorem can be found in Harary [Ha69] and Liu [Li68]. Crossing numbers and thicknesses of graphs are studied in Chartrand, Lesniak, and Zhang [ChLeZh05]. References for graph coloring and the four-color theorem are included in Barnette [Ba83] and Saaty and Kainen [SaKa86]. The original conquest of the four-color theorem is reported in Appel and Haken [ApHa76]. Applications of graph coloring are described by Roberts and Tesman [RoTe03]. The history of graph theory is covered in Biggs, Lloyd, and Wilson [BiLiWi86]. Interconnection networks for parallel processing are discussed in Akl [Ak89] and Siegel and Hsu [SiHs88].

CHAPTER 11

Trees are studied in Deo [De74], Grimaldi [Gr03], Knuth [Kn97a], Roberts and Tesman [RoTe03], and Tucker [Tu06]. The use of trees in computer science is described by Gotlieb and Gotlieb [GoGo78], Horowitz and Sahni [HoSa82], and Knuth [Kn97a, 98]. Roberts and Tesman [RoTe03] covers applications of trees to many different areas. Prefix codes and Huffman coding are covered in Hamming [Ha80]. Backtracking is an old technique; its use to solve maze puzzles can be found in the 1891 book by Lucas [Lu91]. An extensive discussion of how to solve problems using backtracking can be found in Reingold, Nievergelt, and Deo [ReNiDe77]. Gibbons [Gi85] and Reingold, Nievergelt, and Deo [ReNiDe77] contain discussions of algorithms for constructing spanning trees and minimal spanning trees. The background and history of algorithms for finding minimal spanning trees is covered in Graham and Hell [GrHe85]. Prim and Kruskal described their algorithms for finding minimal spanning trees in [Pr57] and [Kr56], respectively. Sollin's algorithm is an example of an algorithm well suited for parallel processing; although Sollin never published a description of it, his algorithm has been described by Even [Ev73] and Goodman and Hedetniemi [GoHe77].

CHAPTER 12

Boolean algebra is studied in Hohn [Ho66], Kohavi [Ko86], and Tremblay and Manohar [TrMa75]. Applications of Boolean al-

gebra to logic circuits and switching circuits are described by Hayes [Ha93], Hohn [Ho66], Katz and Borriello [KaBo04], and Kohavi [Ko86]. The original papers dealing with the minimization of sum-of-products expansions using maps are Karnaugh [Ka53] and Veitch [Ve52]. The Quine-McCluskey method was introduced in McCluskey [Mc56] and Quine [Qu52] and [Qu55]. Threshold functions are covered in Kohavi [Ko86].

CHAPTER 13

General references for formal grammars, automata theory, and the theory of computation include Davis, Sigal, and Weyuker [DaSiWe94]; Denning, Dennis, and Qualitz [DeDeQu81]; Hopcroft, Motwani, and Ullman [HoMoUl06]; Hopkin and Moss [HoMo76]; Lewis and Papadimitriou [LePa97]; McNaughton [Mc82]; and Sipser [Si06]. Mealy machines and Moore machines were originally introduced in Mealy [Me55] and Moore [Mo56]. The original proof of Kleene's theorem can be found in [Kl56]. Powerful models of computation, including pushdown automata and Turing machines, are discussed in Brookshear [Br89], Henkle [He77], Hopcroft and Ullman [HoUl79], Hopkin and Moss [HoMo76], Martin [Ma03], Sipser [Si06], and Wood [Wo87]. Barwise and Etchemendy [BaEt93] is an excellent introduction to Turing machines. Interesting articles about the history and application of Turing machines and related machines can be found in Herken [He88]. Busy beaver machines were first introduced by Rado in [Ra62], and information about them can be found in Dewdney [De84] and [De93], the article by Brady in Herken [He88], and in Wood [Wo87].

APPENDIXES

A discussion of axioms for the real number and for the integers can be found in Morash [Mo91]. Detailed treatments of exponential and logarithmic functions can be found in calculus books such as Apostol [Ap67], Spivak [Sp94], and Thomas and Finney [ThFi96]. Pohl and Shaw [PoSh81] use a form of pseudocode that has the same features as those described in Appendix 3. Most textbooks on algorithms, such as Cormen, Leiserson, Rivest, and Stein [CoLeRiSt09] and Kleinberg and Tardos [KlTa05], use versions of pseudocode similar to the pseudocode in this text.

REFERENCES

- [AgGr06] G. Agnarsson and R. Greenlaw, *Graph Theory: Modeling, Applications, and Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 2006.
- [AhHoUl74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
- [AhUl95] Alfred V. Aho and Jeffrey D. Ullman, *Foundations of Computer Science, C Edition*, Computer Science Press, New York, 1995.

B-4 Suggested Reading

- [AiZiHo09] Martin Aigner, Günter M. Ziegler, and Karl H. Hofmann, *Proofs from THE BOOK*, 4th ed., Springer, Berlin, 2009.
- [Ak89] S. G. Akl, *The Design and Analysis of Parallel Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1989.
- [AlAr78] S. Alagic and M. A. Arbib, *The Design of Well-Structured and Correct Programs*, Springer-Verlag, New York, 1978.
- [AlWiBe00] J. M. Aldous, R. J. Wilson, and S. Best, *Graphs and Applications: An Introductory Approach*, Springer, New York, 2000.
- [AISI10] R.B.J.T. Allenby and A. Slomson, *How to Count: An Introduction to Combinatorics*, 2d. ed., Chapman and Hall/CRC, Boca Raton, Florida, 2010.
- [AlSp00] Noga Alon and Joel H. Spencer, *The Probabilistic Method*, 2d ed., Wiley, New York, 2000.
- [An89] I. Anderson, *A First Course in Combinatorial Mathematics*, 2d. ed., Oxford University Press, New York, 1989.
- [An79] R. B. Anderson, *Proving Programs Correct*, Wiley, New York, 1979.
- [Ap67] T. M. Apostol, *Calculus*, Vol. I, 2d ed., Wiley, New York, 1967.
- [ApHa76] K. Appel and W. Haken, “Every Planar Map Is 4-colorable,” *Bulletin of the AMS*, 82 (1976), 711–712.
- [ArKfMo80] M. A. Arbib, A. J. Kfoury, and R. N. Moll, *A Basis for Theoretical Computer Science*, Springer-Verlag, New York, 1980.
- [AvCh90] B. Averbach and O. Chein, *Problem Solving Through Recreational Mathematics*, W.H. Freeman, San Francisco, 1980.
- [BaGe99] S. Baase and A. Van Gelder, *Computer Algorithms: Introduction to Design and Analysis*, 3d ed., Addison-Wesley, Reading, MA, 1999.
- [Ba86] R. C. Backhouse, *Program Construction and Verification*, Prentice-Hall, Englewood Cliffs, NJ, 1986.
- [Ba83] D. Barnette, *Map Coloring, Polyhedra, and the Four-Color Problem*, Mathematical Association of America, Washington, DC, 1983.
- [BaEt93] Jon Barwise and John Etchemendy, *Turing’s World 3.0 for the Macintosh*, CSLI Publications, Stanford, CA, 1993.
- [Be80] F. S. Beckman, *Mathematical Foundations of Programming*, Addison-Wesley, Reading, MA, 1980.
- [BeCh71] M. Behzad and G. Chartrand, *Introduction to the Theory of Graphs*, Allyn & Bacon, Boston, 1971.
- [BeQu03] A. Benjamin and J. J. Quine, *Proofs that Really Count*, Mathematical Association of America, Washington, DC, 2003.
- [Be86] J. Bentley, *Programming Pearls*, Addison-Wesley, Reading, MA, 1986.
- [BeFr72] G. Berman and K. D. Fryer, *Introduction to Combinatorics*, Academic Press, New York, 1972.
- [BiLiWi99] N. L. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory 1736–1936*, Oxford University Press, Oxford, England, 1999.
- [BoAr74] L. S. Bobrow and M. A. Arbib, *Discrete Mathematics*, Saunders, Philadelphia, 1974.
- [Bo00] K. P. Bogart, *Introductory Combinatorics*, 3d ed. Academic Press, San Diego, 2000.
- [Bo07] M. Bona, *Enumerative Combinatorics*, McGraw-Hill, New York, 2007.
- [BoMu10] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, Springer, New York, 2010.
- [Bo04] P. Bork, L. J. Jensen, C. von Mering, A. K. Ramani, I. Lee, and E. M. Marcotte, “Protein interaction networks from yeast to human,” *Current Opinion in Structural Biology*, 14 (2004), 292–299.
- [BoMa86] R. C. Bose and B. Manvel, *Introduction to Combinatorial Theory*, Wiley, New York, 1986.
- [Bo00] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Statac, A. Tomkins, and Janet Wiener, “Graph structure in the Web,” *Computer Networks*, 33 (2000), 309–320.
- [Br89] J. G. Brookshear, *Theory of Computation*, Benjamin Cummings, Redwood City, CA, 1989.
- [Br09] R. A. Brualdi, *Introductory Combinatorics*, 5th ed., Prentice-Hall, Englewood Cliffs, NJ, 2009.
- [BuHa90] F. Buckley and F. Harary, *Distance in Graphs*, Addison-Wesley, Redwood City, CA, 1990.
- [Ca79] L. Carmony, “Odd Pie Fights,” *Mathematics Teacher* 72 (January, 1979), 61–64.
- [Ca78] L. Carroll, *Symbolic Logic*, Crown, New York, 1978.
- [ChGhMo79] V. Chachra, P. M. Ghare, and J. M. Moore, *Applications of Graph Theory Algorithms*, North-Holland, New York, 1979.
- [Ch77] G. Chartrand, *Graphs as Mathematical Models*, Prindle, Weber & Schmidt, Boston, 1977.
- [ChLeZh10] G. Chartrand, L. Lesniak, and P. Zhang, *Graphs and Digraphs*, 5th ed., Chapman and Hall/CRC, Boca Raton, 2010.
- [ChOe93] G. Chartrand and O. R. Oellermann, *Applied Algorithmic Graph Theory*, McGraw-Hill, New York, 1993.
- [ChZh04] G. Chartrand and P. Zhang, *Introduction to Graph Theory*, McGraw-Hill, New York, 2004.
- [ClMe94] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, 4th ed., Springer-Verlag, New York, 1994.
- [Co78] D. I. A. Cohen, *Basic Techniques of Combinatorial Theory*, Wiley, New York, 1978.
- [Co05] D. Comer, *Internetworking with TCP/IP, Principles, Protocols, and Architecture*, Vol. 1, 5th ed., Prentice-Hall, Englewood Cliffs, NJ, 2005.
- [CoLeRiSt09] T. H. Cormen, C. E. Leierson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed., MIT Press, Cambridge, MA, 2009.
- [CrPo10] Richard Crandall and Carl Pomerance, 2d ed., *Prime Numbers: A Computational Perspective*, Springer-Verlag, New York, 2010.

- [Cu05] Antonella Cupillari, *The Nuts and Bolts of Proofs*, 3d ed., Academic Press, San Diego, 2005.
- [Cu84] C. W. Curtis, *Linear Algebra*, Springer-Verlag, New York, 1984.
- [Da82] C. J. Date, *An Introduction to Database Systems*, 3d ed., Addison-Wesley, Reading, MA, 1982.
- [DaSiWe94] M. Davis, R. Sigal, and E. J. Weyuker, *Computability, Complexity, and Languages*, 2d ed., Academic Press, San Diego, 1994.
- [Da10] T. Davis, “The Mathematics of Sudoku,” November, 2010, available at <http://geometer.org/mathcircles/sudoku.pdf>
- [De82] D. E. R. Denning, *Cryptography and Data Security*, Addison-Wesley, Reading, MA, 1982.
- [DeDeQu81] P. J. Denning, J. B. Dennis, and J. E. Qualitz, *Machines, Languages, and Computation*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
- [De74] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
- [DeOr11] S. L. Devadoss and J. O’Rourke, *Discrete and Computational Geometry*, Princeton University Press, Princeton, NJ, 2011.
- [De02] K. Devlin, *The Millennium Problems: The Seven Greatest Unsolved Mathematical Puzzles of Our Time*, Basic Book, New York, 2002.
- [De84] A. K. Dewdney, “Computer Recreations,” *Scientific American*, 251, no. 2 (August 1984), 19–23; 252, no. 3 (March 1985), 14–23; 251, no. 4 (April 1985), 20–30.
- [De93] A. K. Dewdney, *The New Turing Omnibus: Sixty-Six Excursions in Computer Science*, W. H. Freeman, New York, 1993.
- [Di59] E. Dijkstra, “Two Problems in Connexion with Graphs,” *Numerische Mathematik*, 1 (1959), 269–271.
- [EaKl10] D. Easley and J. Kleinberg, *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*, Cambridge University Press, New York, 2010.
- [Eu53] L. Euler, “The Koenigsberg Bridges,” *Scientific American*, 189, no. 1 (July 1953), 66–70.
- [Ev73] S. Even, *Algorithmic Combinatorics*, Macmillan, New York, 1973.
- [Ev79] S. Even, *Graph Algorithms*, Computer Science Press, Rockville, MD, 1979.
- [Fe68] W. Feller, *An Introduction to Probability Theory and Its Applications*, Vol. 1, 3d ed., Wiley, New York, 1968.
- [Fo92] L. R. Foulds, *Graph Theory Applications*, Springer-Verlag, New York, 1992.
- [GaJo79] Michael R. Garey and David S. Johnson, *Computers and Intractability: A Guide to NP-Completeness*, Freeman, New York, 1979.
- [Gi85] A. Gibbons, *Algorithmic Graph Theory*, Cambridge University Press, Cambridge, England, 1985.
- [Gi70] C. C. Gillispie, ed., *Dictionary of Scientific Biography*, Scribner’s, New York, 1970.
- [Go94] S. W. Golomb, *Polyominoes*, Princeton University Press, Princeton, NJ, 1994.
- [Go84] G. H. Gonnet, *Handbook of Algorithms and Data Structures*, Addison-Wesley, London, 1984.
- [GoHe77] S. E. Goodman and S. T. Hedetniemi, *Introduction to the Design and Analysis of Algorithms*, McGraw-Hill, New York, 1977.
- [GoGo78] C. C. Gotlieb and L. R. Gotlieb, *Data Types and Structures*, Prentice-Hall, Englewood Cliffs, NJ, 1978.
- [GrHe85] R. L. Graham and P. Hell, “On the History of the Minimum Spanning Tree Problem,” *Annals of the History of Computing*, 7 (1985), 43–57.
- [GrKnPa94] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, 2d ed., Addison-Wesley, Reading, MA, 1994.
- [GrRoSp90] Ronald L. Graham, Bruce L. Rothschild, and Joel H. Spencer, *Ramsey Theory*, 2d ed., Wiley, New York, 1990.
- [GrWa77] J. E. Graver and M. E. Watkins, *Combinatorics with Emphasis on the Theory of Graphs*, Springer-Verlag, New York, 1977.
- [GrSc93] D. Gries and F. B. Schneider, *A Logical Approach to Discrete Math*, Springer-Verlag, New York, 1993.
- [Gr03] R. P. Grimaldi, *Discrete and Combinatorial Mathematics*, 5th ed., Addison-Wesley, Reading, MA, 2003.
- [Gr07] J. L. Gross, *Combinatorial Methods with Computer Applications*, Chapman and Hall/CRC, Boca Raton, FL, 2007.
- [GrYe05] J. L. Gross and J. Yellen, *Graph Theory and Its Applications*, 2d ed., CRC Press, Boca Raton, FL, 2005.
- [GrYe03] J. L. Gross and J. Yellen, *Handbook of Graph Theory*, CRC Press, Boca Raton, FL, 2003.
- [Gr90] Jerrold W. Grossman, *Discrete Mathematics: An Introduction to Concepts, Methods, and Applications*, Macmillan, New York, 1990.
- [Gr97] J. Gruska, *Foundations of Computing*, International Thomsen Computer Press, London, 1997.
- [Gu10] D. A. Gunderson, *Handbook of Mathematical Induction*, Chapman and Hall/CRC, Boca Raton, Florida, 2010.
- [Ha60] P. R. Halmos, *Naive Set Theory*, D. Van Nostrand, New York, 1960.
- [Ha80] R. W. Hamming, *Coding and Information Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1980.
- [Ha69] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.
- [HaWrWiHe08] G. H. Hardy, E. M. Wright, A. Wiles, and R. Heath-Brown, *An Introduction to the Theory of Numbers*, 6th ed., Oxford University Press, USA, New York, 2008.
- [Ha87] D. Harel, *Algorithmics, The Spirit of Computing*, Addison-Wesley, Reading, MA, 1987.
- [Ha00a] Brian Hayes, “Graph-Theory in Practice, Part I,” *American Scientist*, 88, no. 1 (2000), 9–13.
- [Ha00b] Brian Hayes, “Graph-Theory in Practice, Part II,” *American Scientist*, 88, no. 2 (2000), 104–109.

B-6 Suggested Reading

- [Ha93] John P. Hayes, *Introduction to Digital Logic Design*, Addison-Wesley, Reading, MA, 1993.
- [He77] F. Hennie, *Introduction to Computability*, Addison-Wesley, Reading, MA, 1977.
- [He88] R. Herken, *The Universal Turing Machine, A Half-Century Survey*, Oxford University Press, New York, 1988.
- [Ho76] C. Ho, “Decomposition of a Polygon into Triangles,” *Mathematical Gazette*, 60 (1976), 132–134.
- [Ho99] D. Hofstadter, Gödel, Escher, Bach: *An Internal Golden Braid*, Basic Books, New York, 1999.
- [Ho66] F. E. Hohn, *Applied Boolean Algebra*, 2d ed., Macmillan, New York, 1966.
- [HoMoUl06] J. E. Hopcroft, R Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3d ed., Addison-Wesley, Boston, MA, 2006.
- [HoMo76] D. Hopkin and B. Moss, *Automata*, Elsevier, North-Holland, New York, 1976.
- [HoSa82] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Rockville, MD, 1982.
- [Hu82] T.C. Hu, *Combinatorial Algorithms*, Addison-Wesley, Reading, MA, 1982.
- [Hu07] W. Huber, V. J. Carey, L. Long, S. Falcon, and R. Gentleman, “Graphs in molecular biology,” *BMC Bioinformatics*, 8 (Supplement 6) (2007).
- [HuRy04] M. Huth and M. Ryan, *Logic in Computer Science*, 2d ed., Cambridge University Press, Cambridge, England, 2004.
- [In93] D. C. Ince, *An Introduction to Discrete Mathematics, Formal System Specification, and Z*, 2d ed., Oxford, New York, 1993.
- [KaMo64] D. Kalish and R. Montague, *Logic: Techniques of Formal Reasoning*, Harcourt, Brace, Jovanovich, New York, 1964.
- [Ka53] M. Karnaugh, “The Map Method for Synthesis of Combinatorial Logic Circuits,” *Transactions of the AIEE*, part I, 72 (1953), 593–599.
- [KaBo04] R. H. Katz and G. Borriello, *Contemporary Logic Design*, Prentice-Hall, Englewood Cliffs, NJ, 2004.
- [Ki56] S. C. Kleene, “Representation of Events by Nerve Nets,” in *Automata Studies*, 3–42, Princeton University Press, Princeton, NJ, 1956.
- [KiTa05] J. Kleinberg and E. Tardos, *Algorithm Design*, Addison-Wesley, Boston, 2005.
- [Kn77] D. E. Knuth, “Algorithms,” *Scientific American*, 236, no. 4 (April 1977), 63–80.
- [Kn97a] D. E. Knuth, *The Art of Computer Programming, Vol. I: Fundamental Algorithms*, 3d ed., Addison-Wesley, Reading, MA, 1997.
- [Kn97b] D. E. Knuth, *The Art of Computer Programming, Vol. II: Seminumerical Algorithms*, 3d ed., Addison-Wesley, Reading, MA, 1997.
- [Kn98] D. E. Knuth, *The Art of Computer Programming, Vol. III: Sorting and Searching*, 2d ed., Addison-Wesley, Reading, MA, 1998.
- [KoKr04] W. Kocay and D. L. Kreher, *Graph Algorithms and Optimization*, Chapman and Hall/CRC, Boca Raton, Florida, 2004.
- [Ko86] Z. Kohavi, *Switching and Finite Automata Theory*, 2d ed., McGraw-Hill, New York, 1986.
- [KrSt98] Donald H. Kreher and Douglas R. Stinson, *Combinatorial Algorithms: Generation, Enumeration, and Search*, CRC Press, Boca Raton, FL, 1998.
- [Kr87] L. Kronsjö, *Algorithms: Their Complexity and Efficiency*, 2d ed., Wiley, New York, 1987.
- [Kr56] J. B. Kruskal, “On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem,” *Proceedings of the AMS*, 1 (1956), 48–50.
- [La10] J. C. Lagarias (ed.), *The Ultimate Challenge: The $3x + 1$ Problem*, The American Mathematical Society, Providence, 2010.
- [Le06] A. V. Levitin, *Introduction to the Design and Analysis of Algorithms*, 2d, ed., Addison-Wesley, Boston, MA, 2006.
- [Le64] D. H. Lehmer, “The Machine Tools of Combinatorics,” in E. F. Beckenbach (ed.), *Applied Combinatorial Mathematics*, Wiley, New York, 1964.
- [Le77] W. J. LeVeque, *Fundamentals of Number Theory*, Addison-Wesley, Reading, MA, 1977.
- [LePa97] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, 2d ed., Prentice-Hall, Englewood Cliffs, NJ, 1997.
- [LiLi81] Y. Lin and S. Y. T. Lin, *Set Theory with Applications*, 2d ed., Mariner, Tampa, FL, 1981.
- [Li68] C. L. Liu, *Introduction to Combinatorial Mathematics*, McGraw-Hill, New York, 1968.
- [Li85] C. L. Liu, *Elements of Discrete Mathematics*, 2d ed., McGraw-Hill, New York, 1985.
- [Lo79] L. Lovász, *Combinatorial Problems and Exercises*, North-Holland, Amsterdam, 1979.
- [Lu91] E. Lucas, *Récréations Mathématiques*, Gauthier-Villars, Paris, 1891.
- [Ma89] U. Manber, *Introduction to Algorithms: A Creative Approach*, Addison-Wesley, Reading, MA, 1989.
- [Ma91] G. E. Martin, *Polyominoes: A Guide to Puzzles and Problems in Tiling*, Mathematical Association of America, Washington, DC, 1991.
- [Ma03] J. C. Martin, *Introduction to Languages and the Theory of Computation*, 3d ed., McGraw-Hill, New York, 2003.
- [Ma93] H. F. Mattson, Jr., *Discrete Mathematics with Applications*, Wiley, New York, 1993.
- [Mc56] E. J. McCluskey, Jr., “Minimization of Boolean Functions,” *Bell System Technical Journal*, 35 (1956), 1417–1444.
- [Mc90] J. A. McHugh, *Algorithmic Graph Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1990.
- [Mc82] R. McNaughton, *Elementary Computability, Formal Languages, and Automata*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

- [Me55] G. H. Mealy, “A Method for Synthesizing Sequential Circuits,” *Bell System Technical Journal*, 34 (1955), 1045–1079.
- [Me09] E. Mendelson, *Introduction to Mathematical Logic*, 5th ed., Chapman and Hall/CRC Press, Boca Raton, 2009.
- [MeOoVa97] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, FL, 1997.
- [MiRo91] J. G. Michaels and K. H. Rosen, *Applications of Discrete Mathematics*, McGraw-Hill, New York, 1991.
- [Mo69] J. R. Monk, *Introduction to Set Theory*, McGraw-Hill, New York, 1969.
- [Mo91] R. P. Morash, *Bridge to Abstract Mathematics*, McGraw-Hill, New York, 1991.
- [Mo56] E. F. Moore, “Gedanken-Experiments on Sequential Machines,” in *Automata Studies*, 129–153, Princeton University Press, Princeton, NJ, 1956.
- [Na00] Paul J. Nabin, *Duelling Idiots and Other Probability Puzzlers*, Princeton University Press, Princeton, NJ, 2000.
- [Ne85] C. V. Negoita, *Expert Systems and Fuzzy Systems*, Benjamin Cummings, Menlo Park, CA, 1985.
- [Ne10] M. Newman, *Networks: An Introduction*, Oxford University Press, New York, 2010.
- [NiMa95] Ulf Nilsson and Jan Maluszynski, *Logic, Programming, and Prolog*, 2d ed., Wiley, Chichester, England, 1995.
- [Or00] J. O’Rourke, *Computational Geometry in C*, Cambridge University Press, New York, 2000.
- [Or63] O. Ore, *Graphs and Their Uses*, Mathematical Association of America, Washington, DC, 1963.
- [Or88] O. Ore, *Number Theory and its History*, Dover, New York, 1988.
- [PaPi01] A. Papoulis and S. U. Pillai, *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill, New York, 2001.
- [Pe87] A. Pelc, “Solution of Ulam’s Problem on Searching with a Lie,” *Journal of Combinatorial Theory, Series A*, 44 (1987), 129–140.
- [Pe09] M. S. Petrović, *Famous Puzzles of Great Mathematicians*, American Mathematical Society, Providence, 2009.
- [PoSh81] I. Pohl and A. Shaw, *The Nature of Computation: An Introduction to Computer Science*, Computer Science Press, Rockville, MD, 1981.
- [Po62] George Pólya, *Mathematical Discovery*, Vols. 1 and 2, Wiley, New York, 1962.
- [Po71] George Pólya, *How to Solve It*, Princeton University Press, Princeton, NJ, 1971.
- [Po90] George Pólya, *Mathematics and Plausible Reasoning*, Princeton University Press, Princeton, NJ, 1990.
- [PoTaWo83] G. Pólya, R. E. Tarjan, and D. R. Woods, *Notes on Introductory Combinatorics*, Birkhäuser, Boston, 1983.
- [Pr57] R. C. Prim, “Shortest Connection Networks and Some Generalizations,” *Bell System Technical Journal*, 36 (1957), 1389–1401.
- [PuBr85] P. W. Purdom, Jr. and C. A. Brown, *The Analysis of Algorithms*, Holt, Rinehart & Winston, New York, 1985.
- [Qu52] W. V. Quine, “The Problem of Simplifying Truth Functions,” *American Mathematical Monthly*, 59 (1952), 521–531.
- [Qu55] W. V. Quine, “A Way to Simplify Truth Functions,” *American Mathematical Monthly*, 62 (1955), 627–631.
- [Ra62] T. Rado, “On Non-Computable Functions,” *Bell System Technical Journal* (May 1962), 877–884.
- [Ra92] Gregory J. E. Rawlins, *Compared to What? An Introduction to the Analysis of Algorithms*, Computer Science Press, New York, 1992.
- [ReNiDe77] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.
- [Ri58] J. Riordan, *An Introduction to Combinatorial Analysis*, Wiley, New York, 1958.
- [Ri68] J. Riordan, *Combinatorial Identities*, Wiley, New York, 1968.
- [RiShAd78] R. Rivest, A. Shamir, and L. Adleman, “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems,” *Communications of the Association for Computing Machinery*, 31, no. 2 (1978), 120–128.
- [Ro86] E. S. Roberts, *Thinking Recursively*, Wiley, New York, 1986.
- [Ro76] F. S. Roberts, *Discrete Mathematics Models*, Prentice-Hall, Englewood Cliffs, NJ, 1976.
- [RoTe03] F. S. Roberts and B. Tesman, *Applied Combinatorics*, 2d ed., Prentice-Hall, Englewood Cliffs, NJ, 2003.
- [RoFo80] D. F. Robinson and L. R. Foulds, *Digraphs: Theory and Techniques*, Gordon and Breach, New York, 1980.
- [Ro84] J. S. Rohl, *Recursion via Pascal*, Cambridge University Press, Cambridge, England, 1984.
- [Ro10] K. H. Rosen, *Elementary Number Theory and Its Applications*, 6th ed., Pearson, Boston, 2010.
- [Ro00] K. H. Rosen, *Handbook of Discrete and Combinatorial Mathematics*, CRC Press, Boca Raton, FL, 2000.
- [Ro09] Jason Rosenhouse, *The Monty Hall Problem*, Oxford University Press, New York, 2009.
- [Ro09a] Sheldon M. Ross, *A First Course in Probability Theory*, 7th ed., Prentice-Hall, Englewood Cliffs, NJ, 2009.
- [Ro02] Sheldon M. Ross, *Probability Models for Computer Science*, Harcourt/Academic Press, San Diego, 2002.
- [Ro59] B. Roy, “Transitivité et Connexité,” *C.R. Acad. Sci. Paris*, 249 (1959), 216.
- [Ry63] H. Ryser, *Combinatorial Mathematics*, Mathematical Association of America, Washington, DC, 1963.
- [SaKa86] T. L. Saaty and P. C. Kainen, *The Four-Color Problem: Assaults and Conquest*, Dover, New York, 1986.
- [Sa85] S. Sahni, *Concepts in Discrete Mathematics*, Camelot, Minneapolis, 1985.
- [Sa00] K. Sayood, *Introduction to Data Compression*, 2d ed., Academic Press, San Diego, 2000.

B-8 Suggested Reading

- [SePi89] J. Seberry and J. Pieprzyk, *Cryptography: An Introduction to Computer Security*, Prentice-Hall, Englewood Cliffs, NJ, 1989.
- [Se03] R. Sedgewick, *Algorithms in Java*, 3d ed., Addison-Wesley, Reading, MA, 2003.
- [SiHs88] H. J. Siegel and W. T. Hsu, "Interconnection Networks," in *Computer Architectures*, V. M. Milutinovic (ed.), North-Holland, New York, 1988, pp. 225–264.
- [Si99] S. Singh, *The Code Book*, Doubleday, New York, 1999.
- [Si66] A. Sinkov, *Elementary Cryptanalysis*, Mathematical Association of America, Washington, DC, 1966.
- [Si06] M. Sipser, *Introduction to the Theory of Computation*, Course Technology, Boston, 2006
- [SIPI95] N. J. A. Sloane and S. Plouffe, *The Encyclopedia of Integer Sequences*, Academic Press, New York, 1995.
- [Sm78] Raymond Smullyan, *What Is the Name of This Book?: The Riddle of Dracula and Other Logical Puzzles*, Prentice-Hall, Englewood Cliffs, NJ, 1978.
- [Sm92] Raymond Smullyan, *Lady or the Tiger? And Other Logic Puzzles Including a Mathematical Novel That Features Gödel's Great Discovery*, Times Book, New York, 1992.
- [Sm98] Raymond Smullyan, *The Riddle of Scheherazade: And Other Amazing Puzzles, Ancient & Modern*, Harvest Book, Fort Washington, PA, 1998.
- [So09] Daniel Solow, *How to Read and Do Proofs: An Introduction to Mathematical Thought Processes*, 5th ed., Wiley, New York, 2009.
- [So61] I. S. Sominskii, *Method of Mathematical Induction*, Blaisdell, New York, 1961.
- [Sp94] M. Spivak, *Calculus*, 3d ed., Publish or Perish, Wilmington, DE, 1994.
- [StMc77] D. Stanat and D. F. McAllister, *Discrete Mathematics in Computer Science*, Prentice-Hall, Englewood Cliffs, NJ, 1977.
- [St78] H. M. Stark, *An Introduction to Number Theory*, MIT Press, Cambridge, MA, 1978.
- [St05] Douglas R. Stinson, *Cryptography, Theory and Practice*, 3d ed., Chapman and Hall/CRC, Boca Raton, FL, 2005.
- [St94] P. K. Stockmeyer, "Variations on the Four-Post Tower of Hanoi Puzzle," *Congressus Numerantium* 102 (1994), 3–12.
- [St74] R. R. Stoll, *Sets, Logic, and Axiomatic Theories*, 2d ed., W. H. Freeman, San Francisco, 1974.
- [St09] G. W. Strang, *Linear Algebra and Its Applications*, 4th ed., Wellesley Cambridge Press, Wellesley, MA, 2009
- [Su87] P. Suppes, *Introduction to Logic*, D. Van Nostrand, Princeton, NJ, 1987.
- [Ta83] R. E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, 1983.
- [ThFi96] G. B. Thomas and R. L. Finney, *Calculus and Analytic Geometry*, 9th ed., Addison-Wesley, Reading, MA, 1996.
- [TrMa75] J. P. Tremblay and R. P. Manohar, *Discrete Mathematical Structures with Applications to Computer Science*, McGraw-Hill, New York, 1975.
- [Tu06] Alan Tucker, *Applied Combinatorics*, 5th ed., Wiley, New York, 2006.
- [Ve52] E. W. Veitch, "A Chart Method for Simplifying Truth Functions," *Proceedings of the ACM* (1952), 127–133.
- [Ve06] David Velleman, *How to Prove It: A Structured Approach*, Cambridge University Press, New York, 2006.
- [Vi71] N. Y. Vilenkin, *Combinatorics*, Academic Press, New York, 1971.
- [Wa80] M. Wand, *Induction, Recursion, and Programming*, North-Holland, New York, 1980.
- [Wa62] S. Warshall, "A Theorem on Boolean Matrices," *Journal of the ACM*, 9 (1962), 11–12.
- [We00] D. B. West, *Introduction to Graph Theory*, 2d ed., Prentice Hall, Englewood Cliffs, NJ, 2000.
- [Wi02] Herbert S. Wilf, *Algorithms and Complexity*, 2d ed., A. K. Peters, Natick, MA, 2002.
- [Wi85] S. G. Williamson, *Combinatorics for Computer Science*, Computer Science Press, Rockville, MD, 1985.
- [Wi85a] R. J. Wilson, *Introduction to Graph Theory*, 3d ed., Longman, Essex, England, 1985.
- [WiBe79] R. J. Wilson and L. W. Beineke, *Applications of Graph Theory*, Academic Press, London, 1979.
- [WiWa90] R. J. Wilson and J. J. Watkins, *Graphs, An Introductory Approach*, Wiley, New York, 1990.
- [Wi76] N. Wirth, *Algorithms + Data Structures = Programs*, Prentice-Hall, Englewood Cliffs, NJ, 1976.
- [Wi84] N. Wirth, "Data Structures and Algorithms," *Scientific American*, 251 (September 1984), 60–69.
- [Wo98] Robert S. Wolf, *Proof, Logic, and Conjecture: The Mathematician's Toolbox*, W. H. Freeman, New York, 1998.
- [Wo87] D. Wood, *Theory of Computation*, Harper & Row, New York, 1987.
- [Zd05] J. Zdziarski, *Ending Spam: Bayesian Content Filtering and the Art of Statistical Language Classification*, No Starch Press, San Francisco, 2005.
- [Zi91] H. J. Zimmermann, *Fuzzy Set Theory and Its Applications*, 2d ed., Kluwer, Boston, 1991.

Answers to Odd-Numbered Exercises

CHAPTER 1

Section 1.1

- 1.** a) Yes, T b) Yes, F c) Yes, T d) Yes, F e) No f) No
3. a) Mei does not have an MP3 player. b) There is pollution in New Jersey. c) $2 + 1 \neq 3$. d) The summer in Maine is not hot or it is not sunny. **5.** a) Steve does not have more than 100 GB free disk space on his laptop b) Zach does not block e-mails from Jennifer, or he does not block texts from Jennifer c) $7 \cdot 11 \cdot 13 \neq 999$ d) Diane did not ride her bike 100 miles on Sunday **7.** a) F b) T c) T d) T e) T **9.** a) Sharks have not been spotted near the shore. b) Swimming at the New Jersey shore is allowed, and sharks have been spotted near the shore. c) Swimming at the New Jersey shore is not allowed, or sharks have been spotted near the shore. d) If swimming at the New Jersey shore is allowed, then sharks have not been spotted near the shore. e) If sharks have not been spotted near the shore, then swimming at the New Jersey shore is allowed. f) If swimming at the New Jersey shore is not allowed, then sharks have not been spotted near the shore. g) Swimming at the New Jersey shore is allowed if and only if sharks have not been spotted near the shore. h) Swimming at the New Jersey shore is not allowed, and either swimming at the New Jersey shore is allowed or sharks have not been spotted near the shore. (Note that we were able to incorporate the parentheses by using the word “either” in the second half of the sentence.) **11.** a) $p \wedge q$ b) $p \wedge \neg q$ c) $\neg p \wedge \neg q$ d) $p \vee q$
 e) $p \rightarrow q$ f) $(p \vee q) \wedge (p \rightarrow \neg q)$ g) $q \leftrightarrow p$ **13.** a) $\neg p$
 b) $p \wedge \neg q$ c) $p \rightarrow q$ d) $\neg p \rightarrow \neg q$ e) $p \rightarrow q$ f) $q \wedge \neg p$
 g) $q \rightarrow p$ **15.** a) $r \wedge \neg p$ b) $\neg p \wedge q \wedge r$ c) $r \rightarrow (q \leftrightarrow \neg p)$
 d) $\neg q \wedge \neg p \wedge r$ e) $(q \rightarrow (\neg r \wedge \neg p)) \wedge \neg((\neg r \wedge \neg p) \rightarrow q)$
 f) $(p \wedge r) \rightarrow \neg q$ **17.** a) False b) True c) True d) True
19. a) Exclusive or: You get only one beverage. b) Inclusive or: Long passwords can have any combination of symbols.
 c) Inclusive or: A student with both courses is even more qualified. d) Either interpretation possible; a traveler might wish to pay with a mixture of the two currencies, or the store may not allow that. **21.** a) Inclusive or: It is allowable to take discrete mathematics if you have had calculus or computer science, or both. Exclusive or: It is allowable to take discrete mathematics if you have had calculus or computer science, but not if you have had both. Most likely the inclusive or is intended. b) Inclusive or: You can take the rebate, or you can get a low-interest loan, or you can get both the rebate and a low-interest loan. Exclusive or: You can take the rebate, or you can get a low-interest loan, but you cannot get both the rebate and a low-interest loan. Most likely the exclusive or is intended. c) Inclusive or: You can order two items from column A and none from column B, or three items from column B and none from column A, or five items including two from column A and three from column B. Exclusive or: You can

order two items from column A or three items from column B, but not both. Almost certainly the exclusive or is intended.

- d)** Inclusive or: More than 2 feet of snow or windchill below -100 , or both, will close school. Exclusive or: More than 2 feet of snow or windchill below -100 , but not both, will close school. Certainly the inclusive or is intended. **23.** a) If the wind blows from the northeast, then it snows. b) If it stays warm for a week, then the apple trees will bloom. c) If the Pistons win the championship, then they beat the Lakers. d) If you get to the top of Long’s Peak, then you must have walked 8 miles. e) If you are world-famous, then you will get tenure as a professor. f) If you drive more than 400 miles, then you will need to buy gasoline. g) If your guarantee is good, then you must have bought your CD player less than 90 days ago. h) If the water is not too cold, then Jan will go swimming. **25.** a) You buy an ice cream cone if and only if it is hot outside. b) You win the contest if and only if you hold the only winning ticket. c) You get promoted if and only if you have connections. d) Your mind will decay if and only if you watch television. e) The train runs late if and only if it is a day I take the train. **27.** a) Converse: “I will ski tomorrow only if it snows today.” Contrapositive: “If I do not ski tomorrow, then it will not have snowed today.” Inverse: “If it does not snow today, then I will not ski tomorrow.” b) Converse: “If I come to class, then there will be a quiz.” Contrapositive: “If I do not come to class, then there will not be a quiz.” Inverse: “If there is not going to be a quiz, then I don’t come to class.” c) Converse: “A positive integer is a prime if it has no divisors other than 1 and itself.” Contrapositive: “If a positive integer has a divisor other than 1 and itself, then it is not prime.” Inverse: “If a positive integer is not prime, then it has a divisor other than 1 and itself.” **29.** a) 2 b) 16 c) 64 d) 16

a)	<table border="1"> <thead> <tr> <th>p</th><th>$\neg p$</th><th>$p \wedge \neg p$</th></tr> </thead> <tbody> <tr> <td>T</td><td>F</td><td>F</td></tr> <tr> <td>F</td><td>T</td><td>F</td></tr> </tbody> </table>	p	$\neg p$	$p \wedge \neg p$	T	F	F	F	T	F	<table border="1"> <thead> <tr> <th>p</th><th>$\neg p$</th><th>$p \vee \neg p$</th></tr> </thead> <tbody> <tr> <td>T</td><td>F</td><td>T</td></tr> <tr> <td>F</td><td>T</td><td>T</td></tr> </tbody> </table>	p	$\neg p$	$p \vee \neg p$	T	F	T	F	T	T
p	$\neg p$	$p \wedge \neg p$																		
T	F	F																		
F	T	F																		
p	$\neg p$	$p \vee \neg p$																		
T	F	T																		
F	T	T																		

p	q	$\neg q$	$p \vee \neg q$	$(p \vee \neg q) \rightarrow q$
T	T	F	T	T
T	F	T	T	F
F	T	F	F	T
F	F	T	T	F

p	q	$p \vee q$	$p \wedge q$	$(p \vee q) \rightarrow (p \wedge q)$
T	T	T	T	T
T	F	T	F	F
F	T	T	F	F
F	F	F	F	T

S-2 Answers to Odd-Numbered Exercises

e)

p	q	$p \rightarrow q$	$\neg q$	$\neg p$	$\neg q \rightarrow \neg p$	$(p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p)$
T	T	T	F	F	T	T
T	F	F	T	F	F	T
F	T	T	F	T	T	T
F	F	T	T	T	T	T

f)

p	q	$p \rightarrow q$	$q \rightarrow p$	$(p \rightarrow q) \rightarrow (q \rightarrow p)$
T	T	T	T	T
T	F	F	T	T
F	T	T	F	F
F	F	T	T	T

33. For parts (a), (b), (c), (d), and (f) we have this table.

p	q	$(p \vee q) \rightarrow (p \oplus q)$	$(p \oplus q) \rightarrow (p \wedge q)$	$(p \vee q) \oplus (p \wedge q)$	$(p \leftrightarrow q) \oplus (\neg p \leftrightarrow q)$	$(p \oplus q) \rightarrow (p \oplus \neg q)$
T	T	F	T	F	T	T
T	F	T	F	T	T	F
F	T	T	F	T	T	F
F	F	T	T	F	T	T

For part (e) we have this table.

p	q	r	$\neg p$	$\neg r$	$p \leftrightarrow q$	$\neg p \leftrightarrow \neg r$	$(p \leftrightarrow q) \oplus (\neg p \leftrightarrow \neg r)$
T	T	T	F	F	T	T	F
T	T	F	F	T	T	F	T
T	F	T	F	F	T	T	T
T	F	F	F	T	F	F	F
F	T	T	T	F	F	F	F
F	T	F	T	T	F	T	T
F	F	T	T	F	T	F	T
F	F	F	T	T	T	T	F

35.

p	q	$p \rightarrow \neg q$	$\neg p \leftrightarrow q$	$(p \rightarrow q) \vee (\neg p \rightarrow q)$	$(p \rightarrow q) \wedge (\neg p \rightarrow q)$	$(p \leftrightarrow q) \vee (\neg p \leftrightarrow q)$	$(\neg p \leftrightarrow \neg q) \leftrightarrow (p \leftrightarrow q)$
T	T	F	F	T	T	T	T
T	F	T	T	T	F	T	T
F	T	T	T	T	T	T	T
F	F	T	F	T	F	T	T

37.

p	q	r	$p \rightarrow (\neg q \vee r)$	$\neg p \rightarrow (q \rightarrow r)$	$(p \rightarrow q) \vee (\neg p \rightarrow r)$	$(p \rightarrow q) \wedge (\neg p \rightarrow r)$	$(p \leftrightarrow q) \vee (\neg q \leftrightarrow r)$	$(\neg p \leftrightarrow \neg q) \leftrightarrow (q \leftrightarrow r)$
T	T	T	T	T	T	T	T	T
T	T	F	F	T	T	T	T	F
T	F	T	T	T	T	F	T	T
T	F	F	T	T	T	F	F	F
F	T	T	T	T	T	T	F	F
F	T	F	T	F	T	F	T	T
F	F	T	T	T	T	T	T	F
F	F	F	T	T	T	F	T	T

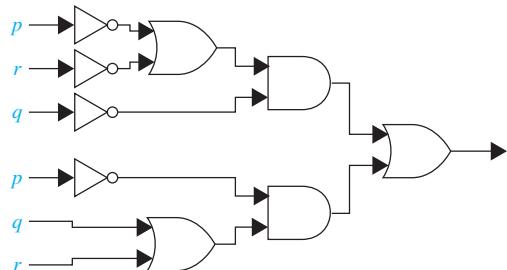
				$p \leftrightarrow q$	$r \leftrightarrow s$	$(p \leftrightarrow q) \leftrightarrow (r \leftrightarrow s)$
p	q	r	s			
T	T	T	T	T	T	T
T	T	T	F	T	F	F
T	T	F	T	T	F	F
T	T	F	F	T	T	T
T	F	T	T	F	T	F
T	F	T	F	F	F	T
T	F	F	T	F	F	T
T	F	F	F	F	T	F
F	T	T	T	F	T	F
F	T	T	F	F	F	T
F	T	F	F	F	T	F
F	F	T	T	T	T	T
F	F	T	F	T	F	F
F	F	F	T	F	F	T
F	F	F	F	T	T	T

41. The first clause is true if and only if at least one of p, q , and r is true. The second clause is true if and only if at least one of the three variables is false. Therefore the entire statement is true if and only if there is at least one T and one F among the truth values of the variables, in other words, that they don't all have the same truth value. **43. a)** Bitwise OR is 111 1111; bitwise AND is 000 0000; bitwise XOR is 111 1111. **b)** Bitwise OR is 1111 1010; bitwise AND is 1010 0000; bitwise XOR is 0101 1010. **c)** Bitwise OR is 10 0111 1001; bitwise AND is 00 0100 0000; bitwise XOR is 10 0011 1001. **d)** Bitwise OR is 11 1111 1111; bitwise AND is 00 0000 0000; bitwise XOR is 11 1111 1111. **45.** 0.2, 0.6 **47.** 0.8, 0.6 **49. a)** The 99th statement is true and the rest are false. **b)** Statements 1 through 50 are all true and statements 51 through 100 are all false. **c)** This cannot happen; it is a paradox, showing that these cannot be statements.

Section 1.2

- 1.** $e \rightarrow a$ **3.** $g \rightarrow (r \wedge (\neg m) \wedge (\neg b))$ **5.** $e \rightarrow (a \wedge (b \vee p) \wedge r)$ **7. a)** $q \rightarrow p$ **b)** $q \wedge \neg p$ **c)** $q \rightarrow p$ **d)** $\neg q \rightarrow \neg p$ **9.** Not consistent **11.** Consistent **13.** NEW AND JERSEY AND BEACHES, (JERSEY AND BEACHES) NOT NEW **15.** "If I were to ask you whether the right branch leads to the ruins, would you answer yes?" **17.** If the first professor did not want coffee, then he would know that the answer to the hostess's question was "no." Therefore the hostess and the remaining professors know that the first professor did want coffee. Similarly, the second professor must want coffee. When the third professor said "no," the hostess knows that the third professor does not want coffee. **19.** A is a knight and B is a knave. **21.** A is a knight and B is a knight. **23.** A is a knave and B is a knight. **25.** A is the knight, B is the spy, C is the knave. **27.** A is the knight, B is the spy, C is the knave. **29.** Any of the three can be the knight, any can be the spy, any can be the knave. **31.** No solutions **33.** In order of decreasing salary: Fred, Maggie, Janice **35.** The detective can

determine that the butler and cook are lying but cannot determine whether the gardener is telling the truth or whether the handyman is telling the truth. **37.** The Japanese man owns the zebra, and the Norwegian drinks water. **39.** One honest, 49 corrupt **41. a)** $\neg(p \wedge (q \vee \neg r))$ **b)** $((\neg p) \wedge (\neg q)) \vee (p \wedge r)$

43.

Section 1.3

- 1.** The equivalences follow by showing that the appropriate pairs of columns of this table agree.

p	$p \wedge T$	$p \vee F$	$p \wedge F$	$p \vee T$	$p \vee p$	$p \wedge p$
T	T	T	F	T	T	T
F	F	F	F	T	F	F

3. a)

p	q	$p \vee q$	$q \vee p$
T	T	T	T
T	F	T	T
F	T	T	T
F	F	F	F

b)

p	q	$p \wedge q$	$q \wedge p$
T	T	T	T
T	F	F	F
F	T	F	F
F	F	F	F

5.

p	q	r	$q \vee r$	$p \wedge (q \vee r)$	$p \wedge q$	$p \wedge r$	$(p \wedge q) \vee (p \wedge r)$
T	T	T	T	T	T	T	T
T	T	F	T	T	T	F	T
T	F	T	T	T	F	T	T
T	F	F	F	F	F	F	F
F	T	T	T	F	F	F	F
F	T	F	T	F	F	F	F
F	F	T	T	F	F	F	F
F	F	F	F	F	F	F	F

- 7. a)** Jan is not rich, or Jan is not happy. **b)** Carlos will not bicycle tomorrow, and Carlos will not run tomorrow. **c)** Mei does not walk to class, and Mei does not take the bus to class. **d)** Ibrahim is not smart, or Ibrahim is not hard working.

9. a)

p	q	$p \wedge q$	$(p \wedge q) \rightarrow p$
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	T

S-4 Answers to Odd-Numbered Exercises

b)

p	q	$p \vee q$	$p \rightarrow (p \vee q)$
T	T	T	T
T	F	T	T
F	T	T	T
F	F	F	T

c)

p	q	$\neg p$	$p \rightarrow q$	$\neg p \rightarrow (p \rightarrow q)$
T	T	F	T	T
T	F	F	F	T
F	T	T	T	T
F	F	T	T	T

d)

p	q	$p \wedge q$	$p \rightarrow q$	$(p \wedge q) \rightarrow (p \rightarrow q)$
T	T	T	T	T
T	F	F	F	T
F	T	F	T	T
F	F	F	T	T

e)

p	q	$p \rightarrow q$	$\neg(p \rightarrow q)$	$\neg(p \rightarrow q) \rightarrow p$
T	T	T	F	T
T	F	F	T	T
F	T	T	F	T
F	F	T	F	T

f)

p	q	$p \rightarrow q$	$\neg(p \rightarrow q)$	$\neg q$	$\neg(p \rightarrow q) \rightarrow \neg q$
T	T	T	F	F	T
T	F	F	T	T	T
F	T	T	F	F	T
F	F	T	F	T	T

11. In each case we will show that if the hypothesis is true, then the conclusion is also. a) If the hypothesis $p \wedge q$ is true, then by the definition of conjunction, the conclusion p must also be true. b) If the hypothesis p is true, by the definition of disjunction, the conclusion $p \vee q$ is also true. c) If the hypothesis $\neg p$ is true, that is, if p is false, then the conclusion $p \rightarrow q$ is true. d) If the hypothesis $p \wedge q$ is true, then both p and q are true, so the conclusion $p \rightarrow q$ is also true. e) If the hypothesis $\neg(p \rightarrow q)$ is true, then $p \rightarrow q$ is false, so the conclusion p is true (and q is false). f) If the hypothesis $\neg(p \rightarrow q)$ is true, then $p \rightarrow q$ is false, so p is true and q is false. Hence, the conclusion $\neg q$ is true. 13. That the fourth column of the truth table shown is identical to the first column proves part (a), and that the sixth column is identical to the first column proves part (b).

p	q	$p \wedge q$	$p \vee (p \wedge q)$	$p \vee q$	$p \wedge (p \vee q)$
T	T	T	T	T	T
T	F	F	T	T	T
F	T	F	F	T	F
F	F	F	F	F	F

15. It is a tautology. 17. Each of these is true precisely when p and q have opposite truth values. 19. The proposition

$\neg p \leftrightarrow q$ is true when $\neg p$ and q have the same truth values, which means that p and q have different truth values. Similarly, $p \leftrightarrow \neg q$ is true in exactly the same cases. Therefore, these two expressions are logically equivalent. 21. The proposition $\neg(p \leftrightarrow q)$ is true when $p \leftrightarrow q$ is false, which means that p and q have different truth values. Because this is precisely when $\neg p \leftrightarrow q$ is true, the two expressions are logically equivalent. 23. For $(p \rightarrow r) \wedge (q \rightarrow r)$ to be false, one of the two conditional statements must be false, which happens exactly when r is false and at least one of p and q is true. But these are precisely the cases in which $p \vee q$ is true and r is false, which is precisely when $(p \vee q) \rightarrow r$ is false. Because the two propositions are false in exactly the same situations, they are logically equivalent. 25. For $(p \rightarrow r) \vee (q \rightarrow r)$ to be false, both of the two conditional statements must be false, which happens exactly when r is false and both p and q are true. But this is precisely the case in which $p \wedge q$ is true and r is false, which is precisely when $(p \wedge q) \rightarrow r$ is false. Because the two propositions are false in exactly the same situations, they are logically equivalent. 27. This fact was observed in Section 1 when the biconditional was first defined. Each of these is true precisely when p and q have the same truth values. 29. The last column is all Ts.

p	q	r	$p \rightarrow q$	$q \rightarrow r$	$(p \rightarrow q) \wedge (q \rightarrow r)$	$p \rightarrow r$	$(p \rightarrow q) \wedge (q \rightarrow r) \rightarrow (p \rightarrow r)$
T	T	T	T	T	T	T	T
T	T	F	T	F	F	F	T
T	F	T	F	T	F	T	T
T	F	F	F	T	F	F	T
F	T	T	T	T	T	T	T
F	T	F	T	F	F	T	T
F	F	T	T	T	T	T	T
F	F	F	T	T	T	T	T

31. These are not logically equivalent because when p , q , and r are all false, $(p \rightarrow q) \rightarrow r$ is false, but $p \rightarrow (q \rightarrow r)$ is true. 33. Many answers are possible. If we let r be true and p , q , and s be false, then $(p \rightarrow q) \rightarrow (r \rightarrow s)$ will be false, but $(p \rightarrow r) \rightarrow (q \rightarrow s)$ will be true. 35. a) $p \vee \neg q \vee \neg r$ b) $(p \vee q \vee r) \wedge s$ c) $(p \wedge T) \vee (q \wedge F)$ 37. If we take duals twice, every \vee changes to an \wedge and then back to an \vee , every \wedge changes to an \vee and then back to an \wedge , every T changes to an F and then back to a T , every F changes to a T and then back to an F . Hence, $(s^*)^* = s$. 39. Let p and q be equivalent compound propositions involving only the operators \wedge , \vee , and \neg , and T and F . Note that $\neg p$ and $\neg q$ are also equivalent. Use De Morgan's laws as many times as necessary to push negations in as far as possible within these compound propositions, changing \vee s to \wedge s, and vice versa, and changing T s to F s, and vice versa. This shows that $\neg p$ and $\neg q$ are the same as p^* and q^* except that each atomic proposition p_i within them is replaced by its negation. From this we can conclude that p^* and q^* are equivalent because $\neg p$ and $\neg q$

are. **41.** $(p \wedge q \wedge \neg r) \vee (p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r)$ **43.** Given a compound proposition p , form its truth table and then write down a proposition q in disjunctive normal form that is logically equivalent to p . Because q involves only \neg , \wedge , and \vee , this shows that these three operators form a functionally complete set. **45.** By Exercise 43, given a compound proposition p , we can write down a proposition q that is logically equivalent to p and involves only \neg , \wedge , and \vee . By De Morgan's law we can eliminate all the \wedge 's by replacing each occurrence of $p_1 \wedge p_2 \wedge \dots \wedge p_n$ with $\neg(\neg p_1 \vee \neg p_2 \vee \dots \vee \neg p_n)$. **47.** $\neg(p \wedge q)$ is true when either p or q , or both, are false, and is false when both p and q are true. Because this was the definition of $p \downarrow q$, the two compound propositions are logically equivalent. **49.** $\neg(p \vee q)$ is true when both p and q are false, and is false otherwise. Because this was the definition of $p \downarrow q$, the two are logically equivalent. **51.** $((p \downarrow p) \downarrow q) \downarrow ((p \downarrow p) \downarrow q)$ **53.** This follows immediately from the truth table or definition of $p \downarrow q$. **55.** 16 **57.** If the database is open, then either the system is in its initial state or the monitor is put in a closed state. **59.** All nine **61.** **a)** Satisfiable **b)** Not satisfiable **c)** Not satisfiable **63.** Use the same propositions as were given in the text for a 9×9 Sudoku puzzle, with the variables indexed from 1 to 4, instead of from 1 to 9, and with a similar change for the propositions for the 2×2 blocks: $\bigwedge_{r=0}^1 \bigwedge_{s=0}^1 \bigwedge_{n=1}^4 \bigvee_{i=1}^2 \bigvee_{j=1}^2 p(2r+i, 2s+j, n)$ **65.** $\bigvee_{i=1}^9 p(i, j, n)$ asserts that column j contains the number n , so $\bigwedge_{n=1}^9 \bigvee_{i=1}^9 p(i, j, n)$ asserts that column j contains all 9 numbers; therefore $\bigwedge_{j=1}^9 \bigwedge_{n=1}^9 \bigvee_{i=1}^9 p(i, j, n)$ asserts that every column contains every number.

Section 1.4

- 1. a)** T **b)** T **c)** F **3. a)** T **b)** F **c)** F **d)** F **5. a)** There is a student who spends more than 5 hours every weekday in class. **b)** Every student spends more than 5 hours every weekday in class. **c)** There is a student who does not spend more than 5 hours every weekday in class. **d)** No student spends more than 5 hours every weekday in class. **7. a)** Every comedian is funny. **b)** Every person is a funny comedian. **c)** There exists a person such that if she or he is a comedian, then she or he is funny. **d)** Some comedians are funny. **9. a)** $\exists x(P(x) \wedge Q(x))$ **b)** $\exists x(P(x) \wedge \neg Q(x))$ **c)** $\forall x(P(x) \vee Q(x))$ **d)** $\forall x \neg(P(x) \vee Q(x))$ **11. a)** T **b)** T **c)** F **d)** F **e)** T **f)** F **13. a)** T **b)** T **c)** T **d)** T **15. a)** T **b)** F **c)** T **d)** F **17. a)** $P(0) \vee P(1) \vee P(2) \vee P(3) \vee P(4)$ **b)** $P(0) \wedge P(1) \wedge P(2) \wedge P(3) \wedge P(4)$ **c)** $\neg P(0) \vee \neg P(1) \vee \neg P(2) \vee \neg P(3) \vee \neg P(4)$ **d)** $\neg P(0) \wedge \neg P(1) \wedge \neg P(2) \wedge \neg P(3) \wedge \neg P(4)$ **e)** $\neg(P(0) \vee P(1) \wedge P(2) \wedge P(3) \wedge P(4))$ **f)** $\neg(P(0) \wedge P(1) \vee P(2) \wedge P(3) \vee P(4))$ **19. a)** $P(1) \vee P(2) \vee P(3) \vee P(4) \vee P(5)$ **b)** $P(1) \wedge P(2) \wedge P(3) \wedge P(4) \wedge P(5)$ **c)** $\neg(P(1) \vee P(2) \vee P(3) \vee P(4) \vee P(5))$ **d)** $\neg(P(1) \wedge P(2) \wedge P(3) \wedge P(4) \wedge P(5))$ **e)** $(P(1) \wedge P(2) \wedge P(4) \wedge P(5)) \vee (\neg P(1) \vee \neg P(2) \vee \neg P(3) \vee \neg P(4))$

21. Many answers are possible. **a)** All students in your discrete mathematics class; all students in the world **b)** All United States senators; all college football players **c)** George W. Bush and Jeb Bush; all politicians in the United States **d)** Bill Clinton and George W. Bush; all politicians in the United States **23.** Let $C(x)$ be the propositional function “ x is in your class.” **a)** $\exists x H(x)$ and $\exists x(C(x) \wedge H(x))$, where $H(x)$ is “ x can speak Hindi” **b)** $\forall x F(x)$ and $\forall x(C(x) \rightarrow F(x))$, where $F(x)$ is “ x is friendly” **c)** $\exists x \neg B(x)$ and $\exists x(C(x) \wedge \neg B(x))$, where $B(x)$ is “ x was born in California” **d)** $\exists x M(x)$ and $\exists x(C(x) \wedge M(x))$, where $M(x)$ is “ x has been in a movie” **e)** $\forall x \neg L(x)$ and $\forall x(C(x) \rightarrow \neg L(x))$, where $L(x)$ is “ x has taken a course in logic programming” **25.** Let $P(x)$ be “ x is perfect”; let $F(x)$ be “ x is your friend”; and let the domain be all people. **a)** $\forall x \neg P(x)$ **b)** $\neg \forall x P(x)$ **c)** $\forall x(F(x) \rightarrow P(x))$ **d)** $\exists x(F(x) \wedge P(x))$ **e)** $\forall x(F(x) \wedge P(x))$ or $(\forall x F(x)) \wedge (\forall x P(x))$ **f)** $(\neg \forall x F(x)) \vee (\exists x \neg P(x))$ **27.** Let $Y(x)$ be the propositional function that x is in your school or class, as appropriate. **a)** If we let $V(x)$ be “ x has lived in Vietnam,” then we have $\exists x V(x)$ if the domain is just your schoolmates, or $\exists x(Y(x) \wedge V(x))$ if the domain is all people. If we let $D(x, y)$ mean that person x has lived in country y , then we can rewrite this last one as $\exists x(Y(x) \wedge D(x, \text{Vietnam}))$. **b)** If we let $H(x)$ be “ x can speak Hindi,” then we have $\exists x \neg H(x)$ if the domain is just your schoolmates, or $\exists x(Y(x) \wedge \neg H(x))$ if the domain is all people. If we let $S(x, y)$ mean that person x can speak language y , then we can rewrite this last one as $\exists x(Y(x) \wedge \neg S(x, \text{Hindi}))$. **c)** If we let $J(x)$, $P(x)$, and $C(x)$ be the propositional functions asserting x 's knowledge of Java, Prolog, and C++, respectively, then we have $\exists x(J(x) \wedge P(x) \wedge C(x))$ if the domain is just your schoolmates, or $\exists x(Y(x) \wedge J(x) \wedge P(x) \wedge C(x))$ if the domain is all people. If we let $K(x, y)$ mean that person x knows programming language y , then we can rewrite this last one as $\exists x(Y(x) \wedge K(x, \text{Java}) \wedge K(x, \text{Prolog}) \wedge K(x, \text{C++}))$. **d)** If we let $T(x)$ be “ x enjoys Thai food,” then we have $\forall x T(x)$ if the domain is just your schoolmates, or $\forall x(Y(x) \rightarrow T(x))$ if the domain is all people. If we let $E(x, y)$ mean that person x enjoys food of type y , then we can rewrite this last one as $\forall x(Y(x) \rightarrow E(x, \text{Thai}))$. **e)** If we let $H(x)$ be “ x plays hockey,” then we have $\exists x \neg H(x)$ if the domain is just your schoolmates, or $\exists x(Y(x) \wedge \neg H(x))$ if the domain is all people. If we let $P(x, y)$ mean that person x plays game y , then we can rewrite this last one as $\exists x(Y(x) \wedge \neg P(x, \text{hockey}))$. **29.** Let $T(x)$ mean that x is a tautology and $C(x)$ mean that x is a contradiction. **a)** $\exists x T(x)$ **b)** $\forall x(C(x) \rightarrow T(\neg x))$ **c)** $\exists x \exists y(\neg T(x) \wedge \neg C(x) \wedge \neg T(y) \wedge \neg C(y) \wedge T(x \vee y))$ **d)** $\forall x \forall y((T(x) \wedge T(y)) \rightarrow T(x \wedge y))$ **31. a)** $Q(0, 0, 0) \wedge Q(0, 1, 0)$ **b)** $Q(0, 1, 1) \vee Q(1, 1, 1) \vee Q(2, 1, 1)$ **c)** $\neg Q(0, 0, 0) \vee \neg Q(0, 0, 1) \wedge \neg Q(1, 0, 1) \vee \neg Q(2, 0, 1)$ **33. a)** Let $T(x)$ be the predicate that x can learn new tricks, and let the domain be old dogs. Original is $\exists x T(x)$. Negation is $\forall x \neg T(x)$: “No old dogs can learn new tricks.” **b)** Let $C(x)$ be the predicate that x knows calculus, and let the domain be rabbits. Original is $\neg \exists x C(x)$.

S-6 Answers to Odd-Numbered Exercises

Negation is $\exists x C(x)$: “There is a rabbit that knows calculus.” **c)** Let $F(x)$ be the predicate that x can fly, and let the domain be birds. Original is $\forall x F(x)$. Negation is $\exists x \neg F(x)$: “There is a bird who cannot fly.” **d)** Let $T(x)$ be the predicate that x can talk, and let the domain be dogs. Original is $\neg \exists x T(x)$. Negation is $\exists x T(x)$: “There is a dog that talks.” **e)** Let $F(x)$ and $R(x)$ be the predicates that x knows French and knows Russian, respectively, and let the domain be people in this class. Original is $\neg \exists x (F(x) \wedge R(x))$. Negation is $\exists x (F(x) \wedge R(x))$: “There is someone in this class who knows French and Russian.” **35. a)** There is no counterexample. **b)** $x = 0$ **c)** $x = 2$ **37. a)** $\forall x ((F(x, 25,000) \vee S(x, 25)) \rightarrow E(x))$, where $E(x)$ is “Person x qualifies as an elite flyer in a given year,” $F(x, y)$ is “Person x flies more than y miles in a given year,” and $S(x, y)$ is “Person x takes more than y flights in a given year” **b)** $\forall x (((M(x) \wedge T(x, 3)) \vee (\neg M(x) \wedge T(x, 3.5))) \rightarrow Q(x))$, where $Q(x)$ is “Person x qualifies for the marathon,” $M(x)$ is “Person x is a man,” and $T(x, y)$ is “Person x has run the marathon in less than y hours” **c)** $M \rightarrow ((H(60) \vee (H(45) \wedge T)) \wedge \forall y G(B, y))$, where M is the proposition “The student received a masters degree,” $H(x)$ is “The student took at least x course hours,” T is the proposition “The student wrote a thesis,” and $G(x, y)$ is “The person got grade x or higher in course y ” **d)** $\exists x ((T(x, 21) \wedge G(x, 4.0))$, where $T(x, y)$ is “Person x took more than y credit hours” and $G(x, p)$ is “Person x earned grade point average p ” (we assume that we are talking about one given semester) **39. a)** If there is a printer that is both out of service and busy, then some job has been lost. **b)** If every printer is busy, then there is a job in the queue. **c)** If there is a job that is both queued and lost, then some printer is out of service. **d)** If every printer is busy and every job is queued, then some job is lost. **41. a)** $(\exists x F(x, 10)) \rightarrow \exists x S(x)$, where $F(x, y)$ is “Disk x has more than y kilobytes of free space,” and $S(x)$ is “Mail message x can be saved” **b)** $(\exists x A(x)) \rightarrow \forall x (Q(x) \rightarrow T(x))$, where $A(x)$ is “Alert x is active,” $Q(x)$ is “Message x is queued,” and $T(x)$ is “Message x is transmitted” **c)** $\forall x ((x \neq \text{main console}) \rightarrow T(x))$, where $T(x)$ is “The diagnostic monitor tracks the status of system x ” **d)** $\forall x (\neg L(x) \rightarrow B(x))$, where $L(x)$ is “The host of the conference call put participant x on a special list” and $B(x)$ is “Participant x was billed” **43.** They are not equivalent. Let $P(x)$ be any propositional function that is sometimes true and sometimes false, and let $Q(x)$ be any propositional function that is always false. Then $\forall x (P(x) \rightarrow Q(x))$ is false but $\forall x P(x) \rightarrow \forall x Q(x)$ is true. **45.** Both statements are true precisely when at least one of $P(x)$ and $Q(x)$ is true for at least one value of x in the domain. **47. a)** If A is true, then both sides are logically equivalent to $\forall x P(x)$. If A is false, the left-hand side is clearly false. Furthermore, for every x , $P(x) \wedge A$ is false, so the right-hand side is false. Hence, the two sides are logically equivalent. **b)** If A is true, then both sides are logically equivalent to $\exists x P(x)$. If A is false, the left-hand side is clearly false. Furthermore, for every x , $P(x) \wedge A$ is false, so $\exists x (P(x) \wedge A)$ is false. Hence, the two sides are logically equivalent. **49.** We can establish these equivalences by arguing that one side is true if and only if the

other side is true. **a)** Suppose that A is true. Then for each x , $P(x) \rightarrow A$ is true; therefore the left-hand side is always true in this case. By similar reasoning the right-hand side is always true in this case. Therefore, the two propositions are logically equivalent when A is true. On the other hand, suppose that A is false. There are two subcases. If $P(x)$ is false for every x , then $P(x) \rightarrow A$ is vacuously true, so the left-hand side is vacuously true. The same reasoning shows that the right-hand side is also true, because in this subcase $\exists x P(x)$ is false. For the second subcase, suppose that $P(x)$ is true for some x . Then for that x , $P(x) \rightarrow A$ is false, so the left-hand side is false. The right-hand side is also false, because in this subcase $\exists x P(x)$ is true but A is false. Thus in all cases, the two propositions have the same truth value. **b)** If A is true, then both sides are trivially true, because the conditional statements have true conclusions. If A is false, then there are two subcases. If $P(x)$ is false for some x , then $P(x) \rightarrow A$ is vacuously true for that x , so the left-hand side is true. The same reasoning shows that the right-hand side is true, because in this subcase $\forall x P(x)$ is false. For the second subcase, suppose that $P(x)$ is true for every x . Then for every x , $P(x) \rightarrow A$ is false, so the left-hand side is false (there is no x making the conditional statement true). The right-hand side is also false, because it is a conditional statement with a true hypothesis and a false conclusion. Thus in all cases, the two propositions have the same truth value. **51.** To show these are not logically equivalent, let $P(x)$ be the statement “ x is positive,” and let $Q(x)$ be the statement “ x is negative” with domain the set of integers. Then $\exists x P(x) \wedge \exists x Q(x)$ is true, but $\exists x (P(x) \wedge Q(x))$ is false. **53. a)** True **b)** False, unless the domain consists of just one element **c)** True **55. a)** Yes **b)** No **c)** juana, kiko **d)** math273, cs301 **e)** juana, kiko **57. sibling(X, Y) :- mother(M, X), mother(M, Y), father(F, X), father(F, Y)** **59. a)** $\forall x (P(x) \rightarrow \neg Q(x))$ **b)** $\forall x (Q(x) \rightarrow R(x))$ **c)** $\forall x (P(x) \rightarrow \neg R(x))$ **d)** The conclusion does not follow. There may be vain professors, because the premises do not rule out the possibility that there are other vain people besides ignorant ones. **61. a)** $\forall x (P(x) \rightarrow \neg Q(x))$ **b)** $\forall x (R(x) \rightarrow \neg S(x))$ **c)** $\forall x (\neg Q(x) \rightarrow S(x))$ **d)** $\forall x (P(x) \rightarrow \neg R(x))$ **e)** The conclusion follows. Suppose x is a baby. Then by the first premise, x is illogical, so by the third premise, x is despised. The second premise says that if x could manage a crocodile, then x would not be despised. Therefore, x cannot manage a crocodile.

Section 1.5

- 1. a)** For every real number x there exists a real number y such that x is less than y . **b)** For every real number x and real number y , if x and y are both nonnegative, then their product is nonnegative. **c)** For every real number x and real number y , there exists a real number z such that $xy = z$. **3. a)** There

is some student in your class who has sent a message to some student in your class. **b)** There is some student in your class who has sent a message to every student in your class. **c)** Every student in your class has sent a message to at least one student in your class. **d)** There is a student in your class who has been sent a message by every student in your class. **e)** Every student in your class has been sent a message from at least one student in your class. **f)** Every student in the class has sent a message to every student in the class. **5. a)** Sarah Smith has visited www.att.com. **b)** At least one person has visited www.imdb.org. **c)** Jose Orez has visited at least one website. **d)** There is a website that both Ashok Puri and Cindy Yoon have visited. **e)** There is a person besides David Belcher who has visited all the websites that David Belcher has visited. **f)** There are two different people who have visited exactly the same websites. **7. a)** Abdallah Hussein does not like Japanese cuisine. **b)** Some student at your school likes Korean cuisine, and everyone at your school likes Mexican cuisine. **c)** There is some cuisine that either Monique Arsenault or Jay Johnson likes. **d)** For every pair of distinct students at your school, there is some cuisine that at least one them does not like. **e)** There are two students at your school who like exactly the same set of cuisines. **f)** For every pair of students at your school, there is some cuisine about which they have the same opinion (either they both like it or they both do not like it). **9. a)** $\forall x L(x, \text{Jerry})$ **b)** $\forall x \exists y L(x, y)$ **c)** $\exists y \forall x L(x, y)$ **d)** $\forall x \exists y \neg L(x, y)$ **e)** $\exists x \neg L(\text{Lydia}, x)$ **f)** $\exists x \forall y \neg L(y, x)$ **g)** $\exists x (\forall y L(y, x) \wedge \forall z ((\forall w L(w, z)) \rightarrow z = x))$ **h)** $\exists x \exists y (x \neq y \wedge L(\text{Lynn}, x) \wedge L(\text{Lynn}, y) \wedge \forall z (L(\text{Lynn}, z) \rightarrow (z = x \vee z = y)))$ **i)** $\forall x L(x, x)$ **j)** $\exists x \forall y (L(x, y) \leftrightarrow x = y)$ **11. a)** $A(\text{Lois, Professor Michaels})$ **b)** $\forall x (S(x) \rightarrow A(x, \text{Professor Gross}))$ **c)** $\forall x (F(x) \rightarrow (A(x, \text{Professor Miller}) \vee A(\text{Professor Miller}, x)))$ **d)** $\exists x (S(x) \wedge \forall y (F(y) \rightarrow \neg A(x, y)))$ **e)** $\exists x (F(x) \wedge \forall y (S(y) \rightarrow \neg A(y, x)))$ **f)** $\forall y (F(y) \rightarrow \exists x (S(x) \vee A(x, y)))$ **g)** $\exists x (F(x) \wedge \forall y ((F(y) \wedge (y \neq x)) \rightarrow A(x, y)))$ **h)** $\exists x (S(x) \wedge \forall y (F(y) \rightarrow \neg A(y, x)))$ **13. a)** $\neg M(\text{Chou, Koko})$ **b)** $\neg M(\text{Arlene, Sarah}) \wedge \neg T(\text{Arlene, Sarah})$ **c)** $\neg M(\text{Deborah, Jose})$ **d)** $\forall x M(x, \text{Ken})$ **e)** $\forall x \neg T(x, \text{Nina})$ **f)** $\forall x (T(x, \text{Avi}) \vee M(x, \text{Avi}))$ **g)** $\exists x \forall y (y \neq x \rightarrow M(x, y))$ **h)** $\exists x \forall y (y \neq x \rightarrow (M(x, y) \vee T(x, y)))$ **i)** $\exists x \exists y (x \neq y \wedge M(x, y) \wedge M(y, x))$ **j)** $\exists x M(x, x)$ **k)** $\exists x \forall y (x \neq y \rightarrow (\neg M(x, y) \wedge \neg T(y, x)))$ **l)** $\forall x (\exists y (x \neq y \wedge (M(y, x) \vee T(y, x))))$ **m)** $\exists x \exists y (x \neq y \wedge M(x, y) \wedge T(y, x))$ **n)** $\exists x \exists y (x \neq y \wedge \forall z ((z \neq x \wedge z \neq y) \rightarrow (M(x, z) \vee M(y, z) \vee T(x, z) \vee T(y, z))))$ **15. a)** $\forall x P(x)$, where $P(x)$ is “ x needs a course in discrete mathematics” and the domain consists of all computer science students **b)** $\exists x P(x)$, where $P(x)$ is “ x owns a personal computer” and the domain consists of all students in this class **c)** $\forall x \exists y P(x, y)$, where $P(x, y)$ is “ x has taken y ,” the domain for x consists of all students in this class, and the domain for y consists of all computer science classes **d)** $\exists x \exists y P(x, y)$, where $P(x, y)$ and domains are the same as in part (c) **e)** $\forall x \forall y P(x, y)$, where $P(x, y)$ is “ x has been in y ,” the domain for x consists of all students in this class, and the domain for y consists of all buildings on campus **f)** $\exists x \exists y \forall z (P(z, y) \rightarrow Q(x, z))$, where $P(z, y)$ is “ z is in

y ” and $Q(x, z)$ is “ x has been in z ; the domain for x consists of all students in the class, the domain for y consists of all buildings on campus, and the domain of z consists of all rooms. **g)** $\forall x \forall y \exists z (P(z, y) \wedge Q(x, z))$, with same environment as in part (f) **17. a)** $\forall u \exists m (A(u, m) \wedge \forall n (n \neq m \rightarrow \neg A(u, n)))$, where $A(u, m)$ means that user u has access to mailbox m **b)** $\exists p \forall e (H(e) \wedge S(p, \text{running})) \rightarrow S$ (kernel, working correctly), where $H(e)$ means that error condition e is in effect and $S(x, y)$ means that the status of x is y **c)** $\forall u \forall s (E(s, .edu) \rightarrow A(u, s))$, where $E(s, x)$ means that website s has extension x , and $A(u, s)$ means that user u can access website s **d)** $\exists x \exists y (x \neq y \wedge \forall z ((\forall s M(z, s)) \leftrightarrow (z = x \vee z = y)))$, where $M(a, b)$ means that system a monitors remote server b **19. a)** $\forall x \forall y ((x < 0) \wedge (y < 0) \rightarrow (x + y < 0))$ **b)** $\neg \forall x \forall y ((x > 0) \wedge (y > 0) \rightarrow (x - y > 0))$ **c)** $\forall x \forall y (x^2 + y^2 \geq (x + y)^2)$ **d)** $\forall x \forall y (|xy| = |x||y|)$ **21.** $\forall x \exists a \exists b \exists c \exists d ((x > 0) \rightarrow x = a^2 + b^2 + c^2 + d^2)$, where the domain consists of all integers **23. a)** $\forall x \forall y ((x < 0) \wedge (y < 0) \rightarrow (xy > 0))$ **b)** $\forall x (x - x = 0)$ **c)** $\forall x \exists a \exists b (a \neq b \wedge \forall c (c^2 = x \leftrightarrow (c = a \vee c = b)))$ **d)** $\forall x ((x < 0) \rightarrow \neg \exists y (x = y^2))$ **25. a)** There is a multiplicative identity for the real numbers. **b)** The product of two negative real numbers is always a positive real number. **c)** There exist real numbers x and y such that x^2 exceeds y but x is less than y . **d)** The real numbers are closed under the operation of addition. **27. a)** True **b)** True **c)** True **d)** True **e)** True **f)** False **g)** False **h)** True **i)** False **29. a)** $P(1,1) \wedge P(1,2) \wedge P(1,3) \wedge P(2,1) \wedge P(2,2) \wedge P(2,3) \wedge P(3,1) \wedge P(3,2) \wedge P(3,3)$ **b)** $P(1,1) \vee P(1,2) \vee P(1,3) \vee P(2,1) \vee P(2,2) \vee P(2,3) \vee P(3,1) \vee P(3,2) \vee P(3,3)$ **c)** $(P(1,1) \wedge P(1,2) \wedge P(1,3)) \vee (P(2,1) \wedge P(2,2) \wedge P(2,3)) \vee (P(3,1) \wedge P(3,2) \wedge P(3,3))$ **d)** $(P(1,1) \vee P(2,1) \vee P(3,1)) \wedge (P(1,2) \vee P(2,2) \vee P(3,2)) \wedge (P(1,3) \vee P(2,3) \vee P(3,3))$ **31. a)** $\exists x \forall y \exists z \neg T(x, y, z)$ **b)** $\exists x \forall y \neg P(x, y) \wedge \exists x \forall y \neg Q(x, y)$ **c)** $\exists x \forall y (\neg P(x, y) \vee \forall z \neg R(x, y, z))$ **d)** $\exists x \forall y (P(x, y) \wedge \neg Q(x, y))$ **33. a)** $\exists x \exists y \neg P(x, y)$ **b)** $\exists y \forall x \neg P(x, y)$ **c)** $\exists y \exists x (\neg P(x, y) \wedge \neg Q(x, y))$ **d)** $(\forall x \forall y P(x, y)) \vee (\exists x \exists y \neg Q(x, y))$ **e)** $\exists x (\forall y \exists z \neg P(x, y, z) \vee \forall z \exists y \neg P(x, y, z))$ **35.** Any domain with four or more members makes the statement true; any domain with three or fewer members makes the statement false. **37. a)** There is someone in this class such that for every two different math courses, these are not the two and only two math courses this person has taken. **b)** Every person has either visited Libya or has not visited a country other than Libya. **c)** Someone has climbed every mountain in the Himalayas. **d)** There is someone who has neither been in a movie with Kevin Bacon nor has been in a movie with someone who has been in a movie with Kevin Bacon. **39. a)** $x = 2, y = -2$ **b)** $x = -4$ **c)** $x = 17, y = -1$ **41.** $\forall x \forall y \forall z ((x \cdot y) \cdot z = x \cdot (y \cdot z))$ **43.** $\forall m \forall b (m \neq 0 \rightarrow \exists x (mx + b = 0 \wedge \forall w (mw + b = 0 \rightarrow w = x)))$ **45. a)** True **b)** False **c)** True **47.** $\neg (\exists x \forall y P(x, y)) \leftrightarrow \forall x (\neg \forall y P(x, y)) \leftrightarrow \forall x \exists y \neg P(x, y)$ **49. a)** Suppose that $\forall x P(x) \wedge \exists x Q(x)$ is true. Then $P(x)$ is

S-8 Answers to Odd-Numbered Exercises

true for all x and there is an element y for which $Q(y)$ is true. Because $P(x) \wedge Q(y)$ is true for all x and there is a y for which $Q(y)$ is true, $\forall x \exists y (P(x) \wedge Q(y))$ is true. Conversely, suppose that the second proposition is true. Let x be an element in the domain. There is a y such that $Q(y)$ is true, so $\exists x Q(x)$ is true. Because $\forall x P(x)$ is also true, it follows that the first proposition is true. **b)** Suppose that $\forall x P(x) \vee \exists x Q(x)$ is true. Then either $P(x)$ is true for all x , or there exists a y for which $Q(y)$ is true. In the former case, $P(x) \vee Q(y)$ is true for all x , so $\forall x \exists y (P(x) \vee Q(y))$ is true. In the latter case, $Q(y)$ is true for a particular y , so $P(x) \vee Q(y)$ is true for all x and consequently $\forall x \exists y (P(x) \vee Q(y))$ is true. Conversely, suppose that the second proposition is true. If $P(x)$ is true for all x , then the first proposition is true. If not, $P(x)$ is false for some x , and for this x there must be a y such that $P(x) \vee Q(y)$ is true. Hence, $Q(y)$ must be true, so $\exists y Q(y)$ is true. It follows that the first proposition must hold. **51.** We will show how an expression can be put into prenex normal form (PNF) if subexpressions in it can be put into PNF. Then, working from the inside out, any expression can be put in PNF. (To formalize the argument, it is necessary to use the method of structural induction that will be discussed in Section 5.3.) By Exercise 45 of Section 1.4, we can assume that the proposition uses only \vee and \neg as logical connectives. Now note that any proposition with no quantifiers is already in PNF. (This is the basis case of the argument.) Now suppose that the proposition is of the form $Qx P(x)$, where Q is a quantifier. Because $P(x)$ is a shorter expression than the original proposition, we can put it into PNF. Then Qx followed by this PNF is again in PNF and is equivalent to the original proposition. Next, suppose that the proposition is of the form $\neg P$. If P is already in PNF, we slide the negation sign past all the quantifiers using the equivalences in Table 2 in Section 1.4. Finally, assume that proposition is of the form $P \vee Q$, where each of P and Q is in PNF. If only one of P and Q has quantifiers, then we can use Exercise 46 in Section 1.4 to bring the quantifier in front of both. If both P and Q have quantifiers, we can use Exercise 45 in Section 1.4, Exercise 48, or part (b) of Exercise 49 to rewrite $P \vee Q$ with two quantifiers preceding the disjunction of a proposition of the form $R \vee S$, and then put $R \vee S$ into PNF.

Section 1.6

- 1.** Modus ponens; valid; the conclusion is true, because the hypotheses are true. **3. a)** Addition **b)** Simplification **c)** Modus ponens **d)** Modus tollens **e)** Hypothetical syllogism **5.** Let w be “Randy works hard,” let d be “Randy is a dull boy,” and let j be “Randy will get the job.” The hypotheses are w , $w \rightarrow d$, and $d \rightarrow \neg j$. Using modus ponens and the first two hypotheses, d follows. Using modus ponens and the last hypothesis, $\neg j$, which is the desired conclusion, “Randy

will not get the job,” follows. **7.** Universal instantiation is used to conclude that “If Socrates is a man, then Socrates is mortal.” Modus ponens is then used to conclude that Socrates is mortal. **9. a)** Valid conclusions are “I did not take Tuesday off,” “I took Thursday off,” “It rained on Thursday.” **b)** “I did not eat spicy foods and it did not thunder” is a valid conclusion. **c)** “I am clever” is a valid conclusion. **d)** “Ralph is not a CS major” is a valid conclusion. **e)** “That you buy lots of stuff is good for the U.S. and is good for you” is a valid conclusion. **f)** “Mice gnaw their food” and “Rabbits are not rodents” are valid conclusions. **11.** Suppose that p_1, p_2, \dots, p_n are true. We want to establish that $q \rightarrow r$ is true. If q is false, then we are done, vacuously. Otherwise, q is true, so by the validity of the given argument form (that whenever p_1, p_2, \dots, p_n, q are true, then r must be true), we know that r is true. **13. a)** Let $c(x)$ be “ x is in this class,” $j(x)$ be “ x knows how to write programs in JAVA,” and $h(x)$ be “ x can get a high-paying job.” The premises are $c(\text{Doug})$, $j(\text{Doug})$, $\forall x (j(x) \rightarrow h(x))$. Using universal instantiation and the last premise, $j(\text{Doug}) \rightarrow h(\text{Doug})$ follows. Applying modus ponens to this conclusion and the second premise, $h(\text{Doug})$ follows. Using conjunction and the first premise, $c(\text{Doug}) \wedge h(\text{Doug})$ follows. Finally, using existential generalization, the desired conclusion, $\exists x (c(x) \wedge h(x))$ follows. **b)** Let $c(x)$ be “ x is in this class,” $w(x)$ be “ x enjoys whale watching,” and $p(x)$ be “ x cares about ocean pollution.” The premises are $\exists x (c(x) \wedge w(x))$ and $\forall x (w(x) \rightarrow p(x))$. From the first premise, $c(y) \wedge w(y)$ for a particular person y . Using simplification, $w(y)$ follows. Using the second premise and universal instantiation, $w(y) \rightarrow p(y)$ follows. Using modus ponens, $p(y)$ follows, and by conjunction, $c(y) \wedge p(y)$ follows. Finally, by existential generalization, the desired conclusion, $\exists x (c(x) \wedge p(x))$, follows. **c)** Let $c(x)$ be “ x is in this class,” $p(x)$ be “ x owns a PC,” and $w(x)$ be “ x can use a word-processing program.” The premises are $c(\text{Zeke})$, $\forall x (c(x) \rightarrow p(x))$, and $\forall x (p(x) \rightarrow w(x))$. Using the second premise and universal instantiation, $c(\text{Zeke}) \rightarrow p(\text{Zeke})$ follows. Using the first premise and modus ponens, $p(\text{Zeke})$ follows. Using the third premise and universal instantiation, $p(\text{Zeke}) \rightarrow w(\text{Zeke})$ follows. Finally, using modus ponens, $w(\text{Zeke})$, the desired conclusion, follows. **d)** Let $j(x)$ be “ x is in New Jersey,” $f(x)$ be “ x lives within 50 miles of the ocean,” and $s(x)$ be “ x has seen the ocean.” The premises are $\forall x (j(x) \rightarrow f(x))$ and $\exists x (j(x) \wedge \neg s(x))$. The second hypothesis and existential instantiation imply that $j(y) \wedge \neg s(y)$ for a particular person y . By simplification, $j(y)$ for this person y . Using universal instantiation and the first premise, $j(y) \rightarrow f(y)$, and by modus ponens, $f(y)$ follows. By simplification, $\neg s(y)$ follows from $j(y) \wedge \neg s(y)$. So $f(y) \wedge \neg s(y)$ follows by conjunction. Finally, the desired conclusion, $\exists x (f(x) \wedge \neg s(x))$, follows by existential generalization. **15. a)** Correct, using universal instantiation and modus ponens **b)** Invalid; fallacy of affirming the conclusion **c)** Invalid; fallacy of denying the hypothesis **d)** Correct, using universal instantiation and modus tollens **17.** We know that *some x* exists that makes

$H(x)$ true, but we cannot conclude that Lola is one such x . **19. a)** Fallacy of affirming the conclusion **b)** Fallacy of begging the question **c)** Valid argument using modus tollens **d)** Fallacy of denying the hypothesis **21.** By the second premise, there is some lion that does not drink coffee. Let Leo be such a creature. By simplification we know that Leo is a lion. By modus ponens we know from the first premise that Leo is fierce. Hence, Leo is fierce and does not drink coffee. By the definition of the existential quantifier, there exist fierce creatures that do not drink coffee, that is, some fierce creatures do not drink coffee. **23.** The error occurs in step (5), because we cannot assume, as is being done here, that the c that makes P true is the same as the c that makes Q true. **25.** We are given the premises $\forall x(P(x) \rightarrow Q(x))$ and $\neg Q(a)$. We want to show $\neg P(a)$. Suppose, to the contrary, that $\neg P(a)$ is not true. Then $P(a)$ is true. Therefore by universal modus ponens, we have $Q(a)$. But this contradicts the given premise $\neg Q(a)$. Therefore our supposition must have been wrong, and so $\neg P(a)$ is true, as desired.

27. Step

	Reason
1. $\forall x(P(x) \wedge R(x))$	Premise
2. $P(a) \wedge R(a)$	Universal instantiation from (1)
3. $P(a)$	Simplification from (2)
4. $\forall x(P(x) \rightarrow (Q(x) \wedge S(x)))$	Premise
5. $Q(a) \wedge S(a)$	Universal modus ponens from (3) and (4)
6. $S(a)$	Simplification from (5)
7. $R(a)$	Simplification from (2)
8. $R(a) \wedge S(a)$	Conjunction from (7) and (6)
9. $\forall x(R(x) \wedge S(x))$	Universal generalization from (5)

29. Step

	Reason
1. $\exists x \neg P(x)$	Premise
2. $\neg P(c)$	Existential instantiation from (1)
3. $\forall x(P(x) \vee Q(x))$	Premise
4. $P(c) \vee Q(c)$	Universal instantiation from (3)
5. $Q(c)$	Disjunctive syllogism from (4) and (2)
6. $\forall x(\neg Q(x) \vee S(x))$	Premise
7. $\neg Q(c) \vee S(c)$	Universal instantiation from (6)
8. $S(c)$	Disjunctive syllogism from (5) and (7)
9. $\forall x(R(x) \rightarrow \neg S(x))$	Premise
10. $R(c) \rightarrow \neg S(c)$	Universal instantiation from (9)
11. $\neg R(c)$	Modus tollens from (8) and (10)
12. $\exists x \neg R(x)$	Existential generalization from (11)

31. Let p be “It is raining”; let q be “Yvette has her umbrella”; let r be “Yvette gets wet.” Assumptions are $\neg p \vee q$, $\neg q \vee \neg r$, and $p \vee \neg r$. Resolution on the first two gives $\neg p \vee \neg r$. Resolution on this and the third assumption gives $\neg r$, as desired. **33.** Assume that this proposition is satisfiable. Using resolution on the first two clauses enables us to conclude $q \vee q$; in other words, we know that q has to be true. Using resolution on the last two clauses enables us to conclude $\neg q \vee \neg q$; in other

words, we know that $\neg q$ has to be true. This is a contradiction. So this proposition is not satisfiable. **35.** Valid

Section 1.7

1. Let $n = 2k + 1$ and $m = 2l + 1$ be odd integers. Then $n+m = 2(k+l+1)$ is even. **3.** Suppose that n is even. Then $n = 2k$ for some integer k . Therefore, $n^2 = (2k)^2 = 4k^2 = 2(2k^2)$. Because we have written n^2 as 2 times an integer, we conclude that n^2 is even. **5.** Direct proof: Suppose that $m+n$ and $n+p$ are even. Then $m+n = 2s$ for some integer s and $n+p = 2t$ for some integer t . If we add these, we get $m+p+2n = 2s+2t$. Subtracting $2n$ from both sides and factoring, we have $m+p = 2s+2t-2n = 2(s+t-n)$. Because we have written $m+p$ as 2 times an integer, we conclude that $m+p$ is even. **7.** Because n is odd, we can write $n = 2k+1$ for some integer k . Then $(k+1)^2 - k^2 = k^2 + 2k + 1 - k^2 = 2k + 1 = n$. **9.** Suppose that r is rational and i is irrational and $s = r+i$ is rational. Then by Example 7, $s+(-r) = i$ is rational, which is a contradiction. **11.** Because $\sqrt{2} \cdot \sqrt{2} = 2$ is rational and $\sqrt{2}$ is irrational, the product of two irrational numbers is not necessarily irrational. **13.** Proof by contraposition: If $1/x$ were rational, then by definition $1/x = p/q$ for some integers p and q with $q \neq 0$. Because $1/x$ cannot be 0 (if it were, then we'd have the contradiction $1 = x \cdot 0$ by multiplying both sides by x), we know that $p \neq 0$. Now $x = 1/(1/x) = 1/(p/q) = q/p$ by the usual rules of algebra and arithmetic. Hence, x can be written as the quotient of two integers with the denominator nonzero. Thus by definition, x is rational. **15.** Assume that it is not true that $x \geq 1$ or $y \geq 1$. Then $x < 1$ and $y < 1$. Adding these two inequalities, we obtain $x+y < 2$, which is the negation of $x+y \geq 2$. **17. a)** Assume that n is odd, so $n = 2k+1$ for some integer k . Then $n^3+5 = 2(4k^3+6k^2+3k+3)$. Because n^3+5 is two times some integer, it is even. **b)** Suppose that n^3+5 is odd and n is odd. Because n is odd and the product of two odd numbers is odd, it follows that n^2 is odd and then that n^3 is odd. But then $5 = (n^3+5) - n^3$ would have to be even because it is the difference of two odd numbers. Therefore, the supposition that n^3+5 and n were both odd is wrong. **19.** The proposition is vacuously true because 0 is not a positive integer. Vacuous proof. **21.** $P(1)$ is true because $(a+b)^1 = a+b \geq a^1+b^1 = a+b$. Direct proof. **23.** If we chose 9 or fewer days on each day of the week, this would account for at most $9 \cdot 7 = 63$ days. But we chose 64 days. This contradiction shows that at least 10 of the days we chose must be on the same day of the week. **25.** Suppose by way of contradiction that a/b is a rational root, where a and b are integers and this fraction is in lowest terms (that is, a and b have no common divisor greater than 1). Plug this proposed root into the equation to obtain $a^3/b^3 + a/b + 1 = 0$. Multiply through by b^3 to obtain $a^3 + ab^2 + b^3 = 0$. If a and b are both odd, then the left-hand side is the sum of three odd numbers and therefore must be odd. If a is odd and b is even, then the left-hand side is odd + even + even, which is again odd. Similarly, if a is even and b is odd, then the left-hand

side is even + even + odd, which is again odd. Because the fraction a/b is in simplest terms, it cannot happen that both a and b are even. Thus in all cases, the left-hand side is odd, and therefore cannot equal 0. This contradiction shows that no such root exists. **27.** First, assume that n is odd, so that $n = 2k + 1$ for some integer k . Then $5n + 6 = 5(2k + 1) + 6 = 10k + 11 = 2(5k + 5) + 1$. Hence, $5n + 6$ is odd. To prove the converse, suppose that n is even, so that $n = 2k$ for some integer k . Then $5n + 6 = 10k + 6 = 2(5k + 3)$, so $5n + 6$ is even. Hence, n is odd if and only if $5n + 6$ is odd. **29.** This proposition is true. Suppose that m is neither 1 nor -1 . Then mn has a factor m larger than 1. On the other hand, $mn = 1$, and 1 has no such factor. Hence, $m = 1$ or $m = -1$. In the first case $n = 1$, and in the second case $n = -1$, because $n = 1/m$. **31.** We prove that all these are equivalent to x being even. If x is even, then $x = 2k$ for some integer k . Therefore $3x + 2 = 3 \cdot 2k + 2 = 6k + 2 = 2(3k + 1)$, which is even, because it has been written in the form $2t$, where $t = 3k + 1$. Similarly, $x + 5 = 2k + 5 = 2k + 4 + 1 = 2(k + 2) + 1$, so $x + 5$ is odd; and $x^2 = (2k)^2 = 2(2k^2)$, so x^2 is even. For the converses, we will use a proof by contraposition. So assume that x is not even; thus x is odd and we can write $x = 2k + 1$ for some integer k . Then $3x + 2 = 3(2k + 1) + 2 = 6k + 5 = 2(3k + 2) + 1$, which is odd (i.e., not even), because it has been written in the form $2t + 1$, where $t = 3k + 2$. Similarly, $x + 5 = 2k + 1 + 5 = 2(k + 3)$, so $x + 5$ is even (i.e., not odd). That x^2 is odd was already proved in Example 1. **33.** We give proofs by contraposition of $(i) \rightarrow (ii)$, $(ii) \rightarrow (i)$, $(i) \rightarrow (iii)$, and $(iii) \rightarrow (i)$. For the first of these, suppose that $3x + 2$ is rational, namely, equal to p/q for some integers p and q with $q \neq 0$. Then we can write $x = ((p/q) - 2)/3 = (p - 2q)/(3q)$, where $3q \neq 0$. This shows that x is rational. For the second conditional statement, suppose that x is rational, namely, equal to p/q for some integers p and q with $q \neq 0$. Then we can write $3x + 2 = (3p + 2q)/q$, where $q \neq 0$. This shows that $3x + 2$ is rational. For the third conditional statement, suppose that $x/2$ is rational, namely, equal to p/q for some integers p and q with $q \neq 0$. Then we can write $x = 2p/q$, where $q \neq 0$. This shows that x is rational. And for the fourth conditional statement, suppose that x is rational, namely, equal to p/q for some integers p and q with $q \neq 0$. Then we can write $x/2 = p/(2q)$, where $2q \neq 0$. This shows that $x/2$ is rational. **35.** No **37.** Suppose that $p_1 \rightarrow p_4 \rightarrow p_2 \rightarrow p_5 \rightarrow p_3 \rightarrow p_1$. To prove that one of these propositions implies any of the others, just use hypothetical syllogism repeatedly. **39.** We will give a proof by contradiction. Suppose that a_1, a_2, \dots, a_n are all less than A , where A is the average of these numbers. Then $a_1 + a_2 + \dots + a_n < nA$. Dividing both sides by n shows that $A = (a_1 + a_2 + \dots + a_n)/n < A$, which is a contradiction. **41.** We will show that the four statements are equivalent by showing that (i) implies (ii) , (ii) implies (iii) , (iii) implies (iv) , and (iv) implies (i) . First, assume that n is even. Then $n = 2k$ for some integer k . Then $n + 1 = 2k + 1$, so $n + 1$ is odd. This shows that (i) implies (ii) . Next, suppose that $n + 1$ is odd, so $n + 1 = 2k + 1$ for some integer k . Then $3n + 1 = 2n + (n + 1) = 2(n + k) + 1$, which

shows that $3n + 1$ is odd, showing that (ii) implies (iii) . Next, suppose that $3n + 1$ is odd, so $3n + 1 = 2k + 1$ for some integer k . Then $3n = (2k + 1) - 1 = 2k$, so $3n$ is even. This shows that (iii) implies (iv) . Finally, suppose that n is not even. Then n is odd, so $n = 2k + 1$ for some integer k . Then $3n = 3(2k + 1) = 6k + 3 = 2(3k + 1) + 1$, so $3n$ is odd. This completes a proof by contraposition that (iv) implies (i) .

Section 1.8

1. $1^2 + 1 = 2 = 2^1$; $2^2 + 1 = 5 \geq 4 = 2^2$; $3^2 + 1 = 10 \geq 8 = 2^3$; $4^2 + 1 = 17 \geq 16 = 2^4$ **3.** If $x \leq y$, then $\max(x, y) + \min(x, y) = y + x = x + y$. If $x \geq y$, then $\max(x, y) + \min(x, y) = x + y$. Because these are the only two cases, the equality always holds. **5.** Because $|x - y| = |y - x|$, the values of x and y are interchangeable. Therefore, without loss of generality, we can assume that $x \geq y$. Then $(x + y - (x - y))/2 = (x + y - x + y)/2 = 2y/2 = y = \min(x, y)$. Similarly, $(x + y + (x - y))/2 = (x + y + x - y)/2 = 2x/2 = x = \max(x, y)$. **7.** There are four cases. *Case 1:* $x \geq 0$ and $y \geq 0$. Then $|x| + |y| = x + y = |x + y|$. *Case 2:* $x < 0$ and $y < 0$. Then $|x| + |y| = -x + (-y) = -(x + y) = |x + y|$ because $x + y < 0$. *Case 3:* $x \geq 0$ and $y < 0$. Then $|x| + |y| = x + (-y) = |x + (-y)| = |x + y|$ because $x + y < 0$. *Case 4:* $x < 0$ and $y \geq 0$. Identical to Case 3 with the roles of x and y reversed. **9.** 10,001, 10,002, ..., 10,100 are all nonsquares, because $100^2 = 10,000$ and $101^2 = 10,201$; constructive. **11.** $8 = 2^3$ and $9 = 3^2$ **13.** Let $x = 2$ and $y = \sqrt{2}$. If $x^y = 2\sqrt{2}$ is irrational, we are done. If not, then let $x = 2\sqrt{2}$ and $y = \sqrt{2}/4$. Then $x^y = (2\sqrt{2})^{\sqrt{2}/4} = 2^{\sqrt{2} \cdot (\sqrt{2})/4} = 2^{1/2} = \sqrt{2}$. **15. a)** This statement asserts the existence of x with a certain property. If we let $y = x$, then we see that $P(x)$ is true. If y is anything other than x , then $P(x)$ is not true. Thus, x is the unique element that makes P true. **b)** The first clause here says that there is an element that makes P true. The second clause says that whenever two elements both make P true, they are in fact the same element. Together these say that P is satisfied by exactly one element. **c)** This statement asserts the existence of an x that makes P true and has the further property that whenever we find an element that makes P true, that element is x . In other words, x is the unique element that makes P true. **17.** The equation $|a - c| = |b - c|$ is equivalent to the disjunction of two equations: $a - c = b - c$ or $a - c = -b + c$. The first of these is equivalent to $a = b$, which contradicts the assumptions made in this problem, so the original equation is equivalent to $a - c = -b + c$. By adding $b + c$ to both sides and dividing by 2, we see that this equation is equivalent to $c = (a + b)/2$. Thus, there is a

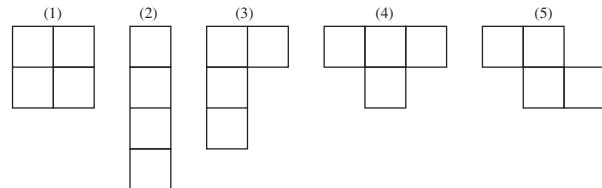
unique solution. Furthermore, this c is an integer, because the sum of the odd integers a and b is even. **19.** We are being asked to solve $n = (k - 2) + (k + 3)$ for k . Using the usual, reversible, rules of algebra, we see that this equation is equivalent to $k = (n - 1)/2$. In other words, this is the one and only value of k that makes our equation true. Because n is odd, $n - 1$ is even, so k is an integer. **21.** If x is itself an integer, then we can take $n = x$ and $\epsilon = 0$. No other solution is possible in this case, because if the integer n is greater than x , then n is at least $x + 1$, which would make $\epsilon \geq 1$. If x is not an integer, then round it up to the next integer, and call that integer n . Let $\epsilon = n - x$. Clearly $0 \leq \epsilon < 1$; this is the only ϵ that will work with this n , and n cannot be any larger, because ϵ is constrained to be less than 1. **23.** The harmonic mean of distinct positive real numbers x and y is always less than their geometric mean. To prove $2xy/(x + y) < \sqrt{xy}$, multiply both sides by $(x + y)/(2\sqrt{xy})$ to obtain the equivalent inequality $\sqrt{xy} < (x + y)/2$, which is proved in Example 14. **25.** The parity (oddness or evenness) of the sum of the numbers written on the board never changes, because $j + k$ and $|j - k|$ have the same parity (and at each step we reduce the sum by $j + k$ but increase it by $|j - k|$). Therefore the integer at the end of the process must have the same parity as $1 + 2 + \dots + (2n) = n(2n + 1)$, which is odd because n is odd. **27.** Without loss of generality we can assume that n is nonnegative, because the fourth power of an integer and the fourth power of its negative are the same. We divide an arbitrary positive integer n by 10, obtaining a quotient k and remainder l , whence $n = 10k + l$, and l is an integer between 0 and 9, inclusive. Then we compute n^4 in each of these 10 cases. We get the following values, where X is some integer that is a multiple of 10, whose exact value we do not care about. $(10k + 0)^4 = 10,000k^4 = 10,000k^4 + 0$, $(10k + 1)^4 = 10,000k^4 + X \cdot k^3 + X \cdot k^2 + X \cdot k + 1$, $(10k + 2)^4 = 10,000k^4 + X \cdot k^3 + X \cdot k^2 + X \cdot k + 16$, $(10k + 3)^4 = 10,000k^4 + X \cdot k^3 + X \cdot k^2 + X \cdot k + 81$, $(10k + 4)^4 = 10,000k^4 + X \cdot k^3 + X \cdot k^2 + X \cdot k + 256$, $(10k + 5)^4 = 10,000k^4 + X \cdot k^3 + X \cdot k^2 + X \cdot k + 625$, $(10k + 6)^4 = 10,000k^4 + X \cdot k^3 + X \cdot k^2 + X \cdot k + 1296$, $(10k + 7)^4 = 10,000k^4 + X \cdot k^3 + X \cdot k^2 + X \cdot k + 2401$, $(10k + 8)^4 = 10,000k^4 + X \cdot k^3 + X \cdot k^2 + X \cdot k + 4096$, $(10k + 9)^4 = 10,000k^4 + X \cdot k^3 + X \cdot k^2 + X \cdot k + 6561$. Because each coefficient indicated by X is a multiple of 10, the corresponding term has no effect on the ones digit of the answer. Therefore the ones digits are 0, 1, 6, 1, 6, 5, 6, 1, 6, 1, respectively, so it is always a 0, 1, 5, or 6. **29.** Because $n^3 > 100$ for all $n > 4$, we need only note that $n = 1$, $n = 2$, $n = 3$, and $n = 4$ do not satisfy $n^2 + n^3 = 100$. **31.** Because $5^4 = 625$, both x and y must be less than 5. Then $x^4 + y^4 \leq 4^4 + 4^4 = 512 < 625$. **33.** If it is not true that $a \leq \sqrt[3]{n}$, $b \leq \sqrt[3]{n}$, or $c \leq \sqrt[3]{n}$, then $a > \sqrt[3]{n}$, $b > \sqrt[3]{n}$, and $c > \sqrt[3]{n}$. Multiplying these inequalities of positive numbers together we obtain $abc < (\sqrt[3]{n})^3 = n$, which implies the negation of our hypothesis that $n = abc$. **35.** By finding a common denominator, we can assume that the given rational numbers are a/b and c/b , where b is a pos-

itive integer and a and c are integers with $a < c$. In particular, $(a + 1)/b \leq c/b$. Thus, $x = (a + \frac{1}{2}\sqrt{2})/b$ is between the two given rational numbers, because $0 < \sqrt{2} < 2$. Furthermore, x is irrational, because if x were rational, then $2(bx - a) = \sqrt{2}$ would be as well, in violation of Example 10 in Section 1.7.

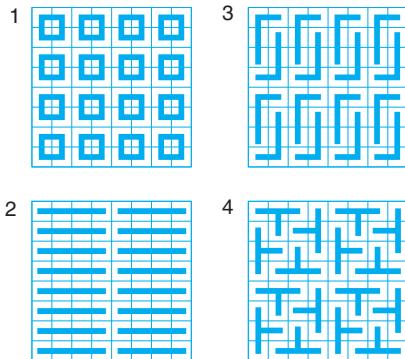
37. a) Without loss of generality, we can assume that the x sequence is already sorted into nondecreasing order, because we can relabel the indices. There are only a finite number of possible orderings for the y sequence, so if we can show that we can increase the sum (or at least keep it the same) whenever we find y_i and y_j that are out of order (i.e., $i < j$ but $y_i > y_j$) by switching them, then we will have shown that the sum is largest when the y sequence is in nondecreasing order. Indeed, if we perform the swap, then we have added $x_i y_j + x_j y_i$ to the sum and subtracted $x_i y_i + x_j y_j$. The net effect is to have added $x_i y_j + x_j y_i - x_i y_i - x_j y_j = (x_j - x_i)(y_i - y_j)$, which is nonnegative by our ordering assumptions. **b)** Similar to part (a) **39. a)** $6 \rightarrow 3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$ **b)** $7 \rightarrow 22 \rightarrow 11 \rightarrow 34 \rightarrow 17 \rightarrow 52 \rightarrow 26 \rightarrow 13 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$ **c)** $17 \rightarrow 52 \rightarrow 26 \rightarrow 13 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$ **d)** $21 \rightarrow 64 \rightarrow 32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$ **41.** Without loss of generality, assume that the upper left and upper right corners of the board are removed. Place three dominoes horizontally to fill the remaining portion of the first row, and fill each of the other seven rows with four horizontal dominoes.

43. Because there is an even number of squares in all, either there is an even number of squares in each row or there is an even number of squares in each column. In the former case, tile the board in the obvious way by placing the dominoes horizontally, and in the latter case, tile the board in the obvious way by placing the dominoes vertically. **45.** We can rotate the board if necessary to make the removed squares be 1 and 16. Square 2 must be covered by a domino. If that domino is placed to cover squares 2 and 6, then the following domino placements are forced in succession: 5-9, 13-14, and 10-11, at which point there is no way to cover square 15. Otherwise, square 2 must be covered by a domino placed at 2-3. Then the following domino placements are forced: 4-8, 11-12, 6-7, 5-9, and 10-14, and again there is no way to cover square 15. **47.** Remove the two black squares adjacent to a white corner, and remove two white squares other than that corner. Then no domino can cover that white corner.

49. a)



b) The picture shows tilings for the first four patterns.



To show that pattern 5 cannot tile the checkerboard, label the squares from 1 to 64, one row at a time from the top, from left to right in each row. Thus, square 1 is the upper left corner, and square 64 is the lower right. Suppose we did have a tiling. By symmetry and without loss of generality, we may suppose that the tile is positioned in the upper left corner, covering squares 1, 2, 10, and 11. This forces a tile to be adjacent to it on the right, covering squares 3, 4, 12, and 13. Continue in this manner and we are forced to have a tile covering squares 6, 7, 15, and 16. This makes it impossible to cover square 8. Thus, no tiling is possible.

Supplementary Exercises

1. a) $q \rightarrow p$ **b)** $q \wedge p$ **c)** $\neg q \vee \neg p$ **d)** $q \leftrightarrow p$ **3. a)** The proposition cannot be false unless $\neg p$ is false, so p is true. If p is true and q is true, then $\neg q \wedge (p \rightarrow q)$ is false, so the conditional statement is true. If p is true and q is false, then $p \rightarrow q$ is false, so $\neg q \wedge (p \rightarrow q)$ is false and the conditional statement is true. **b)** The proposition cannot be false unless q is false. If q is false and p is true, then $(p \vee q) \wedge \neg p$ is false, and the conditional statement is true. If q is false and p is false, then $(p \vee q) \wedge \neg p$ is false, and the conditional statement is true. **5. a)** $\neg q \rightarrow \neg p; p \rightarrow q; \neg p \rightarrow \neg q$ **b)** $(p \wedge q \wedge r \wedge \neg s) \vee (p \wedge q \wedge \neg r \wedge s) \vee (p \wedge \neg q \wedge r \wedge s) \vee (\neg p \wedge q \wedge r \wedge s)$ **c)** Translating these statements into symbols, using the obvious letters, we have $\neg t \rightarrow \neg g$, $\neg g \rightarrow \neg q$, $r \rightarrow q$, and $\neg t \wedge r$. Assume the statements are consistent. The fourth statement tells us that $\neg t$ must be true. Therefore by modus ponens with the first statement, we know that $\neg g$ is true, hence (from the second statement), that $\neg q$ is true. Also, the fourth statement tells us that r must be true, and so again modus ponens (third statement) makes q true. This is a contradiction: $q \wedge \neg q$. Thus the statements are inconsistent. **11.** Reject-accept-reject-accept, accept-accept-accept-accept, accept-accept-reject-accept, reject-reject-reject-reject, reject-reject-accept-reject, and reject-accept-accept-accept **13.** Aaron is a knave and Crystal is a knight; it cannot be determined what Bohan is. **15.** Brenda **17.** The premises cannot both be true, because

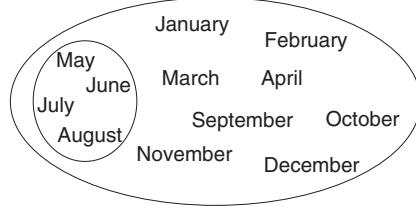
they are contradictory. Therefore it is (vacuously) true that whenever all the premises are true, the conclusion is also true, which by definition makes this a valid argument. Because the premises are not both true, we cannot conclude that the conclusion is true. **19.** Use the same propositions as were given in Section 1.3 for a 9×9 Sudoku puzzle, with the variables indexed from 1 to 16, instead of from 1 to 9, and with a similar change for the propositions for the 4×4 blocks: $\bigwedge_{r=0}^3 \bigwedge_{s=0}^3 \bigwedge_{n=1}^{16} \bigvee_{i=1}^4 \bigvee_{j=1}^4 p(4r+i, 4s+j, n)$. **21. a)** F **b)** T **c)** F **d)** T **e)** F **f)** T **23.** Many answers are possible. One example is United States senators. **25.** $\forall x \exists y \exists z (y \neq z \wedge \forall w (P(w, x) \leftrightarrow (w = y \vee w = z)))$ **27. a)** $\neg \exists x P(x)$ **b)** $\exists x (P(x) \wedge \forall y (P(y) \rightarrow y = x))$ **c)** $\exists x_1 \exists x_2 (P(x_1) \wedge P(x_2) \wedge x_1 \neq x_2 \wedge \forall y (P(y) \rightarrow (y = x_1 \vee y = x_2)))$ **d)** $\exists x_1 \exists x_2 \exists x_3 (P(x_1) \wedge P(x_2) \wedge P(x_3) \wedge x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_2 \neq x_3 \wedge \forall y (P(y) \rightarrow (y = x_1 \vee y = x_2 \vee y = x_3)))$ **29.** Suppose that $\exists x (P(x) \rightarrow Q(x))$ is true. Then either $Q(x_0)$ is true for some x_0 , in which case $\forall x P(x) \rightarrow \exists x Q(x)$ is true; or $P(x_0)$ is false for some x_0 , in which case $\forall x P(x) \rightarrow \exists x Q(x)$ is true. Conversely, suppose that $\exists x (P(x) \rightarrow Q(x))$ is false. That means that $\forall x (P(x) \wedge \neg Q(x))$ is true, which implies $\forall x P(x)$ and $\forall x (\neg Q(x))$. This latter proposition is equivalent to $\neg \exists x Q(x)$. Thus, $\forall x P(x) \rightarrow \exists x Q(x)$ is false. **31.** No **33.** $\forall x \forall z \exists y T(x, y, z)$, where $T(x, y, z)$ is the statement that student x has taken class y in department z , where the domains are the set of students in the class, the set of courses at this university, and the set of departments in the school of mathematical sciences **35.** $\exists! x \exists! y T(x, y)$ and $\exists x \forall z ((\exists y \forall w (T(z, w) \leftrightarrow w = y)) \leftrightarrow z = x)$, where $T(x, y)$ means that student x has taken class y and the domain is all students in this class **37.** $P(a) \rightarrow Q(a)$ and $Q(a) \rightarrow R(a)$ by universal instantiation; then $\neg Q(a)$ by modus tollens and $\neg P(a)$ by modus tollens **39.** We give a proof by contraposition and show that if \sqrt{x} is rational, then x is rational, assuming throughout that $x \geq 0$. Suppose that $\sqrt{x} = p/q$ is rational, $q \neq 0$. Then $x = (\sqrt{x})^2 = p^2/q^2$ is also rational (q^2 is again nonzero). **41.** We can give a constructive proof by letting $m = 10^{500} + 1$. Then $m^2 = (10^{500} + 1)^2 > (10^{500})^2 = 10^{1000}$. **43.** 23 cannot be written as the sum of eight cubes. **45.** 223 cannot be written as the sum of 36 fifth powers.

CHAPTER 2

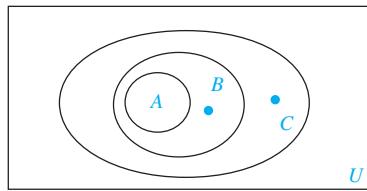
Section 2.1

- 1. a)** $\{-1, 1\}$ **b)** $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ **c)** $\{0, 1, 4, 9, 16, 25, 36, 49, 64, 81\}$ **d)** \emptyset **3. a)** The first is a subset of the second, but the second is not a subset of the first. **b)** Neither is a subset of the other. **c)** The first is a subset of the second, but the second is not a subset of the first. **5. a)** Yes **b)** No **c)** No **7. a)** Yes **b)** No **c)** Yes **d)** No **e)** No **f)** No **9. a)** False **b)** False **c)** False **d)** True **e)** False **f)** False **g)** True **11. a)** True **b)** True **c)** False **d)** True **e)** True **f)** False

13.



15. The dots in certain regions indicate that those regions are not empty.



17. Suppose that $x \in A$. Because $A \subseteq B$, this implies that $x \in B$. Because $B \subseteq C$, we see that $x \in C$. Because $x \in A$ implies that $x \in C$, it follows that $A \subseteq C$. 19. a) 1 b) 1 c) 2 d) 3 21. a) $\{\emptyset, \{a\}\}$ b) $\{\emptyset, \{a\}, \{b\}, \{a, b\}\}$ c) $\{\emptyset, \{\emptyset\}, \{\{\emptyset\}\}, \{\emptyset, \{\emptyset\}\}\}$ 23. a) 8 b) 16 c) 2 25. For the “if” part, given $A \subseteq B$, we want to show that that $\mathcal{P}(A) \subseteq \mathcal{P}(B)$, i.e., if $C \subseteq A$ then $C \subseteq B$. But this follows directly from Exercise 17. For the “only if” part, given that $\mathcal{P}(A) \subseteq \mathcal{P}(B)$, we want to show that $A \subseteq B$. Suppose $a \in A$. Then $\{a\} \subseteq A$, so $\{a\} \in \mathcal{P}(A)$. Since $\mathcal{P}(A) \subseteq \mathcal{P}(B)$, it follows that $\{a\} \in \mathcal{P}(B)$, which means that $\{a\} \subseteq B$. But this implies $a \in B$, as desired. 27. a) $\{(a, y), (b, y), (c, y), (d, y), (a, z), (b, z), (c, z), (d, z)\}$ b) $\{(y, a), (y, b), (y, c), (y, d), (z, a), (z, b), (z, c), (z, d)\}$ 29. The set of triples (a, b, c) , where a is an airline and b and c are cities. A useful subset of this set is the set of triples (a, b, c) for which a flies between b and c . 31. $\emptyset \times A = \{(x, y) \mid x \in \emptyset \text{ and } y \in A\} = \emptyset = \{(x, y) \mid x \in A \text{ and } y \in \emptyset\} = A \times \emptyset$ 33. a) $\{(0, 0), (0, 1), (0, 3), (1, 0), (1, 1), (1, 3), (3, 0), (3, 1), (3, 3)\}$ b) $\{(1, 1), (1, 2), (1, a), (1, b), (2, 1), (2, 2), (2, a), (2, b), (a, 1), (a, 2), (a, a), (a, b), (b, 1), (b, 2), (b, a), (b, b)\}$ 35. m^n 37. m^n 39. The elements of $A \times B \times C$ consist of 3-tuples (a, b, c) , where $a \in A$, $b \in B$, and $c \in C$, whereas the elements of $(A \times B) \times C$ look like $((a, b), c)$ —ordered pairs, the first coordinate of which is again an ordered pair. 41. a) The square of a real number is never -1 . True b) There exists an integer whose square is 2 . False c) The square of every integer is positive. False d) There is a real number equal to its own square. True 43. a) $\{-1, 0, 1\}$ b) $\mathbf{Z} - \{0, 1\}$ c) \emptyset 45. We must show that $\{\{a\}, \{a, b\}\} = \{\{c\}, \{c, d\}\}$ if and only if $a = c$ and $b = d$. The “if” part is immediate. So assume these two sets are equal. First, consider the case when $a \neq b$. Then $\{\{a\}, \{a, b\}\}$ contains exactly two elements, one of which contains one element. Thus, $\{\{c\}, \{c, d\}\}$ must have the same property, so $c \neq d$ and $\{c\}$ is the element containing exactly one element. Hence, $\{a\} = \{c\}$, which implies that $a = c$. Also, the two-element sets $\{a, b\}$ and $\{c, d\}$ must be equal. Because $a = c$ and $a \neq b$, it follows that $b = d$.

Second, suppose that $a = b$. Then $\{\{a\}, \{a, b\}\} = \{\{a\}\}$, a set with one element. Hence, $\{\{c\}, \{c, d\}\}$ has only one element, which can happen only when $c = d$, and the set is $\{\{c\}\}$. It then follows that $a = c$ and $b = d$. 47. Let $S = \{a_1, a_2, \dots, a_n\}$. Represent each subset of S with a bit string of length n , where the i th bit is 1 if and only if $a_i \in S$. To generate all subsets of S , list all 2^n bit strings of length n (for instance, in increasing order), and write down the corresponding subsets.

Section 2.2

1. a) The set of students who live within one mile of school and walk to classes b) The set of students who live within one mile of school or walk to classes (or do both) c) The set of students who live within one mile of school but do not walk to classes d) The set of students who walk to classes but live more than one mile away from school 3. a) $\{0, 1, 2, 3, 4, 5, 6\}$ b) $\{3\}$ c) $\{1, 2, 4, 5\}$ d) $\{0, 6\}$ 5. $\overline{A} = \{x \mid \neg(x \in A)\} = \{x \mid \neg(\neg x \in A)\} = \{x \mid x \in A\} = A$ 7. a) $A \cup U = \{x \mid x \in A \vee x \in U\} = \{x \mid x \in A \vee \mathbf{T}\} = \{x \mid \mathbf{T}\} = U$ b) $A \cap \emptyset = \{x \mid x \in A \wedge x \in \emptyset\} = \{x \mid x \in A \wedge \mathbf{F}\} = \{x \mid \mathbf{F}\} = \emptyset$ 9. a) $A \cup \overline{A} = \{x \mid x \in A \vee x \notin A\} = U$ b) $A \cap \overline{A} = \{x \mid x \in A \wedge x \notin A\} = \emptyset$ 11. a) $A \cup B = \{x \mid x \in A \vee x \in B\} = \{x \mid x \in B \vee x \in A\} = B \cup A$ b) $A \cap B = \{x \mid x \in A \wedge x \in B\} = \{x \mid x \in B \wedge x \in A\} = B \cap A$ 13. Suppose $x \in A \cap (A \cup B)$. Then $x \in A$ and $x \in A \cup B$ by the definition of intersection. Because $x \in A$, we have proved that the left-hand side is a subset of the right-hand side. Conversely, let $x \in A$. Then by the definition of union, $x \in A \cup B$ as well. Therefore $x \in A \cap (A \cup B)$ by the definition of intersection, so the right-hand side is a subset of the left-hand side. 15. a) $x \in \overline{A \cup B} \equiv x \notin A \cup B \equiv \neg(x \in A \vee x \in B) \equiv \neg(x \in A) \wedge \neg(x \in B) \equiv x \notin A \wedge x \notin B \equiv x \in \overline{A} \wedge x \in \overline{B} \equiv x \in \overline{A} \cap \overline{B}$

b)	A	B	$A \cup B$	$\overline{A \cup B}$	\overline{A}	\overline{B}	$\overline{A} \cap \overline{B}$
	1	1	1	0	0	0	0
	1	0	1	0	0	1	0
	0	1	1	0	1	0	0
	0	0	0	1	1	1	1

17. a) $x \in \overline{A \cap B \cap C} \equiv x \notin A \cap B \cap C \equiv x \notin A \vee x \notin B \vee x \notin C \equiv x \in \overline{A} \vee x \in \overline{B} \vee x \in \overline{C} \equiv x \in \overline{A} \cup \overline{B} \cup \overline{C}$

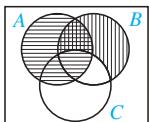
b)	A	B	C	$A \cap B \cap C$	$\overline{A \cap B \cap C}$	\overline{A}	\overline{B}	\overline{C}	$\overline{A} \cup \overline{B} \cup \overline{C}$
	1	1	1	1	0	0	0	0	0
	1	1	0	0	1	0	0	1	1
	1	0	1	0	1	0	1	0	1
	1	0	0	0	1	0	1	1	1
	0	1	1	0	1	1	0	0	1
	0	1	0	0	1	1	0	1	1
	0	0	1	0	1	1	1	0	1
	0	0	0	0	1	1	1	1	1

- 19. a)** Both sides equal $\{x \mid x \in A \wedge x \notin B\}$. **b)** $A = A \cap U = A \cap (B \cup \overline{B}) = (A \cap B) \cup (A \cap \overline{B})$

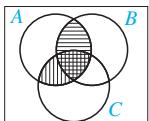
21. $x \in A \cup (B \cup C) \equiv (x \in A) \vee (x \in (B \cup C)) \equiv (x \in A) \vee (x \in B \vee x \in C) \equiv (x \in A \vee x \in B) \vee (x \in C) \equiv x \in (A \cup B) \cup C$

23. $x \in A \cup (B \cap C) \equiv (x \in A) \vee (x \in (B \cap C)) \equiv (x \in A) \vee (x \in B \wedge x \in C) \equiv (x \in A \vee x \in B) \wedge (x \in A \vee x \in C) \equiv x \in (A \cup B) \cap (A \cup C)$

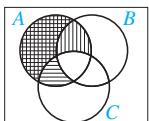
25. a) $\{4, 6\}$ **b)** $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ **c)** $\{4, 5, 6, 8, 10\}$
d) $\{0, 2, 4, 5, 6, 7, 8, 9, 10\}$ **27. a)** The double-shaded portion is the desired set.



- b)** The desired set is the entire shaded portion.



- c) The desired set is the entire shaded portion.



- 29.** a) $B \subseteq A$ b) $A \subseteq B$ c) $A \cap B = \emptyset$ d) Nothing, because this is always true e) $A = B$

31. $A \subseteq B \equiv \forall x(x \in A \rightarrow x \in B) \equiv \forall x(x \notin B \rightarrow x \notin A) \equiv \forall x(x \in \bar{B} \rightarrow x \in A) \equiv \bar{B} \subseteq \bar{A}$

33. The set of students who are computer science majors but not mathematics majors or who are mathematics majors but not computer science majors

35. An element is in $(A \cup B) - (A \cap B)$ if it is in the union of A and B but not in the intersection of A and B , which means that it is in either A or B but not in both A and B . This is exactly what it means for an element to belong to $A \oplus B$.

37. a) $A \oplus A = (A - A) \cup (A - A) = \emptyset \cup \emptyset = \emptyset$

b) $A \oplus \emptyset = (A - \emptyset) \cup (\emptyset - A) = A \cup \emptyset = A$

c) $A \oplus U = (A - U) \cup (U - A) = \emptyset \cup \bar{A} = \bar{A}$

d) $A \oplus \bar{A} = (A - \bar{A}) \cup (\bar{A} - A) = A \cup \bar{A} = U$

39. $B = \emptyset$

41. Yes

43. Yes

45. If $A \cup B$ were finite, then it would have n elements for some natural number n . But A already has more than n elements, because it is infinite, and $A \cup B$ has all the elements that A has, so $A \cup B$ has more than n elements. This contradiction shows that $A \cup B$ must be infinite.

47. a) $\{1, 2, 3, \dots, n\}$ b) $\{1\}$

49. a) A_n b) $\{0, 1\}$

51. a) $\mathbf{Z}, \{-1, 0, 1\}$ b) $\mathbf{Z} - \{0\}, \emptyset$

c) $\mathbf{R}, [-1, 1]$ d) $[1, \infty), \emptyset$

53. a) $\{1, 2, 3, 4, 7, 8, 9, 10\}$

b) $\{2, 4, 5, 6, 7\}$

c) $\{1, 10\}$

55. The bit in the i th position of the bit string of the difference of two sets is 1 if the i th bit of the first string is 1 and the i th bit of the second string is 0, and is 0 otherwise.

57. a) $11\ 1110\ 0000\ 0000\ 0000\ 0000\ 0000 \vee 01\ 1100\ 1000\ 0000\ 0100\ 0101\ 0000 = 11\ 1110\ 1000\ 0000\ 0100\ 0101\ 0000$, representing $\{a, b, c, d, e, g, p, t, v\}$

- b)** $11\ 1110\ 0000\ 0000\ 0000\ 0000 \wedge 01\ 1100\ 1000\ 0000$
 $0100\ 0101\ 0000 = 01\ 1100\ 0000\ 0000\ 0000\ 0000\ 0000$, representing $\{b, c, d\}$ **c)** $(11\ 1110\ 0000\ 0000\ 0000\ 0000 \vee 00\ 0110\ 0110\ 0001\ 1000\ 0110\ 0110) \wedge (01\ 1100\ 1000\ 0000\ 0100\ 0101\ 0000 \vee 00\ 1010\ 0010\ 0000\ 1000\ 0010\ 0111) = 11\ 1110\ 0110\ 0001\ 1000\ 0110\ 0110 \wedge 01\ 1110\ 1010\ 0000\ 1100\ 0111\ 0111 = 01\ 1110\ 0010\ 0000\ 1000\ 0110\ 0110$, representing $\{b, c, d, e, i, o, t, u, x, y\}$ **d)** $11\ 1110\ 0000\ 0000\ 0000\ 0000 \vee 01\ 1100\ 1000\ 0000\ 0100\ 0101\ 0000 \vee 00\ 1010\ 0010\ 0000\ 1000\ 0010\ 0111 \vee 00\ 0110\ 0110\ 0001\ 1000\ 0110\ 0110 = 11\ 1110\ 1110\ 0001\ 1100\ 0111\ 0111$, representing $\{a, b, c, d, e, g, h, i, n, o, p, t, u, v, x, y, z\}$

59. a) $\{1, 2, 3, \{1, 2, 3\}\}$ **b)** $\{\emptyset\}$ **c)** $\{\emptyset, \{\emptyset\}\}$ **d)** $\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$ **61. a)** $\{3 \cdot a, 3 \cdot b, 1 \cdot c, 4 \cdot d\}$ **b)** $\{2 \cdot a, 2 \cdot b\}$ **c)** $\{1 \cdot a, 1 \cdot c\}$ **d)** $\{1 \cdot b, 4 \cdot d\}$ **e)** $\{5 \cdot a, 5 \cdot b, 1 \cdot c, 4 \cdot d\}$

63. $\overline{F} = \{0.4 \text{ Alice}, 0.1 \text{ Brian}, 0.6 \text{ Fred}, 0.9 \text{ Oscar}, 0.5 \text{ Rita}\}$, $\overline{R} = \{0.6 \text{ Alice}, 0.2 \text{ Brian}, 0.8 \text{ Fred}, 0.1 \text{ Oscar}, 0.3 \text{ Rita}\}$

65. $\{0.4 \text{ Alice}, 0.8 \text{ Brian}, 0.2 \text{ Fred}, 0.1 \text{ Oscar}, 0.5 \text{ Rita}\}$

Section 2.3

- 1.** a) $f(0)$ is not defined. b) $f(x)$ is not defined for $x < 0$.
c) $f(x)$ is not well-defined because there are two distinct values assigned to each x . **3.** a) Not a function b) A function
c) Not a function **5.** a) Domain the set of bit strings; range the set of integers b) Domain the set of bit strings; range the set of even nonnegative integers c) Domain the set of bit strings; range the set of nonnegative integers not exceeding 7 d) Domain the set of positive integers; range the set of squares of positive integers = $\{1, 4, 9, 16, \dots\}$

7. a) Domain $\mathbf{Z}^+ \times \mathbf{Z}^+$; range \mathbf{Z}^+ b) Domain \mathbf{Z}^+ ; range $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ c) Domain the set of bit strings; range \mathbf{N} d) Domain the set of bit strings; range \mathbf{N} **9.** a) 1 b) 0 c) 0 d) -1 e) 3 f) -1 g) 2 h) 1 **11.** Only the function in part (a) **13.** Only the functions in parts (a) and (d) **15.** a) Onto b) Not onto c) Onto d) Not onto e) Onto **17.** a) Depends on whether teachers share offices b) One-to-one assuming only one teacher per bus c) Most likely not one-to-one, especially if salary is set by a collective bargaining agreement d) One-to-one **19.** Answers will vary. a) Set of offices at the school; probably not onto b) Set of buses going on the trip; onto, assuming every bus gets a teacher chaperone c) Set of real numbers; not onto d) Set of strings of nine digits with hyphens after third and fifth digits; not onto **21.** a) The function $f(x)$ with $f(x) = 3x + 1$ when $x \geq 0$ and $f(x) = -3x + 2$ when $x < 0$ b) $f(x) = |x| + 1$ c) The function $f(x)$ with $f(x) = 2x + 1$ when $x \geq 0$ and $f(x) = -2x$ when $x < 0$ d) $f(x) = x^2 + 1$ **23.** a) Yes b) No c) Yes d) No **25.** Suppose that f is strictly decreasing. This means that $f(x) > f(y)$ whenever $x < y$. To show that g is strictly increasing, suppose that $x < y$. Then $g(x) = 1/f(x) < 1/f(y) = g(y)$. Conversely, suppose that g is strictly increasing. This means that $g(x) < g(y)$ whenever $x < y$. To show that f is strictly decreasing, suppose that $x < y$. Then $f(x) = 1/g(x) > 1/g(y) = f(y)$. **27.** a) Let f be a given strictly decreasing function from \mathbf{R} to itself. If

$a < b$, then $f(a) > f(b)$; if $a > b$, then $f(a) < f(b)$. Thus if $a \neq b$, then $f(a) \neq f(b)$. **b)** Answers will vary; for example, $f(x) = 0$ for $x < 0$ and $f(x) = -x$ for $x \geq 0$.

29. The function is not one-to-one, so it is not invertible. On the restricted domain, the function is the identity function on the nonnegative real numbers, $f(x) = x$, so it is its own inverse. **31. a)** $f(S) = \{0, 1, 3\}$ **b)** $f(S) = \{0, 1, 3, 5, 8\}$ **c)** $f(S) = \{0, 8, 16, 40\}$ **d)** $f(S) = \{1, 12, 33, 65\}$

33. a) Let x and y be distinct elements of A . Because g is one-to-one, $g(x)$ and $g(y)$ are distinct elements of B . Because f is one-to-one, $f(g(x)) = (f \circ g)(x)$ and $f(g(y)) = (f \circ g)(y)$ are distinct elements of C . Hence, $f \circ g$ is one-to-one. **b)** Let $y \in C$. Because f is onto, $y = f(b)$ for some $b \in B$. Now because g is onto, $b = g(x)$ for some $x \in A$. Hence, $y = f(b) = f(g(x)) = (f \circ g)(x)$. It follows that $f \circ g$ is onto.

35. No. For example, suppose that $A = \{a\}$, $B = \{b, c\}$, and $C = \{d\}$. Let $g(a) = b$, $f(b) = d$, and $f(c) = d$. Then f and $f \circ g$ are onto, but g is not. **37.** $(f + g)(x) = x^2 + x + 3$, $(fg)(x) = x^3 + 2x^2 + x + 2$ **39.** f is one-to-one because $f(x_1) = f(x_2) \rightarrow ax_1 + b = ax_2 + b \rightarrow ax_1 = ax_2 \rightarrow x_1 = x_2$. f is onto because $f((y-b)/a) = y$. $f^{-1}(y) = (y-b)/a$.

41. a) $A = B = \mathbf{R}$, $S = \{x \mid x > 0\}$, $T = \{x \mid x < 0\}$, $f(x) = x^2$ **b)** It suffices to show that $f(S) \cap f(T) \subseteq f(S \cap T)$. Let $y \in B$ be an element of $f(S) \cap f(T)$. Then $y \in f(S)$, so $y = f(x_1)$ for some $x_1 \in S$. Similarly, $y = f(x_2)$ for some $x_2 \in T$. Because f is one-to-one, it follows that $x_1 = x_2$. Therefore $x_1 \in S \cap T$, so $y \in f(S \cap T)$.

43. a) $\{x \mid 0 \leq x < 1\}$ **b)** $\{x \mid -1 \leq x < 2\}$ **c)** \emptyset

45. $f^{-1}(\bar{S}) = \{x \in A \mid f(x) \notin S\} = \overline{\{x \in A \mid f(x) \in S\}} = f^{-1}(S)$ **47.** Let $x = \lfloor x \rfloor + \epsilon$, where ϵ is a real number with $0 \leq \epsilon < 1$. If $\epsilon < \frac{1}{2}$, then $\lfloor x \rfloor - 1 < x - \frac{1}{2} < \lfloor x \rfloor$, so $\lceil x - \frac{1}{2} \rceil = \lfloor x \rfloor$ and this is the integer closest to x . If $\epsilon > \frac{1}{2}$, then $\lfloor x \rfloor < x - \frac{1}{2} < \lfloor x \rfloor + 1$, so $\lceil x - \frac{1}{2} \rceil = \lfloor x \rfloor + 1$ and this is the integer closest to x . If $\epsilon = \frac{1}{2}$, then $\lceil x - \frac{1}{2} \rceil = \lfloor x \rfloor$, which is the smaller of the two integers that surround x and are the same distance from x .

49. Write the real number x as $\lfloor x \rfloor + \epsilon$, where ϵ is a real number with $0 \leq \epsilon < 1$. Because $\epsilon = x - \lfloor x \rfloor$, it follows that $0 \leq -\lfloor x \rfloor < 1$. The first two inequalities, $x - 1 < \lfloor x \rfloor$ and $\lfloor x \rfloor \leq x$, follow directly. For the other two inequalities, write $x = \lfloor x \rfloor - \epsilon'$, where $0 \leq \epsilon' < 1$. Then $0 \leq \lfloor x \rfloor - x < 1$, and the desired inequality follows.

51. a) If $x < n$, because $\lfloor x \rfloor \leq x$, it follows that $\lfloor x \rfloor < n$. Suppose that $x \geq n$. By the definition of the floor function, it follows that $\lfloor x \rfloor \geq n$. This means that if $\lfloor x \rfloor < n$, then $x < n$.

b) If $n < x$, then because $x \leq \lceil x \rceil$, it follows that $n \leq \lceil x \rceil$. Suppose that $n \geq x$. By the definition of the ceiling function, it follows that $\lceil x \rceil \leq n$. This means that if $n < \lceil x \rceil$, then $n < x$.

53. If n is even, then $n = 2k$ for some integer k .

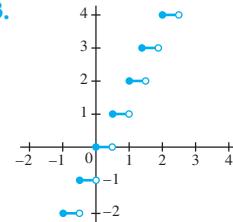
Thus, $\lfloor n/2 \rfloor = \lfloor k \rfloor = k = n/2$. If n is odd, then $n = 2k + 1$ for some integer k . Thus, $\lfloor n/2 \rfloor = \lfloor k + \frac{1}{2} \rfloor = k = (n-1)/2$.

55. Assume that $x \geq 0$. The left-hand side is $\lceil -x \rceil$ and the right-hand side is $-\lfloor x \rfloor$. If x is an integer, then both sides equal $-x$. Otherwise, let $x = n + \epsilon$, where n is a natural number and ϵ is a real number with $0 \leq \epsilon < 1$. Then $\lceil -x \rceil = \lceil -n - \epsilon \rceil = -n$ and $-\lfloor x \rfloor = -\lfloor n + \epsilon \rfloor = -n$ also. When $x < 0$, the equation also holds because it can

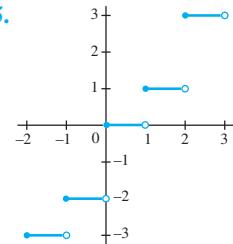
be obtained by substituting $-x$ for x . **57.** $[b] = \lfloor a \rfloor - 1$

- 59. a)** 1 **b)** 3 **c)** 126 **d)** 3600 **61. a)** 100 **b)** 256 **c)** 1030 **d)** 30,200

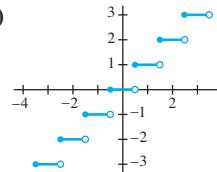
63.



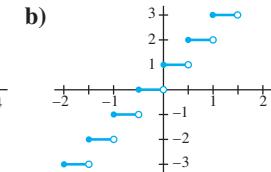
65.



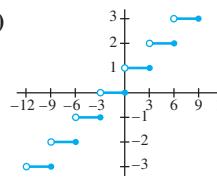
67. a)



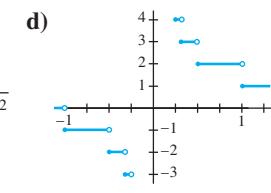
b)



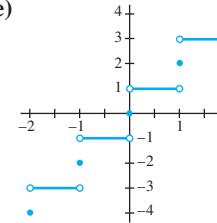
c)



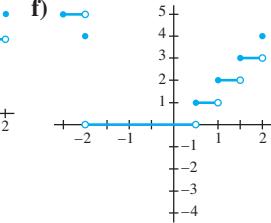
d)



e)



f)



g) See part (a). **69.** $f^{-1}(y) = (y-1)^{1/3}$ **71. a)** $f_{A \cap B}(x) = 1 \Leftrightarrow x \in A \cap B \Leftrightarrow x \in A \text{ and } x \in B \Leftrightarrow f_A(x) = 1 \text{ and } f_B(x) = 1 \Leftrightarrow f_A(x)f_B(x) = 1$ **b)** $f_{A \cup B}(x) = 1 \Leftrightarrow x \in A \cup B \Leftrightarrow x \in A \text{ or } x \in B \Leftrightarrow f_A(x) = 1 \text{ or } f_B(x) = 1 \Leftrightarrow f_A(x) + f_B(x) - f_A(x)f_B(x) = 1$

c) $f_{\bar{A}}(x) = 1 \Leftrightarrow x \in \bar{A} \Leftrightarrow x \notin A \Leftrightarrow f_A(x) = 0 \Leftrightarrow 1 - f_A(x) = 1$ **d)** $f_{A \oplus B}(x) = 1 \Leftrightarrow x \in A \oplus B \Leftrightarrow (x \in A \text{ and } x \notin B) \text{ or } (x \notin A \text{ and } x \in B) \Leftrightarrow f_A(x) + f_B(x) - 2f_A(x)f_B(x) = 1$

73. a) True; because $\lfloor x \rfloor$ is already an integer, $\lceil \lfloor x \rfloor \rceil = \lfloor x \rfloor$.

b) False; $x = \frac{1}{2}$ is a counterexample. **c)** True; if x or y is an integer, then by property 4b in Table 1, the difference is 0. If

neither x nor y is an integer, then $x = n + \epsilon$ and $y = m + \delta$, where n and m are integers and ϵ and δ are positive real numbers less than 1. Then $m + n < x + y < m + n + 2$, so $\lceil x + y \rceil$ is either $m + n + 1$ or $m + n + 2$. Therefore, the given expression is either $(n + 1) + (m + 1) - (m + n + 1) = 1$ or $(n + 1) + (m + 1) - (m + n + 2) = 0$, as desired. **d**) False; $x = \frac{1}{4}$ and $y = 3$ is a counterexample. **e**) False; $x = \frac{1}{2}$ is a counterexample. **75. a)** If x is a positive integer, then the two sides are equal. So suppose that $x = n^2 + m + \epsilon$, where n^2 is the largest perfect square less than x , m is a nonnegative integer, and $0 < \epsilon \leq 1$. Then both \sqrt{x} and $\sqrt{\lceil x \rceil} = \sqrt{n^2 + m}$ are between n and $n + 1$, so both sides equal n . **b)** If x is a positive integer, then the two sides are equal. So suppose that $x = n^2 - m - \epsilon$, where n^2 is the smallest perfect square greater than x , m is a nonnegative integer, and ϵ is a real number with $0 < \epsilon \leq 1$. Then both \sqrt{x} and $\sqrt{\lceil x \rceil} = \sqrt{n^2 - m}$ are between $n - 1$ and n . Therefore, both sides of the equation equal n . **77. a)** Domain is \mathbf{Z} ; codomain is \mathbf{R} ; domain of definition is the set of nonzero integers; the set of values for which f is undefined is $\{0\}$; not a total function. **b)** Domain is \mathbf{Z} ; codomain is \mathbf{Z} ; domain of definition is \mathbf{Z} ; set of values for which f is undefined is \emptyset ; total function. **c)** Domain is $\mathbf{Z} \times \mathbf{Z}$; codomain is \mathbf{Q} ; domain of definition is $\mathbf{Z} \times (\mathbf{Z} - \{0\})$; set of values for which f is undefined is $\mathbf{Z} \times \{0\}$; not a total function. **d)** Domain is $\mathbf{Z} \times \mathbf{Z}$; codomain is \mathbf{Z} ; domain of definition is $\mathbf{Z} \times \mathbf{Z}$; set of values for which f is undefined is \emptyset ; total function. **e)** Domain is $\mathbf{Z} \times \mathbf{Z}$; codomain is \mathbf{Z} ; domain of definitions is $\{(m, n) \mid m > n\}$; set of values for which f is undefined is $\{(m, n) \mid m \leq n\}$; not a total function. **79. a)** By definition, to say that S has cardinality m is to say that S has exactly m distinct elements. Therefore we can assign the first object to 1, the second to 2, and so on. This provides the one-to-one correspondence. **b)** By part (a), there is a bijection f from S to $\{1, 2, \dots, m\}$ and a bijection g from T to $\{1, 2, \dots, m\}$. Then the composition $g^{-1} \circ f$ is the desired bijection from S to T .

Section 2.4

- 1. a)** 3 **b)** -1 **c)** 787 **d)** 2639 **3. a)** $a_0 = 2, a_1 = 3, a_2 = 5, a_3 = 9$ **b)** $a_0 = 1, a_1 = 4, a_2 = 27, a_3 = 256$ **c)** $a_0 = 0, a_1 = 0, a_2 = 1, a_3 = 1$ **d)** $a_0 = 0, a_1 = 1, a_2 = 2, a_3 = 3$ **5. a)** 2, 5, 8, 11, 14, 17, 20, 23, 26, 29 **b)** 1, 1, 1, 2, 2, 2, 3, 3, 3, 4 **c)** 1, 1, 3, 3, 5, 5, 7, 7, 9, 9 **d)** -1, -2, -2, 8, 88, 656, 4912, 40064, 362368, 3627776 **e)** 3, 6, 12, 24, 48, 96, 192, 384, 768, 1536 **f)** 2, 4, 6, 10, 16, 26, 42, 68, 110, 178 **g)** 1, 2, 2, 3, 3, 3, 3, 4, 4, 4 **h)** 3, 3, 5, 4, 4, 3, 5, 5, 4, 3 **7.** Each term could be twice the previous term; the n th term could be obtained from the previous term by adding $n - 1$; the terms could be the positive integers that are not multiples of 3; there are infinitely many other possibilities. **9. a)** 2, 12, 72, 432, 2592 **b)** 2, 4, 16, 256, 65,536 **c)** 1, 2, 5, 11, 26 **d)** 1, 1, 6, 27, 204 **e)** 1, 2, 0, 1, 3 **11. a)** 6, 17, 49, 143, 421 **b)** $49 = 5 \cdot 17 - 6 \cdot 6, 143 = 5 \cdot 49 - 6 \cdot 17, 421 = 5 \cdot 143 - 6 \cdot 49$ **c)** $5a_{n-1} - 6a_{n-2} = 5(2^{n-1} + 5 \cdot$

$$3^{n-1}) - 6(2^{n-2} + 5 \cdot 3^{n-2}) = 2^{n-2}(10 - 6) + 3^{n-2}(75 - 30) = 2^{n-2} \cdot 4 + 3^{n-2} \cdot 9 \cdot 5 = 2^n + 3^n \cdot 5 = a_n$$

13. a) Yes **b)** No **c)** No **d)** Yes **e)** Yes **f)** Yes **g)** No **h)** No

$$\begin{aligned} \text{15. a)} \quad &a_{n-1} + 2a_{n-2} + 2n - 9 = -(n-1) + 2 + 2[-(n-2) + 2] + 2n - 9 = -n+2 = a_n \quad \text{b)} \quad a_{n-1} + \\ &2a_{n-2} + 2n - 9 = 5(-1)^{n-1} - (n-1) + 2 + 2[5(-1)^{n-2} - (n-2) + 2] + 2n - 9 = 5(-1)^{n-2}(-1+2) - n+2 = a_n \end{aligned}$$

$$\begin{aligned} \text{c)} \quad &a_{n-1} + 2a_{n-2} + 2n - 9 = 3(-1)^{n-1} + 2^{n-1} - (n-1) + 2 + 2[3(-1)^{n-2} + 2^{n-2} - (n-2) + 2] + 2n - 9 = 3(-1)^{n-2} \\ &(-1+2) + 2^{n-2}(2+2) - n+2 = a_n \quad \text{d)} \quad a_{n-1} + \\ &2a_{n-2} + 2n - 9 = 7 \cdot 2^{n-1} - (n-1) + 2 + 2[7 \cdot 2^{n-2} - (n-2) + 2] + 2n - 9 = 2^{n-2}(7 \cdot 2 + 2 \cdot 7) - n+2 = a_n \end{aligned}$$

$$\begin{aligned} \text{17. a)} \quad &a_n = 2 \cdot 3^n \quad \text{b)} \quad a_n = 2n + 3 \quad \text{c)} \quad a_n = 1 + n(n+1)/2 \quad \text{d)} \quad a_n = n^2 + 4n + 4 \quad \text{e)} \quad a_n = 1 \quad \text{f)} \quad a_n = (3^{n+1} - 1)/2 \quad \text{g)} \quad a_n = 5n! \quad \text{h)} \quad a_n = 2^n n! \quad \text{19. a)} \quad a_n = 3a_{n-1} \quad \text{b)} \quad 5,904,900 \end{aligned}$$

$$\begin{aligned} \text{21. a)} \quad &a_n = n + a_{n-1}, a_0 = 0 \quad \text{b)} \quad a_{12} = 78 \quad \text{c)} \quad a_n = n(n+1)/2 \quad \text{23. B}(k) = [1 + (0.07/12)]B(k-1) - 100, \text{ with } B(0) = 5000 \quad \text{25. a)} \quad \text{One 1 and one 0, followed by two 1s and two 0s, followed by three 1s and three 0s, and so on; 1, 1, 1} \quad \text{b)} \quad \text{The positive integers are listed in increasing order with each even positive integer listed twice; 9, 10, 10. c)} \quad \text{The terms in odd-numbered locations are the successive powers of 2; the terms in even-numbered locations are all 0; 32, 0, 64. d)} \quad a_n = 3 \cdot 2^{n-1}; 384, 768, 1536 \end{aligned}$$

$$\begin{aligned} \text{e)} \quad &a_n = 15 - 7(n-1) = 22 - 7n; -34, -41, -48 \quad \text{f)} \quad a_n = (n^2 + n + 4)/2; 57, 68, 80 \quad \text{g)} \quad a_n = 2n^3; 1024, 1458, 2000 \quad \text{h)} \quad a_n = n! + 1; 362881, 3628801, 39916801 \quad \text{27. Among the integers } 1, 2, \dots, a_n, \text{ where } a_n \text{ is the } n \text{th positive integer not a perfect square, the nonsquares are } a_1, a_2, \dots, a_n \text{ and the squares are } 1^2, 2^2, \dots, k^2, \text{ where } k \text{ is the integer with } k^2 < n+k < (k+1)^2. \text{ Consequently, } a_n = n+k, \text{ where } k^2 < a_n < (k+1)^2. \text{ To find } k, \text{ first note that } k^2 < n+k < (k+1)^2, \text{ so } k^2 + 1 \leq n+k \leq (k+1)^2 - 1. \text{ Hence, } (k-\frac{1}{2})^2 + \frac{3}{4} = k^2 - k + 1 \leq n \leq k^2 + k = (k+\frac{1}{2})^2 - \frac{1}{4}. \text{ It follows that } k - \frac{1}{2} < \sqrt{n} < k + \frac{1}{2}, \text{ so } k = \lfloor \sqrt{n} \rfloor \text{ and } a_n = n+k = n + \lfloor \sqrt{n} \rfloor. \quad \text{29. a)} \quad 20 \quad \text{b)} \quad 11 \quad \text{c)} \quad 30 \quad \text{d)} \quad 511 \quad \text{31. a)} \quad 1533 \quad \text{b)} \quad 510 \quad \text{c)} \quad 4923 \quad \text{d)} \quad 9842 \quad \text{33. a)} \quad 21 \quad \text{b)} \quad 78 \quad \text{c)} \quad 18 \quad \text{d)} \quad 18 \quad \text{35. } \sum_{j=1}^n (a_j - a_{j-1}) = a_n - a_0 \end{aligned}$$

$$\begin{aligned} \text{37. a)} \quad &n^2 \quad \text{b)} \quad n(n+1)/2 \quad \text{39. } 15150 \quad \text{41. } \frac{n(n+1)(2n+1)}{3} + \frac{n(n+1)}{2} + (n+1)(m - (n+1)^2 + 1), \text{ where } n = \lfloor \sqrt{m} \rfloor - 1 \quad \text{43. a)} \quad 0 \quad \text{b)} \quad 1680 \quad \text{c)} \quad 1 \quad \text{d)} \quad 1024 \quad \text{45. } 34 \end{aligned}$$

Section 2.5

1. a) Countably infinite, -1, -2, -3, -4, ... **b)** Countably infinite, 0, 2, -2, 4, -4, ... **c)** Countably infinite, 99, 98, 97, ... **d)** Uncountable **e)** Finite **f)** Countably infinite, 0, 7, -7, 14, -14, ... **3. a)** Countable: match n with the string of n 1s. **b)** Countable. To find a correspondence, follow the path in Example 4, but omit fractions in the top three rows (as well as continuing to omit fractions not in lowest terms). **c)** Uncountable **d)** Uncountable **5.** Suppose m new guests arrive at the fully occupied hotel. Move the guest in Room n to Room $m+n$ for $n = 1, 2, 3, \dots$; then the new guests can occupy rooms 1 to m . **7.** For $n = 1, 2, 3, \dots$, put

the guest currently in Room $2n$ into Room n , and the guest currently in Room $2n - 1$ into Room n of the new building. **9.** Move the guest currently Room i to Room $2i + 1$ for $i = 1, 2, 3, \dots$. Put the j th guest from the k th bus into Room $2^k(2j + 1)$. **11. a)** $A = [1, 2]$ (closed interval of real numbers from 1 to 2), $B = [3, 4]$ **b)** $A = [1, 2] \cup \mathbf{Z}^+$, $B = [3, 4] \cup \mathbf{Z}^+$ **c)** $A = [1, 3]$, $B = [2, 4]$ **13.** Suppose that A is countable. Then either A has cardinality n for some non-negative integer n , in which case there is a one-to-one function from A to a subset of \mathbf{Z}^+ (the range is the first n positive integers), or there exists a one-to-one correspondence f from A to \mathbf{Z}^+ ; in either case we have satisfied Definition 2. Conversely, suppose that $|A| \leq |\mathbf{Z}^+|$. By definition, this means that there is a one-to-one function from A to \mathbf{Z}^+ , so A has the same cardinality as a subset of \mathbf{Z}^+ (namely the range of that function). By Exercise 16 we conclude that A is countable. **15.** Assume that B is countable. Then the elements of B can be listed as b_1, b_2, b_3, \dots . Because A is a subset of B , taking the subsequence of $\{b_n\}$ that contains the terms that are in A gives a listing of the elements of A . Because A is uncountable, this is impossible. **17.** Assume that $A - B$ is countable. Then, because $A = (A - B) \cup (A \cap B)$, the elements of A can be listed in a sequence by alternating elements of $A - B$ and elements of $A \cap B$. This contradicts the uncountability of A . **19.** We are given bijections f from A to B and g from C to D . Then the function from $A \times C$ to $B \times D$ that sends (a, c) to $(f(a), g(c))$ is a bijection. **21.** By the definition of $|A| \leq |B|$, there is a one-to-one function $f : A \rightarrow B$. Similarly, there is a one-to-one function $g : B \rightarrow C$. By Exercise 33 in Section 2.3, the composition $g \circ f : A \rightarrow C$ is one-to-one. Therefore by definition $|A| \leq |C|$. **23.** Using the Axiom of Choice from set theory, choose distinct elements a_1, a_2, a_3, \dots of A one at a time (this is possible because A is infinite). The resulting set $\{a_1, a_2, a_3, \dots\}$ is the desired infinite subset of A . **25.** The set of finite strings of characters over a finite alphabet is countably infinite, because we can list these strings in alphabetical order by length. Therefore the infinite set S can be identified with an infinite subset of this countable set, which by Exercise 16 is also countably infinite. **27.** Suppose that A_1, A_2, A_3, \dots are countable sets. Because A_i is countable, we can list its elements in a sequence as $a_{i1}, a_{i2}, a_{i3}, \dots$. The elements of the set $\bigcup_{i=1}^n A_i$ can be listed by listing all terms a_{ij} with $i + j = 2$, then all terms a_{ij} with $i + j = 3$, then all terms a_{ij} with $i + j = 4$, and so on. **29.** There are a finite number of bit strings of length m , namely, 2^m . The set of all bit strings is the union of the sets of bit strings of length m for $m = 0, 1, 2, \dots$. Because the union of a countable number of countable sets is countable (see Exercise 27), there are a countable number of bit strings. **31.** It is clear from the formula that the range of values the function takes on for a fixed value of $m + n$, say $m + n = x$, is $(x - 2)(x - 1)/2 + 1$ through $(x - 2)(x - 1)/2 + (x - 1)$, because m can assume the values $1, 2, 3, \dots, (x - 1)$ under these conditions, and the first term in the formula is a fixed positive integer when $m + n$ is fixed. To show that this function is one-to-one and onto, we merely need to show that the range of values for

$x + 1$ picks up precisely where the range of values for x left off, i.e., that $f(x - 1, 1) + 1 = f(1, x)$. We have $f(x - 1, 1) + 1 = \frac{(x-2)(x-1)}{2} + (x-1) + 1 = \frac{x^2-x+2}{2} = \frac{(x-1)x}{2} + 1 = f(1, x)$. **33.** By the Schröder-Bernstein theorem, it suffices to find one-to-one functions $f : (0, 1) \rightarrow [0, 1]$ and $g : [0, 1] \rightarrow (0, 1)$. Let $f(x) = x$ and $g(x) = (x + 1)/3$. **35.** Each element A of the power set of the set of positive integers (i.e., $A \subseteq \mathbf{Z}^+$) can be represented uniquely by the bit string $a_1a_2a_3\dots$, where $a_i = 1$ if $i \in A$ and $a_i = 0$ if $i \notin A$. Assume there were a one-to-one correspondence $f : \mathbf{Z}^+ \rightarrow \mathcal{P}(\mathbf{Z}^+)$. Form a new bit string $s = s_1s_2s_3\dots$ by setting s_i to be 1 minus the i th bit of $f(i)$. Then because s differs in the i th bit from $f(i)$, s is not in the range of f , a contradiction. **37.** For any finite alphabet there are a finite number of strings of length n , whenever n is a positive integer. It follows by the result of Exercise 27 that there are only a countable number of strings from any given finite alphabet. Because the set of all computer programs in a particular language is a subset of the set of all strings of a finite alphabet, which is a countable set by the result from Exercise 16, it is itself a countable set. **39.** Exercise 37 shows that there are only a countable number of computer programs. Consequently, there are only a countable number of computable functions. Because, as Exercise 38 shows, there are an uncountable number of functions, not all functions are computable.

Section 2.6

- 1. a)** 3×4 **b)** $\begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix}$ **c)** $[2 \ 0 \ 4 \ 6]$ **d)** 1
- e)** $\begin{bmatrix} 1 & 2 & 1 \\ 1 & 0 & 1 \\ 1 & 4 & 3 \\ 3 & 6 & 7 \end{bmatrix}$ **3. a)** $\begin{bmatrix} 1 & 11 \\ 2 & 18 \end{bmatrix}$ **b)** $\begin{bmatrix} 2 & -2 & -3 \\ 1 & 0 & 2 \\ 9 & -4 & 4 \end{bmatrix}$
- c)** $\begin{bmatrix} -4 & 15 & -4 & 1 \\ -3 & 10 & 2 & -3 \\ 0 & 2 & -8 & 6 \\ 1 & -8 & 18 & -13 \end{bmatrix}$ **5.** $\begin{bmatrix} 9/5 & -6/5 \\ -1/5 & 4/5 \end{bmatrix}$
- 7. $0 + A = [0 + a_{ij}] = [a_{ij} + 0] = 0 + A$** **9. $A + (B + C) = [a_{ij} + (b_{ij} + c_{ij})] = [(a_{ij} + b_{ij}) + c_{ij}] = (A + B) + C$**
- 11.** The number of rows of A equals the number of columns of B , and the number of columns of A equals the number of rows of B . **13. $A(BC) = [\sum_q a_{iq} (\sum_r b_{qr} c_{rl})] = [\sum_q \sum_r a_{iq} b_{qr} c_{rl}] = [\sum_r (\sum_q a_{iq} b_{qr}) c_{rl}] = (AB)C$**
- 15. $A^n = \begin{bmatrix} 1 & n \\ 0 & 1 \end{bmatrix}$** **17. a)** Let $A = [a_{ij}]$ and $B = [b_{ij}]$. Then $A + B = [a_{ij} + b_{ij}] = [a_{ji} + b_{ji}] = A^t + B^t$. **b)** Using the same notation as in part (a), we have $B^t A^t =$

$\left[\sum_q b_{qi} a_{jq} \right] = \left[\sum_j a_{jq} b_{qi} \right] = (\mathbf{AB})^t$, because the (i, j) th entry is the (j, i) th entry of \mathbf{AB} . **19.** The result follows because $\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} = \begin{bmatrix} ad - bc & 0 \\ 0 & ad - bc \end{bmatrix} = (ad - bc)\mathbf{I}_2 = \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. **21.** $\mathbf{A}^n(\mathbf{A}^{-1})^n = \mathbf{A}(\mathbf{A} \cdots (\mathbf{A}(\mathbf{AA}^{-1})\mathbf{A}^{-1}) \cdots \mathbf{A}^{-1})\mathbf{A}^{-1}$ by the associative law. Because $\mathbf{AA}^{-1} = \mathbf{I}$, working from the inside shows that $\mathbf{A}^n(\mathbf{A}^{-1})^n = \mathbf{I}$. Similarly $(\mathbf{A}^{-1})^n\mathbf{A}^n = \mathbf{I}$. Therefore $(\mathbf{A}^n)^{-1} = (\mathbf{A}^{-1})^n$. **23.** The (i, j) th entry of $\mathbf{A} + \mathbf{A}^t$ is $a_{ij} + a_{ji}$, which equals $a_{ji} + a_{ij}$, the (j, i) th entry of $\mathbf{A} + \mathbf{A}^t$, so by definition $\mathbf{A} + \mathbf{A}^t$ is symmetric. **25.** $x_1 = 1$, $x_2 = -1$, $x_3 = -2$

27. a) $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$ b) $\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ c) $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$
29. a) $\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$ b) $\begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$ c) $\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

31. a) $\mathbf{A} \vee \mathbf{B} = [a_{ij} \vee b_{ij}] = [b_{ij} \vee a_{ij}] = \mathbf{B} \vee \mathbf{A}$ b) $\mathbf{A} \wedge \mathbf{B} = [a_{ij} \wedge b_{ij}] = [b_{ij} \wedge a_{ij}] = \mathbf{B} \wedge \mathbf{A}$ **33.** a) $\mathbf{A} \vee (\mathbf{B} \wedge \mathbf{C}) = [a_{ij}] \vee [b_{ij} \wedge c_{ij}] = [a_{ij} \vee (b_{ij} \wedge c_{ij})] = [(a_{ij} \vee b_{ij}) \wedge (a_{ij} \vee c_{ij})] = [a_{ij} \vee b_{ij}] \wedge [a_{ij} \vee c_{ij}] = (\mathbf{A} \vee \mathbf{B}) \wedge (\mathbf{A} \vee \mathbf{C})$
b) $\mathbf{A} \wedge (\mathbf{B} \vee \mathbf{C}) = [a_{ij}] \wedge [b_{ij} \vee c_{ij}] = [a_{ij} \wedge (b_{ij} \vee c_{ij})] = [(a_{ij} \wedge b_{ij}) \vee (a_{ij} \wedge c_{ij})] = [a_{ij} \wedge b_{ij}] \vee [a_{ij} \wedge c_{ij}] = (\mathbf{A} \wedge \mathbf{B}) \vee (\mathbf{A} \wedge \mathbf{C})$ **35.** $\mathbf{A} \odot (\mathbf{B} \odot \mathbf{C}) = \left[\bigvee_q a_{iq} \wedge (\bigvee_r (b_{qr} \wedge c_{rl})) \right] = \left[\bigvee_q \bigvee_r (a_{iq} \wedge b_{qr} \wedge c_{rl}) \right] = \left[\bigvee_r \bigvee_q (a_{iq} \wedge b_{qr} \wedge c_{rl}) \right] = \left[\bigvee_r \left(\bigvee_q (a_{iq} \wedge b_{qr}) \right) \wedge c_{rl} \right] = (\mathbf{A} \odot \mathbf{B}) \odot \mathbf{C}$

Supplementary Exercises

- 1.** a) \overline{A} b) $A \cap B$ c) $A - B$ d) $\overline{A} \cap \overline{B}$ e) $\overline{A} \oplus B$ **3.** Yes
5. $A - (A - B) = A - (A \cap \overline{B}) = A \cap (A \cap \overline{B}) = A \cap (\overline{A} \cup B) = (A \cap \overline{A}) \cup (A \cap B) = \emptyset \cup (A \cap B) = A \cap B$ **7.** Let $A = \{1\}$, $B = \emptyset$, $C = \{1\}$. Then $(A - B) - C = \emptyset$, but $A - (B - C) = \{1\}$. **9.** No. For example, let $A = B = \{a, b\}$, $C = \emptyset$, and $D = \{a\}$. Then $(A - B) - (C - D) = \emptyset - \emptyset = \emptyset$, but $(A - C) - (B - D) = \{a, b\} - \{b\} = \{a\}$.
11. a) $|\emptyset| \leq |A \cap B| \leq |A| \leq |A \cup B| \leq |U|$ b) $|\emptyset| \leq |A - B| \leq |A \oplus B| \leq |A \cup B| \leq |A| + |B|$ **13.** a) Yes, no
b) Yes, no c) f has inverse with $f^{-1}(a) = 3$, $f^{-1}(b) = 4$, $f^{-1}(c) = 2$, $f^{-1}(d) = 1$; g has no inverse. **15.** If f is one-to-one, then f provides a bijection between S and $f(S)$, so they have the same cardinality. If f is not one-to-one, then there exist elements x and y in S such that $f(x) = f(y)$. Let $S = \{x, y\}$. Then $|S| = 2$ but $|f(S)| = 1$. **17.** Let $x \in A$. Then $S_f(\{x\}) = \{f(y) \mid y \in \{x\}\} = \{f(x)\}$. By

the same reasoning, $S_g(\{x\}) = \{g(x)\}$. Because $S_f = S_g$, we can conclude that $\{f(x)\} = \{g(x)\}$, and so necessarily $f(x) = g(x)$. **19.** The equation is true if and only if the sum of the fractional parts of x and y is less than 1. **21.** The equation is true if and only if either both x and y are integers, or x is not an integer but the sum of the fractional parts of x and y is less than or equal to 1. **23.** If x is an integer, then $\lfloor x \rfloor + \lfloor m - x \rfloor = x + m - x = m$. Otherwise, write x in terms of its integer and fractional parts: $x = n + \epsilon$, where $n = \lfloor x \rfloor$ and $0 < \epsilon < 1$. In this case $\lfloor x \rfloor + \lfloor m - x \rfloor = \lfloor n + \epsilon \rfloor + \lfloor m - n - \epsilon \rfloor = n + m - n - 1 = m - 1$. **25.** Write $n = 2k + 1$ for some integer k . Then $n^2 = 4k^2 + 4k + 1$, so $n^2/4 = k^2 + k + \frac{1}{4}$. Therefore, $\lceil n^2/4 \rceil = k^2 + k + 1$. But $(n^2+3)/4 = (4k^2+4k+1+3)/4 = k^2+k+1$. **27.** Let $x = n + (r/m) + \epsilon$, where n is an integer, r is a nonnegative integer less than m , and ϵ is a real number with $0 \leq \epsilon < 1/m$. The left-hand side is $\lfloor nm + r + m\epsilon \rfloor = nm + r$. On the right-hand side, the terms $\lfloor x \rfloor$ through $\lfloor x + (m+r-1)/m \rfloor$ are all just n and the terms from $\lfloor x + (m-r)/m \rfloor$ on are all $n+1$. Therefore, the right-hand side is $(m-r)n + r(n+1) = nm + r$, as well. **29.** 101 **31.** $a_1 = 1$; $a_{2n+1} = n \cdot a_{2n}$ for all $n > 0$; and $a_{2n} = n + a_{2n-1}$ for all $n > 0$. The next four terms are 5346, 5353, 37471, and 37479. **33.** If each $f^{-1}(j)$ is countable, then $S = f^{-1}(1) \cup f^{-1}(2) \cup \dots$ is the countable union of countable sets and is therefore countable by Exercise 27 in Section 2.5. **35.** Because there is a one-to-one correspondence between \mathbf{R} and the open interval $(0, 1)$ (given by $f(x) = 2 \arctan(x/\pi)$), it suffices to show that $|(0, 1) \times (0, 1)| = |(0, 1)|$. By the Schröder-Bernstein theorem it suffices to find injective functions $f : (0, 1) \rightarrow (0, 1) \times (0, 1)$ and $g : (0, 1) \times (0, 1) \rightarrow (0, 1)$. Let $f(x) = (x, \frac{1}{2})$. For g we follow the hint. Suppose $(x, y) \in (0, 1) \times (0, 1)$, and represent x and y with their decimal expansions $x = 0.x_1x_2x_3\dots$ and $y = 0.y_1y_2y_3\dots$, never choosing the expansion of any number that ends in an infinite string of 9s. Let $g(x, y)$ be the decimal expansion obtained by interweaving these two strings, namely $0.x_1y_1x_2y_2x_3y_3\dots$

37. $\mathbf{A}^{4n} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\mathbf{A}^{4n+1} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$, $\mathbf{A}^{4n+2} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$, $\mathbf{A}^{4n+3} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$, for $n \geq 0$ **39.** Suppose that $\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. Let $\mathbf{B} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$. Because $\mathbf{AB} = \mathbf{BA}$, it follows that $c = 0$ and $a = d$. Let $\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$. Because

$\mathbf{AB} = \mathbf{BA}$, it follows that $b = 0$. Hence, $\mathbf{A} = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix} = a\mathbf{I}$.

- 41.** a) Let $\mathbf{A} \odot \mathbf{0} = [b_{ij}]$. Then $b_{ij} = (a_{i1} \wedge 0) \vee \dots \vee (a_{ip} \wedge 0) = 0$. Hence, $\mathbf{A} \odot \mathbf{0} = \mathbf{0}$. Similarly $\mathbf{0} \odot \mathbf{A} = \mathbf{0}$.
b) $\mathbf{A} \vee \mathbf{0} = [a_{ij} \vee 0] = [a_{ij}] = \mathbf{A}$. Hence $\mathbf{A} \vee \mathbf{0} = \mathbf{A}$. Similarly $\mathbf{0} \vee \mathbf{A} = \mathbf{A}$. c) $\mathbf{A} \wedge \mathbf{0} = [a_{ij} \wedge 0] = [0] = \mathbf{0}$. Hence $\mathbf{A} \wedge \mathbf{0} = \mathbf{0}$. Similarly $\mathbf{0} \wedge \mathbf{A} = \mathbf{0}$.

CHAPTER 3

Section 3.1

1. $max := 1, i := 2, max := 8, i := 3, max := 12, i := 4, i := 5, i := 6, i := 7, max := 14, i := 8, i := 9, i := 10, i := 11$

3. **procedure** *AddUp*(a_1, \dots, a_n : integers)

```
sum :=  $a_1$ 
for  $i := 2$  to  $n$ 
    sum := sum +  $a_i$ 
return sum
```

5. **procedure** *duplicates*(a_1, a_2, \dots, a_n : integers in nondecreasing order)

$k := 0$ {this counts the duplicates}

$j := 2$

while $j \leq n$

if $a_j = a_{j-1}$ **then**

$k := k + 1$

$c_k := a_j$

while $j \leq n$ and $a_j = c_k$

$j := j + 1$

$j := j + 1$

{ c_1, c_2, \dots, c_k is the desired list}

7. **procedure** *last even location*(a_1, a_2, \dots, a_n : integers)

$k := 0$

for $i := 1$ **to** n

if a_i is even **then** $k := i$

return k { $k = 0$ if there are no evens}

9. **procedure** *palindrome check*($a_1 a_2 \dots a_n$: string)

$answer := \text{true}$

for $i := 1$ **to** $\lfloor n/2 \rfloor$

if $a_i \neq a_{n+1-i}$ **then** $answer := \text{false}$

return $answer$

11. **procedure** *interchange*(x, y : real numbers)

$z := x$

$x := y$

$y := z$

The minimum number of assignments needed is three.

13. Linear search: $i := 1, i := 2, i := 3, i := 4, i := 5, i := 6, i := 7, location := 7$; binary search: $i := 1, j := 8, m := 4, i := 5, m := 6, i := 7, m := 7, j := 7, location := 7$

15. **procedure** *insert*(x, a_1, a_2, \dots, a_n : integers)

{the list is in order: $a_1 \leq a_2 \leq \dots \leq a_n$ }

$a_{n+1} := x + 1$

$i := 1$

while $x > a_i$

$i := i + 1$

for $j := 0$ **to** $n - i$

$a_{n-j+1} := a_{n-j}$

$a_i := x$

{ x has been inserted into correct position}

17. **procedure** *first largest*(a_1, \dots, a_n : integers)

```
max :=  $a_1$ 
location := 1
for  $i := 2$  to  $n$ 
    if  $max < a_i$  then
        max :=  $a_i$ 
        location :=  $i$ 
return location
```

19. **procedure** *mean-median-max-min*(a, b, c : integers)

$mean := (a + b + c) / 3$

{the six different orderings of a, b, c with respect to \geq will be handled separately}

if $a \geq b$ **then**

if $b \geq c$ **then** median := b ; max := a ; min := c

 :

(The rest of the algorithm is similar.)

21. **procedure** *first-three*(a_1, a_2, \dots, a_n : integers)

if $a_1 > a_2$ **then** interchange a_1 and a_2

if $a_2 > a_3$ **then** interchange a_2 and a_3

if $a_1 > a_2$ **then** interchange a_1 and a_2

23. **procedure** *onto*(f : function from A to B where

$A = \{a_1, \dots, a_n\}, B = \{b_1, \dots, b_m\}, a_1, \dots, a_n, b_1, \dots, b_m$ are integers)

for $i := 1$ **to** m

$hit(b_i) := 0$

 count := 0

for $j := 1$ **to** n

if $hit(f(a_j)) = 0$ **then**

$hit(f(a_j)) := 1$

 count := count + 1

if count = m **then** **return** true **else** **return** false

25. **procedure** *ones*(a : bit string, $a = a_1 a_2 \dots a_n$)

count := 0

for $i := 1$ **to** n

if $a_i := 1$ **then**

 count := count + 1

return count

27. **procedure** *ternary search*(s : integer, a_1, a_2, \dots, a_n : increasing integers)

$i := 1$

$j := n$

while $i < j - 1$

$l := \lfloor (i + j)/3 \rfloor$

$u := \lfloor 2(i + j)/3 \rfloor$

if $x > a_u$ **then** $i := u + 1$

else if $x > a_l$ **then**

$i := l + 1$

$j := u$

else $j := l$

if $x = a_i$ **then** location := i

else if $x = a_j$ **then** location := j

```

else location := 0
return location {0 if not found}

29. procedure find a mode( $a_1, a_2, \dots, a_n$ : nondecreasing
   integers)
modecount := 0
i := 1
while  $i \leq n$ 
   value :=  $a_i$ 
   count := 1
   while  $i \leq n$  and  $a_i = value$ 
      count := count + 1
      i := i + 1
   if count > modecount then
      modecount := count
      mode := value
   return mode

31. procedure find duplicate( $a_1, a_2, \dots, a_n$ : integers)
location := 0
i := 2
while  $i \leq n$  and location = 0
   j := 1
   while  $j < i$  and location = 0
      if  $a_i = a_j$  then location := i
      else j := j + 1
   i := i + 1
return location
{location is the subscript of the first value that
 repeats a previous value in the sequence}

33. procedure find decrease( $a_1, a_2, \dots, a_n$ : positive
   integers)
location := 0
i := 2
while  $i \leq n$  and location = 0
   if  $a_i < a_{i-1}$  then location := i
   else i := i + 1
return location
{location is the subscript of the first value less than
 the immediately preceding one}

35. At the end of the first pass: 1, 3, 5, 4, 7; at the end of the
 second pass: 1, 3, 4, 5, 7; at the end of the third pass: 1, 3, 4,
 5, 7; at the end of the fourth pass: 1, 3, 4, 5, 7

37. procedure better bubblesort( $a_1, \dots, a_n$ : integers)
i := 1; done := false
while  $i < n$  and done = false
   done := true
   for j := 1 to  $n - i$ 
      if  $a_j > a_{j+1}$  then
         interchange  $a_j$  and  $a_{j+1}$ 
         done := false
   i := i + 1
{ $a_1, \dots, a_n$  is in increasing order}

39. At the end of the first, second, and third passes: 1, 3, 5, 7, 4;
 at the end of the fourth pass: 1, 3, 4, 5, 7   41. a) 1, 5, 4, 3,
 2; 1, 2, 4, 3, 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5   b) 1, 4, 3, 2,
 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5   c) 1, 2, 3, 4,
 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5; 1, 2, 3, 4, 5   43. We carry
out the linear search algorithm given as Algorithm 2 in this
section, except that we replace  $x \neq a_i$  by  $x < a_i$ , and
we replace the else clause with else location :=  $n + 1$ .
45.  $2 + 3 + 4 + \dots + n = (n^2 + n - 2)/2$    47. Find the
location for the 2 in the list 3 (one comparison), and insert it
in front of the 3, so the list now reads 2, 3, 4, 5, 1, 6. Find the
location for the 4 (compare it to the 2 and then the 3),
and insert it, leaving 2, 3, 4, 5, 1, 6. Find the location for the
5 (compare it to the 3 and then the 4), and insert it, leaving
2, 3, 4, 5, 1, 6. Find the location for the 1 (compare it to the
3 and then the 2 and then the 2 again), and insert it, leaving
1, 2, 3, 4, 5, 6. Find the location for the 6 (compare it to the
3 and then the 4 and then the 5), and insert it, giving the final
answer 1, 2, 3, 4, 5, 6.

49. procedure binary insertion sort( $a_1, a_2, \dots, a_n$ :
   real numbers with  $n \geq 2$ )
for j := 2 to n
   {binary search for insertion location i}
   left := 1
   right := j - 1
   while left < right
      middle :=  $\lfloor (left + right)/2 \rfloor$ 
      if  $a_j > a_{middle}$  then left := middle + 1
      else right := middle
   if  $a_j < a_{left}$  then i := left else i := left + 1
   {insert  $a_j$  in location i by moving  $a_i$  through  $a_{j-1}$ 
    toward back of list}
   m :=  $a_j$ 
   for k := 0 to j - i - 1
       $a_{j-k} := a_{j-k-1}$ 
    $a_i := m$ 
{ $a_1, a_2, \dots, a_n$  are sorted}

51. The variation from Exercise 50   53. a) Two quarters, one
penny   b) Two quarters, one dime, one nickel, four pennies
c) A three quarters, one penny   d) Two quarters, one dime
55. Greedy algorithm uses fewest coins in parts (a), (c), and
(d).   a) Two quarters, one penny   b) Two quarters, one dime,
nine pennies   c) Three quarters, one penny   d) Two quarters,
one dime   57. The 9:00–9:45 talk, the 9:50–10:15 talk, the
10:15–10:45 talk, the 11:00–11:15 talk   59. a) Order the
talks by starting time. Number the lecture halls 1, 2, 3, and
so on. For each talk, assign it to lowest numbered lecture hall
that is currently available.   b) If this algorithm uses  $n$  lecture
halls, then at the point the  $n$ th hall was first assigned, it had
to be used (otherwise a lower-numbered hall would have been
assigned), which means that  $n$  talks were going on simultane-
ously (this talk just assigned and the  $n - 1$  talks currently
in halls 1 through  $n - 1$ ).   61. Here we assume that the men
are the suitors and the women the suitees.

procedure stable( $M_1, M_2, \dots, M_s, W_1, W_2, \dots, W_s$ :
   preference lists)
for i := 1 to s
   mark man i as rejected
for i := 1 to s
   set man i's rejection list to be empty
for j := 1 to s

```

```

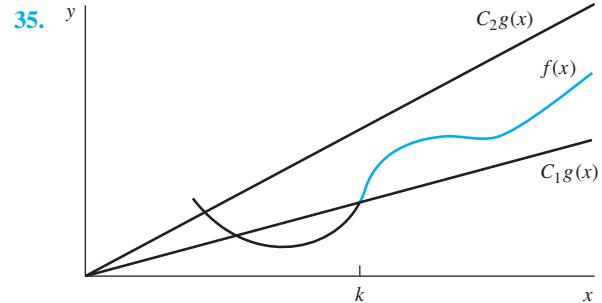
set woman  $j$ 's proposal list to be empty
while rejected men remain
  for  $i := 1$  to  $s$ 
    if man  $i$  is marked rejected then add  $i$  to the
      proposal list for the woman  $j$  who ranks highest
      on his preference list but does not appear on his
      rejection list, and mark  $i$  as not rejected
  for  $j := 1$  to  $s$ 
    if woman  $j$ 's proposal list is nonempty then
      remove from  $j$ 's proposal list all men  $i$ 
      except the man  $i_0$  who ranks highest on her
      preference list, and for each such man  $i$  mark
      him as rejected and add  $j$  to his rejection list
for  $j := 1$  to  $s$ 
  match  $j$  with the one man on  $j$ 's proposal list
{This matching is stable.}
63. If the assignment is not stable, then there is a man  $m$  and a
woman  $w$  such that  $m$  prefers  $w$  to the woman  $w'$  with whom
he is matched, and  $w$  prefers  $m$  to the man with whom she is
matched. But  $m$  must have proposed to  $w$  before he proposed
to  $w'$ , because he prefers the former. Because  $m$  did not end
up matched with  $w$ , she must have rejected him. Women re-
ject a suitor only when they get a better proposal, and they
eventually get matched with a pending suitor, so the woman
with whom  $w$  is matched must be better in her eyes than  $m$ ,
contradicting our original assumption. Therefore the marriage
is stable. 65. Run the two programs on their inputs concur-
rently and report which one halts.

```

Section 3.2

- 1.** The choices of C and k are not unique. **a)** $C = 1, k = 10$ **b)** $C = 4, k = 7$ **c)** $C = 5, k = 1$ **e)** $C = 1, k = 0$ **f)** $C = 1, k = 2$ **3.** $x^4 + 9x^3 + 4x + 7 \leq 4x^4$ for all $x > 9$; witnesses $C = 4, k = 9$ **5.** $(x^2 + 1)/(x + 1) = x - 1 + 2/(x + 1) < x$ for all $x > 1$; witnesses $C = 1, k = 1$ **7.** The choices of C and k are not unique. **a)** $n = 3, C = 3, k = 1$ **b)** $n = 3, C = 4, k = 1$ **c)** $n = 1, C = 2, k = 1$ **d)** $n = 0, C = 2, k = 1$ **9.** $x^2 + 4x + 17 \leq 3x^3$ for all $x > 17$, so $x^2 + 4x + 17$ is $O(x^3)$, with witnesses $C = 3, k = 17$. However, if x^3 were $O(x^2 + 4x + 17)$, then $x^3 \leq C(x^2 + 4x + 17) \leq 3Cx^2$ for some C , for all sufficiently large x , which implies that $x \leq 3C$ for all sufficiently large x , which is impossible. Hence, x^3 is not $O(x^2 + 4x + 17)$. **11.** $3x^4 + 1 \leq 4x^4 = 8(x^4/2)$ for all $x > 1$, so $3x^4 + 1$ is $O(x^4/2)$, with witnesses $C = 8, k = 1$. Also $x^4/2 \leq 3x^4 + 1$ for all $x > 0$, so $x^4/2$ is $O(3x^4 + 1)$, with witnesses $C = 1, k = 0$. **13.** Because $2^n \leq 3^n$ for all $n > 0$, it follows that 2^n is $O(3^n)$, with witnesses $C = 1, k = 0$. However, if 3^n were $O(2^n)$, then for some C , $3^n \leq C \cdot 2^n$ for all sufficiently large n . This says that $C \geq (3/2)^n$ for all suffi-
ciently large n , which is impossible. Hence, 3^n is not $O(2^n)$. **15.** All functions for which there exist real numbers k and C with $|f(x)| \leq C$ for $x > k$. These are the functions $f(x)$ that are bounded for all sufficiently large x . **17.** There are constants C_1, C_2, k_1 , and k_2 such that $|f(x)| \leq C_1|g(x)|$ for all $x > k_1$ and $|g(x)| \leq C_2|h(x)|$ for all $x > k_2$. Hence, for $x >$

$\max(k_1, k_2)$ it follows that $|f(x)| \leq C_1|g(x)| \leq C_1C_2|h(x)|$. This shows that $f(x)$ is $O(h(x))$. **19.** 2^{n+1} is $O(2^n)$; 2^{2n} is not. **21.** $1000 \log n, \sqrt{n}, n \log n, n^2/1000000, 2^n, 3^n, 2n!$ **23.** The algorithm that uses $n \log n$ operations **25. a)** $O(n^3)$ **b)** $O(n^5)$ **c)** $O(n^3 \cdot n!)$ **27. a)** $O(n^2 \log n)$ **b)** $O(n^2(\log n)^2)$ **c)** $O(n^{2^n})$ **29. a)** Neither $\Theta(x^2)$ nor $\Omega(x^2)$ **b)** $\Theta(x^2)$ and $\Omega(x^2)$ **c)** Neither $\Theta(x^2)$ nor $\Omega(x^2)$ **d)** $\Omega(x^2)$, but not $\Theta(x^2)$ **e)** $\Omega(x^2)$, but not $\Theta(x^2)$ **f)** $\Omega(x^2)$ and $\Theta(x^2)$ **31.** If $f(x)$ is $\Theta(g(x))$, then there exist constants C_1 and C_2 with $C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$. It follows that $|f(x)| \leq C_2|g(x)|$ and $|g(x)| \leq (1/C_1)|f(x)|$ for $x > k$. Thus, $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$. Conversely, suppose that $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$. Then there are constants C_1, C_2, k_1 , and k_2 such that $|f(x)| \leq C_1|g(x)|$ for $x > k_1$ and $|g(x)| \leq C_2|f(x)|$ for $x > k_2$. We can assume that $C_2 > 0$ (we can always make C_2 larger). Then we have $(1/C_2)|g(x)| \leq |f(x)| \leq C_1|g(x)|$ for $x > \max(k_1, k_2)$. Hence, $f(x)$ is $\Theta(g(x))$. **33.** If $f(x)$ is $\Theta(g(x))$, then $f(x)$ is both $O(g(x))$ and $\Omega(g(x))$. Hence, there are positive constants C_1, k_1, C_2 , and k_2 such that $|f(x)| \leq C_2|g(x)|$ for all $x > k_2$ and $|f(x)| \geq C_1|g(x)|$ for all $x > k_1$. It follows that $C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$ whenever $x > k$, where $k = \max(k_1, k_2)$. Conversely, if there are positive constants C_1, C_2 , and k such that $C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$ for $x > k$, then taking $k_1 = k_2 = k$ shows that $f(x)$ is both $O(g(x))$ and $\Theta(g(x))$.



- 37.** If $f(x)$ is $\Theta(1)$, then $|f(x)|$ is bounded between positive constants C_1 and C_2 . In other words, $f(x)$ cannot grow larger than a fixed bound or smaller than the negative of this bound and must not get closer to 0 than some fixed bound. **39.** Because $f(x)$ is $O(g(x))$, there are constants C and k such that $|f(x)| \leq C|g(x)|$ for $x > k$. Hence, $|f^n(x)| \leq C^n|g^n(x)|$ for $x > k$, so $f^n(x)$ is $O(g^n(x))$ by taking the constant to be C^n . **41.** Because $f(x)$ and $g(x)$ are increasing and unbounded, we can assume $f(x) \geq 1$ and $g(x) \geq 1$ for sufficiently large x . There are constants C and k with $f(x) \leq Cg(x)$ for $x > k$. This implies that $\log f(x) \leq \log C + \log g(x) < 2 \log g(x)$ for sufficiently large x . Hence, $\log f(x)$ is $O(\log g(x))$. **43.** By definition there are positive constraints $C_1, C'_1, C_2, C'_2, k_1, k'_1, k_2$, and k'_2 such that $f_1(x) \geq C_1|g(x)|$ for all $x > k_1$, $f_1(x) \leq C'_1|g(x)|$ for all $x > k'_1$, $f_2(x) \geq C_2|g(x)|$ for all $x > k_2$, and $f_2(x) \leq C'_2|g(x)|$ for all $x > k'_2$. Adding the first and third inequalities shows that $f_1(x) + f_2(x) \geq (C_1 + C_2)|g(x)|$ for all $x > k$ where

$k = \max(k_1, k_2)$. Adding the second and fourth inequalities shows that $f_1(x) + f_2(x) \leq (C'_1 + C'_2)|g(x)|$ for all $x > k'$ where $k' = \max(k'_1, k'_2)$. Hence, $f_1(x) + f_2(x)$ is $\Theta(g(x))$. This is no longer true if f_1 and f_2 can assume negative values.

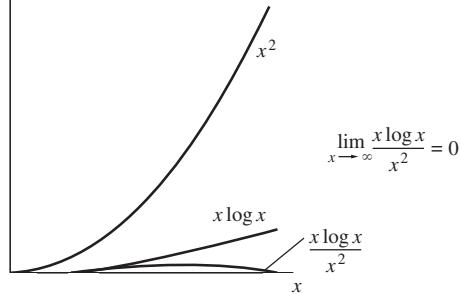
45. This is false. Let $f_1 = x^2 + 2x$, $f_2(x) = x^2 + x$, and $g(x) = x^2$. Then $f_1(x)$ and $f_2(x)$ are both $O(g(x))$, but $(f_1 - f_2)(x)$ is not. **47.** Take $f(n)$ to be the function with $f(n) = n$ if n is an odd positive integer and $f(n) = 1$ if n is an even positive integer and $g(n)$ to be the function with $g(n) = 1$ if n is an odd positive integer and $g(n) = n$ if n is an even positive integer. **49.** There are positive constants $C_1, C_2, C'_1, C'_2, k_1, k'_1, k_2$, and k'_2 such that $|f_1(x)| \geq C_1|g_1(x)|$ for all $x > k_1$, $|f_1(x)| \leq C'_1|g_1(x)|$ for all $x \geq k'_1$, $|f_2(x)| > C_2|g_2(x)|$ for all $x > k_2$, and $|f_2(x)| \leq C'_2|g_2(x)|$ for all $x > k'_2$. Because f_2 and g_2 are never zero, the last two inequalities can be rewritten as $|1/f_2(x)| \leq (1/C_2)|1/g_2(x)|$ for all $x > k_2$ and $|1/f_2(x)| \geq (1/C'_2)|1/g_2(x)|$ for all $x > k'_2$. Multiplying the first and rewritten fourth inequalities shows that $|f_1(x)/f_2(x)| \geq (C_1/C'_2)|g_1(x)/g_2(x)|$ for all $x > \max(k_1, k'_2)$, and multiplying the second and rewritten third inequalities gives $|f_1(x)/f_2(x)| \leq (C'_1/C_2)|g_1(x)/g_2(x)|$ for all $x > \max(k'_1, k_2)$. It follows that f_1/f_2 is big-Theta of g_1/g_2 .

51. There exist positive constants $C_1, C_2, k_1, k_2, k'_1, k'_2$ such that $|f(x, y)| \leq C_1|g(x, y)|$ for all $x > k_1$ and $y > k_2$ and $|f(x, y)| \geq C_2|g(x, y)|$ for all $x > k'_1$ and $y > k'_2$.

53. $(x^2 + xy + x \log y)^3 < (3x^2y^3) = 27x^6y^3$ for $x > 1$ and $y > 1$, because $x^2 < x^2y$, $xy < x^2y$, and $x \log y < x^2y$. Hence, $(x^2 + xy + x \log y)^3$ is $O(x^6y^3)$.

55. For all positive real numbers x and y , $\lfloor xy \rfloor \leq xy$. Hence, $\lfloor xy \rfloor$ is $O(xy)$ from the definition, taking $C = 1$ and $k_1 = k_2 = 0$. **57.** Clearly $n^d < n^c$ for all $n \geq 2$; therefore n^d is $O(n^c)$. The ratio $n^d/n^c = n^{d-c}$ is unbounded so there is no constant C such that $n^d \leq Cn^c$ for large n . **59.** If f and g are positive-valued functions such that $\lim_{n \rightarrow \infty} f(x)/g(x) = C < \infty$, then $f(x) < (C+1)g(x)$ for large enough x , so $f(n)$ is $O(g(n))$. If that limit is ∞ , then clearly $f(n)$ is not $O(g(n))$. Here repeated applications of L'Hôpital's rule shows that $\lim_{x \rightarrow \infty} x^d/b^x = 0$ and $\lim_{x \rightarrow \infty} b^x/x^d = \infty$. **61. a)** $\lim_{x \rightarrow \infty} x^2/x^3 = \lim_{x \rightarrow \infty} 1/x = 0$ **b)** $\lim_{x \rightarrow \infty} \frac{x \log x}{x^2} = \lim_{x \rightarrow \infty} \frac{\log x}{x} = \lim_{x \rightarrow \infty} \frac{1}{x \ln 2} = 0$ (using L'Hôpital's rule) **c)** $\lim_{x \rightarrow \infty} \frac{x^2}{2^x} = \lim_{x \rightarrow \infty} \frac{2x}{2^x \cdot (\ln 2)^2} = 0$ (using L'Hôpital's rule) **d)** $\lim_{x \rightarrow \infty} \frac{x^2+x+1}{x^2} = \lim_{x \rightarrow \infty} \left(1 + \frac{1}{x} + \frac{1}{x^2}\right) = 1 \neq 0$

63.



65. No. Take $f(x) = 1/x^2$ and $g(x) = 1/x$. **67. a)** Because $\lim_{x \rightarrow \infty} f(x)/g(x) = 0$, $|f(x)|/|g(x)| < 1$ for sufficiently large x . Hence, $|f(x)| < |g(x)|$ for $x > k$ for some constant k . Therefore, $f(x)$ is $O(g(x))$. **b)** Let $f(x) = g(x) = x$. Then $f(x)$ is $O(g(x))$, but $f(x)$ is not $o(g(x))$ because $f(x)/g(x) = 1$. **69.** Because $f_2(x)$ is $o(g(x))$, from Exercise 67(a) it follows that $f_2(x)$ is $O(g(x))$. By Corollary 1, we have $f_1(x) + f_2(x)$ is $O(g(x))$. **71.** We can easily show that $(n-i)(i+1) \geq n$ for $i = 0, 1, \dots, n-1$. Hence, $(n!)^2 = (n \cdot 1)((n-1) \cdot 2) \cdots ((n-2) \cdot 3) \cdots (2 \cdot (n-1)) \cdot (1 \cdot n) \geq n^n$. Therefore, $2 \log n! \geq n \log n$. **73.** Compute that $\log 5! \approx 6.9$ and $(5 \log 5)/4 \approx 2.9$, so the inequality holds for $n = 5$. Assume $n \geq 6$. Because $n!$ is the product of all the integers from n down to 1, we have $n! > n(n-1)(n-2) \cdots \lceil n/2 \rceil$ (because at least the term 2 is missing). Note that there are more than $n/2$ terms in this product, and each term is at least as big as $n/2$. Therefore the product is greater than $(n/2)^{(n/2)}$. Taking the log of both sides of the inequality, we have $\log n! > \log \left(\frac{n}{2}\right)^{n/2} = \frac{n}{2} \log \frac{n}{2} = \frac{n}{2}(\log n - 1) > (n \log n)/4$, because $n > 4$ implies $\log n - 1 > (\log n)/2$. **75.** All are not asymptotic.

Section 3.3

1. $O(1)$ **3.** $O(n^2)$ **5.** $2n - 1$ **7.** Linear **9.** $O(n)$

11. a) procedure disjointpair(S_1, S_2, \dots, S_n :

subsets of $\{1, 2, \dots, n\}$)

answer := false

for *i* := 1 **to** *n*

for *j* := *i* + 1 **to** *n*

disjoint := true

for *k* := 1 **to** *n*

if *k* ∈ *S_i* and *k* ∈ *S_j* **then** *disjoint* := false

if *disjoint* **then** *answer* := true

return *answer*

b) $O(n^3)$ **13. a)** *power* := 1, *y* := 1; *i* := 1, *power* := 2, *y* := 3; *i* := 2, *power* := 4, *y* := 15

b) $2n$ multiplications and n additions **15. a)** $2^{10^9} \approx 10^3 \cdot 10^8$

b) 10^9 **c)** 3.96×10^7 **d)** 3.16×10^4 **e)** 29 **f)** 12

17. a) $2^{2^{60 \cdot 10^{12}}}$ **b)** $2^{60 \cdot 10^{12}}$ **c)** $\lfloor 2^{\sqrt{60 \cdot 10^6}} \rfloor \approx 2 \times 10^{2331768}$

d) 60,000,000 **e)** 7,745,966 **f)** 45 **g)** 6 **19. a)** 36 years

b) 13 days **c)** 19 minutes **21. a)** Less than 1 millisecond more **b)** 100 milliseconds more **c)** $2n + 1$ milliseconds more **d)** $3n^2 + 3n + 1$ milliseconds more **e)** Twice as much time **f)** 2^{2n+1} times as many milliseconds **g)** $n + 1$ times as many milliseconds

23. The average number of comparisons is $(3n+4)/2$. **25.** $O(\log n)$ **27.** $O(n)$ **29.** $O(n^2)$

31. $O(n)$ **33.** $O(n)$ **35.** $O(\log n)$ comparisons; $O(n^2)$ swaps **37.** $O(n^2 2^n)$ **39. a)** doubles **b)** increases by 1

41. Use Algorithm 1, where **A** and **B** are now $n \times n$ upper triangular matrices, by replacing m by n in line 1, and

having q iterate only from i to j , rather than from 1 to k .

43. $n(n + 1)(n + 2)/6$ **45.** A(BC)D

Supplementary Exercises

1. a) **procedure** *last max*(a_1, \dots, a_n : integers)

```
max :=  $a_1$ 
last := 1
i := 2
while  $i \leq n$ 
  if  $a_i \geq max$  then
    max :=  $a_i$ 
    last := i
  i :=  $i + 1$ 
return last
```

b) $2n - 1 = O(n)$ comparisons

3. a) **procedure** *pair zeros*($b_1 b_2 \dots b_n$: bit string, $n \geq 2$)

```
x :=  $b_1$ 
y :=  $b_2$ 
k := 2
while  $k < n$  and ( $x \neq 0$  or  $y \neq 0$ )
  k :=  $k + 1$ 
  x := y
  y :=  $b_k$ 
if  $x = 0$  and  $y = 0$  then print "YES"
  else print "NO"
```

b) $O(n)$

5. a) and **b)**

procedure *smallest and largest*(a_1, a_2, \dots, a_n : integers)

```
min :=  $a_1$ 
max :=  $a_1$ 
for  $i := 2$  to  $n$ 
  if  $a_i < min$  then min :=  $a_i$ 
  if  $a_i > max$  then max :=  $a_i$ 
{min is the smallest integer among the input, and max is the largest}
```

c) $2n - 2$

7. Before any comparisons are done, there is a possibility that each element could be the maximum and a possibility that it could be the minimum. This means that there are $2n$ different possibilities, and $2n - 2$ of them have to be eliminated through comparisons of elements, because we need to find the unique maximum and the unique minimum. We classify comparisons of two elements as “virgin” or “nonvirgin,” depending on whether or not both elements being compared have been in any previous comparison. A virgin comparison eliminates the possibility that the larger one is the minimum and that the smaller one is the maximum; thus each virgin comparison eliminates two possibilities, but it clearly cannot do more. A nonvirgin comparison must be between two elements that are still in the running to be the maximum or two elements that are still in the running to be the minimum, and at least one of these elements must *not* be in the running for

the other category. For example, we might be comparing x and y , where all we know is that x has been eliminated as the minimum. If we find that $x > y$ in this case, then only one possibility has been ruled out—we now know that y is not the maximum. Thus in the worst case, a nonvirgin comparison eliminates only one possibility. (The cases of other nonvirgin comparisons are similar.) Now there are at most $\lfloor n/2 \rfloor$ comparisons of elements that have not been compared before, each removing two possibilities; they remove $2\lfloor n/2 \rfloor$ possibilities altogether. Therefore we need $2n - 2 - 2\lfloor n/2 \rfloor$ more comparisons that, as we have argued, can remove only one possibility each, in order to find the answers in the worst case, because $2n - 2$ possibilities have to be eliminated. This gives us a total of $2n - 2 - 2\lfloor n/2 \rfloor + \lfloor n/2 \rfloor$ comparisons in all. But $2n - 2 - 2\lfloor n/2 \rfloor + \lfloor n/2 \rfloor = 2n - 2 - \lfloor n/2 \rfloor = 2n - 2 + \lceil -n/2 \rceil = \lceil 2n - n/2 \rceil - 2 = \lceil 3n/2 \rceil - 2$, as desired.

9. The following algorithm has worst-case complexity $O(n^4)$.

procedure *equal sums*(a_1, a_2, \dots, a_n)

```
for  $i := 1$  to  $n$ 
  for  $j := i + 1$  to  $n$  {since we want  $i < j$ }
    for  $k := 1$  to  $n$ 
      for  $l := k + 1$  to  $n$  {since we want  $k < l$ }
        if  $a_i + a_j = a_k + a_l$  and  $(i, j) \neq (k, l)$ 
          then output these pairs
```

11. At end of first pass: 3, 1, 4, 5, 2, 6; at end of second pass: 1, 3, 2, 4, 5, 6; at end of third pass: 1, 2, 3, 4, 5, 6; fourth pass finds nothing to exchange and algorithm terminates

13. There are possibly as many as n passes through the list, and each pass uses $O(n)$ comparisons. Thus there are $O(n^2)$ comparisons in all. **15.** Because $\log n < n$, we have $(n \log n + n^2)^3 \leq (n^2 + n^2)^3 \leq (2n^2)^3 = 8n^6$ for all $n > 0$. This proves that $(n \log n + n^2)^3$ is $O(n^6)$, with witnesses $C = 8$ and $k = 0$. **17.** $O(x^{22^x})$ **19.** Note that $\frac{n!}{2^n} = \frac{n}{2} \cdot \frac{n-1}{2} \cdots \frac{3}{2} \cdot \frac{2}{2} \cdot \frac{1}{2} > \frac{n}{2} \cdot 1 \cdot 1 \cdots 1 \cdot \frac{1}{2} = \frac{n}{4}$. **21.** All of these functions are of the same order. **23.** 2^{10^7} **25.** $(\log n)^2$, $2\sqrt{\log_2 n}$, $n(\log n)^{1001}$, $n^{1.0001}$, 1.0001^n , n^n **27.** For example, $f(n) = n^{2\lfloor n/2 \rfloor + 1}$ and $g(n) = n^{2\lceil n/2 \rceil}$

29. a)

procedure *brute*(a_1, a_2, \dots, a_n : integers)

```
for  $i := 1$  to  $n - 1$ 
  for  $j := i + 1$  to  $n$ 
    for  $k := 1$  to  $n$ 
      if  $a_i + a_j = a_k$  then return true else return false
```

b) $O(n^3)$

31. For m_1 : w_1 and w_2 ; for m_2 : w_1 and w_3 ; for m_3 : w_2 and w_3 ; for w_1 : m_1 and m_2 ; for w_2 : m_1 and m_3 ; for w_3 : m_2 and m_3

33. A matching in which each woman is assigned her valid partner ranking highest on her preference list is female optimal; a matching in which each man is assigned his valid partner ranking lowest on his preference list is male pessimal. **35. a)** Modify the preamble to Exercise 60 in Section 3.1 so that there are s men m_1, m_2, \dots, m_s and t women w_1, w_2, \dots, w_t . A matching will contain $\min(s, t)$ marriages. The definition of “stable marriage” is the same, with the understanding that each person prefers any mate to being unmatched. **b)** Create $|s - t|$ fictitious people (men or women,

whichever is in shorter supply) so that the number of men and the number of women become the same, and put these fictitious people at the bottom of everyone's preference lists. **c)** This follows immediately from Exercise 63 in Section 3.1. **37.** 5; 15 **39.** The first situation in Exercise 37 **41. a)** For each subset S of $\{1, 2, \dots, n\}$, compute $\sum_{j \in S} w_j$. Keep track of the subset giving the largest such sum that is less than or equal to W , and return that subset as the output of the algorithm. **b)** The food pack and the portable stove **43. a)** The makespan is always at least as large as the load on the processor assigned to do the lengthiest job, which must be at least $\max_{j=1,2,\dots,n} t_j$. Therefore the minimum makespan satisfies this inequality. **b)** The total amount of time the processors need to spend working on the jobs (the total load) is $\sum_{j=1}^n t_j$. Therefore the average load per processor is $\frac{1}{p} \sum_{j=1}^n t_j$. The maximum load cannot be any smaller than the average, so the minimum makespan is always at least this large. **45.** Processor 1: jobs 1, 4; processor 2: job 2; processor 3: jobs 3, 5

CHAPTER 4

Section 4.1

1. a) Yes **b)** No **c)** Yes **d)** No **3.** Suppose that $a \mid b$. Then there exists an integer k such that $ka = b$. Because $a(ck) = bc$ it follows that $a \mid bc$. **5.** If $a \mid b$ and $b \mid a$, there are integers c and d such that $b = ac$ and $a = bd$. Hence, $a = acd$. Because $a \neq 0$ it follows that $cd = 1$. Thus either $c = d = 1$ or $c = d = -1$. Hence, either $a = b$ or $a = -b$. **7.** Because $ac \mid bc$ there is an integer k such that $ack = bc$. Hence, $ak = b$, so $a \mid b$. **9. a)** 2, 5 **b)** -11, 10 **c)** 34, 7 **d)** 77, 0 **e)** 0, 0 **f)** 0, 3 **g)** -1, 2 **h)** 4, 0 **11. a)** 7:00 **b)** 8:00 **c)** 10:00 **13. a)** 10 **b)** 8 **c)** 0 **d)** 9 **e)** 6 **f)** 11 **15.** If $a \text{ mod } m = b \text{ mod } m$, then a and b have the same remainder when divided by m . Hence, $a = q_1m + r$ and $b = q_2m + r$, where $0 \leq r < m$. It follows that $a - b = (q_1 - q_2)m$, so $m \mid (a - b)$. It follows that $a \equiv b \pmod{m}$. **17.** There is some b with $(b - 1)k < n \leq bk$. Hence, $(b - 1)k \leq n - 1 < bk$. Divide by k to obtain $b - 1 < n/k \leq b$ and $b - 1 \leq (n - 1)/k < b$. Hence, $\lceil n/k \rceil = b$ and $\lfloor (n - 1)/k \rfloor = b - 1$. **19. x mod m** if $x \text{ mod } m \leq \lceil m/2 \rceil$ and $(x \text{ mod } m) - m$ if $x \text{ mod } m > \lceil m/2 \rceil$ **21. a)** 1 **b)** 2 **c)** 3 **d)** 9 **23. a)** 1, 109 **b)** 40, 89 **c)** -31, 222 **d)** -21, 38259 **25. a)** -15 **b)** -7 **c)** 140 **27.** -1, -26, -51, -76, 24, 49, 74, 99 **29. a)** No **b)** No **c)** Yes **d)** No **31. a)** 13 **a)** 6 **33. a)** 9 **b)** 4 **c)** 25 **d)** 0 **35.** Let $m = tn$. Because $a \equiv b \pmod{m}$ there exists an integer s such that $a = b + sm$. Hence, $a = b + (st)n$, so $a \equiv b \pmod{n}$. **37. a)** Let $m = c = 2$, $a = 0$, and $b = 1$. Then $0 = ac \equiv bc = 2 \pmod{2}$, but $0 = a \not\equiv b = 1 \pmod{2}$. **b)** Let $m = 5$, $a = b = 3$, $c = 1$, and $d = 6$. Then $3 \equiv 3 \pmod{5}$ and $1 \equiv 6 \pmod{5}$, but $3^1 = 3 \not\equiv 4 \equiv 729 = 3^6 \pmod{5}$. **39.** By Exercise 38 the sum of two squares must be either $0 + 0 = 0$, $0 + 1 = 1$, or $1 + 1 = 2$, modulo 4, never 3, and therefore not of the form $4k + 3$. **41.** Because $a \equiv b \pmod{m}$, there exists an

integer s such that $a = b + sm$, so $a - b = sm$. Then $a^k - b^k = (a - b)(a^{k-1} + a^{k-2}b + \dots + ab^{k-2} + b^{k-1})$, $k \geq 2$, is also a multiple of m . It follows that $a^k \equiv b^k \pmod{m}$.

43. To prove closure, note that $a \cdot_m b = (a \cdot b) \text{ mod } m$, which by definition is an element of \mathbf{Z}_m . Multiplication is associative because $(a \cdot_m b) \cdot_m c$ and $a \cdot_m (b \cdot_m c)$ both equal $(a \cdot b \cdot c) \text{ mod } m$ and multiplication of integers is associative. Similarly, multiplication in \mathbf{Z}_m is commutative because multiplication in \mathbf{Z} is commutative, and 1 is the multiplicative identity for \mathbf{Z}_m because 1 is the multiplicative identity for \mathbf{Z} .

45. $0+50 = 0$, $0+51 = 1$, $0+52 = 2$, $0+53 = 3$, $0+54 = 4$; $1+51 = 2$, $1+52 = 3$, $1+53 = 4$, $1+54 = 0$; $2+52 = 4$, $2+53 = 0$, $2+54 = 1$; $3+53 = 1$, $3+54 = 2$; $4+44 = 3$ and $0 \cdot 50 = 0$, $0 \cdot 51 = 0$, $0 \cdot 52 = 0$, $0 \cdot 53 = 0$, $0 \cdot 54 = 0$; $1 \cdot 51 = 1$, $1 \cdot 52 = 2$, $1 \cdot 53 = 3$, $1 \cdot 54 = 4$; $2 \cdot 52 = 4$, $2 \cdot 53 = 1$, $2 \cdot 54 = 3$; $3 \cdot 53 = 4$, $3 \cdot 54 = 2$; $4 \cdot 54 = 1$ **47.** f is onto but not one-to-one (unless $d = 1$); g is neither.

Section 4.2

1. a) 1110 0111 **b)** 1 0001 1011 0100 **c)** 1 0111 11010110 1100 **3. a)** 31 **b)** 513 **c)** 341 **d)** 26,896 **5. a)** 1 0111 1010 **b)** 11 1000 0100 **c)** 1 0001 0011 **d)** 101 0000 1111 **7. a)** 1000 0000 1110 **b)** 1 0011 0101 1010 1011 **c)** 10101011 1011 1010 **d)** 1101 1110 1111 1010 11001110 1101 **9.** 1010 1011 1100 1101 1110 1111 **11.** (B7B)₁₆

13. Adding up to three leading 0s if necessary, write the binary expansion as $(\dots b_{23}b_{22}b_{21}b_{20}b_{13}b_{12}b_{11}b_{10}b_{03}b_{02}b_{01}b_{00})_2$. The value of this numeral is $b_{00} + 2b_{01} + 4b_{02} + 8b_{03} + 2^4b_{10} + 2^5b_{11} + 2^6b_{12} + 2^7b_{13} + 2^8b_{20} + 2^9b_{21} + 2^{10}b_{22} + 2^{11}b_{23} + \dots$, which we can rewrite as $b_{00} + 2b_{01} + 4b_{02} + 8b_{03} + (b_{10} + 2b_{11} + 4b_{12} + 8b_{13}) \cdot 2^4 + (b_{20} + 2b_{21} + 4b_{22} + 8b_{23}) \cdot 2^8 + \dots$. Now $(b_{i3}b_{i2}b_{i1}b_{i0})_2$ translates into the hexadecimal digit h_i . So our number is $h_0 + h_1 \cdot 2^4 + h_2 \cdot 2^8 + \dots = h_0 + h_1 \cdot 16 + h_2 \cdot 16^2 + \dots$, which is the hexadecimal expansion $(\dots h_1h_1h_0)_{16}$. **15.** Adding up to two leading 0s if necessary, write the binary expansion as $(\dots b_{22}b_{21}b_{20}b_{12}b_{11}b_{10}b_{02}b_{01}b_{00})_2$. The value of this numeral is $b_{00} + 2b_{01} + 4b_{02} + 2^3b_{10} + 2^4b_{11} + 2^5b_{12} + 2^6b_{20} + 2^7b_{21} + 2^8b_{22} + \dots$, which we can rewrite as $b_{00} + 2b_{01} + 4b_{02} + (b_{10} + 2b_{11} + 4b_{12}) \cdot 2^3 + (b_{20} + 2b_{21} + 4b_{22}) \cdot 2^6 + \dots$. Now $(b_{i2}b_{i1}b_{i0})_2$ translates into the octal digit h_i . So our number is $h_0 + h_1 \cdot 2^3 + h_2 \cdot 2^6 + \dots = h_0 + h_1 \cdot 8 + h_2 \cdot 8^2 + \dots$, which is the octal expansion $(\dots h_1h_1h_0)_8$. **17.** 1 1101 1100 1010 1101 0001, 1273₈ **19.** Convert the given octal numeral to binary, then convert from binary to hexadecimal using Example 7. **21. a)** 1011 1110, 10 0001 0000 0001 **b)** 1 1010 1100, 1011 0000 0111 0011 **c)** 100 1001 1010, 101 0010 1001 0110 0000 **d)** 110 0000 1000 0000 0001 1111 1111 **23. a)** 1132, 144,305 **b)** 6273, 2,134,272 **c)** 2110, 1,107,667 **d)** 57,777, 237,326,216 **25. 436 27. 27 29.** The binary expansion of the integer is the unique such sum. **31.** Let $a = (a_{n-1}a_{n-2}\dots a_1a_0)_{10}$. Then $a = 10^{n-1}a_{n-1} + 10^{n-2}a_{n-2} + \dots + 10a_1 + a_0 \equiv a_{n-1} + a_{n-2} + \dots + a_1 + a_0 \pmod{3}$, because

$10^j \equiv 1 \pmod{3}$ for all nonnegative integers j . It follows that $3 \mid a$ if and only if 3 divides the sum of the decimal digits of a . **33.** Let $a = (a_{n-1}a_{n-2}\dots a_1a_0)_2$. Then $a = a_0 + 2a_1 + 2^2a_2 + \dots + 2^{n-1}a_{n-1} \equiv a_0 - a_1 + a_2 - a_3 + \dots \pm a_{n-1} \pmod{3}$. It follows that a is divisible by 3 if and only if the sum of the binary digits in the even-numbered positions minus the sum of the binary digits in the odd-numbered positions is divisible by 3. **35. a) -6 b) 13 c) -14 d) 0** **37.** The one's complement of the sum is found by adding the one's complements of the two integers except that a carry in the leading bit is used as a carry to the last bit of the sum. **39.** If $m \geq 0$, then the leading bit a_{n-1} of the one's complement expansion of m is 0 and the formula reads $m = \sum_{i=0}^{n-2} a_i 2^i$. This is correct because the right-hand side is the binary expansion of m . When m is negative, the leading bit a_{n-1} of the one's complement expansion of m is 1. The remaining $n-1$ bits can be obtained by subtracting $-m$ from $111\dots 1$ (where there are $n-1$ 1s), because subtracting a bit from 1 is the same as complementing it. Hence, the bit string $a_{n-2}\dots a_0$ is the binary expansion of $(2^{n-1} - 1) - (-m)$. Solving the equation $(2^{n-1} - 1) - (-m) = \sum_{i=0}^{n-2} a_i 2^i$ for m gives the desired equation because $a_{n-1} = 1$. **41. a) -7 b) 13 c) -15 d) -1** **43.** To obtain the two's complement representation of the sum of two integers, add their two's complement representations (as binary integers are added) and ignore any carry out of the leftmost column. However, the answer is invalid if an overflow has occurred. This happens when the leftmost digits in the two's complement representation of the two terms agree and the leftmost digit of the answer differs. **45.** If $m \geq 0$, then the leading bit a_{n-1} is 0 and the formula reads $m = \sum_{i=0}^{n-2} a_i 2^i$. This is correct because the right-hand side is the binary expansion of m . If $m < 0$, its two's complement expansion has 1 as its leading bit and the remaining $n-1$ bits are the binary expansion of $2^{n-1} - (-m)$. This means that $(2^{n-1}) - (-m) = \sum_{i=0}^{n-2} a_i 2^i$. Solving for m gives the desired equation because $a_{n-1} = 1$. **47. 4n**

49. procedure Cantor(x : positive integer)

```

 $n := 1; f := 1$ 
while  $(n+1) \cdot f \leq x$ 
   $n := n + 1$ 
   $f := f \cdot n$ 
   $y := x$ 
  while  $n > 0$ 
     $a_n := \lfloor y/f \rfloor$ 
     $y := y - a_n \cdot f$ 
     $f := f/n$ 
     $n := n - 1$ 
   $x = a_n n! + a_{n-1}(n-1)! + \dots + a_1 1!$ 
}

```

51. First step: $c = 0, d = 0, s_0 = 1$; second step: $c = 0, d = 1, s_1 = 0$; third step: $c = 1, d = 1, s_2 = 0$; fourth step: $c = 1, d = 1, s_3 = 0$; fifth step: $c = 1, d = 1, s_4 = 1$; sixth step: $c = 1, s_5 = 1$

53. procedure subtract(a, b : positive integers, $a > b$,

```

 $a = (a_{n-1}a_{n-2}\dots a_1a_0)_2,$ 
 $b = (b_{n-1}b_{n-2}\dots b_1b_0)_2)$ 
 $B := 0 \{B \text{ is the borrow}\}$ 
for  $j := 0$  to  $n-1$ 
  if  $a_j \geq b_j + B$  then
     $s_j := a_j - b_j - B$ 
     $B := 0$ 
  else
     $s_j := a_j + 2 - b_j - B$ 
     $B := 1$ 
   $\{(s_{n-1}s_{n-2}\dots s_1s_0)_2 \text{ is the difference}\}$ 
}

```

55. procedure compare(a, b : positive integers,

```

 $a = (a_n a_{n-1}\dots a_1 a_0)_2,$ 
 $b = (b_n b_{n-1}\dots b_1 b_0)_2)$ 
 $k := n$ 
while  $a_k = b_k$  and  $k > 0$ 
   $k := k - 1$ 
if  $a_k = b_k$  then print "a equals b"
if  $a_k > b_k$  then print "a is greater than b"
if  $a_k < b_k$  then print "a is less than b"
}

```

57. $O(\log n)$ **59.** The only time-consuming part of the algorithm is the **while** loop, which is iterated q times. The work done inside is a subtraction of integers no bigger than a , which has $\log a$ bits. The result now follows from Example 9.

Section 4.3

- 1.** 29, 71, 97 prime; 21, 111, 143 not prime **3. a) $2^3 \cdot 11$ b) $2 \cdot 3^2 \cdot 7$ c) 3^6 d) $7 \cdot 11 \cdot 13$ e) $11 \cdot 101$ f) $2 \cdot 3^3 \cdot 5 \cdot 7 \cdot 13 \cdot 37$ 5. $2^8 \cdot 3^4 \cdot 5^2 \cdot 7$**

7. procedure primetest(n : integer greater than 1)
 $isprime := \text{true}$
 $d := 2$
while $isprime$ and $d \leq \sqrt{n}$
if $n \bmod d = 0$ **then** $isprime := \text{false}$
else $d := d + 1$
return $isprime$

9. Write $n = rs$, where $r > 1$ and $s > 1$. Then $2^n - 1 = 2^{rs} - 1 = (2^r)^s - 1 = (2^r - 1)((2^r)^{s-1} + (2^r)^{s-2} + (2^r)^{s-3} + \dots + 1)$. The first factor is at least $2^2 - 1 = 3$ and the second factor is at least $2^2 + 1 = 5$. This provides a factoring of $2^n - 1$ into two factors greater than 1, so $2^n - 1$ is composite.

11. Suppose that $\log_2 3 = a/b$ where $a, b \in \mathbb{Z}^+$ and $b \neq 0$. Then $2^{a/b} = 3$, so $2^a = 3^b$. This violates the fundamental theorem of arithmetic. Hence, $\log_2 3$ is irrational. **13.** 3, 5, and 7 are primes of the desired form. **15.** 1, 7, 11, 13, 17, 19, 23, 29 **17. a) Yes b) No c) Yes d) Yes** **19.** Suppose that n is not prime, so that $n = ab$, where a and b are integers greater than 1. Because $a > 1$, by the identity in the hint, $2^a - 1$ is a factor of $2^n - 1$ that is greater than 1, and the second

factor in this identity is also greater than 1. Hence, $2^n - 1$ is not prime. **21.** a) 2 b) 4 c) 12 **23.** $\phi(p^k) = p^k - p^{k-1}$ **25.** a) $3^5 \cdot 5^3$ b) 1 c) 23^{17} d) $41 \cdot 43 \cdot 53$ e) 1 f) 1111 **27.** a) $2^{11} \cdot 3^7 \cdot 5^9 \cdot 7^3$ b) $2^9 \cdot 3^7 \cdot 5^5 \cdot 7^3 \cdot 11 \cdot 13 \cdot 17$ c) 23^{31} d) $41 \cdot 43 \cdot 53$ e) $2^{12}3^{13}5^{17}7^{21}$ f) Undefined **29.** $\gcd(92928, 123552) = 1056$; $\text{lcm}(92928, 123552) = 10,872,576$; both products are 11,481,440,256. **31.** Because $\min(x, y) + \max(x, y) = x + y$, the exponent of p_i in the prime factorization of $\gcd(a, b) \cdot \text{lcm}(a, b)$ is the sum of the exponents of p_i in the prime factorizations of a and b . **33.** a) 6 b) 3 c) 11 d) 3 e) 40 f) 12 **35.** 9 **37.** By Exercise 36 it follows that $\gcd(2^b - 1, (2^a - 1) \bmod (2^b - 1)) = \gcd(2^b - 1, 2^a \bmod b - 1)$. Because the exponents involved in the calculation are b and $a \bmod b$, the same as the quantities involved in computing $\gcd(a, b)$, the steps used by the Euclidean algorithm to compute $\gcd(2^a - 1, 2^b - 1)$ run in parallel to those used to compute $\gcd(a, b)$ and show that $\gcd(2^a - 1, 2^b - 1) = 2^{\gcd(a,b)} - 1$. **39.** a) 1 = $(-1) \cdot 10 + 1 \cdot 11$ b) 1 = $21 \cdot 21 + (-10) \cdot 44$ c) 12 = $(-1) \cdot 36 + 48$ d) 1 = $13 \cdot 55 + (-21) \cdot 34$ e) 3 = $11 \cdot 213 + (-20) \cdot 117$ f) 223 = $1 \cdot 0 + 1 \cdot 223$ g) 1 = $37 \cdot 2347 + (-706) \cdot 123$ h) 2 = $1128 \cdot 3454 + (-835) \cdot 4666$ i) 1 = $2468 \cdot 9999 + (-2221) \cdot 11111$ **41.** $(-3) \cdot 26 + 1 \cdot 91 = 13$ **43.** $34 \cdot 144 + (-55) \cdot 89 = 1$

45. procedure extended Euclidean(a, b : positive integers)

```

 $x := a$ 
 $y := b$ 
 $oldolds := 1$ 
 $olds := 0$ 
 $oldoldt := 0$ 
 $oldt := 1$ 
while  $y \neq 0$ 
   $q := x \bmod y$ 
   $r := x - q \cdot y$ 
   $x := y$ 
   $y := r$ 
   $s := oldolds - q \cdot olds$ 
   $t := oldoldt - q \cdot oldt$ 
   $oldolds := olds$ 
   $oldoldt := oldt$ 
   $olds := s$ 
   $oldt := t$ 
{ $\gcd(a, b)$  is  $x$ , and  $(oldolds)a + (oldoldt)b = x$ }
```

47. a) $a_n = 1$ if n is prime and $a_n = 0$ otherwise. b) a_n is the smallest prime factor of n with $a_1 = 1$. c) a_n is the number of positive divisors of n . d) $a_n = 1$ if n has no divisors that are perfect squares greater than 1 and $a_n = 0$ otherwise. e) a_n is the largest prime less than or equal to n . f) a_n is the product of the first $n - 1$ primes. **49.** Because every second integer is divisible by 2, the product is divisible by 2. Because every third integer is divisible by 3, the product is divisible by 3. Therefore the product has both 2 and 3 in its prime factorization and is therefore divisible by $3 \cdot 2 = 6$. **51.** $n = 1601$ is a counterexample. **53** Setting $k = a + b + 1$ will produce the composite number $a(a + b + 1) + b = a^2 + ab + a + b = (a + 1)(a + b)$.

55. Suppose that there are only finitely many primes of the form $4k + 3$, namely q_1, q_2, \dots, q_n , where $q_1 = 3, q_2 = 7$, and so on. Let $Q = 4q_1q_2 \cdots q_n - 1$. Note that Q is of the form $4k + 3$ (where $k = q_1q_2 \cdots q_n - 1$). If Q is prime, then we have found a prime of the desired form different from all those listed. If Q is not prime, then Q has at least one prime factor not in the list q_1, q_2, \dots, q_n , because the remainder when Q is divided by q_j is $q_j - 1$, and $q_j - 1 \neq 0$. Because all odd primes are either of the form $4k + 1$ or of the form $4k + 3$, and the product of primes of the form $4k + 1$ is also of this form (because $(4k+1)(4m+1) = 4(4km+k+m)+1$), there must be a factor of Q of the form $4k + 3$ different from the primes we listed. **57.** Given a positive integer x , we show that there is exactly one positive rational number m/n (in lowest terms) such that $K(m/n) = x$. From the prime factorization of x , read off the m and n such that $K(m/n) = x$. The primes that occur to even powers are the primes that occur in the prime factorization of m , with the exponents being half the corresponding exponents in x ; and the primes that occur to odd powers are the primes that occur in the prime factorization of n , with the exponents being half of one more than the exponents in x .

Section 4.4

1. $15 \cdot 7 = 105 \equiv 1 \pmod{26}$ **3.** 7 **5.** a) 7 b) 52 c) 34 d) 73 **7.** Suppose that b and c are both inverses of a modulo m . Then $ba \equiv 1 \pmod{m}$ and $ca \equiv 1 \pmod{m}$. Hence, $ba \equiv ca \pmod{m}$. Because $\gcd(a, m) = 1$ it follows by Theorem 7 in Section 4.3 that $b \equiv c \pmod{m}$. **9.** 8 **11.** a) 67 b) 88 c) 146 **13.** 3 and 6 **15.** Let $m' = m/\gcd(c, m)$. Because all the common factors of m and c are divided out of m to obtain m' , it follows that m' and c are relatively prime. Because m divides $ac - bc = (a - b)c$, it follows that m' divides $(a - b)c$. By Lemma 3 in Section 4.3, we see that m' divides $a - b$, so $a \equiv b \pmod{m'}$. **17.** Suppose that $x^2 \equiv 1 \pmod{p}$. Then p divides $x^2 - 1 = (x + 1)(x - 1)$. By Lemma 2 it follows that $p \mid x + 1$ or $p \mid x - 1$, so $x \equiv -1 \pmod{p}$ or $x \equiv 1 \pmod{p}$. **19.** a) Suppose that $ia \equiv ja \pmod{p}$, where $1 \leq i < j < p$. Then p divides $ja - ia = a(j - i)$. By Theorem 1, because a is not divisible by p , p divides $j - i$, which is impossible because $j - i$ is a positive integer less than p . b) By part (a), because no two of $a, 2a, \dots, (p-1)a$ are congruent modulo p , each must be congruent to a different number from 1 to $p - 1$. It follows that $a \cdot 2a \cdot 3a \cdots (p-1)a \equiv 1 \cdot 2 \cdot 3 \cdots (p-1) \pmod{p}$. It follows that $(p-1)! \cdot a^{p-1} \equiv p-1 \pmod{p}$. c) By Wilson's theorem and part (b), if p does not divide a , it follows that $(-1) \cdot a^{p-1} \equiv -1 \pmod{p}$. Hence, $a^{p-1} \equiv 1 \pmod{p}$. d) If $p \mid a$, then $p \mid a^p$. Hence, $a^p \equiv a \equiv 0 \pmod{p}$. If p does not divide a , then $a^{p-1} \equiv a \pmod{p}$, by part (c). Multiplying both sides of this congruence by a gives $a^p \equiv a \pmod{p}$. **21.** All integers of the form $323 + 330k$, where k is an integer **23.** All integers of the form $53 + 60k$, where k is an integer

25. procedure *chinese*(m_1, m_2, \dots, m_n : relatively prime positive integers ; a_1, a_2, \dots, a_n : integers)

```

m := 1
for k := 1 to n
  m := m ·  $m_k$ 
for k := 1 to n
   $M_k := m/m_k$ 
   $y_k := M_k^{-1} \bmod m_k$ 
  x := 0
for k := 1 to n
  x := x +  $a_k M_k y_k$ 
while x ≥ m
  x := x - m
return x {the smallest solution to the system
  { $x \equiv a_k \pmod{m_k}, k = 1, 2, \dots, n$ }}
```

- 27.** All integers of the form $16 + 252k$, where k is an integer. **29.** Suppose that p is a prime appearing in the prime factorization of $m_1 m_2 \cdots m_n$. Because the m_i s are relatively prime, p is a factor of exactly one of the m_i s, say m_j . Because m_j divides $a - b$, it follows that $a - b$ has the factor p in its prime factorization to a power at least as large as the power to which it appears in the prime factorization of m_j . It follows that $m_1 m_2 \cdots m_n$ divides $a - b$, so $a \equiv b \pmod{m_1 m_2 \cdots m_n}$. **31.** $x \equiv 1 \pmod{6}$ **33.** 7 **35.** $a^{p-2} \cdot a = a \cdot a^{p-2} = a^{p-1} \equiv 1 \pmod{p}$ **37.** a) By Fermat's little theorem, we have $2^{10} \equiv 1 \pmod{11}$. Hence, $2^{340} = (2^{10})^{34} \equiv 1^{34} = 1 \pmod{11}$. b) Because $32 \equiv 1 \pmod{31}$, it follows that $2^{340} = (2^5)^{68} = 32^{68} \equiv 1^{68} = 1 \pmod{31}$. c) Because 11 and 31 are relatively prime, and $11 \cdot 31 = 341$, it follows by parts (a) and (b) and Exercise 29 that $2^{340} \equiv 1 \pmod{341}$. **39.** a) 3, 4, 8 b) 983 **41.** Suppose that q is an odd prime with $q \mid 2^p - 1$. By Fermat's little theorem, $q \mid 2^{q-1} - 1$. From Exercise 37 in Section 4.3, $\gcd(2^p - 1, 2^{q-1} - 1) = 2^{\gcd(p, q-1)} - 1$. Because q is a common divisor of $2^p - 1$ and $2^{q-1} - 1$, $\gcd(2^p - 1, 2^{q-1} - 1) > 1$. Hence, $\gcd(p, q-1) = p$, because the only other possibility, namely, $\gcd(p, q-1) = 1$, gives us $\gcd(2^p - 1, 2^{q-1} - 1) = 1$. Hence, $p \mid q - 1$, and therefore there is a positive integer m such that $q - 1 = mp$. Because q is odd, m must be even, say, $m = 2k$, and so every prime divisor of $2^p - 1$ is of the form $2kp + 1$. Furthermore, the product of numbers of this form is also of this form. Therefore, all divisors of $2^p - 1$ are of this form. **43.** M_{11} is not prime; M_{17} is prime. **45.** First, $2047 = 23 \cdot 89$ is composite. Write $2047 - 1 = 2046 = 2 \cdot 1023$, so $s = 1$ and $t = 1023$ in the definition. Then $2^{1023} = (2^{11})^{93} = 2048^{93} \equiv 1^{93} = 1 \pmod{2047}$, as desired. **47.** We must show that $b^{2820} \equiv 1 \pmod{2821}$ for all b relatively prime to 2821. Note that $2821 = 7 \cdot 13 \cdot 31$, and if $\gcd(b, 2821) = 1$, then $\gcd(b, 7) = \gcd(b, 13) = \gcd(b, 31) = 1$. Using Fermat's little theorem we find that $b^6 \equiv 1 \pmod{7}$, $b^{12} \equiv 1 \pmod{13}$, and $b^{30} \equiv 1 \pmod{31}$. It follows that $b^{2820} \equiv (b^6)^{470} \equiv 1 \pmod{7}$, $b^{2820} \equiv (b^{12})^{235} \equiv 1 \pmod{13}$, and $b^{2820} \equiv (b^{30})^{94} \equiv 1 \pmod{31}$. By Exercise 29 (or the Chinese remainder theorem) it follows that $b^{2820} \equiv 1 \pmod{2821}$, as desired. **49.** a) If we multiply out this expression, we get

$n = 1296m^3 + 396m^2 + 36m + 1$. Clearly $6m \mid n - 1$, $12m \mid n - 1$, and $18m \mid n - 1$. Therefore, the conditions of Exercise 48 are met, and we conclude that n is a Carmichael number. b) Letting $m = 51$ gives $n = 172,947,529$. **51.** 0 = (0, 0), 1 = (1, 1), 2 = (2, 2), 3 = (0, 3), 4 = (1, 4), 5 = (2, 0), 6 = (0, 1), 7 = (1, 2), 8 = (2, 3), 9 = (0, 4), 10 = (1, 0), 11 = (2, 1), 12 = (0, 2), 13 = (1, 3), 14 = (2, 4) **53.** We have $m_1 = 99, m_2 = 98, m_3 = 97$, and $m_4 = 95$, so $m = 99 \cdot 98 \cdot 97 \cdot 95 = 89,403,930$. We find that $M_1 = m/m_1 = 903,070, M_2 = m/m_2 = 912,285, M_3 = m/m_3 = 921,690$, and $M_4 = m/m_4 = 941,094$. Using the Euclidean algorithm, we compute that $y_1 = 37, y_2 = 33, y_3 = 24$, and $y_4 = 4$ are inverses of M_k modulo m_k for $k = 1, 2, 3, 4$, respectively. It follows that the solution is $65 \cdot 903,070 \cdot 37 + 2 \cdot 912,285 \cdot 33 + 51 \cdot 921,690 \cdot 24 + 10 \cdot 941,094 \cdot 4 = 3,397,886,480 \equiv 537,140 \pmod{89,403,930}$. **55.** $\log_2 5 = 16, \log_2 6 = 14$ **57.** $\log_3 1 = 0, \log_3 2 = 14, \log_3 3 = 1, \log_3 4 = 12, \log_3 5 = 5, \log_3 6 = 15, \log_3 7 = 11, \log_3 8 = 10, \log_3 9 = 2, \log_3 10 = 3, \log_3 11 = 7, \log_3 12 = 13, \log_3 13 = 4, \log_3 14 = 9, \log_3 15 = 6, \log_3 16 = 8$ **59.** Assume that s is a solution of $x^2 \equiv a \pmod{p}$. Then because $(-s)^2 = s^2$, $-s$ is also a solution. Furthermore, $s \not\equiv -s \pmod{p}$. Otherwise, $p \mid 2s$, which implies that $p \mid s$, and this implies, using the original assumption, that $p \mid a$, which is a contradiction. Furthermore, if s and t are incongruent solutions modulo p , then because $s^2 \equiv t^2 \pmod{p}$, $p \mid s^2 - t^2$. This implies that $p \mid (s+t)(s-t)$, and by Lemma 3 in Section 4.3, $p \mid s-t$ or $p \mid s+t$, so $s \equiv t \pmod{p}$ or $s \equiv -t \pmod{p}$. Hence, there are at most two solutions. **61.** The value of $(\frac{a}{p})$ depends only on whether a is a quadratic residue modulo p , that is, whether $x^2 \equiv a \pmod{p}$ has a solution. Because this depends only on the equivalence class of a modulo p , it follows that $(\frac{a}{p}) = (\frac{b}{p})$ if $a \equiv b \pmod{p}$. **63.** By Exercise 62, $(\frac{a}{p})(\frac{b}{p}) = a^{(p-1)/2}b^{(p-1)/2} = (ab)^{(p-1)/2} \equiv (\frac{ab}{p}) \pmod{p}$. **65.** $x \equiv 8, 13, 22$, or $27 \pmod{35}$ **67.** Compute $r^e \bmod p$ for $e = 0, 1, 2, \dots, p-2$ until we get the answer a . Worst case and average case time complexity are $O(p \log p)$.

Section 4.5

- 1.** 91, 57, 21, 5 **3.** a) 7, 19, 7, 7, 18, 0 b) Take the next available space **mod** 31. **5.** 1, 5, 4, 1, 5, 4, 1, 5, 4, ... **7.** 2, 6, 7, 10, 8, 2, 6, 7, 10, 8, ... **9.** 2357, 5554, 8469, 7239, 4031, 2489, 1951, 8064 **11.** 2, 1, 1, 1, ... **13.** Only string (d) **15.** 4 **17.** Correctly, of course **19.** a) Not valid b) Valid c) Valid d) Not valid **21.** a) No b) 5 c) 7 d) 8 **23.** Transposition errors involving the last digit **25.** a) Yes b) No c) Yes d) No **27.** Transposition errors will be detected if and only if the transposed digits are an odd number of positions apart and do not differ by 5. **29.** a) Valid b) Not valid c) Valid d) Valid **31.** Yes, as long as the two digits do not differ by 7 **33.** a) Not valid b) Valid c) Valid d) Not valid **35.** The given congruence is equivalent to $3d_1 + 4d_2 + 5d_3 + 6d_4 + 7d_5 + 8d_6 + 9d_7 + 10d_8 \equiv 0 \pmod{11}$. Transposing adjacent digits x and y (with x on the

left) causes the left-hand side to increase by $x - y$. Because $x \not\equiv y \pmod{11}$, the congruence will no longer hold. Therefore errors of this type are always detected.

Section 4.6

- 1. a)** GR QRW SDVV JR **b)** QB ABG CNFFT B **c)** QX UXM AHJJ ZX **3. a)** KOHQV MCIF GHSD **b)** RVBXP TJPZ NBZX **c)** DBYNE PHRM FYZA **5. a)** SURRENDER NOW **b)** BE MY FRIEND **c)** TIME FOR FUN **7.** TO SLEEP PERCHANCE TO DREAM **9.** ANY SUFFICIENTLY ADVANCED TECHNOLOGY IS INDISTINGUISHABLE FROM MAGIC **11.** $p = 7c + 13 \pmod{26}$ **13.** $a = 18$, $b = 5$ **15.** BEWARE OF MARTIANS **17.** Presumably something like an affine cipher **19.** HURRICANE **21.** The length of the key may well be the greatest common divisor of the distances between the starts of the repeated string (or a factor of the gcd). **23.** Suppose we know both $n = pq$ and $(p-1)(q-1)$. To find p and q , first note that $(p-1)(q-1) = pq - p - q + 1 = n - (p+q) + 1$. From this we can find $s = p+q$. Because $q = s-p$, we have $n = p(s-p)$. Hence, $p^2 - ps + n = 0$. We now can use the quadratic formula to find p . Once we have found p , we can find q because $q = n/p$. **25.** 2545 2757 1211 **27.** SILVER **29.** Alice sends $5^8 \pmod{23} = 16$ to Bob. Bob sends $5^5 \pmod{23} = 20$ to Alice. Alice computes $20^8 \pmod{23} = 6$ and Bob computes $16^5 \pmod{23} = 6$. The shared key is 6. **31.** 2186 2087 1279 1251 0326 0816 1948 **33.** Alice can decrypt the first part of Cathy's message to learn the key, and Bob can decrypt the second part of Cathy's message, which Alice forwarded to him, to learn the key. No one else besides Cathy can learn the key, because all of these communications use secure private keys.

Supplementary Exercises

- 1.** The actual number of miles driven is $46518 + 100000k$ for some natural number k . **3.** 5, 22, -12, -29 **5.** Because $ac \equiv bc \pmod{m}$ there is an integer k such that $ac = bc + km$. Hence, $a - b = km/c$. Because $a - b$ is an integer, $c \mid km$. Letting $d = \gcd(m, c)$, write $c = de$. Because no factor of e divides m/d , it follows that $d \mid m$ and $e \mid k$. Thus $a - b = (k/e)(m/d)$, where $k/e \in \mathbf{Z}$ and $m/d \in \mathbf{Z}$. Therefore $a \equiv b \pmod{m/d}$. **7.** Proof of the contrapositive: If n is odd, then $n = 2k + 1$ for some integer k . Therefore $n^2 + 1 = (2k + 1)^2 + 1 = 4k^2 + 4k + 2 \equiv 2 \pmod{4}$. But perfect squares of even numbers are congruent to 0 modulo 4 (because $(2m)^2 = 4m^2$), and perfect squares of odd numbers are congruent to 1 or 3 modulo 4, so $n^2 + 1$ is not a perfect square. **9.** n is divisible by 8 if and only if the binary expansion of n ends with 000. **11.** We assume that someone has chosen a positive integer less than 2^n , which we are to guess. We ask the person to write the number in binary, using leading 0s if necessary to make it n bits long. We then ask "Is the first bit a 1?", "Is the second bit a 1?", "Is the third bit a 1?", and so

on. After we know the answers to these n questions, we will know the number, because we will know its binary expansion.

- 13.** $(a_n a_{n-1} \dots a_1 a_0)_10 = \sum_{k=0}^n 10^k a_k \equiv \sum_{k=0}^n a_k \pmod{9}$ because $10^k \equiv 1 \pmod{9}$ for every nonnegative integer k . **15.** Because for all $k \leq n$, when Q_n is divided by k the remainder will be 1, it follows that no prime number less than or equal to n is a factor of Q_n . Thus by the fundamental theorem of arithmetic, Q_n must have a prime factor greater than n . **17.** Take $a = 10$ and $b = 1$ in Dirichlet's theorem. **19.** Every number greater than 11 can be written as either $8+2n$ or $9+2n$ for some $n \geq 2$. **21.** Assume that every even integer greater than 2 is the sum of two primes, and let n be an integer greater than 5. If n is odd, write $n = 3 + (n-3)$ and decompose $n-3 = p+q$ into the sum of two primes; if n is even, then write $n = 2 + (n-2)$ and decompose $n-2 = p+q$ into the sum of two primes. For the converse, assume that every integer greater than 5 is the sum of three primes, and let n be an even integer greater than 2. Write $n+2$ as the sum of three primes, one of which is necessarily 2, so $n+2 = 2 + p+q$, whence $n = p+q$. **23.** Recall that a nonconstant polynomial can take on the same value only a finite number of times. Thus f can take on the values 0 and ± 1 only finitely many times, so if there is not some y such that $f(y)$ is composite, then there must be some x_0 such that $\pm f(x_0)$ is prime, say p . Look at $f(x_0 + kp)$. When we plug $x_0 + kp$ in for x in the polynomial and multiply it out, every term will contain a factor of p except for the terms that form $f(x_0)$. Therefore $f(x_0 + kp) = f(x_0) + mp = (m \pm 1)p$ for some integer m . As k varies, this value can be 0, p , or $-p$ only finitely many times; therefore it must be a composite number for some values of k . **25.** 1 **27.** 1 **29.** If not, then suppose that q_1, q_2, \dots, q_n are all the primes of the form $6k + 5$. Let $Q = 6q_1 q_2 \dots q_n - 1$. Note that Q is of the form $6k + 5$, where $k = q_1 q_2 \dots q_n - 1$. Let $Q = p_1 p_2 \dots p_l$ be the prime factorization of Q . No p_i is 2, 3, or any q_j , because the remainder when Q is divided by 2 is 1, by 3 is 2, and by q_j is $q_j - 1$. All odd primes other than 3 are of the form $6k + 1$ or $6k + 5$, and the product of primes of the form $6k + 1$ is also of this form. Therefore at least one of the p_i 's must be of the form $6k + 5$, a contradiction. **31.** The product of numbers of the form $4k + 1$ is of the form $4k + 1$, but numbers of this form might have numbers not of this form as their only prime factors. For example, $49 = 4 \cdot 12 + 1$, but the prime factorization of 49 is $7 \cdot 7 = (4 \cdot 1 + 3)(4 \cdot 1 + 3)$. **33. a)** Not mutually relatively prime **b)** Mutually relatively prime **c)** Mutually relatively prime **d)** Mutually relatively prime **35.** 1 **37.** $x \equiv 28 \pmod{30}$ **39.** By the Chinese remainder theorem, it suffices to show that $n^9 - n \equiv 0 \pmod{2}$, $n^9 - n \equiv 0 \pmod{3}$, and $n^9 - n \equiv 0 \pmod{5}$. Each in turn follows from applying Fermat's little theorem. **41.** By Fermat's little theorem, $p^{q-1} \equiv 1 \pmod{q}$ and clearly $q^{p-1} \equiv 0 \pmod{q}$. Therefore $p^{q-1} + q^{p-1} \equiv 1 + 0 = 1 \pmod{q}$. Similarly, $p^{q-1} + q^{p-1} \equiv 1 \pmod{p}$. It follows from the Chinese remainder theorem that $p^{q-1} + q^{p-1} \equiv 1 \pmod{pq}$. **43.** If a_i is changed from x to y , then the change in the left-hand side of the congruence is either $y - x$ or $3(y - x)$, modulo 10, neither of which can be 0 because 1 and

3 are relatively prime to 10. Therefore the sum can no longer be 0 modulo 10. **45.** Working modulo 10, solve for d_9 . The check digit for 11100002 is 5. **47.** PLEASE SEND MONEY **49.** **a)** QAL HUVEM AT WVESGB **b)** QXB EVZZL ZEVZZRFS

CHAPTER 5

Section 5.1

- 1.** Let $P(n)$ be the statement that the train stops at station n . *Basis step:* We are told that $P(1)$ is true. *Inductive step:* We are told that $P(n)$ implies $P(n+1)$ for each $n \geq 1$. Therefore by the principle of mathematical induction, $P(n)$ is true for all positive integers n . **3. a)** $1^2 = 1 \cdot 2 \cdot 3/6$ **b)** Both sides of $P(1)$ shown in part (a) equal 1. **c)** $1^2 + 2^2 + \dots + k^2 = k(k+1)(2k+1)/6$ **d)** For each $k \geq 1$ that $P(k)$ implies $P(k+1)$; in other words, that assuming the inductive hypothesis [see part (c)] we can show $1^2 + 2^2 + \dots + k^2 + (k+1)^2 = (k+1)(k+2)(2k+3)/6$ **e)** $(1^2 + 2^2 + \dots + k^2) + (k+1)^2 = [k(k+1)(2k+1)/6] + (k+1)^2 = [(k+1)/6][k(2k+1) + 6(k+1)] = [(k+1)/6](2k^2 + 7k + 6) = [(k+1)/6](k+2)(2k+3) = (k+1)(k+2)(2k+3)/6$ **f)** We have completed both the basis step and the inductive step, so by the principle of mathematical induction, the statement is true for every positive integer n . **5.** Let $P(n)$ be “ $1^2 + 3^2 + \dots + (2n+1)^2 = (n+1)(2n+1)(2n+3)/3$.” *Basis step:* $P(0)$ is true because $1^2 = 1 = (0+1)(2\cdot 0+1)(2\cdot 0+3)/3$. *Inductive step:* Assume that $P(k)$ is true. Then $1^2 + 3^2 + \dots + (2k+1)^2 + [2(k+1)+1]^2 = (k+1)(2k+1)(2k+3)/3 + (2k+3)^2 = (2k+3)[(k+1)(2k+1)/3 + (2k+3)] = (2k+3)(2k^2+9k+10)/3 = (2k+3)(2k+5)(k+2)/3 = [(k+1)+1][2(k+1)+1][2(k+1)+3]/3$. **7.** Let $P(n)$ be “ $\sum_{j=0}^n 3 \cdot 5^j = 3(5^{n+1} - 1)/4$.” *Basis step:* $P(0)$ is true because $\sum_{j=0}^0 3 \cdot 5^j = 3 = 3(5^1 - 1)/4$. *Inductive step:* Assume that $\sum_{j=0}^k 3 \cdot 5^j = 3(5^{k+1} - 1)/4$. Then $\sum_{j=0}^{k+1} 3 \cdot 5^j = (\sum_{j=0}^k 3 \cdot 5^j) + 3 \cdot 5^{k+1} = 3(5^{k+1} - 1)/4 + 3 \cdot 5^{k+1} = 3(5^{k+1} + 4 \cdot 5^{k+1} - 1)/4 = 3(5^{k+2} - 1)/4$. **9. a)** $2+4+6+\dots+2n = n(n+1)$ **b)** *Basis step:* $2 = 1 \cdot (1+1)$ is true. *Inductive step:* Assume that $2+4+6+\dots+2k = k(k+1)$. Then $(2+4+6+\dots+2k) + 2(k+1) = k(k+1) + 2(k+1) = (k+1)(k+2)$. **11. a)** $\sum_{j=1}^n 1/2^j = (2^n - 1)/2^n$ **b)** *Basis step:* $P(1)$ is true because $\frac{1}{2} = (2^1 - 1)/2^1$. *Inductive step:* Assume that $\sum_{j=1}^k 1/2^j = (2^k - 1)/2^k$. Then $\sum_{j=1}^{k+1} \frac{1}{2^j} = (\sum_{j=1}^k \frac{1}{2^j}) + \frac{1}{2^{k+1}} = \frac{2^k - 1}{2^k} + \frac{1}{2^{k+1}} = \frac{2^{k+1} - 2 + 1}{2^{k+1}} = \frac{2^{k+1} - 1}{2^{k+1}}$. **13.** Let $P(n)$ be “ $1^2 - 2^2 + 3^2 - \dots + (-1)^{n-1}n^2 = (-1)^{n-1}n(n+1)/2$.” *Basis step:* $P(1)$ is true because $1^2 = 1 = (-1)^0 1^2$. *Inductive step:* Assume that $P(k)$ is true. Then $1^2 - 2^2 + 3^2 - \dots + (-1)^{k-1}k^2 + (-1)^k(k+1)^2 = (-1)^{k-1}k(k+1)/2 + (-1)^k(k+1)^2 = (-1)^k(k+1)[-k/2 + (k+1)] = (-1)^k(k+1)[(k/2) + 1] = (-1)^k(k+1)(k+2)/2$. **15.** Let $P(n)$ be “ $1 \cdot 2 + 2 \cdot 3 + \dots + n(n+1) = n(n+1)(n+2)/3$.” *Basis step:* $P(1)$ is true because

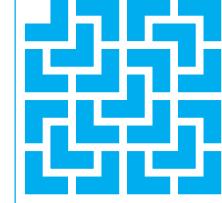
- $1 \cdot 2 = 2 = 1(1+1)(1+2)/3$. *Inductive step:* Assume that $P(k)$ is true. Then $1 \cdot 2 + 2 \cdot 3 + \dots + k(k+1) + (k+1)(k+2) = [k(k+1)(k+2)/3] + (k+1)(k+2) = (k+1)(k+2)[(k/3) + 1] = (k+1)(k+2)(k+3)/3$. **17.** Let $P(n)$ be the statement that $1^4 + 2^4 + 3^4 + \dots + n^4 = n(n+1)(2n+1)(3n^2+3n-1)/30$. $P(1)$ is true because $1 \cdot 2 \cdot 3 \cdot 5/30 = 1$. Assume that $P(k)$ is true. Then $(1^4 + 2^4 + 3^4 + \dots + k^4) + (k+1)^4 = k(k+1)(2k+1)(3k^2+3k-1)/30 + (k+1)^4 = [(k+1)/30][k(2k+1)(3k^2+3k-1) + 30(k+1)^3] = [(k+1)/30](6k^4+39k^3+91k^2+89k+30) = [(k+1)/30](k+2)(2k+3)[3(k+1)^2+3(k+1)-1]$. This demonstrates that $P(k+1)$ is true. **19. a)** $1 + \frac{1}{4} < 2 - \frac{1}{2}$ **b)** This is true because $5/4$ is less than $6/4$. **c)** $1 + \frac{1}{4} + \dots + \frac{1}{k^2} < 2 - \frac{1}{k}$ **d)** For each $k \geq 2$ that $P(k)$ implies $P(k+1)$; in other words, we want to show that assuming the inductive hypothesis [see part (c)] we can show $1 + \frac{1}{4} + \dots + \frac{1}{k^2} + \frac{1}{(k+1)^2} < 2 - \frac{1}{k+1}$ **e)** $1 + \frac{1}{4} + \dots + \frac{1}{k^2} + \frac{1}{(k+1)^2} < 2 - \frac{1}{k} + \frac{1}{(k+1)^2} = 2 - [\frac{1}{k} - \frac{1}{(k+1)^2}] = 2 - [\frac{k^2+2k+1-k}{k(k+1)^2}] = 2 - \frac{k^2+k}{k(k+1)^2} - \frac{1}{k(k+1)^2} = 2 - \frac{1}{k+1} - \frac{1}{k(k+1)^2} < 2 - \frac{1}{k+1}$ **f)** We have completed both the basis step and the inductive step, so by the principle of mathematical induction, the statement is true for every integer n greater than 1. **21.** Let $P(n)$ be “ $2^n > n^2$.” *Basis step:* $P(5)$ is true because $2^5 = 32 > 25 = 5^2$. *Inductive step:* Assume that $P(k)$ is true, that is, $2^k > k^2$. Then $2^{k+1} = 2 \cdot 2^k > k^2 + k^2 > k^2 + 4k \geq k^2 + 2k + 1 = (k+1)^2$ because $k > 4$. **23.** By inspection we find that the inequality $2n+3 \leq 2^n$ does not hold for $n = 0, 1, 2, 3$. Let $P(n)$ be the proposition that this inequality holds for the positive integer n . $P(4)$, the basis case, is true because $2 \cdot 4 + 3 = 11 \leq 16 = 2^4$. For the inductive step assume that $P(k)$ is true. Then, by the inductive hypothesis, $2(k+1)+3 = (2k+3)+2 < 2^k+2$. But because $k \geq 1$, $2^k+2 \leq 2^k+2^k = 2^{k+1}$. This shows that $P(k+1)$ is true. **25.** Let $P(n)$ be “ $1 + nh \leq (1+h)^n$, $h > -1$.” *Basis step:* $P(0)$ is true because $1 + 0 \cdot h = 1 \leq 1 = (1+h)^0$. *Inductive step:* Assume $1 + kh \leq (1+h)^k$. Then because $(1+h) > 0$, $(1+h)^{k+1} = (1+h)(1+h)^k \geq (1+h)(1+kh) = 1 + (k+1)h + kh^2 \geq 1 + (k+1)h$. **27.** Let $P(n)$ be “ $1/\sqrt{1} + 1/\sqrt{2} + 1/\sqrt{3} + \dots + 1/\sqrt{n} > 2(\sqrt{n+1} - 1)$.” *Basis step:* $P(1)$ is true because $1 > 2(\sqrt{2} - 1)$. *Inductive step:* Assume that $P(k)$ is true. Then $1 + 1/\sqrt{2} + \dots + 1/\sqrt{k} + 1/\sqrt{k+1} > 2(\sqrt{k+1} - 1) + 1/\sqrt{k+1}$. If we show that $2(\sqrt{k+1} - 1) + 1/\sqrt{k+1} > 2(\sqrt{k+2} - 1)$, it follows that $P(k+1)$ is true. This inequality is equivalent to $2(\sqrt{k+2} - \sqrt{k+1}) < 1/\sqrt{k+1}$, which is equivalent to $2(\sqrt{k+2} - \sqrt{k+1})(\sqrt{k+2} + \sqrt{k+1}) < \sqrt{k+1}/\sqrt{k+1} + \sqrt{k+2}/\sqrt{k+1}$. This is equivalent to $2 < 1 + \sqrt{k+2}/\sqrt{k+1}$, which is clearly true. **29.** Let $P(n)$ be “ $H_{2^n} \leq 1 + n$.” *Basis step:* $P(0)$ is true because $H_{2^0} = H_1 = 1 \leq 1 + 0$. *Inductive step:* Assume that $H_{2^k} \leq 1 + k$. Then $H_{2^{k+1}} = H_{2^k} + \sum_{j=2^k+1}^{2^{k+1}} \frac{1}{j} \leq 1 + k + 2^k \left(\frac{1}{2^{k+1}} \right) < 1 + k + 1 = 1 + (k+1)$. **31.** *Basis step:* $1^2 + 1 = 2$ is divisible by 2. *Inductive step:* Assume the inductive hypothesis, that $k^2 + k$ is divisible by 2. Then $(k+1)^2 + (k+1) = k^2 + 2k + 1 + k + 1 = (k^2 + k) + 2(k+1)$,

the sum of a multiple of 2 (by the inductive hypothesis) and a multiple of 2 (by definition), hence, divisible by 2. **33.** Let $P(n)$ be “ $n^5 - n$ is divisible by 5.” *Basis step:* $P(0)$ is true because $0^5 - 0 = 0$ is divisible by 5. *Inductive step:* Assume that $P(k)$ is true, that is, $k^5 - k$ is divisible by 5. Then $(k+1)^5 - (k+1) = (k^5 + 5k^4 + 10k^3 + 10k^2 + 5k + 1) - (k+1) = (k^5 - k) + 5(k^4 + 2k^3 + 2k^2 + k)$ is also divisible by 5, because both terms in this sum are divisible by 5. **35.** Let $P(n)$ be the proposition that $(2n - 1)^2 - 1$ is divisible by 8. The basis case $P(1)$ is true because $8 \mid 0$. Now assume that $P(k)$ is true. Because $[(2(k+1) - 1)^2 - 1] = [(2k - 1)^2 - 1] + 8k$, $P(k+1)$ is true because both terms on the right-hand side are divisible by 8. This shows that $P(n)$ is true for all positive integers n , so $m^2 - 1$ is divisible by 8 whenever m is an odd positive integer. **37.** *Basis step:* $11^{1+1} + 12^{2-1-1} = 121 + 12 = 133$. *Inductive step:* Assume the inductive hypothesis, that $11^{n+1} + 12^{2n-1}$ is divisible by 133. Then $11^{(n+1)+1} + 12^{2(n+1)-1} = 11 \cdot 11^{n+1} + 144 \cdot 12^{2n-1} = 11 \cdot 11^{n+1} + (11 + 133) \cdot 12^{2n-1} = 11(11^{n+1} + 12^{2n-1}) + 133 \cdot 12^{2n-1}$. The expression in parentheses is divisible by 133 by the inductive hypothesis, and obviously the second term is divisible by 133, so the entire quantity is divisible by 133, as desired. **39.** *Basis step:* $A_1 \subseteq B_1$ tautologically implies that $\bigcap_{j=1}^1 A_j \subseteq \bigcap_{j=1}^1 B_j$. *Inductive step:* Assume the inductive hypothesis that if $A_j \subseteq B_j$ for $j = 1, 2, \dots, k$, then $\bigcap_{j=1}^k A_j \subseteq \bigcap_{j=1}^k B_j$. We want to show that if $A_j \subseteq B_j$ for $j = 1, 2, \dots, k+1$, then $\bigcap_{j=1}^{k+1} A_j \subseteq \bigcap_{j=1}^{k+1} B_j$. Let x be an arbitrary element of $\bigcap_{j=1}^{k+1} A_j = (\bigcap_{j=1}^k A_j) \cap A_{k+1}$. Because $x \in \bigcap_{j=1}^k A_j$, we know by the inductive hypothesis that $x \in \bigcap_{j=1}^k B_j$; because $x \in A_{k+1}$, we know from the given fact that $A_{k+1} \subseteq B_{k+1}$ that $x \in B_{k+1}$. Therefore, $x \in (\bigcap_{j=1}^k B_j) \cap B_{k+1} = \bigcap_{j=1}^{k+1} B_j$. **41.** Let $P(n)$ be “ $(A_1 \cup A_2 \cup \dots \cup A_n) \cap B = (A_1 \cap B) \cup (A_2 \cap B) \cup \dots \cup (A_n \cap B)$.” *Basis step:* $P(1)$ is trivially true. *Inductive step:* Assume that $P(k)$ is true. Then $(A_1 \cup A_2 \cup \dots \cup A_k \cup A_{k+1}) \cap B = [(A_1 \cup A_2 \cup \dots \cup A_k) \cup A_{k+1}] \cap B = [(A_1 \cup A_2 \cup \dots \cup A_k) \cap B] \cup (A_{k+1} \cap B) = [(A_1 \cap B) \cup (A_2 \cap B) \cup \dots \cup (A_k \cap B)] \cup (A_{k+1} \cap B) = (A_1 \cap B) \cup (A_2 \cap B) \cup \dots \cup (A_k \cap B) \cup (A_{k+1} \cap B)$. **43.** Let $P(n)$ be “ $\overline{\bigcup_{k=1}^n A_k} = \bigcap_{k=1}^n \overline{A_k}$.” *Basis step:* $P(1)$ is trivially true. *Inductive step:* Assume that $P(k)$ is true. Then $\overline{\bigcup_{j=1}^{k+1} A_j} = \overline{(\bigcup_{j=1}^k A_j) \cup A_{k+1}} = \overline{(\bigcup_{j=1}^k A_j)} \cap \overline{A_{k+1}} = (\bigcap_{j=1}^k \overline{A_j}) \cap \overline{A_{k+1}} = \bigcap_{j=1}^{k+1} \overline{A_j}$. **45.** Let $P(n)$ be the statement that a set with n elements has $n(n-1)/2$ two-element subsets. $P(2)$, the basis case, is true, because a set with two elements has one subset with two elements—namely, itself—and $2(2-1)/2 = 1$. Now assume that $P(k)$ is true. Let S be a set with $k+1$ elements. Choose an element a in S and let $T = S - \{a\}$. A two-element subset of S either contains a or does not. Those subsets not containing a are the subsets of T with two elements; by the inductive hypothesis there are $k(k-1)/2$ of these. There are k subsets of S with two elements that contain a , because such a subset contains a and one of the k elements in T . Hence, there are $k(k-1)/2 + k = (k+1)k/2$ two-element subsets of S . This

completes the inductive proof. **47.** Reorder the locations if necessary so that $x_1 \leq x_2 \leq x_3 \leq \dots \leq x_d$. Place the first tower at position $t_1 = x_1 + 1$. Assume tower k has been placed at position t_k . Then place tower $k+1$ at position $t_{k+1} = x + 1$, where x is the smallest x_i greater than $t_k + 1$. **49.** The two sets do not overlap if $n+1 = 2$. In fact, the conditional statement $P(1) \rightarrow P(2)$ is false. **51.** The mistake is in applying the inductive hypothesis to look at $\max(x-1, y-1)$, because even though x and y are positive integers, $x-1$ and $y-1$ need not be (one or both could be 0). **53.** For the basis step ($n = 2$) the first person cuts the cake into two portions that she thinks are each $1/2$ of the cake, and the second person chooses the portion he thinks is at least $1/2$ of the cake (at least one of the pieces must satisfy that condition). For the inductive step, suppose there are $k+1$ people. By the inductive hypothesis, we can suppose that the first k people have divided the cake among themselves so that each person is satisfied that he got at least a fraction $1/k$ of the cake. Each of them now cuts his or her piece into $k+1$ pieces of equal size. The last person gets to choose one piece from each of the first k people’s portions. After this is done, each of the first k people is satisfied that she still has $(1/k)(k/(k+1)) = 1/(k+1)$ of the cake. To see that the last person is satisfied, suppose that he thought that the i th person ($1 \leq i \leq k$) had a portion p_i of the cake, where $\sum_{i=1}^k p_i = 1$. By choosing what he thinks is the largest piece from each person, he is satisfied that he has at least $\sum_{i=1}^k p_i / (k+1) = (1/(k+1)) \sum_{i=1}^k p_i = 1/(k+1)$ of the cake. **55.** We use the notation (i, j) to mean the square in row i and column j and use induction on $i+j$ to show that every square can be reached by the knight. *Basis step:* There are six base cases, for the cases when $i+j \leq 2$. The knight is already at $(0, 0)$ to start, so the empty sequence of moves reaches that square. To reach $(1, 0)$, the knight moves $(0, 0) \rightarrow (2, 1) \rightarrow (0, 2) \rightarrow (1, 0)$. Similarly, to reach $(0, 1)$, the knight moves $(0, 0) \rightarrow (1, 2) \rightarrow (2, 0) \rightarrow (0, 1)$. Note that the knight has reached $(2, 0)$ and $(0, 2)$ in the process. For the last basis step there is $(0, 0) \rightarrow (1, 2) \rightarrow (2, 0) \rightarrow (0, 1) \rightarrow (2, 2) \rightarrow (0, 3) \rightarrow (1, 1)$. *Inductive step:* Assume the inductive hypothesis, that the knight can reach any square (i, j) for which $i+j = k$, where k is an integer greater than 1. We must show how the knight can reach each square (i, j) when $i+j = k+1$. Because $k+1 \geq 3$, at least one of i and j is at least 2. If $i \geq 2$, then by the inductive hypothesis, there is a sequence of moves ending at $(i-2, j+1)$, because $i-2+j+1 = i+j-1 = k$; from there it is just one step to (i, j) ; similarly, if $j \geq 2$. **57.** *Basis step:* The base cases $n = 0$ and $n = 1$ are true because the derivative of x^0 is 0 and the derivative of $x^1 = x$ is 1. *Inductive step:* Using the product rule, the inductive hypothesis, and the basis step shows that $\frac{d}{dx} x^{k+1} = \frac{d}{dx} (x \cdot x^k) = x \cdot \frac{d}{dx} x^k + x^k \frac{d}{dx} x = x \cdot kx^{k-1} + x^k \cdot 1 = kx^k + x^k = (k+1)x^k$. **59.** *Basis step:* For $k = 0, 1 \equiv 1 \pmod{m}$. *Inductive step:* Suppose that $a \equiv b \pmod{m}$ and $a^k \equiv b^k \pmod{m}$; we must show that $a^{k+1} \equiv b^{k+1} \pmod{m}$. By Theorem 5 from Section 4.1, $a \cdot a^k \equiv b \cdot b^k \pmod{m}$, which by defin-

tion says that $a^{k+1} \equiv b^{k+1} \pmod{m}$. **61.** Let $P(n)$ be “[$(p_1 \rightarrow p_2) \wedge (p_2 \rightarrow p_3) \wedge \cdots \wedge (p_{n-1} \rightarrow p_n)] \rightarrow [(p_1 \wedge \cdots \wedge p_{n-1}) \rightarrow p_n]$. Basis step: $P(2)$ is true because $(p_1 \rightarrow p_2) \rightarrow (p_1 \rightarrow p_2)$ is a tautology. Inductive step: Assume $P(k)$ is true. To show $[(p_1 \rightarrow p_2) \wedge \cdots \wedge (p_{k-1} \rightarrow p_k) \wedge (p_k \rightarrow p_{k+1})] \rightarrow [(p_1 \wedge \cdots \wedge p_{k-1} \wedge p_k) \rightarrow p_{k+1}]$ is a tautology, assume that the hypothesis of this conditional statement is true. Because both the hypothesis and $P(k)$ are true, it follows that $(p_1 \wedge \cdots \wedge p_{k-1}) \rightarrow p_k$ is true. Because this is true, and because $p_k \rightarrow p_{k+1}$ is true (it is part of the assumption) it follows by hypothetical syllogism that $(p_1 \wedge \cdots \wedge p_{k-1}) \rightarrow p_{k+1}$ is true. The weaker statement $(p_1 \wedge \cdots \wedge p_{k-1} \wedge p_k) \rightarrow p_{k+1}$ follows from this. **63.** We will first prove the result when n is a power of 2, that is, if $n = 2^k$, $k = 1, 2, \dots$. Let $P(k)$ be the statement $A \geq G$, where A and G are the arithmetic and geometric means, respectively, of a set of $n = 2^k$ positive real numbers. Basis step: $k = 1$ and $n = 2^1 = 2$. Note that $(\sqrt{a_1} - \sqrt{a_2})^2 \geq 0$. Expanding this shows that $a_1 - 2\sqrt{a_1 a_2} + a_2 \geq 0$, that is, $(a_1 + a_2)/2 \geq (a_1 a_2)^{1/2}$. Inductive step: Assume that $P(k)$ is true, with $n = 2^k$. We will show that $P(k+1)$ is true. We have $2^{k+1} = 2n$. Now $(a_1 + a_2 + \cdots + a_{2n})/(2n) = [(a_1 + a_2 + \cdots + a_n)/n + (a_{n+1} + a_{n+2} + \cdots + a_{2n})/n]/2$ and similarly $(a_1 a_2 \cdots a_{2n})^{1/(2n)} = [(a_1 \cdots a_n)^{1/n} (a_{n+1} \cdots a_{2n})^{1/n}]^{1/2}$. To simplify the notation, let $A(x, y, \dots)$ and $G(x, y, \dots)$ denote the arithmetic mean and geometric mean of x, y, \dots , respectively. Also, if $x \leq x'$, $y \leq y'$, and so on, then $A(x, y, \dots) \leq A(x', y', \dots)$ and $G(x, y, \dots) \leq G(x', y', \dots)$. Hence, $A(a_1, \dots, a_{2n}) = A(A(a_1, \dots, a_n), A(a_{n+1}, \dots, a_{2n})) \geq A(G(a_1, \dots, a_n), G(a_{n+1}, \dots, a_{2n})) \geq G(G(a_1, \dots, a_n), G(a_{n+1}, \dots, a_{2n})) = G(a_1, \dots, a_{2n})$. This finishes the proof for powers of 2. Now if n is not a power of 2, let m be the next higher power of 2, and let a_{n+1}, \dots, a_m all equal $A(a_1, \dots, a_n) = \bar{a}$. Then we have $[(a_1 a_2 \cdots a_n) \bar{a}^{m-n}]^{1/m} \leq A(a_1, \dots, a_m)$, because m is a power of 2. Because $A(a_1, \dots, a_m) = \bar{a}$, it follows that $(a_1 \cdots a_n)^{1/m} \bar{a}^{1-n/m} \leq \bar{a}^{n/m}$. Raising both sides to the (m/n) th power gives $G(a_1, \dots, a_n) \leq A(a_1, \dots, a_n)$. **65.** Basis step: For $n = 1$, the left-hand side is just $\frac{1}{1}$, which is 1. For $n = 2$, there are three nonempty subsets $\{1\}$, $\{2\}$, and $\{1, 2\}$, so the left-hand side is $\frac{1}{1} + \frac{1}{2} + \frac{1}{1 \cdot 2} = 2$. Inductive step: Assume that the statement is true for k . The set of the first $k+1$ positive integers has many nonempty subsets, but they fall into three categories: a nonempty subset of the first k positive integers together with $k+1$, a nonempty subset of the first k positive integers, or just $\{k+1\}$. By the inductive hypothesis, the sum of the first category is k . For the second category, we can factor out $1/(k+1)$ from each term of the sum and what remains is just k by the inductive hypothesis, so this part of the sum is $k/(k+1)$. Finally, the third category simply yields $1/(k+1)$. Hence, the entire summation is $k + k/(k+1) + 1/(k+1) = k+1$. **67.** Basis step: If $A_1 \subseteq A_2$, then A_1 satisfies the condition of being a subset of each set in the collection; otherwise $A_2 \subseteq A_1$, so A_2 satisfies the condition. Inductive step: Assume the inductive hypothesis, that the conditional statement is true for k sets,

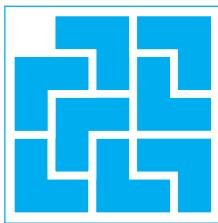
and suppose we are given $k+1$ sets that satisfy the given conditions. By the inductive hypothesis, there must be a set A_i for some $i \leq k$ such that $A_i \subseteq A_j$ for $1 \leq j \leq k$. If $A_i \subseteq A_{k+1}$, then we are done. Otherwise, we know that $A_{k+1} \subseteq A_i$, and this tells us that A_{k+1} satisfies the condition of being a subset of A_j for $1 \leq j \leq k+1$. **69.** $G(1) = 0$, $G(2) = 1$, $G(3) = 3$, $G(4) = 4$ **71.** To show that $2n-4$ calls are sufficient to exchange all the gossip, select persons 1, 2, 3, and 4 to be the central committee. Every person outside the central committee calls one person on the central committee. At this point the central committee members *as a group* know all the scandals. They then exchange information among themselves by making the calls 1-2, 3-4, 1-3, and 2-4 in that order. At this point, *every* central committee member knows all the scandals. Finally, again every person outside the central committee calls one person on the central committee, at which point everyone knows all the scandals. [The total number of calls is $(n-4) + 4 + (n-4) = 2n-4$.] That this cannot be done with fewer than $2n-4$ calls is much harder to prove; see Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman, “A survey of gossiping and broadcasting in communication networks,” *Networks* **18** (1988), no. 4, 319–349, for details. **73.** We prove this by mathematical induction. The basis step ($n = 2$) is true tautologically. For $n = 3$, suppose that the intervals are (a, b) , (c, d) , and (e, f) , where without loss of generality we can assume that $a \leq c \leq e$. Because $(a, b) \cap (e, f) \neq \emptyset$, we must have $e < b$; for a similar reason, $e < d$. It follows that the number halfway between e and the smaller of b and d is common to all three intervals. Now for the inductive step, assume that whenever we have k intervals that have pairwise nonempty intersections then there is a point common to all the intervals, and suppose that we are given intervals I_1, I_2, \dots, I_{k+1} that have pairwise nonempty intersections. For each i from 1 to k , let $J_i = I_i \cap I_{k+1}$. We claim that the collection J_1, J_2, \dots, J_k satisfies the inductive hypothesis, that is, that $J_{i_1} \cap J_{i_2} \neq \emptyset$ for each choice of subscripts i_1 and i_2 . This follows from the $n = 3$ case proved above, using the sets I_{i_1}, I_{i_2} , and I_{k+1} . We can now invoke the inductive hypothesis to conclude that there is a number common to all of the sets J_i for $i = 1, 2, \dots, k$, which is in the intersection of all the sets I_i for $i = 1, 2, \dots, k+1$. **75.** Pair up the people. Have the people stand at mutually distinct small distances from their partners but far away from everyone else. Then each person throws a pie at his or her partner, so everyone gets hit.

77.

79. Let $P(n)$ be the statement that every $2^n \times 2^n \times 2^n$ checkerboard with a $1 \times 1 \times 1$ cube removed can be covered by tiles

that are $2 \times 2 \times 2$ cubes each with a $1 \times 1 \times 1$ cube removed. The basis step, $P(1)$, holds because one tile coincides with the solid to be tiled. Now assume that $P(k)$ holds. Now consider a $2^{k+1} \times 2^{k+1} \times 2^{k+1}$ cube with a $1 \times 1 \times 1$ cube removed. Split this object into eight pieces using planes parallel to its faces and running through its center. The missing $1 \times 1 \times 1$ piece occurs in one of these eight pieces. Now position one tile with its center at the center of the large object so that the missing $1 \times 1 \times 1$ cube lies in the octant in which the large object is missing a $1 \times 1 \times 1$ cube. This creates eight $2^k \times 2^k \times 2^k$ cubes, each missing a $1 \times 1 \times 1$ cube. By the inductive hypothesis we can fill each of these eight objects with tiles. Putting these tilings together produces the desired tiling.

81.



83. Let $Q(n)$ be $P(n+b-1)$. The statement that $P(n)$ is true for $n = b, b+1, b+2, \dots$ is the same as the statement that $Q(m)$ is true for all positive integers m . We are given that $P(b)$ is true [i.e., that $Q(1)$ is true], and that $P(k) \rightarrow P(k+1)$ for all $k \geq b$ [i.e., that $Q(m) \rightarrow Q(m+1)$ for all positive integers m]. Therefore, by the principle of mathematical induction, $Q(m)$ is true for all positive integers m .

Section 5.2

1. Basis step: We are told we can run one mile, so $P(1)$ is true. **Inductive step:** Assume the inductive hypothesis, that we can run any number of miles from 1 to k . We must show that we can run $k+1$ miles. If $k=1$, then we are already told that we can run two miles. If $k > 1$, then the inductive hypothesis tells us that we can run $k-1$ miles, so we can run $(k-1)+2=k+1$ miles. **3. a)** $P(8)$ is true, because we can form 8 cents of postage with one 3-cent stamp and one 5-cent stamp. $P(9)$ is true, because we can form 9 cents of postage with three 3-cent stamps. $P(10)$ is true, because we can form 10 cents of postage with two 5-cent stamps. **b)** The statement that using just 3-cent and 5-cent stamps we can form j cents postage for all j with $8 \leq j \leq k$, where we assume that $k \geq 10$. **c)** Assuming the inductive hypothesis, we can form $k+1$ cents postage using just 3-cent and 5-cent stamps. **d)** Because $k \geq 10$, we know that $P(k-2)$ is true, that is, that we can form $k-2$ cents of postage. Put one more 3-cent stamp on the envelope, and we have formed $k+1$ cents of postage. **e)** We have completed both the basis step and the inductive step, so by the principle of strong induction, the statement is true for every integer n greater than or equal to 8. **5. a)** 4, 8, 11, 12, 15, 16, 19, 20, 22, 23, 24, 26, 27, 28, and all values greater than or equal to 30. **b)** Let $P(n)$ be the statement that

we can form n cents of postage using just 4-cent and 11-cent stamps. We want to prove that $P(n)$ is true for all $n \geq 30$. For the basis step, $30 = 11 + 11 + 4 + 4$. Assume that we can form k cents of postage (the inductive hypothesis); we will show how to form $k+1$ cents of postage. If the k cents included an 11-cent stamp, then replace it by three 4-cent stamps. Otherwise, k cents was formed from just 4-cent stamps. Because $k \geq 30$, there must be at least eight 4-cent stamps involved. Replace eight 4-cent stamps by three 11-cent stamps, and we have formed $k+1$ cents in postage. **c)** $P(n)$ is the same as in part (b). To prove that $P(n)$ is true for all $n \geq 30$, we check for the basis step that $30 = 11 + 11 + 4 + 4$, $31 = 11 + 4 + 4 + 4 + 4 + 4$, $32 = 4 + 4 + 4 + 4 + 4 + 4 + 4$, and $33 = 11 + 11 + 11$. For the inductive step, assume the inductive hypothesis, that $P(j)$ is true for all j with $30 \leq j \leq k$, where k is an arbitrary integer greater than or equal to 33. We want to show that $P(k+1)$ is true. Because $k-3 \geq 30$, we know that $P(k-3)$ is true, that is, that we can form $k-3$ cents of postage. Put one more 4-cent stamp on the envelope, and we have formed $k+1$ cents of postage. In this proof, our inductive hypothesis was that $P(j)$ was true for all values of j between 30 and k inclusive, rather than just that $P(30)$ was true. **7.** We can form all amounts except \$1 and \$3. Let $P(n)$ be the statement that we can form n dollars using just 2-dollar and 5-dollar bills. We want to prove that $P(n)$ is true for all $n \geq 5$. (It is clear that \$1 and \$3 cannot be formed and that \$2 and \$4 can be formed.) For the basis step, note that $5 = 5$ and $6 = 2+2+2$. Assume the inductive hypothesis, that $P(j)$ is true for all j with $5 \leq j \leq k$, where k is an arbitrary integer greater than or equal to 6. We want to show that $P(k+1)$ is true. Because $k-1 \geq 5$, we know that $P(k-1)$ is true, that is, that we can form $k-1$ dollars. Add another 2-dollar bill, and we have formed $k+1$ dollars. **9.** Let $P(n)$ be the statement that there is no positive integer b such that $\sqrt{2} = n/b$. **Basis step:** $P(1)$ is true because $\sqrt{2} > 1 \geq 1/b$ for all positive integers b . **Inductive step:** Assume that $P(j)$ is true for all $j \leq k$, where k is an arbitrary positive integer; we prove that $P(k+1)$ is true by contradiction. Assume that $\sqrt{2} = (k+1)/b$ for some positive integer b . Then $2b^2 = (k+1)^2$, so $(k+1)^2$ is even, and hence, $k+1$ is even. So write $k+1 = 2t$ for some positive integer t , whence $2b^2 = 4t^2$ and $b^2 = 2t^2$. By the same reasoning as before, b is even, so $b = 2s$ for some positive integer s . Then $\sqrt{2} = (k+1)/b = (2t)/(2s) = t/s$. But $t \leq k$, so this contradicts the inductive hypothesis, and our proof of the inductive step is complete. **11. Basis step:** There are four base cases. If $n = 1 = 4 \cdot 0 + 1$, then clearly the second player wins. If there are two, three, or four matches ($n = 4 \cdot 0 + 2$, $n = 4 \cdot 0 + 3$, or $n = 4 \cdot 1$), then the first player can win by removing all but one match. **Inductive step:** Assume the strong inductive hypothesis, that in games with k or fewer matches, the first player can win if $k \equiv 0, 2$, or $3 \pmod{4}$ and the second player can win if $k \equiv 1 \pmod{4}$. Suppose we have a game with $k+1$ matches, with $k \geq 4$. If $k+1 \equiv 0 \pmod{4}$, then the first player can remove three matches, leaving $k-2$ matches for the other player. Because $k-2 \equiv 1 \pmod{4}$, by the inductive hypothesis, this is a game that the second player

at that point (who is the first player in our game) can win. Similarly, if $k + 1 \equiv 2 \pmod{4}$, then the first player can remove one match; and if $k + 1 \equiv 3 \pmod{4}$, then the first player can remove two matches. Finally, if $k + 1 \equiv 1 \pmod{4}$, then the first player must leave k , $k - 1$, or $k - 2$ matches for the other player. Because $k \equiv 0 \pmod{4}$, $k - 1 \equiv 3 \pmod{4}$, and $k - 2 \equiv 2 \pmod{4}$, by the inductive hypothesis, this is a game that the first player at that point (who is the second player in our game) can win. **13.** Let $P(n)$ be the statement that exactly $n - 1$ moves are required to assemble a puzzle with n pieces. Now $P(1)$ is trivially true. Assume that $P(j)$ is true for all $j \leq k$, and consider a puzzle with $k + 1$ pieces. The final move must be the joining of two blocks, of size j and $k + 1 - j$ for some integer j with $1 \leq j \leq k$. By the inductive hypothesis, it required $j - 1$ moves to construct the one block, and $k + 1 - j - 1 = k - j$ moves to construct the other. Therefore, $1 + (j - 1) + (k - j) = k$ moves are required in all, so $P(k + 1)$ is true. **15.** Let the Chomp board have n rows and n columns. We claim that the first player can win the game by making the first move to leave just the top row and leftmost column. Let $P(n)$ be the statement that if a player has presented his opponent with a Chomp configuration consisting of just n cookies in the top row and n cookies in the leftmost column, then he can win the game. We will prove $\forall n P(n)$ by strong induction. We know that $P(1)$ is true, because the opponent is forced to take the poisoned cookie at his first turn. Fix $k \geq 1$ and assume that $P(j)$ is true for all $j \leq k$. We claim that $P(k + 1)$ is true. It is the opponent's turn to move. If she picks the poisoned cookie, then the game is over and she loses. Otherwise, assume she picks the cookie in the top row in column j , or the cookie in the left column in row j , for some j with $2 \leq j \leq k + 1$. The first player now picks the cookie in the left column in row j , or the cookie in the top row in column j , respectively. This leaves the position covered by $P(j - 1)$ for his opponent, so by the inductive hypothesis, he can win. **17.** Let $P(n)$ be the statement that if a simple polygon with n sides is triangulated, then at least two of the triangles in the triangulation have two sides that border the exterior of the polygon. We will prove $\forall n \geq 4 P(n)$. The statement is clearly true for $n = 4$, because there is only one diagonal, leaving two triangles with the desired property. Fix $k \geq 4$ and assume that $P(j)$ is true for all j with $4 \leq j \leq k$. Consider a polygon with $k + 1$ sides, and some triangulation of it. Pick one of the diagonals in this triangulation. First suppose that this diagonal divides the polygon into one triangle and one polygon with k sides. Then the triangle has two sides that border the exterior. Furthermore, the k -gon has, by the inductive hypothesis, two triangles that have two sides that border the exterior of that k -gon, and only one of these triangles can fail to be a triangle that has two sides that border the exterior of the original polygon. The only other case is that this diagonal divides the polygon into two polygons with j sides and $k + 3 - j$ sides for some j with $4 \leq j \leq k - 1$. By the inductive hypothesis, each of these two polygons has two triangles that have two sides that border their exterior, and in each case only one of these triangles can fail to be a tri-

gle that has two sides that border the exterior of the original polygon. **19.** Let $P(n)$ be the statement that the area of a simple polygon with n sides and vertices all at lattice points is given by $I(P) + B(P)/2 - 1$. We will prove $P(n)$ for all $n \geq 3$. We begin with an additivity lemma: If P is a simple polygon with all vertices at lattice points, divided into polygons P_1 and P_2 by a diagonal, then $I(P) + B(P)/2 - 1 = [I(P_1) + B(P_1)/2 - 1] + [I(P_2) + B(P_2)/2 - 1]$. To prove this, suppose there are k lattice points on the diagonal, not counting its endpoints. Then $I(P) = I(P_1) + I(P_2) + k$ and $B(P) = B(P_1) + B(P_2) - 2k - 2$; and the result follows by simple algebra. What this says in particular is that if Pick's formula gives the correct area for P_1 and P_2 , then it must give the correct formula for P , whose area is the sum of the areas for P_1 and P_2 ; and similarly if Pick's formula gives the correct area for P and one of the P_i 's, then it must give the correct formula for the other P_i . Next we prove the theorem for rectangles whose sides are parallel to the coordinate axes. Such a rectangle necessarily has vertices at $(a, b), (a, c), (d, b)$, and (d, c) , where a, b, c , and d are integers with $b < c$ and $a < d$. Its area is $(c - b)(d - a)$. Also, $B = 2(c - b + d - a)$ and $I = (c - b - 1)(d - a - 1) = (c - b)(d - a) - (c - b) - (d - a) + 1 = (c - b + d - a) - 1 = (c - b)(d - a)$, which is the desired area. Next consider a right triangle whose legs are parallel to the coordinate axes. This triangle is half a rectangle of the type just considered, for which Pick's formula holds, so by the additivity lemma, it holds for the triangle as well. (The values of B and I are the same for each of the two triangles, so if Pick's formula gave an answer that was either too small or too large, then it would give a correspondingly wrong answer for the rectangle.) For the next step, consider an arbitrary triangle with vertices at lattice points that is not of the type already considered. Embed it in as small a rectangle as possible. There are several possible ways this can happen, but in any case (and adding one more edge in one case), the rectangle will have been partitioned into the given triangle and two or three right triangles with sides parallel to the coordinate axes. Again by the additivity lemma, we are guaranteed that Pick's formula gives the correct area for the given triangle. This completes the proof of $P(3)$, the basis step in our strong induction proof. For the inductive step, given an arbitrary polygon, use Lemma 1 in the text to split it into two polygons. Then by the additivity lemma above and the inductive hypothesis, we know that Pick's formula gives the correct area for this polygon. **21. a)** In the left figure $\angle abp$ is smallest, but \overline{bp} is not an interior diagonal. **b)** In the right figure \overline{bd} is not an interior diagonal. **c)** In the right figure \overline{bd} is not an interior diagonal. **23. a)** When we try to prove the inductive step and find a triangle in each subpolygon with at least two sides bordering the exterior, it may happen in each case that the triangle we are guaranteed in fact borders the diagonal (which is part of the boundary of that polygon). This leaves us with no triangles guaranteed to touch the boundary of the *original* polygon. **b)** We proved the stronger statement $\forall n \geq 4 T(n)$ in Exercise 17. **25. a)** The inductive step here allows us to conclude that

$P(3), P(5), \dots$ are all true, but we can conclude nothing about $P(2), P(4), \dots$. **b)** $P(n)$ is true for all positive integers n , using strong induction. **c)** The inductive step here enables us to conclude that $P(2), P(4), P(8), P(16), \dots$ are all true, but we can conclude nothing about $P(n)$ when n is not a power of 2.

d) This is mathematical induction; we can conclude that $P(n)$ is true for all positive integers n . **27.** Suppose, for a proof by contradiction, that there is some positive integer n such that $P(n)$ is not true. Let m be the smallest positive integer greater than n for which $P(m)$ is true; we know that such an m exists because $P(m)$ is true for infinitely many values of m . But we know that $P(m) \rightarrow P(m-1)$, so $P(m-1)$ is also true. Thus, $m-1$ cannot be greater than n , so $m-1 = n$ and $P(n)$ is in fact true. This contradiction shows that $P(n)$ is true for all n .

29. The error is in going from the base case $n = 0$ to the next case, $n = 1$; we cannot write 1 as the sum of two smaller natural numbers. **31.** Assume that the well-ordering property holds. Suppose that $P(1)$ is true and that the conditional statement $[P(1) \wedge P(2) \wedge \dots \wedge P(n)] \rightarrow P(n+1)$ is true for every positive integer n . Let S be the set of positive integers n for which $P(n)$ is false. We will show $S = \emptyset$. Assume that $S \neq \emptyset$. Then by the well-ordering property there is a least integer m in S . We know that m cannot be 1 because $P(1)$ is true. Because $n = m$ is the least integer such that $P(n)$ is false, $P(1), P(2), \dots, P(m-1)$ are true, and $m-1 \geq 1$. Because $[P(1) \wedge P(2) \wedge \dots \wedge P(m-1)] \rightarrow P(m)$ is true, it follows that $P(m)$ must also be true, which is a contradiction. Hence, $S = \emptyset$. **33.** In each case, give a proof by contradiction based on a “smallest counterexample,” that is, values of n and k such that $P(n, k)$ is not true and n and k are smallest in some sense.

a) Choose a counterexample with $n+k$ as small as possible. We cannot have $n = 1$ and $k = 1$, because we are given that $P(1, 1)$ is true. Therefore, either $n > 1$ or $k > 1$. In the former case, by our choice of counterexample, we know that $P(n-1, k)$ is true. But the inductive step then forces $P(n, k)$ to be true, a contradiction. The latter case is similar. So our supposition that there is a counterexample must be wrong, and $P(n, k)$ is true in all cases. **b)** Choose a counterexample with n as small as possible. We cannot have $n = 1$, because we are given that $P(1, k)$ is true for all k . Therefore, $n > 1$. By our choice of counterexample, we know that $P(n-1, k)$ is true. But the inductive step then forces $P(n, k)$ to be true, a contradiction. **c)** Choose a counterexample with k as small as possible. We cannot have $k = 1$, because we are given that $P(n, 1)$ is true for all n . Therefore, $k > 1$. By our choice of counterexample, we know that $P(n, k-1)$ is true. But the inductive step then forces $P(n, k)$ to be true, a contradiction.

35. Let $P(n)$ be the statement that if x_1, x_2, \dots, x_n are n distinct real numbers, then $n-1$ multiplications are used to find the product of these numbers no matter how parentheses are inserted in the product. We will prove that $P(n)$ is true using strong induction. The basis case $P(1)$ is true because $1-1=0$ multiplications are required to find the product of x_1 , a product with only one factor. Suppose that $P(k)$ is true for $1 \leq k \leq n$. The last multiplication used to find the product of the $n+1$ distinct real numbers $x_1, x_2, \dots, x_n, x_{n+1}$ is a multiplication

of the product of the first k of these numbers for some k and the product of the last $n+1-k$ of them. By the inductive hypothesis, $k-1$ multiplications are used to find the product of k of the numbers, no matter how parentheses were inserted in the product of these numbers, and $n-k$ multiplications are used to find the product of the other $n+1-k$ of them, no matter how parentheses were inserted in the product of these numbers. Because one more multiplication is required to find the product of all $n+1$ numbers, the total number of multiplications used equals $(k-1)+(n-k)+1=n$. Hence, $P(n+1)$ is true. **37.** Assume that $a = dq+r = dq'+r'$ with $0 \leq r < d$ and $0 \leq r' < d$. Then $d(q-q') = r'-r$. It follows that d divides $r'-r$. Because $-d < r'-r < d$, we have $r'-r=0$. Hence, $r'=r$. It follows that $q=q'$. **39.** This is a paradox caused by self-reference. The answer is clearly “no.” There are a finite number of English words, so only a finite number of strings of 15 words or fewer; therefore, only a finite number of positive integers can be so described, not all of them. **41.** Suppose that the well-ordering property were false. Let S be a nonempty set of nonnegative integers that has no least element. Let $P(n)$ be the statement “ $i \notin S$ for $i = 0, 1, \dots, n$.” $P(0)$ is true because if $0 \in S$ then S has a least element, namely, 0. Now suppose that $P(n)$ is true. Thus, $0 \notin S, 1 \notin S, \dots, n \notin S$. Clearly, $n+1$ cannot be in S , for if it were, it would be its least element. Thus $P(n+1)$ is true. So by the principle of mathematical induction, $n \notin S$ for all nonnegative integers n . Thus, $S = \emptyset$, a contradiction. **43.** Strong induction implies the principle of mathematical induction, for if one has shown that $P(k) \rightarrow P(k+1)$ is true, then one has also shown that $[P(1) \wedge \dots \wedge P(k)] \rightarrow P(k+1)$ is true. By Exercise 41, the principle of mathematical induction implies the well-ordering property. Therefore by assuming strong induction as an axiom, we can prove the well-ordering property.

Section 5.3

- 1. a)** $f(1) = 3, f(2) = 5, f(3) = 7, f(4) = 9$ **b)** $f(1) = 3, f(2) = 9, f(3) = 27, f(4) = 81$ **c)** $f(1) = 2, f(2) = 4, f(3) = 16, f(4) = 65,536$ **d)** $f(1) = 3, f(2) = 13, f(3) = 183, f(4) = 33,673$ **3. a)** $f(2) = -1, f(3) = 5, f(4) = 2, f(5) = 17$ **b)** $f(2) = -4, f(3) = 32, f(4) = -4096, f(5) = 536,870,912$ **c)** $f(2) = 8, f(3) = 176, f(4) = 92,672, f(5) = 25,764,174,848$ **d)** $f(2) = -\frac{1}{2}, f(3) = -4, f(4) = \frac{1}{8}, f(5) = -32$ **5. a)** Not valid **b)** $f(n) = 1-n$. Basis step: $f(0) = 1 = 1-0$. Inductive step: if $f(k) = 1-k$, then $f(k+1) = f(k)-1 = 1-k-1 = 1-(k+1)$. **c)** $f(n) = 4-n$ if $n > 0$, and $f(0) = 2$. Basis step: $f(0) = 2$ and $f(1) = 3 = 4-1$. Inductive step (with $k \geq 1$): $f(k+1) = f(k)-1 = (4-k)-1 = 4-(k+1)$. **d)** $f(n) = 2^{\lfloor (n+1)/2 \rfloor}$. Basis step: $f(0) = 1 = 2^{\lfloor (0+1)/2 \rfloor}$ and $f(1) = 2 = 2^{\lfloor (1+1)/2 \rfloor}$. Inductive step (with $k \geq 1$): $f(k+1) = 2f(k-1) = 2 \cdot 2^{\lfloor k/2 \rfloor} = 2^{\lfloor (k+1)/2 \rfloor+1} = 2^{\lfloor ((k+1)+1)/2 \rfloor}$. **e)** $f(n) = 3^n$. Basis step: Trivial. Inductive step: For odd n , $f(n) = 3f(n-1) = 3 \cdot 3^{n-1} = 3^n$; and for even $n > 1$, $f(n) = 9f(n-2) = 9 \cdot 3^{n-2} = 3^n$. **7.** There

are many possible correct answers. We will supply relatively simple ones.

a) $a_{n+1} = a_n + 6$ for $n \geq 1$ and $a_1 = 6$

b) $a_{n+1} = a_n + 2$ for $n \geq 1$ and $a_1 = 3$

c) $a_{n+1} = 10a_n$ for $n \geq 1$ and $a_1 = 10$

d) $a_{n+1} = a_n$ for $n \geq 1$ and $a_1 = 5$

9. $F(0) = 0$, $F(n) = F(n - 1) + n$ for $n \geq 1$

11. $P_m(0) = 0$, $P_m(n + 1) = P_m(n) + m$

13. Let $P(n)$ be “ $f_1 + f_3 + \cdots + f_{2n-1} = f_{2n}$.” *Basis step:* $P(1)$ is true because $f_1 = 1 = f_2$. *Inductive step:* Assume that $P(k)$ is true. Then $f_1 + f_3 + \cdots + f_{2k-1} + f_{2k+1} = f_{2k} + f_{2k+1} = f_{2k+2} + f_{2(k+1)}$.

15. Basis step: $f_0f_1 + f_1f_2 = 0 \cdot 1 + 1 \cdot 1 = 1^2 = f_2^2$.

Inductive step: Assume that $f_0f_1 + f_1f_2 + \cdots + f_{2k-1}f_{2k} = f_{2k}^2$. Then $f_0f_1 + f_1f_2 + \cdots + f_{2k-1}f_{2k} + f_{2k}f_{2k+1} + f_{2k+1}f_{2k+2} = f_{2k}^2 + f_{2k}f_{2k+1} + f_{2k+1}f_{2k+2} = f_{2k}(f_{2k} + f_{2k+1}) + f_{2k+1}f_{2k+2} = f_{2k}f_{2k+2} + f_{2k+1}f_{2k+2} = (f_{2k} + f_{2k+1})f_{2k+2} = f_{2k+2}^2$.

17. The number of divisions used by the Euclidean algorithm to find $\gcd(f_{n+1}, f_n)$ is 0 for $n = 0, 1$ for $n = 1$, and $n - 1$ for $n \geq 2$. To prove this result for $n \geq 2$ we use mathematical induction. For $n = 2$, one division shows that $\gcd(f_3, f_2) = \gcd(2, 1) = \gcd(1, 0) = 1$. Now assume that $k - 1$ divisions are used to find $\gcd(f_{k+1}, f_k)$. To find $\gcd(f_{k+2}, f_{k+1})$, first divide f_{k+2} by f_{k+1} to obtain $f_{k+2} = 1 \cdot f_{k+1} + f_k$. After one division we have $\gcd(f_{k+2}, f_{k+1}) = \gcd(f_{k+1}, f_k)$. By the inductive hypothesis it follows that exactly $k - 1$ more divisions are required. This shows that k divisions are required to find $\gcd(f_{k+2}, f_{k+1})$, finishing the inductive proof.

19. $|A| = -1$. Hence, $|A^n| = (-1)^n$. It follows that $f_{n+1}f_{n-1} - f_n^2 = (-1)^n$.

21. a) Proof by induction. *Basis step:* For $n = 1$, $\max(-a_1) = -a_1 = -\min(a_1)$. For $n = 2$, there are two cases. If $a_2 \geq a_1$, then $-a_1 \geq -a_2$, so $\max(-a_1, -a_2) = -a_1 = -\min(a_1, a_2)$. If $a_2 < a_1$, then $-a_1 < -a_2$, so $\max(-a_1, -a_2) = -a_2 = -\min(a_1, a_2)$. *Inductive step:* Assume true for k with $k \geq 2$. Then $\max(-a_1, -a_2, \dots, -a_k, -a_{k+1}) = \max(\max(-a_1, \dots, -a_k), -a_{k+1}) = \max(-\min(a_1, \dots, a_k), -a_{k+1}) = -\min(\min(a_1, \dots, a_k), a_{k+1}) = -\min(a_1, \dots, a_{k+1})$.

b) Proof by mathematical induction. *Basis step:* For $n = 1$, the result is the identity $a_1 + b_1 = a_1 + b_1$. For $n = 2$, first consider the case in which $a_1 + b_1 \geq a_2 + b_2$. Then $\max(a_1 + b_1, a_2 + b_2) = a_1 + b_1$. Also note that $a_1 \leq \max(a_1, a_2)$ and $b_1 \leq \max(b_1, b_2)$, so $a_1 + b_1 \leq \max(a_1, a_2) + \max(b_1, b_2)$. Therefore, $\max(a_1 + b_1, a_2 + b_2) = a_1 + b_1 \leq \max(a_1, a_2) + \max(b_1, b_2)$. The case with $a_1 + b_1 < a_2 + b_2$ is similar. *Inductive step:* Assume that the result is true for k . Then $\max(a_1 + b_1, a_2 + b_2, \dots, a_k + b_k, a_{k+1} + b_{k+1}) = \max(\max(a_1 + b_1, a_2 + b_2, \dots, a_k + b_k), a_{k+1} + b_{k+1}) \leq \max(\max(a_1, a_2, \dots, a_k) + \max(b_1, b_2, \dots, b_k), a_{k+1} + b_{k+1}) \leq \max(\max(a_1, a_2, \dots, a_k), b_{k+1}) = \max(a_1, a_2, \dots, a_k, a_{k+1}) + \max(b_1, b_2, \dots, b_k, b_{k+1})$.

c) Same as part (b), but replace every occurrence of “max” by “min” and invert each inequality.

23. $5 \in S$, and $x + y \in S$ if $x, y \in S$.

25. a) $0 \in S$, and if $x \in S$, then $x + 2 \in S$ and $x - 2 \in S$.

b) $2 \in S$, and if $x \in S$, then $x + 3 \in S$.

c) $1 \in S, 2 \in S, 3 \in S, 4 \in S$, and if $x \in S$, then $x + 5 \in S$.

27. a) $(0, 1), (1, 1), (2, 1); (0, 2), (1, 2), (2, 2), (3, 2), (4, 2); (0, 3), (1, 3), (2, 3), (3, 3), (4, 3), (5, 3), (6, 3); (0, 4), (1, 4), (2, 4), (3, 4), (4, 4), (5, 4), (6, 4), (7, 4), (8, 4)$

b) Let $P(n)$ be the statement that $a \leq 2b$ whenever $(a, b) \in S$ is obtained by n applications of the recursive step. *Basis step:* $P(0)$ is true, because the only element of S obtained with no applications of the recursive step is $(0, 0)$, and indeed $0 \leq 2 \cdot 0$. *Inductive step:* Assume that $a \leq 2b$ whenever $(a, b) \in S$ is obtained by k or fewer applications of the recursive step, and consider an element obtained with $k + 1$ applications of the recursive step. Because the final application of the recursive step to an element (a, b) must be applied to an element obtained with fewer applications of the recursive step, we know that $a \leq 2b$. Add $0 \leq 2, 1 \leq 2$, and $2 \leq 2$, respectively, to obtain $a \leq 2(b + 1), a + 1 \leq 2(b + 1)$, and $a + 2 \leq 2(b + 1)$, as desired.

c) This holds for the basis step, because $0 \leq 0$. If this holds for (a, b) , then it also holds for the elements obtained from (a, b) in the recursive step, because adding $0 \leq 2, 1 \leq 2$, and $2 \leq 2$, respectively, to $a \leq 2b$ yields $a \leq 2(b + 1), a + 1 \leq 2(b + 1)$, and $a + 2 \leq 2(b + 1)$.

29. a) Define S by $(1, 1) \in S$, and if $(a, b) \in S$, then $(a + 2, b) \in S, (a, b + 2) \in S$, and $(a + 1, b + 1) \in S$. All elements put in S satisfy the condition, because $(1, 1)$ has an even sum of coordinates, and if (a, b) has an even sum of coordinates, then so do $(a + 2, b), (a, b + 2)$, and $(a + 1, b + 1)$. Conversely, we show by induction on the sum of the coordinates that if $a + b$ is even, then $(a, b) \in S$. If the sum is 2, then $(a, b) = (1, 1)$, and the basis step put (a, b) into S . Otherwise the sum is at least 4, and at least one of $(a - 2, b), (a, b - 2)$, and $(a - 1, b - 1)$ must have positive integer coordinates whose sum is an even number smaller than $a + b$, and therefore must be in S . Then one application of the recursive step shows that $(a, b) \in S$.

b) Define S by $(1, 1), (1, 2)$, and $(2, 1)$ are in S , and if $(a, b) \in S$, then $(a + 2, b)$ and $(a, b + 2)$ are in S . To prove that our definition works, we note first that $(1, 1), (1, 2)$, and $(2, 1)$ all have an odd coordinate, and if (a, b) has an odd coordinate, then so do $(a + 2, b)$ and $(a, b + 2)$. Conversely, we show by induction on the sum of the coordinates that if (a, b) has at least one odd coordinate, then $(a, b) \in S$. If $(a, b) = (1, 1)$ or $(a, b) = (1, 2)$ or $(a, b) = (2, 1)$, then the basis step put (a, b) into S . Otherwise either a or b is at least 3, so at least one of $(a - 2, b)$ and $(a, b - 2)$ must have positive integer coordinates whose sum is smaller than $a + b$, and therefore must be in S . Then one application of the recursive step shows that $(a, b) \in S$.

c) $(1, 6) \in S$ and $(2, 3) \in S$, and if $(a, b) \in S$, then $(a + 2, b) \in S$ and $(a, b + 6) \in S$. To prove that our definition works, we note first that $(1, 6)$ and $(2, 3)$ satisfy the condition, and if (a, b) satisfies the condition, then so do $(a + 2, b)$ and $(a, b + 6)$. Conversely we show by induction on the sum of the coordinates that if (a, b) satisfies the condition, then $(a, b) \in S$. For sums 5 and 7, the only points are $(1, 6)$, which the basis step put into S , $(2, 3)$, which the basis step put into S , and $(4, 3) = (2 + 2, 3)$, which is in S by one application of the recursive definition. For a sum greater than 7, either $a \geq 3$, or

$a \leq 2$ and $b \geq 9$, in which case either $(a - 2, b)$ or $(a, b - 6)$ must have positive integer coordinates whose sum is smaller than $a + b$ and satisfy the condition for being in S . Then one application of the recursive step shows that $(a, b) \in S$.

31. If x is a set or a variable representing a set, then x is a well-formed formula. If x and y are well-formed formulae, then so are \bar{x} , $(x \cup y)$, $(x \cap y)$, and $(x - y)$. **33. a)** If $x \in D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, then $m(x) = x$; if $s = tx$, where $t \in D^*$ and $x \in D$, then $m(s) = \min(m(s), x)$. **b)** Let $t = wx$, where $w \in D^*$ and $x \in D$. If $w = \lambda$, then $m(st) = m(sx) = \min(m(s), x) = \min(m(s), m(x))$ by the recursive step and the basis step of the definition of m . Otherwise, $m(st) = m((sw)x) = \min(m(sw), x)$ by the definition of m . Now $m(sw) = \min(m(s), m(w))$ by the inductive hypothesis of the structural induction, so $m(st) = \min(\min(m(s), m(w)), x) = \min(m(s), \min(m(w), x))$ by the meaning of \min . But $\min(m(w), x) = m(wx) = m(t)$ by the recursive step of the definition of m . Thus, $m(st) = \min(m(s), m(t))$. **35.** $\lambda^R = \lambda$ and $(ux)^R = xu^R$ for $x \in \Sigma$, $u \in \Sigma^*$. **37.** $w^0 = \lambda$ and $w^{n+1} = ww^n$. **39.** When the string consists of n 0s followed by n 1s for some non-negative integer n . **41.** Let $P(i)$ be “ $l(w^i) = i \cdot l(w)$.” $P(0)$ is true because $l(w^0) = 0 = 0 \cdot l(w)$. Assume $P(i)$ is true. Then $l(w^{i+1}) = l(ww^i) = l(w) + l(w^i) = l(w) + i \cdot l(w) = (i+1) \cdot l(w)$. **43. Basis step:** For the full binary tree consisting of just a root the result is true because $n(T) = 1$ and $h(T) = 0$, and $1 \geq 2 \cdot 0 + 1$. **Inductive step:** Assume that $n(T_1) \geq 2h(T_1) + 1$ and $n(T_2) \geq 2h(T_2) + 1$. By the recursive definitions of $n(T)$ and $h(T)$, we have $n(T) = 1 + n(T_1) + n(T_2)$ and $h(T) = 1 + \max(h(T_1), h(T_2))$. Therefore $n(T) = 1 + n(T_1) + n(T_2) \geq 1 + 2h(T_1) + 1 + 2h(T_2) + 1 \geq 1 + 2 \cdot \max(h(T_1), h(T_2)) + 2 = 1 + 2(\max(h(T_1), h(T_2)) + 1) = 1 + 2h(T)$. **45. Basis step:** $a_{0,0} = 0 = 0 + 0$. **Inductive step:** Assume that $a_{m',n'} = m' + n'$ whenever (m', n') is less than (m, n) in the lexicographic ordering of $\mathbb{N} \times \mathbb{N}$. If $n = 0$ then $a_{m,n} = a_{m-1,n} + 1 = m - 1 + n + 1 = m + n$. If $n > 0$, then $a_{m,n} = a_{m,n-1} + 1 = m + n - 1 + 1 = m + n$.

47. a) $P_{m,m} = P_m$ because a number exceeding m cannot be used in a partition of m . **b)** Because there is only one way to partition 1, namely, $1 = 1$, it follows that $P_{1,1} = 1$. Because there is only one way to partition m into 1s, $P_{m,1} = 1$. When $n > m$ it follows that $P_{m,n} = P_{m,m}$ because a number exceeding m cannot be used. $P_{m,m} = 1 + P_{m,m-1}$ because one extra partition, namely, $m = m$, arises when m is allowed in the partition. $P_{m,n} = P_{m,n-1} + P_{m-n,n}$ if $m > n$ because a partition of m into integers not exceeding n either does not use any n s and hence, is counted in $P_{m,n-1}$ or else uses an n and a partition of $m - n$, and hence, is counted in $P_{m-n,n}$. **c)** $P_5 = 7$, $P_6 = 11$.

49. Let $P(n)$ be “ $A(n, 2) = 4$.” **Basis step:** $P(1)$ is true because $A(1, 2) = A(0, A(1, 1)) = A(0, 2) = 2 \cdot 2 = 4$. **Inductive step:** Assume that $P(n)$ is true, that is, $A(n, 2) = 4$. Then $A(n+1, 2) = A(n, A(n+1, 1)) = A(n, 2) = 4$.

51. a) 16 **b)** 65,536 **53.** Use a double induction argument to prove the stronger statement: $A(m, k) > A(m, l)$ when $k > l$. **Basis step:** When $m = 0$ the statement is true because

$k > l$ implies that $A(0, k) = 2k > 2l = A(0, l)$. **Inductive step:** Assume that $A(m, x) > A(m, y)$ for all nonnegative integers x and y with $x > y$. We will show that this implies that $A(m+1, k) > A(m+1, l)$ if $k > l$. **Basis steps:** When $l = 0$ and $k > 0$, $A(m+1, l) = 0$ and either $A(m+1, k) = 2$ or $A(m+1, k) = A(m, A(m+1, k-1))$. If $m = 0$, this is $2A(1, k-1) = 2^k$. If $m > 0$, this is greater than 0 by the inductive hypothesis. In all cases, $A(m+1, k) > 0$, and in fact, $A(m+1, k) \geq 2$. If $l = 1$ and $k > 1$, then $A(m+1, l) = 2$ and $A(m+1, k) = A(m, A(m+1, k-1))$, with $A(m+1, k-1) \geq 2$. Hence, by the inductive hypothesis, $A(m, A(m+1, k-1)) \geq A(m, 2) > A(m, 1) = 2$. **Inductive step:** Assume that $A(m+1, r) > A(m+1, s)$ for all $r > s$, $s = 0, 1, \dots, l$. Then if $k+1 > l+1$ it follows that $A(m+1, k+1) = A(m, A(m+1, k)) > A(m, A(m+1, k)) = A(m+1, l+1)$. **55.** From Exercise 54 it follows that $A(i, j) \geq A(i-1, j) \geq \dots \geq A(0, j) = 2j \geq j$.

57. Let $P(n)$ be “ $F(n)$ is well-defined.” Then $P(0)$ is true because $F(0)$ is specified. Assume that $P(k)$ is true for all $k < n$. Then $F(n)$ is well-defined at n because $F(n)$ is given in terms of $F(0), F(1), \dots, F(n-1)$. So $P(n)$ is true for all integers n . **59. a)** The value of $F(1)$ is ambiguous. **b)** $F(2)$ is not defined because $F(0)$ is not defined. **c)** $F(3)$ is ambiguous and $F(4)$ is not defined because $F(\frac{4}{3})$ makes no sense. **d)** The definition of $F(1)$ is ambiguous because both the second and third clause seem to apply. **e)** $F(2)$ cannot be computed because trying to compute $F(2)$ gives $F(2) = 1 + F(F(1)) = 1 + F(2)$. **61. a)** 1 **b)** 2 **c)** 3 **d)** 3 **e)** 4 **f)** 4 **g)** 5 **63.** $f_0^*(n) = \lceil n/a \rceil$ **65.** $f_2^*(n) = \lceil \log \log n \rceil$ for $n \geq 2$, $f_2^*(1) = 0$

Section 5.4

1. First, we use the recursive step to write $5! = 5 \cdot 4!$. We then use the recursive step repeatedly to write $4! = 4 \cdot 3!$, $3! = 3 \cdot 2!$, $2! = 2 \cdot 1!$, and $1! = 1 \cdot 0!$. Inserting the value of $0! = 1$, and working back through the steps, we see that $1! = 1 \cdot 1 = 1$, $2! = 2 \cdot 1! = 2 \cdot 1 = 2$, $3! = 3 \cdot 2! = 3 \cdot 2 = 6$, $4! = 4 \cdot 3! = 4 \cdot 6 = 24$, and $5! = 5 \cdot 4! = 5 \cdot 24 = 120$.

3. With this input, the algorithm uses the **else** clause to find that $\gcd(8, 13) = \gcd(13 \bmod 8, 8) = \gcd(5, 8)$. It uses this clause again to find that $\gcd(5, 8) = \gcd(8 \bmod 5, 5) = \gcd(3, 5)$, then to get $\gcd(3, 5) = \gcd(5 \bmod 3, 3) = \gcd(2, 3)$, then $\gcd(2, 3) = \gcd(3 \bmod 2, 2) = \gcd(1, 2)$, and once more to get $\gcd(1, 2) = \gcd(2 \bmod 1, 1) = \gcd(0, 1)$. Finally, to find $\gcd(0, 1)$ it uses the first step with $a = 0$ to find that $\gcd(0, 1) = 1$. Consequently, the algorithm finds that $\gcd(8, 13) = 1$. **5.** First, because $n = 11$ is odd, we use the **else** clause to see that $\text{mpower}(3, 11, 5) = (\text{mpower}(3, 5, 5)^2 \bmod 5 \cdot 3 \bmod 5) \bmod 5$. We next use the **else** clause again to see that $\text{mpower}(3, 5, 5) = (\text{mpower}(3, 2, 5)^2 \bmod 5 \cdot 3 \bmod 5) \bmod 5$. Then we use the **else if** clause to see that $\text{mpower}(3, 2, 5) = \text{mpower}(3, 1, 5)^2 \bmod 5$. Using the **else** clause again, we have $\text{mpower}(3, 1, 5) = (\text{mpower}(3, 0, 5)^2 \bmod 5 \cdot 3 \bmod 5) \bmod 5$. Finally, us-

ing the **if** clause, we see that $\text{mpower}(3, 0, 5) = 1$. Working backward it follows that $\text{mpower}(3, 1, 5) = (1^2 \bmod 5 \cdot 3 \bmod 5) \bmod 5 = 3$, $\text{mpower}(3, 2, 5) = (3^2 \bmod 5 = 4, \text{mpower}(3, 5, 5) = (4^2 \bmod 5 \cdot 3 \bmod 5) \bmod 5 = 3$, and finally $\text{mpower}(3, 11, 5) = (3^2 \bmod 5 \cdot 3 \bmod 5) \bmod 5 = 2$. We conclude that $3^{11} \bmod 5 = 2$.

7. **procedure** *mult*(*n*: positive integer, *x*: integer)
if *n* = 1 **then return** *x*
else return *x* + *mult*(*n* − 1, *x*)
9. **procedure** *sum of odds*(*n*: positive integer)
if *n* = 1 **then return** 1
else return *sum of odds*(*n* − 1) + 2*n* − 1
11. **procedure** *smallest*(*a*₁, ..., *a*_{*n*}: integers)
if *n* = 1 **then return** *a*₁
else return
 $\min(\text{smallest}(a_1, \dots, a_{n-1}), a_n)$
13. **procedure** *modfactorial*(*n*, *m*: positive integers)
if *n* = 1 **then return** 1
else return
 $(n \cdot \text{modfactorial}(n - 1, m)) \bmod m$
15. **procedure** *gcd*(*a*, *b*: nonnegative integers)
{*a* < *b* assumed to hold}
if *a* = 0 **then return** *b*
else if *a* = *b* − *a* **then return** *a*
else if *a* < *b* − *a* **then return** *gcd*(*a*, *b* − *a*)
else return *gcd*(*b* − *a*, *a*)
17. **procedure** *multiply*(*x*, *y*: nonnegative integers)
if *y* = 0 **then return** 0
else if *y* is even **then**
return 2 · *multiply*(*x*, *y*/2)
else return 2 · *multiply*(*x*, (*y* − 1)/2) + *x*

19. We use strong induction on *a*. *Basis step*: If *a* = 0, we know that $\text{gcd}(0, b) = b$ for all *b* > 0, and that is precisely what the **if** clause does. *Inductive step*: Fix *k* > 0, assume the inductive hypothesis—that the algorithm works correctly for all values of its first argument less than *k*—and consider what happens with input (*k*, *b*), where *k* < *b*. Because *k* > 0, the **else** clause is executed, and the answer is whatever the algorithm gives as output for inputs (*b* **mod** *k*, *k*). Because *b* **mod** *k* < *k*, the input pair is valid. By our inductive hypothesis, this output is in fact $\text{gcd}(b \bmod k, k)$, which equals $\text{gcd}(k, b)$ by Lemma 1 in Section 4.3. 21. If *n* = 1, then $nx = x$, and the algorithm correctly returns *x*. Assume that the algorithm correctly computes *kx*. To compute $(k+1)x$ it recursively computes the product of $k+1-1 = k$ and *x*, and then adds *x*. By the inductive hypothesis, it computes that product correctly, so the answer returned is $kx+x = (k+1)x$, which is correct.

23. **procedure** *square*(*n*: nonnegative integer)
if *n* = 0 **then return** 0
else return *square*(*n* − 1) + 2(*n* − 1) + 1

Let *P*(*n*) be the statement that this algorithm correctly computes n^2 . Because $0^2 = 0$, the algorithm works correctly (using the **if** clause) if the input is 0. Assume that the algorithm works correctly for input *k*. Then for input *k* + 1, it

gives as output (because of the **else** clause) its output when the input is *k*, plus $2(k+1-1) + 1$. By the inductive hypothesis, its output at *k* is k^2 , so its output at *k* + 1 is $k^2 + 2(k+1-1) + 1 = k^2 + 2k + 1 = (k+1)^2$, as desired.

25. *n* multiplications versus 2^n 27. $O(\log n)$ versus *n*

29. **procedure** *a*(*n*: nonnegative integer)
if *n* = 0 **then return** 1
else if *n* = 1 **then return** 2
else return *a*(*n* − 1) · *a*(*n* − 2)
31. Iterative
33. **procedure** *iterative*(*n*: nonnegative integer)
if *n* = 0 **then** *z* := 1
else if *n* = 1 **then** *z* := 2
else
x := 1
y := 2
z := 3
for *i* := 1 **to** *n* − 2
w := *x* + *y* + *z*
x := *y*
y := *z*
z := *w*
return *z* {*z* is the *n*th term of the sequence}

35. We first give a recursive procedure and then an iterative procedure.

- procedure** *r*(*n*: nonnegative integer)
if *n* < 3 **then return** $2n + 1$
else return *r*(*n* − 1) · (*r*(*n* − 2))² · (*r*(*n* − 3))³

- procedure** *i*(*n*: nonnegative integer)
if *n* = 0 **then** *z* := 1
else if *n* = 1 **then** *z* := 3
else
x := 1
y := 3
z := 5
for *i* := 1 **to** *n* − 2
w := *z* · *y*² · *x*³
x := *y*
y := *z*
z := *w*
return *z* {*z* is the *n*th term of the sequence}

The iterative version is more efficient.

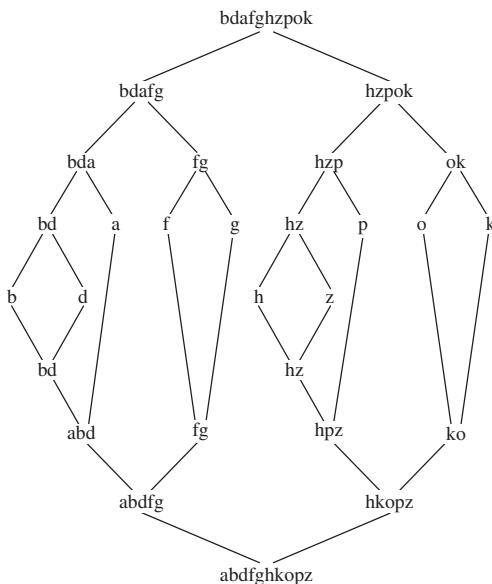
37. **procedure** *reverse*(*w*: bit string)
n := length(*w*)
if *n* ≤ 1 **then return** *w*
else return
 $\text{substr}(w, n, n)\text{reverse}(\text{substr}(w, 1, n-1))$
{*substr*(*w*, *a*, *b*) is the substring of *w* consisting of the symbols in the *a*th through *b*th positions}

39. The procedure correctly gives the reversal of λ as λ (basis step), and because the reversal of a string consists of its last character followed by the reversal of its first *n* − 1 characters (see Exercise 35 in Section 5.3), the algorithm behaves correctly when *n* > 0 by the inductive hypothesis. 41. The

algorithm implements the idea of Example 14 in Section 5.1. If $n = 1$ (basis step), place the one right triomino so that its armpit corresponds to the hole in the 2×2 board. If $n > 1$, then divide the board into four boards, each of size $2^{n-1} \times 2^{n-1}$, notice which quarter the hole occurs in, position one right triomino at the center of the board with its armpit in the quarter where the missing square is (see Figure 7 in Section 5.1), and invoke the algorithm recursively four times—once on each of the $2^{n-1} \times 2^{n-1}$ boards, each of which has one square missing (either because it was missing to begin with, or because it is covered by the central triomino).

43. procedure $A(m, n)$: nonnegative integers)
if $m = 0$ **then return** $2n$
else if $n = 0$ **then return** 0
else if $n = 1$ **then return** 2
else return $A(m - 1, A(m, n - 1))$

45.



47. Let the two lists be $1, 2, \dots, m - 1, m + n - 1$ and $m, m + 1, \dots, m + n - 2, m + n$, respectively. **49.** If $n = 1$, then the algorithm does nothing, which is correct because a list with one element is already sorted. Assume that the algorithm works correctly for $n = 1$ through $n = k$. If $n = k + 1$, then the list is split into two lists, L_1 and L_2 . By the inductive hypothesis, *mergesort* correctly sorts each of these sublists; furthermore, *merge* correctly merges two sorted lists into one because with each comparison the smallest element in $L_1 \cup L_2$ not yet put into L is put there. **51.** $O(n)$ **53.** 6 **55.** $O(n^2)$

Section 5.5

1. Suppose that $x = 0$. The program segment first assigns the value 1 to y and then assigns the value $x + y = 0 + 1 = 1$ to z . **3.** Suppose that $y = 3$. The program segment assigns the value 2 to x and then assigns the value $x + y = 2 + 3 = 5$ to z .

Because $y = 3 > 0$ it then assigns the value $z + 1 = 5 + 1 = 6$ to z .

$$\begin{aligned} \mathbf{5.} \quad & (p \wedge \text{condition1})\{S_1\}q \\ & (p \wedge \neg \text{condition1} \wedge \text{condition2})\{S_2\}q \\ & \quad \vdots \\ & (p \wedge \neg \text{condition1} \wedge \neg \text{condition2}) \\ & \quad \quad \quad \cdots \wedge \neg \text{condition}(n-1)\{S_n\}q \end{aligned}$$

$$\therefore \begin{array}{l} p \{\text{if condition1 then } S_1; \\ \quad \quad \quad \text{else if condition2 then } S_2; \dots; \text{ else } S_n\}q \end{array}$$

7. We will show that p : “ $\text{power} = x^{i-1}$ and $i \leq n + 1$ ” is a loop invariant. Note that p is true initially, because before the loop starts, $i = 1$ and $\text{power} = 1 = x^0 = x^{1-1}$. Next, we must show that if p is true and $i \leq n$ after an execution of the loop, then p remains true after one more execution. The loop increments i by 1. Hence, because $i \leq n$ before this pass, $i \leq n + 1$ after this pass. Also the loop assigns $\text{power} \cdot x$ to power . By the inductive hypothesis we see that power is assigned the value $x^{i-1} \cdot x = x^i$. Hence, p remains true. Furthermore, the loop terminates after n traversals of the loop with $i = n + 1$ because i is assigned the value 1 prior to entering the loop, is incremented by 1 on each pass, and the loop terminates when $i > n$. Consequently, at termination $\text{power} = x^n$, as desired.

9. Suppose that p is “ m and n are integers.” Then if the condition $n < 0$ is true, $a = -n = |n|$ after S_1 is executed. If the condition $n < 0$ is false, then $a = n = |n|$ after S_1 is executed. Hence, $p\{S_1\}q$ is true where q is $p \wedge (a = |n|)$. Because S_2 assigns the value 0 to both k and x , it is clear that $q\{S_2\}r$ is true where r is $q \wedge (k = 0) \wedge (x = 0)$. Suppose that r is true. Let $P(k)$ be “ $x = mk$ and $k \leq a$.” We can show that $P(k)$ is a loop invariant for the loop in S_3 . $P(0)$ is true because before the loop is entered $x = 0 = m \cdot 0$ and $0 \leq a$. Now assume $P(k)$ is true and $k < a$. Then $P(k+1)$ is true because x is assigned the value $x + m = mk + m = m(k+1)$. The loop terminates when $k = a$, and at that point $x = ma$. Hence, $r\{S_3\}s$ is true where s is “ $a = |n|$ and $x = ma$.” Now assume that s is true. Then if $n < 0$ it follows that $a = -n$, so $x = -mn$. In this case S_4 assigns $-x = mn$ to $product$. If $n > 0$ then $x = ma = mn$, so S_4 assigns mn to $product$. Hence, $s\{S_4\}t$ is true. **11.** Suppose that the initial assertion p is true. Then because $p\{S\}q_0$ is true, q_0 is true after the segment S is executed. Because $q_0 \rightarrow q_1$ is true, it also follows that q_1 is true after S is executed. Hence, $p\{S\}q_1$ is true. **13.** We will use the proposition p , “ $\text{gcd}(a, b) = \text{gcd}(x, y)$ and $y \geq 0$,” as the loop invariant. Note that p is true before the loop is entered, because at that point $x = a$, $y = b$, and y is a positive integer, using the initial assertion. Now assume that p is true and $y > 0$; then the loop will be executed again. Inside the loop, x and y are replaced by y and $x \bmod y$, respectively. By Lemma 1 of Section 4.3, $\text{gcd}(x, y) = \text{gcd}(y, x \bmod y)$. Therefore, after execution of the loop, the value of $\text{gcd}(x, y)$ is the same as it was before. Moreover, because y is the remainder, it is at least 0. Hence, p remains true, so it is a loop invariant. Furthermore, if the loop terminates, then $y = 0$. In this case, we have $\text{gcd}(x, y) = x$, the final assertion. Therefore, the program, which gives x as its output, has correctly computed

$\gcd(a, b)$. Finally, we can prove the loop must terminate, because each iteration causes the value of y to decrease by at least 1. Therefore, the loop can be iterated at most b times.

Supplementary Exercises

1. Let $P(n)$ be the statement that this equation holds. *Basis step:* $P(1)$ says $2/3 = 1 - (1/3^1)$, which is true. *Inductive step:* Assume that $P(k)$ is true. Then $2/3 + 2/9 + 2/27 + \dots + 2/3^n + 2/3^{n+1} = 1 - 1/3^n + 2/3^{n+1}$ (by the inductive hypothesis), and this equals $1 - 1/3^{n+1}$, as desired. 3. Let $P(n)$ be “ $1 \cdot 1 + 2 \cdot 2 + \dots + n \cdot 2^{n-1} = (n-1)2^n + 1$.” *Basis step:* $P(1)$ is true because $1 \cdot 1 = 1 = (1-1)2^1 + 1$. *Inductive step:* Assume that $P(k)$ is true. Then $1 \cdot 1 + 2 \cdot 2 + \dots + k \cdot 2^{k-1} + (k+1) \cdot 2^k = (k-1)2^k + 1 + (k+1)2^k = 2k \cdot 2^k + 1 = [(k+1)-1]2^{k+1} + 1$. 5. Let $P(n)$ be “ $1/(1 \cdot 4) + \dots + 1/[(3n-2)(3n+1)] = n/(3n+1)$.” *Basis step:* $P(1)$ is true because $1/(1 \cdot 4) = 1/4$. *Inductive step:* Assume $P(k)$ is true. Then $1/(1 \cdot 4) + \dots + 1/[(3k-2)(3k+1)] + 1/[(3k+1)(3k+4)] = k/(3k+1) + 1/[(3k+1)(3k+4)] = [k(3k+4) + 1]/[(3k+1)(3k+4)] = [(3k+1)(k+1)]/[(3k+1)(3k+4)] = (k+1)/(3k+4)$. 7. Let $P(n)$ be “ $2^n > n^3$.” *Basis step:* $P(10)$ is true because $1024 > 1000$. *Inductive step:* Assume $P(k)$ is true. Then $(k+1)^3 = k^3 + 3k^2 + 3k + 1 \leq k^3 + 9k^2 \leq k^3 + k^3 = 2k^3 < 2 \cdot 2^k = 2^{k+1}$. 9. Let $P(n)$ be “ $a - b$ is a factor of $a^n - b^n$.” *Basis step:* $P(1)$ is trivially true. Assume $P(k)$ is true. Then $a^{k+1} - b^{k+1} = a^{k+1} - ab^k + ab^k - b^{k+1} = a(a^k - b^k) + b^k(a - b)$. Then because $a - b$ is a factor of $a^k - b^k$ and $a - b$ is a factor of $a - b$, it follows that $a - b$ is a factor of $a^{k+1} - b^{k+1}$. 11. *Basis step:* When $n = 1$, $6^{n+1} + 7^{2n-1} = 36 + 7 = 43$. *Inductive step:* Assume the inductive hypothesis, that 43 divides $6^{n+1} + 7^{2n-1}$; we must show that 43 divides $6^{n+2} + 7^{2n+1}$. We have $6^{n+2} + 7^{2n+1} = 6 \cdot 6^{n+1} + 49 \cdot 7^{2n-1} = 6 \cdot 6^{n+1} + 6 \cdot 7^{2n-1} + 43 \cdot 7^{2n-1} = 6(6^{n+1} + 7^{2n-1}) + 43 \cdot 7^{2n-1}$. By the inductive hypothesis the first term is divisible by 43, and the second term is clearly divisible by 43; therefore the sum is divisible by 43. 13. Let $P(n)$ be “ $a + (a+d) + \dots + (a+nd) = (n+1)(2a+nd)/2$.” *Basis step:* $P(1)$ is true because $a + (a+d) = 2a+d = 2(2a+d)/2$. *Inductive step:* Assume that $P(k)$ is true. Then $a + (a+d) + \dots + (a+kd) + [a + (k+1)d] = (k+1)(2a+kd)/2 + a + (k+1)d = \frac{1}{2}(2ak+2a+k^2d+kd+2a+2kd+2d) = \frac{1}{2}(2ak+4a+k^2d+3kd+2d) = \frac{1}{2}(k+2)[2a+(k+1)d]$. 15. *Basis step:* This is true for $n = 1$ because $5/6 = 10/12$. *Inductive step:* Assume that the equation holds for $n = k$, and consider $n = k+1$. Then $\sum_{i=1}^{k+1} \frac{i+4}{i(i+1)(i+2)} = \sum_{i=1}^k \frac{i+4}{i(i+1)(i+2)} + \frac{k+5}{(k+1)(k+2)(k+3)} = \frac{1}{2(k+1)(k+2)} + \frac{k+5}{(k+1)(k+2)(k+3)}$ (by the inductive hypothesis) $= \frac{1}{(k+1)(k+2)} \cdot \left(\frac{k(3k+7)}{2} + \frac{k+5}{k+3}\right) = \frac{1}{2(k+1)(k+2)(k+3)} \cdot [k(3k+7)(k+3) + 2(k+5)] = \frac{1}{2(k+1)(k+2)(k+3)} \cdot (3k^3+16k^2+23k+10) = \frac{1}{2(k+1)(k+2)(k+3)} \cdot (3k+10)(k+1)^2 = \frac{1}{2(k+2)(k+3)} \cdot (3k+10)(k+1) = \frac{(k+1)(3(k+1)+7)}{2((k+1)+1)((k+1)+2)}$, as desired. 17. *Basis step:* The statement is true for $n = 1$ be-

cause the derivative of $g(x) = xe^x$ is $x \cdot e^x + e^x = (x+1)e^x$ by the product rule. *Inductive step:* Assume that the statement is true for $n = k$, i.e., the k th derivative is given by $g^{(k)} = (x+k)e^x$. Differentiating by the product rule gives the $(k+1)$ st derivative: $g^{(k+1)} = (x+k)e^x + e^x = [x+(k+1)]e^x$, as desired. 19. We will use strong induction to show that f_n is even if $n \equiv 0 \pmod{3}$ and is odd otherwise. *Basis step:* This follows because $f_0 = 0$ is even and $f_1 = 1$ is odd. *Inductive step:* Assume that if $j \leq k$, then f_j is even if $j \equiv 0 \pmod{3}$ and is odd otherwise. Now suppose $k+1 \equiv 0 \pmod{3}$. Then $f_{k+1} = f_k + f_{k-1}$ is even because f_k and f_{k-1} are both odd. If $k+1 \equiv 1 \pmod{3}$, then $f_{k+1} = f_k + f_{k-1}$ is odd because f_k is even and f_{k-1} is odd. Finally, if $k+1 \equiv 2 \pmod{3}$, then $f_{k+1} = f_k + f_{k-1}$ is odd because f_k is odd and f_{k-1} is even. 21. Let $P(n)$ be the statement that $f_k f_n + f_{k+1} f_{n+1} = f_{n+k+1}$ for every nonnegative integer k . *Basis step:* This consists of showing that $P(0)$ and $P(1)$ both hold. $P(0)$ is true because $f_k f_0 + f_{k+1} f_1 = f_{k+1} \cdot 0 + f_{k+1} \cdot 1 = f_1$. Because $f_k f_1 + f_{k+1} f_2 = f_k + f_{k+1} = f_{k+2}$, it follows that $P(1)$ is true. *Inductive step:* Now assume that $P(j)$ holds. Then, by the inductive hypothesis and the recursive definition of the Fibonacci numbers, it follows that $f_{k+1} f_{j+1} + f_{k+2} f_{j+2} = f_k(f_{j-1} + f_j) + f_{k+1}(f_j + f_{j+1}) = (f_k f_{j-1} + f_{k+1} f_j) + (f_k f_j + f_{k+1} f_{j+1}) = f_{j-1+k+1} + f_{j+k+1} = f_{j+k+2}$. This shows that $P(j+1)$ is true. 23. Let $P(n)$ be the statement $l_0^2 + l_1^2 + \dots + l_n^2 = l_n l_{n+1} + 2$. *Basis step:* $P(0)$ and $P(1)$ both hold because $l_0^2 = 2^2 = 2 \cdot 1 + 2 = l_0 l_1 + 2$ and $l_0^2 + l_1^2 = 2^2 + 1^2 = 1 \cdot 3 + 2 = l_1 l_3 + 2$. *Inductive step:* Assume that $P(k)$ holds. Then by the inductive hypothesis $l_0^2 + l_1^2 + \dots + l_k^2 + l_{k+1}^2 = l_k l_{k+1} + 2 + l_{k+1}^2 = l_{k+1}(l_k + l_{k+1}) + 2 = l_{k+1} l_{k+2} + 2$. This shows that $P(k+1)$ holds. 25. Let $P(n)$ be the statement that the identity holds for the integer n . *Basis step:* $P(1)$ is obviously true. *Inductive step:* Assume that $P(k)$ is true. Then $\cos((k+1)x) + i \sin((k+1)x) = \cos(kx+x) + i \sin(kx+x) = \cos kx \cos x - \sin kx \sin x + i(\sin kx \cos x + \cos kx \sin x) = \cos x(\cos kx + i \sin kx)(\cos x + i \sin x) = (\cos x + i \sin x)^k(\cos x + i \sin x) = (\cos x + i \sin x)^{k+1}$. It follows that $P(k+1)$ is true. 27. Rewrite the right-hand side as $2^{n+1}(n^2 - 2n + 3) - 6$. For $n = 1$ we have $2 = 4 \cdot 2 - 6$. Assume that the equation holds for $n = k$, and consider $n = k+1$. Then $\sum_{j=1}^{k+1} j^2 2^j = \sum_{j=1}^k j^2 2^j + (k+1)^2 2^{k+1} = 2^{k+1}(k^2 - 2k + 3) - 6 + (k^2 + 2k + 1)2^{k+1}$ (by the inductive hypothesis) $= 2^{k+1}(2k^2 + 4) - 6 = 2^{k+2}(k^2 + 2) - 6 = 2^{k+2}[(k+1)^2 - 2(k+1) + 3] - 6$. 29. Let $P(n)$ be the statement that this equation holds. *Basis step:* In $P(2)$ both sides reduce to $1/3$. *Inductive step:* Assume that $P(k)$ is true. Then $\sum_{j=1}^{k+1} 1/(j^2 - 1) = \left(\sum_{j=1}^k 1/(j^2 - 1)\right) + 1/[(k+1)^2 - 1] = (k-1)(3k+2)/[4k(k+1)] + 1/[(k+1)^2 - 1]$ by the inductive hypothesis. This simplifies to $(k-1)(3k+2)/[4k(k+1)] + 1/(k^2 + 2k) = (3k^3 + 5k^2)/[4k(k+1)(k+2)] = [(k+1)-1][3(k+1)+2]/[4(k+1)(k+2)]$, which is exactly what $P(k+1)$ asserts. 31. Let $P(n)$ be the assertion that at least $n+1$ lines are needed to cover the lattice points in the given triangular region. *Basis step:* $P(0)$ is true, because

we need at least one line to cover the one point at $(0, 0)$. *Inductive step:* Assume the inductive hypothesis, that at least $k + 1$ lines are needed to cover the lattice points with $x \geq 0, y \geq 0$, and $x + y \leq k$. Consider the triangle of lattice points defined by $x \geq 0, y \geq 0$, and $x + y \leq k + 1$. By way of contradiction, assume that $k + 1$ lines could cover this set. Then these lines must cover the $k + 2$ points on the line $x + y = k + 1$. But only the line $x + y = k + 1$ itself can cover more than one of these points, because two distinct lines intersect in at most one point. Therefore none of the $k + 1$ lines that are needed (by the inductive hypothesis) to cover the set of lattice points within the triangle but not on this line can cover more than one of the points on this line, and this leaves at least one point uncovered. Therefore our assumption that $k + 1$ lines could cover the larger set is wrong, and our proof is complete.

33. Let $P(n)$ be $\mathbf{B}^k = \mathbf{M}\mathbf{A}^k\mathbf{M}^{-1}$. *Basis step:* Part of the given conditions. *Inductive step:* Assume the inductive hypothesis. Then $\mathbf{B}^{k+1} = \mathbf{B}\mathbf{B}^k = \mathbf{M}\mathbf{A}\mathbf{M}^{-1}\mathbf{B}^k = \mathbf{M}\mathbf{A}\mathbf{M}^{-1}\mathbf{M}\mathbf{A}^k\mathbf{M}^{-1}$ (by the inductive hypothesis) = $\mathbf{M}\mathbf{A}\mathbf{I}\mathbf{A}^k\mathbf{M}^{-1} = \mathbf{M}\mathbf{A}\mathbf{A}^k\mathbf{M}^{-1} = \mathbf{M}\mathbf{A}^{k+1}\mathbf{M}^{-1}$. **35.** We prove by mathematical induction the following stronger statement: For every $n \geq 3$, we can write $n!$ as the sum of n of its distinct positive divisors, one of which is 1. That is, we can write $n! = a_1 + a_2 + \dots + a_n$, where each a_i is a divisor of $n!$, the divisors are listed in strictly decreasing order, and $a_n = 1$. *Basis step:* $3! = 3 + 2 + 1$. *Inductive step:* Assume that we can write $k!$ as a sum of the desired form, say $k! = a_1 + a_2 + \dots + a_k$, where each a_i is a divisor of $n!$, the divisors are listed in strictly decreasing order, and $a_n = 1$. Consider $(k+1)!$. Then we have $(k+1)! = (k+1)k! = (k+1)(a_1 + a_2 + \dots + a_k) = (k+1)a_1 + (k+1)a_2 + \dots + (k+1)a_k = (k+1)a_1 + (k+1)a_2 + \dots + k \cdot a_k + a_k$. Because each a_i was a divisor of $k!$, each $(k+1)a_i$ is a divisor of $(k+1)!$. Furthermore, $k \cdot a_k = k$, which is a divisor of $(k+1)!$, and $a_k = 1$, so the new last summand is again 1. (Notice also that our list of summands is still in strictly decreasing order.) Thus we have written $(k+1)!$ in the desired form. **37.** When $n = 1$ the statement is vacuously true. Assume that the statement is true for $n = k$, and consider $k + 1$ people standing in a line, with a woman first and a man last. If the k th person is a woman, then we have that woman standing in front of the man at the end. If the k th person is a man, then the first k people in line satisfy the conditions of the inductive hypothesis for the first k people in line, so again we can conclude that there is a woman directly in front of a man somewhere in the line. **39.** *Basis step:* When $n = 1$ there is one circle, and we can color the inside blue and the outside red to satisfy the conditions. *Inductive step:* Assume the inductive hypothesis that if there are k circles, then the regions can be 2-colored such that no regions with a common boundary have the same color, and consider a situation with $k + 1$ circles. Remove one of the circles, producing a picture with k circles, and invoke the inductive hypothesis to color it in the prescribed manner. Then replace the removed circle and change the color of every region inside this circle. The resulting figure satisfies the condition, because if two regions have a common boundary, then either that boundary involved the new circle, in which

$$\begin{aligned} & \text{case the regions on either side used to be the same region and} \\ & \text{now the inside portion is different from the outside, or else} \\ & \text{the boundary did not involve the new circle, in which case} \\ & \text{the regions are colored differently because they were colored} \\ & \text{differently before the new circle was restored. } 41. \text{ If } n = 1 \\ & \text{then the equation reads } 1 \cdot 1 = 1 \cdot 2/2, \text{ which is true. Assume} \\ & \text{that the equation is true for } n \text{ and consider it for } n + 1. \text{ Then} \\ & \sum_{j=1}^{n+1} (2j - 1) \left(\sum_{k=j}^{n+1} \frac{1}{k} \right) = \sum_{j=1}^n (2j - 1) \left(\sum_{k=j}^{n+1} \frac{1}{k} \right) + \\ & [2(n+1) - 1] \cdot \frac{1}{n+1} = \sum_{j=1}^n (2j - 1) \left(\frac{1}{n+1} + \sum_{k=j}^n \frac{1}{k} \right) + \\ & \frac{2n+1}{n+1} = \left(\frac{1}{n+1} \sum_{j=1}^n (2j - 1) \right) + \left(\sum_{j=1}^n (2j - 1) \right) \\ & \sum_{k=j}^n \frac{1}{k} + \frac{2n+1}{n+1} = \left(\frac{1}{n+1} \cdot n^2 \right) + \frac{n(n+1)}{2} + \frac{2n+1}{n+1} \\ & (\text{by the inductive hypothesis}) = \frac{2n^2+n(n+1)^2+(4n+2)}{2(n+1)} = \\ & \frac{2(n+1)^2+n(n+1)^2}{2(n+1)} = \frac{(n+1)(n+2)}{2}. \end{aligned}$$

43. Let $T(n)$ be the statement that the sequence of towers of 2 is eventually constant modulo n . We use strong induction to prove that $T(n)$ is true for all positive integers n . *Basis step:* When $n = 1$ (and $n = 2$), the sequence of towers of 2 modulo n is the sequence of all 0s. *Inductive step:* Suppose that k is an integer with $k \geq 2$. Suppose that $T(j)$ is true for $1 \leq j \leq k - 1$. In the proof of the inductive step we denote the r th term of the sequence modulo n by a_r . First suppose k is even. Let $k = 2^s q$ where $s \geq 1$ and $q < k$ is odd. When j is large enough, $a_{j-2} \geq s$, and for such j , $a_j = 2^{2^{a_{j-2}}}$ is a multiple of 2^s . It follows that for sufficiently large j , $a_j \equiv 0 \pmod{2^s}$. Hence, for large enough i , 2^s divides $a_{i+1} - a_i$. By the inductive hypothesis $T(q)$ is true, so the sequence a_1, a_2, a_3, \dots is eventually constant modulo q . This implies that for large enough i , q divides $a_{i+1} - a_i$. Because $\gcd(q, 2^s) = 1$ and for sufficiently large i both q and 2^s divide $a_{i+1} - a_i$, $k = 2^s q$ divides $a_{i+1} - a_i$ for sufficiently large i . Hence, for sufficiently large i , $a_{i+1} - a_i \equiv 0 \pmod{k}$. This means that the sequence is eventually constant modulo k . Finally, suppose k is odd. Then $\gcd(2, k) = 1$, so by Euler's theorem (found in elementary number theory books, such as [Ro10]), we know that $2^{\phi(k)} \equiv 1 \pmod{k}$. Let $r = \phi(k)$. Because $r < k$, by the inductive hypothesis $T(r)$, the sequence a_1, a_2, a_3, \dots is eventually constant modulo r , say equal to c . Hence for large enough i , for some integer t_i , $a_i = t_i r + c$. Hence $a_{i+1} = 2^{a_i} = 2^{t_i r + c} = (2^r)^{t_i} 2^c \equiv 2^c \pmod{k}$. This shows that a_1, a_2, \dots is eventually constant modulo k .

45. a) 92 b) 91 c) 91 d) 91 e) 91 f) 91 **47.** The basis step is incorrect because $n \neq 1$ for the sum shown.

49. Let $P(n)$ be “the plane is divided into $n^2 - n + 2$ regions by n circles if every two of these circles have two common points but no three have a common point.” *Basis step:* $P(1)$ is true because a circle divides the plane into $2 = 1^2 - 1 + 2$ regions. *Inductive step:* Assume that $P(k)$ is true, that is, k circles with the specified properties divide the plane into $k^2 - k + 2$ regions. Suppose that a $(k + 1)$ st circle is added. This circle intersects each of the other k circles in two points, so these points of intersection form $2k$ new arcs, each of which splits an old region. Hence, there are $2k$ regions split, which shows that there are $2k$ more regions than there were previously. Hence, $k + 1$ circles satisfying the specified prop-

erties divide the plane into $k^2 - k + 2 + 2k = (k^2 + 2k + 1) - (k + 1) + 2 = (k + 1)^2 - (k + 1) + 2$ regions. **51.** Suppose $\sqrt{2}$ were rational. Then $\sqrt{2} = a/b$, where a and b are positive integers. It follows that the set $S = \{n\sqrt{2} \mid n \in \mathbb{N}\} \cap \mathbb{N}$ is a nonempty set of positive integers, because $b\sqrt{2} = a$ belongs to S . Let t be the least element of S , which exists by the well-ordering property. Then $t = s\sqrt{2}$ for some integer s . We have $t - s = s\sqrt{2} - s = s(\sqrt{2} - 1)$, so $t - s$ is a positive integer because $\sqrt{2} > 1$. Hence, $t - s$ belongs to S . This is a contradiction because $t - s = s\sqrt{2} - s < s$. Hence, $\sqrt{2}$ is irrational. **53. a)** Let $d = \gcd(a_1, a_2, \dots, a_n)$. Then d is a divisor of each a_i and so must be a divisor of $\gcd(a_{n-1}, a_n)$. Hence, d is a common divisor of a_1, a_2, \dots, a_{n-2} , and $\gcd(a_{n-1}, a_n)$. To show that it is the greatest common divisor of these numbers, suppose that c is a common divisor of them. Then c is a divisor of a_i for $i = 1, 2, \dots, n-2$ and a divisor of $\gcd(a_{n-1}, a_n)$, so it is a divisor of a_{n-1} and a_n . Hence, c is a common divisor of a_1, a_2, \dots, a_{n-1} , and a_n . Hence, it is a divisor of d , the greatest common divisor of a_1, a_2, \dots, a_n . It follows that d is the greatest common divisor, as claimed. **b)** If $n = 2$, apply the Euclidean algorithm. Otherwise, apply the Euclidean algorithm to a_{n-1} and a_n , obtaining $d = \gcd(a_{n-1}, a_n)$, and then apply the algorithm recursively to $a_1, a_2, \dots, a_{n-2}, d$. **55.** $f(n) = n^2$. Let $P(n)$ be “ $f(n) = n^2$.” *Basis step:* $P(1)$ is true because $f(1) = 1 = 1^2$, which follows from the definition of f . *Inductive step:* Assume $f(n) = n^2$. Then $f(n+1) = f((n+1)-1) + 2(n+1)-1 = f(n) + 2n+1 = n^2 + 2n+1 = (n+1)^2$. **57. a)** $\lambda, 0, 1, 00, 01, 11, 000, 001, 011, 111, 0000, 0001, 0011, 0111, 1111, 00000, 00001, 00011, 00111, 01111, 11111$ **b)** $S = \{\alpha\beta \mid \alpha$ is a string of m 0s and β is a string of n 1s, $m \geq 0, n \geq 0\}$ **59.** Apply the first recursive step to λ to get $() \in B$. Apply the second recursive step to this string to get $(()) \in B$. Apply the first recursive step to this string to get $((()) \in B$. By Exercise 62, $((())$ is not in B because the number of left parentheses does not equal the number of right parentheses. **61.** $\lambda, (), (), ()()$ **63. a)** 0 **b)** -2 **c)** 2 **d)** 0

65.

```

procedure generate(n: nonnegative integer)
if n is odd then
  S := S(n - 1) {the S constructed by generate(n - 1)}
  T := T(n - 1) {the T constructed by generate(n - 1)}
else if n = 0 then
  S := ∅
  T := {λ}
else
  S' := S(n - 2) {the S constructed by generate(n - 2)}
  T' := T(n - 2) {the T constructed by generate(n - 2)}
  T := T' ∪ {(x) | x ∈ T' ∪ S' ∧ length(x) = n - 2}
  S := S' ∪ {xy | x ∈ T' ∧ y ∈ T' ∪ S' ∧ length(xy) = n}
  {T ∪ S is the set of balanced strings of length at most n}
67. If  $x \leq y$  initially, then  $x := y$  is not executed, so  $x \leq y$  is a true final assertion. If  $x > y$  initially, then  $x := y$  is executed, so  $x \leq y$  is again a true final assertion.
69. procedure zerocount( $a_1, a_2, \dots, a_n$ : list of integers)
  if  $n = 1$  then

```

```

if  $a_1 = 0$  then return 1
else return 0
else
  if  $a_n = 0$  then return zerocount( $a_1, a_2, \dots, a_{n-1}$ ) + 1
  else return zerocount( $a_1, a_2, \dots, a_{n-1}$ )

```

71. We will prove that $a(n)$ is a natural number and $a(n) \leq n$. This is true for the base case $n = 0$ because $a(0) = 0$. Now assume that $a(n-1)$ is a natural number and $a(n-1) \leq n-1$. Then $a(a(n-1))$ is a applied to a natural number less than or equal to $n-1$. Hence, $a(a(n-1))$ is also a natural number minus than or equal to $n-1$. Therefore, $n - a(a(n-1))$ is n minus some natural number less than or equal to $n-1$, which is a natural number less than or equal to n . **73.** From Exercise 72, $a(n) = \lfloor (n+1)\mu \rfloor$ and $a(n-1) = \lfloor n\mu \rfloor$. Because $\mu < 1$, these two values are equal or they differ by 1. First suppose that $\mu n - \lfloor \mu n \rfloor < 1 - \mu$. This is equivalent to $\mu(n+1) < 1 + \lfloor \mu n \rfloor$. If this is true, then $\lfloor \mu(n+1) \rfloor = \lfloor \mu n \rfloor$. On the other hand, if $\mu n - \lfloor \mu n \rfloor \geq 1 - \mu$, then $\mu(n+1) \geq 1 + \lfloor \mu n \rfloor$, so $\lfloor \mu(n+1) \rfloor = \lfloor \mu n \rfloor + 1$, as desired. **75.** $f(0) = 1, m(0) = 0; f(1) = 1, m(1) = 0; f(2) = 2, m(2) = 1; f(3) = 2, m(3) = 2; f(4) = 3, m(4) = 2; f(5) = 3, m(5) = 3; f(6) = 4, m(6) = 4; f(7) = 5, m(7) = 4; f(8) = 5, m(8) = 5; f(9) = 6, m(9) = 6$ **77.** The last occurrence of n is in the position for which the total number of 1s, 2s, ..., n s all together is that position number. But because a_k is the number of occurrences of k , this is just $\sum_{k=1}^n a_k$, as desired. Because $f(n)$ is the sum of the first n terms of the sequence, $f(f(n))$ is the sum of the first $f(n)$ terms of the sequence. But because $f(n)$ is the last term whose value is n , this means that the sum is the sum of all terms of the sequence whose value is at most n . Because there are a_k terms of the sequence whose value is k , this sum is $\sum_{k=1}^n k \cdot a_k$, as desired

CHAPTER 6

Section 6.1

- 1. a)** 5850 **b)** 343 **3. a)** 4^{10} **b)** 5^{10} **5. 42** **7. 26³**
- 9. 676** **11. 2⁸** **13. $n + 1$** (counting the empty string)
- 15. 475,255** (counting the empty string) **17. 1,321,368,961**
- 19. a)** 729 **b)** 256 **c)** 1024 **d)** 64 **21. a)** Seven: 56, 63, 70, 77, 84, 91, 98 **b)** Five: 55, 66, 77, 88, 99
- c)** One: 77 **23. a)** 128 **b)** 450 **c)** 9 **d)** 675 **e)** 450
- f)** 450 **g)** 225 **h)** 75 **25. a)** 990 **b)** 500 **c)** 27 **27. 3⁵⁰**
- 29. 52,457,600** **31. 20,077,200** **33. a)** 37,822,859,361
- b)** 8,204,716,800 **c)** 40,159,050, 880 **d)** 12,113,640,000
- e)** 171,004,205,215 **f)** 72,043,541,640 **g)** 6,230,721,635
- h)** 223,149,655 **35. a)** 0 **b)** 120 **c)** 720 **d)** 2520 **37. a)** 2 if $n = 1, 2$ if $n = 2, 0$ if $n \geq 3$ **b)** 2^{n-2} for $n > 1$; 1 if $n = 1$ **c)** $2(n-1)$ **39. $(n+1)^m$** **41.** If n is even, $2^{n/2}$; if n is odd, $2^{(n+1)/2}$ **43. a)** 175 **b)** 248 **c)** 232 **d)** 84
- 45. 60** **47. a)** 240 **b)** 480 **c)** 360 **49. 352** **51. 147**
- 53. 33** **55. a)** $9,920,671,339,261,325,541,376 \approx 9.9 \times 10^{21}$ **b)** $6,641,514,961,387,068,437,760 \approx 6.6 \times 10^{21}$ **c)** About 314,000 years **57. $54(64^{65536} - 1)/63$**

- 59.** 7,104,000,000,000 **61.** $16^{10} + 16^{26} + 16^{58}$
63. 666,667 **65.** 18 **67.** 17 **69.** 22 **71.** Let $P(m)$ be the sum rule for m tasks. For the basis case take $m = 2$. This is just the sum rule for two tasks. Now assume that $P(m)$ is true. Consider $m + 1$ tasks, $T_1, T_2, \dots, T_m, T_{m+1}$, which can be done in $n_1, n_2, \dots, n_m, n_{m+1}$ ways, respectively, such that no two of these tasks can be done at the same time. To do one of these tasks, we can either do one of the first m of these or do task T_{m+1} . By the sum rule for two tasks, the number of ways to do this is the sum of the number of ways to do one of the first m tasks, plus n_{m+1} . By the inductive hypothesis, this is $n_1 + n_2 + \dots + n_m + n_{m+1}$, as desired. **73.** $n(n - 3)/2$

Section 6.2

- 1.** Because there are six classes, but only five weekdays, the pigeonhole principle shows that at least two classes must be held on the same day. **3. a)** 3 **b)** 14 **5.** Because there are four possible remainders when an integer is divided by 4, the pigeonhole principle implies that given five integers, at least two have the same remainder. **7.** Let $a, a + 1, \dots, a + n - 1$ be the integers in the sequence. The integers $(a + i) \bmod n, i = 0, 1, 2, \dots, n - 1$, are distinct, because $0 < (a + j) - (a + k) < n$ whenever $0 \leq k < j \leq n - 1$. Because there are n possible values for $(a + i) \bmod n$ and there are n different integers in the set, each of these values is taken on exactly once. It follows that there is exactly one integer in the sequence that is divisible by n . **9.** 4951 **11.** The midpoint of the segment joining the points (a, b, c) and (d, e, f) is $((a+d)/2, (b+e)/2, (c+f)/2)$. It has integer coefficients if and only if a and d have the same parity, b and e have the same parity, and c and f have the same parity. Because there are eight possible triples of parity [such as (even, odd, even)], by the pigeonhole principle at least two of the nine points have the same triple of parities. The midpoint of the segment joining two such points has integer coefficients. **13. a)** Group the first eight positive integers into four subsets of two integers each so that the integers of each subset add up to 9: $\{1, 8\}$, $\{2, 7\}$, $\{3, 6\}$, and $\{4, 5\}$. If five integers are selected from the first eight positive integers, by the pigeonhole principle at least two of them come from the same subset. Two such integers have a sum of 9, as desired. **b)** No. Take $\{1, 2, 3, 4\}$, for example. **15.** 4 **17.** 21,251 **19. a)** If there were fewer than 9 freshmen, fewer than 9 sophomores, and fewer than 9 juniors in the class, there would be no more than 8 with each of these three class standings, for a total of at most 24 students, contradicting the fact that there are 25 students in the class. **b)** If there were fewer than 3 freshmen, fewer than 19 sophomores, and fewer than 5 juniors, then there would be at most 2 freshmen, at most 18 sophomores, and at most 4 juniors, for a total of at most 24 students. This contradicts the fact that there are 25 students in the class. **21.** 4, 3, 2, 1, 8, 7, 6, 5, 12, 11, 10, 9, 16, 15, 14, 13 **23.** Number the seats around the table from 1 to 50, and think of seat 50 as being adjacent to seat 1. There are 25 seats with odd numbers and 25 seats with even numbers. If no more than 12 boys occupied the odd-numbered

seats, then at least 13 boys would occupy the even-numbered seats, and vice versa. Without loss of generality, assume that at least 13 boys occupy the 25 odd-numbered seats. Then at least two of those boys must be in consecutive odd-numbered seats, and the person sitting between them will have boys as both of his or her neighbors.

25. procedure *long*(a_1, \dots, a_n : positive integers)

{first find longest increasing subsequence}

max := 0; *set* := 00...00 {*n* bits}

for *i* := 1 **to** 2^n

last := 0; *count* := 0, *OK* := **true**

for *j* := 1 **to** *n*

if *set(j)* = 1 **then**

if *a_j* > *last* **then** *last* := *a_j*

count := *count* + 1

else *OK* := **false**

if *count* > *max* **then**

max := *count*

best := *set*

set := *set* + 1 (binary addition)

{*max* is length and *best* indicates the sequence}

{repeat for decreasing subsequence with only

 changes being *a_j* < *last* instead of *a_j* > *last*

 and *last* := ∞ instead of *last* := 0}

- 27.** By symmetry we need prove only the first statement. Let *A* be one of the people. Either *A* has at least four friends, or *A* has at least six enemies among the other nine people (because $3 + 5 < 9$). Suppose, in the first case, that *B*, *C*, *D*, and *E* are all *A*'s friends. If any two of these are friends with each other, then we have found three mutual friends. Otherwise $\{B, C, D, E\}$ is a set of four mutual enemies. In the second case, let $\{B, C, D, E, F, G\}$ be a set of enemies of *A*. By Example 11, among *B*, *C*, *D*, *E*, *F*, and *G* there are either three mutual friends or three mutual enemies, who form, with *A*, a set of four mutual enemies. **29.** We need to show two things: that if we have a group of n people, then among them we must find either a pair of friends or a subset of n of them all of whom are mutual enemies; and that there exists a group of $n - 1$ people for which this is not possible. For the first statement, if there is any pair of friends, then the condition is satisfied, and if not, then every pair of people are enemies, so the second condition is satisfied. For the second statement, if we have a group of $n - 1$ people all of whom are enemies of each other, then there is neither a pair of friends nor a subset of n of them all of whom are mutual enemies. **31.** There are 6,432,816 possibilities for the three initials and a birthday. So, by the generalized pigeonhole principle, there are at least $\lceil 37,000,000/6,432,816 \rceil = 6$ people who share the same initials and birthday. **33.** Because $800,001 > 200,000$, the pigeonhole principle guarantees that there are at least two Parisians with the same number of hairs on their heads. The generalized pigeonhole principle guarantees that there are at least $\lceil 800,001/200,000 \rceil = 5$ Parisians with the same number of hairs on their heads. **35.** 18 **37.** Because there are six computers, the number of other computers a computer is connected to is an integer between 0 and 5, inclusive. However, 0 and 5 cannot both occur. To see this, note that if some

computer is connected to no others, then no computer is connected to all five others, and if some computer is connected to all five others, then no computer is connected to no others. Hence, by the pigeonhole principle, because there are at most five possibilities for the number of computers a computer is connected to, there are at least two computers in the set of six connected to the same number of others. **39.** Label the computers C_1 through C_{100} , and label the printers P_1 through P_{20} . If we connect C_k to P_k for $k = 1, 2, \dots, 20$ and connect each of the computers C_{21} through C_{100} to *all* the printers, then we have used a total of $20 + 80 \cdot 20 = 1620$ cables. Clearly this is sufficient, because if computers C_1 through C_{20} need printers, then they can use the printers with the same subscripts, and if any computers with higher subscripts need a printer instead of one or more of these, then they can use the printers that are not being used, because they are connected to all the printers. Now we must show that 1619 cables is not enough. Because there are 1619 cables and 20 printers, the average number of computers per printer is $1619/20$, which is less than 81. Therefore some printer must be connected to fewer than 81 computers. That means it is connected to 80 or fewer computers, so there are 20 computers that are not connected to it. If those 20 computers all needed a printer simultaneously, then they would be out of luck, because they are connected to at most the 19 other printers. **41.** Let a_i be the number of matches completed by hour i . Then $1 \leq a_1 < a_2 < \dots < a_{75} \leq 125$. Also $25 \leq a_1 + 24 < a_2 + 24 < \dots < a_{75} + 24 \leq 149$. There are 150 numbers $a_1, \dots, a_{75}, a_1 + 24, \dots, a_{75} + 24$. By the pigeonhole principle, at least two are equal. Because all the a_i 's are distinct and all the $(a_i + 24)$'s are distinct, it follows that $a_i = a_j + 24$ for some $i > j$. Thus, in the period from the $(j+1)$ st to the i th hour, there are exactly 24 matches. **43.** Use the generalized pigeonhole principle, placing the $|S|$ objects $f(s)$ for $s \in S$ in $|T|$ boxes, one for each element of T . **45.** Let d_j be $jx - N(jx)$, where $N(jx)$ is the integer closest to jx for $1 \leq j \leq n$. Each d_j is an irrational number between $-1/2$ and $1/2$. We will assume that n is even; the case where n is odd is messier. Consider the n intervals $\{x \mid j/n < x < (j+1)/n\}, \{x \mid -(j+1)/n < x < -j/n\}$ for $j = 0, 1, \dots, (n/2) - 1$. If d_j belongs to the interval $\{x \mid 0 < x < 1/n\}$ or to the interval $\{x \mid -1/n < x < 0\}$ for some j , we are done. If not, because there are $n-2$ intervals and n numbers d_j , the pigeonhole principle tells us that there is an interval $\{x \mid (k-1)/n < x < k/n\}$ containing d_r and d_s with $r < s$. The proof can be finished by showing that $(s-r)x$ is within $1/n$ of its nearest integer. **47. a)** Assume that $i_k \leq n$ for all k . Then by the generalized pigeonhole principle, at least $\lceil (n^2 + 1)/n \rceil = n + 1$ of the numbers $i_1, i_2, \dots, i_{n^2+1}$ are equal. **b)** If $a_{k_j} < a_{k_{j+1}}$, then the subsequence consisting of a_{k_j} followed by the increasing subsequence of length $i_{k_{j+1}}$ starting at $a_{k_{j+1}}$ contradicts the fact that $i_{k_j} = i_{k_{j+1}}$. Hence, $a_{k_j} > a_{k_{j+1}}$. **c)** If there is no increasing subsequence of length greater than n , then parts (a) and (b) apply. Therefore, we have $a_{k_{n+1}} > a_{k_n} > \dots > a_{k_2} > a_{k_1}$, a decreasing sequence of length $n + 1$.

Section 6.3

- 1.** $abc, acb, bac, bca, cab, cba$ **3.** 720 **5. a)** 120
b) 720 **c)** 8 **d)** 6720 **e)** 40,320 **f)** 3,628,800 **7.** 15,120
9. 1320 **11. a)** 210 **b)** 386 **c)** 848 **d)** 252 **13.** $2(n!)^2$
15. 65,780 **17.** $2^{100} - 5051$ **19. a)** 1024 **b)** 45
c) 176 **d)** 252 **21. a)** 120 **b)** 24 **c)** 120 **d)** 24
e) 6 **f)** 0 **23.** 609,638,400 **25. a)** 94,109,400 **b)** 941,094
c) 3,764,376 **d)** 90,345,024 **e)** 114,072 **f)** 2328 **g)** 24
h) 79,727,040 **i)** 3,764,376 **j)** 109,440 **27. a)** 12,650
b) 303,600 **29. a)** 37,927 **b)** 18,915 **31. a)** 122,523,030
b) 72,930,375 **c)** 223,149,655 **d)** 100,626,625 **33.** 54,600
35. 45 **37.** 912 **39.** 11,232,000 **41.** $n!/(r(n-r)!)$
43. 13 **45.** 873

Section 6.4

- 1.** $x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$ **3.** $x^6 + 6x^5y + 15x^4y^2 + 20x^3y^3 + 15x^2y^4 + 6xy^5 + y^6$
5. 101 **7.** $-2^{10} \binom{19}{9} = -94,595,072$ **9.** $-2^{101} 3^{99} \binom{200}{99}$
11. $(-1)^{(200-k)/3} \binom{100}{(200-k)/3}$ if $k \equiv 2 \pmod{3}$ and $-100 \leq k \leq 200$; 0 otherwise **13.** 1 9 36 84 126 126 84
36 9 1 **15.** The sum of *all* the positive numbers $\binom{n}{k}$, as k runs from 0 to n , is 2^n , so each one of them is no bigger than this sum. **17.** $\binom{n}{k} = \frac{n(n-1)(n-2)\dots(n-k+1)}{k(k-1)(k-2)\dots2} \leq \frac{n \cdot n \cdots n}{2 \cdot 2 \cdots 2} = n^k / 2^{k-1}$ **19.** $\binom{n}{k-1} + \binom{n}{k} = \frac{n!}{(k-1)!(n-k+1)!} + \frac{n!}{k!(n-k)!} = \frac{n!}{k!(n-k+1)!} \cdot [k + (n - k + 1)] = \frac{(n+1)!}{k!(n+1-k)!} = \binom{n+1}{k}$
21. a) We show that each side counts the number of ways to choose from a set with n elements a subset with k elements and a distinguished element of that set. For the left-hand side, first choose the k -set (this can be done in $\binom{n}{k}$ ways) and then choose one of the k elements in this subset to be the distinguished element (this can be done in k ways). For the right-hand side, first choose the distinguished element out of the entire n -set (this can be done in n ways), and then choose the remaining $k-1$ elements of the subset from the remaining $n-1$ elements of the set (this can be done in $\binom{n-1}{k-1}$ ways). **b)** $k \binom{n}{k} = k \cdot \frac{n!}{k!(n-k)!} = \frac{n \cdot (n-1)!}{(k-1)!(n-k)!} = n \binom{n-1}{k-1}$
23. $\binom{n+1}{k} = \frac{(n+1)!}{k!(n+1-k)!} = \frac{(n+1)}{k} \frac{n!}{(k-1)!(n-(k-1))!} = (n+1) \binom{n}{k-1}/k$. This identity together with $\binom{n}{0} = 1$ gives a recursive definition. **25.** $\binom{2n}{n+1} + \binom{2n}{n} = \binom{2n+1}{n+1} = \frac{1}{2} \left[\binom{2n+1}{n+1} + \binom{2n+1}{n+1} \right] = \frac{1}{2} \left[\binom{2n+1}{n+1} + \binom{2n+1}{n} \right] = \frac{1}{2} \left[\binom{2n+2}{n+1} \right] = \frac{1}{2} \binom{2n+2}{n+1}$ **27. a)** $\binom{n+r+1}{r}$ counts the number of ways to choose a sequence of r 0s and $n+1$ 1s by choosing the positions of the 0s. Alternately, suppose that the $(j+1)$ st term is the last term equal to 1, so that $n \leq j \leq n+r$. Once we have determined where the last 1 is, we decide where the 0s are to be placed in the j spaces before the last 1. There are n 1s and $j-n$ 0s in this range. By the sum rule it follows that there are $\sum_{j=n}^{n+r} \binom{j}{j-n} = \sum_{k=0}^r \binom{n+k}{k}$ ways to do this. **b)** Let $P(r)$ be the statement to be proved. The basis step is the equation $\binom{n}{0} = \binom{n+1}{0}$, which is just $1 = 1$. Assume that $P(r)$ is true. Then $\sum_{k=0}^{r+1} \binom{n+k}{k} = \sum_{k=0}^r \binom{n+k}{k} + \binom{n+r+1}{r+1} = \binom{n+r+1}{r} + \binom{n+r+1}{r+1} = \binom{n+r+2}{r+1}$, using the inductive hypothesis

and Pascal's identity. **29.** We can choose the leader first in n different ways. We can then choose the rest of the committee in 2^{n-1} ways. Hence, there are $n2^{n-1}$ ways to choose the committee and its leader. Meanwhile, the number of ways to select a committee with k people is $\binom{n}{k}$. Once we have chosen a committee with k people, there are k ways to choose its leader. Hence, there are $\sum_{k=1}^n k \binom{n}{k}$ ways to choose the committee and its leader. Hence, $\sum_{k=1}^n k \binom{n}{k} = n2^{n-1}$.

31. Let the set have n elements. From Corollary 2 we have $\binom{n}{0} - \binom{n}{1} + \binom{n}{2} - \cdots + (-1)^n \binom{n}{n} = 0$. It follows that $\binom{n}{0} + \binom{n}{2} + \binom{n}{4} + \cdots = \binom{n}{1} + \binom{n}{3} + \binom{n}{5} + \cdots$. The left-hand side gives the number of subsets with an even number of elements, and the right-hand side gives the number of subsets with an odd number of elements. **33. a)** A path of the desired type consists of m moves to the right and n moves up. Each such path can be represented by a bit string of length $m+n$ with m 0s and n 1s, where a 0 represents a move to the right and a 1 a move up. **b)** The number of bit strings of length $m+n$ containing exactly n 1s equals $\binom{m+n}{n} = \binom{m+n}{m}$ because such a string is determined by specifying the positions of the n 1s or by specifying the positions of the m 0s. **35.** By Exercise 33 the number of paths of length n of the type described in that exercise equals 2^n , the number of bit strings of length n . On the other hand, a path of length n of the type described in Exercise 33 must end at a point that has n as the sum of its coordinates, say $(n-k, k)$ for some k between 0 and n , inclusive. By Exercise 33, the number of such paths ending at $(n-k, k)$ equals $\binom{n-k+k}{k} = \binom{n}{k}$. Hence, $\sum_{k=0}^n \binom{n}{k} = 2^n$. **37.** By Exercise 33 the number of paths from $(0, 0)$ to $(n+1, r)$ of the type described in that exercise equals $\binom{n+r+1}{r}$. But such a path starts by going j steps vertically for some j with $0 \leq j \leq r$. The number of these paths beginning with j vertical steps equals the number of paths of the type described in Exercise 33 that go from $(1, j)$ to $(n+1, r)$. This is the same as the number of such paths that go from $(0, 0)$ to $(n, r-j)$, which by Exercise 33 equals $\binom{n+r-j}{r-j}$. Because $\sum_{j=0}^r \binom{n+r-j}{r-j} = \sum_{k=0}^r \binom{n+k}{k}$, it follows that $\sum_{k=1}^r \binom{n+k}{k} = \binom{n+r-1}{r}$. **39. a)** $\binom{n+1}{2}$ **b)** $\binom{n+2}{3}$ **c)** $\binom{2n-2}{n-1}$ **d)** $\binom{n-1}{\lfloor (n-1)/2 \rfloor}$ **e)** Largest odd entry in n th row of Pascal's triangle **f)** $\binom{3n-3}{n-1}$

Section 6.5

1. 243 **3.** 26^6 **5.** 125 **7.** 35 **9. a)** 1716 **b)** 50,388
- c) 2,629,575 **d)** 330 **11. 9** **13.** 4,504,501 **15. a)** 10,626
- b) 1,365 **c)** 11,649 **d)** 106 **17.** 2,520 **19.** 302,702,400
- 21.** 3003 **23.** 7,484,400 **25.** 30,492 **27.** $C(59, 50)$
- 29.** 35 **31.** 83,160 **33. 63** **35.** 19,635 **37.** 210
- 39.** 27,720 **41.** $52!/(7!^5 17!)$ **43.** Approximately 6.5×10^{32}
- 45. a)** $C(k+n-1, n)$ **b)** $(k+n-1)!/(k-1)!$ **47.** There are $C(n, n_1)$ ways to choose n_1 objects for the first box. Once these objects are chosen, there are $C(n-n_1, n_2)$ ways to choose objects for the second box. Similarly, there are $C(n-n_1-n_2, n_3)$ ways to choose objects for the third box. Continue in this way until there is

$C(n-n_1-n_2-\cdots-n_{k-1}, n_k) = C(n_k, n_k) = 1$ way to choose the objects for the last box (because $n_1+n_2+\cdots+n_k=n$). By the product rule, the number of ways to make the entire assignment is $C(n, n_1)C(n-n_1, n_2)C(n-n_1-n_2, n_3)\cdots C(n-n_1-n_2-\cdots-n_{k-1}, n_k)$, which equals $n!/(n_1!n_2!\cdots n_k!)$, as straightforward simplification shows. **49. a)** Because $x_1 \leq x_2 \leq \cdots \leq x_r$, it follows that $x_1 + 0 < x_2 + 1 < \cdots < x_r + r - 1$. The inequalities are strict because $x_j + j - 1 < x_{j+1} + j$ as long as $x_j \leq x_{j+1}$. Because $1 \leq x_j \leq n+r-1$, this sequence is made up of r distinct elements from T . **b)** Suppose that $1 \leq x_1 < x_2 < \cdots < x_r \leq n+r-1$. Let $y_k = x_k - (k-1)$. Then it is not hard to see that $y_k \leq y_{k+1}$ for $k = 1, 2, \dots, r-1$ and that $1 \leq y_k \leq n$ for $k = 1, 2, \dots, r$. It follows that $\{y_1, y_2, \dots, y_r\}$ is an r -combination with repetitions allowed of S . **c)** From parts (a) and (b) it follows that there is a one-to-one correspondence of r -combinations with repetitions allowed of S and r -combinations of T , a set with $n+r-1$ elements. We conclude that there are $C(n+r-1, r)$ r -combinations with repetitions allowed of S . **51.** 65 **53.** 65 **55. 2** **57. 3** **59. a)** 150 **b)** 25 **c)** 6 **d)** 2 **61.** 90,720 **63.** The terms in the expansion are of the form $x_1^{n_1} x_2^{n_2} \cdots x_m^{n_m}$, where $n_1+n_2+\cdots+n_m=n$. Such a term arises from choosing the x_1 in n_1 factors, the x_2 in n_2 factors, ..., and the x_m in n_m factors. This can be done in $C(n; n_1, n_2, \dots, n_m)$ ways, because a choice is a permutation of n_1 labels "1," n_2 labels "2," ..., and n_m labels "m." **65.** 2520

Section 6.6

1. 14532, 15432, 21345, 23451, 23514, 31452, 31542, 43521, 45213, 45321 **3.** AAA1, AAA2, AAB1, AAB2, AAC1, AAC2, ABA1, ABA2, ABB1, ABB2, ABC1, ABC2, ACA1, ACA2, ACB1, ACB2, ACC1, ACC2, BAA1, BAA2, BAB1, BAB2, BAC1, BAC2, BBA1, BBA2, BBB1, BBB2, BBC1, BBC2, BCA1, BCA2, BCB1, BCB2, BCC1, BCC2, CAA1, CAA2, CAB1, CAB2, CAC1, CAC2, CBA1, CBA2, CBB1, CBB2, CBC1, CBC2, CCA1, CCA2, CCB1, CCB2, CCC1, CCC2 **5. a)** 2134 **b)** 54132 **c)** 12534 **d)** 45312 **e)** 7,1234, 1243, 1324, 1342, 1423, 1432, 2134, 2143, 2314, 2341, 2413, 2431, 3124, 3142, 3214, 3241, 3412, 3421, 4123, 4132, 4213, 4231, 4312, 4321 **9.** {1, 2, 3}, {1, 2, 4}, {1, 2, 5}, {1, 3, 4}, {1, 3, 5}, {1, 4, 5}, {2, 3, 4}, {2, 3, 5}, {2, 4, 5}, {3, 4, 5} **11.** The bit string representing the next larger r -combination must differ from the bit string representing the original one in position i because positions $i+1, \dots, r$ are occupied by the largest possible numbers. Also $a_i + 1$ is the smallest possible number we can put in position i if we want a combination greater than the original one. Then $a_i + 2, \dots, a_i + r - i + 1$ are the smallest allowable numbers for positions $i+1$ to r . Thus, we have produced the next r -combination. **13.** 123, 132, 213, 231, 312, 321, 124, 142, 214, 241, 412, 421, 125, 152, 215, 251, 512, 521, 134, 143, 314, 341, 413, 431, 135, 153, 315, 351, 513, 531, 145, 154, 415, 451, 514, 541, 234, 243, 324, 342, 423, 432,

235, 253, 325, 352, 523, 532, 245, 254, 425, 452, 524, 542, 345, 354, 435, 453, 534, 543 **15.** We will show that it is a bijection by showing that it has an inverse. Given a positive integer less than $n!$, let a_1, a_2, \dots, a_{n-1} be its Cantor digits. Put n in position $n - a_{n-1}$; then clearly, a_{n-1} is the number of integers less than n that follow n in the permutation. Then put $n - 1$ in free position $(n - 1) - a_{n-2}$, where we have numbered the free positions $1, 2, \dots, n - 1$ (excluding the position that n is already in). Continue until 1 is placed in the only free position left. Because we have constructed an inverse, the correspondence is a bijection.

17. procedure *Cantor permutation(n, i)*: integers with

```

 $n \geq 1$  and  $0 \leq i < n!$ 
x := n
for j := 1 to n
  p_j := 0
for k := 1 to n - 1
  c := ⌊x/(n - k)!⌋; x := x - c(n - k)!; h := n
  while p_h ≠ 0
    h := h - 1
  for j := 1 to c
    h := h - 1
    while p_h ≠ 0
      h := h - 1
    p_h := n - k + 1
  h := 1
  while p_h ≠ 0
    h := h + 1
  p_h := 1
{p_1 p_2 ... p_n} is the permutation corresponding
  to i
}
```

Supplementary Exercises

- 1. a)** 151,200 **b)** 1,000,000 **c)** 210 **d)** 5005 **3.** 3^{100}
5. 24,600 **7. a)** 4060 **b)** 2688 **c)** 25,009,600 **9. a)** 192

b) 301 **c)** 300 **d)** 300 **11.** 639 **13.** The maximum possible sum is 240, and the minimum possible sum is 15. So the number of possible sums is 226. Because there are 252 subsets with five elements of a set with 10 elements, by the pigeonhole principle it follows that at least two have the same sum. **15. a)** 50 **b)** 50 **c)** 14 **d)** 17 **17.** Let a_1, a_2, \dots, a_m be the integers, and let $d_i = \sum_{j=1}^i a_j$. If $d_i \equiv 0 \pmod{m}$ for some i , we are done. Otherwise $d_1 \pmod{m}, d_2 \pmod{m}, \dots, d_m \pmod{m}$ are m integers with values in $\{1, 2, \dots, m - 1\}$. By the pigeonhole principle $d_k \equiv d_l \pmod{m}$ for some $1 \leq k < l \leq m$. Then $\sum_{j=k+1}^l a_j \equiv d_l - d_k \equiv 0 \pmod{m}$. **19.** The decimal expansion of the rational number a/b can be obtained by division of b into a , where a is written with a decimal point and an arbitrarily long string of 0s following it. The basic step is finding the next digit of the quotient, namely, $\lfloor r/b \rfloor$, where r is the remainder with the next digit of the dividend brought down. The current remainder is obtained from the previous remainder by subtracting b times the previous digit of the quotient. Eventually the dividend has nothing but 0s to bring down. Furthermore, there are only

b possible remainders. Thus, at some point, by the pigeonhole principle, we will have the same situation as had previously arisen. From that point onward, the calculation must follow the same pattern. In particular, the quotient will repeat. **21. a)** 125,970 **b)** 20 **c)** 141,120,525 **d)** 141,120,505 **e)** 177,100 **f)** 141,078,021 **23. a)** 10 **b)** 8 **c)** 7 **25.** 3^n

$$\begin{aligned} \mathbf{27. } C(n+2, r+1) &= C(n+1, r+1) + C(n+1, r) = \\ &2C(n+1, r+1) - C(n+1, r+1) + C(n+1, r) = \\ &2C(n+1, r+1) - (C(n, r+1) + C(n, r)) + (C(n, r) + \\ &C(n, r-1)) = 2C(n+1, r+1) - C(n, r+1) + C(n, r-1) \end{aligned}$$

29. Substitute $x = 1$ and $y = 3$ into the binomial theorem.

31. Both sides count the number of ways to choose a subset of three distinct numbers $\{i, j, k\}$ with $i < j < k$ from $\{1, 2, \dots, n\}$. **33.** $C(n+1, 5)$ **35.** 3,491,888,400 **37.** 5^{24}

39. a) 45 **b)** 57 **c)** 12 **41. a)** 386 **b)** 56 **43.** 0 if $n < m$; $C(n-1, n-m)$ if $n \geq m$ **45. a)** 15,625 **b)** 202 **c)** 210

d) 10 **47. a)** 3 **b)** 11 **c)** 6 **d)** 10 **49.** There are two possibilities: three people seated at one table with everyone else sitting alone, which can be done in $2C(n, 3)$ ways (choose the three people and seat them in one of two arrangements), or two groups of two people seated together with everyone else sitting alone, which can be done in $3C(n, 4)$ ways (choose four people and then choose one of the three ways to pair them up). Both $2C(n, 3) + 3C(n, 4)$ and $(3n-1)C(n, 3)/4$ equal $n^4/8 - 5n^3/12 + 3n^2/8 - n/12$. **51.** The number of permutations of $2n$ objects of n different types, two of each type, is $(2n)!/2^n$. Because this must be an integer, the denominator must divide the numerator. **53.** CCGGUCCGAAAG

55. procedure *next permutation(n)*: positive integer,

a_1, a_2, \dots, a_r : positive integers not exceeding n with $a_1 a_2 \dots a_r \neq nn \dots n$

```

i := r
while  $a_i = n$ 
   $a_i := 1$ 
  i := i - 1
   $a_i := a_i + 1$ 
{ $a_1 a_2 \dots a_r$  is the next permutation in lexicographic
order}
```

57. We must show that if there are $R(m, n - 1) + R(m - 1, n)$ people at a party, then there must be at least m mutual friends or n mutual enemies. Consider one person; let's call him Jerry. Then there are $R(m - 1, n) + R(m, n - 1) - 1$ other people at the party, and by the pigeonhole principle there must be at least $R(m - 1, n)$ friends of Jerry or $R(m, n - 1)$ enemies of Jerry among these people. First let's suppose there are $R(m - 1, n)$ friends of Jerry. By the definition of R , among these people we are guaranteed to find either $m - 1$ mutual friends or n mutual enemies. In the former case, these $m - 1$ mutual friends together with Jerry are a set of m mutual friends; and in the latter case, we have the desired set of n mutual enemies. The other situation is similar: Suppose there are $R(m, n - 1)$ enemies of Jerry; we are guaranteed to find among them either m mutual friends or $n - 1$ mutual enemies. In the former case, we have the desired set of m mutual friends, and in the latter case, these $n - 1$ mutual enemies together with Jerry are a set of n mutual enemies.

CHAPTER 7

Section 7.1

- 1.** 1/13 **3.** 1/2 **5.** 1/2 **7.** 1/64 **9.** 47/52 **11.** 1/C(52, 5) **13.** 1 – [C(48, 5)/C(52, 5)] **15.** C(13, 2)C(4, 2)C(4, 2)C(44, 1)/C(52, 5) **17.** 10,240/C(52, 5) **19.** 1,302,540/C(52, 5) **21.** 1/64 **23.** 8/25 **25.** a) 1/ C(50, 6) = 1/15,890,700 b) 1/C(52, 6) = 1/20,358,520 c) 1/C(56, 6) = 1/32,468,436 d) 1/C(60, 6) = 1/50,063,860 **27.** a) 139,128/319,865 b) 212,667/511,313 c) 151,340/386,529 d) 163,647/446,276 **29.** 1/C(100, 8) **31.** 3/100 **33.** a) 1/7,880,400 b) 1/8,000,000 **35.** a) 9/19 b) 81/361 c) 1/19 d) 1,889,568/2,476,099 e) 48/361 **37.** Three dice **39.** The door the contestant chooses is chosen at random without knowing where the prize is, but the door chosen by the host is not chosen at random, because he always avoids opening the door with the prize. This makes any argument based on symmetry invalid. **41.** a) 671/1296 b) 1 – 35²⁴/36²⁴; no c) The former

Section 7.2

- 1.** $p(T) = 1/4$, $p(H) = 3/4$ **3.** $p(1) = p(3) = p(5) = p(6) = 1/16$; $p(2) = p(4) = 3/8$ **5.** 9/49 **7.** a) 1/2 b) 1/2 c) 1/3 d) 1/4 e) 1/4 **9.** a) 1/26! b) 1/26 c) 1/2 d) 1/26 e) 1/650 f) 1/15,600 **11.** Clearly, $p(E \cup F) \geq p(E) = 0.7$. Also, $p(E \cup F) \leq 1$. If we apply Theorem 2 from Section 7.1, we can rewrite this as $p(E) + p(F) - p(E \cap F) \leq 1$, or $0.7 + 0.5 - p(E \cap F) \leq 1$. Solving for $p(E \cap F)$ gives $p(E \cap F) \geq 0.2$. **13.** Because $p(E \cup F) = p(E) + p(F) - p(E \cap F)$ and $p(E \cup F) \leq 1$, it follows that $1 \geq p(E) + p(F) - p(E \cap F)$. From this inequality we conclude that $p(E) + p(F) \leq 1 + p(E \cap F)$. **15.** We will use mathematical induction to prove that the inequality holds for $n \geq 2$. Let $P(n)$ be the statement that $p(\bigcup_{j=1}^n E_j) \leq \sum_{j=1}^n p(E_j)$. **Basis step:** $P(2)$ is true because $p(E_1 \cup E_2) = p(E_1) + p(E_2) - p(E_1 \cap E_2) \leq p(E_1) + p(E_2)$. **Inductive step:** Assume that $P(k)$ is true. Using the basis case and the inductive hypothesis, it follows that $p(\bigcup_{j=1}^{k+1} E_j) \leq p(\bigcup_{j=1}^k E_j) + p(E_{k+1}) \leq \sum_{j=1}^{k+1} p(E_j)$. This shows that $P(k+1)$ is true, completing the proof by mathematical induction. **17.** Because $E \cup \bar{E}$ is the entire sample space S , the event F can be split into two disjoint events: $F = S \cap F = (E \cup \bar{E}) \cap F = (E \cap F) \cup (\bar{E} \cap F)$, using the distributive law. Therefore, $p(F) = p((E \cap F) \cup (\bar{E} \cap F)) = p(E \cap F) + p(\bar{E} \cap F)$, because these two events are disjoint. Subtracting $p(E \cap F)$ from both sides, using the fact that $p(E \cap F) = p(E) \cdot p(F)$ (the hypothesis that E and F are independent), and factoring, we have $p(F)[1 - p(E)] = p(\bar{E} \cap F)$. Because $1 - p(E) = p(\bar{E})$, this says that $p(\bar{E} \cap F) = p(\bar{E}) \cdot p(F)$, as desired. **19.** a) 1/12 b) $1 - \frac{11}{12} \cdot \frac{10}{12} \dots \frac{13-n}{12}$ c) 5 **21.** 614 **23.** 1/4 **25.** 3/8 **27.** a) Not independent b) Not independent c) Not independent **29.** 3/16 **31.** a) $1/32 = 0.03125$ b) $0.49^5 \approx$

0.02825 c) 0.03795012 **33.** a) 5/8 b) 0.627649 c) 0.6431

- 35.** a) p^n b) $1 - p^n$ c) $p^n + n \cdot p^{n-1} \cdot (1 - p)$ d) $1 - [p^n + n \cdot p^{n-1} \cdot (1 - p)]$ **37.** $p(\bigcup_{i=1}^{\infty} E_i)$ is the sum of $p(s)$ for each outcome s in $\bigcup_{i=1}^{\infty} E_i$. Because the E_i 's are pairwise disjoint, this is the sum of the probabilities of all the outcomes in any of the E_i 's, which is what $\sum_{i=1}^{\infty} p(E_i)$ is. (We can rearrange the summands and still get the same answer because this series converges absolutely.) **39.** a) $\bar{E} = \bigcup_{j=1}^{\binom{n}{k}} F_j$, so the given inequality now follows from Boole's Inequality (Exercise 15). b) The probability that a particular player not in the j th set beats all k of the players in the j th set is $(1/2)^k = 2^{-k}$. Therefore, the probability that this player does not do so is $1 - 2^{-k}$, so the probability that all $m - k$ of the players not in the j th set are unable to boast of a perfect record against everyone in the j th set is $(1 - 2^{-k})^{m-k}$. That is precisely $p(F_j)$. c) The first inequality follows immediately, because all the summands are the same and there are $\binom{m}{k}$ of them. If this probability is less than 1, then it must be possible that \bar{E} fails, i.e., that E happens. So there is a tournament that meets the conditions of the problem as long as the second inequality holds. d) $m \geq 21$ for $k = 2$, and $m \geq 91$ for $k = 3$

41. **procedure** probabilistic prime(n, k)

```
composite := false
i := 0
while composite = false and i < k
    i := i + 1
    choose b uniformly at random with 1 < b < n
    apply Miller's test to base b
    if n fails the test then composite := true
if composite = true then print ("composite")
else print ("probably prime")
```

Section 7.3

NOTE: In the answers for Section 7.3, all probabilities given in decimal form are rounded to three decimal places. **1.** 3/5 **3.** 3/4 **5.** 0.481 **7.** a) 0.999 b) 0.324

- 9.** a) 0.740 b) 0.260 c) 0.002 d) 0.998 **11.** 0.724 **13.** 3/17 **15.** a) 1/3 b) $p(M = j \mid W = k) = 1$ if i, j , and k are distinct; $p(M = j \mid W = k) = 0$ if $j = k$ or $j = i$; $p(M = j \mid W = k) = 1/2$ if $i = k$ and $j \neq i$ c) 2/3 d) You should change doors, because you now have a 2/3 chance to win by switching. **17.** The definition of conditional probability tells us that $p(F_j \mid E) = p(E \cap F_j)/p(E)$. For the numerator, again using the definition of conditional probability, we have $p(E \cap F_j) = p(E \mid F_j)p(F_j)$, as desired. For the denominator, we show that $p(E) = \sum_{i=1}^n p(E \mid F_i)p(F_i)$. The events $E \cap F_i$ partition the event E ; that is, $(E \cap F_{i_1}) \cap (E \cap F_{i_2}) = \emptyset$ when $i_1 \neq i_2$ (because the F_i 's are mutually exclusive), and $\bigcup_{i=1}^n (E \cap F_i) = E$ (because the $\bigcup_{i=1}^n F_i = S$). Therefore, $p(E) = \sum_{i=1}^n p(E \cap F_i) = \sum_{i=1}^n p(E \mid F_i)p(F_i)$. **19.** No **21.** Yes **23.** By Bayes' theorem, $p(S \mid E_1 \cap E_2) = p(E_1 \cap E_2 \mid S)p(S)/[p(E_1 \cap E_2 \mid S)p(S) + p(E_1 \cap E_2 \mid \bar{S})p(\bar{S})]$.

Because we are assuming no prior knowledge about whether a message is or is not spam, we set $p(S) = p(\bar{S}) = 0.5$, and so the equation above simplifies to $p(S \mid E_1 \cap E_2) = p(E_1 \cap E_2 \mid S)/[p(E_1 \cap E_2 \mid S) + p(E_1 \cap E_2 \mid \bar{S})]$. Because of the assumed independence of E_1 , E_2 , and S , we have $p(E_1 \cap E_2 \mid S) = p(E_1 \mid S) \cdot p(E_2 \mid S)$, and similarly for \bar{S} .

Section 7.4

1. 2.5 3. 5/3 5. 336/49 7. 170 9. $(4n + 6)/3$
11. $50,700,551/10,077,696 \approx 5.03$ 13. 6 15. $p(X \geq j) = \sum_{k=j}^{\infty} p(X = k) = \sum_{k=j}^{\infty} (1-p)^{k-1} p = p(1-p)^{j-1} \sum_{k=0}^{\infty} (1-p)^k = p(1-p)^{j-1}/(1-(1-p)) = (1-p)^{j-1}$ 17. 2302 19. $(7/2) \cdot 7 \neq 329/12$ 21. 10
23. 1472 pounds 25. $p + (n-1)p(1-p)$ 27. 5/2
29. a) 0 b) n 31. This is not true. For example, let X be the number of heads in one flip of a fair coin, and let Y be the number of heads in one flip of a second fair coin. Then $A(X) + A(Y) = 1$ but $A(X + Y) = 0.5$. 33. a) We are told that X_1 and X_2 are independent. To see that X_1 and X_3 are independent, we enumerate the eight possibilities for (X_1, X_2, X_3) and find that $(0, 0, 0)$, $(1, 0, 1)$, $(0, 1, 1)$, $(1, 1, 0)$ each have probability $1/4$ and the others have probability 0 (because of the definition of X_3). Thus, $p(X_1 = 0 \wedge X_3 = 0) = 1/4$, $p(X_1 = 0) = 1/2$, and $p(X_3 = 0) = 1/2$, so it is true that $p(X_1 = 0 \wedge X_3 = 0) = p(X_1 = 0)p(X_3 = 0)$. Essentially the same calculation shows that $p(X_1 = 0 \wedge X_3 = 1) = p(X_1 = 0)p(X_3 = 1)$, $p(X_1 = 1 \wedge X_3 = 0) = p(X_1 = 1)p(X_3 = 0)$, and $p(X_1 = 1 \wedge X_3 = 1) = p(X_1 = 1)p(X_3 = 1)$. Therefore by definition, X_1 and X_3 are independent. The same reasoning shows that X_2 and X_3 are independent. To see that X_3 and $X_1 + X_2$ are not independent, we observe that $p(X_3 = 1 \wedge X_1 + X_2 = 2) = 0$. But $p(X_3 = 1)p(X_1 + X_2 = 2) = (1/2)(1/4) = 1/8$. b) We see from the calculation in part (a) that X_1 , X_2 , and X_3 are all Bernoulli random variables, so the variance of each is $(1/2)(1/2) = 1/4$. Therefore, $V(X_1) + V(X_2) + V(X_3) = 3/4$. We use the calculations in part (a) to see that $E(X_1 + X_2 + X_3) = 3/2$, and then $V(X_1 + X_2 + X_3) = 3/4$. c) In order to use the first part of Theorem 7 to show that $V((X_1 + X_2 + \dots + X_k) + X_{k+1}) = V(X_1 + X_2 + \dots + X_k) + V(X_{k+1})$ in the inductive step of a proof by mathematical induction, we would have to know that $X_1 + X_2 + \dots + X_k$ and X_{k+1} are independent, but we see from part (a) that this is not necessarily true. 35. 1/100
37. $E(X)/a = \sum_r (r/a) \cdot p(X = r) \geq \sum_{r \geq a} 1 \cdot p(X = r) = p(X \geq a)$ 39. a) 10/11 b) 0.9999 41. a) Each of the $n!$ permutations occurs with probability $1/n!$, so $E(X)$ is the number of comparisons, averaged over all these permutations.
- b) Even if the algorithm continues $n - 1$ rounds, X will be at most $n(n - 1)/2$. It follows from the formula for expectation that $E(X) \leq n(n - 1)/2$. c) The algorithm proceeds by comparing adjacent elements and then swapping them if necessary. Thus, the only way that inverted elements can become uninverted is for them to be compared and swapped.

- d) Because $X(P) \geq I(P)$ for all P , it follows from the definition of expectation that $E(X) \geq E(I)$. e) This summation counts 1 for every instance of an inversion. f) This follows from Theorem 3. g) By Theorem 2 with $n = 1$, the expectation of $I_{j,k}$ is the probability that a_k precedes a_j in the permutation. This is clearly $1/2$ by symmetry. h) The summation in part (f) consists of $C(n, 2) = n(n - 1)/2$ terms, each equal to $1/2$, so the sum is $n(n - 1)/4$. i) From part (a) and part (b) we know that $E(X)$, the object of interest, is at most $n(n - 1)/2$, and from part (d) and part (h) we know that $E(X)$ is at least $n(n - 1)/4$, both of which are $\Theta(n^2)$.
43. 1 45. $V(X + Y) = E((X + Y)^2) - E(X + Y)^2 = E(X^2 + 2XY + Y^2) - [E(X) + E(Y)]^2 = E(X^2) + 2E(XY) + E(Y^2) - E(X)^2 - 2E(X)E(Y) - E(Y)^2 = E(X^2) - E(X)^2 + 2[E(XY) - E(X)E(Y)] + E(Y^2) - E(Y)^2 = V(X) + 2\text{Cov}(X, Y) + V(Y)$ 47. $[(n - 1)/n]^m$
49. $(n - 1)^m / n^{m-1}$

Supplementary Exercises

1. 1/109,668 3. a) 1/195,249,054 b) 1/5,138,133
- c) 45/357,599 d) 18,285/18,821 5. a) $1/C(52, 13)$
- b) $4/C(52, 13)$ c) $2,944,656/C(52, 13)$ d) $35,335,872/C(52, 13)$
7. a) 9/2 b) 21/4 9. a) 9 b) 21/2 11. a) 8
- b) 49/6 13. a) $n/2^{n-1}$ b) $p(1-p)^{k-1}$, where $p = n/2^{n-1}$
- c) $2^{n-1}/n$ 15. $\frac{(m-1)(n-1)+\gcd(m,n)-1}{mn-1}$ 17. a) 2/3 b) 2/3
19. 1/32 21. a) The probability that one wins 2^n dollars is $1/2^n$, because that happens precisely when the player gets $n - 1$ tails followed by a head. The expected value of the winnings is therefore the sum of 2^n times $1/2^n$ as n goes from 1 to infinity. Because each of these terms is 1, the sum is infinite. In other words, one should be willing to wager any amount of money and expect to come out ahead in the long run. b) \$9, \$9
23. a) 1/3 when $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$, $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and $B = \{1, 2, 3, 4\}$; 1/12 when $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$, $A = \{4, 5, 6, 7, 8, 9, 10, 11, 12\}$, and $B = \{1, 2, 3, 4\}$
- b) 1 when $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$, $A = \{4, 5, 6, 7, 8, 9, 10, 11, 12\}$, and $B = \{1, 2, 3, 4\}$; 3/4 when $S = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$, $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, and $B = \{1, 2, 3, 4\}$
25. a) $p(E_1 \cap E_2) = p(E_1)p(E_2)$, $p(E_1 \cap E_3) = p(E_1)p(E_3)$, $p(E_2 \cap E_3) = p(E_2)p(E_3)$, $p(E_1 \cap E_2 \cap E_3) = p(E_1)p(E_2)p(E_3)$
- b) Yes c) Yes; yes d) Yes; no e) $2^n - n - 1$
27. a) 1/2 under first interpretation; 1/3 under second interpretation
- b) Let M be the event that both of Mr. Smith's children are boys and let B be the event that Mr. Smith chose a boy for today's walk. Then $p(M) = 1/4$, $p(B \mid M) = 1$, and $p(B \mid \bar{M}) = 1/3$. Apply Bayes' theorem to compute $p(M \mid B) = 1/2$.
- c) This variation is equivalent to the second interpretation discussed in part (a), so the answer is unambiguously 1/3.
29. $V(aX + b) = E((aX + b)^2) - E(aX + b)^2 = E(a^2X^2 + 2abX + b^2) - [aE(X) + b]^2 = E(a^2X^2) + E(2abX) + E(b^2) - [a^2E(X)^2 + 2abE(X) + b^2] = a^2E(X^2) + 2abE(X) + b^2 - a^2E(X)^2 - 2abE(X) - b^2 =$

$a^2[E(X^2) - E(X)^2] = a^2V(X)$

31. To count every element in the sample space exactly once, we must include every element in each of the sets and then take away the double counting of the elements in the intersections. Thus $p(E_1 \cup E_2 \cup \dots \cup E_m) = p(E_1) + p(E_2) + \dots + p(E_m) - p(E_1 \cap E_2) - p(E_1 \cap E_3) - \dots - p(E_1 \cap E_m) - p(E_2 \cap E_3) - p(E_2 \cap E_4) - \dots - p(E_2 \cap E_m) - \dots - p(E_{m-1} \cap E_m) = qm - (m(m-1)/2)r$, because $C(m, 2)$ terms are being subtracted. But $p(E_1 \cup E_2 \cup \dots \cup E_m) = 1$, so we have $qm - [m(m-1)/2]r = 1$. Because $r \geq 0$, this equation tells us that $qm \geq 1$, so $q \geq 1/m$. Because $q \leq 1$, this equation also implies that $[m(m-1)/2]r = qm - 1 \leq m - 1$, from which it follows that $r \leq 2/m$.

33. **a)** We purchase the cards until we have gotten one of each type. That means we have purchased X cards in all. On the other hand, that also means that we purchased X_0 cards until we got the first type we got, and then purchased X_1 more cards until we got the second type we got, and so on. Thus, X is the sum of the X_j 's. **b)** Once j distinct types have been obtained, there are $n - j$ new types available out of a total of n types available. Because it is equally likely that we get each type, the probability of success on the next purchase (getting a new type) is $(n - j)/n$. **c)** This follows immediately from the definition of geometric distribution, the definition of X_j , and part (b). **d)** From part (c) it follows that $E(X_j) = n/(n - j)$. Thus by the linearity of expectation and part (a), we have $E(X) = E(X_0) + E(X_1) + \dots + E(X_{n-1}) = \frac{n}{n} + \frac{n}{n-1} + \dots + \frac{n}{1} = n\left(\frac{1}{n} + \frac{1}{n-1} + \dots + \frac{1}{1}\right)$. **e)** About 224.46

35. $24 \cdot 13^4 / (52 \cdot 51 \cdot 50 \cdot 49)$

CHAPTER 8

Section 8.1

1. Let $P(n)$ be “ $H_n = 2^n - 1$.” **Basis step:** $P(1)$ is true because $H_1 = 1$. **Inductive step:** Assume that $H_n = 2^n - 1$. Then because $H_{n+1} = 2H_n + 1$, it follows that $H_{n+1} = 2(2^n - 1) + 1 = 2^{n+1} - 1$.

3. a) $a_n = 2a_{n-1} + a_{n-5}$ for $n \geq 5$ **b)** $a_0 = 1$, $a_1 = 2$, $a_2 = 4$, $a_3 = 8$, $a_4 = 16$ **c)** 1217 **5.** 9494

7. a) $a_n = a_{n-1} + a_{n-2} + 2^{n-2}$ for $n \geq 2$ **b)** $a_0 = 0$, $a_1 = 0$ **c)** 94

9. a) $a_n = a_{n-1} + a_{n-2} + a_{n-3}$ for $n \geq 3$ **b)** $a_0 = 1$, $a_1 = 2$, $a_2 = 4$ **c)** 81

11. a) $a_n = a_{n-1} + a_{n-2}$ for $n \geq 2$ **b)** $a_0 = 1$, $a_1 = 1$ **c)** 34

13. a) $a_n = 2a_{n-1} + 2a_{n-2}$ for $n \geq 2$ **b)** $a_0 = 1$, $a_1 = 3$ **c)** 448

15. a) $a_n = 2a_{n-1} + a_{n-2}$ for $n \geq 2$ **b)** $a_0 = 1$, $a_1 = 3$ **c)** 239

17. a) $a_n = 2a_{n-1}$ for $n \geq 2$ **b)** $a_1 = 3$ **c)** 96

19. a) $a_n = a_{n-1} + a_{n-2}$ for $n \geq 2$ **b)** $a_0 = 1$, $a_1 = 1$ **c)** 89

21. a) $R_n = n + R_{n-1}$, $R_0 = 1$ **b)** $R_n = n(n+1)/2 + 1$

23. a) $S_n = S_{n-1} + (n^2 - n + 2)/2$, $S_0 = 1$ **b)** $S_n = (n^3 + 5n + 6)/6$

25. a) $a_n = 2a_{n-1} + 2a_{n-2}$ **b)** $a_0 = 1$, $a_1 = 3$ **c)** 1224

29. Clearly, $S(m, 1) = 1$ for $m \geq 1$. If $m \geq n$, then a function that is not onto from the set with m elements to the set with n elements can be specified by picking the size of the range, which is an integer between 1 and $n - 1$ inclusive, picking the elements of the range, which can be done in $C(n, k)$ ways, and picking an onto function onto the range, which can be

done in $S(m, k)$ ways. Hence, there are $\sum_{k=1}^{n-1} C(n, k)S(m, k)$ functions that are not onto. But there are n^m functions altogether, so $S(m, n) = n^m - \sum_{k=1}^{n-1} C(n, k)S(m, k)$.

31. a) $C_5 = C_0C_4 + C_1C_3 + C_2C_2 + C_3C_1 + C_4C_0 = 1 \cdot 14 + 1 \cdot 5 + 2 \cdot 2 + 5 \cdot 1 + 14 \cdot 1 = 42$ **b)** $C(10, 5)/6 = 42$

33. $J(1) = 1$, $J(2) = 1$, $J(3) = 3$, $J(4) = 1$, $J(5) = 3$,

$J(6) = 5$, $J(7) = 7$, $J(8) = 1$, $J(9) = 3$, $J(10) = 5$,

$J(11) = 7$, $J(12) = 9$, $J(13) = 11$, $J(14) = 13$, $J(15) = 15$,

$J(16) = 1$ **35.** First, suppose that the number of people is even, say $2n$. After going around the circle once and returning to the first person, because the people at locations with even numbers have been eliminated, there are exactly n people left and the person currently at location i is the person who was originally at location $2i - 1$. Therefore, the survivor [originally in location $J(2n)$] is now in location $J(n)$; this was the person who was at location $2J(n) - 1$. Hence, $J(2n) = 2J(n) - 1$.

Similarly, when there are an odd number of people, say $2n + 1$, then after going around the circle once and then eliminating person 1, there are n people left and the person currently at location i is the person who was at location $2i + 1$. Therefore, the survivor will be the player currently occupying location $J(n)$, namely, the person who was originally at location $2J(n) + 1$. Hence, $J(2n + 1) = 2J(n) + 1$. The basis step is $J(1) = 1$.

37. 73, 977, 3617 **39.** These nine moves solve the puzzle:

Move disk 1 from peg 1 to peg 2; move disk 2 from peg 1 to peg 3; move disk 1 from peg 2 to peg 3; move disk 3 from peg 1 to peg 2; move disk 4 from peg 1 to peg 4; move disk 3 from peg 2 to peg 4; move disk 1 from peg 3 to peg 2; move disk 2 from peg 3 to peg 4; move disk 1 from peg 2 to peg 4.

To see that at least nine moves are required, first note that at least seven moves are required no matter how many pegs are present: three to unstack the disks, one to move the largest disk 4, and three more moves to restack them. At least two other moves are needed, because to move disk 4 from peg 1 to peg 4 the other three disks must be on pegs 2 and 3, so at least one move is needed to restack them and one move to unstack them.

41. The base cases are obvious. If $n > 1$, the algorithm consists of three stages. In the first stage, by the inductive hypothesis, $R(n - k)$ moves are used to transfer the smallest $n - k$ disks to peg 2. Then using the usual three-peg Tower of Hanoi algorithm, it takes $2^k - 1$ moves to transfer the rest of the disks (the largest k disks) to peg 4, avoiding peg 2. Then again by the inductive hypothesis, it takes $R(n - k)$ moves to transfer the smallest $n - k$ disks to peg 4; all the pegs are available for this, because the largest disks, now on peg 4, do not interfere. This establishes the recurrence relation.

43. First note that $R(n) = \sum_{j=1}^n [R(j) - R(j - 1)]$ [which follows because the sum is telescoping and $R(0) = 0$]. By Exercise 42, this is the sum of 2^{k-1} for this range of values of j . Therefore, the sum is $\sum_{i=1}^k i2^{i-1}$, except that if n is not a triangular number, then the last few values when $i = k$ are missing, and that is what the final term in the given expression accounts for.

45. By Exercise 43, $R(n)$ is no larger than $\sum_{i=1}^k i2^{i-1}$. It can be shown that this sum equals $(k+1)2^k - 2^{k+1} + 1$, so it is no greater than $(k+1)2^k$. Because $n > k(k-1)/2$, the quadratic formula can be used to show that

$k < 1 + \sqrt{2n}$ for all $n > 1$. Therefore, $R(n)$ is bounded above by $(1 + \sqrt{2n} + 1)2^{1+\sqrt{2n}} < 8\sqrt{n}2^{\sqrt{2n}}$ for all $n > 2$. Hence, $R(n)$ is $O(\sqrt{n}2^{\sqrt{2n}})$. **47.** a) 0 b) 0 c) 2 d) $2^{n-1} - 2^{n-2}$ **49.** $a_n - 2\nabla a_n + \nabla^2 a_n = a_n - 2(a_n - a_{n-1}) + (\nabla a_n - \nabla a_{n-1}) = -a_n + 2a_{n-1} + [(a_n - a_{n-1}) - (a_{n-1} - a_{n-2})] = -a_n + 2a_{n-1} + (a_n - 2a_{n-1} + a_{n-2}) = a_{n-2}$ **51.** $a_n = a_{n-1} + a_{n-2} = (a_n - \nabla a_n) + (a_n - 2\nabla a_n + \nabla^2 a_n) = 2a_n - 3\nabla a_n + \nabla^2 a_n$, or $a_n = 3\nabla a_n - \nabla^2 a_n$ **53.** Insert $S(0) := \emptyset$ after $T(0) := 0$ (where $S(j)$ will record the optimal set of talks among the first j talks), and replace the statement $T(j) := \max(w_j + T(p(j)), T(j - 1))$ with the following code:

```
if  $w_j + T(p(j)) > T(j - 1)$  then
   $T(j) := w_j + T(p(j))$ 
   $S(j) := S(p(j)) \cup \{j\}$ 
else
   $T(j) := T(j - 1)$ 
   $S(j) := S(j - 1)$ 
```

55. a) Talks 1, 3, and 7 b) Talks 1 and 6, or talks 1, 3, and 7 c) Talks 1, 3, and 7 d) Talks 1 and 6 **57.** a) This follows immediately from Example 5 and Exercise 41c in Section 8.4. b) The last step in computing \mathbf{A}_{ij} is to multiply \mathbf{A}_{ik} by $\mathbf{A}_{k+1,j}$ for some k between i and $j - 1$ inclusive, which will require $m_i m_{k+1} m_{j+1}$ integer multiplications, independent of the manner in which \mathbf{A}_{ik} and $\mathbf{A}_{k+1,j}$ are computed. Therefore to minimize the total number of integer multiplications, each of those two factors must be computed in the most efficient manner. c) This follows immediately from part (b) and the definition of $M(i, j)$.

d) procedure matrix order(m_1, \dots, m_{n+1} :

```
  positive integers)
  for  $i := 1$  to  $n$ 
     $M(i, i) := 0$ 
  for  $d := 1$  to  $n - 1$ 
    for  $i := 1$  to  $n - d$ 
      min := 0
      for  $k := i$  to  $i + d$ 
        new :=  $M(i, k) + M(k + 1, i + d) + m_i m_{k+1} m_{i+d+1}$ 
        if new < min then
          min := new
          where( $i, i + d$ ) := k
       $M(i, i + d) := min$ 
```

e) The algorithm has three nested loops, each of which is indexed over at most n values.

Section 8.2

1. a) Degree 3 b) No c) Degree 4 d) No e) No f) Degree 2 g) No **3.** a) $a_n = 3 \cdot 2^n$ b) $a_n = 2$ c) $a_n = 3 \cdot 2^n - 2 \cdot 3^n$ d) $a_n = 6 \cdot 2^n - 2 \cdot n2^n$ e) $a_n = n(-2)^{n-1}$ f) $a_n = 2^n - (-2)^n$ g) $a_n = (1/2)^{n+1} - (-1/2)^{n+1}$ **5.** $a_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^{n+1} - \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^{n+1}$ **7.** $[2^{n+1} + (-1)^n]/3$ **9.** a) $P_n = 1.2P_{n-1} + 0.45P_{n-2}$, $P_0 = 100,000$, $P_1 = 120,000$ b) $P_n = (250,000/3)(3/2)^n + (50,000/3)(-3/10)^n$

11. a) *Basis step:* For $n = 1$ we have $1 = 0 + 1$, and for $n = 2$ we have $3 = 1 + 2$. *Inductive step:* Assume true for $k \leq n$. Then $L_{n+1} = L_n + L_{n-1} = f_{n-1} + f_{n+1} + f_{n-2} + f_n = (f_{n-1} + f_{n-2}) + (f_{n+1} + f_n) = f_n + f_{n+2}$. b) $L_n = \left(\frac{1+\sqrt{5}}{2} \right)^n + \left(\frac{1-\sqrt{5}}{2} \right)^n$ **13.** $a_n = 8(-1)^n - 3(-2)^n + 4 \cdot 3^n$ **15.** $a_n = 5 + 3(-2)^n - 3^n$ **17.** Let $a_n = C(n, 0) + C(n-1, 1) + \dots + C(n-k, k)$ where $k = \lfloor n/2 \rfloor$. First, assume that n is even, so that $k = n/2$, and the last term is $C(n, k)$. By Pascal's identity we have $a_n = 1 + C(n-2, 0) + C(n-2, 1) + C(n-3, 1) + C(n-3, 2) + \dots + C(n-k, k-2) + C(n-k, k-1) + 1 = 1 + C(n-2, 1) + C(n-3, 2) + \dots + C(n-k, k-1) + C(n-2, 0) + C(n-3, 1) + \dots + C(n-k, k-2) + 1 = a_{n-1} + a_{n-2}$ because $\lfloor (n-1)/2 \rfloor = k-1 = \lfloor (n-2)/2 \rfloor$. A similar calculation works when n is odd. Hence, $\{a_n\}$ satisfies the recurrence relation $a_n = a_{n-1} + a_{n-2}$ for all positive integers n , $n \geq 2$. Also, $a_1 = C(1, 0) = 1$ and $a_2 = C(2, 0) + C(1, 1) = 2$, which are f_2 and f_3 . It follows that $a_n = f_{n+1}$ for all positive integers n . **19.** $a_n = (n^2 + 3n + 5)(-1)^n$ **21.** $(a_{1,0} + a_{1,1}n + a_{1,2}n^2 + a_{1,3}n^3) + (a_{2,0} + a_{2,1}n + a_{2,2}n^2)(-2)^n + (a_{3,0} + a_{3,1}n)3^n + a_{4,0}(-4)^n$ **23.** a) $3a_{n-1} + 2^n = 3(-2)^n + 2^n = 2^n(-3 + 1) = -2^{n+1} = a_n$ b) $a_n = \alpha 3^n - 2^{n+1}$ c) $a_n = 3^{n+1} - 2^{n+1}$ **25.** a) $A = -1$, $B = -7$ b) $a_n = \alpha 2^n - n - 7$ c) $a_n = 11 \cdot 2^n - n - 7$ **27.** a) $p_3n^3 + p_2n^2 + p_1n + p_0$ b) $n^2 p_0 (-2)^n$ c) $n^2 (p_1n + p_0) 2^n$ d) $(p_2n^2 + p_1n + p_0) 4^n$ e) $n^2 (p_2n^2 + p_1n + p_0) (-2)^n$ f) $n^2 (p_4n^4 + p_3n^3 + p_2n^2 + p_1n + p_0) 2^n$ g) p_0 **29.** a) $a_n = \alpha 2^n + 3^{n+1}$ b) $a_n = -2 \cdot 2^n + 3^{n+1}$ **31.** $a_n = \alpha 2^n + \beta 3^n - n \cdot 2^{n+1} + 3n/2 + 21/4$ **33.** $a_n = (\alpha + \beta n + n^2 + n^3/6) 2^n$ **35.** $a_n = -4 \cdot 2^n - n^2/4 - 5n/2 + 1/8 + (39/8)3^n$ **37.** $a_n = n(n+1)(n+2)/6$ **39.** a) 1, -1, i , $-i$ b) $a_n = \frac{1}{4} - \frac{1}{4}(-1)^n + \frac{2+i}{4}i^n + \frac{2-i}{4}(-i)^n$ **41.** a) Using the formula for f_n , we see that $|f_n - \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n| = \left| \frac{1}{\sqrt{5}} \left(\frac{1-\sqrt{5}}{2} \right)^n \right| < 1/\sqrt{5} < 1/2$. This means that f_n is the integer closest to $\frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n$. b) Less when n is even; greater when n is odd **43.** $a_n = f_{n-1} + 2f_n - 1$ **45.** a) $a_n = 3a_{n-1} + 4a_{n-2}$, $a_0 = 2$, $a_1 = 6$ b) $a_n = [4^{n+1} + (-1)^n]/5$ **47.** a) $a_n = 2a_{n+1} + (n-1)10,000$ b) $a_n = 70,000 \cdot 2^{n-1} - 10,000n - 10,000$ **49.** $a_n = 5n^2/12 + 13n/12 + 1$ **51.** See Chapter 11, Section 5 in [Ma93]. **53.** $6^n \cdot 4^{n-1}/n$

Section 8.3

1. 14 **3.** The first step is $(1110)_2(1010)_2 = (2^4 + 2^2)(11)_2 (10)_2 + 2^2[(11)_2 - (10)_2][(10)_2 - (10)_2] + (2^2 + 1)(10)_2 \cdot (10)_2$. The product is $(10001100)_2$. **5.** $C = 50,665C + 729 = 33,979$ **7.** a) 2 b) 4 c) 7 **9.** a) 79 b) 48,829 c) 30,517,579 **11.** $O(\log n)$ **13.** $O(n^{\log_3 2})$ **15.** 5 **17.** a) *Basis step:* If the sequence has just one element, then the one person on the list is the winner. *Recursive step:* Divide the list into two parts—the first half and the second half—as equally as possible. Apply the algorithm recursively to each half to come up with at most two

names. Then run through the entire list to count the number of occurrences of each of those names to decide which, if either, is the winner. **b)** $O(n \log n)$ **19. a)** $f(n) = f(n/2) + 2$
b) $O(\log n)$ **21. a)** 7 **b)** $O(\log n)$

23. a) **procedure** largest sum(a_1, \dots, a_n)

```
best := 0 {empty subsequence has sum 0}
for i := 1 to n
    sum := 0
    for j := i + 1 to n
        sum := sum +  $a_j$ 
        if sum > best then best := sum
{best is the maximum possible sum of numbers
    in the list}
```

b) $O(n^2)$ **c)** We divide the list into a first half and a second half and apply the algorithm recursively to find the largest sum of consecutive terms for each half. The largest sum of consecutive terms in the entire sequence is either one of these two numbers or the sum of a sequence of consecutive terms that crosses the middle of the list. To find the largest possible sum of a sequence of consecutive terms that crosses the middle of the list, we start at the middle and move forward to find the largest possible sum in the second half of the list, and move backward to find the largest possible sum in the first half of the list; the desired sum is the sum of these two quantities. The final answer is then the largest of this sum and the two answers obtained recursively. The base case is that the largest sum of a sequence of one term is the larger of that number and 0. **d)** 11, 9, 14 **e)** $S(n) = 2S(n/2) + n$, $C(n) = 2C(n/2) + n + 2$, $S(1) = 0$, $C(1) = 1$ **f)** $O(n \log n)$, better than $O(n^2)$ **25.** (1, 6) and (3, 6) at distance 2 **27.** The algorithm is essentially the same as the algorithm given in Example 12. The central strip still has width $2d$ but we need to consider just two boxes of size $d \times d$ rather than eight boxes of size $(d/2) \times (d/2)$. The recurrence relation is the same as the recurrence relation in Example 12, except that the coefficient 7 is replaced by 1. **29.** With $k = \log_b n$, it follows that $f(n) = a^k f(1) + \sum_{j=0}^{k-1} a^j c(n/b^j)^d = a^k f(1) + \sum_{j=0}^{k-1} cn^d = a^k f(1) + kcn^d = a^{\log_b n} f(1) + c(\log_b n)n^d = n^{\log_b a} f(1) + cn^d \log_b n = n^d f(1) + cn^d \log_b n$. **31.** Let $k = \log_b n$ where n is a power of b . *Basis step:* If $n = 1$ and $k = 0$, then $c_1 n^d + c_2 n^{\log_b a} = c_1 + c_2 = b^d c / (b^d - a) + f(1) + b^d c / (a - b^d) = f(1)$. *Inductive step:* Assume true for k , where $n = b^k$. Then for $n = b^{k+1}$, $f(n) = af(n/b) + cn^d = a\{[b^d c / (b^d - a)](n/b)^d + [f(1) + b^d c / (a - b^d)] \cdot (n/b)^{\log_b a}\} + cn^d = b^d c / (b^d - a)n^d a / b^d + [f(1) + b^d c / (a - b^d)]n^{\log_b a} + cn^d = n^d [ac / (b^d - a) + c(b^d - a) / (b^d - a)] + [f(1) + b^d c / (a - b^d)c]n^{\log_b a} = [b^d c / (b^d - a)]n^d + [f(1) + b^d c / (a - b^d)]n^{\log_b a}$. **33.** If $a > b^d$, then $\log_b a > d$, so the second term dominates, giving $O(n^{\log_b a})$. **35.** $O(n^{\log_4 5})$ **37.** $O(n^3)$

Section 8.4

- 1.** $f(x) = 2(x^6 - 1)/(x - 1)$ **3. a)** $f(x) = 2x(1 - x^6)/(1 - x)$ **b)** $x^3/(1 - x)$ **c)** $x/(1 - x^3)$ **d)** $2/(1 - 2x)$ **e)** $(1 + x)^7$ **f)** $2/(1 + x)$ **g)** $[1/(1 - x)] - x^2$ **h)** $x^3/(1 - x)^2$ **5. a)** $5/(1 - x)$ **b)** $1/(1 - 3x)$ **c)** $2x^3/(1 - x)$ **d)** $(3 - x)/(1 - x)^2$ **e)** $(1 + x)^8$ **7. a)** $a_0 = -64$, $a_1 = 144$, $a_2 = -108$, $a_3 = 27$, and $a_n = 0$ for all $n \geq 4$ **b)** The only nonzero coefficients are $a_0 = 1$, $a_3 = 3$, $a_6 = 3$, $a_9 = 1$. **c)** $a_n = 5^n$ **d)** $a_n = (-3)^{n-3}$ for $n \geq 3$, and $a_0 = a_1 = a_2 = 0$ **e)** $a_0 = 8$, $a_1 = 3$, $a_2 = 2$, $a_n = 0$ for odd n greater than 2 and $a_n = 1$ for even n greater than 2 **f)** $a_n = 1$ if n is a positive multiple 4, $a_n = -1$ if $n < 4$, and $a_n = 0$ otherwise **g)** $a_n = n - 1$ for $n \geq 2$ and $a_0 = a_1 = 0$ **h)** $a_n = 2^{n+1}/n!$ **9. a)** 6 **b)** 3 **c)** 9 **d)** 0 **e)** 5 **11. a)** 1024 **b)** 11 **c)** 66 **d)** 292,864 **e)** 20,412 **13. 10** **15. 50** **17. 20** **19. f(x) = 1/[(1 - x)(1 - x^2)(1 - x^5)(1 - x^{10})]** **21. 15** **23. a)** $x^4(1 + x + x^2 + x^3)^2 / (1 - x)$ **b)** 6 **25. a)** The coefficient of x^r in the power series expansion of $1/[(1 - x^3)(1 - x^4)(1 - x^{20})]$ **b)** $1/(1 - x^3 - x^4 - x^{20})$ **c)** 7 **d)** 3224 **27. a)** 3 **b)** 29 **c)** 29 **d)** 242 **29. a)** 10 **b)** 49 **c)** 2 **d)** 4 **31. a)** $G(x) - a_0 - a_1x - a_2x^2$ **b)** $G(x^2)$ **c)** $x^4G(x)$ **d)** $G(2x)$ **e)** $\int_0^x G(t)dt$ **f)** $G(x)/(1 - x)$ **33. a)** $k = 2 \cdot 3^k - 1$ **35. a)** $k = 18 \cdot 3^k - 12 \cdot 2^k$ **37. a)** $k^2 + 8k + 20 + (6k - 18)2^k$ **39.** Let $G(x) = \sum_{k=0}^{\infty} f_k x^k$. After shifting indices of summation and adding series, we see that $G(x) - xG(x) - x^2G(x) = f_0 + (f_1 - f_0)x + \sum_{k=2}^{\infty} (f_k - f_{k-1} - f_{k-2})x^k = 0 + x + \sum_{k=2}^{\infty} ox^k$. Hence, $G(x) - xG(x) - x^2G(x) = x$. Solving for $G(x)$ gives $G(x) = x/(1 - x - x^2)$. By the method of partial fractions, it can be shown that $x/(1 - x - x^2) = (1/\sqrt{5})[1/(1 - \alpha x) - 1/(1 - \beta x)]$, where $\alpha = (1 + \sqrt{5})/2$ and $\beta = (1 - \sqrt{5})/2$. Using the fact that $1/(1 - \alpha x) = \sum_{k=0}^{\infty} \alpha^k x^k$, it follows that $G(x) = (1/\sqrt{5}) \cdot \sum_{k=0}^{\infty} (\alpha^k - \beta^k)x^k$. Hence, $f_k = (1/\sqrt{5}) \cdot (\alpha^k - \beta^k)$. **41. a)** Let $G(x) = \sum_{n=0}^{\infty} C_n x^n$ be the generating function for $\{C_n\}$. Then $G(x)^2 = \sum_{n=0}^{\infty} (\sum_{k=0}^n C_k C_{n-k}) x^n = \sum_{n=1}^{\infty} (\sum_{k=0}^{n-1} C_k C_{n-1-k}) x^{n-1} = \sum_{n=1}^{\infty} C_n x^{n-1}$. Hence, $xG(x)^2 = \sum_{n=1}^{\infty} C_n x^n$, which implies that $xG(x)^2 - G(x) + 1 = 0$. Applying the quadratic formula shows that $G(x) = \frac{1 \pm \sqrt{1 - 4x}}{2x}$. We choose the minus sign in this formula because the choice of the plus sign leads to a division by zero. **b)** By Exercise 40, $(1 - 4x)^{-1/2} = \sum_{n=0}^{\infty} \binom{2n}{n} x^n$. Integrating term by term (which is valid by a theorem from calculus) shows that $\int_0^x (1 - 4t)^{-1/2} dt = \sum_{n=0}^{\infty} \frac{1}{n+1} \binom{2n}{n} x^{n+1} = x \sum_{n=0}^{\infty} \frac{1}{n+1} \binom{2n}{n} x^n$. Because $\int_0^x (1 - 4t)^{-1/2} dt = \frac{1 - \sqrt{1 - 4x}}{2} = xG(x)$, equating coefficients shows that $C_n = \frac{1}{n+1} \binom{2n}{n}$. **c)** Verify the basis step for $n = 1, 2, 3, 4, 5$. Assume the inductive hypothesis that $C_j \geq 2^{j-1}$ for $1 \leq j < n$, where $n \geq 6$. Then $C_n = \sum_{k=0}^{n-1} C_k C_{n-k-1} \geq \sum_{k=1}^{n-2} C_k C_{n-k-1} \geq (n-2)2^{k-1}2^{n-k-2} = (n-2)2^{n-1}/4 \geq 2^{n-1}$. **43.** Applying the binomial theorem to the equality $(1 + x)^{m+n} = (1 + x)^m(1 + x)^n$, shows that $\sum_{r=0}^{m+n} C(m + n, r)x^r = \sum_{r=0}^m C(m, r)x^r \cdot \sum_{r=0}^n C(n, r)x^r = \sum_{r=0}^{m+n} [\sum_{k=0}^r C(m, r-k)C(n, k)]x^r$. Comparing coefficients gives the desired identity. **45. a)** $2e^x$ **b)** e^{-x} **c)** e^{3x} **d)** $xe^x + e^x$ **47. a)** $a_n = (-1)^n$ **b)** $a_n = 3 \cdot 2^n$

c) $a_n = 3^n - 3 \cdot 2^n$ **d)** $a_n = (-2)^n$ for $n \geq 2$, $a_1 = -3$, $a_0 = 2$ **e)** $a_n = (-2)^n + n!$ **f)** $a_n = (-3)^n + n! \cdot 2^n$ for $n \geq 2$, $a_0 = 1$, $a_1 = -2$ **g)** $a_n = 0$ if n is odd and $a_n = n!/((n/2)!)$ if n is even **49. a)** $a_n = 6a_{n-1} + 8^{n-1}$ for $n \geq 1$, $a_0 = 1$ **b)** The general solution of the associated linear homogeneous recurrence relation is $a_n^{(h)} = \alpha 6^n$. A particular solution is $a_n^{(p)} = \frac{1}{2} \cdot 8^n$. Hence, the general solution is $a_n = \alpha 6^n + \frac{1}{2} \cdot 8^n$. Using the initial condition, it follows that $\alpha = \frac{1}{2}$. Hence, $a_n = (6^n + 8^n)/2$. **c)** Let $G(x) = \sum_{k=0}^{\infty} a_k x^k$. Using the recurrence relation for $\{a_k\}$, it can be shown that $G(x) - 6xG(x) = (1-7x)/(1-8x)$. Hence, $G(x) = (1-7x)/[(1-6x)(1-8x)]$. Using partial fractions, it follows that $G(x) = (1/2)/(1-6x) + (1/2)/(1-8x)$. With the help of Table 1, it follows that $a_n = (6^n + 8^n)/2$.

51. $\frac{1}{1-x} \cdot \frac{1}{1-x^2} \cdot \frac{1}{1-x^3} \cdots$ **53.** $(1+x)(1+x)^2(1+x)^3 \cdots$

55. The generating functions obtained in Exercises 52 and 53 are equal because $(1+x)(1+x^2)(1+x^3) \cdots = \frac{1-x^2}{1-x} \cdot \frac{1-x^4}{1-x^2} \cdots = \frac{1-x^6}{1-x^3} \cdots = \frac{1}{1-x} \cdot \frac{1}{1-x^3} \cdot \frac{1}{1-x^5} \cdots$

57. a) $G_X(1) = \sum_{k=0}^{\infty} p(X=k) \cdot 1^k = \sum_{k=0}^{\infty} P(X=k) = 1$ **b)** $G'_X(1) = \frac{d}{dx} \sum_{k=0}^{\infty} p(X=k) \cdot x^k|_{x=1} = \sum_{k=0}^{\infty} p(X=k) \cdot k \cdot x^{k-1}|_{x=1} = \sum_{k=0}^{\infty} p(X=k) \cdot k = E(X)$ **c)** $G''_X(1) = \frac{d^2}{dx^2} \sum_{k=0}^{\infty} p(X=k) \cdot x^k|_{x=1} = \sum_{k=0}^{\infty} p(X=k) \cdot k(k-1) \cdot x^{k-2}|_{x=1} = \sum_{k=0}^{\infty} p(X=k) \cdot (k^2-k) = V(X) + E(X)^2 - E(X)$. Combining this with part (b) gives the desired results. **59. a)** $G(x) = p^m/(1-qx)^m$ **b)** $V(x) = mq/p^2$

Section 8.5

- 1. a)** 30 **b)** 29 **c)** 24 **d)** 18 **3. 1%** **5. a)** 300 **b)** 150 **c)** 175 **d)** 100 **7. 492** **9. 974** **11. 55** **13. 248** **15. 50,138** **17. 234**
- 19.** $|A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5| = |A_1| + |A_2| + |A_3| + |A_4| + |A_5| - |A_1 \cap A_2| - |A_1 \cap A_3| - |A_1 \cap A_4| - |A_1 \cap A_5| - |A_2 \cap A_3| - |A_2 \cap A_4| - |A_2 \cap A_5| - |A_3 \cap A_4| - |A_3 \cap A_5| - |A_4 \cap A_5| + |A_1 \cap A_2 \cap A_3| + |A_1 \cap A_2 \cap A_4| + |A_1 \cap A_2 \cap A_5| + |A_1 \cap A_3 \cap A_4| + |A_1 \cap A_3 \cap A_5| + |A_1 \cap A_4 \cap A_5| + |A_2 \cap A_3 \cap A_4| + |A_2 \cap A_3 \cap A_5| + |A_2 \cap A_4 \cap A_5| + |A_3 \cap A_4 \cap A_5| - |A_1 \cap A_2 \cap A_3 \cap A_4| - |A_1 \cap A_2 \cap A_3 \cap A_5| - |A_1 \cap A_2 \cap A_4 \cap A_5| - |A_1 \cap A_3 \cap A_4 \cap A_5| - |A_2 \cap A_3 \cap A_4 \cap A_5| + |A_1 \cap A_2 \cap A_3 \cap A_4 \cap A_5|$
- 21.** $|A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 \cup A_6| = |A_1| + |A_2| + |A_3| + |A_4| + |A_5| + |A_6| - |A_1 \cap A_2| - |A_1 \cap A_3| - |A_1 \cap A_4| - |A_1 \cap A_5| - |A_1 \cap A_6| - |A_2 \cap A_3| - |A_2 \cap A_4| - |A_2 \cap A_5| - |A_2 \cap A_6| - |A_3 \cap A_4| - |A_3 \cap A_5| - |A_3 \cap A_6| - |A_4 \cap A_5| - |A_4 \cap A_6| - |A_5 \cap A_6|$
- 23.** $p(E_1 \cup E_2 \cup E_3) = p(E_1) + p(E_2) + p(E_3) - p(E_1 \cap E_2) - p(E_1 \cap E_3) - p(E_2 \cap E_3) + p(E_1 \cap E_2 \cap E_3)$
- 25.** $4972/71,295$ **27.** $p(E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5) = p(E_1) + p(E_2) + p(E_3) + p(E_4) + p(E_5) - p(E_1 \cap E_2) - p(E_1 \cap E_3) - p(E_1 \cap E_4) - p(E_1 \cap E_5) - p(E_2 \cap E_3) - p(E_2 \cap E_4) - p(E_2 \cap E_5) - p(E_3 \cap E_4) - p(E_3 \cap E_5) - p(E_4 \cap E_5) + p(E_1 \cap E_2 \cap E_3) + p(E_1 \cap E_2 \cap E_4) + p(E_1 \cap E_2 \cap E_5) + p(E_1 \cap E_3 \cap E_4) + p(E_1 \cap E_3 \cap E_5) + p(E_1 \cap E_4 \cap E_5) + p(E_2 \cap E_3 \cap E_4) + p(E_2 \cap E_3 \cap E_5) + p(E_2 \cap E_4 \cap E_5) + p(E_3 \cap E_4 \cap E_5)$
- 29.** $p(\bigcup_{i=1}^n E_i) = \sum_{1 \leq i \leq n} p(E_i) - \sum_{1 \leq i < j \leq n} p(E_i \cap E_j) + \sum_{1 \leq i < j < k \leq n} p(E_i \cap E_j \cap E_k) - \cdots + (-1)^{n+1} p(\bigcap_{i=1}^n E_i)$

Section 8.6

- 1. 75** **3. 6** **5. 46** **7. 9875** **9. 540** **11. 2100** **13. 1854**
- 15. a)** $D_{100}/100!$ **b)** $100D_{99}/100!$ **c)** $C(100,2)/100!$
- d)** 0 **e)** $1/100!$ **17.** 2,170,680 **19.** By Exercise 18 we have $D_n - nD_{n-1} = -[D_{n-1} - (n-1)D_{n-2}]$. Iterating, we have $D_n - nD_{n-1} = -[D_{n-1} - (n-1)D_{n-2}] = -[-(D_{n-2} - (n-2)D_{n-3})] = D_{n-2} - (n-2)D_{n-3} = \cdots = (-1)^n(D_2 - 2D_1) = (-1)^n$ because $D_2 = 1$ and $D_1 = 0$. **21.** When n is odd **23.** $\phi(n) = n - \sum_{i=1}^m \frac{n}{p_i} + \sum_{1 \leq i < j \leq m} \frac{n}{p_i p_j} - \cdots \pm \frac{n}{p_1 p_2 \cdots p_m} = n \prod_{i=1}^m \left(a - \frac{1}{p_i}\right)$ **25. 4**
- 27.** There are n^m functions from a set with m elements to a set with n elements, $C(n, 1)(n-1)^m$ functions from a set with m elements to a set with n elements that miss exactly one element, $C(n, 2)(n-2)^m$ functions from a set with m elements to a set with n elements that miss exactly two elements, and so on, with $C(n, n-1) \cdot 1^m$ functions from a set with m elements to a set with n elements that miss exactly $n-1$ elements. Hence, by the principle of inclusion-exclusion, there are $n^m - C(n, 1)(n-1)^m + C(n, 2)(n-2)^m - \cdots + (-1)^{n-1} C(n, n-1) \cdot 1^m$ onto functions.

Supplementary Exercises

- 1. a)** $A_n = 4A_{n-1}$ **b)** $A_1 = 40$ **c)** $A_n = 10 \cdot 4^n$
- 3. a)** $M_n = M_{n-1} + 160,000$ **b)** $M_1 = 186,000$ **c)** $M_n = 160,000n + 26,000$ **d)** $T_n = T_{n-1} + 160,000n + 26,000$
- e)** $T_n = 80,000n^2 + 106,000n$ **5. a)** $a_n = a_{n-2} + a_{n-3}$
- b)** $a_1 = 0$, $a_2 = 1$, $a_3 = 1$ **c)** $a_{12} = 12$ **7. a)** 2 **b)** 5 **c)** 8
- d)** 16 **9. a)** $a_n = 2^n$ **11. a)** $a_n = 2 + 4n/3 + n^2/2 + n^3/6$
- 13. a)** $a_n = a_{n-2} + a_{n-3}$ **15. a)** Under the given conditions, one longest common subsequence clearly ends at the last term in each sequence, so $a_m = b_n = c_p$. Furthermore, a longest common subsequence of what is left of the a -sequence and the b -sequence after those last terms are deleted has to form the beginning of a longest common subsequence of the original sequences. **b)** If $c_p \neq a_m$, then the longest common subsequence's appearance in the a -sequence must terminate before the end; therefore the c -sequence must be a longest common subsequence of a_1, a_2, \dots, a_{m-1} and b_1, b_2, \dots, b_n . The other half is similar.
- 17. procedure howlong($a_1, \dots, a_m, b_1, \dots, b_n$: sequences)**
- ```

for i := 1 to m
 L(i, 0) := 0
for j := 1 to n
 L(0, j) := 0
for i := 1 to m
 for j := 1 to n
 if $a_i = b_j$ then $L(i, j) := L(i-1, j-1) + 1$
 else $L(i, j) := \max(L(i, j-1), L(i-1, j))$
return L(m, n)

```

- 19.**  $f(n) = (4n^2 - 1)/3$    **21.**  $O(n^4)$    **23.**  $O(n)$    **25.** Using just two comparisons, the algorithm is able to narrow the search for  $m$  down to the first half or the second half of the original sequence. Since the length of the sequence is cut in half each time, only about  $2 \log_2 n$  comparisons are needed in all.   **27.** a)  $18n + 18$    b) 18   c) 0   **29.**  $\Delta(a_n b_n) = a_{n+1} b_{n+1} - a_n b_n = a_{n+1}(b_{n+1} - b_n) + b_n(a_{n+1} - a_n) = a_{n+1} \Delta b_n + b_n \Delta a_n$    **31.** a) Let  $G(x) = \sum_{n=0}^{\infty} a_n x^n$ . Then  $G'(x) = \sum_{n=1}^{\infty} n a_n x^{n-1} = \sum_{n=0}^{\infty} (n+1) a_{n+1} x^n$ . Therefore,  $G'(x) - G(x) = \sum_{n=0}^{\infty} [(n+1)a_{n+1} - a_n] x^n = \sum_{n=0}^{\infty} x^n / n! = e^x$ , as desired. That  $G(0) = a_0 = 1$  is given. b) We have  $[e^{-x} G(x)]' = e^{-x} G'(x) - e^{-x} G(x) = e^{-x} [G'(x) - G(x)] = e^{-x} \cdot e^x = 1$ . Hence,  $e^{-x} G(x) = x + c$ , where  $c$  is a constant. Consequently,  $G(x) = x e^x + c e^x$ . Because  $G(0) = 1$ , it follows that  $c = 1$ . c) We have  $G(x) = \sum_{n=0}^{\infty} x^{n+1} / n! + \sum_{n=0}^{\infty} x^n / n! = \sum_{n=1}^{\infty} x^n / (n-1)! + \sum_{n=0}^{\infty} x^n / n!$ . Therefore,  $a_n = 1/(n-1)! + 1/n!$  for all  $n \geq 1$ , and  $a_0 = 1$ .   **33.** 7   **35.** 110   **37.** 0   **39.** a) 19   b) 65   c) 122   d) 167   e) 168   **41.**  $D_{n-1}/(n-1)!$    **43.** 11/32

## CHAPTER 9

### Section 9.1

- 1.** a)  $\{(0, 0), (1, 1), (2, 2), (3, 3)\}$    b)  $\{(1, 3), (2, 2), (3, 1), (4, 0)\}$    c)  $\{(1, 0), (2, 0), (2, 1), (3, 0), (3, 1), (3, 2), (4, 0), (4, 1), (4, 2), (4, 3)\}$    d)  $\{(1, 0), (1, 1), (1, 2), (1, 3), (2, 0), (2, 2), (3, 0), (3, 3), (4, 0)\}$    e)  $\{(0, 1), (1, 0), (1, 1), (1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2), (4, 1), (4, 3)\}$    f)  $\{(1, 2), (2, 1), (2, 2)\}$    **3.** a) Transitive   b) Reflexive, symmetric, transitive   c) Symmetric   d) Antisymmetric   e) Reflexive, symmetric, antisymmetric, transitive   f) None of these properties   **5.** a) Reflexive, transitive   b) Symmetric   c) Symmetric   d) Symmetric   **7.** a) Symmetric   b) Symmetric, transitive   c) Symmetric   d) Reflexive, symmetric, transitive   e) Reflexive, transitive   f) Reflexive, symmetric, transitive   g) Antisymmetric   h) Antisymmetric, transitive   **9.** Each of the three properties is vacuously satisfied.   **11.** (c), (d), (f)   **13.** a) Not irreflexive   b) Not irreflexive   c) Not irreflexive   d) Not irreflexive   **15.** Yes, for instance  $\{(1, 1)\}$  on  $\{1, 2\}$    **17.**  $(a, b) \in R$  if and only if  $a$  is taller than  $b$    **19.** (a)   **21.** None   **23.**  $\forall a \forall b [(a, b) \in R \rightarrow (b, a) \notin R]$    **25.**  $2^{mn}$    **27.** a)  $\{(a, b) \mid b \text{ divides } a\}$    b)  $\{(a, b) \mid a \text{ does not divide } b\}$    **29.** The graph of  $f^{-1}$    **31.** a)  $\{(a, b) \mid a \text{ is required to read or has read } b\}$    b)  $\{(a, b) \mid a \text{ is required to read and has read } b\}$    c)  $\{(a, b) \mid \text{either } a \text{ is required to read } b \text{ but has not read it or } a \text{ has read } b \text{ but is not required to}\}$    d)  $\{(a, b) \mid a \text{ is required to read } b \text{ but has not read it}\}$    e)  $\{(a, b) \mid a \text{ has read } b \text{ but is not required to}\}$    **33.**  $S \circ R = \{(a, b) \mid a \text{ is a parent of } b \text{ and } b \text{ has a sibling}\}$ ,  $R \circ S = \{(a, b) \mid a \text{ is an aunt}$

- or uncle of } b\}
- 35.** a)  $\mathbf{R}^2$    b)  $R_6$    c)  $R_3$    d)  $R_3$    e)  $\emptyset$    f)  $R_1$    g)  $R_4$    h)  $R_4$    **37.** a)  $R_1$    b)  $R_2$    c)  $R_3$    d)  $\mathbf{R}^2$    e)  $R_3$    f)  $\mathbf{R}^2$    g)  $\mathbf{R}^2$    h)  $\mathbf{R}^2$    **39.**  $b$  got his or her doctorate under someone who got his or her doctorate under  $a$ ; there is a sequence of  $n + 1$  people, starting with  $a$  and ending with  $b$ , such that each is the advisor of the next person in the sequence
- 41.** a)  $\{(a, b) \mid a - b \equiv 0, 3, 4, 6, , 8, \text{ or } 9 \pmod{12}\}$    b)  $\{(a, b) \mid a \equiv b \pmod{12}\}$    c)  $\{(a, b) \mid a - b \equiv 3, 6, \text{ or } 9 \pmod{12}\}$    d)  $\{(a, b) \mid a - b \equiv 4 \text{ or } 8 \pmod{12}\}$    e)  $\{(a, b) \mid a - b \equiv 3, 4, 6, 8, \text{ or } 9 \pmod{12}\}$    **43.** 8   **45.** a) 65,536   b) 32,768   **47.** a)  $2^{n(n+1)/2}$    b)  $2^n 3^{n(n-1)/2}$    c)  $3^{n(n-1)/2}$    d)  $2^{n(n-1)}$    e)  $2^{n(n-1)/2}$    f)  $2^{n^2} - 2 \cdot 2^{n(n-1)}$
- 49.** There may be no such  $b$ .   **51.** If  $R$  is symmetric and  $(a, b) \in R$ , then  $(b, a) \in R$ , so  $(a, b) \in R^{-1}$ . Hence,  $R \subseteq R^{-1}$ . Similarly,  $R^{-1} \subseteq R$ . So  $R = R^{-1}$ . Conversely, if  $R = R^{-1}$  and  $(a, b) \in R$ , then  $(a, b) \in R^{-1}$ , so  $(b, a) \in R$ . Thus  $R$  is symmetric.   **53.**  $R$  is reflexive if and only if  $(a, a) \in R$  for all  $a \in A$  if and only if  $(a, a) \in R^{-1}$  [because  $(a, a) \in R$  if and only if  $(a, a) \in R^{-1}$ ] if and only if  $R^{-1}$  is reflexive.   **55.** Use mathematical induction. The result is trivial for  $n = 1$ . Assume  $R^n$  is reflexive and transitive. By Theorem 1,  $R^{n+1} \subseteq R$ . To see that  $R \subseteq R^{n+1} = R^n \circ R$ , let  $(a, b) \in R$ . By the inductive hypothesis,  $R^n = R$  and hence, is reflexive. Thus  $(b, b) \in R^n$ . Therefore  $(a, b) \in R^{n+1}$ .   **57.** Use mathematical induction. The result is trivial for  $n = 1$ . Assume  $R^n$  is reflexive. Then  $(a, a) \in R^n$  for all  $a \in A$  and  $(a, a) \in R$ . Thus  $(a, a) \in R^n \circ R = R^{n+1}$  for all  $a \in A$ .   **59.** No, for instance, take  $R = \{(1, 2), (2, 1)\}$ .

### Section 9.2

- 1.**  $\{(1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 4)\}$    **3.** (Nadir, 122, 34, Detroit, 08:10), (Acme, 221, 22, Denver, 08:17), (Acme, 122, 33, Anchorage, 08:22), (Acme, 323, 34, Honolulu 08:30), (Nadir, 199, 13, Detroit, 08:47), (Acme, 222, 22, Denver, 09:10), (Nadir, 322, 34, Detroit, 09:44)   **5.** Airline and flight number, airline and departure time   **7.** a) Yes   b) No   c) No   **9.** a) Social Security number   b) There are no two people with the same name who happen to have the same street address.   c) There are no two people with the same name living together.   **11.** (Nadir, 122, 34, Detroit, 08 : 10), (Nadir, 199, 13, Detroit, 08 : 47), (Nadir, 322, 34, Detroit, 09 : 44)   **13.** (Nadir, 122, 34, Detroit, 08 : 10), (Nadir, 199, 13, Detroit, 08 : 47), (Nadir, 322, 34, Detroit, 09 : 44), (Acme, 221, 22, Denver, 08 : 17), (Acme, 222, 22, Denver, 09 : 10)   **15.** P<sub>3.5.6</sub>

| 17. <b>Airline</b> | <b>Destination</b> |
|--------------------|--------------------|
| Nadir              | Detroit            |
| Acme               | Denver             |
| Acme               | Anchorage          |
| Acme               | Honolulu           |

| Supplier | Part_number | Project | Quantity | Color_code |
|----------|-------------|---------|----------|------------|
| 23       | 1092        | 1       | 2        | 2          |
| 23       | 1101        | 3       | 1        | 1          |
| 23       | 9048        | 4       | 12       | 2          |
| 31       | 4975        | 3       | 6        | 2          |
| 31       | 3477        | 2       | 25       | 2          |
| 32       | 6984        | 4       | 10       | 1          |
| 32       | 9191        | 2       | 80       | 4          |
| 33       | 1001        | 1       | 14       | 8          |

21. Both sides of this equation pick out the subset of  $R$  consisting of those  $n$ -tuples satisfying both conditions  $C_1$  and  $C_2$ .  
 23. Both sides of this equation pick out the set of  $n$ -tuples that are in  $R$ , are in  $S$ , and satisfy condition  $C$ .  
 25. Both sides of this equation pick out the  $m$ -tuples consisting of  $i_1$ th,  $i_2$ th, ...,  $i_m$ th components of  $n$ -tuples in either  $R$  or  $S$ .  
 27. Let  $R = \{(a, b)\}$  and  $S = \{(a, c)\}$ ,  $n = 2$ ,  $m = 1$ , and  $i_1 = 1$ ;  $P_1(R - S) = \{(a)\}$ , but  $P_1(R) - P_1(S) = \emptyset$ .  
 29. a)  $J_2$  followed by  $P_{1,3}$  b)  $(23, 1), (23, 3), (31, 3), (32, 4)$   
 31. There is no primary key.

### Section 9.3

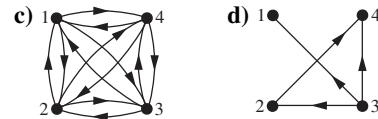
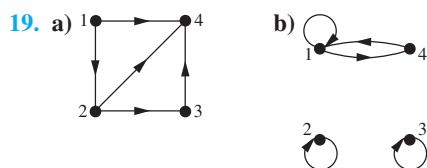
1. a)  $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$  b)  $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$   
 c)  $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$  d)  $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$

3. a)  $(1, 1), (1, 3), (2, 2), (3, 1), (3, 3)$  b)  $(1, 2), (2, 2), (3, 2)$  c)  $(1, 1), (1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2), (3, 3)$   
 5. The relation is irreflexive if and only if the main diagonal of the matrix contains only 0s. 7. a) Reflexive, symmetric, transitive b) Antisymmetric, transitive c) Symmetric  
 9. a) 4950 b) 9900 c) 99 d) 100 e) 1 11. Change each 0 to a 1 and each 1 to a 0.

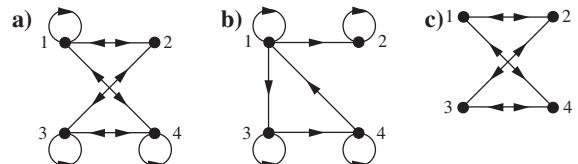
13. a)  $\begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$  b)  $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$  c)  $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

15. a)  $\begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$  b)  $\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$  c)  $\begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

17.  $n^2 - k$



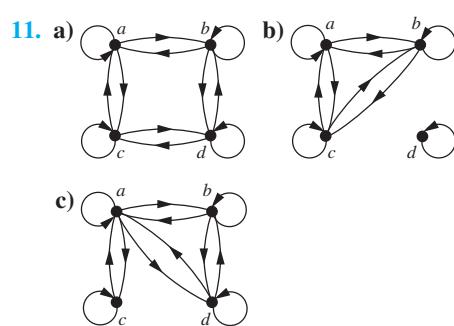
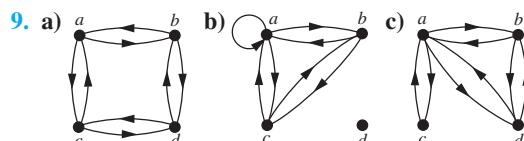
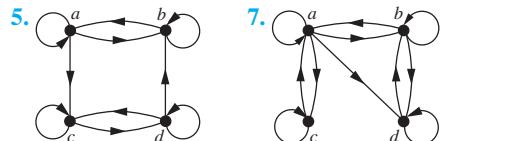
21. For simplicity we have indicated pairs of edges between the same two vertices in opposite directions by using a double arrowhead, rather than drawing two separate lines.



23.  $\{(a, b), (a, c), (b, c), (c, b)\}$  25.  $(a, c), (b, a), (c, d), (d, b)$  27.  $\{(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, a), (c, b), (d, d)\}$  29. The relation is asymmetric if and only if the directed graph has no loops and no closed paths of length 2. 31. Exercise 23: irreflexive. Exercise 24: reflexive, antisymmetric, transitive. Exercise 25: irreflexive, anti-symmetric. 33. Reverse the direction on every edge in the digraph for  $R$ . 35. Proof by mathematical induction. *Basis step:* Trivial for  $n = 1$ . *Inductive step:* Assume true for  $k$ . Because  $R^{k+1} = R^k \circ R$ , its matrix is  $\mathbf{M}_R \odot \mathbf{M}_{R^k}$ . By the inductive hypothesis this is  $\mathbf{M}_R \odot \mathbf{M}_R^{[k]} = \mathbf{M}_R^{[k+1]}$ .

### Section 9.4

1. a)  $\{(0, 0), (0, 1), (1, 1), (1, 2), (2, 0), (2, 2), (3, 0), (3, 3)\}$   
 b)  $\{(0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2), (3, 0)\}$  3.  $\{(a, b) \mid a \text{ divides } b \text{ or } b \text{ divides } a\}$



13. The symmetric closure of  $R$  is  $R \cup R^{-1}$ .  $\mathbf{M}_{R \cup R^{-1}} = \mathbf{M}_R \vee \mathbf{M}_{R^{-1}} = \mathbf{M}_R \vee \mathbf{M}_R^t$ . 15. Only when  $R$  is irreflexive,

in which case it is its own closure. **17.**  $a, a, a, a; a, b, e, a; a, d, e, a; b, c, c, b; b, e, a, b; c, b, c, c; c, c, b, c; c, c, c, c; d, e, a, d; d, e, e, d; e, a, b, e; e, a, d, e; e, d, e, e; e, e, d, e; e, e, e$  **19. a)**  $\{(1, 1), (1, 5), (2, 3), (3, 1), (3, 2), (3, 3), (3, 4), (4, 1), (4, 5), (5, 3), (5, 4)\}$  **b)**  $\{(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 5), (3, 1), (3, 3), (3, 4), (3, 5), (4, 1), (4, 2), (4, 3), (4, 4), (5, 1), (5, 3), (5, 5)\}$  **c)**  $\{(1, 1), (1, 3), (1, 4), (1, 5), (2, 1), (2, 2), (2, 3), (2, 4), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5)\}$  **d)**  $\{(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (2, 3), (2, 4), (2, 5), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5)\}$  **e)**  $\{(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5)\}$  **f)**  $\{(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (2, 2), (2, 3), (2, 4), (2, 5), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5), (5, 1), (5, 2), (5, 3), (5, 4), (5, 5)\}$  **21. a)** If there is a student  $c$  who shares a class with  $a$  and a class with  $b$  **b)** If there are two students  $c$  and  $d$  such that  $a$  and  $c$  share a class,  $c$  and  $d$  share a class, and  $d$  and  $b$  share a class **c)** If there is a sequence  $s_0, \dots, s_n$  of students with  $n \geq 1$  such that  $s_0 = a$ ,  $s_n = b$ , and for each  $i = 1, 2, \dots, n$ ,  $s_i$  and  $s_{i-1}$  share a class **23.** The result follows from  $(R^*)^{-1} = (\bigcup_{n=1}^{\infty} R^n)^{-1} = \bigcup_{n=1}^{\infty} (R^n)^{-1} = \bigcup_{n=1}^{\infty} R^n = R^*$ .

**25. a)**  $\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$  **b)**  $\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$

**c)**  $\begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$  **d)**  $\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$

**27.** Answers same as for Exercise 25. **29. a)**  $\{(1, 1), (1, 2), (1, 4), (2, 2), (3, 3), (4, 1), (4, 2), (4, 4)\}$  **b)**  $\{(1, 1), (1, 2), (1, 4), (2, 1), (2, 2), (2, 4), (3, 3), (4, 1), (4, 2), (4, 4)\}$  **c)**  $\{(1, 1), (1, 2), (1, 4), (2, 1), (2, 2), (2, 4), (3, 3), (4, 1), (4, 2), (4, 4)\}$  **31.** Algorithm 1:  $O(n^{3.8})$ ; Algorithm 2:  $O(n^3)$  **33.** Initialize with  $A := M_R \vee I_n$  and loop only for  $i := 2$  to  $n - 1$ . **35. a)** Because  $R$  is reflexive, every relation containing it must also be reflexive. **b)** Both  $\{(0, 0), (0, 1), (0, 2), (1, 1), (2, 2)\}$  and  $\{(0, 0), (0, 1), (1, 0), (1, 1), (2, 2)\}$  contain  $R$  and have an odd number of elements, but neither is a subset of the other.

## Section 9.5

- 1. a)** Equivalence relation **b)** Not reflexive, not transitive
- c)** Equivalence relation **d)** Not transitive **e)** Not symmetric, not transitive **3. a)** Equivalence relation **b)** Not transitive
- c)** Not reflexive, not symmetric, not transitive **d)** Equivalence relation **e)** Not reflexive, not transitive **5.** Many answers are possible. (1) Two buildings are equivalent if they were

opened during the same year; an equivalence class consists of the set of buildings opened in a given year (as long as there was at least one building opened that year). (2) Two buildings are equivalent if they have the same number of stories; the equivalence classes are the set of 1-story buildings, the set of 2-story buildings, and so on (one class for each  $n$  for which there is at least one  $n$ -story building). (3) Every building in which you have a class is equivalent to every building in which you have a class (including itself), and every building in which you don't have a class is equivalent to every building in which you don't have a class (including itself); there are two equivalence classes—the set of buildings in which you have a class and the set of buildings in which you don't have a class (assuming these are nonempty). **7.** The statement " $p$  is equivalent to  $q$ " means that  $p$  and  $q$  have the same entries in their truth tables.  $R$  is reflexive, because  $p$  has the same truth table as  $p$ .  $R$  is symmetric, for if  $p$  and  $q$  have the same truth table, then  $q$  and  $p$  have the same truth table. If  $p$  and  $q$  have the same entries in their truth tables and  $q$  and  $r$  have the same entries in their truth tables, then  $p$  and  $r$  also do, so  $R$  is transitive. The equivalence class of  $T$  is the set of all tautologies; the equivalence class of  $F$  is the set of all contradictions. **9. a)**  $(x, x) \in R$  because  $f(x) = f(x)$ . Hence,  $R$  is reflexive.  $(x, y) \in R$  if and only if  $f(x) = f(y)$ , which holds if and only if  $f(y) = f(x)$  if and only if  $(y, x) \in R$ . Hence,  $R$  is symmetric. If  $(x, y) \in R$  and  $(y, z) \in R$ , then  $f(x) = f(y)$  and  $f(y) = f(z)$ . Hence,  $f(x) = f(z)$ . Thus,  $(x, z) \in R$ . It follows that  $R$  is transitive. **b)** The sets  $f^{-1}(b)$  for  $b$  in the range of  $f$  **11.** Let  $x$  be a bit string of length 3 or more. Because  $x$  agrees with itself in the first three bits,  $(x, x) \in R$ . Hence,  $R$  is reflexive. Suppose that  $(x, y) \in R$ . Then  $x$  and  $y$  agree in the first three bits. Hence,  $y$  and  $x$  agree in the first three bits. Thus,  $(y, x) \in R$ . If  $(x, y)$  and  $(y, z)$  are in  $R$ , then  $x$  and  $y$  agree in the first three bits, as do  $y$  and  $z$ . Hence,  $x$  and  $z$  agree in the first three bits. Hence,  $(x, z) \in R$ . It follows that  $R$  is transitive. **13.** This follows from Exercise 9, where  $f$  is the function that takes a bit string of length 3 or more to the ordered pair with its first bit as the first component and the third bit as its second component. **15.** For reflexivity,  $((a, b), (a, b)) \in R$  because  $a + b = b + a$ . For symmetry, if  $((a, b), (c, d)) \in R$ , then  $a + d = b + c$ , so  $c + b = d + a$ , so  $((c, d), (a, b)) \in R$ . For transitivity, if  $((a, b), (c, d)) \in R$  and  $((c, d), (e, f)) \in R$ , then  $a + d = b + c$  and  $c + e = d + f$ , so  $a + d + c + e = b + c + d + f$ , so  $a + e = b + f$ , so  $((a, b), (e, f)) \in R$ . An easier solution is to note that by algebra, the given condition is the same as the condition that  $f((a, b)) = f((c, d))$ , where  $f((x, y)) = x - y$ ; therefore by Exercise 9 this is an equivalence relation. **17. a)** This follows from Exercise 9, where the function  $f$  from the set of differentiable functions (from  $\mathbf{R}$  to  $\mathbf{R}$ ) to the set of functions (from  $\mathbf{R}$  to  $\mathbf{R}$ ) is the differentiation operator. **b)** The set of all functions of the form  $g(x) = x^2 + C$  for some constant  $C$  **19.** This follows from Exercise 9, where the function  $f$  from the set of all URLs to the set of all Web pages is the function that assigns to each URL the Web page for that URL. **21.** No **23.** No **25.**  $R$  is reflexive because a bit string  $s$

has the same number of 1s as itself.  $R$  is symmetric because  $s$  and  $t$  having the same number of 1s implies that  $t$  and  $s$  do.  $R$  is transitive because  $s$  and  $t$  having the same number of 1s, and  $t$  and  $u$  having the same number of 1s implies that  $s$  and  $u$  have the same number of 1s. **27.** a) The sets of people of the same age b) The sets of people with the same two parents **29.** The set of all bit strings with exactly two 1s. **31.** a) The set of all bit strings of length 3 b) The set of all bit strings of length 4 that end with a 1 c) The set of all bit strings of length 5 that end 11 d) The set of all bit strings of length 8 that end 10101 **33.** Each of the 15 bit strings of length less than four is in an equivalence class by itself:  $[\lambda]_{R_4} = \{\lambda\}$ ,  $[0]_{R_4} = \{0\}$ ,  $[1]_{R_4} = \{1\}$ ,  $[00]_{R_4} = \{00\}$ ,  $[01]_{R_4} = \{01\}$ , ...,  $[111]_{R_4} = \{111\}$ . The remaining 16 equivalence classes are determined by the bit strings of length 4:  $[0000]_{R_4} = \{0000, 00000, 00001, 000000, 000001, 000010, 000011, 0000000, \dots\}$ ,  $[0001]_{R_4} = \{0001, 00010, 00011, 000100, 000101, 000110, 000111, 0001000, \dots\}$ , ...,  $[1111]_{R_4} = \{1111, 11110, 11111, 111100, 111101, 111110, 111111, 1111000, \dots\}$  **35.** a)  $[2]_5 = \{i \mid i \equiv 2 \pmod{5}\} = \{\dots, -8, -3, 2, 7, 12, \dots\}$  b)  $[3]_5 = \{i \mid i \equiv 3 \pmod{5}\} = \{\dots, -7, -2, 3, 8, 13, \dots\}$  c)  $[6]_5 = \{i \mid i \equiv 6 \pmod{5}\} = \{\dots, -9, -4, 1, 6, 11, \dots\}$  d)  $[-3]_5 = \{i \mid i \equiv -3 \pmod{5}\} = \{\dots, -8, -3, 2, 7, 12, \dots\}$  **37.**  $\{6n+k \mid n \in \mathbb{Z}\}$  for  $k \in \{0, 1, 2, 3, 4, 5\}$  **39.** a)  $[(1, 2)] = \{(a, b) \mid a - b = -1\} = \{(1, 2), (3, 4), (4, 5), (5, 6), \dots\}$  b) Each equivalence class can be interpreted as an integer (negative, positive, or zero); specifically,  $[(a, b)]$  can be interpreted as  $a - b$ . **41.** a) No b) Yes c) Yes d) No **43.** a), c), e) **45.** b), d), e) **47.** a)  $\{(0, 0), (1, 1), (1, 2), (2, 1), (2, 2), (3, 3), (3, 4), (3, 5), (4, 3), (4, 4), (4, 5), (5, 3), (5, 4), (5, 5)\}$  b)  $\{(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1), (2, 2), (3, 2), (3, 3), (3, 4), (4, 4), (4, 5), (5, 4), (5, 5)\}$  c)  $\{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2), (3, 3), (3, 4), (3, 5), (4, 3), (4, 4), (4, 5), (5, 3), (5, 4), (5, 5)\}$  d)  $\{(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5)\}$  **49.**  $[0]_6 \subseteq [0]_3$ ,  $[1]_6 \subseteq [1]_3$ ,  $[2]_6 \subseteq [2]_3$ ,  $[3]_6 \subseteq [0]_3$ ,  $[4]_6 \subseteq [1]_3$ ,  $[5]_6 \subseteq [2]_3$  **51.** Let  $A$  be a set in the first partition. Pick a particular element  $x$  of  $A$ . The set of all bit strings of length 16 that agree with  $x$  on the last four bits is one of the sets in the second partition, and clearly every string in  $A$  is in that set. **53.** We claim that each equivalence class  $[x]_{R_{31}}$  is a subset of the equivalence class  $[x]_{R_8}$ . To show this, choose an arbitrary element  $y \in [x]_{R_{31}}$ . Then  $y$  is equivalent to  $x$  under  $R_{31}$ , so either  $y = x$  or  $y$  and  $x$  are each at least 31 characters long and agree on their first 31 characters. Because strings that are at least 31 characters long and agree on their first 31 characters must be at least 8 characters long and agree on their first 8 characters, we know that either  $y = x$  or  $y$  and  $x$  are each at least 8 characters long and agree on their first 8 characters. This means that  $y$  is equivalent to  $x$  under  $R_8$ , so  $y \in [x]_{R_8}$ . **55.**  $\{(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, a), (c, b), (c, c), (d, d), (d, e), (e, d), (e, e)\}$  **57.** a)  $\mathbb{Z}$  b)  $\{n + \frac{1}{2} \mid n \in \mathbb{Z}\}$  **59.** a)  $R$  is reflexive because any coloring can be obtained from itself via a 360-degree rotation. To see that  $R$  is symmetric and transitive, use the fact that each rotation is the composition of

two reflections and conversely the composition of two reflections is a rotation. Hence,  $(C_1, C_2)$  belongs to  $R$  if and only if  $C_2$  can be obtained from  $C_1$  by a composition of reflections. So if  $(C_1, C_2)$  belongs to  $R$ , so does  $(C_2, C_1)$  because the inverse of the composition of reflections is also a composition of reflections (in the opposite order). Hence,  $R$  is symmetric. To see that  $R$  is transitive, suppose  $(C_1, C_2)$  and  $(C_2, C_3)$  belong to  $R$ . Taking the composition of the reflections in each case yields a composition of reflections, showing that  $(C_1, C_3)$  belongs to  $R$ . b) We express colorings with sequences of length four, with  $r$  and  $b$  denoting red and blue, respectively. We list letters denoting the colors of the upper left square, upper right square, lower left square, and lower right square, in that order. The equivalence classes are:  $\{rrrr\}$ ,  $\{bbbb\}$ ,  $\{rrrb, rrbr, rbrr, brrr\}$ ,  $\{bbbr, bbrb, brbb, rbbb\}$ ,  $\{rbbr, brrb\}$ ,  $\{rrbb, brbr, bbrr, rbrb\}$ . **61.** 5 **63.** Yes **65.**  $R$  **67.** First form the reflexive closure of  $R$ , then form the symmetric closure of the reflexive closure, and finally form the transitive closure of the symmetric closure of the reflexive closure. **69.**  $p(0) = 1$ ,  $p(1) = 1$ ,  $p(2) = 2$ ,  $p(3) = 5$ ,  $p(4) = 15$ ,  $p(5) = 52$ ,  $p(6) = 203$ ,  $p(7) = 877$ ,  $p(8) = 4140$ ,  $p(9) = 21147$ ,  $p(10) = 115975$

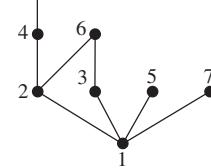
## Section 9.6

- 1.** a) Is a partial ordering b) Not antisymmetric, not transitive  
 c) Is a partial ordering d) Is a partial ordering e) Not anti-symmetric, not transitive **3.** a) No b) No c) Yes **5.** a) Yes b) No c) Yes d) No **7.** a) No b) Yes c) No **9.** No **11.** Yes **13.** a)  $\{(0, 0), (1, 0), (1, 1), (2, 0), (2, 1), (2, 2)\}$  b)  $(\mathbb{Z}, \leq)$  c)  $(P(\mathbb{Z}), \subseteq)$  d)  $(\mathbb{Z}^+, \text{"is a multiple of"})$  **15.** a)  $\{0\}$  and  $\{1\}$ , for instance **16.** a) 4 and 6, for instance **17.** a)  $(1, 1, 2) < (1, 2, 1)$  b)  $(0, 1, 2, 3) < (0, 1, 3, 2)$  c)  $(0, 1, 1, 0) < (1, 0, 1, 0, 1)$  **19.**  $0 < 0001 < 001 < 01 < 010 < 0101 < 011 < 11$

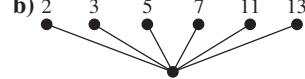
**21.** 15

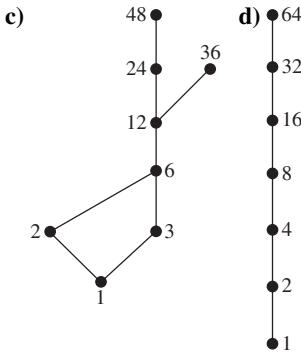


**23.** a) 8



b)





- 25.** (a, b), (a, c), (a, d), (b, c), (b, d), (a, a), (b, b), (c, c), (d, d)   **27.** (a, a), (a, g), (a, d), (a, e), (a, f), (b, b), (b, g), (b, d), (b, e), (b, f), (c, c), (c, g), (c, d), (c, e), (c, f), (g, d), (g, e), (g, f), (g, g), (d, d), (e, e), (f, f)   **29.** ( $\emptyset$ , {a}), ( $\emptyset$ , {b}), ( $\emptyset$ , {c}), ({a}, {a, b}), ({a}, {a, c}), ({b}, {a, b}), ({b}, {b, c}), ({c}, {a, c}), ({c}, {b, c}), ({a, b}, {a, b, c}), ({a, c}, {a, b, c})({b, c}, {a, b, c})   **31.** Let  $(S, \preccurlyeq)$  be a finite poset. We will show that this poset is the reflexive transitive closure of its covering relation. Suppose that  $(a, b)$  is in the reflexive transitive closure of the covering relation. Then  $a = b$  or  $a \prec b$ , so  $a \preccurlyeq b$ , or else there is a sequence  $a_1, a_2, \dots, a_n$  such that  $a \prec a_1 \prec a_2 \prec \dots \prec a_n \prec b$ , in which case again  $a \preccurlyeq b$  by the transitivity of  $\preccurlyeq$ . Conversely, suppose that  $a \prec b$ . If  $a = b$  then  $(a, b)$  is in the reflexive transitive closure of the covering relation. If  $a \prec b$  and there is no  $z$  such that  $a \prec z \prec b$ , then  $(a, b)$  is in the covering relation and therefore in its reflexive transitive closure. Otherwise, let  $a \prec a_1 \prec a_2 \prec \dots \prec a_n \prec b$  be a longest possible sequence of this form (which exists because the poset is finite). Then no intermediate elements can be inserted, so each pair  $(a, a_1), (a_1, a_2), \dots, (a_n, b)$  is in the covering relation, so again  $(a, b)$  is in its reflexive transitive closure.   **33. a)** 24, 45   **b)** 3, 5   **c)** No   **d)** No   **e)** 15, 45   **f)** 15   **g)** 15, 5, 3   **h)** 15   **35. a)** {1, 2}, {1, 3, 4}, {2, 3, 4}   **b)** {1}, {2}, {4}   **c)** No   **d)** No   **e)** {2, 4}, {2, 3, 4}   **f)** {2, 4}   **g)** {3, 4}, {4}   **h)** {3, 4}   **37.** Because  $(a, b) \preccurlyeq (a, b)$ ,  $\preccurlyeq$  is reflexive. If  $(a_1, a_2) \preccurlyeq (b_1, b_2)$  and  $(a_1, a_2) \neq (b_1, b_2)$ , either  $a_1 \prec b_1$ , or  $a_1 = b_1$  and  $a_2 \prec b_2$ . In either case,  $(b_1, b_2)$  is not less than or equal to  $(a_1, a_2)$ . Hence,  $\preccurlyeq$  is antisymmetric. Suppose that  $(a_1, a_2) \prec (b_1, b_2) \prec (c_1, c_2)$ . Then if  $a_1 \prec b_1$  or  $b_1 \prec c_1$ , we have  $a_1 \prec c_1$ , so  $(a_1, a_2) \prec (c_1, c_2)$ , but if  $a_1 = b_1 = c_1$ , then  $a_2 \prec b_2 \prec c_2$ , which implies that  $(a_1, a_2) \prec (c_1, c_2)$ . Hence,  $\preccurlyeq$  is transitive.   **39.** Because  $(s, t) \preccurlyeq (s, t)$ ,  $\preccurlyeq$  is reflexive. If  $(s, t) \preccurlyeq (u, v)$  and  $(u, v) \preccurlyeq (s, t)$ , then  $s \preccurlyeq u \preccurlyeq s$  and  $t \preccurlyeq v \preccurlyeq t$ ; hence,  $s = u$  and  $t = v$ . Hence,  $\preccurlyeq$  is antisymmetric. Suppose that  $(s, t) \preccurlyeq (u, v) \preccurlyeq (w, x)$ . Then  $s \preccurlyeq u$ ,  $t \preccurlyeq v$ ,  $u \preccurlyeq w$ , and  $v \preccurlyeq x$ . It follows that  $s \preccurlyeq w$  and  $t \preccurlyeq x$ . Hence,  $(s, t) \preccurlyeq (w, x)$ . Hence,  $\preccurlyeq$  is transitive.   **41. a)** Suppose that  $x$  is maximal and that  $y$  is the largest element. Then  $x \preccurlyeq y$ . Because  $x$  is not less than  $y$ , it follows that  $x = y$ . By Exercise 40(a)  $y$  is unique. Hence,  $x$  is unique.   **b)** Suppose that  $x$  is minimal and that  $y$  is the smallest element. Then  $x \succcurlyeq y$ . Because  $x$  is not greater than  $y$ , it

follows that  $x = y$ . By Exercise 40(b)  $y$  is unique. Hence,  $x$  is unique.   **43. a)** Yes   **b)** No   **c)** Yes   **45.** Use mathematical induction. Let  $P(n)$  be “Every subset with  $n$  elements from a lattice has a least upper bound and a greatest lower bound.” *Basis step:*  $P(1)$  is true because the least upper bound and greatest lower bound of  $\{x\}$  are both  $x$ . *Inductive step:* Assume that  $P(k)$  is true. Let  $S$  be a set with  $k + 1$  elements. Let  $x \in S$  and  $S' = S - \{x\}$ . Because  $S'$  has  $k$  elements, by the inductive hypothesis, it has a least upper bound  $y$  and a greatest lower bound  $a$ . Now because we are in a lattice, there are elements  $z = \text{lub}(x, y)$  and  $b = \text{glb}(x, a)$ . We are done if we can show that  $z$  is the least upper bound of  $S$  and  $b$  is the greatest lower bound of  $S$ . To show that  $z$  is the least upper bound of  $S$ , first note that if  $w \in S$ , then  $w = x$  or  $w \in S'$ . If  $w = x$  then  $w \preccurlyeq z$  because  $z$  is the least upper bound of  $x$  and  $y$ . If  $w \in S'$ , then  $w \preccurlyeq z$  because  $w \preccurlyeq y$ , which is true because  $y$  is the least upper bound of  $S'$ , and  $y \preccurlyeq z$ , which is true because  $z = \text{lub}(x, y)$ . To see that  $z$  is the least upper bound of  $S$ , suppose that  $u$  is an upper bound of  $S$ . Note that such an element  $u$  must be an upper bound of  $x$  and  $y$ , but because  $z = \text{lub}(x, y)$ , it follows that  $z \preccurlyeq u$ . We omit the similar argument that  $b$  is the greatest lower bound of  $S$ .   **47. a)** No   **b)** Yes   **c)** (Proprietary, {Cheetah, Puma}), (Restricted, {Cheetah, Puma}), (Registered, {Cheetah, Puma, Impala}), (Restricted, {Cheetah, Puma, Impala}), (Registered, {Cheetah, Puma, Impala})   **d)** (Non-proprietary, {Impala, Puma}), (Proprietary, {Impala, Puma}), (Restricted, {Impala, Puma}), (Nonproprietary, {Impala}), (Proprietary, {Impala}), (Restricted, {Impala}), (Nonproprietary, {Puma}), (Proprietary, {Puma}), (Restricted, {Puma}), (Nonproprietary,  $\emptyset$ ), (Proprietary,  $\emptyset$ ), (Restricted,  $\emptyset$ )   **49.** Let  $\Pi$  be the set of all partitions of a set  $S$  with  $P_1 \preccurlyeq P_2$  if  $P_1$  is a refinement of  $P_2$ , that is, if every set in  $P_1$  is a subset of a set in  $P_2$ . First, we show that  $(\Pi, \preccurlyeq)$  is a poset. Because  $P \preccurlyeq P$  for every partition  $P$ ,  $\preccurlyeq$  is reflexive. Now suppose that  $P_1 \preccurlyeq P_2$  and  $P_2 \preccurlyeq P_1$ . Let  $T \in P_1$ . Because  $P_1 \preccurlyeq P_2$ , there is a set  $T' \in P_2$  such that  $T \subseteq T'$ . Because  $P_2 \preccurlyeq P_1$  there is a set  $T'' \in P_1$  such that  $T' \subseteq T''$ . It follows that  $T \subseteq T''$ . But because  $P_1$  is a partition,  $T = T''$ , which implies that  $T = T'$  because  $T \subseteq T' \subseteq T''$ . Thus,  $T \in P_2$ . By reversing the roles of  $P_1$  and  $P_2$  it follows that every set in  $P_2$  is also in  $P_1$ . Hence,  $P_1 = P_2$  and  $\preccurlyeq$  is antisymmetric. Next, suppose that  $P_1 \preccurlyeq P_2$  and  $P_2 \preccurlyeq P_3$ . Let  $T \in P_1$ . Then there is a set  $T' \in P_2$  such that  $T \subseteq T'$ . Because  $P_2 \preccurlyeq P_3$  there is a set  $T'' \in P_3$  such that  $T' \subseteq T''$ . This means that  $T \subseteq T''$ . Hence,  $P_1 \preccurlyeq P_3$ . It follows that  $\preccurlyeq$  is transitive. The greatest lower bound of the partitions  $P_1$  and  $P_2$  is the partition  $P$  whose subsets are the nonempty sets of the form  $T_1 \cap T_2$  where  $T_1 \in P_1$  and  $T_2 \in P_2$ . We omit the justification of this statement here. The least upper bound of the partitions  $P_1$  and  $P_2$  is the partition that corresponds to the equivalence relation in which  $x \in S$  is related to  $y \in S$  if there is a sequence  $x = x_0, x_1, x_2, \dots, x_n = y$  for some nonnegative integer  $n$  such that for each  $i$  from 1 to  $n$ ,  $x_{i-1}$  and  $x_i$  are in the same element of  $P_1$  or of  $P_2$ . We omit the details that this is an equivalence relation and the details of the proof that this is

the least upper bound of the two partitions. **51.** By Exercise 45 there is a least upper bound and a greatest lower bound for the entire finite lattice. By definition these elements are the greatest and least elements, respectively. **53.** The least element of a subset of  $\mathbf{Z}^+ \times \mathbf{Z}^+$  is that pair that has the smallest possible first coordinate, and, if there is more than one such pair, that pair among those that has the smallest second coordinate. **55.** If  $x$  is an integer in a decreasing sequence of elements of this poset, then at most  $|x|$  elements can follow  $x$  in the sequence, namely, integers whose absolute values are  $|x| - 1, |x| - 2, \dots, 1, 0$ . Therefore there can be no infinite decreasing sequence. This is not a totally ordered set, because 5 and -5, for example, are incomparable. **57.** To find which of two rational numbers is larger, write them with a positive common denominator and compare numerators. To show that this set is dense, suppose that  $x < y$  are two rational numbers. Then their average, i.e.,  $(x + y)/2$ , is a rational number between them. **59.** Let  $(S, \preccurlyeq)$  be a partially ordered set. It is enough to show that every nonempty subset of  $S$  contains a least element if and only if there is no infinite decreasing sequence of elements  $a_1, a_2, a_3, \dots$  in  $S$  (i.e., where  $a_{i+1} \prec a_i$  for all  $i$ ). An infinite decreasing sequence of elements clearly has no least element. Conversely, let  $A$  be any nonempty subset of  $S$  that has no least element. Because  $A$  is nonempty, choose  $a_1 \in A$ . Because  $a_1$  is not the least element of  $A$ , choose  $a_2 \in A$  with  $a_2 \prec a_1$ . Because  $a_2$  is not the least element of  $A$ , choose  $a_3 \in A$  with  $a_3 \prec a_2$ . Continue in this manner, producing an infinite decreasing sequence in  $S$ .

**61.**  $a \prec_t b \prec_t c \prec_t d \prec_t e \prec_t f \prec_t g \prec_t h \prec_t i \prec_t j \prec_t k \prec_t l \prec_t m$     **63.**  $1 \prec 5 \prec 2 \prec 4 \prec 12 \prec 20, 1 \prec 2 \prec 5 \prec 4 \prec 12 \prec 20, 1 \prec 2 \prec 4 \prec 5 \prec 12 \prec 20, 1 \prec 2 \prec 4 \prec 12 \prec 5 \prec 20, 1 \prec 5 \prec 2 \prec 4 \prec 20 \prec 12, 1 \prec 2 \prec 5 \prec 4 \prec 20 \prec 12, 1 \prec 2 \prec 4 \prec 5 \prec 20 \prec 12$

**65.**  $A \prec C \prec E \prec B \prec D \prec F \prec G, A \prec E \prec C \prec B \prec D \prec F \prec G, C \prec A \prec E \prec B \prec D \prec F \prec G, C \prec E \prec A \prec B \prec D \prec F \prec G, E \prec A \prec C \prec B \prec D \prec F \prec G, E \prec C \prec A \prec B \prec D \prec F \prec G, A \prec C \prec B \prec E \prec D \prec F \prec G, C \prec A \prec B \prec E \prec D \prec F \prec G, A \prec C \prec B \prec D \prec E \prec F \prec G, C \prec A \prec B \prec D \prec E \prec F \prec G, A \prec C \prec E \prec B \prec F \prec D \prec G, C \prec A \prec E \prec B \prec F \prec D \prec G, C \prec E \prec A \prec B \prec F \prec D \prec G, E \prec A \prec C \prec B \prec F \prec D \prec G, E \prec C \prec A \prec B \prec F \prec D \prec G, A \prec C \prec B \prec E \prec F \prec D \prec G, C \prec A \prec B \prec E \prec F \prec D \prec G$     **67.** Determine user needs  $\prec$  Write functional requirements  $\prec$  Set up test sites  $\prec$  Develop system requirements  $\prec$  Write documentation  $\prec$  Develop module  $A \prec$  Develop module  $B \prec$  Develop module  $C \prec$  Integrate modules  $\prec \alpha$  test  $\prec \beta$  test  $\prec$  Completion

transitive    **3.**  $((a, b), (a, b)) \in R$  because  $a + b = a + b$ . Hence,  $R$  is reflexive. If  $((a, b), (c, d)) \in R$  then  $a + d = b + c$ , so that  $c + b = d + a$ . It follows that  $((c, d), (a, b)) \in R$ . Hence,  $R$  is symmetric. Suppose that  $((a, b), (c, d))$  and  $((c, d), (e, f))$  belong to  $R$ . Then  $a + d = b + c$  and  $c + f = d + e$ . Adding these two equations and subtracting  $c + d$  from both sides gives  $a + f = b + e$ . Hence,  $((a, b), (e, f))$  belongs to  $R$ . Hence,  $R$  is transitive.

**5.** Suppose that  $(a, b) \in R$ . Because  $(b, b) \in R$  it follows that  $(a, b) \in R^2$ .    **7.** Yes, yes    **9.** Yes, yes    **11.** Two records with identical keys in the projection would have identical keys in the original.    **13.**  $(\Delta \cup R)^{-1} = \Delta^{-1} \cup R^{-1} = \Delta \cup R^{-1}$

**15. a)**  $R = \{(a, b), (a, c)\}$ . The transitive closure of the symmetric closure of  $R$  is  $\{(a, a), (a, b), (a, c), (b, a), (b, b), (b, c), (c, a), (c, b), (c, c)\}$  and is different from the symmetric closure of the transitive closure of  $R$ , which is  $\{(a, b), (a, c), (b, a), (c, a)\}$ . **b)** Suppose that  $(a, b)$  is in the symmetric closure of the transitive closure of  $R$ . We must show that  $(a, b)$  is in the transitive closure of the symmetric closure of  $R$ . We know that at least one of  $(a, b)$  and  $(b, a)$  is in the transitive closure of  $R$ . Hence, there is either a path from  $a$  to  $b$  in  $R$  or a path from  $b$  to  $a$  in  $R$  (or both). In the former case, there is a path from  $a$  to  $b$  in the symmetric closure of  $R$ . In the latter case, we can form a path from  $a$  to  $b$  in the symmetric closure of  $R$  by reversing the directions of all the edges in a path from  $b$  to  $a$ , going backward. Hence,  $(a, b)$  is in the transitive closure of the symmetric closure of  $R$ .

**17.** The closure of  $S$  with respect to property **P** is a relation with property **P** that contains  $R$  because  $R \subseteq S$ . Hence, the closure of  $S$  with respect to property **P** contains the closure of  $R$  with respect to property **P**.    **19.** Use the basic idea of Warshall's algorithm, except let  $w_{ij}^{[k]}$  equal the length of the longest path from  $v_i$  to  $v_j$  using interior vertices with subscripts not exceeding  $k$ , and equal to -1 if there is no such path. To find  $w_{ij}^{[k]}$  from the entries of  $\mathbf{W}_{k-1}$ , determine for each pair  $(i, j)$  whether there are paths from  $v_i$  to  $v_k$  and from  $v_k$  to  $v_j$  using no vertices labeled greater than  $k$ . If either  $w_{ik}^{[k-1]}$  or  $w_{kj}^{[k-1]}$  is -1, then such a pair of paths does not exist, so set  $w_{ij}^{[k]} = w_{ij}^{[k-1]}$ . If such a pair of paths exists, then there are two possibilities. If  $w_{kk}^{[k-1]} > 0$ , there are paths of arbitrary long length from  $v_i$  to  $v_j$ , so set  $w_{ij}^{[k]} = \infty$ . If  $w_{kk}^{[k-1]} = 0$ , set  $w_{ij}^{[k-1]} = \max(w_{ij}^{[k-1]}, w_{ik}^{[k-1]} + w_{kj}^{[k-1]})$ . (Initially take  $\mathbf{W}_0 = \mathbf{M}_R$ .)

**21. 25**    **23.** Because  $A_i \cap B_j$  is a subset of  $A_i$  and of  $B_j$ , the collection of subsets is a refinement of each of the given partitions. We must show that it is a partition. By construction, each of these sets is nonempty. To see that their union is  $S$ , suppose that  $s \in S$ . Because  $P_1$  and  $P_2$  are partitions of  $S$ , there are sets  $A_i$  and  $B_j$  such that  $s \in A_i$  and  $s \in B_j$ . Therefore  $s \in A_i \cap B_j$ . Hence, the union of these sets is  $S$ . To see that they are pairwise disjoint, note that unless  $i = i'$  and  $j = j'$ ,  $(A_i \cap B_j) \cap (A_{i'} \cap B_{j'}) = (A_i \cap A_{i'}) \cap (B_j \cap B_{j'}) = \emptyset$ .

**25.** The subset relation is a partial ordering on any collection of sets, because it is reflexive, antisymmetric, and transitive. Here the collection of sets is  $\mathbf{R}(S)$ .    **27.** Find recipe  $\prec$  Buy

## Supplementary Exercises

- 1. a)** Irreflexive (we do not include the empty string), symmetric    **b)** Irreflexive, symmetric    **c)** Irreflexive, antisymmetric,

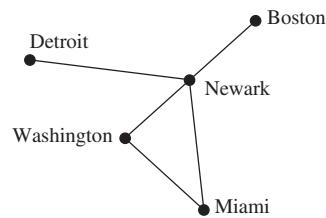
seafood  $\prec$  Buy groceries  $\prec$  Wash shellfish  $\prec$  Cut ginger and garlic  $\prec$  Clean fish  $\prec$  Steam rice  $\prec$  Cut fish  $\prec$  Wash vegetables  $\prec$  Chop water chestnuts  $\prec$  Make garnishes  $\prec$  Cook in wok  $\prec$  Arrange on platter  $\prec$  Serve **29.** a) The only antichain with more than one element is  $\{c, d\}$ . b) The only antichains with more than one element are  $\{b, c\}$ ,  $\{c, e\}$ , and  $\{d, e\}$ . c) The only antichains with more than one element are  $\{a, b\}$ ,  $\{a, c\}$ ,  $\{b, c\}$ ,  $\{a, b, c\}$ ,  $\{d, e\}$ ,  $\{d, f\}$ ,  $\{e, f\}$ , and  $\{d, e, f\}$ . **31.** Let  $(S, \preccurlyeq)$  be a finite poset, and let  $A$  be a maximal chain. Because  $(A, \preccurlyeq)$  is also a poset it must have a minimal element  $m$ . Suppose that  $m$  is not minimal in  $S$ . Then there would be an element  $a$  of  $S$  with  $a \prec m$ . However, this would make the set  $A \cup \{a\}$  a larger chain than  $A$ . To show this, we must show that  $a$  is comparable with every element of  $A$ . Because  $m$  is comparable with every element of  $A$  and  $m$  is minimal, it follows that  $m \prec x$  when  $x$  is in  $A$  and  $x \neq m$ . Because  $a \prec m$  and  $m \prec x$ , the transitive law shows that  $a \prec x$  for every element of  $A$ . **33.** Let  $aRb$  denote that  $a$  is a descendant of  $b$ . By Exercise 32, if no set of  $n+1$  people none of whom is a descendant of any other (an antichain) exists, then  $k \leq n$ , so the set can be partitioned into  $k \leq n$  chains. By the pigeonhole principle, at least one of these chains contains at least  $m+1$  people. **35.** We prove by contradiction that if  $S$  has no infinite decreasing sequence and  $\forall x (\{\forall y [y \prec x \rightarrow P(y)]\} \rightarrow P(x))$ , then  $P(x)$  is true for all  $x \in S$ . If it does not hold that  $P(x)$  is true for all  $x \in S$ , let  $x_1$  be an element of  $S$  such that  $P(x_1)$  is not true. Then by the conditional statement already given, it must be the case that  $\forall y [y \prec x_1 \rightarrow P(y)]$  is not true. This means that there is some  $x_2$  with  $x_2 \prec x_1$  such that  $P(x_2)$  is not true. Again invoking the conditional statement, we get an  $x_3 \prec x_2$  such that  $P(x_3)$  is not true, and so on forever. This contradicts the well-foundedness of our poset. Therefore,  $P(x)$  is true for all  $x \in S$ . **37.** Suppose that  $R$  is a quasi-ordering. Because  $R$  is reflexive, if  $a \in A$ , then  $(a, a) \in R$ . This implies that  $(a, a) \in R^{-1}$ . Hence,  $a \in R \cap R^{-1}$ . It follows that  $R \cap R^{-1}$  is reflexive.  $R \cap R^{-1}$  is symmetric for any relation  $R$  because, for any relation  $R$ , if  $(a, b) \in R$  then  $(b, a) \in R^{-1}$  and vice versa. To show that  $R \cap R^{-1}$  is transitive, suppose that  $(a, b) \in R \cap R^{-1}$  and  $(b, c) \in R \cap R^{-1}$ . Because  $(a, b) \in R$  and  $(b, c) \in R$ ,  $(a, c) \in R$ , because  $R$  is transitive. Similarly, because  $(a, b) \in R^{-1}$  and  $(b, c) \in R^{-1}$ ,  $(b, a) \in R$  and  $(c, b) \in R$ , so  $(c, a) \in R$  and  $(a, c) \in R^{-1}$ . Hence,  $(a, c) \in R \cap R^{-1}$ . It follows that  $R \cap R^{-1}$  is an equivalence relation. **39.** a) Because  $\text{glb}(x, y) = \text{glb}(y, x)$  and  $\text{lub}(x, y) = \text{lub}(y, x)$ , it follows that  $x \wedge y = y \wedge x$  and  $x \vee y = y \vee x$ . b) Using the definition,  $(x \wedge y) \wedge z$  is a lower bound of  $x$ ,  $y$ , and  $z$  that is greater than every other lower bound. Because  $x$ ,  $y$ , and  $z$  play interchangeable roles,  $x \wedge (y \wedge z)$  is the same element. Similarly,  $(x \vee y) \vee z$  is an upper bound of  $x$ ,  $y$ , and  $z$  that is less than every other upper bound. Because  $x$ ,  $y$ , and  $z$  play interchangeable roles,  $x \vee (y \vee z)$  is the same element. c) To show that  $x \wedge (x \vee y) = x$  it is sufficient to show that  $x$  is the greatest lower bound of  $x$ , and  $x \vee y$ . Note that  $x$  is a lower bound of  $x$ , and because  $x \vee y$  is by definition greater than  $x$ ,  $x$  is a lower bound for it as well. Therefore,  $x$  is a lower bound. But any lower

bound of  $x$  has to be less than  $x$ , so  $x$  is the greatest lower bound. The second statement is the dual of the first; we omit its proof. d)  $x$  is a lower, and an upper, bound for itself and itself, and the greatest, and least, such bound. **41.** a) Because 1 is the only element greater than or equal to 1, it is the only upper bound for 1 and therefore the only possible value of the least upper bound of  $x$  and 1. b) Because  $x \preccurlyeq 1$ ,  $x$  is a lower bound for both  $x$  and 1 and no other lower bound can be greater than  $x$ , so  $x \wedge 1 = x$ . c) Because 0  $\preccurlyeq x$ ,  $x$  is an upper bound for both  $x$  and 0 and no other bound can be less than  $x$ , so  $x \vee 0 = x$ . d) Because 0 is the only element less than or equal to 0, it is the only lower bound for 0 and therefore the only possible value of the greatest lower bound of  $x$  and 0. **43.**  $L = (S, \subseteq)$  where  $S = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$  **45.** Yes **47.** The complement of a subset  $X \subseteq S$  is its complement  $S - X$ . To prove this, note that  $X \vee (S - X) = 1$  and  $X \wedge (S - X) = 0$  because  $X \cup (S - X) = S$  and  $X \cap (S - X) = \emptyset$ . **49.** Think of the rectangular grid as representing elements in a matrix. Thus we number from top to bottom and within that from left to right. The partial order is that  $(a, b) \preceq (c, d)$  iff  $a \leq c$  and  $b \leq d$ . Note that  $(1, 1)$  is the least element under this relation. The rules for Chomp as explained in Chapter 1 coincide with the rules stated in the preamble here. But now we can identify the point  $(a, b)$  with the natural number  $p^{a-1}q^{b-1}$  for all  $a$  and  $b$  with  $1 \leq a \leq m$  and  $1 \leq b \leq n$ . This identifies the points in the rectangular grid with the set  $S$  in this exercise, and the partial order  $\preceq$  just described is the same as the divides relation, because  $p^{a-1}q^{b-1} \mid p^{c-1}q^{d-1}$  if and only if the exponent of  $p$  on the left does not exceed the exponent of  $p$  on the right, and similarly for  $q$ .

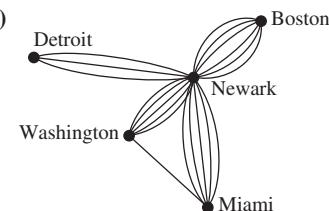
## CHAPTER 10

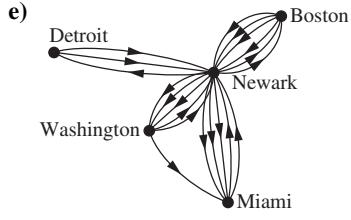
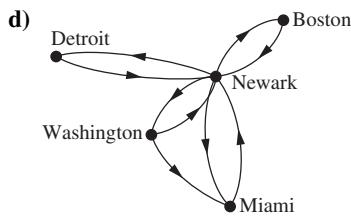
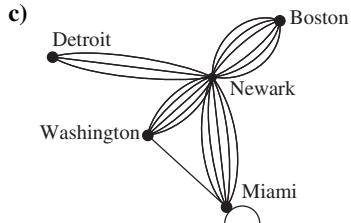
### Section 10.1

**1. a)**

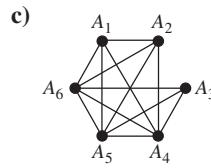
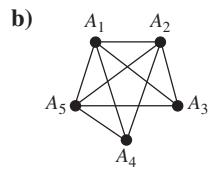
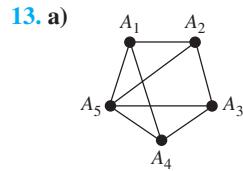


**b)**

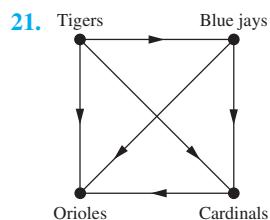
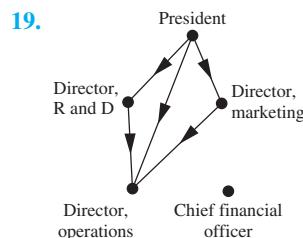
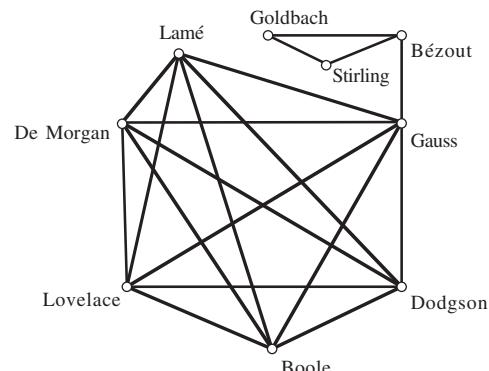
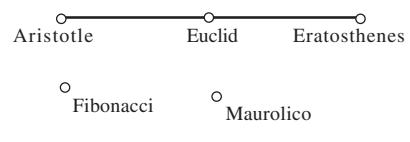
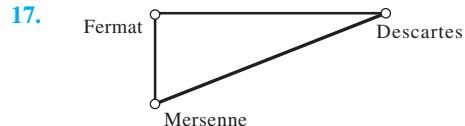




3. Simple graph    5. Pseudograph    7. Directed graph  
 9. Directed multigraph    11. If  $uRv$ , then there is an edge associated with  $\{u, v\}$ . But  $\{u, v\} = \{v, u\}$ , so this edge is associated with  $\{v, u\}$  and therefore  $vRu$ . Thus, by definition,  $R$  is a symmetric relation. A simple graph does not allow loops; therefore,  $uRu$  never holds, and so by definition  $R$  is irreflexive.



- 15.
- |                                                   |                              |
|---------------------------------------------------|------------------------------|
| Hermit thrush<br>Robin<br>Mockingbird<br>Blue jay | Hairy woodpecker<br>Nuthatch |
|---------------------------------------------------|------------------------------|
- 



23. We find the telephone numbers in the call graph for February that are not present in the call graph for January and vice versa. For each number we find, we make a list of the numbers they called or were called by using the edges in the call graph. We examine these lists to find new telephone numbers in February that had similar calling patterns to defunct telephone numbers in January.    25. We use the graph model that has e-mail addresses as vertices and for each message sent, an edge from the e-mail address of the sender to the e-mail address of the recipient. For each e-mail address, we can make a list of other addresses they sent messages to and a list of other addresses from which they received messages. If two e-mail addresses had almost the same pattern, we conclude that these addresses might have belonged to the same

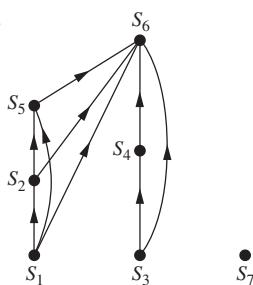
person who had recently changed his or her e-mail address.

**27.** Let  $V$  be the set of people at the party. Let  $E$  be the set of ordered pairs  $(u, v)$  in  $V \times V$  such that  $u$  knows  $v$ 's name. The edges are directed, but multiple edges are not allowed. Literally, there is a loop at each vertex, but for simplicity, the model could omit the loops.

**29.** Vertices are the courses; edges are directed; edge  $uv$  means that course  $u$  is prerequisite for course  $v$ ; courses without prerequisites are vertices with in-degree 0; courses that are not prerequisite for any other courses are vertices with out-degree 0.

**31.** Let the set of vertices be a set of people, and two vertices are joined by an edge if the two people were ever married. Ignoring complications, this graph has the property that there are two types of vertices (men and women), and every edge joins vertices of opposite types.

**33.**

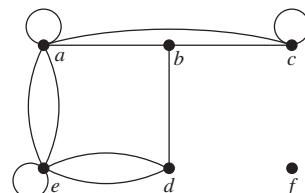


**35.** Represent people in the group by vertices. Put a directed edge into the graph for every pair of vertices. Label the edge from the vertex representing  $A$  to the vertex representing  $B$  with a + (plus) if  $A$  likes  $B$ , a - (minus) if  $A$  dislikes  $B$ , and a 0 if  $A$  is neutral about  $B$ .

## Section 10.2

- 1.**  $v = 6$ ;  $e = 6$ ;  $\deg(a) = 2$ ,  $\deg(b) = 4$ ,  $\deg(c) = 1$ ,  $\deg(d) = 0$ ,  $\deg(e) = 2$ ,  $\deg(f) = 3$ ;  $c$  is pendant;  $d$  is isolated.
- 3.**  $v = 9$ ;  $e = 12$ ;  $\deg(a) = 3$ ,  $\deg(b) = 2$ ,  $\deg(c) = 4$ ,  $\deg(d) = 0$ ,  $\deg(e) = 6$ ,  $\deg(f) = 0$ ;  $\deg(g) = 4$ ;  $\deg(h) = 2$ ;  $\deg(i) = 3$ ;  $d$  and  $f$  are isolated.
- 5.** No
- 7.**  $v = 4$ ;  $e = 7$ ;  $\deg^-(a) = 3$ ,  $\deg^-(b) = 1$ ,  $\deg^-(c) = 2$ ,  $\deg^-(d) = 1$ ,  $\deg^+(a) = 1$ ,  $\deg^+(b) = 2$ ,  $\deg^+(c) = 1$ ,  $\deg^+(d) = 3$
- 9.** 5 vertices, 13 edges;  $\deg^-(a) = 6$ ,  $\deg^+(a) = 1$ ,  $\deg^-(b) = 1$ ,  $\deg^+(b) = 5$ ,  $\deg^-(c) = 2$ ,  $\deg^+(c) = 5$ ,  $\deg^-(d) = 4$ ,  $\deg^+(d) = 2$ ,  $\deg^-(e) = 0$ ,  $\deg^+(e) = 0$

**11.**



- 13.** The number of coauthors that person has; that person's coauthors; a person who has no coauthors; a person who has only one coauthor
- 15.** In the directed graph  $\deg^-(v) =$  number of calls  $v$  received,  $\deg^+(v) =$  number of

calls  $v$  made; in the undirected graph,  $\deg(v)$  is the number of calls either made or received by  $v$ .

**17.**  $(\deg^+(v), \deg^-(v))$  is the win-loss record of  $v$ .

**19.** In the undirected graph model in which the vertices are people in the group and two vertices are adjacent if those two people are friends, the degree of a vertex is the number of friends in the group that person has. By Exercise 18, there are two vertices with the same degree, which means that there are two people in the group with the same number of friends in the group.

**21.** Bipartite

**23.** Not bipartite

**25.** Not bipartite

**27.** a) Parts  $\{h, s, n, w\}$  and  $\{P, Q, R, S\}$ ,  $E = \{\{P, n\}, \{P, w\}, \{Q, s\}, \{Q, n\}, \{R, n\}, \{R, w\}, \{S, h\}, \{S, s\}\}$

b) There is.

c)  $\{Pw, Qs, Rn, Sh\}$  among others

**29.** Only Barry is willing to marry Uma and Xia.

**31.** Model this with an undirected bipartite graph, with an edge between a man and a woman if they are willing to marry each other. By Hall's theorem, it is enough to show that for every set  $S$  of women, the set  $N(S)$  of men willing to marry them has cardinality at least  $|S|$ . Let  $m$  be the number of edges between  $S$  and  $N(S)$ . Since every vertex in  $S$  has degree  $k$ , it follows that  $m = k|S|$ . Because these edges are incident to  $N(S)$ , it follows that  $m \leq k|N(S)|$ . Therefore  $k|S| \leq k|N(S)|$ , so  $|N(S)| \geq |S|$ .

**33.** a)  $(\{a, b, c, f\}, \{\{a, b\}, \{a, f\}, \{b, c\}, \{b, f\}\})$

b)  $(\{a, x, c, f\}, \{\{a, x\}, \{c, x\}, \{e, x\}\})$

**35.** a)  $n$  vertices,  $n(n-1)/2$  edges

b)  $n$  vertices,  $n$  edges

c)  $n+1$  vertices,  $2n$  edges

d)  $m+n$  vertices,  $mn$  edges

e)  $2^n$  vertices,  $n2^{n-1}$  edges

**37.** a) 3, 3, 3, 3

b) 2, 2, 2, 2

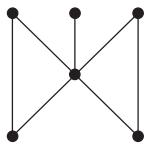
c) 4, 3, 3, 3, 3

d) 3, 3, 2, 2, 2

e) 3, 3, 3, 3, 3, 3, 3

**39.** Each of the  $n$  vertices is adjacent to each of the other  $n-1$  vertices, so the degree sequence is  $n-1, n-1, \dots, n-1$  ( $n$  terms).

**41.** 7



**43.** a) Yes



b) No

c) No

d) No

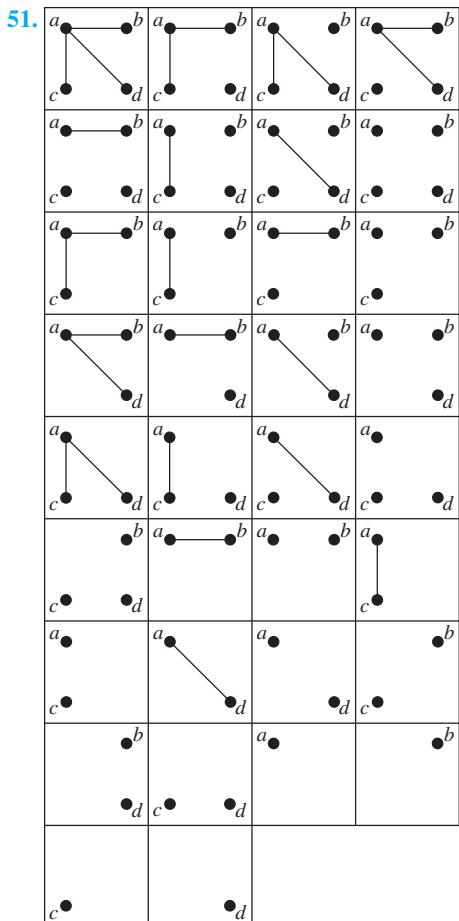
e) Yes



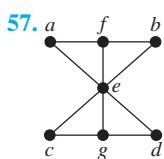
f) No.

**45.** First, suppose that  $d_1, d_2, \dots, d_n$  is graphic. We must show that the sequence whose terms are  $d_2-1, d_3-1, \dots, d_{d_1+1}-1, d_{d_1+2}, d_{d_1+3}, \dots, d_n$  is graphic once it is put into non-increasing order. In Exercise 44 it is proved that if the original sequence is graphic, then in fact there is a graph having this degree sequence in which the vertex of degree  $d_1$  is adjacent to the vertices of degrees  $d_2, d_3, \dots, d_{d_1+1}$ . Remove from this graph the vertex of highest degree ( $d_1$ ). The resulting graph has the desired degree sequence. Conversely, suppose that  $d_1, d_2, \dots, d_n$  is a nonincreasing sequence such that the sequence

$d_2 - 1, d_3 - 1, \dots, d_{d_1+1} - 1, d_{d_1+2}, d_{d_1+3}, \dots, d_n$  is graphic once it is put into nonincreasing order. Take a graph with this latter degree sequence, where vertex  $v_i$  has degree  $d_i - 1$  for  $2 \leq i \leq d_1 + 1$  and vertex  $v_i$  has degree  $d_i$  for  $d_1 + 2 \leq i \leq n$ . Adjoin one new vertex (call it  $v_1$ ), and put in an edge from  $v_1$  to each of the vertices  $v_2, v_3, \dots, v_{d_1+1}$ . The resulting graph has degree sequence  $d_1, d_2, \dots, d_n$ . **47.** Let  $d_1, d_2, \dots, d_n$  be a nonincreasing sequence of nonnegative integers with an even sum. Construct a graph as follows: Take vertices  $v_1, v_2, \dots, v_n$  and put  $\lfloor d_i/2 \rfloor$  loops at vertex  $v_i$ , for  $i = 1, 2, \dots, n$ . For each  $i$ , vertex  $v_i$  now has degree either  $d_i$  or  $d_i - 1$ . Because the original sum was even, the number of vertices for which  $\deg(v_i) = d_i - 1$  is even. Pair them up arbitrarily, and put in an edge joining the vertices in each pair. **49.** 17



**53. a)** For all  $n \geq 1$  **b)** For all  $n \geq 3$  **c)** For  $n = 3$  **d)** For all  $n \geq 0$  **55.** 5

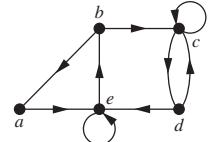


**59. a)** The graph with  $n$  vertices and no edges **b)** The disjoint union of  $K_m$  and  $K_n$  **c)** The graph with vertices  $\{v_1, \dots, v_n\}$

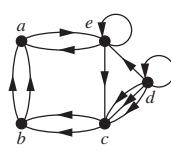
with an edge between  $v_i$  and  $v_j$  unless  $i \equiv j \pm 1 \pmod{n}$

**d)** The graph whose vertices are represented by bit strings of length  $n$  with an edge between two vertices if the associated bit strings differ in more than one bit **61.**  $v(v-1)/2 - e$  **63.**  $n-1-d_n, n-1-d_{n-1}, \dots, n-1-d_2, n-1-d_1$  **65.** The union of  $G$  and  $\overline{G}$  contains an edge between each pair of the  $n$  vertices. Hence, this union is  $K_n$ .

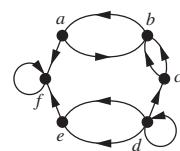
**67. Exercise 7:**



**Exercise 8:**

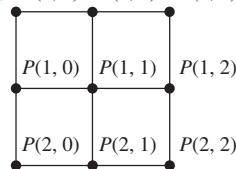


**Exercise 9:**



**69.** A directed graph  $G = (V, E)$  is its own converse if and only if it satisfies the condition  $(u, v) \in E$  if and only if  $(v, u) \in E$ . But this is precisely the condition that the associated relation must satisfy to be symmetric.

**71.**  $P(0, 0) \quad P(0, 1) \quad P(0, 2)$



**73.** We can connect  $P(i, j)$  and  $P(k, l)$  by using  $|i-k|$  hops to connect  $P(i, j)$  and  $P(k, j)$  and  $|j-l|$  hops to connect  $P(k, j)$  and  $P(k, l)$ . Hence, the total number of hops required to connect  $P(i, j)$  and  $P(k, l)$  does not exceed  $|i-k| + |j-l|$ . This is less than or equal to  $m + m = 2m$ , which is  $O(m)$ .

### Section 10.3

| Vertex | Adjacent Vertices |
|--------|-------------------|
| a      | b, c, d           |
| b      | a, d              |
| c      | a, d              |
| d      | a, b, c           |

| Vertex | Terminal Vertices |
|--------|-------------------|
| a      | a, b, c, d        |
| b      | d                 |
| c      | a, b              |
| d      | b, c, d           |

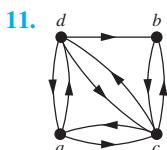
**5.** 
$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

**7.** 
$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$
    **9. a)** 
$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

b)  $\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$  c)  $\begin{bmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$

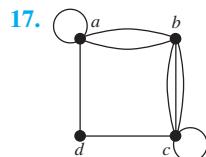
d)  $\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$  e)  $\begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$

f)  $\begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$



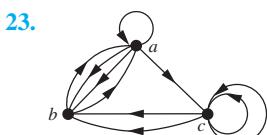
13.  $\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 \\ 0 & 2 & 1 & 0 \end{bmatrix}$

15.  $\begin{bmatrix} 1 & 0 & 2 & 1 \\ 0 & 1 & 1 & 2 \\ 2 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{bmatrix}$



19.  $\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$

21.  $\begin{bmatrix} 1 & 1 & 2 & 1 \\ 1 & 0 & 0 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 2 & 1 & 0 \end{bmatrix}$



25. Yes

27. Exercise 13:  $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$

Exercise 14:  $\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$

Exercise 15:  $\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$

29.  $\deg(v)$  – number of loops at  $v$ ;  $\deg^-(v)$  31. 2 if  $e$  is not a loop, 1 if  $e$  is a loop

33. a)  $\begin{bmatrix} 1 & 1 & \dots & 1 & 0 & \dots & 0 \\ 1 & 0 & \dots & 0 & 1 & \dots & 0 \\ 0 & 1 & \dots & 0 & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 1 \\ 0 & 0 & \dots & 1 & 0 & \dots & 1 \end{bmatrix}$

b)  $\begin{bmatrix} 1 & 0 & \dots & 0 & 1 \\ 1 & 1 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & \dots & 1 & 1 \end{bmatrix}$

c)  $\begin{bmatrix} 0 & 0 & \dots & 0 & 1 & 1 & \dots & 1 \\ & & & & 1 & 0 & \dots & 0 \\ \mathbf{B} & & & & 0 & 1 & \dots & 0 \\ & & & & & \vdots & & \vdots \\ & & & & & 0 & \dots & 1 \end{bmatrix}$

where  $\mathbf{B}$  is the answer to (b)

d)  $\begin{bmatrix} 1 & 1 & \dots & 1 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & \dots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & 1 \\ 1 & 0 & \dots & 0 & 1 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & \dots & 1 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 1 & 0 & \dots & 0 \end{bmatrix}$

35. Isomorphic 37. Isomorphic 39. Isomorphic 41. Not isomorphic 43. Isomorphic 45.  $G$  is isomorphic to itself by the identity function, so isomorphism is reflexive. Suppose that  $G$  is isomorphic to  $H$ . Then there exists a one-to-one correspondence  $f$  from  $G$  to  $H$  that preserves adjacency and nonadjacency. It follows that  $f^{-1}$  is a one-to-one correspondence from  $H$  to  $G$  that preserves adjacency and nonadjacency. Hence, isomorphism is symmetric. If  $G$  is isomorphic to  $H$  and  $H$  is isomorphic to  $K$ , then there are one-to-one correspondences  $f$  and  $g$  from  $G$  to  $H$  and from  $H$  to  $K$  that preserve adjacency and nonadjacency. It follows that  $g \circ f$  is a one-to-one correspondence from  $G$  to  $K$  that preserves adjacency and nonadjacency. Hence, isomorphism is transitive.

47. All zeros 49. Label the vertices in order so that all of the vertices in the first set of the partition of the vertex set come first. Because no edges join vertices in the same set of the partition, the matrix has the desired form. 51.  $C_5$  53.  $n = 5$  only 55. 4 57. a) Yes b) No c) No 59.  $G = (V_1, E_1)$  is isomorphic to  $H = (V_2, E_2)$  if and only if there exist functions  $f$  from  $V_1$  to  $V_2$  and  $g$  from  $E_1$  to  $E_2$  such that each is a one-to-one correspondence and for every edge  $e$  in  $E_1$  the endpoints of  $g(e)$  are  $f(v)$  and  $f(w)$  where  $v$  and  $w$  are the

endpoints of  $e$ . **61.** Yes **63.** Yes **65.** If  $f$  is an isomorphism from a directed graph  $G$  to a directed graph  $H$ , then  $f$  is also an isomorphism from  $G^{\text{conv}}$  to  $H^{\text{conv}}$ . To see this note that  $(u, v)$  is an edge of  $G^{\text{conv}}$  if and only if  $(v, u)$  is an edge of  $G$  if and only if  $(f(v), f(u))$  is an edge of  $H$  if and only if  $(f(u), f(v))$  is an edge of  $H^{\text{conv}}$ . **67.** Many answers are possible; for example,  $C_6$  and  $C_3 \cup C_3$ . **69.** The product is  $[a_{ij}]$  where  $a_{ij}$  is the number of edges from  $v_i$  to  $v_j$  when  $i \neq j$  and  $a_{ii}$  is the number of edges incident to  $v_i$ . **71.** The graphs in Exercise 41 provide a devil's pair.

## Section 10.4

**1.** a) Path of length 4; not a circuit; not simple **b)** Not a path **c)** Not a path **d)** Simple circuit of length 5 **3.** No **5.** No **7.** Maximal sets of people with the property that for any two of them, we can find a string of acquaintances that takes us from one to the other **9.** If a person has Erdős number  $n$ , then there is a path of length  $n$  from that person to Erdős in the collaboration graph, so by definition, that means that that person is in the same component as Erdős. If a person is in the same component as Erdős, then there is a path from that person to Erdős, and the length of the shortest such path is that person's Erdős number. **11.** a) Weakly connected **b)** Weakly connected **c)** Not strongly or weakly connected **13.** The maximal sets of phone numbers for which it is possible to find directed paths between every two different numbers in the set **15.** a)  $\{a, b, f\}, \{c, d, e\}$  **b)**  $\{a, b, c, d, e, h\}, \{f\}, \{g\}$  **c)**  $\{a, b, d, e, f, g, h, i\}, \{c\}$  **17.** Suppose the strong components of  $u$  and  $v$  are not disjoint, say with vertex  $w$  in both. Suppose  $x$  is a vertex in the strong component of  $u$ . Then  $x$  is also in the strong component of  $v$ , because there is a path from  $x$  to  $v$  (namely the path from  $x$  to  $u$  followed by the path from  $u$  to  $w$  followed by the path from  $w$  to  $v$ ) and vice versa. Thus  $x$  is in the strong component of  $v$ . This shows that the strong component of  $u$  is a subgraph of the strong component of  $v$ , and equality follows by symmetry. **19.** a) 2 **b)** 7 **c)** 20 **d)** 61 **21.** Not isomorphic ( $G$  has a triangle;  $H$  does not) **23.** Isomorphic (the path  $u_1, u_2, u_7, u_6, u_5, u_4, u_3, u_8, u_1$  corresponds to the path  $v_1, v_2, v_3, v_4, v_5, v_8, v_7, v_6, v_1$ ) **25.** a) 3 **b)** 0 **c)** 27 **d)** 0 **27.** a) 1 **b)** 0 **c)** 2 **d)** 1 **e)** 5 **f)** 3 **29.**  $R$  is reflexive by definition. Assume that  $(u, v) \in R$ ; then there is a path from  $u$  to  $v$ . Then  $(v, u) \in R$  because there is a path from  $v$  to  $u$ , namely, the path from  $u$  to  $v$  traversed backward. Assume that  $(u, v) \in R$  and  $(v, w) \in R$ ; then there are paths from  $u$  to  $v$  and from  $v$  to  $w$ . Putting these two paths together gives a path from  $u$  to  $w$ . Hence,  $(u, w) \in R$ . It follows that  $R$  is transitive. **31.** c **33.** b, c, e, i **35.** If a vertex is pendant it is clearly not a cut vertex. So an endpoint of a cut edge that is a cut vertex is not pendant. Removal of a cut edge produces a graph with more connected components than in the original graph. If an endpoint of a cut edge is not pendant, the connected component it is in after the removal of the cut edge contains more than just this vertex. Consequently, removal of that vertex and all edges incident to it, including the original

cut edge, produces a graph with more connected components than were in the original graph. Hence, an endpoint of a cut edge that is not pendant is a cut vertex. **37.** Assume there exists a connected graph  $G$  with at most one vertex that is not a cut vertex. Define the distance between the vertices  $u$  and  $v$ , denoted by  $d(u, v)$ , to be the length of the shortest path between  $u$  and  $v$  in  $G$ . Let  $s$  and  $t$  be vertices in  $G$  such that  $d(s, t)$  is a maximum. Either  $s$  or  $t$  (or both) is a cut vertex, so without loss of generality suppose that  $s$  is a cut vertex. Let  $w$  belong to the connected component that does not contain  $t$  of the graph obtained by deleting  $s$  and all edges incident to  $s$  from  $G$ . Because every path from  $w$  to  $t$  contains  $s$ ,  $d(w, t) > d(s, t)$ , which is a contradiction. **39.** a) Denver–Chicago, Boston–New York **b)** Seattle–Portland, Portland–San Francisco, Salt Lake City–Denver, New York–Boston, Boston–Burlington, Boston–Bangor **41.** A minimal set of people who collectively influence everyone (directly or indirectly); {Deborah} **43.** An edge cannot connect two vertices in different connected components. Because there are at most  $C(n_i, 2)$  edges in the connected component with  $n_i$  vertices, it follows that there are at most  $\sum_{i=1}^k C(n_i, 2)$  edges in the graph. **45.** Suppose that  $G$  is not connected. Then it has a component of  $k$  vertices for some  $k$ ,  $1 \leq k \leq n - 1$ . The most edges  $G$  could have is  $C(k, 2) + C(n - k, 2) = [k(k - 1) + (n - k)(n - k - 1)]/2 = k^2 - nk + (n^2 - n)/2$ . This quadratic function of  $f$  is minimized at  $k = n/2$  and maximized at  $k = 1$  or  $k = n - 1$ . Hence, if  $G$  is not connected, the number of edges does not exceed the value of this function at 1 and at  $n - 1$ , namely,  $(n - 1)(n - 2)/2$ . **47.** a) 1 **b)** 2 **c)** 6 **d)** 21 **49.** a) Removing an edge from a cycle leaves a path, which is still connected. **b)** Removing an edge from the cycle portion of the wheel leaves that portion still connected and the central vertex still connected to it as well. Removing a spoke leaves the cycle intact and the central vertex still connected to it as well. **c)** Any four vertices, two from each part of the bipartition, are connected by a 4-cycle; removing one edge does not disconnect them. **d)** Deleting the edge joining  $(b_1, b_2, \dots, b_{i-1}, 0, b_{i+1}, \dots, b_n)$  and  $(b_1, b_2, \dots, b_{i-1}, 1, b_{i+1}, \dots, b_n)$  does not disconnect the graph because these two vertices are still joined via the path  $(b_1, b_2, \dots, b_{i-1}, 0, b_{i+1}, \dots, 0)$ ,  $(b_1, b_2, \dots, b_{i-1}, 0, b_{i+1}, \dots, 1)$ ,  $(b_1, b_2, \dots, b_{i-1}, 1, b_{i+1}, \dots, 1)$ ,  $(b_1, b_2, \dots, b_{i-1}, 1, b_{i+1}, \dots, 0)$  if  $n < 2$  and  $b_n = 0$ , and similarly in the other three cases. **51.** If  $G$  is complete, then removing vertices one by one leaves a complete graph at each step, so we never get a disconnected graph. Conversely, if edge  $uv$  is missing from  $G$ , then removing all the vertices except  $u$  and  $v$  creates a disconnected graph. **53.** Both equal  $\min(m, n)$ . **55.** Let  $G$  be a graph with  $n$  vertices; then  $\kappa(G) \leq n - 1$ . Let  $C$  be a smallest edge cut, leaving a nonempty proper subset  $S$  of the vertices of  $G$  disconnected from the complementary set  $S' = V - S$ . If  $xy$  is an edge of  $G$  for every  $x \in S$  and  $y \in S'$ , then the size of  $C$  is  $|S||S'|$ , which is at least  $n - 1$ , so  $\kappa(G) \leq \lambda(G)$ . Otherwise, let  $x \in S$  and  $y \in S'$  be nonadjacent vertices. Let  $T$  consist of all neighbors of  $x$  in  $S'$  together with all vertices of  $S - \{x\}$  with neighbors

in  $S'$ . Then  $T$  is a vertex cut, because it separates  $x$  and  $y$ . Now look at the edges from  $x$  to  $T \cap S'$  and one edge from each vertex of  $T \cap S$  to  $S'$ ; this gives us  $|T|$  distinct edges that lie in  $C$ , so  $\lambda(G) = |C| \geq |T| \geq \kappa(G)$ . **57.** 2 **59.** Let the simple paths  $P_1$  and  $P_2$  be  $u = x_0, x_1, \dots, x_n = v$  and  $u = y_0, y_1, \dots, y_m = v$ , respectively. The paths thus start out at the same vertex. Since the paths do not contain the same set of edges, they must diverge eventually. If they diverge only after one of them has ended, then the rest of the other path is a simple circuit from  $v$  to  $v$ . Otherwise we can suppose that  $x_0 = y_0, x_1 = y_1, \dots, x_i = y_i$ , but  $x_{i+1} \neq y_{i+1}$ . To form our simple circuit, we follow the path  $y_i, y_{i+1}, y_{i+2}$ , and so on, until it once again first encounters a vertex on  $P_1$  (possibly as early as  $y_{i+1}$ , no later than  $y_m$ ). Once we are back on  $P_1$ , we follow it along—fowards or backwards, as necessary—to return to  $x_i$ . Since  $x_i = y_i$ , this certainly forms a circuit. It must be a simple circuit, since no edge among the  $x_k$ s or the  $y_l$ s can be repeated ( $P_1$  and  $P_2$  are simple by hypothesis) and no edge among the  $x_k$ s can equal one of the edges  $y_l$  that we used, since we abandoned  $P_2$  for  $P_1$  as soon as we hit  $P_1$ . **61.** The graph  $G$  is connected if and only if every off-diagonal entry of  $\mathbf{A} + \mathbf{A}^2 + \mathbf{A}^3 + \dots + \mathbf{A}^{n-1}$  is positive, where  $\mathbf{A}$  is the adjacency matrix of  $G$ . **63.** If the graph is bipartite, say with parts  $A$  and  $B$ , then the vertices in every path must alternately lie in  $A$  and  $B$ . Therefore a path that starts in  $A$ , say, will end in  $B$  after an odd number of steps and in  $A$  after an even number of steps. Because a circuit ends at the same vertex where it starts, the length must be even. Conversely, suppose that all circuits have even length; we must show that the graph is bipartite. We can assume that the graph is connected, because if it is not, then we can just work on one component at a time. Let  $v$  be a vertex of the graph, and let  $A$  be the set of all vertices to which there is a path of odd length starting at  $v$ , and let  $B$  be the set of all vertices to which there is a path of even length starting at  $v$ . Because the component is connected, every vertex lies in  $A$  or  $B$ . No vertex can lie in both  $A$  and  $B$ , because if one did, then following the odd-length path from  $v$  to that vertex and then back along the even-length path from that vertex to  $v$  would produce an odd circuit, contrary to the hypothesis. Thus, the set of vertices has been partitioned into two sets. To show that every edge has endpoints in different parts, suppose that  $xy$  is an edge, where  $x \in A$ . Then the odd-length path from  $v$  to  $x$  followed by  $xy$  produces an even-length path from  $v$  to  $y$ , so  $y \in B$ . (Similarly, if  $x \in B$ .) **65.**  $(H_1 W_1 H_2 W_2 \langle \text{boat} \rangle, \emptyset) \rightarrow (H_2 W_2, H_1 W_1 \langle \text{boat} \rangle) \rightarrow (H_1 H_2 W_2 \langle \text{boat} \rangle, W_1) \rightarrow (W_2, H_1 W_1 H_2 \langle \text{boat} \rangle) \rightarrow (H_2 W_2 \langle \text{boat} \rangle, H_1 W_1) \rightarrow (\emptyset, H_1 W_1 H_2 W_2 \langle \text{boat} \rangle)$

## Section 10.5

- 1.** Neither **3.** No Euler circuit;  $a, e, c, e, b, e, d, b, a, c, d$  **5.**  $a, b, c, d, c, e, d, b, e, a, e, a$  **7.**  $a, i, h, g, d, e, f, g, c, e, h, d, c, a, b, i, c, b, h, a$  **9.** No,  $A$  still has odd degree. **11.** When the graph in which vertices represent intersections and edges

streets has an Euler path **13.** Yes **15.** No **17.** If there is an Euler path, then as we follow it each vertex except the starting and ending vertices must have equal in-degree and out-degree, because whenever we come to a vertex along an edge, we leave it along another edge. The starting vertex must have out-degree 1 larger than its in-degree, because we use one edge leading out of this vertex and whenever we visit it again we use one edge leading into it and one leaving it. Similarly, the ending vertex must have in-degree 1 greater than its out-degree. Because the Euler path with directions erased produces a path between any two vertices, in the underlying undirected graph, the graph is weakly connected. Conversely, suppose the graph meets the degree conditions stated. If we add one more edge from the vertex of deficient out-degree to the vertex of deficient in-degree, then the graph has every vertex with equal in-degree and out-degree. Because the graph is still weakly connected, by Exercise 16 this new graph has an Euler circuit. Now delete the added edge to obtain the Euler path. **19.** Neither **21.** No Euler circuit;  $a, d, e, d, b, a, e, c, e, b, c, b, e$  **23.** Neither **25.** Follow the same procedure as Algorithm 1, taking care to follow the directions of edges. **27.** **a**)  $n = 2$  **b**) None **c**) None **d**)  $n = 1$  **29.** Exercise 1:1 time; Exercises 2–7: 0 times **31.**  $a, b, c, d, e, a$  is a Hamilton circuit. **33.** No Hamilton circuit exists, because once a purported circuit has reached  $e$  it would have nowhere to go. **35.** No Hamilton circuit exists, because every edge in the graph is incident to a vertex of degree 2 and therefore must be in the circuit. **37.**  $a, b, c, f, d, e$  is a Hamilton path. **39.**  $f, e, d, a, b, c$  is a Hamilton path. **41.** No Hamilton path exists. There are eight vertices of degree 2, and only two of them can be end vertices of a path. For each of the other six, their two incident edges must be in the path. It is not hard to see that if there is to be a Hamilton path, exactly one of the inside corner vertices must be an end, and that this is impossible. **43.**  $a, b, c, f, i, h, g, d, e$  is a Hamilton path. **45.**  $m = n \geq 2$  **47.** **a**) (i) No, (ii) No, (iii) Yes **b**) (i) No, (ii) No, (iii) Yes **c**) (i) Yes, (ii) Yes, (iii) Yes **d**) (i) Yes, (ii) Yes, (iii) Yes **49.** The result is trivial for  $n = 1$ : code is 0, 1. Assume we have a Gray code of order  $n$ . Let  $c_1, \dots, c_k, k = 2^n$  be such a code. Then  $0c_1, \dots, 0c_k, 1c_k, \dots, 1c_1$  is a Gray code of order  $n + 1$ .

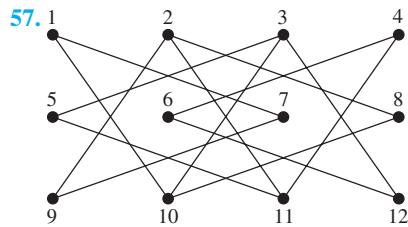
**51. procedure**  $\text{Fleury}(G = (V, E)$ : connected multigraph with the degrees of all vertices even,  $V = \{v_1, \dots, v_n\}$

```

 $v := v_1$
 $circuit := v$
 $H := G$
while H has edges
 $e :=$ first edge with endpoint v in H (with respect to listing of V) such that e is not a cut edge of H , if one exists, and simply the first edge in H with endpoint v otherwise
 $w :=$ other endpoint of e
 $circuit := circuit$ with e, w added
 $v := w$
 $H := H - e$
return $circuit$ { $circuit$ is an Euler circuit}

```

**53.** If  $G$  has an Euler circuit, then it also has an Euler path. If not, add an edge between the two vertices of odd degree and apply the algorithm to get an Euler circuit. Then delete the new edge. **55.** Suppose  $G = (V, E)$  is a bipartite graph with  $V = V_1 \cup V_2$ , where  $V_1 \cap V_2 = \emptyset$  and no edge connects a vertex in  $V_1$  and a vertex in  $V_2$ . Suppose that  $G$  has a Hamilton circuit. Such a circuit must be of the form  $a_1, b_1, a_2, b_2, \dots, a_k, b_k, a_1$ , where  $a_i \in V_1$  and  $b_i \in V_2$  for  $i = 1, 2, \dots, k$ . Because the Hamilton circuit visits each vertex exactly once, except for  $v_1$ , where it begins and ends, the number of vertices in the graph equals  $2k$ , an even number. Hence, a bipartite graph with an odd number of vertices cannot have a Hamilton circuit.



**59.** We represent the squares of a  $3 \times 4$  chessboard as follows:

|   |    |    |    |
|---|----|----|----|
| 1 | 2  | 3  | 4  |
| 5 | 6  | 7  | 8  |
| 9 | 10 | 11 | 12 |

A knight's tour can be made by following the moves 8, 10, 1, 7, 9, 2, 11, 5, 3, 12, 6, 4. **61.** We represent the squares of a  $4 \times 4$  chessboard as follows:

|    |    |    |    |
|----|----|----|----|
| 1  | 2  | 3  | 4  |
| 5  | 6  | 7  | 8  |
| 9  | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

There are only two moves from each of the four corner squares. If we include all the edges 1–10, 1–7, 16–10, and 16–7, a circuit is completed too soon, so at least one of these edges must be missing. Without loss of generality, assume the path starts 1–10, 10–16, 16–7. Now the only moves from square 3 are to squares 5, 10, and 12, and square 10 already has two incident edges. Therefore, 3–5 and 3–12 must be in the Hamilton circuit. Similarly, edges 8–2 and 8–15 must be in the circuit. Now the only moves from square 9 are to squares 2, 7, and 15. If there were edges from square 9 to both squares 2 and 15, a circuit would be completed too soon. Therefore the edge 9–7 must be in the circuit giving square 7 its full complement

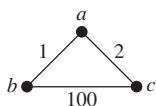
of edges. But now square 14 is forced to be joined to squares 5 and 12, completing a circuit too soon (5–14–12–3–5). This contradiction shows that there is no knight's tour on the  $4 \times 4$  board. **63.** Because there are  $mn$  squares on an  $m \times n$  board, if both  $m$  and  $n$  are odd, there are an odd number of squares. Because by Exercise 62 the corresponding graph is bipartite, by Exercise 55 it has no Hamilton circuit. Hence, there is no reentrant knight's tour. **65.** **a)** If  $G$  does not have a Hamilton circuit, continue as long as possible adding missing edges one at a time in such a way that we do not obtain a graph with a Hamilton circuit. This cannot go on forever, because once we've formed the complete graph by adding all missing edges, there is a Hamilton circuit. Whenever the process stops, we have obtained a (necessarily noncomplete) graph  $H$  with the desired property. **b)** Add one more edge to  $H$ . This produces a Hamilton circuit, which uses the added edge. The path consisting of this circuit with the added edge omitted is a Hamilton path in  $H$ . **c)** Clearly  $v_1$  and  $v_n$  are not adjacent in  $H$ , because  $H$  has no Hamilton circuit. Therefore they are not adjacent in  $G$ . But the hypothesis was that the sum of the degrees of vertices not adjacent in  $G$  was at least  $n$ . This inequality can be rewritten as  $n - \deg(v_n) \leq \deg(v_1)$ . But  $n - \deg(v_n)$  is just the number of vertices not adjacent to  $v_n$ . **d)** Because there is no vertex following  $v_n$  in the Hamilton path,  $v_n$  is not in  $S$ . Each one of the  $\deg(v_1)$  vertices adjacent to  $v_1$  gives rise to an element of  $S$ , so  $S$  contains  $\deg(v_1)$  vertices. **e)** By part (c) there are at most  $\deg(v_1) - 1$  vertices other than  $v_n$  not adjacent to  $v_n$ , and by part (d) there are  $\deg(v_1)$  vertices in  $S$ , none of which is  $v_n$ . Therefore at least one vertex of  $S$  is adjacent to  $v_n$ . By definition, if  $v_k$  is this vertex, then  $H$  contains edges  $v_k v_n$  and  $v_1 v_{k+1}$ , where  $1 < k < n - 1$ . **f)** Now  $v_1, v_2, \dots, v_{k-1}, v_k, v_n, v_{n-1}, \dots, v_{k+1}, v_1$  is a Hamilton circuit in  $H$ , contradicting the construction of  $H$ . Therefore, our assumption that  $G$  did not originally have a Hamilton circuit is wrong, and our proof by contradiction is complete.

## Section 10.6

- 1. a)** Vertices are the stops, edges join adjacent stops, weights are the times required to travel between adjacent stops.
- b)** Same as part (a), except weights are distances between adjacent stops.
- c)** Same as part (a), except weights are fares between stops.
- 3. 16 5.** Exercise 2: *a, b, e, d, z*; Exercise 3: *a, c, d, e, g, z*; Exercise 4: *a, b, e, h, l, m, p, s, z*
- 7. a, c, d b) a, c, d, f c) c, d, f e) b, d, e, g, z**
- 9. a)** Direct
- b)** Via New York
- c)** Via Atlanta and Chicago
- d)** Via New York
- 11. a)** Via Chicago
- b)** Via Chicago
- c)** Via Los Angeles
- d)** Via Chicago
- 13. a)** Via Chicago
- b)** Via Chicago
- c)** Via Los Angeles
- d)** Via Chicago
- 15.** Do not stop the algorithm when  $z$  is added to the set  $S$ .
- 17. a)** Via Woodbridge, via Woodbridge and Camden
- b)** Via Woodbridge, via Woodbridge and Camden
- 19.** For instance, sightseeing tours, street cleaning

| 21.      | <i>a</i> | <i>b</i> | <i>c</i> | <i>d</i> | <i>e</i> | <i>z</i> |
|----------|----------|----------|----------|----------|----------|----------|
| <i>a</i> | 4        | 3        | 2        | 8        | 10       | 13       |
| <i>b</i> | 3        | 2        | 1        | 5        | 7        | 10       |
| <i>c</i> | 2        | 1        | 2        | 6        | 8        | 11       |
| <i>d</i> | 8        | 5        | 6        | 4        | 2        | 5        |
| <i>e</i> | 10       | 7        | 8        | 2        | 4        | 3        |
| <i>z</i> | 13       | 10       | 11       | 5        | 3        | 6        |

23.  $O(n^3)$  25.  $a-c-b-d-a$  (or the same circuit starting at some other point and/or traversing the vertices in reverse order) 27. San Francisco–Denver–Detroit–New York–Los Angeles–San Francisco (or the same circuit starting at some other point and/or traversing the vertices in reverse order)  
 29. Consider this graph:

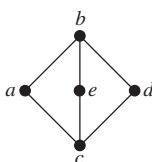


The circuit  $a-b-a-c-a$  visits each vertex at least once (and the vertex  $a$  twice) and has total weight 6. Every Hamilton circuit has total weight 103. 31. Let  $v_1, v_2, \dots, v_n$  be a topological ordering of the vertices of the given directed acyclic graph. Let  $w(i, j)$  be the weight of edge  $v_i v_j$ . Iteratively define  $P(i)$  with the intent that it will be the weight of a longest path ending at  $v_i$  and  $C(i)$  with the intent that it will be the vertex preceding  $v_i$  in some longest path: For  $i$  from 1 to  $n$ , let  $P(i)$  be the maximum of  $P(j) + w(j, i)$  over all  $j < i$  such that  $v_j v_i$  is an edge in the directed graph (and if such a  $j$  exists let  $C(i)$  be a value of  $j$  for which this maximum is achieved) and let  $P(i) = 0$  if there are no such values of  $j$ . At the conclusion of this loop, a longest path can be found by choosing  $i$  that maximizes  $P(i)$  and following the  $C$  links back to the start of the path.

## Section 10.7

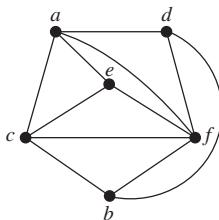
1. Yes

3.



5. No

7. Yes

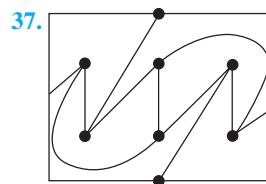


9. No 11. A triangle is formed by the planar representation of the subgraph of  $K_5$  consisting of the edges connecting  $v_1, v_2$ , and  $v_3$ . The vertex  $v_4$  must be placed either within the triangle or outside of it. We will consider only the case when  $v_4$  is inside the triangle; the other case is similar. Drawing the

three edges from  $v_1, v_2$ , and  $v_3$  to  $v_4$  forms four regions. No matter which of these four regions  $v_5$  is in, it is possible to join it to only three, and not all four, of the other vertices.

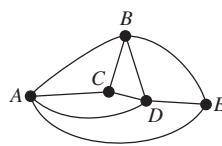
13. 8 15. Because there are no loops or multiple edges and no simple circuits of length 3, and the degree of the unbounded region is at least 4, each region has degree at least 4. Thus  $2e \geq 4r$ , or  $r \leq e/2$ . But  $r = e - v + 2$ , so we have  $e - v + 2 \leq e/2$ , which implies that  $e \leq 2v - 4$ . 17. As in the argument in the proof of Corollary 1, we have  $2e \geq 5r$  and  $r = e - v + 2$ . Thus  $e - v + 2 \leq 2e/5$ , which implies that  $e \leq (5/3)v - (10/3)$ . 19. Only (a) and (c) 21. Not homeomorphic to  $K_{3,3}$  23. Planar 25. Nonplanar 27. a) 1 b) 3 c) 9 d) 2 e) 4 f) 16 29. Draw  $K_{m,n}$  as described in the hint. The number of crossings is four times the number in the first quadrant. The vertices on the  $x$ -axis to the right of the origin are  $(1, 0), (2, 0), \dots, (m/2, 0)$  and the vertices on the  $y$ -axis above the origin are  $(0, 1), (0, 2), \dots, (0, n/2)$ . We obtain all crossings by choosing any two numbers  $a$  and  $b$  with  $1 \leq a < b \leq m/2$  and two numbers  $r$  and  $s$  with  $1 \leq r < s \leq n/2$ ; we get exactly one crossing in the graph between the edge connecting  $(a, 0)$  and  $(0, s)$  and the edge connecting  $(b, 0)$  and  $(0, r)$ . Hence, the number of crossings in the first quadrant is  $C\left(\frac{m}{2}, 2\right) \cdot C\left(\frac{n}{2}, 2\right) = \frac{(m/2)(m/2-1)}{2} \cdot \frac{(n/2)(n/2-1)}{2}$ . Hence, the total number of crossings is  $4 \cdot mn(m-2)(n-2)/64 = mn(m-2)(n-2)/16$ .

31. a) 2 b) 2 c) 2 d) 2 e) 2 f) 2 33. The formula is valid for  $n \leq 4$ . If  $n > 4$ , by Exercise 32 the thickness of  $K_n$  is at least  $C(n, 2)/(3n - 6) = (n+1 + \frac{2}{n-2})/6$  rounded up. Because this quantity is never an integer, it equals  $\lfloor (n+7)/6 \rfloor$ . 35. This follows from Exercise 34 because  $K_{m,n}$  has  $mn$  edges and  $m+n$  vertices and has no triangles because it is bipartite.

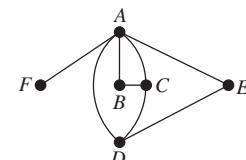


## Section 10.8

1. Four colors



3. Three colors



5. 3 7. 3 9. 2 11. 3 13. Graphs with no edges 15. 3 if  $n$  is even, 4 if  $n$  is odd 17. Period 1: Math 115, Math 185; period 2: Math 116, CS 473; period 3: Math 195, CS 101; period 4: CS 102; period 5: CS 273 19. 5 21. Exercise 5: 3 Exercise 6: 6 Exercise 7: 3 Exercise 8: 4 Exercise 9: 3 Exercise 10: 6 Exercise 11: 4 23. a) 2 if  $n$  is even, 3 if  $n$  is odd b)  $n$  25. Two edges that have the same color share no endpoints. Therefore if more than  $n/2$  edges were colored the

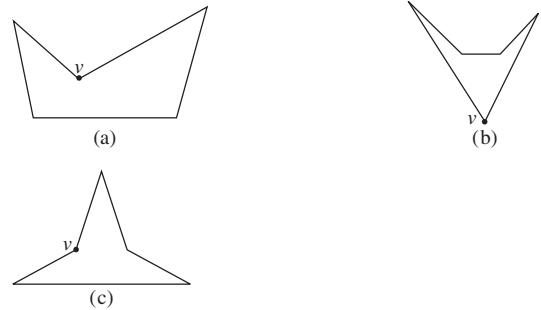
same, the graph would have more than  $2(n/2) = n$  vertices.

**27. 5** **29.** Color 1:  $e, f, d$ ; color 2:  $c, a, i, g$ ; color 3:  $h, b, j$

**31.** Color  $C_6$  **33.** Four colors are needed to color  $W_n$  when  $n$  is an odd integer greater than 1, because three colors are needed for the rim (see Example 4), and the center vertex, being adjacent to all the rim vertices, will require a fourth color. To see that the graph obtained from  $W_n$  by deleting one edge can be colored with three colors, consider two cases. If we remove a rim edge, then we can color the rim with two colors, by starting at an endpoint of the removed edge and using the colors alternately around the portion of the rim that remains. The third color is then assigned to the center vertex. If we remove a spoke edge, then we can color the rim by assigning color #1 to the rim endpoint of the removed edge and colors #2 and #3 alternately to the remaining vertices on the rim, and then assign color #1 to the center. **35.** Suppose that  $G$  is chromatically  $k$ -critical but has a vertex  $v$  of degree  $k - 2$  or less. Remove from  $G$  one of the edges incident to  $v$ . By definition of “ $k$ -critical,” the resulting graph can be colored with  $k - 1$  colors. Now restore the missing edge and use this coloring for all vertices except  $v$ . Because we had a proper coloring of the smaller graph, no two adjacent vertices have the same color. Furthermore,  $v$  has at most  $k - 2$  neighbors, so we can color  $v$  with an unused color to obtain a proper  $(k - 1)$ -coloring of  $G$ . This contradicts the fact that  $G$  has chromatic number  $k$ . Therefore, our assumption was wrong, and every vertex of  $G$  must have degree at least  $k - 1$ .

**37. a) 6 b) 7 c) 9 d) 11** **39.** Represent frequencies by colors and zones by vertices. Join two vertices with an edge if the zones these vertices represent interfere with one another. Then a  $k$ -tuple coloring is precisely an assignment of frequencies that avoids interference. **41.** We use induction on the number of vertices of the graph. Every graph with five or fewer vertices can be colored with five or fewer colors, because each vertex can get a different color. That takes care of the basis case(s). So we assume that all graphs with  $k$  vertices can be 5-colored and consider a graph  $G$  with  $k + 1$  vertices. By Corollary 2 in Section 10.7,  $G$  has a vertex  $v$  with degree at most 5. Remove  $v$  to form the graph  $G'$ . Because  $G'$  has only  $k$  vertices, we 5-color it by the inductive hypothesis. If the neighbors of  $v$  do not use all five colors, then we can 5-color  $G$  by assigning to  $v$  a color not used by any of its neighbors. The difficulty arises if  $v$  has five neighbors, and each has a different color in the 5-coloring of  $G'$ . Suppose that the neighbors of  $v$ , when considered in clockwise order around  $v$ , are  $a, b, c, m$ , and  $p$ . (This order is determined by the clockwise order of the curves representing the edges incident to  $v$ .) Suppose that the colors of the neighbors are azure, blue, chartreuse, magenta, and purple, respectively. Consider the azure-chartreuse subgraph (i.e., the vertices in  $G$  colored azure or chartreuse and all the edges between them). If  $a$  and  $c$  are not in the same component of this graph, then in the component containing  $a$  we can interchange these two colors (make the azure vertices chartreuse and vice versa), and  $G'$  will still be properly colored. That makes  $a$  chartreuse, so we can now color  $v$  azure, and  $G$  has been properly colored. If  $a$  and  $c$  are in the same component,

then there is a path of vertices alternately colored azure and chartreuse joining  $a$  and  $c$ . This path together with edges  $av$  and  $vc$  divides the plane into two regions, with  $b$  in one of them and  $m$  in the other. If we now interchange blue and magenta on all the vertices in the same region as  $b$ , we will still have a proper coloring of  $G'$ , but now blue is available for  $v$ . In this case, too, we have found a proper coloring of  $G$ . This completes the inductive step, and the theorem is proved. **43.** We follow the hint. Because the measures of the interior angles of a pentagon total  $540^\circ$ , there cannot be as many as three interior angles of measure more than  $180^\circ$  (reflex angles). If there are no reflex angles, then the pentagon is convex, and a guard placed at any vertex can see all points. If there is one reflex angle, then the pentagon must look essentially like figure (a) below, and a guard at vertex  $v$  can see all points. If there are two reflex angles, then they can be adjacent or nonadjacent (figures (b) and (c)); in either case, a guard at vertex  $v$  can see all points. [In figure (c), choose the reflex vertex closer to the bottom side.] Thus for all pentagons, one guard suffices, so  $g(5) = 1$ .



**45.** The figure suggested in the hint (generalized to have  $k$  prongs for any  $k \geq 1$ ) has  $3k$  vertices. The sets of locations from which the tips of different prongs are visible are disjoint. Therefore, a separate guard is needed for each of the  $k$  prongs, so at least  $k$  guards are needed. This shows that  $g(3k) \geq k = \lfloor 3k/3 \rfloor$ . If  $n = 3k + i$ , where  $0 \leq i \leq 2$ , then  $g(n) \geq g(3k) \geq k = \lfloor n/3 \rfloor$ .

### Supplementary Exercises

- 1. 2500** **3. Yes** **5. Yes** **7.**  $\sum_{i=1}^m n_i$  vertices,  $\sum_{i < j} n_i n_j$  edges **9. a)** If  $x \in N(A \cup B)$ , then  $x$  is adjacent to some vertex  $v \in A \cup B$ . WLOG suppose  $v \in A$ ; then  $x \in N(A)$  and therefore also in  $N(A) \cup N(B)$ . Conversely, if  $x \in N(A) \cup N(B)$ , then WLOG suppose  $x \in N(A)$ . Thus  $x$  is adjacent to some vertex  $v \in A \subseteq A \cup B$ , so  $x \in N(A \cup B)$ . **b)** If  $x \in N(A \cap B)$ , then  $x$  is adjacent to some vertex  $v \in A \cap B$ . Since both  $v \in A$  and  $v \in B$ , it follows that  $x \in N(A)$  and  $x \in N(B)$ , whence  $x \in N(A) \cap N(B)$ . For the counterexample, let  $G = (\{u, v, w\}, \{\{u, v\}, \{v, w\}\})$ ,  $A = \{u\}$ , and  $B = \{w\}$ . **11.**  $(c, a, p, x, n, m)$  and many others **13.**  $(c, d, a, b)$  and many others **15.** 6 times the

number of triangles divided by the number of paths of length 2

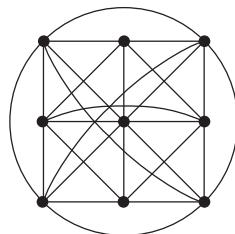
**17. a)** The probability that two actors each of whom has appeared in a film with a randomly chosen actor have appeared in a film together **b)** The probability that two of a randomly chosen person's Facebook friends are themselves Facebook friends **c)** The probability that two of a randomly chosen person's coauthors are themselves coauthors **d)** The probability that two proteins that each interact with a randomly chosen protein interact with each other **e)** The probability that two routers each of which has a communications link to a randomly chosen router are themselves linked

**19.** Complete subgraphs containing the following sets of vertices:  $\{b, c, e, f\}$ ,  $\{a, b, g\}$ ,  $\{a, d, g\}$ ,  $\{d, e, g\}$ ,  $\{b, e, g\}$

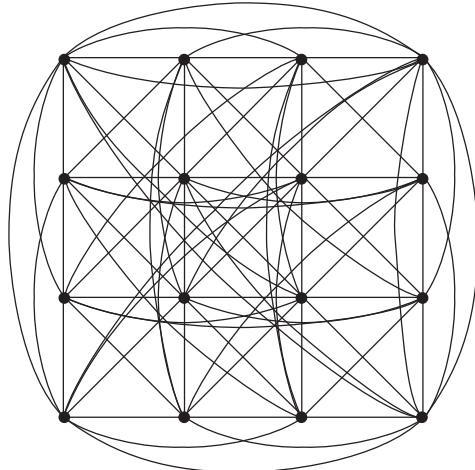
**21.** Complete subgraphs containing the following sets of vertices:  $\{b, c, d, j, k\}$ ,  $\{a, b, j, k\}$ ,  $\{e, f, g, i\}$ ,  $\{a, b, i\}$ ,  $\{a, i, j\}$ ,  $\{b, d, e\}$ ,  $\{b, e, i\}$ ,  $\{b, i, j\}$ ,  $\{g, h, i\}$ ,  $\{h, i, j\}$

**23.**  $\{c, d\}$  is a minimum dominating set.

**25. a)**



**b)**



**27. a) 1 b) 2 c) 3** **29. a)** A path from  $u$  to  $v$  in a graph  $G$  induces a path from  $f(u)$  to  $f(v)$  in an isomorphic graph  $H$ . **b)** Suppose  $f$  is an isomorphism from  $G$  to  $H$ . If  $v_0, v_1, \dots, v_n, v_0$  is a Hamilton circuit in  $G$ , then  $f(v_0), f(v_1), \dots, f(v_n), f(v_0)$  must be a Hamilton circuit in  $H$  because it is still a circuit and  $f(v_i) \neq f(v_j)$  for  $0 \leq i < j \leq n$ . **c)** Suppose  $f$  is an isomorphism from  $G$  to  $H$ . If  $v_0, v_1, \dots, v_n, v_0$  is an Euler circuit in  $G$ , then  $f(v_0), f(v_1), \dots, f(v_n), f(v_0)$  must be an Euler circuit in  $H$  because it is a circuit that contains each edge exactly once. **d)** Two isomorphic graphs must have the same crossing number because they can be drawn exactly the same way in the plane. **e)** Suppose  $f$  is an isomorphism from  $G$  to  $H$ . Then  $v$  is isolated in  $G$  if and only if  $f(v)$  is isolated in  $H$ . Hence, the graphs must have the same number of isolated vertices.

**f)** Suppose  $f$  is an isomorphism from  $G$  to  $H$ . If  $G$  is bipartite, then the vertex set of  $G$  can be partitioned into  $V_1$  and  $V_2$  with no edge connecting vertices within  $V_1$  or vertices within  $V_2$ . Then the vertex set of  $H$  can be partitioned into  $f(V_1)$  and  $f(V_2)$  with no edge connecting vertices within  $f(V_1)$  or vertices within  $f(V_2)$ . **31. 3** **33. a)** Yes **b)** No **35. No** **37. Yes** **39.** If  $e$  is a cut edge with endpoints  $u$  and  $v$ , then if we direct  $e$  from  $u$  to  $v$ , there will be no path in the directed graph from  $v$  to  $u$ , or else  $e$  would not have been a cut edge. Similar reasoning works if we direct  $e$  from  $v$  to  $u$ . **41.  $n - 1$**  **43.** Let the vertices represent the chickens. We include the edge  $(u, v)$  in the graph if and only if chicken  $u$  dominates chicken  $v$ . **45.** By the handshaking theorem, the average vertex degree is  $2m/n$ , which equals the minimum degree; it follows that all the vertex degrees are equal. **47.  $K_{3,3}$**  and the skeleton of a triangular prism **49. a)** A Hamilton circuit in the graph exactly corresponds to a seating of the knights at the Round Table such that adjacent knights are friends. **b)** The degree of each vertex in this graph is at least  $2n - 1 - (n - 1) = n \geq (2n/2)$ , so by Dirac's theorem, this graph has a Hamilton circuit. **c) a, b, d, f, g, z** **51. a) 4 b) 2 c) 3 d) 4 e) 4 f) 2** **53. a)** Suppose that  $G = (V, E)$ . Let  $a, b \in V$ . We must show that the distance between  $a$  and  $b$  in  $\bar{G}$  is at most 2. If  $\{a, b\} \notin E$  this distance is 1, so assume  $\{a, b\} \in E$ . Because the diameter of  $G$  is greater than 3, there are vertices  $u$  and  $v$  such that the distance in  $G$  between  $u$  and  $v$  is greater than 3. Either  $u$  or  $v$ , or both, is not in the set  $\{a, b\}$ . Assume that  $u$  is different from both  $a$  and  $b$ . Either  $\{a, u\}$  or  $\{b, u\}$  belongs to  $E$ ; otherwise  $a, u, b$  would be a path in  $\bar{G}$  of length 2. So, without loss of generality, assume  $\{a, u\} \in E$ . Thus  $v$  cannot be  $a$  or  $b$ , and by the same reasoning either  $\{a, v\} \in E$  or  $\{b, v\} \in E$ . In either case, this gives a path of length less than or equal to 3 from  $u$  to  $v$  in  $G$ , a contradiction. **b)** Suppose  $G = (V, E)$ . Let  $a, b \in V$ . We must show that the distance between  $a$  and  $b$  in  $\bar{G}$  does not exceed 3. If  $\{a, b\} \notin E$ , the result follows, so assume that  $\{a, b\} \in E$ . Because the diameter of  $G$  is greater than or equal to 3, there exist vertices  $u$  and  $v$  such that the distance in  $G$  between  $u$  and  $v$  is greater than or equal to 3. Either  $u$  or  $v$ , or both, is not in the set  $\{a, b\}$ . Assume  $u$  is different from both  $a$  and  $b$ . Either  $\{a, u\} \in E$  or  $\{b, u\} \in E$ ; otherwise  $a, u, b$  is a path of length 2 in  $\bar{G}$ . So, without loss of generality, assume  $\{a, u\} \in E$ . Thus  $v$  is different from  $a$  and from  $b$ . If  $\{a, v\} \in E$ , then  $u, a, v$  is a path of length 2 in  $G$ , so  $\{a, v\} \notin E$  and thus  $\{b, v\} \in E$  (or else there would be a path  $a, v, b$  of length 2 in  $\bar{G}$ ). Hence,  $\{u, b\} \notin E$ ; otherwise  $u, b, v$  is a path of length 2 in  $G$ . Thus,  $a, v, u, b$  is a path of length 3 in  $\bar{G}$ , as desired. **55. a, b, e, z** **57. a, c, d, f, g, z** **59.** If  $G$  is planar, then because  $e \leq 3v - 6$ ,  $G$  has at most 27 edges. (If  $G$  is not connected it has even fewer edges.) Similarly,  $\bar{G}$  has at most 27 edges. But the union of  $G$  and  $\bar{G}$  is  $K_{11}$ , which has 55 edges, and  $55 > 27 + 27$ . **61.** Suppose that  $G$  is colored with  $k$  colors and has independence number  $i$ . Because each color class must be an independent set, each color class has no more than  $i$  elements. Thus there are at most  $ki$

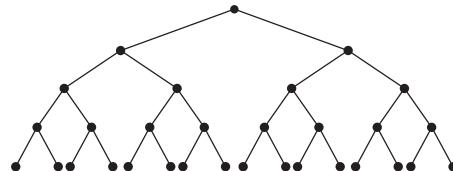
vertices. **63. a)**  $C(n, m)p^m(1 - p)^{n-m}$  **b)**  $np$  **c)** To generate a labeled graph  $G$ , as we apply the process to pairs of vertices, the random number  $x$  chosen must be less than or equal to  $1/2$  when  $G$  has an edge between that pair of vertices and greater than  $1/2$  when  $G$  has no edge there. Hence, the probability of making the correct choice is  $1/2$  for each edge and  $1/2^{C(n, 2)}$  overall. Hence, all labeled graphs are equally likely. **65.** Suppose  $P$  is monotone increasing. If the property of not having  $P$  were not retained whenever edges are removed from a simple graph, there would be a simple graph  $G$  not having  $P$  and another simple graph  $G'$  with the same vertices but with some of the edges of  $G$  missing that has  $P$ . But  $P$  is monotone increasing, so because  $G'$  has  $P$ , so does  $G$  obtained by adding edges to  $G'$ . This is a contradiction. The proof of the converse is similar.

## CHAPTER 11

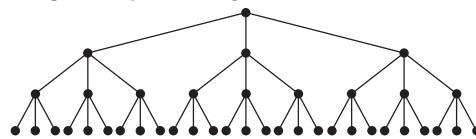
### Section 11.1

- 1.** (a), (c), (e) **3. a)**  $a$  **b)**  $a, b, c, d, f, h, j, q, t$  **c)**  $e, g, i, k, l, m, n, o, p, r, s, u$  **d)**  $q, r$  **e)**  $c$  **f)**  $p$  **g)**  $f, b, a$  **h)**  $e, f, l, m, n$  **5.** No **7.** Level 0:  $a$ ; level 1:  $b, c, d$ ; level 2:  $e$  through  $k$  (in alphabetical order); level 3:  $l$  through  $r$ ; level 4:  $s, t$ ; level 5:  $u$  **9. a)** The entire tree **b)**  $c, g, h, o, p$  and the four edges  $cg, ch, ho, hp$  **c)**  $e$  alone **11. a)** 1 **b)** 2 **13. a)** 3 **b)** 9 **15. a)** The “only if” part is Theorem 2 and the definition of a tree. Suppose  $G$  is a connected simple graph with  $n$  vertices and  $n - 1$  edges. If  $G$  is not a tree, it contains, by Exercise 14, an edge whose removal produces a graph  $G'$ , which is still connected. If  $G'$  is not a tree, remove an edge to produce a connected graph  $G''$ . Repeat this procedure until the result is a tree. This requires at most  $n - 1$  steps because there are only  $n - 1$  edges. By Theorem 2, the resulting graph has  $n - 1$  edges because it has  $n$  vertices. It follows that no edges were deleted, so  $G$  was already a tree. **b)** Suppose that  $G$  is a tree. By part (a),  $G$  has  $n - 1$  edges, and by definition,  $G$  has no simple circuits. Conversely, suppose that  $G$  has no simple circuits and has  $n - 1$  edges. Let  $c$  equal the number of components of  $G$ , each of which is necessarily a tree, say with  $n_i$  vertices, where  $\sum_{i=1}^c n_i = n$ . By part (a), the total number of edges in  $G$  is  $\sum_{i=1}^c (n_i - 1) = n - c$ . Since we are given that this equals  $n - 1$ , it follows that  $c = 1$ , i.e.,  $G$  is connected and therefore satisfies the definition of a tree. **17.** 9999 **19.** 2000 **21.** 999 **23.** 1,000,000 dollars **25.** No such tree exists by Theorem 4 because it is impossible for  $m = 2$  or  $m = 84$ .

**27.** Complete binary tree of height 4:



Complete 3-ary tree of height 3:



**29. a)** By Theorem 3 it follows that  $n = mi + 1$ . Because  $i + l = n$ , we have  $l = n - i$ , so  $l = (mi + 1) - i = (m - 1)i + 1$ .

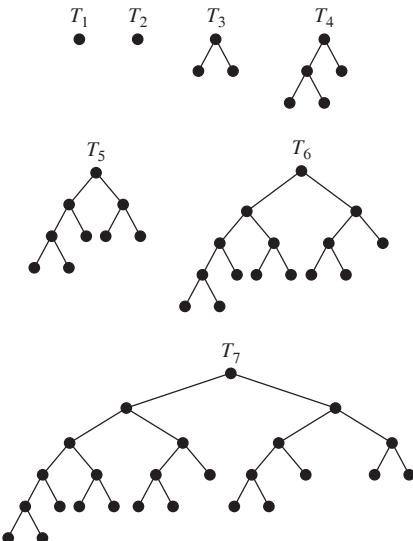
**b)** We have  $n = mi + 1$  and  $i + l = n$ . Hence,  $i = n - l$ . It follows that  $n = m(n - l) + 1$ . Solving for  $n$  gives  $n = (ml - 1)/(m - 1)$ . From  $i = n - l$  we obtain  $i = [(ml - 1)/(m - 1)] - l = (l - 1)/(m - 1)$ . **31.**  $n - t$

**33. a)** 1 **b)** 3 **c)** 5 **35. a)** The parent directory **b)** A subdirectory or contained file **c)** A subdirectory or contained file in the same parent directory **d)** All directories in the path name **e)** All subdirectories and files continued in the directory or a subdirectory of this directory, and so on **f)** The length of the path to this directory or file **g)** The depth of the system, i.e., the length of the longest path

**37.** Let  $n = 2^k$ , where  $k$  is a positive integer. If  $k = 1$ , there is nothing to prove because we can add two numbers with  $n - 1 = 1$  processor in  $\log 2 = 1$  step. Assume we can add  $n = 2^k$  numbers in  $\log n$  steps using a tree-connected network of  $n - 1$  processors. Let  $x_1, x_2, \dots, x_{2n}$  be  $2n = 2^{k+1}$  numbers that we wish to add. The tree-connected network of  $2n - 1$  processors consists of the tree-connected network of  $n - 1$  processors together with two new processors as children of each leaf. In one step we can use the leaves of the larger network to find  $x_1 + x_2, x_3 + x_4, \dots, x_{2n-1} + x_{2n}$ , giving us  $n$  numbers, which, by the inductive hypothesis, we can add in  $\log n$  steps using the rest of the network. Because we have used  $\log n + 1$  steps and  $\log(2n) = \log 2 + \log n = 1 + \log n$ , this completes the proof.

**39. c only** **41. c and h** **43.** Suppose a tree  $T$  has at least two centers. Let  $u$  and  $v$  be distinct centers, both with eccentricity  $e$ , with  $u$  and  $v$  not adjacent. Because  $T$  is connected, there is a simple path  $P$  from  $u$  to  $v$ . Let  $c$  be any other vertex on this path. Because the eccentricity of  $c$  is at least  $e$ , there is a vertex  $w$  such that the unique simple path from  $c$  to  $w$  has length at least  $e$ . Clearly, this path cannot contain both  $u$  and  $v$  or else there would be a simple circuit. In fact, this path from  $c$  to  $w$  leaves  $P$  and does not return to  $P$  once it, possibly, follows part of  $P$  toward either  $u$  or  $v$ . Without loss of generality, assume this path does not follow  $P$  toward  $u$ . Then the path from  $u$  to  $c$  to  $w$  is simple and of length more than  $e$ , a contradiction. Hence,  $u$  and  $v$  are adjacent. Now because any two centers are adjacent, if there were more than two centers,  $T$  would contain  $K_3$ , a simple circuit, as a subgraph, which is a contradiction.

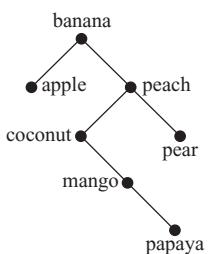
45.



47. The statement is that *every* tree with  $n$  vertices has a path of length  $n - 1$ , and it was shown only that there exists a tree with  $n$  vertices having a path of length  $n - 1$ .

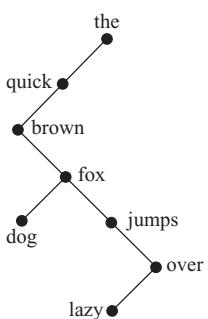
## Section 11.2

1.



3. a) 3 b) 1 c) 4 d) 5

5.

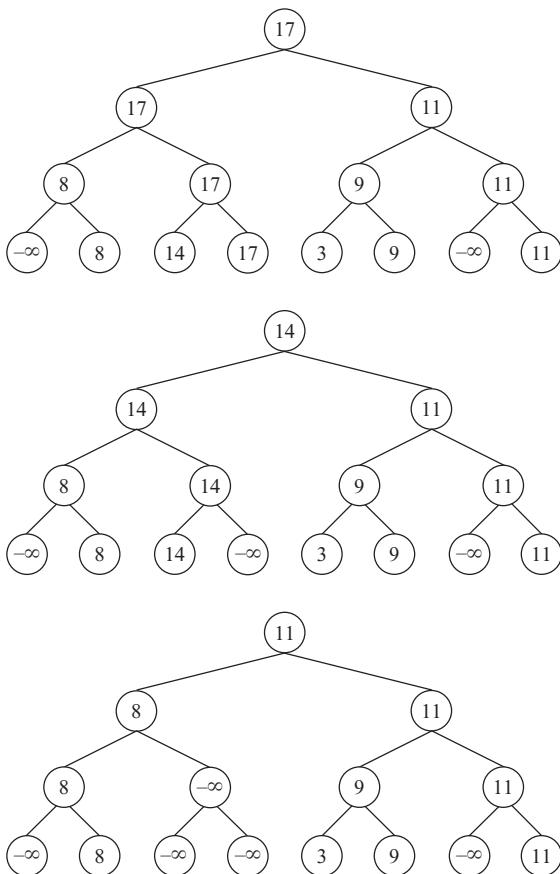


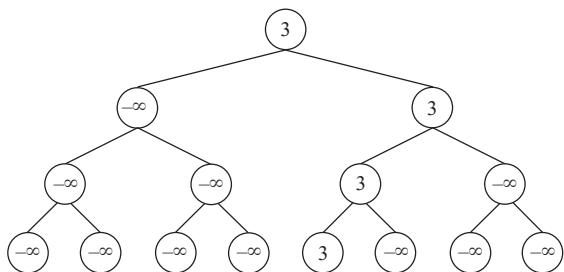
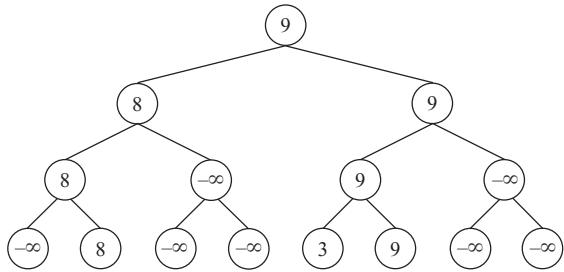
7. At least  $\lceil \log_3 4 \rceil = 2$  weighings are needed, because there are only four outcomes (because it is not required to determine whether the coin is lighter or heavier). In fact, two weighings suffice. Begin by weighing coin 1 against coin 2. If they balance, weigh coin 1 against coin 3. If coin 1 and coin 3 are the same weight, coin 4 is the counterfeit coin, and if they are not the same weight, then coin 3 is the counterfeit coin. If coin 1 and coin 2 are not the same weight, again weigh coin 1 against coin 3. If they balance, coin 2 is the counterfeit coin; if they do not balance, coin 1 is the counterfeit coin. 9. At least

$\lceil \log_3 13 \rceil = 3$  weighings are needed. In fact, three weighings suffice. Start by putting coins 1, 2, and 3 on the left-hand side of the balance and coins 4, 5, and 6 on the right-hand side. If equal, apply Example 3 to coins 1, 2, 7, 8, 9, 10, 11, and 12. If unequal, apply Example 3 to 1, 2, 3, 4, 5, 6, 7, and 8.

11. The least number is five. Call the elements  $a, b, c$ , and  $d$ . First compare  $a$  and  $b$ ; then compare  $c$  and  $d$ . Without loss of generality, assume that  $a < b$  and  $c < d$ . Next compare  $a$  and  $c$ . Whichever is smaller is the smallest element of the set. Again without loss of generality, suppose  $a < c$ . Finally, compare  $b$  with both  $c$  and  $d$  to completely determine the ordering.

13. The first two steps are shown in the text. After 22 has been identified as the second largest element, we replace the leaf 22 by  $-\infty$  in the tree and recalculate the winner in the path from the leaf where 22 used to be up to the root. Next, we see that 17 is the third largest element, so we repeat the process: replace the leaf 17 by  $-\infty$  and recalculate. Next, we see that 14 is the fourth largest element, so we repeat the process: replace the leaf 14 by  $-\infty$  and recalculate. Next, we see that 11 is the fifth largest element, so we repeat the process: replace the leaf 11 by  $-\infty$  and recalculate. The process continues in this manner. We determine that 9 is the sixth largest element, 8 is the seventh largest element, and 3 is the eighth largest element. The trees produced in all steps, except the second to last, are shown here.





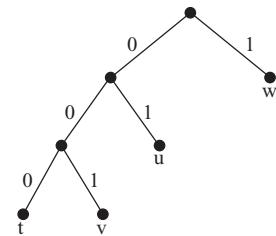
**15.** The value of a vertex is the list element currently there, and the label is the name (i.e., location) of the leaf responsible for that value.

```

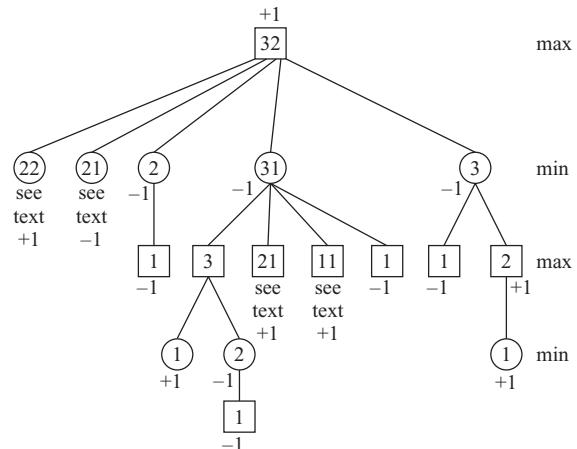
procedure tournament sort(a_1, \dots, a_n)
 $k := \lceil \log n \rceil$
build a binary tree of height k
for $i := 1$ to n
 set the value of the i th leaf to be a_i and its label to
 be itself
for $i := n + 1$ to 2^k
 set the value of the i th leaf to be $-\infty$ and its label to
 be itself
for $i := k - 1$ downto 0
 for each vertex v at level i
 set the value of v to the larger of the values of its
 children and its label to be the label of the child
 with the larger value
for $i := 1$ to n
 $c_i :=$ value at the root
 let v be the label of the root
 set the value of v to be $-\infty$
 while the label at the root is still v
 $v := parent(v)$
 set the value of v to the larger of the values of its
 children and its label to be the label of the child
 with the larger value
{ c_1, \dots, c_n is the list in nonincreasing order}

7. $k - 1$, where $n = 2^k$ 19. a) Yes b) No c) Yes d) No
21. $a: 000, e: 001, i: 01, k: 1100, o: 1101, p: 11110, u: 11101$
23. a: 11; b: 101; c: 100; d: 01; e: 00; 2.25 bits (Note: the
 decoding depends on how ties are broken, but the average number
 of bits is always the same.) 25. There are four possible
 answers in all, the one shown here and three more obtainable
 from this one by swapping t and v and/or swapping u and v.

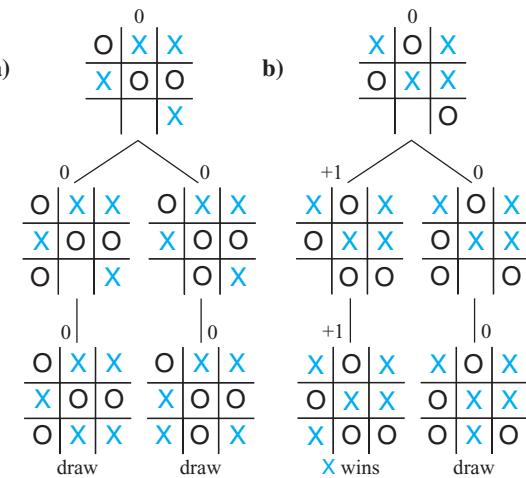
```

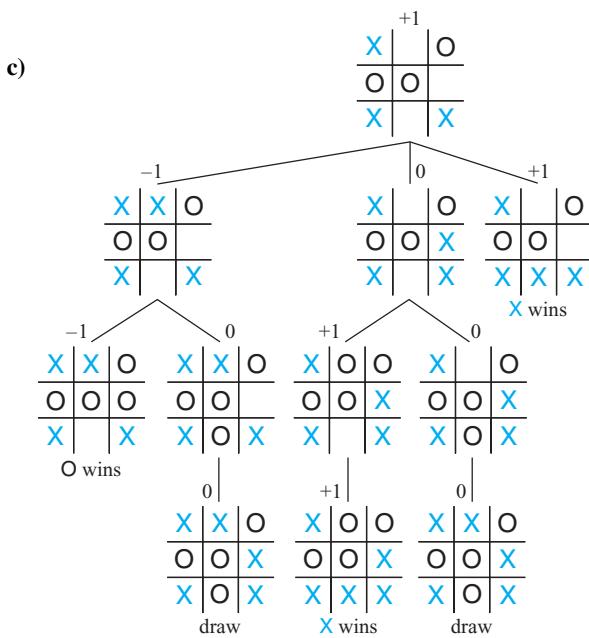


27. A:0001; B:101001; C:11001; D:00000; E:100;  
 F:001100; G:001101; H:0101; I:0100; J:110100101;  
 K:1101000; L:00001; M:10101; N:0110; O:0010; P:101000;  
 Q:1101001000; R:1011; S:0111; T:111; U:00111; V:110101;  
 W:11000; X:11010011; Y:11011; Z:1101001001    29. A:2;  
 E:1; N:010; R:011; T:02; Z:00    31. n    33. Because the tree  
 is rather large, we have indicated in some places to “see text.”  
 Refer to Figure 9; the subtree rooted at these square or circle  
 vertices is exactly the same as the corresponding subtree in  
 Figure 9. First player wins.

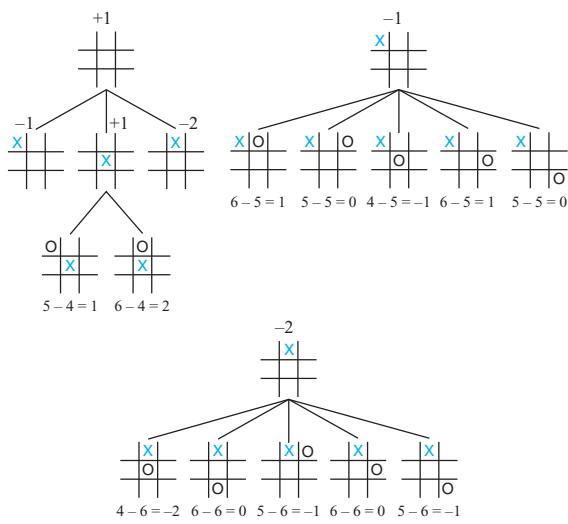


- 35.** a) \$1 b) \$3 c)  $-\$3$    **37.** See the figures shown next.  
**a)** 0   **b)** 0   **c)** 1   **d)** This position cannot have occurred in a game; this picture is impossible.



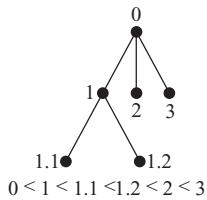


**39.** Proof by strong induction: *Basis step:* When there are  $n = 2$  stones in each pile, if first player takes two stones from a pile, then second player takes one stone from the remaining pile and wins. If first player takes one stone from a pile, then second player takes two stones from the other pile and wins. *Inductive step:* Assume inductive hypothesis that second player can always win if the game starts with two piles of  $j$  stones for all  $2 \leq j \leq k$ , where  $k \geq 2$ , and consider a game with two piles containing  $k+1$  stones each. If first player takes all the stones from one of the piles, then second player takes all but one stone from the remaining pile and wins. If first player takes all but one stone from one of the piles, then second player takes all the stones from the other pile and wins. Otherwise first player leaves  $j$  stones in one pile, where  $2 \leq j \leq k$ , and  $k+1$  stones in the other pile. Second player takes the same number of stones from the larger pile, also leaving  $j$  stones there. At this point the game consists of two piles of  $j$  stones each. By the inductive hypothesis, the second player in that game, who is also the second player in our actual game, can win, and the proof by strong induction is complete. **41.** 7; 49 **43.** Value of tree is 1. Note: The second and third trees are the subtrees of the two children of the root in the first tree whose subtrees are not shown because of space limitations. They should be thought of as spliced into the first picture.

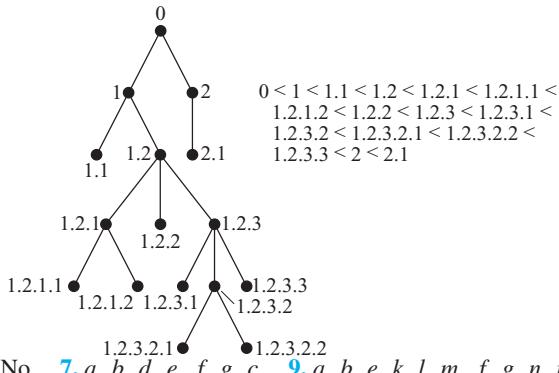


### Section 11.3

1.

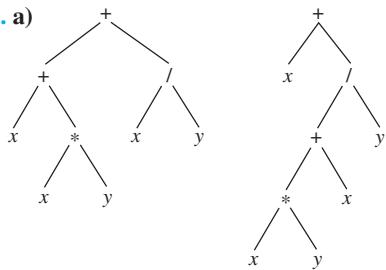


3.



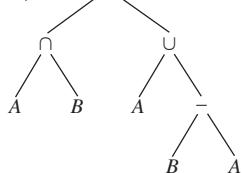
5. No **7.** a, b, d, e, f, g, c **9.** a, b, e, k, l, m, f, g, n, r, s, c, d, h, o, i, j, p, q **11.** d, b, i, e, m, j, n, o, a, f, c, g, k, h, p, l **13.** d, f, g, e, b, c, a **15.** k, l, m, e, f, r, s, n, g, b, c, o, h, i, p, q, j, d, a

17. a)



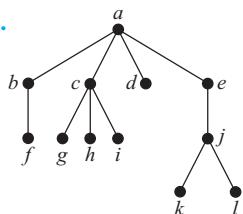
- b)**  $\text{++}x\ast xy/xy, +x/+*xyxy$  **c)**  $xx*y+xy/+xxy*x+y/+$   
**d)**  $((x + (x * y)) + (x/y)), (x + ((x * y) + x)/y))$

19. a)



- b)**  $- \cap A B \cup A - B A$  **c)**  $A B \cap A B A - \cup -$   
**d)**  $((A \cap B) - (A \cup (B - A)))$  **21.** 14 **23.** a) 1 b) 1 c) 4  
**d)** 2205

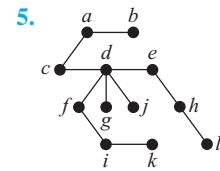
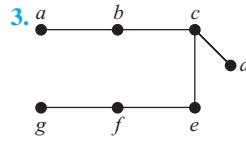
25.



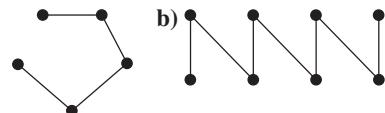
**27.** Use mathematical induction. The result is trivial for a list with one element. Assume the result is true for a list with  $n$  elements. For the inductive step, start at the end. Find the sequence of vertices at the end of the list starting with the last leaf, ending with the root, each vertex being the last child of the one following it. Remove this leaf and apply the inductive hypothesis. **29.** c, d, b, f, g, h, e, a in each case

**31.** Proof by mathematical induction. Let  $S(X)$  and  $O(X)$  represent the number of symbols and number of operators in the well-formed formula  $X$ , respectively. The statement is true for well-formed formulae of length 1, because they have 1 symbol and 0 operators. Assume the statement is true for all well-formed formulae of length less than  $n$ . A well-formed formula of length  $n$  must be of the form  $*XY$ , where  $*$  is an operator and  $X$  and  $Y$  are well-formed formulae of length less than  $n$ . Then by the inductive hypothesis  $S(*XY) = S(X) + S(Y) = [O(X) + 1] + [O(Y) + 1] = O(X) + O(Y) + 2$ . Because  $O(*XY) = 1 + O(X) + O(Y)$ , it follows that  $S(*XY) = O(*XY) + 1$ . **33.**  $x y + z x \circ + x \circ, xyz ++ yx ++, xyxyooyyoozo+, xz \times, zz+\circ, yyyyooo, zx+yz+\circ$ , for instance

### Section 11.4

1.  $m - n + 1$ 

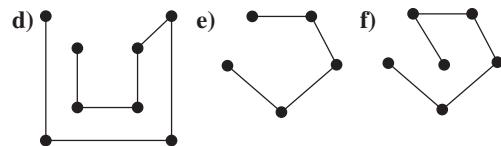
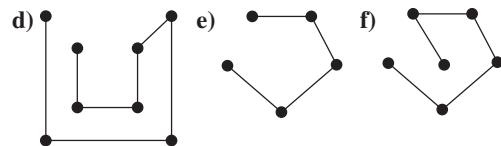
7. a)



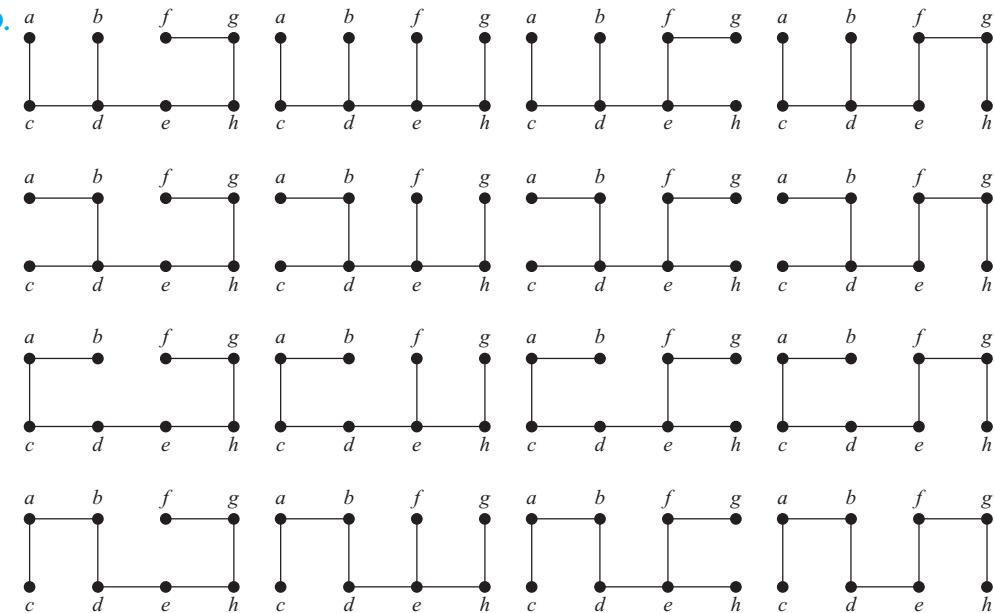
c)



d)

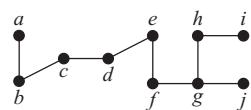


**9.**

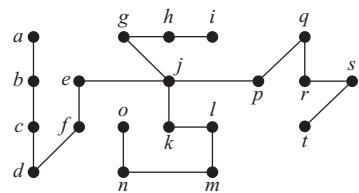


**11.** a) 3 b) 16 c) 4 d) 5

**13.**



**15.**



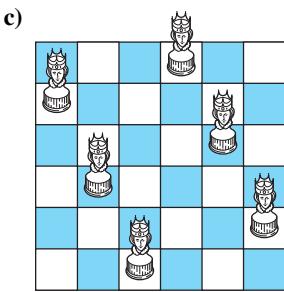
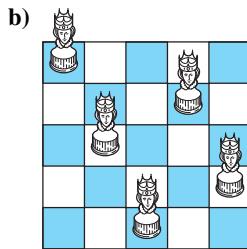
- 17.** a) A path of length 6 b) A path of length 5 c) A path of length 6 d) Depends on order chosen to visit the vertices; may be a path of length 7
- 19.** With breadth-first search, the initial vertex is the middle vertex, and the  $n$  spokes are added to the tree as this vertex is processed. Thus, the resulting tree is  $K_{1,n}$ . With depth-first search, we start at the vertex in the middle of the wheel and visit a neighbor—one of the vertices on the rim. From there we move to an adjacent vertex on the rim, and so on all the way around until we have reached every vertex. Thus, the resulting spanning tree is a path of length  $n$ .
- 21.** With breadth-first search, we fan out from a vertex of degree  $m$  to all the vertices of degree  $n$  as the first step. Next, a vertex of degree  $n$  is processed, and the edges from it to all

the remaining vertices of degree  $m$  are added. The result is a  $K_{1,n-1}$  and a  $K_{1,m-1}$  with their centers joined by an edge. With depth-first search, we travel back and forth from one partite set to the other until we can go no further. If  $m = n$  or  $m = n - 1$ , then we get a path of length  $m + n - 1$ . Otherwise, the path ends while some vertices in the larger partite set have not been visited, so we back up one link in the path to a vertex  $v$  and then successively visit the remaining vertices in that set from  $v$ . The result is a path with extra pendant edges coming out of one end of the path.

**23.** A possible set of flights to discontinue are: Boston–New York, Detroit–Boston, Boston–Washington, New York–Washington, New York–Chicago, Atlanta–Washington, Atlanta–Dallas, Atlanta–Los Angeles, Atlanta–St. Louis, St. Louis–Dallas, St. Louis–Detroit, St. Louis–Denver, Dallas–San Diego, Dallas–Los Angeles, Dallas–San Francisco, San Diego–Los Angeles, Los Angeles–San Francisco, San Francisco–Seattle.

**25.** Proof by induction on the length of the path: If the path has length 0, then the result is trivial. If the length is 1, then  $u$  is adjacent to  $v$ , so  $u$  is at level 1 in the breadth-first spanning tree. Assume that the result is true for paths of length  $l$ . If the length of a path is  $l + 1$ , let  $u'$  be the next-to-last vertex in a shortest path from  $v$  to  $u$ . By the inductive hypothesis,  $u'$  is at level  $l$  in the breadth-first spanning tree. If  $u$  were at a level not exceeding  $l$ , then clearly the length of the shortest path from  $v$  to  $u$  would also not exceed  $l$ . So  $u$  has not been added to the breadth-first spanning tree yet after the vertices of level  $l$  have been added. Because  $u$  is adjacent to  $u'$ , it will be added at level  $l + 1$  (although the edge connecting  $u'$  and  $u$  is not necessarily added).

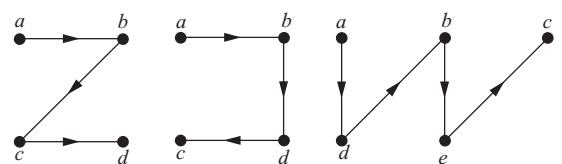
**27.** a) No solution



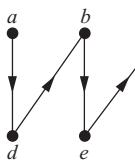
- 29.** Start at a vertex and proceed along a path without repeating vertices as long as possible, allowing the return to the start after all vertices have been visited. When it is impossible to continue along a path, backtrack and try another extension of the current path. **31.** Take the union of the spanning trees of the connected components of  $G$ . They are disjoint, so the result is a forest. **33.**  $m - n + c$  **35.** Assume that we wish to find the length of a shortest path from  $v_1$  to every other vertex of  $G$  using Algorithm 1. In line 2 of that algorithm, add  $L(v_1) := 0$ , and add the following as a third step in the **then** clause at the end:  $L(w) := 1 + L(v)$ . **37.** Add an instruction to the BFS algorithm to mark each vertex as it is encountered. When BFS terminates we have found (all the vertices of) one component of the graph. Repeat, starting at an unmarked vertex, and continue in this way until all vertices have been marked. **39.** Trees **41.** Use depth-first search on each component. **43.** If an edge  $uv$  is not followed while we are processing vertex  $u$  during the depth-first search process, then it must be the case that the vertex  $v$  had already been visited. There are two cases. If vertex  $v$  was visited after we started processing  $u$ , then, because we are not finished processing  $u$  yet,  $v$  must appear in the subtree rooted at  $u$  (and hence, must be a descendant of  $u$ ). On the other hand, if the processing of  $v$  had already begun before we started processing  $u$ , then why wasn't this edge followed at that time? It must be that we had not finished processing  $v$ , in other words, that we are still forming the subtree rooted at  $v$ , so  $u$  is a descendant of  $v$ , and hence,  $v$  is an ancestor of  $u$ . **45.** Certainly these two procedures produce the identical spanning trees if the graph we are working with is a tree itself, because in this case there is only one spanning tree (the whole graph). This is the only case in which that happens, however. If the original graph has any other edges, then by Exercise 43 they must be back edges and hence, join a vertex to an ancestor or descendant, whereas by Exercise 34, they must connect vertices at the same level or at levels that differ by 1. Clearly these two possibilities are mutually exclusive. Therefore there can be no edges other than tree edges if the two spanning trees are to be the same. **47.** Because the edges not in the spanning tree are not followed in the process, we can ignore them. Thus we can assume that the graph was a rooted tree to begin with. The basis step is trivial (there is only one vertex), so we assume the inductive hypothesis that breadth-first search applied to trees with  $n$  vertices have their vertices visited in order of their level in the tree and consider a tree  $T$  with  $n + 1$  vertices. The last vertex to be visited during breadth-first search of this tree, say  $v$ , is

the one that was added last to the list of vertices waiting to be processed. It was added when its parent, say  $u$ , was being processed. We must show that  $v$  is at the lowest (bottom-most, i.e., numerically greatest) level of the tree. Suppose not; say vertex  $x$ , whose parent is vertex  $w$ , is at a lower level. Then  $w$  is at a lower level than  $u$ . Clearly  $v$  must be a leaf, because any child of  $v$  could not have been seen before  $v$  is seen. Consider the tree  $T'$  obtained from  $T$  by deleting  $v$ . By the inductive hypothesis, the vertices in  $T'$  must be processed in order of their level in  $T'$  (which is the same as their level in  $T$ , and the absence of  $v$  in  $T'$  has no effect on the rest of the algorithm). Therefore  $u$  must have been processed before  $w$ , and therefore  $v$  would have joined the waiting list before  $x$  did, a contradiction. Therefore  $v$  is at the bottom-most level of the tree, and the proof is complete. **49.** We modify the pseudocode given in Algorithm 2 by initializing  $m$  to be 0 at the beginning of the algorithm, and adding the statements " $m := m + 1$ " and "assign  $m$  to vertex  $v$ " after the statement that removes vertex  $v$  from  $L$ . **51.** If a directed edge  $uv$  is not followed while we are processing its tail  $u$  during the depth-first search process, then it must be the case that its head  $v$  had already been visited. There are three cases. If vertex  $v$  was visited after we started processing  $u$ , then, because we are not finished processing  $u$  yet,  $v$  must appear in the subtree rooted at  $u$  (and hence, must be a descendant of  $u$ ), so we have a forward edge. Otherwise, the processing of  $v$  must have already begun before we started processing  $u$ . If it had not yet finished (i.e., we are still forming the subtree rooted at  $v$ ), then  $u$  is a descendant of  $v$ , and hence,  $v$  is an ancestor of  $u$  (we have a back edge). Finally, if the processing of  $v$  had already finished, then by definition we have a cross edge. **53.** Let  $T$  be the spanning tree constructed in Figure 3 and  $T_1, T_2, T_3$ , and  $T_4$  the spanning trees in Figure 4. Denote by  $d(T', T'')$  the distance between  $T'$  and  $T''$ . Then  $d(T, T_1) = 6$ ,  $d(T, T_2) = 4$ ,  $d(T, T_3) = 4$ ,  $d(T, T_4) = 2$ ,  $d(T_1, T_2) = 4$ ,  $d(T_1, T_3) = 4$ ,  $d(T_1, T_4) = 6$ ,  $d(T_2, T_3) = 4$ ,  $d(T_2, T_4) = 2$ , and  $d(T_3, T_4) = 4$ . **55.** Suppose  $e_1 = \{u, v\}$  is as specified. Then  $T_2 \cup \{e_1\}$  contains a simple circuit  $C$  containing  $e_1$ . The graph  $T_1 - \{e_1\}$  has two connected components; the endpoints of  $e_1$  are in different components. Travel  $C$  from  $u$  in the direction opposite to  $e_1$  until you come to the first vertex in the same component as  $v$ . The edge just crossed is  $e_2$ . Clearly,  $T_2 \cup \{e_1\} - \{e_2\}$  is a tree, because  $e_2$  was on  $C$ . Also  $T_1 - \{e_1\} \cup \{e_2\}$  is a tree, because  $e_2$  reunited the two components.

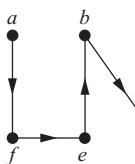
**57.** Exercise 18:      **Exercise 19:**      **Exercise 20:**



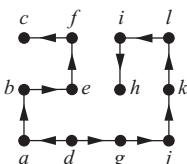
### Exercise 21:



### Exercise 22:



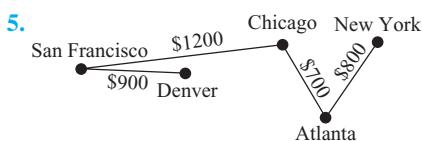
### Exercise 23:



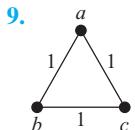
- 59.** First construct an Euler circuit in the directed graph. Then delete from this circuit every edge that goes to a vertex previously visited. **61.** According to Exercise 60, a directed graph contains a circuit if and only if there are any back edges. We can detect back edges as follows. Add a marker on each vertex  $v$  to indicate what its status is: not yet seen (the initial situation), seen (i.e., put into  $T$ ) but not yet finished (i.e.,  $visit(v)$  has not yet terminated), or finished (i.e.,  $visit(v)$  has terminated). A few extra lines in Algorithm 1 will accomplish this bookkeeping. Then to determine whether a directed graph has a circuit, we just have to check when looking at edge  $uv$  whether the status of  $v$  is “seen.” If that ever happens, then we know there is a circuit; if not, then there is no circuit.

## Section 11.5

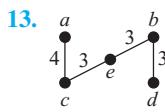
- 1.** Deep Springs–Oasis, Oasis–Dyer, Oasis–Silver Peak, Silver Peak–Goldfield, Lida–Gold Point, Gold Point–Beatty, Lida–Goldfield, Goldfield–Tonopah, Tonopah–Manhattan, Tonopah–Warm Springs   **3.**  $\{e, f\}$ ,  $\{c, f\}$ ,  $\{e, h\}$ ,  $\{h, i\}$ ,  $\{b, c\}$ ,  $\{b, d\}$ ,  $\{a, d\}$ ,  $\{g, h\}$



- $$7. \{e, f\}, \{a, d\}, \{h, i\}, \{b, d\}, \{c, f\}, \{e, h\}, \{b, c\}, \{g, h\}$$



- 11.** Instead of choosing minimum-weight edges at each stage, choose maximum-weight edges at each stage with the same properties.

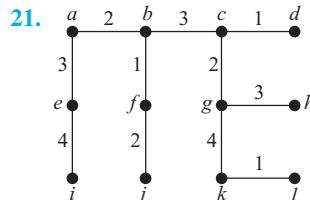


- 

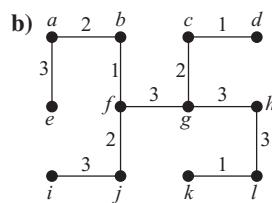
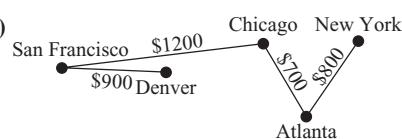
- 17.** First find a minimum spanning tree  $T$  of the graph  $G$  with  $n$  edges. Then for  $i = 1$  to  $n - 1$ , delete only the  $i$ th edge of  $T$  from  $G$  and find a minimum spanning tree of the remaining

graph. Pick the one of these  $n - 1$  trees with the shortest length.

- 19.** If all edges have different weights, then a contradiction is obtained in the proof that Prim's algorithm works when an edge  $e_{k+1}$  is added to  $T$  and an edge  $e$  is deleted, instead of possibly producing another spanning tree.



- 23.** Same as Kruskal's algorithm, except start with  $T :=$  this set of edges and iterate from  $i = 1$  to  $i = n - 1 - s$ , where  $s$  is the number of edges you start with.



- 27.** By Exercise 24, at each stage of Sollin's algorithm a forest results. Hence, after  $n - 1$  edges are chosen, a tree results. It remains to show that this tree is a minimum spanning tree. Let  $T$  be a minimum spanning tree with as many edges in common with Sollin's tree  $S$  as possible. If  $T \neq S$ , then there is an edge  $e \in S - T$  added at some stage in the algorithm, where prior to that stage all edges in  $S$  are also in  $T$ .  $T \cup \{e\}$  contains a unique simple circuit. Find an edge  $e' \in S - T$  and an edge  $e'' \in T - S$  on this circuit and "adjacent" when viewing the trees of this stage as "supervertices." Then by the algorithm,  $w(e') \leq w(e'')$ . So replace  $T$  by  $T - \{e''\} \cup \{e'\}$  to produce a minimum spanning tree closer to  $S$  than  $T$  was.

- 29.** Each of the  $r$  trees is joined to at least one other tree by a new edge. Hence, there are at most  $r/2$  trees in the result (each new tree contains two or more old trees). To accomplish this, we need to add  $r - (r/2) = r/2$  edges. Because the number of edges added is integral, it is at least  $\lceil r/2 \rceil$ . **31.** If  $k \geq \log n$ , then  $n/2^k \leq 1$ , so  $\lceil n/2^k \rceil = 1$ , so by Exercise 30 the algorithm is finished after at most  $\log n$  iterations.

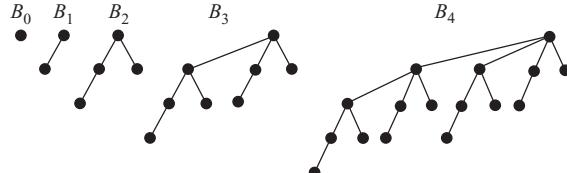
33. Suppose that a minimum spanning tree  $T$  contains edge  $e = uv$  that is the maximum weight edge in simple circuit  $C$ . Delete  $e$  from  $T$ . This creates a forest with two components, one containing  $u$  and the other containing  $v$ . Follow the edges of the path  $C - e$ , starting at  $u$ . At some point this path must jump from the component of  $T - e$  containing  $u$  to the component of  $T - e$  containing  $v$ , say using edge  $f$ . This edge cannot be in  $T$ , because  $e$  can be the only edge of  $T$  joining the two components (otherwise there would be a simple circuit in  $T$ ). Because  $e$  is the edge of greatest weight in  $C$ , the

weight of  $f$  is smaller. The tree formed by replacing  $e$  by  $f$  in  $T$  therefore has smaller weight, a contradiction. **35.** The reverse-delete algorithm must terminate and produce a spanning tree, because the algorithm never disconnects the graph and upon termination there can be no more simple circuits. The edge deleted at each stage of the algorithm must have been the edge of maximum weight in whatever circuits it was a part of. Therefore by Exercise 33 it cannot be in any minimum spanning tree. Since only edges that could not have been in any minimum spanning tree have been deleted, the result must be a minimum spanning tree.

### Supplementary Exercises

**1.** Suppose  $T$  is a tree. Then clearly  $T$  has no simple circuits. If we add an edge  $e$  connecting two nonadjacent vertices  $u$  and  $v$ , then obviously a simple circuit is formed, because when  $e$  is added to  $T$  the resulting graph has too many edges to be a tree. The only simple circuit formed is made up of the edge  $e$  together with the unique path in  $T$  from  $v$  to  $u$ . Suppose  $T$  satisfies the given conditions. All that is needed is to show that  $T$  is connected, because there are no simple circuits in the graph. Assume that  $T$  is not connected. Then let  $u$  and  $v$  be in separate connected components. Adding  $e = \{u, v\}$  does not satisfy the conditions. **3.** Suppose that a tree  $T$  has  $n$  vertices of degrees  $d_1, d_2, \dots, d_n$ , respectively. Because  $2e = \sum_{i=1}^n d_i$  and  $e = n - 1$ , we have  $2(n - 1) = \sum_{i=1}^n d_i$ . Because each  $d_i \geq 1$ , it follows that  $2(n - 1) = n + \sum_{i=1}^n (d_i - 1)$ , or that  $n - 2 = \sum_{i=1}^n (d_i - 1)$ . Hence, at most  $n - 2$  of the terms of this sum can be 1 or more. Hence, at least two of them are 0. It follows that  $d_i = 1$  for at least two values of  $i$ . **5.**  $2n - 2$  **7.** A tree has no circuits, so it cannot have a subgraph homeomorphic to  $K_{3,3}$  or  $K_5$ . **9.** Color each connected component separately. For each of these connected components, first root the tree, then color all vertices at even levels red and all vertices at odd levels blue. **11.** Upper bound:  $k^h$ ; lower bound:  $2 \lceil k / 2 \rceil^{h-1}$

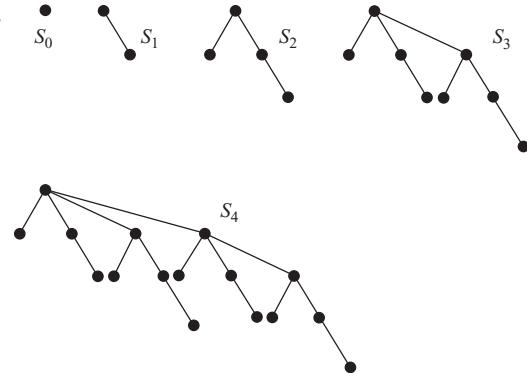
**13.**



**15.** Because  $B_{k+1}$  is formed from two copies of  $B_k$ , one shifted down one level, the height increases by 1 as  $k$  increases by 1. Because  $B_0$  had height 0, it follows by induction that  $B_k$  has height  $k$ . **17.** Because the root of  $B_{k+1}$  is the root of  $B_k$  with one additional child (namely the root of the other  $B_k$ ), the degree of the root increases by 1 as  $k$  increases by 1. Be-

cause  $B_0$  had a root with degree 0, it follows by induction that  $B_k$  has a root with degree  $k$ .

**19.**



**21.** Use mathematical induction. The result is trivial for  $k = 0$ . Suppose it is true for  $k - 1$ .  $T_{k-1}$  is the parent tree for  $T$ . By induction, the child tree for  $T$  can be obtained from  $T_0, \dots, T_{k-2}$  in the manner stated. The final connection of  $r_{k-2}$  to  $r_{k-1}$  is as stated in the definition of  $S_k$ -tree.

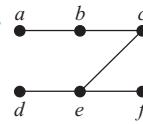
**23. procedure** level( $T$ : ordered rooted tree with root  $r$ )

```
queue := sequence consisting of just the root r
while queue contains at least one term
 v := first vertex in queue
 list v
 remove v from queue and put children of v onto
 the end of queue
```

**25.** Build the tree by inserting a root for the address 0, and then inserting a subtree for each vertex labeled  $i$ , for  $i$  a positive integer, built up from subtrees for each vertex labeled  $i.j$  for  $j$  a positive integer, and so on. **27. a)** Yes **b)** No **c)** Yes

**29.** The resulting graph has no edge that is in more than one simple circuit of the type described. Hence, it is a cactus.

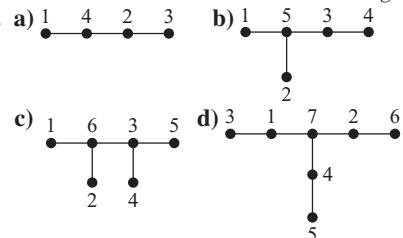
**31.**



**33.**



**35.**



**37. 6**

**39. a)** 0 for 00, 11 for 01, 100 for 10, 101 for 11 (exact coding depends on how ties were broken, but all versions are equivalent);  $0.645n$  for string of length  $n$  **b)** 0 for 000, 100 for 001, 101 for 010, 110 for 100, 11100 for 011, 11101 for 101, 11110 for 110, 11111 for 111 (exact coding depends on how ties were broken, but all versions are equivalent);  $0.532\bar{6}n$  for string of length  $n$  **41.** Let  $G'$  be the graph obtained by delet-

ing from  $G$  the vertex  $v$  and all edges incident to  $v$ . A minimum spanning tree of  $G$  can be obtained by taking an edge of minimal weight incident to  $v$  together with a minimum spanning tree of  $G'$ . **43.** Suppose that edge  $e$  is the edge of least weight incident to vertex  $v$ , and suppose that  $T$  is a spanning tree that does not include  $e$ . Add  $e$  to  $T$ , and delete from the simple circuit formed thereby the other edge of the circuit that contains  $v$ . The result will be a spanning tree of strictly smaller weight (because the deleted edge has weight greater than the weight of  $e$ ). This is a contradiction, so  $T$  must include  $e$ . **45.** Because paths in trees are unique, an arborescence  $T$  of a directed graph  $G$  is just a subgraph of  $G$  that is a tree rooted at  $r$ , containing all the vertices of  $G$ , with all the edges directed away from the root. Thus the in-degree of each vertex other than  $r$  is 1. For the converse, it is enough to show that for each  $v \in V$  there is a unique directed path from  $r$  to  $v$ . Because the in-degree of each vertex other than  $r$  is 1, we can follow the edges of  $T$  backwards from  $v$ . This path can never return to a previously visited vertex, because that would create a simple circuit. Therefore the path must eventually stop, and it can stop only at  $r$ , whose in-degree is not necessarily 1. Following this path forward gives the path from  $r$  to  $v$  required by the definition of arborescence. **47. a)** Run the breadth-first search algorithm, starting from  $v$  and respecting the directions of the edges, marking each vertex encountered as reachable. **b)** Running breadth-first search on  $G^{conv}$ , again starting at  $v$ , respecting the directions of the edges, and marking each vertex encountered, will identify all the vertices from which  $v$  is reachable. **c)** Choose a vertex  $v_1$  and using parts (a) and (b) find the strong component containing  $v_1$ , namely all vertices  $w$  such that  $w$  is reachable from  $v_1$  and  $v_1$  is reachable from  $w$ . Then choose another vertex  $v_2$  not yet in a strong component and find the strong component of  $v_2$ . Repeat until all vertices have been included. The correctness of this algorithm follows from the definition of strong component and Exercise 17 in Section 10.4.

## CHAPTER 12

### Section 12.1

**1. a)** 1   **b)** 1   **c)** 0   **d)** 0   **3. a)**  $(1 \cdot 1) + (\bar{0} \cdot \bar{1} + 0) = 1 + (\bar{0} + 0) = 1 + (1 + 0) = 1 + 1 = 1$    **b)**  $(\mathbf{T} \wedge \mathbf{T}) \vee (\neg(\mathbf{F} \wedge \mathbf{T}) \vee \mathbf{F}) \equiv \mathbf{T}$

**5. a)**

| $x$ | $y$ | $z$ | $\bar{x}y$ |
|-----|-----|-----|------------|
| 1   | 1   | 1   | 0          |
| 1   | 1   | 0   | 0          |
| 1   | 0   | 1   | 0          |
| 1   | 0   | 0   | 0          |
| 0   | 1   | 1   | 1          |
| 0   | 1   | 0   | 1          |
| 0   | 0   | 1   | 0          |
| 0   | 0   | 0   | 0          |

**b)**

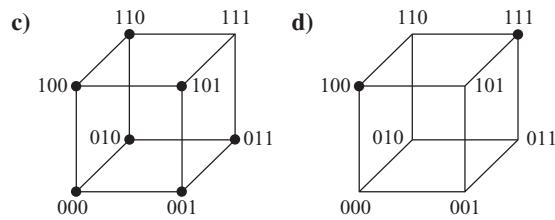
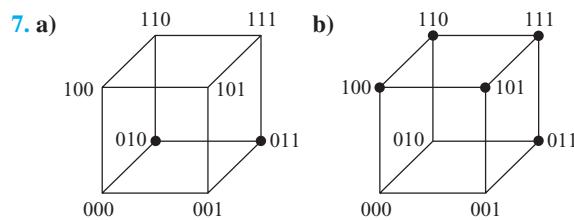
| $x$ | $y$ | $z$ | $x + yz$ |
|-----|-----|-----|----------|
| 1   | 1   | 1   | 1        |
| 1   | 1   | 0   | 1        |
| 1   | 0   | 1   | 1        |
| 1   | 0   | 0   | 1        |
| 0   | 1   | 1   | 1        |
| 0   | 1   | 0   | 0        |
| 0   | 0   | 1   | 0        |
| 0   | 0   | 0   | 0        |

**c)**

| $x$ | $y$ | $z$ | $x\bar{y} + \bar{x}yz$ |
|-----|-----|-----|------------------------|
| 1   | 1   | 1   | 0                      |
| 1   | 1   | 0   | 1                      |
| 1   | 0   | 1   | 1                      |
| 1   | 0   | 0   | 1                      |
| 0   | 1   | 1   | 1                      |
| 0   | 1   | 0   | 1                      |
| 0   | 0   | 1   | 1                      |
| 0   | 0   | 0   | 1                      |

**d)**

| $x$ | $y$ | $z$ | $x(yz + \bar{y}\bar{z})$ |
|-----|-----|-----|--------------------------|
| 1   | 1   | 1   | 1                        |
| 1   | 1   | 0   | 0                        |
| 1   | 0   | 1   | 0                        |
| 1   | 0   | 0   | 1                        |
| 0   | 1   | 1   | 0                        |
| 0   | 1   | 0   | 0                        |
| 0   | 0   | 1   | 0                        |
| 0   | 0   | 0   | 0                        |



**9.** (0, 0) and (1, 1)   **11.**  $x + xy = x \cdot 1 + xy = x(1 + y) = x(y + 1) = x \cdot 1 = x$

**13.**

| $x$ | $y$ | $z$ | $x\bar{y}$ | $y\bar{z}$ | $\bar{x}z$ | $x\bar{y} + y\bar{z} + \bar{x}z$ | $\bar{x}y$ | $\bar{y}z$ | $x\bar{z}$ | $\bar{x}y + \bar{y}z + x\bar{z}$ |
|-----|-----|-----|------------|------------|------------|----------------------------------|------------|------------|------------|----------------------------------|
| 1   | 1   | 1   | 0          | 0          | 0          | 0                                | 0          | 0          | 0          | 0                                |
| 1   | 1   | 0   | 0          | 1          | 0          | 1                                | 0          | 0          | 1          | 1                                |
| 1   | 0   | 1   | 1          | 0          | 0          | 1                                | 0          | 1          | 0          | 1                                |
| 1   | 0   | 0   | 1          | 0          | 0          | 1                                | 0          | 0          | 1          | 1                                |
| 0   | 1   | 1   | 0          | 0          | 1          | 1                                | 1          | 0          | 0          | 1                                |
| 0   | 1   | 0   | 0          | 1          | 0          | 1                                | 1          | 0          | 0          | 1                                |
| 0   | 0   | 1   | 0          | 0          | 1          | 1                                | 0          | 1          | 0          | 1                                |
| 0   | 0   | 0   | 0          | 0          | 0          | 0                                | 0          | 0          | 0          | 0                                |

**15.**

| $x$ | $x + x$ | $x \cdot x$ |
|-----|---------|-------------|
| 0   | 0       | 0           |
| 1   | 1       | 1           |

**17.**

| $x$ | $x + 1$ | $x \cdot 0$ |
|-----|---------|-------------|
| 0   | 1       | 0           |
| 1   | 1       | 0           |

19.

| $x$ | $y$ | $z$ | $y + z$ | $x + (y + z)$ | $x + y$ | $(x + y) + z$ | $yz$ | $x(yz)$ | $xy$ | $(xy)z$ |
|-----|-----|-----|---------|---------------|---------|---------------|------|---------|------|---------|
| 1   | 1   | 1   | 1       | 1             | 1       | 1             | 1    | 1       | 1    | 1       |
| 1   | 1   | 0   | 1       | 1             | 1       | 1             | 0    | 0       | 1    | 0       |
| 1   | 0   | 1   | 1       | 1             | 1       | 1             | 0    | 0       | 0    | 0       |
| 1   | 0   | 0   | 0       | 1             | 1       | 1             | 0    | 0       | 0    | 0       |
| 0   | 1   | 1   | 1       | 1             | 1       | 1             | 1    | 0       | 0    | 0       |
| 0   | 1   | 0   | 1       | 1             | 1       | 1             | 0    | 0       | 0    | 0       |
| 0   | 0   | 1   | 1       | 1             | 0       | 1             | 0    | 0       | 0    | 0       |
| 0   | 0   | 0   | 0       | 0             | 0       | 0             | 0    | 0       | 0    | 0       |

21.

| $x$ | $y$ | $xy$ | $\overline{(xy)}$ | $\bar{x}$ | $\bar{y}$ | $\bar{x} + \bar{y}$ | $x + y$ | $\overline{(x + y)}$ | $\bar{x}\bar{y}$ |
|-----|-----|------|-------------------|-----------|-----------|---------------------|---------|----------------------|------------------|
| 1   | 1   | 1    | 0                 | 0         | 0         | 0                   | 1       | 0                    | 0                |
| 1   | 0   | 0    | 1                 | 0         | 1         | 1                   | 1       | 0                    | 0                |
| 0   | 1   | 0    | 1                 | 1         | 0         | 1                   | 1       | 0                    | 0                |
| 0   | 0   | 0    | 1                 | 1         | 1         | 1                   | 0       | 1                    | 1                |

23.  $0 \cdot \bar{0} = 0 \cdot 1 = 0; 1 \cdot \bar{1} = 1 \cdot 0 = 0$

25.

| $x$ | $y$ | $x \oplus y$ | $x + y$ | $xy$ | $\overline{(xy)}$ | $(x + y)\overline{(xy)}$ | $x\bar{y}$ | $\bar{x}y$ | $x\bar{y} + \bar{x}y$ |
|-----|-----|--------------|---------|------|-------------------|--------------------------|------------|------------|-----------------------|
| 1   | 1   | 0            | 1       | 1    | 0                 | 0                        | 0          | 0          | 0                     |
| 1   | 0   | 1            | 1       | 0    | 1                 | 1                        | 1          | 0          | 1                     |
| 0   | 1   | 1            | 1       | 0    | 1                 | 1                        | 0          | 1          | 1                     |
| 0   | 0   | 0            | 0       | 0    | 1                 | 0                        | 0          | 0          | 0                     |

27. a) True, as a table of values can show b) False; take  $x = 1, y = 1, z = 1$ , for instance c) False; take  $x = 1, y = 1, z = 0$ , for instance 29. By De Morgan's laws, the complement of an expression is like the dual except that the complement of each variable has been taken. 31. 16 33. If we replace each 0 by F, 1 by T, Boolean sum by  $\vee$ , Boolean product by  $\wedge$ , and  $\bar{\phantom{x}}$  by  $\neg$  (and  $x$  by  $p$  and  $y$  by  $q$  so that the variables look like they represent propositions, and the equals sign by the logical equivalence symbol), then  $\bar{xy} = \bar{x} + \bar{y}$  becomes  $\neg(p \wedge q) \equiv \neg p \vee \neg q$  and  $\bar{x} + \bar{y} = \bar{x}\bar{y}$  becomes  $\neg(p \vee q) \equiv \neg p \wedge \neg q$ . 35. By the domination, distributive, and identity laws,  $x \vee x = (x \vee x) \wedge 1 = (x \vee x) \wedge (x \vee \bar{x}) = x \vee (x \wedge \bar{x}) = x \vee 0 = x$ . Similarly,  $x \wedge x = (x \wedge x) \vee 0 = (x \wedge x) \vee (x \wedge \bar{x}) = x \wedge (x \vee \bar{x}) = x \wedge 1 = x$ . 37. Because  $0 \vee 1 = 1$  and  $0 \wedge 1 = 0$  by the identity and commutative laws, it follows that  $\bar{0} = 1$ . Similarly, because  $1 \vee 0 = 1$  and  $1 \wedge 0 = 0$ , it follows that  $\bar{1} = 0$ . 39. First, note that  $x \wedge 0 = 0$  and  $x \vee 1 = 1$  for all  $x$ , as can easily be proved. To prove the first identity, it is sufficient to show that  $(x \vee y) \vee (\bar{x} \wedge \bar{y}) = 1$  and  $(x \vee y) \wedge (\bar{x} \wedge \bar{y}) = 0$ . By the associative, commutative, distributive, domination, and identity laws,  $(x \vee y) \vee (\bar{x} \wedge \bar{y}) = y \vee [x \vee (\bar{x} \wedge \bar{y})] = y \vee [(x \vee \bar{x}) \wedge (x \vee \bar{y})] = y \vee [1 \wedge (x \vee \bar{y})] = y \vee (x \vee \bar{y}) = (y \vee \bar{y}) \vee x = 1 \vee x = 1$  and  $(x \vee y) \wedge (\bar{x} \wedge \bar{y}) = \bar{y} \wedge [\bar{x} \wedge (x \vee y)] = \bar{y} \wedge [\bar{x} \wedge x] \vee [\bar{x} \wedge y] = \bar{y} \wedge [0 \vee (\bar{x} \wedge y)] = \bar{y} \wedge (\bar{x} \wedge y) = \bar{x} \wedge (y \wedge \bar{y}) = \bar{x} \wedge 0 = 0$ . The second identity is proved in a similar way. 41. Using the

hypotheses, Exercise 35, and the distributive law it follows that  $x = x \vee 0 = x \vee (x \vee y) = (x \vee x) \vee y = x \vee y = 0$ . Similarly,  $y = 0$ . To prove the second statement, note that  $x = x \wedge 1 = x \wedge (x \wedge y) = (x \wedge x) \wedge y = x \wedge y = 1$ . Similarly,  $y = 1$ . 43. Use Exercises 39 and 41 in the Supplementary Exercises in Chapter 9 and the definition of a complemented, distributed lattice to establish the five pairs of laws in the definition.

## Section 12.2

1. a)  $\bar{x}\bar{y}z$  b)  $\bar{x}y\bar{z}$  c)  $\bar{x}yz$  d)  $\bar{x}\bar{y}\bar{z}$  3. a)  $xyz + xy\bar{z} + x\bar{y}z + x\bar{y}\bar{z} + \bar{x}yz + \bar{x}\bar{y}\bar{z}$  b)  $xyz + xy\bar{z} + \bar{x}yz$  c)  $xyz + xy\bar{z} + x\bar{y}\bar{z} + x\bar{y}\bar{z}$  d)  $\bar{x}yz + x\bar{y}\bar{z}$  5.  $wxyz + wx\bar{y}z + w\bar{x}yz + \bar{w}x\bar{y}z + \bar{w}\bar{x}\bar{y}z + \bar{w}\bar{x}\bar{y}\bar{z}$  7. a)  $\bar{x} + \bar{y} + z$  b)  $x + y + z$  c)  $x + \bar{y} + z$  9.  $y_1 + y_2 + \dots + y_n = 0$  if and only if  $y_i = 0$  for  $i = 1, 2, \dots, n$ . This holds if and only if  $x_i = 0$  when  $y_i = x_i$  and  $x_i = 1$  when  $y_i = \bar{x}_i$ . 11. a)  $x + y + z$  b)  $(x + y + z)(x + \bar{y} + z)(\bar{x} + y + \bar{z})$  c)  $(x + y + z)(x + y + \bar{z})(x + \bar{y} + \bar{z})(x + \bar{y} + \bar{z})$  d)  $(x + y + z)(x + y + \bar{z})(x + \bar{y} + \bar{z})(x + \bar{y} + \bar{z})$  13. a)  $x + y + z$  b)  $x + [y + (\bar{x} + z)]$  c)  $(x + \bar{y})$  d)  $[x + (\bar{x} + \bar{y} + \bar{z})]$

15. a)

| $x$ | $\bar{x}$ | $x \downarrow x$ |
|-----|-----------|------------------|
| 1   | 0         | 0                |
| 0   | 1         | 1                |

b)

| $x$ | $y$ | $xy$ | $x \downarrow x$ | $y \downarrow y$ | $(x \downarrow x) \downarrow (y \downarrow y)$ |
|-----|-----|------|------------------|------------------|------------------------------------------------|
| 1   | 1   | 1    | 0                | 0                | 1                                              |
| 1   | 0   | 0    | 0                | 1                | 0                                              |
| 0   | 1   | 0    | 1                | 0                | 0                                              |
| 0   | 0   | 0    | 1                | 1                | 0                                              |

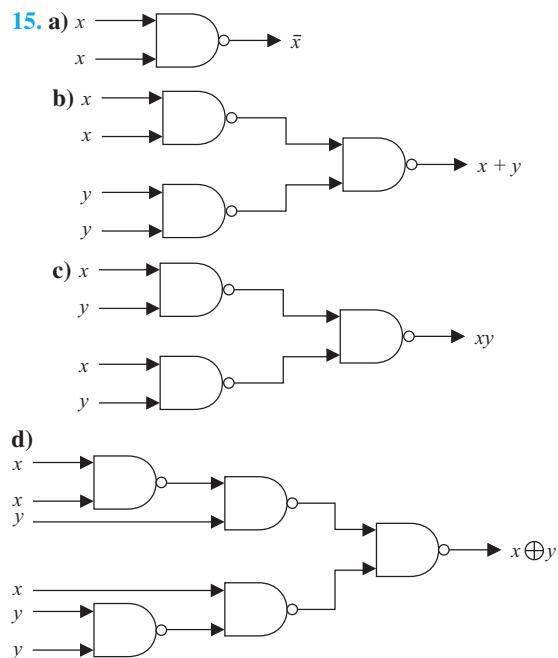
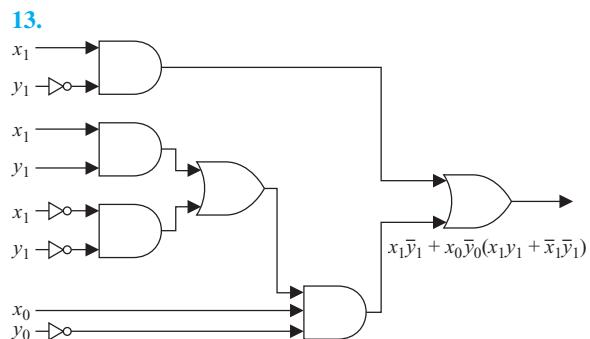
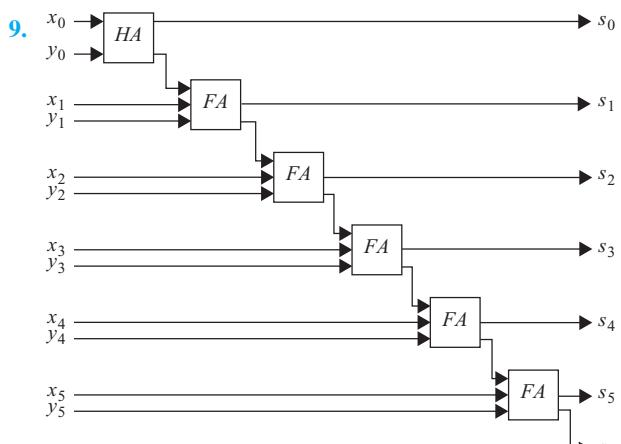
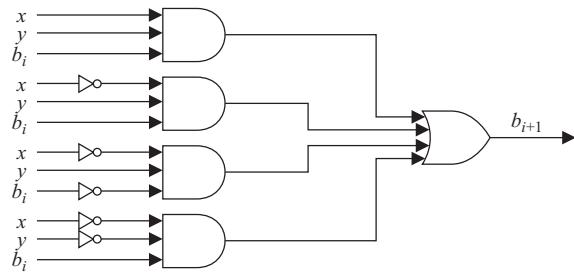
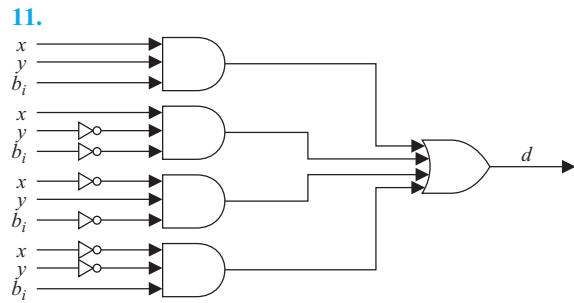
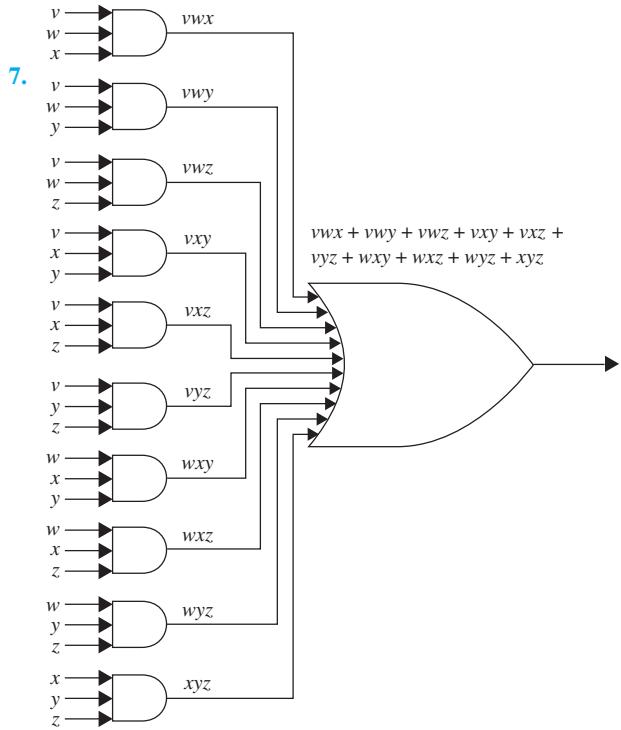
c)

| $x$ | $y$ | $x + y$ | $(x \downarrow y)$ | $(x \downarrow y) \downarrow (x \downarrow y)$ |
|-----|-----|---------|--------------------|------------------------------------------------|
| 1   | 1   | 1       | 0                  | 1                                              |
| 1   | 0   | 1       | 0                  | 1                                              |
| 0   | 1   | 1       | 0                  | 1                                              |
| 0   | 0   | 0       | 1                  | 0                                              |

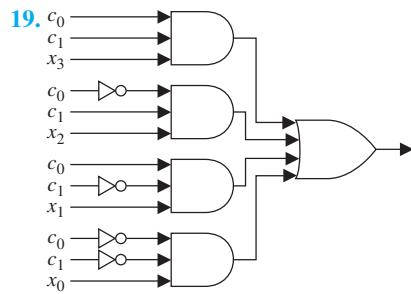
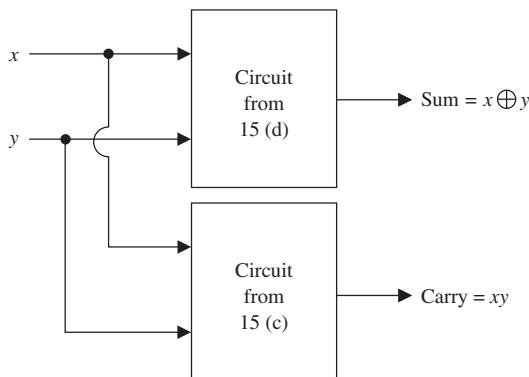
17. a)  $\{(x \mid x) \mid (y \mid y)\} \mid (z \mid z)$  b)  $\{[(x \mid x) \mid (z \mid z)] \mid y\} \mid \{[(x \mid x) \mid (z \mid z)] \mid y\}$  c)  $x$  d)  $[x \mid (y \mid y)] \mid [x \mid (y \mid y)]$  19. It is impossible to represent  $\bar{x}$  using  $+$  and  $\cdot$  because there is no way to get the value 0 if the input is 1.

## Section 12.3

1.  $(x + y)\bar{y}$  3.  $\overline{(xy)} + (\bar{z} + x)$  5.  $(x + y + z) + (\bar{x} + y + z) + (\bar{x} + \bar{y} + z) + (\bar{x} + \bar{y} + \bar{z})$



17.



## Section 12.4

a)

| $y$       | $\bar{y}$ |
|-----------|-----------|
| $x$       |           |
| $\bar{x}$ | 1         |

b)  $xy$  and  $\bar{x}\bar{y}$ 

|   |  |
|---|--|
|   |  |
| 1 |  |

a)

| $y$       | $\bar{y}$ |
|-----------|-----------|
| $x$       |           |
| $\bar{x}$ |           |

c)

| $y$       | $\bar{y}$ |
|-----------|-----------|
| $x$       | 1         |
| $\bar{x}$ | 1         |

a)

| $yz$      | $y\bar{z}$ | $\bar{y}z$ | $\bar{y}\bar{z}$ |
|-----------|------------|------------|------------------|
| $x$       |            |            |                  |
| $\bar{x}$ | 1          |            |                  |

b)  $\bar{x}yz$ ,  $\bar{x}\bar{y}\bar{z}$ ,  $xy\bar{z}$ 

7. a)

| $y\bar{z}$ | $y\bar{z}$ | $\bar{y}\bar{z}$ | $\bar{y}z$ |
|------------|------------|------------------|------------|
| $x$        |            | 1                |            |
| $\bar{x}$  |            |                  |            |

b)

| $yz$      | $y\bar{z}$ | $\bar{y}\bar{z}$ | $\bar{y}z$ |
|-----------|------------|------------------|------------|
| $x$       |            |                  |            |
| $\bar{x}$ | 1          |                  | 1          |

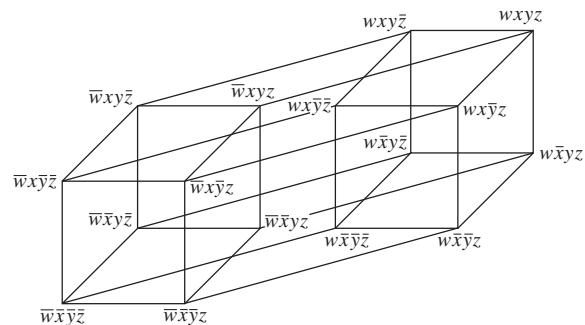
c)

| $yz$      | $y\bar{z}$ | $\bar{y}\bar{z}$ | $\bar{y}z$ |
|-----------|------------|------------------|------------|
| $x$       | 1          | 1                |            |
| $\bar{x}$ |            |                  | 1          |

9. Implicants:  $xyz$ ,  $xy\bar{z}$ ,  $x\bar{y}\bar{z}$ ,  $\bar{x}y\bar{z}$ ,  $xy$ ,  $x\bar{z}$ ,  $y\bar{z}$ ; prime implicants:  $xy$ ,  $x\bar{z}$ ,  $y\bar{z}$ ; essential prime implicants:  $xy$ ,  $x\bar{z}$ ,  $y\bar{z}$ 

| $y\bar{z}$ | $y\bar{z}$ | $\bar{y}\bar{z}$ | $\bar{y}z$ |
|------------|------------|------------------|------------|
| $x$        | 1          | 1                | 1          |
| $\bar{x}$  |            | 1                |            |

11. The 3-cube on the right corresponds to  $w$ ; the 3-cube given by the top surface of the whole figure represents  $x$ ; the 3-cube given by the back surface of the whole figure represents  $y$ ; the 3-cube given by the right surfaces of both the left and the right 3-cube represents  $z$ . In each case, the opposite 3-face represents the complemented literal. The 2-cube that represents  $wz$  is the right face of the 3-cube on the right; the 2-cube that represents  $\bar{x}y$  is bottom rear; the 2-cube that represents  $\bar{y}\bar{z}$  is front left.



**S-82** Answers to Odd-Numbered Exercises

**13. a)**

|                  | $yz$ | $y\bar{z}$ | $\bar{y}z$ | $\bar{y}\bar{z}$ |
|------------------|------|------------|------------|------------------|
| $wx$             |      |            |            |                  |
| $w\bar{x}$       |      |            |            |                  |
| $\bar{w}\bar{x}$ |      |            |            |                  |
| $\bar{w}x$       |      | 1          |            |                  |

**b)**  $\bar{w}xyz$ ,  $\bar{w}\bar{x}y\bar{z}$ ,  $\bar{w}x\bar{y}\bar{z}$ ,  $wxy\bar{z}$

**15. a)**

|                      | $x_3x_4x_5$ | $x_3x_4\bar{x}_5$ | $x_3\bar{x}_4\bar{x}_5$ | $x_3\bar{x}_4x_5$ | $\bar{x}_3\bar{x}_4x_5$ | $\bar{x}_3\bar{x}_4\bar{x}_5$ | $\bar{x}_3x_4\bar{x}_5$ | $\bar{x}_3x_4x_5$ |
|----------------------|-------------|-------------------|-------------------------|-------------------|-------------------------|-------------------------------|-------------------------|-------------------|
| $x_1x_2$             | 1           | 1                 |                         |                   |                         |                               |                         |                   |
| $x_1\bar{x}_2$       |             |                   |                         |                   |                         |                               |                         |                   |
| $\bar{x}_1\bar{x}_2$ |             |                   |                         |                   |                         |                               |                         |                   |
| $\bar{x}_1x_2$       |             |                   |                         |                   |                         |                               |                         |                   |

**b)**

|                      | $x_3x_4x_5$ | $x_3x_4\bar{x}_5$ | $x_3\bar{x}_4\bar{x}_5$ | $x_3\bar{x}_4x_5$ | $\bar{x}_3\bar{x}_4x_5$ | $\bar{x}_3\bar{x}_4\bar{x}_5$ | $\bar{x}_3x_4\bar{x}_5$ | $\bar{x}_3x_4x_5$ |
|----------------------|-------------|-------------------|-------------------------|-------------------|-------------------------|-------------------------------|-------------------------|-------------------|
| $x_1x_2$             |             |                   |                         |                   |                         |                               |                         |                   |
| $x_1\bar{x}_2$       |             |                   |                         |                   |                         |                               |                         |                   |
| $\bar{x}_1\bar{x}_2$ | 1           |                   |                         | 1                 |                         |                               |                         |                   |
| $\bar{x}_1x_2$       | 1           |                   |                         | 1                 |                         |                               |                         |                   |

**c)**

|                      | $x_3x_4x_5$ | $x_3x_4\bar{x}_5$ | $x_3\bar{x}_4\bar{x}_5$ | $x_3\bar{x}_4x_5$ | $\bar{x}_3\bar{x}_4x_5$ | $\bar{x}_3\bar{x}_4\bar{x}_5$ | $\bar{x}_3x_4\bar{x}_5$ | $\bar{x}_3x_4x_5$ |
|----------------------|-------------|-------------------|-------------------------|-------------------|-------------------------|-------------------------------|-------------------------|-------------------|
| $x_1x_2$             | 1           | 1                 |                         |                   |                         | 1                             | 1                       |                   |
| $x_1\bar{x}_2$       |             |                   |                         |                   |                         |                               |                         |                   |
| $\bar{x}_1\bar{x}_2$ |             |                   |                         |                   |                         |                               |                         |                   |
| $\bar{x}_1x_2$       | 1           | 1                 |                         |                   |                         | 1                             | 1                       |                   |

**d)**

|                      | $x_3x_4x_5$ | $x_3x_4\bar{x}_5$ | $x_3\bar{x}_4\bar{x}_5$ | $x_3\bar{x}_4x_5$ | $\bar{x}_3\bar{x}_4x_5$ | $\bar{x}_3\bar{x}_4\bar{x}_5$ | $\bar{x}_3x_4\bar{x}_5$ | $\bar{x}_3x_4x_5$ |
|----------------------|-------------|-------------------|-------------------------|-------------------|-------------------------|-------------------------------|-------------------------|-------------------|
| $x_1x_2$             |             |                   |                         |                   | 1                       | 1                             |                         |                   |
| $x_1\bar{x}_2$       |             |                   |                         |                   | 1                       | 1                             |                         |                   |
| $\bar{x}_1\bar{x}_2$ |             |                   |                         |                   | 1                       | 1                             |                         |                   |
| $\bar{x}_1x_2$       |             |                   |                         |                   | 1                       | 1                             |                         |                   |

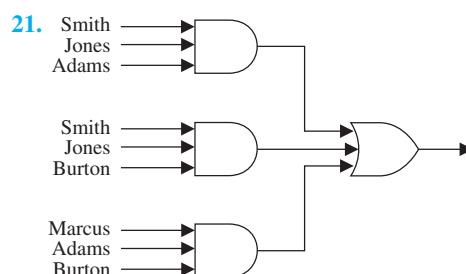
**e)**

|                      | $x_3x_4x_5$ | $x_3x_4\bar{x}_5$ | $x_3\bar{x}_4\bar{x}_5$ | $x_3\bar{x}_4x_5$ | $\bar{x}_3\bar{x}_4x_5$ | $\bar{x}_3\bar{x}_4\bar{x}_5$ | $\bar{x}_3x_4\bar{x}_5$ | $\bar{x}_3x_4x_5$ |
|----------------------|-------------|-------------------|-------------------------|-------------------|-------------------------|-------------------------------|-------------------------|-------------------|
| $x_1x_2$             | 1           | 1                 | 1                       | 1                 |                         |                               |                         |                   |
| $x_1\bar{x}_2$       | 1           | 1                 | 1                       | 1                 |                         |                               |                         |                   |
| $\bar{x}_1\bar{x}_2$ |             |                   |                         |                   |                         |                               |                         |                   |
| $\bar{x}_1x_2$       |             |                   |                         |                   |                         |                               |                         |                   |

**f)**

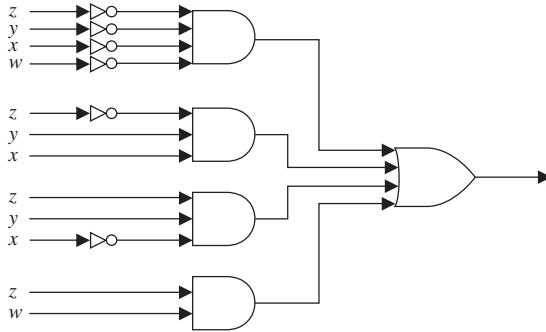
|                      | $x_3x_4x_5$ | $x_3x_4\bar{x}_5$ | $x_3\bar{x}_4\bar{x}_5$ | $x_3\bar{x}_4x_5$ | $\bar{x}_3\bar{x}_4x_5$ | $\bar{x}_3\bar{x}_4\bar{x}_5$ | $\bar{x}_3x_4\bar{x}_5$ | $\bar{x}_3x_4x_5$ |
|----------------------|-------------|-------------------|-------------------------|-------------------|-------------------------|-------------------------------|-------------------------|-------------------|
| $x_1x_2$             |             | 1                 | 1                       |                   |                         | 1                             | 1                       |                   |
| $x_1\bar{x}_2$       |             | 1                 | 1                       |                   |                         | 1                             | 1                       |                   |
| $\bar{x}_1\bar{x}_2$ |             |                   |                         |                   |                         | 1                             | 1                       |                   |
| $\bar{x}_1x_2$       |             |                   |                         |                   |                         | 1                             | 1                       |                   |

**17. a) 64** **b) 6** **19.** Rows 1 and 4 are considered adjacent. The pairs of columns considered adjacent are: columns 1 and 4, 1 and 12, 1 and 16, 2 and 11, 2 and 15, 3 and 6, 3 and 10, 4 and 9, 5 and 8, 5 and 16, 6 and 15, 7 and 10, 7 and 14, 8 and 13, 9 and 12, 11 and 14, 13 and 16.



- 23. a)  $\bar{x}z$**  **b)  $y$**  **c)  $x\bar{z} + \bar{x}z + \bar{y}z$**  **d)  $xz + \bar{x}y + \bar{y}\bar{z}$**   
**25. a)  $wxz + wx\bar{y} + w\bar{y}z + w\bar{x}y\bar{z}$**  **b)  $x\bar{y}z + \bar{w}\bar{y}z + wxy\bar{z} + w\bar{x}yz + \bar{w}\bar{x}y\bar{z}$**  **c)  $\bar{y}z + wxy + w\bar{x}\bar{y} + \bar{w}\bar{x}y\bar{z}$**   
**d)  $wy + yz + \bar{x}y + wxz + \bar{w}\bar{x}z$**  **27.  $x(y + z)$**

29.

31.  $\bar{x}\bar{z} + xz$ 

**33.** We use induction on  $n$ . If  $n = 1$ , then we are looking at a line segment, labeled 0 at one end and 1 at the other end. The only possible value of  $k$  is also 1, and if the literal is  $x_1$ , then the subcube we have is the 0-dimensional subcube consisting of the endpoint labeled 1, and if the literal is  $\bar{x}_1$ , then the subcube we have is the 0-dimensional subcube consisting of the endpoint labeled 0. Now assume that the statement is true for  $n$ ; we must show that it is true for  $n + 1$ . If the literal  $x_{n+1}$  (or its complement) is not part of the product, then by the inductive hypothesis, the product when viewed in the setting of  $n$  variables corresponds to an  $(n - k)$ -dimensional subcube of the  $n$ -dimensional cube, and the Cartesian product of that subcube with the line segment  $[0, 1]$  gives us a subcube one dimension higher in our given  $(n + 1)$ -dimensional cube, namely having dimension  $(n + 1) - k$ , as desired. On the other hand, if the literal  $x_{n+1}$  (or its complement) is part of the product, then the product of the remaining  $k - 1$  literals corresponds to a subcube of dimension  $n - (k - 1) = (n + 1) - k$  in the  $n$ -dimensional cube, and that slice, at either the 1-end or the 0-end in the last variable, is the desired subcube.

## Supplementary Exercises

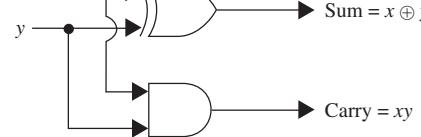
- 1.** a)  $x = 0, y = 0, z = 0; x = 1, y = 1, z = 1$  b)  $x = 0, y = 0, z = 0; x = 0, y = 0, z = 1; x = 0, y = 1, z = 0; x = 1, y = 0, z = 1; x = 1, y = 1, z = 0; x = 1, y = 1, z = 1$  c) No values **3.** a) Yes b) No c) No d) Yes **5.**  $2^{2^n-1}$  **7.** a) If  $F(x_1, \dots, x_n) = 1$ , then  $(F + G)(x_1, \dots, x_n) = F(x_1, \dots, x_n) + G(x_1, \dots, x_n) = 1$  by the dominance law. Hence,  $F \leq F + G$ . b) If  $(FG)(x_1, \dots, x_n) = 1$ , then  $F(x_1, \dots, x_n) \cdot G(x_1, \dots, x_n) = 1$ . Hence,  $F(x_1, \dots, x_n) = 1$ . It follows that  $FG \leq F$ . **9.** Because  $F(x_1, \dots, x_n) = 1$  implies that  $F(x_1, \dots, x_n) = 1$ ,  $\leq$  is reflexive. Suppose that  $F \leq G$  and  $G \leq F$ . Then  $F(x_1, \dots, x_n) = 1$  if and only if  $G(x_1, \dots, x_n) = 1$ . This implies that  $F = G$ . Hence,  $\leq$  is antisymmetric. Suppose that  $F \leq G \leq H$ . Then if  $F(x_1, \dots, x_n) = 1$ , it follows that  $G(x_1, \dots, x_n) = 1$ , which implies that  $H(x_1, \dots, x_n) = 1$ . Hence,  $F \leq H$ , so  $\leq$  is transitive. **11.** a)  $x = 1, y = 0, z = 0$  b)  $x = 1, y = 0, z = 0$ , c)  $x = 1, y = 0, z = 0$

13.

| $x$ | $y$ | $x \odot y$ | $x \oplus y$ | $(x \oplus y)$ |
|-----|-----|-------------|--------------|----------------|
| 1   | 1   | 1           | 0            | 1              |
| 1   | 0   | 0           | 1            | 0              |
| 0   | 1   | 0           | 1            | 0              |
| 0   | 0   | 1           | 0            | 1              |

**15.** Yes, as a truth table shows **17.** a) 6 b) 5 c) 5 d) 6

19.



**21.**  $x_3 + x_2\bar{x}_1$  **23.** Suppose it were with weights  $a$  and  $b$ . Then there would be a real number  $T$  such that  $xa + yb \geq T$  for  $(1,0)$  and  $(0,1)$ , but with  $xa + yb < T$  for  $(0,0)$  and  $(1,1)$ . Hence,  $a \geq T$ ,  $b \geq T$ ,  $0 < T$ , and  $a + b < T$ . Thus,  $a$  and  $b$  are positive, which implies that  $a + b > a \geq T$ , a contradiction.

## CHAPTER 13

### Section 13.1

**1.** a) sentence  $\Rightarrow$  noun phrase intransitive verb phrase  $\Rightarrow$  article adjective noun intransitive verb phrase  $\Rightarrow$  article adjective noun intransitive verb  $\Rightarrow \dots$  (after 3 steps)  $\dots \Rightarrow$  the happy hare runs.

b) sentence  $\Rightarrow$  noun phrase intransitive verb phrase  $\Rightarrow$  article adjective noun intransitive verb phrase  $\Rightarrow$  article adjective noun intransitive verb  $\Rightarrow$  adverb... (after 4 steps)  $\dots \Rightarrow$  the sleepy tortoise runs quickly

c) sentence  $\Rightarrow$  noun phrase transitive verb phrase noun phrase  $\Rightarrow$  article noun transitive verb phrase noun phrase  $\Rightarrow$  article noun transitive verb noun phrase  $\Rightarrow$  article noun transitive verb article noun  $\Rightarrow \dots$  (after 4 steps)  $\dots \Rightarrow$  the tortoise passes the hare

d) sentence  $\Rightarrow$  noun phrase transitive verb phrase noun phrase  $\Rightarrow$  article adjective noun transitive verb phrase noun phrase  $\Rightarrow$  article adjective noun transitive verb noun phrase  $\Rightarrow$  article adjective noun transitive verb article adjective noun  $\Rightarrow \dots$  (after 6 steps)  $\dots \Rightarrow$  the sleepy hare passes the happy tortoise

**3.** The only way to get a noun, such as *tortoise*, at the end is to have a noun phrase at the end, which can be achieved only via the production sentence  $\rightarrow$  noun phrase transitive verb phrase noun phrase. However, transitive verb phrase  $\rightarrow$  transitive verb  $\rightarrow$  passes, and this sentence does not contain *passes*.

**5.** a)  $S \Rightarrow 1A \Rightarrow 10B \Rightarrow 101A \Rightarrow 1010B \Rightarrow 10101$  b) Because of the productions in this grammar, every 1 must be followed by a 0 unless it occurs at the end of the string. c) All strings consisting of a 0 or a 1 followed by one or more repetitions of 01

S-84 Answers to Odd-Numbered Exercises

7.  $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 000111$

9. a)  $S \Rightarrow 0S \Rightarrow 00S \Rightarrow 00S1 \Rightarrow 00S11 \Rightarrow 00S111 \Rightarrow 00S1111 \Rightarrow 001111$       b)  $S \Rightarrow 0S \Rightarrow 00S \Rightarrow 001A \Rightarrow 0011A \Rightarrow 00111A \Rightarrow 001111$

11.  $S \Rightarrow 0SAB \Rightarrow 00SABAB \Rightarrow 00ABAB \Rightarrow 00 AABB \Rightarrow 001ABB \Rightarrow 001BB \Rightarrow 00112B \Rightarrow 001122$

13. a)  $S \rightarrow 0, S \rightarrow 1, S \rightarrow 11$       b)  $S \rightarrow 1S, S \rightarrow \lambda$       c)  $S \rightarrow 0A1, A \rightarrow 1A, A \rightarrow 0A, A \rightarrow \lambda$       d)  $S \rightarrow 0A, A \rightarrow 11A, A \rightarrow \lambda$

15. a)  $S \rightarrow 00S, S \rightarrow \lambda$       b)  $S \rightarrow 10A, A \rightarrow 00A, A \rightarrow \lambda$       c)  $S \rightarrow AAS, S \rightarrow BBS, AB \rightarrow BA, BA \rightarrow AB, S \rightarrow \lambda, A \rightarrow 0, B \rightarrow 1$

d)  $S \rightarrow 0000000000A, A \rightarrow 0A, A \rightarrow \lambda$       e)  $S \rightarrow AS, S \rightarrow ABS, S \rightarrow A, AB \rightarrow BA, BA \rightarrow AB, A \rightarrow 0, B \rightarrow 1$

f)  $S \rightarrow ABS, S \rightarrow \lambda, AB \rightarrow BA, BA \rightarrow AB, A \rightarrow 0, B \rightarrow 1$       g)  $S \rightarrow ABS, S \rightarrow T, S \rightarrow U, T \rightarrow AT, T \rightarrow A, U \rightarrow BU, U \rightarrow B, AB \rightarrow BA, BA \rightarrow AB, A \rightarrow 0, B \rightarrow 1$

17. a)  $S \rightarrow 0S, S \rightarrow \lambda$       b)  $S \rightarrow A0, A \rightarrow 1A, A \rightarrow \lambda$       c)  $S \rightarrow 000S, S \rightarrow \lambda$

19. a) Type 2, not type 3      b) Type 3      c) Type 0, not type 1

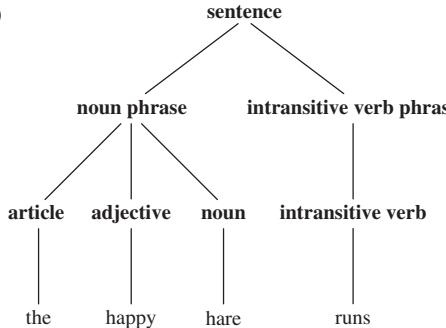
d) Type 2, not type 3      e) Type 2, not type 3      f) Type 0, not type 1

g) Type 3      h) Type 0, not type 1      i) Type 2, not type 3

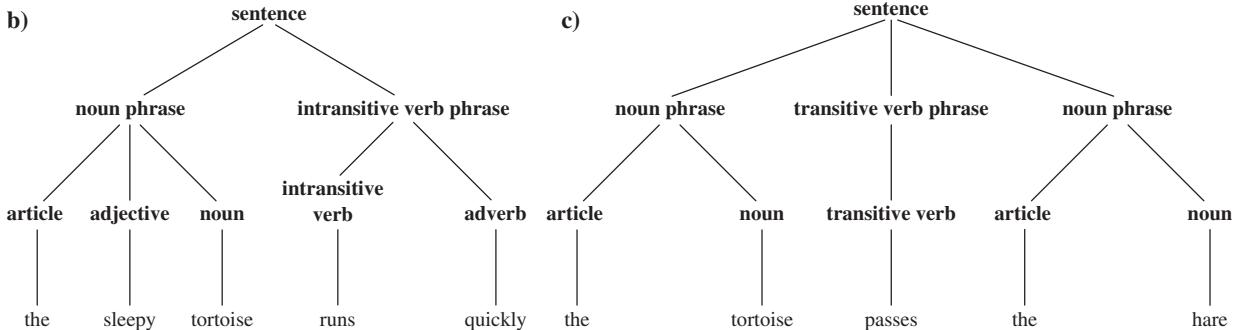
j) Type 2, not type 3      21. Let  $S_1$  and  $S_2$  be the start symbols of  $G_1$  and  $G_2$ , respectively. Let  $S$  be a new start symbol.

a) Add  $S$  and productions  $S \rightarrow S_1$  and  $S \rightarrow S_2$ .      b) Add  $S$  and production  $S \rightarrow S_1S_2$ .      c) Add  $S$  and production  $S \rightarrow \lambda$  and  $S \rightarrow S_1S$ .

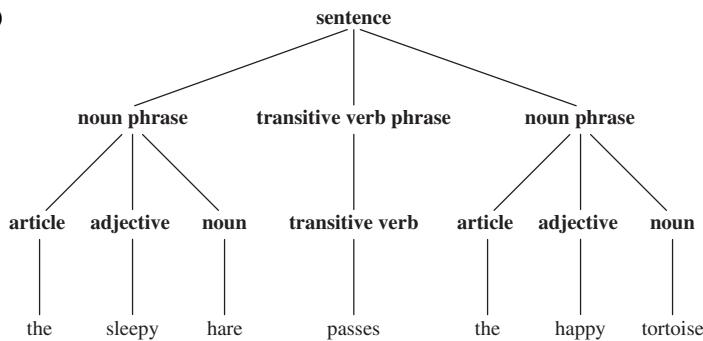
23. a)



b)

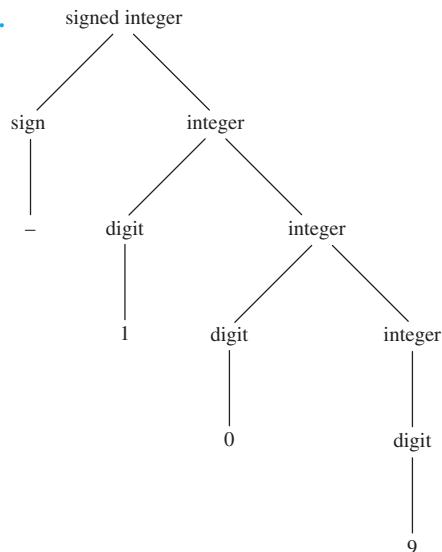
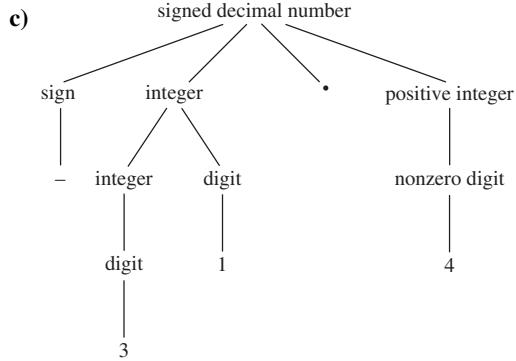


d)



25. a) Yes b) No c) Yes d) No

27.

29. a)  $S \rightarrow \langle sign \rangle \langle integer \rangle$  $S \rightarrow \langle sign \rangle \langle integer \rangle . \langle positive\ integer \rangle$  $\langle sign \rangle \rightarrow +$  $\langle sign \rangle \rightarrow -$  $\langle integer \rangle \rightarrow \langle digit \rangle$  $\langle integer \rangle \rightarrow \langle integer \rangle \langle digit \rangle$  $\langle digit \rangle \rightarrow i, i = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0$  $\langle positive\ integer \rangle \rightarrow \langle integer \rangle \langle nonzero\ digit \rangle$  $\langle positive\ integer \rangle \rightarrow \langle nonzero\ digit \rangle \langle integer \rangle$  $\langle positive\ integer \rangle \rightarrow \langle integer \rangle \langle nonzero\ digit \rangle \langle integer \rangle$  $\langle positive\ integer \rangle \rightarrow \langle nonzero\ digit \rangle \langle integer \rangle$  $\langle nonzero\ digit \rangle \rightarrow i, i = 1, 2, 3, 4, 5, 6, 7, 8, 9$ b)  $\langle signed\ decimal\ number \rangle ::= \langle sign \rangle \langle integer \rangle |$  $\langle sign \rangle \langle integer \rangle . \langle positive\ integer \rangle$  $\langle sign \rangle ::= + | -$  $\langle integer \rangle ::= \langle digit \rangle | \langle integer \rangle \langle digit \rangle$  $\langle digit \rangle ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$  $\langle nonzero\ digit \rangle ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$  $\langle positive\ integer \rangle ::= \langle integer \rangle \langle nonzero\ digit \rangle |$  $\langle nonzero\ digit \rangle \langle integer \rangle | \langle integer \rangle$  $\langle nonzero\ integer \rangle \langle integer \rangle | \langle nonzero\ digit \rangle$ 

**31. a)**  $\langle \text{identifier} \rangle ::= \langle \text{lcletter} \rangle \mid \langle \text{identifier} \rangle \langle \text{lcletter} \rangle$

$\langle \text{lcletter} \rangle ::= a \mid b \mid c \mid \dots \mid z$

**b)**  $\langle \text{identifier} \rangle ::= \langle \text{lcletter} \rangle \langle \text{lcletter} \rangle \langle \text{lcletter} \rangle \mid \langle \text{lcletter} \rangle \langle \text{lcletter} \rangle \langle \text{lcletter} \rangle \langle \text{lcletter} \rangle \mid$

$\langle \text{lcletter} \rangle \langle \text{lcletter} \rangle \langle \text{lcletter} \rangle \langle \text{lcletter} \rangle \mid$

$\langle \text{lcletter} \rangle \langle \text{lcletter} \rangle \langle \text{lcletter} \rangle \langle \text{lcletter} \rangle \langle \text{lcletter} \rangle \mid$

$\langle \text{lcletter} \rangle ::= a \mid b \mid c \mid \dots \mid z$

**c)**  $\langle \text{identifier} \rangle ::= \langle \text{ucletter} \rangle \mid \langle \text{ucletter} \rangle \langle \text{letter} \rangle \mid \langle \text{ucletter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle \mid$

$\langle \text{ucletter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle \mid \langle \text{ucletter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle \mid$

$\langle \text{ucletter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle \langle \text{letter} \rangle \mid$

$\langle \text{letter} \rangle ::= \langle \text{lcletter} \rangle \mid \langle \text{ucletter} \rangle$

$\langle \text{lcletter} \rangle ::= a \mid b \mid c \mid \dots \mid z$

$\langle \text{ucletter} \rangle ::= A \mid B \mid C \mid \dots \mid Z$

**d)**  $\langle \text{identifier} \rangle ::= \langle \text{lcletter} \rangle \langle \text{digitorus} \rangle \langle \text{alphanumeric} \rangle \langle \text{alphanumeric} \rangle \langle \text{alphanumeric} \rangle \mid$

$\langle \text{lcletter} \rangle \langle \text{digitorus} \rangle \langle \text{alphanumeric} \rangle \langle \text{alphanumeric} \rangle \langle \text{alphanumeric} \rangle \langle \text{alphanumeric} \rangle$

$\langle \text{digitorus} \rangle ::= \langle \text{digit} \rangle \mid \_$

$\langle \text{alphanumeric} \rangle ::= \langle \text{letter} \rangle \mid \langle \text{digit} \rangle$

$\langle \text{letter} \rangle ::= \langle \text{lcletter} \rangle \mid \langle \text{ucletter} \rangle$

$\langle \text{lcletter} \rangle ::= a \mid b \mid c \mid \dots \mid z$

$\langle \text{ucletter} \rangle ::= A \mid B \mid C \mid \dots \mid Z$

$\langle \text{digit} \rangle ::= 0 \mid 1 \mid 2 \mid \dots \mid 9$

**33.**  $\langle \text{identifier} \rangle ::= \langle \text{letterorus} \rangle \mid \langle \text{identifier} \rangle \langle \text{symbol} \rangle$

$\langle \text{letterorus} \rangle ::= \langle \text{letter} \rangle \mid \_$

$\langle \text{symbol} \rangle ::= \langle \text{letterorus} \rangle \mid \langle \text{digit} \rangle$

$\langle \text{letter} \rangle ::= \langle \text{lcletter} \rangle \mid \langle \text{ucletter} \rangle$

$\langle \text{lcletter} \rangle ::= a \mid b \mid c \mid \dots \mid z$

$\langle \text{ucletter} \rangle ::= A \mid B \mid C \mid \dots \mid Z$

$\langle \text{digit} \rangle ::= 0 \mid 1 \mid 2 \mid \dots \mid 9$

**35.**  $\text{numeral} ::= \text{sign? nonzerodigit digit* decimal?} \mid \text{sign? 0 decimal?}$

$\text{sign} ::= + \mid -$

$\text{nonzerodigit} ::= 1 \mid 2 \mid \dots \mid 9$

$\text{digit} ::= 0 \mid \text{nonzerodigit}$

$\text{decimal} ::= .\text{digit}^*$

**37.**  $\text{identifier} ::= \text{letterorus symbol*}$

$\text{letterorus} ::= \text{letter} \mid \_$

$\text{symbol} ::= \text{letterorus} \mid \text{digit}$

$\text{letter} ::= \text{lcletter} \mid \text{ucletter}$

$\langle \text{lcletter} \rangle ::= a \mid b \mid c \mid \dots \mid z$

$\langle \text{ucletter} \rangle ::= A \mid B \mid C \mid \dots \mid Z$

$\langle \text{digit} \rangle ::= 0 \mid 1 \mid 2 \mid \dots \mid 9$

**39. a)**  $\langle \text{expression} \rangle$

$\langle \text{term} \rangle \langle \text{term} \rangle \langle \text{addOperator} \rangle$

$\langle \text{factor} \rangle \langle \text{factor} \rangle \langle \text{factor} \rangle \langle \text{mulOperator} \rangle \langle \text{addOperator} \rangle$

$\langle \text{identifier} \rangle \langle \text{identifier} \rangle \langle \text{identifier} \rangle \langle \text{mulOperator} \rangle \langle \text{addOperator} \rangle$

$a \ b \ c \ * \ +$

**b)** Not generated

**c)**  $\langle \text{expression} \rangle$

$\langle \text{term} \rangle$

$\langle \text{factor} \rangle \langle \text{factor} \rangle \langle \text{mulOperator} \rangle$

$\langle \text{expression} \rangle \langle \text{factor} \rangle \langle \text{mulOperator} \rangle$

$\langle \text{term} \rangle \langle \text{term} \rangle \langle \text{addOperator} \rangle \langle \text{factor} \rangle \langle \text{mulOperator} \rangle$

$\langle \text{factor} \rangle \langle \text{factor} \rangle \langle \text{addOperator} \rangle \langle \text{factor} \rangle \langle \text{mulOperator} \rangle$

$\langle \text{identifier} \rangle \langle \text{identifier} \rangle \langle \text{addOperator} \rangle \langle \text{identifier} \rangle \langle \text{mulOperator} \rangle$

$x \ y \ - \ z \ *$

**d)  $\langle expression \rangle$** 

$\langle term \rangle$   
 $\langle factor \rangle \langle factor \rangle \langle mulOperator \rangle$   
 $\langle factor \rangle \langle expression \rangle \langle mulOperator \rangle$   
 $\langle factor \rangle \langle term \rangle \langle mulOperator \rangle$   
 $\langle factor \rangle \langle factor \rangle \langle factor \rangle \langle mulOperator \rangle \langle mulOperator \rangle$   
 $\langle factor \rangle \langle factor \rangle \langle expression \rangle \langle mulOperator \rangle \langle mulOperator \rangle$   
 $\langle factor \rangle \langle factor \rangle \langle term \rangle \langle term \rangle \langle addOperator \rangle \langle mulOperator \rangle \langle mulOperator \rangle$   
 $\langle factor \rangle \langle factor \rangle \langle factor \rangle \langle factor \rangle \langle addOperator \rangle \langle mulOperator \rangle \langle mulOperator \rangle$   
 $\langle identifier \rangle \langle identifier \rangle \langle identifier \rangle \langle identifier \rangle \langle addOperator \rangle \langle mulOperator \rangle \langle mulOperator \rangle$   
 $w \ x \ y \ z - * /$

**e)  $\langle expression \rangle$** 

$\langle term \rangle$   
 $\langle factor \rangle \langle factor \rangle \langle mulOperator \rangle$   
 $\langle factor \rangle \langle expression \rangle \langle mulOperator \rangle$   
 $\langle factor \rangle \langle term \rangle \langle term \rangle \langle addOperator \rangle \langle mulOperator \rangle$   
 $\langle factor \rangle \langle factor \rangle \langle factor \rangle \langle addOperator \rangle \langle mulOperator \rangle$   
 $\langle identifier \rangle \langle identifier \rangle \langle identifier \rangle \langle addOperator \rangle \langle mulOperator \rangle$   
 $a \ d \ e - *$

**41. a)** Not generated**b)  $\langle expression \rangle$** 

$\langle term \rangle \langle addOperator \rangle \langle term \rangle$   
 $\langle factor \rangle \langle mulOperator \rangle \langle factor \rangle \langle addOperator \rangle \langle factor \rangle \langle mulOperator \rangle \langle factor \rangle$   
 $\langle identifier \rangle \langle mulOperator \rangle \langle identifier \rangle \langle addOperator \rangle \langle identifier \rangle \langle mulOperator \rangle \langle identifier \rangle$   
 $a/b + c/d$

**c)  $\langle expression \rangle$** 

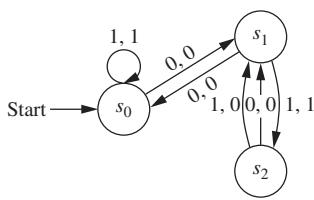
$\langle term \rangle$   
 $\langle factor \rangle \langle mulOperator \rangle \langle factor \rangle$   
 $\langle factor \rangle \langle mulOperator \rangle (\langle expression \rangle)$   
 $\langle factor \rangle \langle mulOperator \rangle (\langle term \rangle \langle addOperator \rangle \langle term \rangle)$   
 $\langle factor \rangle \langle mulOperator \rangle (\langle factor \rangle \langle addOperator \rangle \langle factor \rangle)$   
 $\langle identifier \rangle \langle mulOperator \rangle (\langle identifier \rangle \langle addOperator \rangle \langle identifier \rangle)$   
 $m * (n + p)$

**d) Not generated****e)  $\langle expression \rangle$** 

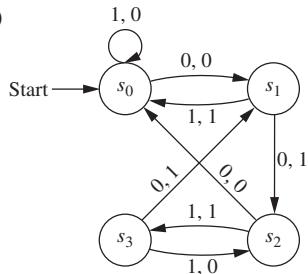
$\langle term \rangle$   
 $\langle factor \rangle \langle mulOperator \rangle \langle factor \rangle$   
 $(\langle expression \rangle) \langle mulOperator \rangle (\langle expression \rangle)$   
 $(\langle term \rangle \langle addOperator \rangle \langle term \rangle) \langle mulOperator \rangle (\langle term \rangle \langle addOperator \rangle \langle term \rangle) \quad (\langle factor \rangle \langle addOperator \rangle \langle factor \rangle) \langle mulOperator \rangle (\langle factor \rangle \langle addOperator \rangle \langle factor \rangle)$   
 $(\langle identifier \rangle \langle addOperator \rangle \langle identifier \rangle) \langle mulOperator \rangle (\langle identifier \rangle \langle addOperator \rangle \langle identifier \rangle)$   
 $(m + n) * (p - q)$

## Section 13.2

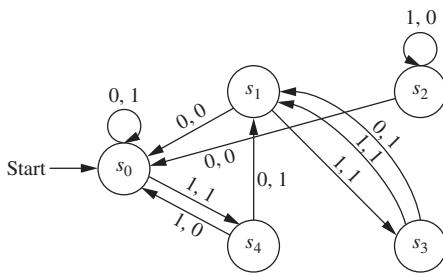
1. a)



b)

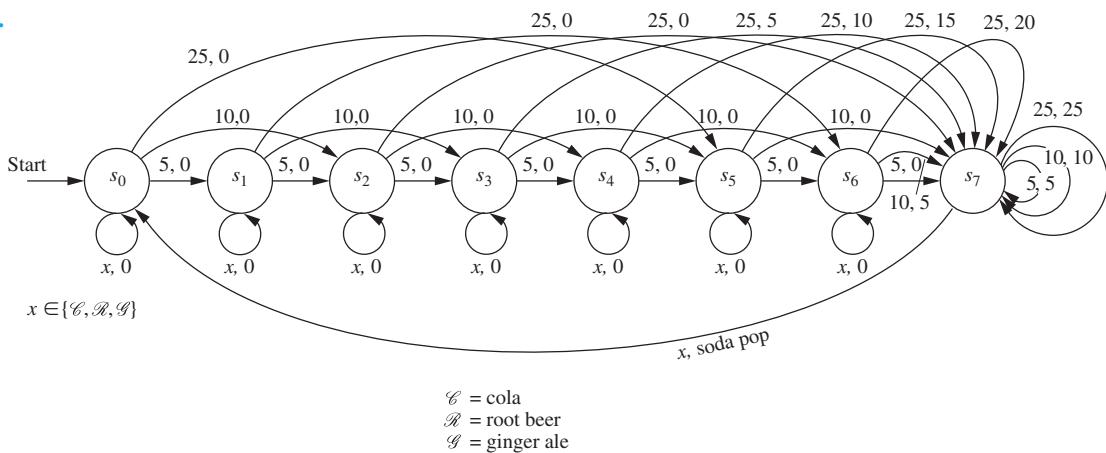


c)

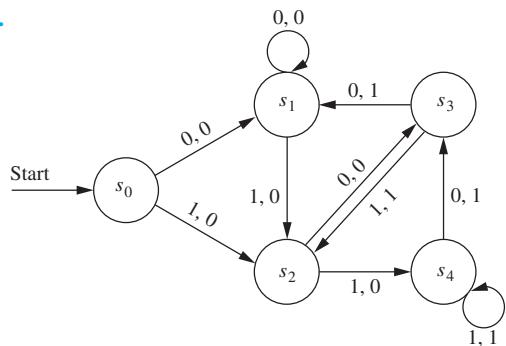


3. a) 01010 b) 01000 c) 11011 5. a) 1100 b) 00110110 c) 111111111111

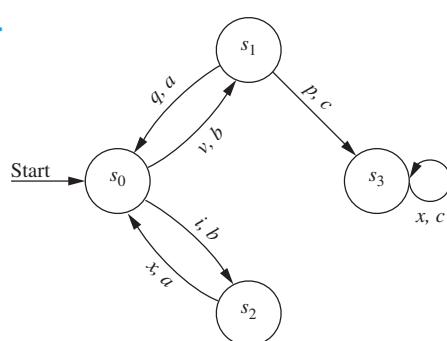
7.



9.



11.



$v$  = Valid ID

$i$  = Invalid ID

$p$  = Valid password

$q$  = Invalid password

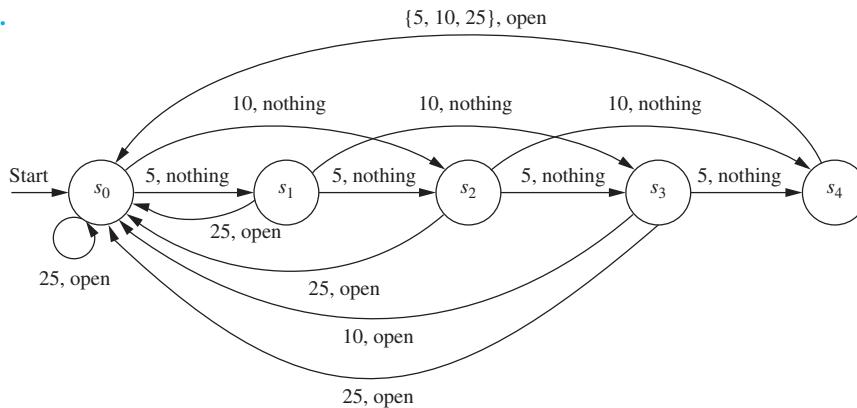
$a$  = "Enter user ID"

$b$  = "Enter password"

$c$  = Prompt

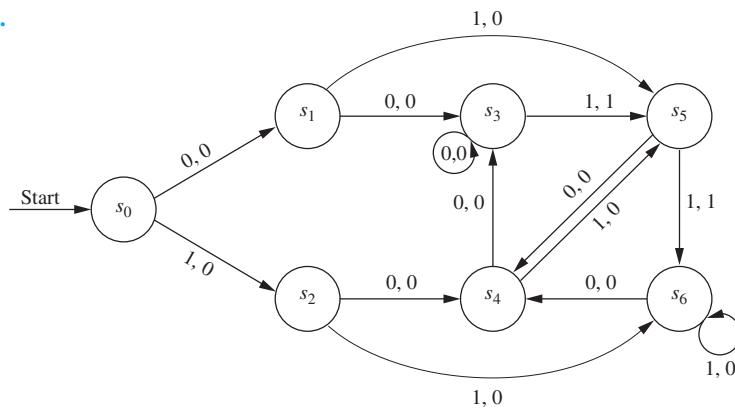
$x$  = Any input

13.

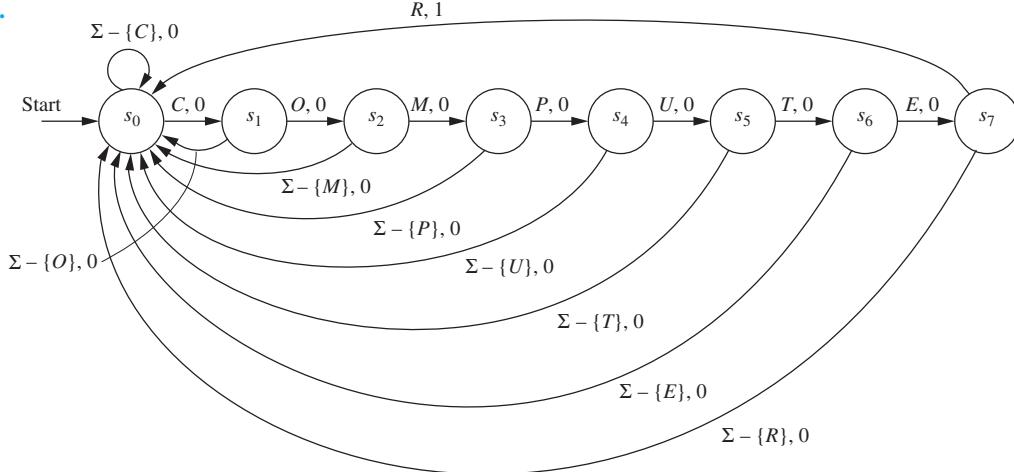


15. Let  $s_0$  be the start state and let  $s_1$  be the state representing a successful call. From  $s_0$ , inputs of 2, 3, 4, 5, 6, 7, or 8 send the machine back to  $s_0$  with output of an error message for the user. From  $s_0$  an input of 0 sends the machine to state  $s_1$ , with the output being that the 0 is sent to the network. From  $s_0$  an input of 9 sends the machine to state  $s_2$  with no output; from there an input of 1 sends the machine to state  $s_3$  with no output; from there an input of 1 sends the machine to state  $s_1$  with the output being that the 911 is sent to the network. All other inputs while in states  $s_2$  or  $s_3$  send the machine back to  $s_0$  with output of an error message for the user. From  $s_0$  an input of 1 sends the machine to state  $s_4$  with no output; from  $s_4$  an input of 2 sends the machine to state  $s_5$  with no output; and this path continues in a similar manner to the 911 path, looking next for 1, then 2, then any seven digits, at which point the machine goes to state  $s_1$  with the output being that the ten-digit input is sent to the network. Any “incorrect” input while in states  $s_5$  or  $s_6$  (that is, anything except a 1 while in  $s_5$  or a 2 while in  $s_6$ ) sends the machine back to  $s_0$  with output of an error message for the user. Similarly, from  $s_4$  an input of 8 followed by appropriate successors drives us eventually to  $s_1$ , but inappropriate outputs drive us back to  $s_0$  with an error message. Also, inputs while in state  $s_4$  other than 2 or 8 send the machine back to state  $s_0$  with output of an error message for the user.

17.



19.



21.

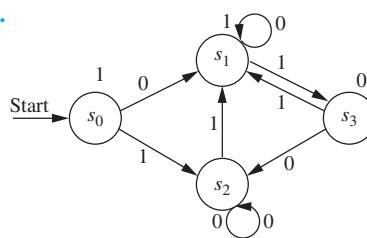
| State | $f$   |       | g |
|-------|-------|-------|---|
|       | 0     | 1     |   |
| $s_0$ | $s_1$ | $s_2$ | 1 |
| $s_1$ | $s_1$ | $s_0$ | 1 |
| $s_2$ | $s_1$ | $s_2$ | 0 |

23. a) 11111

b) 1000000

c) 100011001100

25.



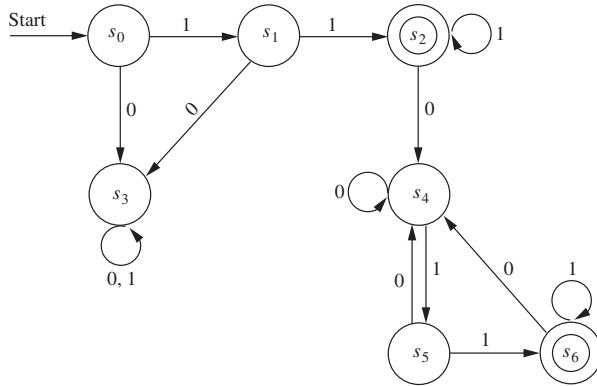
### Section 13.3

1. a) {000, 001, 1100, 1101}      b) {000, 0011, 010, 0111}  
 c) {00, 011, 110, 1111}      d) {000000, 000001, 000100, 000101, 010000, 010001, 010100, 010101} 3.  $A = \{1, 101\}$ ,  $B = \{0, 11, 000\}$ ;  $A = \{10, 111, 1010, 1000, 10111, 101000\}$ ,  $B = \{\lambda\}$ ;  $A = \{\lambda, 10\}$ ,  $B = \{10, 111, 1000\}$  or  $A = \{\lambda\}$ ,  $B = \{10, 111, 1010, 1000, 10111, 101000\}$  5. a) The set of strings consisting of zero or more consecutive bit pairs 10  
 b) The set of strings consisting of all 1s such that the number of 1s is divisible by 3, including the null string c) The set of strings in which every 1 is immediately preceded by a 0 d) The set of strings that begin and end with a 1 and have at least two 1s between every pair of 0s 7. A string is in  $A^*$  if and only if it is a concatenation of an arbitrary number of strings in  $A$ . Because each string in  $A$  is also in  $B$ , it follows that a string in  $A^*$  is also a concatenation of strings in  $B$ . Hence,  $A^* \subseteq B^*$ .  
 9. a) Yes b) Yes c) No d) No e) Yes f) Yes 11. a) Yes b) No c) Yes d) No 13. a) Yes b) Yes c) No d) No e) No f) No 15. We use structural induction on the input string  $y$ . The basis step considers  $y = \lambda$ , and for the inductive step we write  $y = wa$ , where  $w \in I^*$  and  $a \in I$ . For the basis step, we have  $xy = x$ , so we must show that  $f(s, x) = f(f(s, x), \lambda)$ . But part (i) of the definition of the extended transition function says that this is true. We then assume the inductive

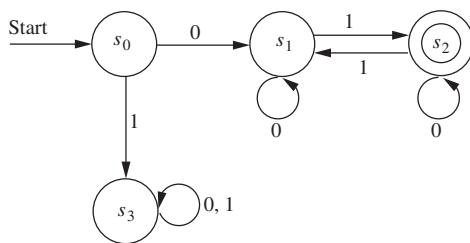
hypothesis that the equation holds for  $w$  and prove that  $f(s, xwa) = f(f(s, x), wa)$ . By part (ii) of the definition, the left-hand side of this equation equals  $f(f(s, xw), a)$ . By the inductive hypothesis,  $f(s, xw) = f(f(s, x), w)$ , so  $f(f(s, xw), a) = f(f(f(s, x), w), a)$ . The right-hand side of our desired equality is, by part (ii) of the definition, also equal to  $f(f(f(s, x), w), a)$ , as desired.

17.  $\{0, 10, 11\}\{0, 1\}^*$       19.  $\{0^m 1^n \mid m \geq 0 \text{ and } n \geq 1\}$

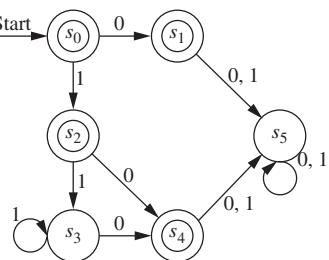
21.  $\{\lambda\} \cup \{0\}\{1\}^*\{0\} \cup \{10, 11\}\{0, 1\}^* \cup \{0\}\{1\}^*\{01\}\{0, 1\}^* \cup \{0\}\{1\}^*\{00\}\{0\}^*\{1\}\{0, 1\}^*$  23. Let  $s_2$  be the only final state, and put transitions from  $s_2$  to itself on either input. Put a transition from the start state  $s_0$  to  $s_1$  on input 0, and a transition from  $s_1$  to  $s_2$  on input 1. Create state  $s_3$ , and have the other transitions from  $s_0$  and  $s_1$  (as well as both transitions from  $s_3$ ) lead to  $s_3$ . 25. Start state  $s_0$ , only final state  $s_3$ ; transitions from  $s_0$  to  $s_0$  on 0, from  $s_0$  to  $s_1$  on 1, from  $s_1$  to  $s_2$  on 0, from  $s_1$  to  $s_1$  on 1, from  $s_2$  to  $s_0$  on 0, from  $s_2$  to  $s_3$  on 1, from  $s_3$  to  $s_3$  on 0, from  $s_3$  to  $s_3$  on 1 27. Have five states, with only  $s_3$  final. For  $i = 0, 1, 2, 3$ , transition from  $s_i$  to itself on input 1 and to  $s_{i+1}$  on input 0. Both transitions from  $s_4$  are to itself. 29. Have four states, with only  $s_3$  final. For  $i = 0, 1, 2$ , transition from  $s_i$  to  $s_{i+1}$  on input 1 but back to  $s_0$  on input 0. Both transitions from  $s_3$  are to itself.

**31.**

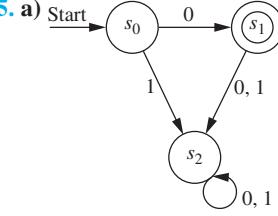
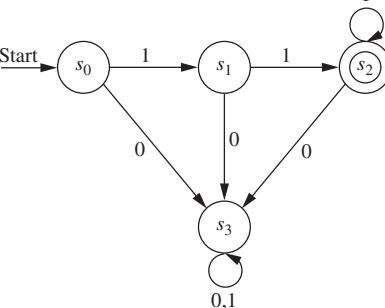
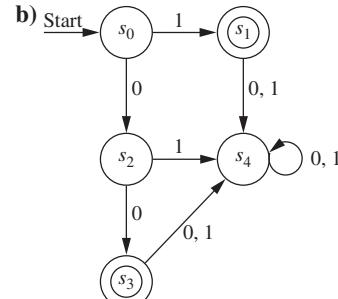
**33.** Start state  $s_0$ , only final state  $s_1$ ; transitions from  $s_0$  to  $s_0$  on 1, from  $s_0$  to  $s_1$  on 0, from  $s_1$  to  $s_1$  on 1; from  $s_1$  to  $s_0$  on 0

**35.**

**37.** Suppose that such a machine exists, with start state  $s_0$  and other state  $s_1$ . Because the empty string is not in the language but some strings are accepted, we must have  $s_1$  as the only final state, with at least one transition from  $s_0$  to  $s_1$ . Because the string 0 is not in the language, the transition from  $s_0$  on input 0 must be to itself, so the transition from  $s_0$  on input 1 must be to  $s_1$ . But this contradicts the fact that 1 is not in the language. **39.** Change each final state to a nonfinal state and vice versa. **41.** Same machine as in Exercise 25, but with  $s_0$ ,  $s_1$ , and  $s_2$  as the final states. **43.**  $\{0, 01, 11\}$  **45.**  $\{\lambda, 0\} \cup \{0^m 1^n \mid m \geq 1, n \geq 1\}$  **47.**  $\{10^n \mid n \geq 0\} \cup \{10^n 10^m \mid n, m \geq 0\}$  **49.** The union of the set of all strings that start with a 0 and the set of all strings that have no 0s

**51.**

**53.** Add a nonfinal state  $s_3$  with transitions to  $s_3$  from  $s_0$  on input 0, from  $s_1$  on input 1, and from  $s_3$  on input 0 or 1.

**55. a)****b)**   
**c)**

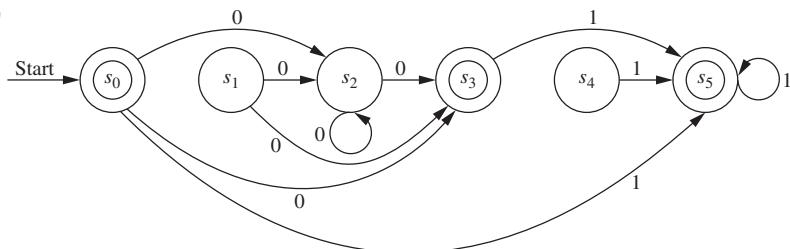
**57.** Suppose that  $M$  is a finite-state automaton that accepts the set of bit strings containing an equal number of 0s and 1s. Suppose  $M$  has  $n$  states. Consider the string  $0^{n+1} 1^{n+1}$ . By the pigeonhole principle, as  $M$  processes this string, it must encounter the same state more than once as it reads the first  $n+1$  0s; so let  $s$  be a state it hits at least twice. Then  $k$  0s in the input takes  $M$  from state  $s$  back to itself for some positive integer  $k$ . But then  $M$  ends up exactly at the same place after reading  $0^{n+k+1} 1^{n+1}$  as it will after reading  $0^{n+1} 1^{n+1}$ . Therefore, because  $M$  accepts  $0^{n+1} 1^{n+1}$  it also accepts  $0^{n+k+1} 1^{n+1}$ , which is a contradiction. **59.** We know from Exercise 58d that the equivalence classes of  $R_k$  are a refinement of the equivalence classes of  $R_{k-1}$  for each positive integer  $k$ . The equivalence classes are finite sets, and finite sets cannot be refined indefinitely (the most refined they can be is for each equivalence class to contain just one state). Therefore this sequence of refinements must remain unchanged from some point onward. It remains to show that as soon as we have  $R_n = R_{n+1}$ , then  $R_n = R_m$  for all  $m > n$ , from which it follows that  $R_n = R_*$ , and so the equivalence classes for these two relations will be the same. By induction, it suffices to show that if  $R_n = R_{n+1}$ , then  $R_{n+1} = R_{n+2}$ . Suppose that  $R_{n+1} \neq R_{n+2}$ . This means that there are states  $s$  and  $t$  that are  $(n+1)$ -equivalent but not

$(n + 2)$ -equivalent. Thus there is a string  $x$  of length  $n + 2$  such that, say,  $f(s, x)$  is final but  $f(t, x)$  is nonfinal. Write  $x = aw$ , where  $a \in I$ . Then  $f(s, a)$  and  $f(t, a)$  are not  $(n + 1)$ -equivalent, because  $w$  drives the first to a final state and the second to a nonfinal state. But  $f(s, a)$  and  $f(t, a)$  are  $n$ -equivalent, because  $s$  and  $t$  are  $(n + 1)$ -equivalent. This contradicts the fact that  $R_n = R_{n+1}$ . **61. a)** By the way the machine  $\bar{M}$  was constructed, a string will drive  $M$  from the start state to a final state if and only if that string drives  $\bar{M}$  from the start state to a final state. **b)** For a proof of this theorem, see a source such as *Introduction to Automata Theory, Languages, and Computation* (2nd Edition) by John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman (Addison-Wesley, 2000).

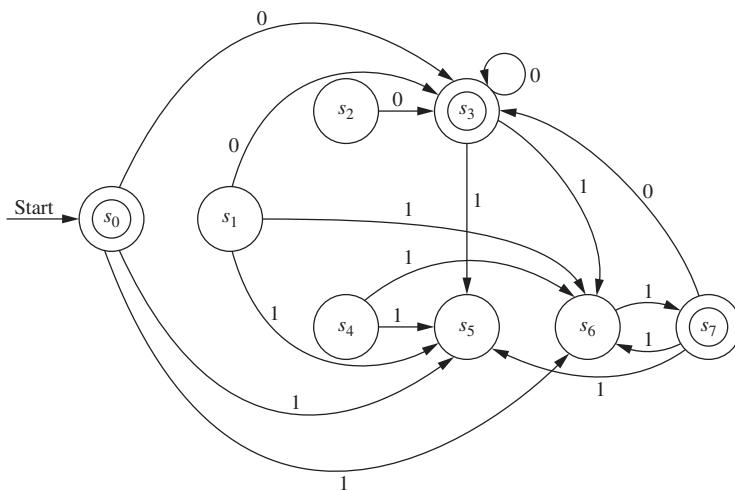
## Section 13.4

- 1. a)** Any number of 1s followed by a 0 **b)** Any number of 1s followed by one or more 0s **c)** 111 or 001 **d)** A string of any number of 1s or 00s or some of each in a row **e)**  $\lambda$  or a string that ends with a 1 and has one or more 0s before each 1 **f)** A string of length at least 3 that ends with 00 **3. a)** No **b)** No **c)** Yes **d)** Yes **e)** Yes **f)** No **g)** No **h)** Yes **5. a)**  $0 \cup 11 \cup 010$  **b)**  $000000^*$  **c)**  $(0 \cup 1)((0 \cup 1)(0 \cup 1))^*$  **d)**  $0^*10^*$  **e)**  $(1 \cup 01 \cup 001)^*$  **7. a)**  $00^*1$  **b)**  $(0 \cup 1)(0 \cup 1)(0 \cup 1)^*0000^*$  **c)**  $0^*1^* \cup 1^*0^*$  **d)**  $11(111)^*(00)^*$  **9. a)** Have the start state  $s_0$ , nonfinal, with no transitions. **b)** Have the start state  $s_0$ , final, with no transitions. **c)** Have the nonfinal start state  $s_0$  and a final state  $s_1$  and the transition from  $s_0$  to  $s_1$  on input  $a$ . **11.** Use an inductive proof. If the regular expression for  $A$  is  $\emptyset$ ,  $\lambda$ , or  $x$ , the result is trivial. Otherwise, suppose the regular expression for  $A$  is  $BC$ . Then  $A = BC$  where  $B$  is the set generated by  $B$  and  $C$  is the set generated by  $C$ . By the inductive hypothesis there are regular expressions  $B'$  and  $C'$  that generate  $B^R$  and  $C^R$ , respectively. Because  $A^R = (BC)^R = C^R B^R$ ,  $C'B'$  is a regular expression for  $A^R$ . If the regular expression for  $A$  is  $B \cup C$ , then the regular expression for  $A^R$  is  $B' \cup C'$  because  $(B \cup C)^R = (B^R) \cup (C^R)$ . Finally, if the regular expression for  $A$  is  $B^*$ , then it is easy to see that  $(B')^*$  is a regular expression for  $A^R$ .

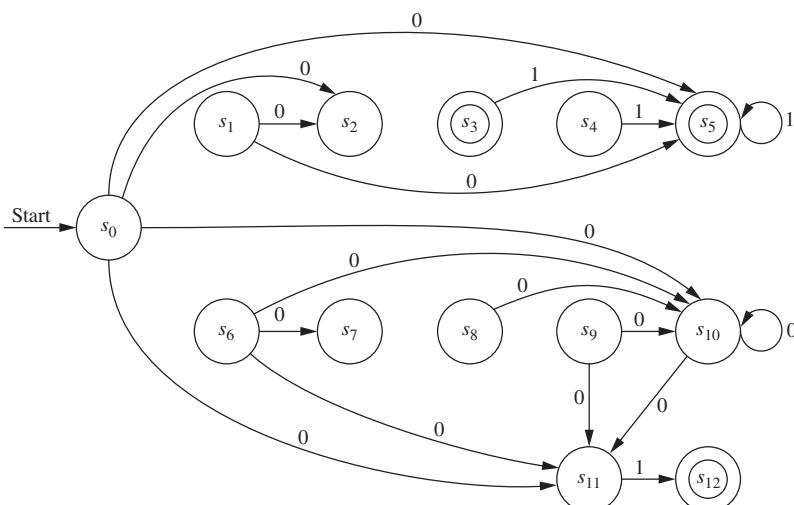
**13. a)**



b)



c)



**15.**  $S \rightarrow 0A$ ,  $S \rightarrow 1B$ ,  $S \rightarrow 0$ ,  $A \rightarrow 0B$ ,  $A \rightarrow 1B$ ,  $B \rightarrow 0B$ ,  $B \rightarrow 1B$    **17.**  $S \rightarrow 0C$ ,  $S \rightarrow 1A$ ,  $S \rightarrow 1$ ,  $A \rightarrow 1A$ ,  $A \rightarrow 0C$ ,  $A \rightarrow 1$ ,  $B \rightarrow 0B$ ,  $B \rightarrow 1B$ ,  $B \rightarrow 0$ ,  $B \rightarrow 1$ ,  $C \rightarrow 0C$ ,  $C \rightarrow 1B$ ,  $C \rightarrow 1$ .   **19.** This follows because input that leads to a final state in the automaton corresponds uniquely to a derivation in the grammar.   **21.** The “only if” part is clear because  $I$  is finite. For the “if” part let the states be  $s_{i_0}, s_{i_1}, s_{i_2}, \dots, s_{i_n}$ , where  $n = l(x)$ . Because  $n \geq |S|$ , some state is repeated by the pigeonhole principle. Let  $y$  be the part of  $x$  that causes the loop, so that  $x = uyy$  and  $y$  sends  $s_j$  to  $s_j$ , for some  $j$ . Then  $uy^k v \in L(M)$  for all  $k$ . Hence,  $L(M)$  is infinite.   **23.** Suppose that  $L = \{0^{2n}1^n \mid n = 0, 1, 2, \dots\}$  were regular. Let  $S$  be the set of states of a finite-state machine recognizing this set. Let  $z = 0^{2n}1^n$  where  $3n \geq |S|$ . Then by the pumping lemma,  $z = 0^{2n}1^n = uvw$ ,  $l(v) \geq 1$ , and  $uv^iw \in \{0^{2n}1^n \mid n \geq 0\}$ . Obviously  $v$  cannot contain both 0 and 1, because  $v^2$  would then contain 10. So  $v$  is all 0s or all 1s, and hence,  $uv^2w$  contains too many 0s or too many 1s, so it is not in  $L$ . This contradiction shows that  $L$  is not regular.   **25.** Suppose that the set of palindromes over  $\{0, 1\}$

were regular. Let  $S$  be the set of states of a finite-state machine recognizing this set. Let  $z = 0^n10^n$ , where  $n > |S|$ . Apply the pumping lemma to get  $uv^iw \in L$  for all nonnegative integers  $i$  where  $l(v) \geq 1$ , and  $l(uv) \leq |S|$ , and  $z = 0^n10^n = uvw$ . Then  $v$  must be a string of 0s (because  $n > |S|$ ), so  $uv^2w$  is not a palindrome. Hence, the set of palindromes is not regular.   **27.** Let  $z = 1$ ; then  $111 \notin L$  but  $101 \in L$ , so 11 and 10 are distinguishable. For the second question, the only way for  $1z$  to be in  $L$  is for  $z$  to end with 01, and that is also the only way for  $11z$  to be in  $L$ , so 1 and 11 are indistinguishable.   **29.** This follows immediately from Exercise 28, because the  $n$  distinguishable strings must drive the machine from the start state to  $n$  different states.   **31.** Any two distinct strings of the same length are distinguishable with respect to the language  $P$  of all palindromes, because if  $x$  and  $y$  are distinct strings of length  $n$ , then  $xx^R \in P$  but  $yx^R \notin P$ . Because there are  $2^n$  different strings of length  $n$ , Exercise 29 tells us that any deterministic finite-state automaton for recognizing palindromes must have at least  $2^n$  states. Because  $n$  is arbitrary, this is impossible.

## Section 13.5

**1. a)** The nonblank portion of the tape contains the string 1111 when the machine halts. **b)** The nonblank portion of the tape contains the string 011 when the machine halts. **c)** The nonblank portion of the tape contains the string 00001 when the machine halts. **d)** The nonblank portion of the tape contains the string 00 when the machine halts. **3. a)** The machine halts (and accepts) at the blank following the input, having changed the tape from 11 to 01. **b)** The machine changes every other occurrence of a 1, if any, starting with the first, to a 0, and otherwise leaves the string unchanged; it halts (and accepts) when it comes to the end of the string. **5. a)** Halts with 01 on the tape, and does not accept **b)** The first 1 (if any) is changed to a 0 and the others are left alone. The input is not accepted.

**7.**  $(s_0, 0, s_1, 1, R), (s_0, 1, s_0, 1, R)$    **9.**  $(s_0, 0, s_0, 0, R), (s_0, 1, s_1, 1, R), (s_1, s_1, 0, R), (s_1, 1, s_1, 0, R)$    **11.**  $(s_0, 0, s_1, 0, R), (s_0, 1, s_0, 0, R), (s_1, 0, s_1, 0, R), (s_1, 1, s_0, 0, R), (s_1, B, s_2, B, R)$    **13.**  $(s_0, 0, s_0, 0, R), (s_0, 1, s_1, 1, R), (s_1, 0, s_1, 0, R), (s_1, 1, s_0, 1, R), (s_0, B, s_2, B, R)$    **15.** If the input string is blank or starts with a 1 the machine halts in nonfinal state  $s_0$ . Otherwise, the initial 0 is changed to an  $M$  and the machine skips past all the intervening 0s and 1s until it either comes to the end of the input string or else comes to an  $M$ . At this point, it backs up one square and enters state  $s_2$ . Because the acceptable strings must have a 1 at the right for each 0 at the left, there must be a 1 here if the string is acceptable. Therefore, the only transition out of  $s_2$  occurs when this square contains a 1. If it does, the machine replaces it with an  $M$  and makes its way back to the left; if it does not, the machine halts in nonfinal state  $s_2$ . On its way back, it stays in  $s_3$  as long as it sees 1s, then stays in  $s_4$  as long as it sees 0s. Eventually either it encounters a 1 while in  $s_4$  at which point it halts without accepting or else it reaches the rightmost  $M$  that had been written over a 0 at the start of the string. If it is in  $s_3$  when this happens, then there are no more 0s in the string, so it had better be the case that there are no more 1s either; this is accomplished by the transitions  $(s_3, M, s_5, M, R)$  and  $(s_5, M, s_6, M, R)$ , and  $s_6$  is a final state. Otherwise the machine halts in nonfinal state  $s_5$ . If it is in  $s_4$  when this  $M$  is encountered, things start all over again, except now the string will have had its leftmost remaining 0 and its rightmost remaining 1 replaced by  $M$ s. So the machine moves, staying in state  $s_4$ , to the leftmost remaining 0 and goes back into state  $s_0$  to repeat the process.

**17.**  $(s_0, B, s_9, B, L), (s_0, 0, s_1, 0, L), (s_1, B, s_2, E, R), (s_2, M, s_2, M, R), (s_2, 0, s_3, M, R), (s_3, 0, s_3, 0, R), (s_3, M, s_3, M, R), (s_3, 1, s_4, M, R), (s_4, 1, s_4, 1, R), (s_4, M, s_4, M, R), (s_4, 2, s_5, M, R), (s_5, 2, s_5, 2, R), (s_5, B, s_6, B, L), (s_6, M, s_8, M, L), (s_6, 2, s_7, 2, L), (s_7, 0, s_7, 0, L), (s_7, 1, s_7, 1, L), (s_7, 2, s_7, 2, L), (s_7, M, s_7, M, L), (s_7, E, s_2, E, R), (s_8, M, s_8, M, L), (s_8, E, s_9, E, L)$  where  $M$  and  $E$  are markers, with  $E$  marking the left end of the input

**19.**  $(s_0, 1, s_1, B, R), (s_1, 1, s_2, B, R), (s_2, 1, s_3, B, R), (s_3, 1, s_4, 1, R), (s_1, B, s_4, 1, R), (s_2, B, s_4, 1, R), (s_3, B, s_4, 1, R)$

**21.**  $(s_0, 1, s_1, B, R), (s_1, 1, s_2, B, R), (s_1, B, s_6, B, R), (s_2, 1, s_3, B, R), (s_2, B, s_6, B, R), (s_3, 1, s_4, B, R), (s_3, B, s_6, B, R), (s_4, 1, s_5, B, R), (s_4, B, s_6, B, R), (s_6, B, s_{10}, 1, R), (s_5, 1, s_5, B, R), (s_5, B, s_7, 1, R), (s_7, B, s_8, 1, R), (s_8, B, s_9, 1, R), (s_9, B, s_{10}, 1, R)$

**23.**  $(s_0, 1, s_0, 1, R), (s_0, B, s_1, B, L), (s_1, 1, s_2, 0, L), (s_2, 0, s_2, 0, L), (s_2, 1, s_3, 0, R), (s_2, B, s_6, B, R), (s_3, 0, s_3, 0, R), (s_3, 1, s_3, 1, R), (s_3, B, s_4, 1, R), (s_4, B, s_5, 1, L), (s_5, 1, s_5, 1, L), (s_5, 0, s_2, 0, L), (s_6, 0, s_6, 1, R), (s_6, 1, s_7, 1, R), (s_6, B, s_7, B, R)$

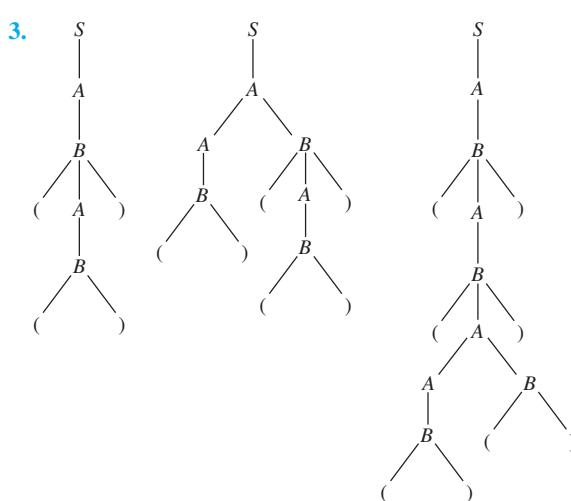
**25.**  $(s_0, 0, s_0, 0, R), (s_0, *, s_5, B, R), (s_3, *, s_3, *, L), (s_3, 0, s_3, 0, L), (s_3, 1, s_3, 1, L), (s_3, B, s_0, B, R), (s_5, 1, s_5, B, R), (s_5, 0, s_5, B, R), (s_5, B, s_6, B, L), (s_6, B, s_6, B, L), (s_6, 0, s_7, 1, L), (s_7, 0, s_7, 1, L), (s_0, 1, s_1, 0, R), (s_1, 1, s_1, 1, R), (s_1, *, s_2, *, R), (s_2, 0, s_2, 0, R), (s_2, 1, s_3, 0, L), (s_2, B, s_4, B, L), (s_4, 0, s_4, 1, L), (s_4, *, s_8, B, L), (s_8, 0, s_8, B, L), (s_8, 1, s_8, B, L)$

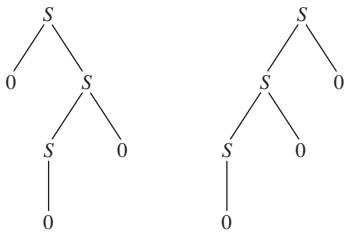
**27.** Suppose that  $s_m$  is the only halt state for the Turing machine in Exercise 22, where  $m$  is the largest state number, and suppose that we have designed that machine so that when the machine halts the tape head is reading the leftmost 1 of the answer. Renumber each state in the machine for Exercise 18 by adding  $m$  to each subscript, and take the union of the two sets of five-tuples. **29. a)** No **b)** Yes **c)** Yes **d)** Yes

**31.**  $(s_0, B, s_1, 1, L), (s_0, 1, s_1, 1, R), (s_1, B, s_0, 1, R)$

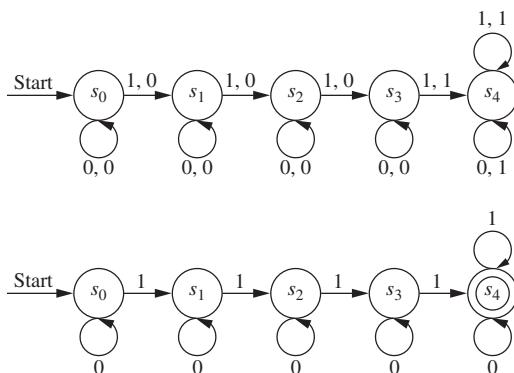
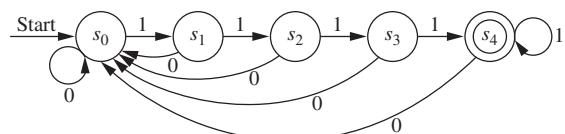
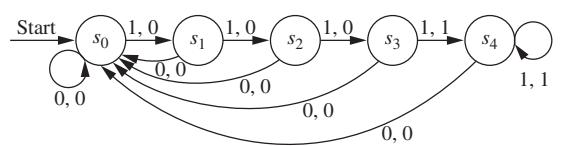
## Supplementary Exercises

**1. a)**  $S \rightarrow 00S111, S \rightarrow \lambda$    **b)**  $S \rightarrow AABS, AB \rightarrow BA, BA \rightarrow AB, A \rightarrow 0, B \rightarrow 1, S \rightarrow \lambda$    **c)**  $S \rightarrow ET, T \rightarrow 0TA, T \rightarrow 1TB, T \rightarrow \lambda, 0A \rightarrow A0, 1A \rightarrow A1, 0B \rightarrow B0, 1B \rightarrow B1, EA \rightarrow E0, EB \rightarrow E1, E \rightarrow \lambda$

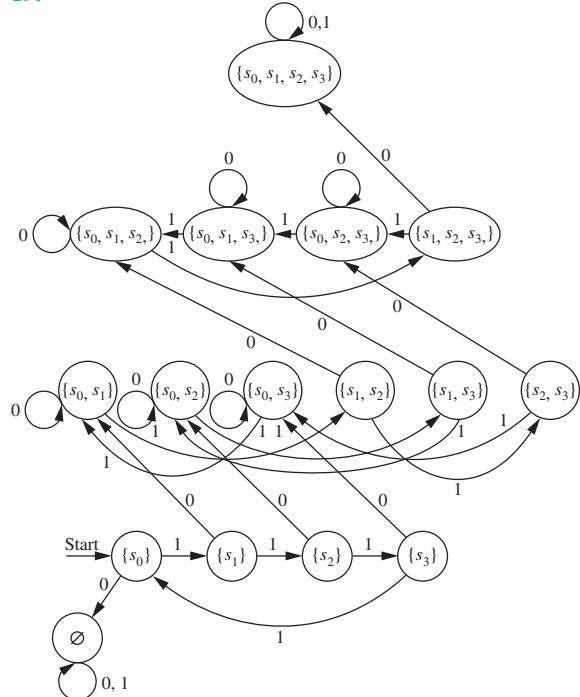
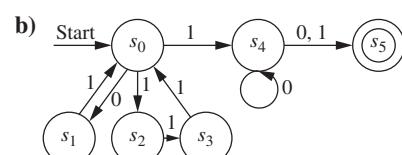
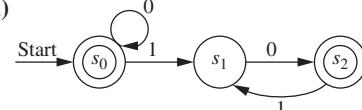
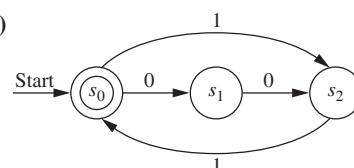


**5.**

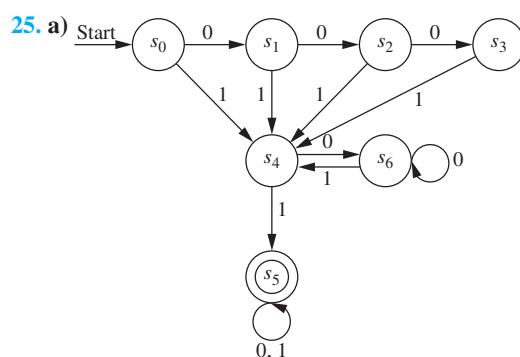
- 7.** No, take  $A = \{1, 10\}$  and  $B = \{0, 00\}$ .   **9.** No, take  $A = \{00, 000, 0000\}$  and  $B = \{00, 000\}$ .   **11.** a) 1   b) 1   c) 2  
d) 3   e) 2   f) 4

**13.****15.**

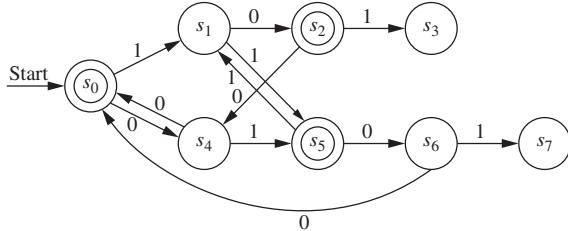
- 17.** a)  $n^{nk+1}m^{nk}$    b)  $n^{nk+1}m^n$

**19.****21. a)****c)**

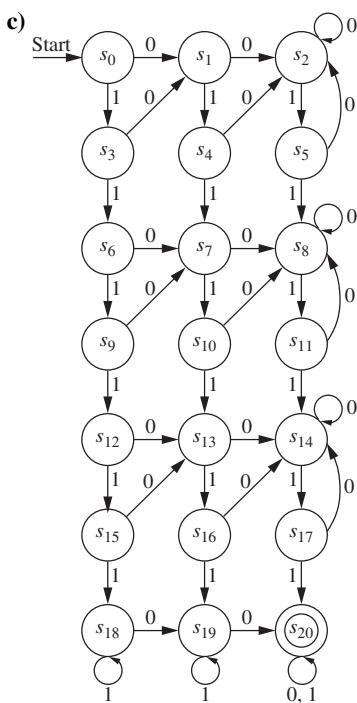
- 23.** Construct the deterministic finite automaton for  $A$  with states  $S$  and final states  $F$ . For  $\bar{A}$  use the same automaton but with final states  $S - F$ .

**25. a)**

b)



c)



**27.** Suppose that  $L = \{1^p \mid p \text{ is prime}\}$  is regular, and let  $S$  be the set of states in a finite-state automaton recognizing  $L$ . Let  $z = 1^p$  where  $p$  is a prime with  $p > |S|$  (such a prime exists because there are infinitely many primes). By the pumping lemma it must be possible to write  $z = uvw$  with  $l(uv) \leq |S|$ ,  $l(v) \geq 1$ , and for all nonnegative integers  $i$ ,  $uv^i w \in L$ . Because  $z$  is a string of all 1s,  $u = 1^a$ ,  $v = 1^b$ , and  $w = 1^c$ , where  $a + b + c = p$ ,  $a + b \leq n$ , and  $b \geq 1$ . This means that  $uv^i w = 1^a 1^b 1^c = 1^{(a+b+c)+b(i-1)} = 1^{p+b(i-1)}$ . Now take  $i = p+1$ . Then  $uv^i w = 1^{p(1+b)}$ . Because  $p(1+b)$  is not prime,  $uv^i w \notin L$ , which is a contradiction. **29.**  $(s_0, *, s_5, B, L)$ ,  $(s_0, 0, s_0, 0, R)$ ,  $(s_0, 1, s_1, 0, R)$ ,  $(s_1, *, s_2, *, R)$ ,  $(s_1, 1, s_1, 1, R)$ ,  $(s_2, 0, s_2, 0, R)$ ,  $(s_2, 1, s_3, 0, L)$ ,  $(s_2, B, s_4, B, L)$ ,  $(s_3, *, s_3, *, L)$ ,  $(s_3, 0, s_3, 0, L)$ ,  $(s_3, 1, s_3, 1, L)$ ,  $(s_3, B, s_0, B, R)$ ,  $(s_4, *, s_8, B, L)$ ,  $(s_4, 0, s_4, B, L)$ ,  $(s_5, 0, s_5, B, L)$ ,  $(s_5, B, s_6, B, R)$ ,  $(s_6, 0, s_7, 1, R)$ ,  $(s_6, B, s_6, B, R)$ ,  $(s_7, 0, s_7, 1, R)$ ,  $(s_7, 1, s_7, 1, R)$ ,  $(s_8, 0, s_8, 1, L)$ ,  $(s_8, 1, s_8, 1, L)$

## APPENDIXES

### Appendix 1

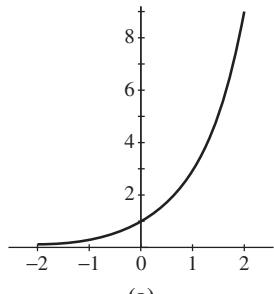
**1.** Suppose that  $1'$  is also a multiplicative identity for the real numbers. Then, by definition, we have both  $1 \cdot 1' = 1$  and  $1 \cdot 1' = 1'$ , so  $1' = 1$ . **3.** For the first part, it suffices to show that  $[(-x) \cdot y] + (x \cdot y) = 0$ , because Theorem 2 guarantees that additive inverses are unique. Thus  $[(-x) \cdot y] + (x \cdot y) = (-x + x) \cdot y$  (by the distributive law) =  $0 \cdot y$  (by the inverse law) =  $y \cdot 0$  (by the commutative law) =  $0$  (by Theorem 5). The second part is almost identical. **5.** It suffices to show that  $[(-x) \cdot (-y)] + [-(x \cdot y)] = 0$ , because Theorem 2 guarantees that additive inverses are unique:  $[(-x) \cdot (-y)] + [-(x \cdot y)] = [(-x) \cdot (-y)] + [(-x) \cdot y]$  (by Exercise 3) =  $(-x) \cdot [(-y) + y]$  (by the distributive law) =  $(-x) \cdot 0$  (by the inverse law) =  $0$  (by Theorem 5). **7.** By definition,  $-(-x)$  is the additive inverse of  $-x$ . But  $-x$  is the additive inverse of  $x$ , so  $x$  is the additive inverse of  $-x$ . Therefore  $-(-x) = x$  by Theorem 2. **9.** It suffices to show that  $(-x - y) + (x + y) = 0$ , because Theorem 2 guarantees that additive inverses are unique:  $(-x - y) + (x + y) = [(-x) + (-y)] + (x + y)$  (by definition of subtraction) =  $[(-y) + (-x)] + (x + y)$  (by the commutative law) =  $(-y) + [(-x) + (x + y)]$  (by the associative law) =  $(-y) + [(-x + x) + y]$  (by the associative law) =  $(-y) + (0 + y)$  (by the inverse law) =  $(-y) + y$  (by the identity law) =  $0$  (by the inverse law). **11.** By definition of division and uniqueness of multiplicative inverses (Theorem 4) it suffices to prove that  $[(w/x) + (y/z)] \cdot (x \cdot z) = w \cdot z + x \cdot y$ . But this follows after several steps, using the distributive law, the associative and commutative laws for multiplication, and the definition that division is the same as multiplication by the inverse. **13.** We must show that if  $x > 0$  and  $y > 0$ , then  $x \cdot y > 0$ . By the multiplicative compatibility law, the commutative law, and Theorem 5, we have  $x \cdot y > 0 \cdot y = 0$ . **15.** First note that if  $z < 0$ , then  $-z > 0$  (add  $-z$  to both sides of the hypothesis). Now given  $x > y$  and  $-z > 0$ , we have  $x \cdot (-z) > y \cdot (-z)$  by the multiplicative compatibility law. But by Exercise 3 this is equivalent to  $-(x \cdot z) > -(y \cdot z)$ . Then add  $x \cdot z$  and  $y \cdot z$  to both sides and apply the various laws in the obvious ways to yield  $x \cdot z < y \cdot z$ . **17.** The additive compatibility law tells us that  $w + y < x + y$  and (together with the commutative law) that  $x + y < x + z$ . By the transitivity law, this gives the desired conclusion. **19.** By Theorem 8, applied to  $1/x$  in place of  $x$ , there is an integer  $n$  (necessarily positive, because  $1/x$  is positive) such that  $n > 1/x$ . By the multiplicative compatibility law, this means that  $n \cdot x > 1$ . **21.** We must show that if  $(w, x) \sim (w', x')$  and  $(y, z) \sim (y', z')$ , then  $(w + y, x + z) \sim (w' + y', x' + z')$  and that  $(w \cdot y + x \cdot z, x \cdot y + w \cdot z) \sim (w' \cdot y' + x' \cdot z', x' \cdot y' + w' \cdot z')$ . Thus we are given that  $w + x' = x + w'$  and that  $y + z' = z + y'$ , and we want to show that  $w + y + x' + z' = x + z + w' + y'$  and that  $w \cdot y + x \cdot z + x' \cdot y' + w' \cdot z' = x \cdot y + w \cdot z + w' \cdot y' + x' \cdot z'$ . For the first of the desired conclusions, add the two given equations.

For the second, rewrite the given equations as  $w - x = w' - x'$  and  $y - z = y' - z'$ , multiply them, and do the algebra.

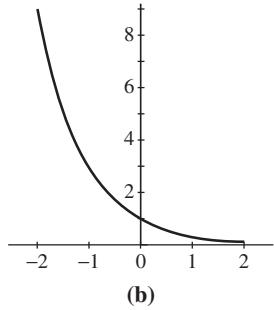
## Appendix 2

1. a)  $2^3$  b)  $2^6$  c)  $2^4$     3. a)  $2y$  b)  $2y/3$  c)  $y/2$

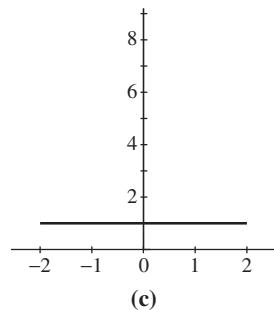
5.



(a)



(b)



(c)

## Appendix 3

1. After the first block is executed,  $a$  has been assigned the original value of  $b$  and  $b$  has been assigned the original value of  $c$ , whereas after the second block is executed,  $b$  is assigned the original value of  $c$  and  $a$  the original value of  $c$  as well.  
 3. The following construction does the same thing.

$i := \text{initial value}$   
**while**  $i \leq \text{final value}$   
 statement  
 $i := i + 1$

# Photo Credits

- CHAPTER 1** **Page 2:** © National Library of Medicine; **p. 5:** Library of Congress, Prints and Photographs Division [LC-USZ62-61664]; **p. 11:** © University Archives. Department of Rare Books and Special Collections. Princeton University Library; **p. 20:** Courtesy Indiana University Archives; **p. 29:** By permission of the London Mathematical Society; **p. 31:** © Mary Evans Picture Library; **p. 34:** Imaging Department © President and Fellows of Harvard College: Mabel Lisle Ducasse. Henry Maurice Sheffer (1883-1964), 20th century Pastel on paper; sight: 60.9 x 46.9 cm (24 x 18 1/2 in.) Harvard Art Museums/Fogg Art Museum, Harvard University Portrait Collection, Department of Philosophy, CNA1; **p. 38:** © Bettman/Corbis; **p. 50:** © The Granger Collection, New York; **p. 97, 98:** © Master and Fellows of Trinity College, Cambridge.
- CHAPTER 2** **Page 117:** Library of Congress, Prints and Photographs Division [LC-USZ62-74393]; **p. 119:** Library of Congress, Prints and Photographs Division [LC-USZ62-49535]; **p. 120:** © The Royal Society; **p. 122:** Scientists and Inventors Portrait File Collection, Archives Center, National Museum of American History, Smithsonian Institution; **p. 163:** Courtesy Neil Sloane and AT&T Shannon Lab; **p. 171:** © AIP Emilio Segre Visual Archives, Lande Collection.
- CHAPTER 3** **Page 192:** © from THE SCIENTISTS OF THE ANCIENT WORLD, by Anderson/Stephenson. pg. 90. Illustration by Marie le Glatin Keis; **p. 207(top):** Bachmann, Paul. *Die Arithmetik Der Quadratischen Formen*. Verlag und Druck von BG Teubner. Leipzig. Berlin. 1923; **p. 207(bottom):** Smith Collection, Rare Book & Manuscript Library, Columbia University in the City of New York; **p. 208:** © Stanford University News Service; **p. 227:** Photo courtesy of Dr. Stephen Cook/Photo by Yvonne Heath.
- CHAPTER 4** **Page 241:** Scientists and Inventors Portrait File Collection, Archives Center, National Museum of American History, Smithsonian Institution; **p. 259:** From Math Tutor Archive; **p. 261:** © Mary Evans Picture Library; **p. 263:** © Reed Hutchinson/UCLA; **p. 267:** Scientists and Inventors Portrait File Collection, Archives Center, National Museum of American History, Smithsonian Institution; **p. 269:** © Académie des sciences-Institut de France; **p. 282:** From *A History of Science, Technology and Philosophy in the 16th and 17th Centuries*, by Abraham Wolf. Reprinted by permission of Harper Collins Publishers Ltd. © Abraham Wolf; **p. 283:** © The Mathematical Association of America 2011. All rights reserved; **p. 299:** © David Meyer, ZDNet UK; **p. 300(top):** Courtesy Ronald L. Rivest; **p. 300(middle):** © Kristen Tsolis; **p. 300(bottom):** Courtesy Leonard Adleman.
- CHAPTER 5** **Page 348:** © The Granger Collection, New York; **p. 349:** © Mary Evans Picture Library; **p. 374:** Courtesy Tony Hoare; **p. 381:** © Matthew Naythons/TimePix/Getty Images.
- CHAPTER 6** **Page 400:** © The Granger Collection, New York; **p. 404:** Courtesy Mrs. Jane Burch; **p. 419:** © National Library of Medicine.
- CHAPTER 7** **Page 446:** © Roger-Viollet/The Image Works; **p. 447:** From *A Concise History of Mathematics* by Dirk Struik, 1967, Dover Publications; **p. 459:** © The Granger Collection, New York; **p. 472:** Courtesy Stephen Stigler; **p. 489:** © Académie des sciences-Institut de France; **p. 490:** Smith Collection, Rare Book & Manuscript Library, Columbia University in the City of New York.
- CHAPTER 8** **Page 507:** © KIK-IRPA, Brussels; **p. 508:** Photo courtesy of SIAM.

## C-2 Photo Credits

**CHAPTER 9** **Page 604:** Courtesy Stephen Warshall; **p. 623:** © Deutsches Museum; **p. 636:** © The Royal Society.

**CHAPTER 10** **Page 660:** © The Royal Society; **p. 695:** Scientists and Inventors Portrait File Collection, Archives Center, National Museum of American History, Smithsonian Institution; **p. 700:** © Master and Fellows of Trinity College, Cambridge; **p. 701(top):** © Rosemarie Dirac; **p. 701(bottom):** © Reverence to University History Photobase; **p. 706:** Julius Petersen: *Metoder og Teorier. Edition no. 6*, Det Schoenbergske Forlag, Copenhagen 1912. Photo courtesy Bjarne Toft; **p. 710:** Courtesy, Edsger W. Dijkstra; **p. 723:** Courtesy, the Archive of Polish Mathematicians, at the Institute of Mathematics of the Polish Academy of Sciences; **p. 728:** By permission of the London Mathematical Society.

**CHAPTER 11** **Page 750:** © Master and Fellows of Trinity College, Cambridge; **p. 763:** Courtesy, California State University, Santa Cruz; **p. 782:** Courtesy, the Archive of Polish Mathematicians, at the Institute of Mathematics of the Polish Academy of Sciences; **p. 798:** Courtesy, Robert Clay Prim; **p. 800:** Courtesy Joseph B. Kruskal. Photo by Rachel Kruskal.

**CHAPTER 12** **Page 812:** Courtesy MIT Museum; **p. 830:** Courtesy Maurice Karnaugh; **p. 837:** Courtesy Edward J. McCluskey; **p. 839:** Courtesy the Quine Family.

**CHAPTER 13** **Page 853:** © Donna Coveney/MIT; **p. 854(top):** Courtesy IBM, Almaden Research Center; **p. 854(bottom):** Courtesy Peter Naur; **p. 867:** Photo from *I Have a Photographic Memory* by Paul Halmos (1987), American Mathematical Society; **p. 872:** © Bettman/Corbis; **p. 886:** © Science Photo Library/Photo Researchers, Inc.; **p. 897:** © University Archives. Department of Rare Books and Special Collections. Princeton University Library.

**APPENDIX 1** **Page A-4:** © Hulton-Deutsch Collection/Corbis.

# Index of Biographies

- Ada, Augusta (Countess of Lovelace), 31  
Adleman, Leonard, 300  
al-Khowarizmi, Abu Ja'far Mohammed Ibn Musa, 192  
Archimedes, A–4  
Aristotle, 2
- Bachmann, Paul Gustav Heinrich, 207  
Backus, John, 854  
Bayes, Thomas, 472  
Bellman, Richard, 508  
Bernoulli, James, 459  
Bézout, Étienne, 269  
Bienaymé, Irenée-Jules, 489  
Boole, George, 5
- Cantor, Georg, 117  
Cardano, Girolamo, 446  
Carmichael, Robert Daniel, 283  
Carroll, Lewis (Charles Dodgson), 50  
Catalan, Eugène Charles, 507  
Cayley, Arthur, 750  
Chebyshev, Pafnuty Lvovich, 490  
Chomsky, Avram Noam, 853  
Church, Alonzo, 897  
Cocks, Clifford, 299  
Cook, Stephen, 227
- De Morgan, Augustus, 29  
Descartes, René, 122  
Dijkstra, Edsger Wybe, 710  
Dirac, Gabriel Andrew, 701  
Dirichlet, G. Lejeune, 500  
Dodgson, Charles (Lewis Carroll), 50
- Eratosthenes, 259  
Erdős, Paul, 636  
Euclid, 267  
Euler, Leonhard, 695
- Fermat, Pierre de, 282  
Fibonacci, 348
- Gauss, Karl Friedrich, 241  
Goldbach, Christian, 264
- Hall, Philip, 660  
Hamilton, William Rowan, 700  
Hardy, Godfrey Harold, 97  
Hasse, Helmut, 623  
Hilbert, David, 171  
Hoare, C. Anthony R., 374
- Hopper, Grace Brewster Murray, 872  
Huffman, David A., 763
- Karnaugh, Maurice, 830  
Kempe, Alfred Bray, 728  
Kleene, Stephen Cole, 867  
Knuth, Donald E., 208  
Kruskal, Joseph Bernard, 800  
Kuratowski, Kazimierz, 723
- Lamé, Gabriel, 349  
Landau, Edmund, 207  
Laplace, Pierre-Simon, 447  
Łukasiewicz, Jan, 782
- McCarthy, John, 381  
McCluskey, Edward J., 837  
Mersenne, Marin, 261
- Naur, Peter, 854
- Ore, Øystein, 701
- Pascal, Blaise, 419  
Peirce, Charles Sanders, 38  
Petersen, Julius Peter Christian, 706  
Prim, Robert Clay, 798
- Quine, Willard van Orman, 839
- Ramanujan, Srinivasa, 98  
Ramsey, Frank Plumpton, 404  
Rivest, Ronald, 300  
Russell, Bertrand, 110
- Shamir, Adi, 300  
Shannon, Claude Elwood, 812  
Sheffer, Henry Maurice, 34  
Sloane, Neil, 163  
Smullyan, Raymond, 20  
Stirling, James, 151
- Tao, Terence, 263  
Tukey, John Wilder, 11  
Turing, Alan Mathison, 886
- Vandermonde, Alexandre-Théophile, 420  
Venn, John, 120
- Warshall, Stephen, 604

# Index

- $3x + 1$  conjecture, 107  
23 problems  
Hilbert's list, 171, 895
- Absorption laws, 27  
for Boolean algebra, 815, 816  
for lattices, 637  
for propositions, 27  
for sets, 129
- Abstract definition of Boolean algebra, 817
- Academic collaboration graph, 645
- Academic papers, 645
- Academy, Plato's, 2
- Accepted language, 868
- Accepted strings, 868
- Accepting state  
finite-state automaton, 867
- Ackermann's function, 359
- Ackermann, Wilhelm, 359
- Acquaintanceship graph, 645  
paths in, 680
- Actors, 645
- Ada, Augusta (Countess of Lovelace), 29, 31
- Adapting proofs, 101
- Adders, 826–827  
full, 826, 843  
half, 826, 843
- Adding an edge to a graph, 664
- Addition,  
algorithm for, 250  
finite-state machine for, 862  
of functions, 141  
of matrices, 178  
of multisets, 138  
of subsets, 793  
of term in a geometric progression, 318–319
- Addition (rule of inference), 71
- Additive Compatibility Law, A-2
- Additive inverses  
for integers modulo  $m$ , 243
- Address system, universal, 772–773
- Adjacency list, 668
- Adjacency matrices, 669–671, 736  
counting paths between vertices by, 688–689
- Adjacent cells, 830
- Adjacent vertices, 651, 654, 735
- Adjective, 848
- Adleman, Leonard, 299, 300
- Advanced Encryption Standard (AES), 298
- Adverb, 848
- Affine cipher, 296, 306  
decryption, 296  
encryption, 296
- Affine transformation, 296
- Affirming the conclusion  
fallacy of, 75
- Agrawal, Manindra, 262
- Airline routes, 647
- Airline system, weighted graphs modeling, 708
- Airline ticket identification numbers, 293
- Al-Khowarizmi, Abu Ja'far Mohammed ibn Musa, 192
- Alcuin of York, 692
- Aleph null, 171
- Alexander the Great, 2
- Algebra  
fundamental theorem of, 241  
origin of term, 192
- Algebra, Boolean  
abstract definition of, 817  
definition of, 843  
identities of, 814–816
- ALGOL, 854
- Algorithm(s), 191–202, 218–229, 232,  
**div** and **mod**, 253  
addition, 250  
approximation, 235, 715  
binary search, 194  
binary search tree, 757–769  
Boolean product of zero–one matrices, 223  
brute-force, 224  
brute-force for closest pair of points, 225  
coloring graphs, 731, 734  
complexity of, 218–229  
average-case, 482–484  
of breadth-first search algorithms, 791  
of depth-first search algorithms, 789  
of Dijkstra's algorithm, 714  
of merge sort, 532
- computer time used by, 228
- constructing base  $b$  expansions, 248
- correctness of, 193
- deferred acceptance, 204, 343
- definition of, 191
- definiteness of, 193
- describing using English, 192
- Dijkstra's, 709–714, 737
- divide-and-conquer, 527, 565
- division, 239, 306  
division, 239  
effectiveness of, 193
- Euclidean, 267, 347
- Euler circuits, 696–697
- Euler paths, 697–698  
evaluating polynomials, 230
- extended Euclidean, 270, 273
- fast integer multiplication, 528–529
- fast matrix multiplication, 528
- Fibonacci numbers, 365–367  
finding maximum element, 193  
finiteness of, 193
- Fleury's, 697, 706
- Floyd's, 717
- Gale-Shapley, 204
- generality of, 193  
generating combinations, 437–438  
generating permutations, 434–436
- greedy, 198, 232, 324–325, 764, 798, 804  
greedy change-making, 199  
history of word, 192
- Huffman coding, 763–764
- inorder traversal, 779
- input to an, 193
- iterative, for Fibonacci numbers, 366
- Kruskal's, 799–801, 804
- linear search, 194, 232
- matrix multiplication, 222
- merge sort, 367–370
- minimum spanning trees, 798–802
- modular exponentiation, 253
- Monte Carlo, 463–465
- multiplication, 251
- optimal, 230
- origin of term, 246
- output of an, 193
- parallel, 661
- postorder traversal, 779
- preorder traversal, 777
- Prim's, 798–799, 804
- probabilistic, 445, 463–465, 494
- properties of, 193
- proving primality, 262
- pseudocode, 192–193
- reverse delete, 803
- recursive, 360–370, 378  
binary search, 363  
correctness of, 364–367  
for computing  $a^n$ , 361  
for computing greatest common divisor, 362  
for Fibonacci numbers, 365  
linear search, 363  
modular exponentiation, 363
- searching, 194–196, 232  
binary, 528  
breadth-first, 789–791, 804  
depth-first, 787–789, 804  
linear average-case complexity of, 483–484  
linear recursive sequential, 363
- sequential search, 194
- serial, 661
- shortest-path, 709–714
- Sollin's, 803
- sorting, 196–198  
topological, 627–629, 634
- spanning trees, 798–802
- specifying, 192
- ternary search, 203
- topological sorting, 628
- transitive closure, 603
- traversal, 773–779
- Warshall's, 603–606, 634
- Algorithm for finding maximum  
time complexity of, 219
- Algorithmic paradigm, 224, 232  
backtracking, 224  
brute force, 224  
divide-and-conquer, 224  
dynamic programming, 224, 507  
greedy algorithms, 198, 224  
probabilistic algorithms, 224, CH7
- Alice, 302, 303, 305
- Alice in Wonderland*, 50
- Alpha-beta pruning, 769, 809
- Alphabet, 849, 899
- Alphanumeric character, 391
- A Mathematician's Apology*, 97
- Ambiguous context-free grammar, 901
- Amino acids, 389
- Analytic Engine, 31
- Ancestors of vertex, 747, 803
- AND, 18
- AND gate, 21, 823, 843
- Antecedent of a conditional statement, 6
- Antichain, 637

- Antisymmetric relation, 577–578, 633  
 representing  
   using digraphs, 595  
   using matrices, 591–592
- Appel, Kenneth, 728
- Application(s)  
   of backtracking, 791–793  
   of Bayes' theorem, 471–472  
   of congruences, 287  
   of graphs colorings, 731–732  
   of graph theory, 644–649, 658–663, 680–681, 687,  
     698, 702, 731–732  
   of inclusion–exclusion, 558–564  
   of pigeonhole principle, 403–405  
   of propositional logic, 16–22  
   of satisfiability, 32–34  
   of trees, 757–769
- Approximation algorithm, 235, 715
- Arborescence, 807
- Archimedean property, A-5
- Archimedes, A-4
- Archimedes' screw, A-4
- Argument, 69, 110  
   Cantor diagonalization, 173, 186  
   premises of, 69  
   propositional logic, 70  
   valid, 69, 110
- Argument form, 70, 110  
   valid, 70, 110
- Aristotle, 2, 3
- Arithmetic  
   fundamental theorem of, 306, 336–337  
   modular, 240–306  
   modulo  $m$ , 243
- Arithmetic mean, 100, 332
- Arithmetoricorum Libri Duo* (Maurolico), 313
- Arithmetic progression, 186  
   of primes, 262, 263
- Array linear, 662  
   two-dimensional, 662
- Arrow, Peirce, 36
- Ars Conjectandi* (Bernoulli), 459
- Art Gallery  
   problem, 735  
   theorem, 735
- Article, 848
- Articulation points, 683
- Artificial intelligence, 138, 380  
   fuzzy logic in, 16
- Assertion  
   final, 372, 378  
   initial, 372, 378
- Assignment, stable, 343  
   optimal, 343
- Assignment of jobs, 565
- Assignments, frequency, 732
- Assignment statements, A-11–A-12
- Associated homogenous recurrence relations, 520
- Associative laws, 27  
   for addition, A-1  
   for Boolean algebra, 815, 818  
   for lattices, 637  
   for multiplication, A-1  
   for propositions, 27  
   for sets, 129
- Associativity  
   for integers modulo  $m$ , 243
- Asymmetric relation, 582
- Asymptotic functions, 218
- Asynchronous transfer mode (ATM), 149
- AT&T network, 682
- Autokey cipher  
   keystream, 309
- Autokey ciphers, 309
- Automated theorem proving, 114
- Automaton  
   deterministic, 872, 873  
   deterministic finite-state, 899  
   finite-state, 866–872  
     nondeterministic, 873  
   linear bounded, 886  
   nondeterministic, 873  
   nondeterministic finite-state, 899  
   pushdown, 886  
   quotient, 878  
     set not recognized by, 885
- Average-case complexity of algorithms, 482–484
- Average-case time complexity, 220, 232  
   of linear search, 221
- Avian influenza, 476
- AVL-trees, 808
- Award  
   Turing, 854
- Axiom(s), 185, A-1–A-5  
   Completeness, A-2  
   field, A-1  
   for real numbers, A-1–A-2  
   for set of positive integers, A-5  
   in proving basic facts, A-3–A-5  
   of Mathematical Induction, A-5  
   order, A-2  
   Zermelo–Fraenkel, 176
- B-tree of degree  $k$ , 805
- Bézout coefficients, 270, 306
- Bézout's identity, 270
- Bézout's theorem, 269, 306
- Bézout, Étienne, 269
- Bachmann, Paul, 206
- Bachmann, Paul Gustav Heinrich, 207
- Back edges, 788
- Back substitution  
   solving systems of linear congruences using, 279
- Backtracking, 787–789, 804  
   applications of, 791–793  
   in directed graphs, 793
- Backus, John, 854
- Backus–Naur form, 899
- Backus–Naur form (BNF), 854
- Backward differences, 513
- Backward substitution  
   iteration using, 160
- Bacon, Kevin, 680–681
- Bacon number, 680
- Bacteria, 168, 501
- Balanced rooted trees, 753, 804
- Balanced strings of parentheses, 382
- Balanced ternary expansions, 256
- Bandwidth of graph, 741
- Bar  
   chocolate, 342
- Barber paradox, 16
- Base  $b$   
   expansions, 246, 306  
     algorithm for constructing, 248  
   Miller's test for, 286  
   pseudoprime to the, 282  
     strong pseudoprime to, 286
- Base-10 expansions, 246
- Base conversion, 247
- Bases of DNA molecules, 389
- BASIC, 391
- Basis, vertex, 691
- Basis step, 313, 333, 349, 377, 752, 767
- Basketball, 434
- Bayes' theorem, 468–475  
   generalized, 472
- Bayes, Thomas, 470, 472
- Bayesian spam filters, 472–475
- Begging the question, 90, 110
- Bell, E. T., 618
- Bellman, Richard, 508
- Bell numbers, 618
- Bernoulli's inequality, 330
- Bernoulli, James, 458, 459
- Bernoulli family tree, 745
- Bernoulli trial, 479, 494  
   and binomial distribution, 458–460
- Bernstein, Felix, 175
- Bi-implication, 9
- Biconditional, 9–10, 110  
   implicit use of, 10  
   statement, 9  
   truth tables for, 9
- Biconnected graph, 684
- Bicycle racers, 475
- Bidirectional bubble sort, 233
- Bienaymé's formula, 489
- Bienaymé, Irénée-Jules, 489
- big- $O$  estimates  
   for combinations of functions, 212  
   for exponential functions, 212  
   of important functions, 209  
   of  $n!$ , 210  
   for polynomials, 209, 212  
   for products of functions, 213  
   for sums of functions, 212  
   of logarithms, 212
- Big- $O$  notation, 205–232  
   careless use of, 216  
   origin of, 206
- Big- $O$  relationship  
   witnesses to, 205
- big-Omega ( $\Omega$ ) notation, 214, 232
- big-Theta ( $\Theta$ ) notation, 215, 232
- Big-Theta estimates for polynomials, 216
- Bigit, 11
- Bijection, 144, 186
- Bijective proofs, 412
- Binary coded decimal expansion, 836
- Binary digit, 11
- Binary expansions, 246
- Binary insertion sort, 196, 203
- Binary relation, 573, 633
- Binary representation, 306
- Binary search  
   worst-case complexity, 220
- Binary search algorithm, 194, 232, 528  
   recursive, 363
- Binary search trees, 757–759, 804
- Binary trees, 352–353, 748, 749  
   definition of, 804  
   extended, 352  
     full, 352–355  
     height of, 355  
   number of vertices of, 355  
   with prefix code, 762–763
- Binit, 11
- Binomial coefficients, 410, 415–421, 439  
   extended, 539
- Binomial distribution, 458–460
- Binomial expression, 415
- Binomial theorem, 415–418, 439  
   extended, 540
- Binomial trees, 805
- Bipartite graphs, 656–658, 736  
   coloring of, 657
- Bird flu, 476
- Birthday problem, 461–463

## I-4 Index

- Bit, 11, 110  
  origin of term, 11  
  parity check, 290  
Bit operation(s), 11, 110  
Bits, 149  
Bit string, 12, 110 subitem representing set with, 134  
Bit strings  
  counting, 388  
  without consecutive 0s, 505  
  decoding, 762–763  
  generating next largest, 437  
  in interstellar communication, 476  
  length of, 505  
Bitwise operation, 110  
Bitwise operators, 12  
Block  
  Sudoku, 32  
Block cipher, 297, 306  
Blocks of statements, A-14–A-15  
BNF (Backus–Naur form), 854  
Boat, 692  
Bob, 302, 305  
Bonferroni's inequality, 467  
Book number of graph, 744  
Boole's inequality, 467  
Boole, George, 3, 5, 472, 811  
Boolean algebra, 811–846  
  abstract definition of, 815  
  definition of, 843  
  identities, 27  
  identities of, 814–816  
Boolean expressions, 812–814, 843  
  dual of, 816, 843  
Boolean functions, 811–819, 843  
  dual of, 816, 843  
  functionally complete set of operators for, 821  
  implicant of, 832  
  minimization of, 828–840, 843  
  representing, 813–821  
  self-dual, 844  
  threshold, 845  
Boolean power of a square zero–one matrix, 183  
Boolean product, 811–812, 821, 843  
  of two zero–one matrices, 182  
Boolean product of zero–one matrices  
  algorithm for, 223  
Boolean searches, 18  
Boolean sum, 811–812, 821, 843  
Boolean variable, 11, 110, 814, 820, 843  
Boruvka, Otakar, 800  
Bots, 794  
Bottom-up parsing, 853  
Bound  
  greatest lower, 634  
  least upper, 634, A-2  
  of poset, 625  
  of lattice, 637  
  of poset, 625–626, 634  
  upper, A-2  
Bounded lattices, 637  
Bound occurrence of variable, 44  
Bound variable, 44, 110  
Boxes, distributing objects into, 428–431  
  distinguishable, 428  
  indistinguishable, 428  
Breadth-first search, 789–791, 804  
Breaking codes, 296  
Bridge, in graphs, 683  
Bridge problem, Königsberg, 693–694, 696, 697  
Brute-force algorithm, 224  
Brute-force algorithm for closest pair of points, 225  
Brute-force paradigm, 232
- Bubble sort, 196, 232  
  bidirectional, 233  
  worst-case complexity, 221  
Bug, computer, 872  
Building arguments using rules of inference, 73  
Buoyancy, principle of, A-4  
Busy beaver function, 896, 899  
Butane, 750  
Byte, 149
- Cabbage, 692  
Cactus, 806  
CAD programs, 834  
Caesar, Julius, 294  
Caesar cipher, 294  
  decryption, 295  
  encryption, 294  
  key, 295  
Cake, division of, 331  
Calculus  
  predicate, 40  
  propositional, 3  
Call graphs, 646  
  connected components of, 682  
Canterbury Puzzles, The (Dudeney), 504  
Cantor's theorem, 177  
Cantor, Georg, 117, 118, 173, 176  
Cantor diagonalization argument, 173, 186  
Cantor digits, 438  
Cantor expansion, 256, 438  
Cardano, Girolamo, 445  
Card hands, 411, 429  
Cardinality, 121, 170–175, 185, 186  
  of a countable set ( $\aleph_0$ ), 171, 186  
  of union of sets, 128  
Cardinality of the set of real numbers, 186  
Cardinal number, 121  
Carmichael, Robert Daniel, 283  
Carmichael number, 283, 306  
Carroll, Lewis (Charles Dodgson), 50  
Carry, 250  
Cartesian product, 123  
Cartesian product of more than two sets, 123  
Cases, proof by, 92–110  
Cash box, 424–425  
Catalan, Eugène Charles, 507  
Catalan numbers, 507  
Caterpillar, 807  
Cathy, 305  
Cayley, Arthur, 745, 750  
Ceiling function, 149, 186  
  properties of, 150  
Celebrity problem, 332  
Cells, adjacent, 830  
Center, 757  
Ceres, 241  
Chain, 637  
Chain letter, 753  
Character cipher, 297, 306  
Characteristic equation, 515  
Characteristic roots, 515  
Chebyshev's inequality, 491, 495  
Chebyshev, Pafnuty Lvovich, 490, 491  
Checkerboard  
  tiles, 103  
Check digit, 289–292  
  for airline ticket identification numbers, 293  
ISBN, 291  
ISBN-13, 308  
ISSN, 293  
RTN, 308  
UPC, 290
- Checkerboard, 103
- Checkerboard, tiling of, 326, 333  
Cheese, in Reve's puzzle, 504  
Chen, J. R., 264  
Chess, game tree for, 769  
Chessboard  
  n-queens problem on, 792–793  
  knight's tour on, 707  
  squares controlled by queen on, 740  
Chi (Greek letter), 728  
Chickens, 741  
Child of vertex, 747, 803  
Chinese meal, preparing, 635  
Chinese postman problem, 698, 744  
Chinese remainder theorem, 277, 306  
Chocolate bar, 342  
Chomp, 98, 342, 638  
  winning strategy for, 98  
Chomsky, Avram Noam, 847, 851, 853  
Chomsky classification of grammars, 851–852  
Chromatic number, 728, 737  
  k-critical, 734  
  edge, 734  
  of graph, 728  
Chung-Wu Ho, 340  
Church, Alonzo, 897  
Church–Turing thesis, 893  
Cipher  
  affine, 296, 306  
  autokey, 309  
  block, 306  
  Caesar, 294  
  character, 297, 306  
  monoalphabetic, 297  
  shift, 295, 306  
  transposition, 297  
  Vigenère, 304  
Cipher block, 297  
Circuits  
  combinational, 823  
  depth of, 828  
  examples of, 823–825  
  minimization of, 828–840  
digital, 21  
Euler, 693–698, 736  
Hamilton, 698–703, 736  
in directed graph, 599, 633  
in graph, 679  
logic, 20, 110  
multiple output, 826  
simple, 679  
Circular permutation, 415  
Circular reasoning, 90, 110  
Circular relation, 636  
Circular table, 394  
Citation graphs, 646  
Civil War, 38  
C language identifier, 857  
Class  
  congruence, 241  
  equivalence, 610–611, 633  
  and partitions, 612–614  
  definition of, 610  
  representative of, 610  
of nondeterministic polynomial-time problems, 900  
NP, 227, 896  
of NP-complete problems, 897, 900  
of polynomial-time problems, 227, 896, 900  
  Class P, 227, 896, 900  
Class A addresses, 392  
Class B addresses, 392  
Class C addresses, 392  
Class D addresses, 392  
Class E addresses, 392  
Clause, 74

- Climbing rock, 163  
 Clique, 739  
 Clock, 240  
 Closed formula  
     for terms of a sequence, 159  
 Closed interval, 117  
 Closed walk, 679  
 Closest-pair problem, 532–535  
 Closure  
     for integers modulo  $m$ , 243  
     Kleene, 899  
 closure  
     Kleene, 866  
 Closure, Kleene, 866  
 Closure laws  
     for addition, A-1  
     for multiplication, A-1  
 Closures of relations, 597–606, 633  
     reflexive, 598, 633  
     symmetric, 598, 634  
     transitive, 597, 600–603, 634  
         computing, 603–606  
 Coast Survey, U. S., 38  
 COBOL, 854, 872  
 Cocks, Clifford, 299  
 Code  
     breaking, 296  
 Codes, Gray, 702–703  
 Codes, prefix, 762–764, 804  
 Codeword enumeration, 506  
 Coding, Huffman, 763–764, 804  
 Codomain  
     of a function, 139, 186  
     of partial function, 152  
 Coefficient(s)  
     Bézout, 270, 306  
     binomial, 409, 415–421, 439  
     constant  
         linear homogenous recurrent relations with, 515–520, 565  
         linear nonhomogenous recurrent relations with, 520–524, 565  
     extended, 539  
     multinomial, 434  
 Collaboration graphs, 645  
     paths in, 680  
 Collatz problem, 107  
 Collection of sets  
     intersection of, 133  
     union of, 133  
 Collision  
     in hashing, 288  
     in hashing functions, probability of, 462–463  
 Coloring  
     chromatic number in, 728  
     of bipartite graphs, 657  
     of graphs, 727–732, 736  
     of maps, 727  
 Combinational circuits, 823  
     depth of, 828  
     examples of, 823–825  
     minimization of, 828–840  
 Combinations, 409–413  
     generating, 437–438  
     of events, 449–450, 455–456  
     with repetition, 424–427  
 Combinatorial identities, 417–421  
 Combinatorial proof, 412, 439  
 Combinatorics, 385, 404, 431, 439  
 Comments in pseudocode, A-12  
 Common difference, 157  
 Common errors  
     with exhaustive proofs, 95  
     with proofs by cases, 95  
 Common ratio, 157  
 Commutative group, 244  
 Commutative laws, 27  
     for addition, A-1  
     for Boolean algebra, 815, 818  
     for lattices, 637  
     for multiplication, A-1  
     for sets, 129  
 Commutative ring, 244  
 Commutativity  
     for integers modulo  $m$ , 243  
 Comparable elements in poset, 619, 633  
 Compatibility laws, A-2  
 Compatible total ordering, 628, 634  
 Compilers, 611, 872  
 Complement, 811, 843  
     double, law of, in Boolean algebra, 815, 818, 819  
     of Boolean function, 811  
 Complementary event, 449  
 Complementary graph, 667  
 Complementary relation, 582  
 Complementation law, 129  
 Complemented lattice, 637, 817  
 Complement law, 129  
 Complement of a fuzzy set, 138  
 Complement of a set, 128, 129, 186  
 Complete  $m$ -ary tree, 756  
 Complete  $m$ -partite graph, 738  
 Complete bipartite graph, 658, 736  
 Complete graphs, 655, 736  
 Complete induction, 334  
 Complexity  
     computational, 219  
     constant, 225  
     exponential, 225  
     factorial, 225  
     linear, 225  
     linearithmic, 225  
     logarithmic, 225  
     of algorithm for Boolean product of zero-one matrices, 223  
     of matrix multiplication, 223  
     polynomial, 225  
     space, 219, 232  
     time, 219, 232  
 Complexity of algorithms, 218–229  
     average-case, 482–484  
     computational  
         of Dijkstra's algorithm, 714  
         of breadth-first search algorithms, 791  
         of depth-first search algorithms, 789  
         of merge sort, 532  
 Complexity of merge sort, 532  
 Complex numbers  
     set of, 116  
 Components of graphs, connected, 682–685  
     strongly, 686, 736  
 Composite, 257  
 Composite integer, 306  
 Composite key, 585, 633  
 Composite of relations, 580, 633  
 Composition of functions, 146, 186  
 Composition rule, 373  
 Compound interest, 160  
 Compound proposition, 25, 109  
     dual of, 35  
     satisfiable, 30, 110  
 Compound propositions, 3  
     consistent, 110  
     equivalences of, in Boolean algebra, 812  
     equivalent, 8  
     logically equivalent, 25, 110  
     truth tables of, 10  
     well-formed formula for, 354  
 Computable  
     function, 175  
 Computable function, 177, 186, 896, 900  
 Computable numbers, 886  
 Computation, models of, 847–897  
 Computational complexity, 219  
 Computational complexity of algorithms  
     of Dijkstra's algorithm, 714  
     Turing machine in making precise, 894  
 Computational geometry, 338–340, 532–535  
 Computer-aided design (CAD) programs, 834  
 Computer arithmetic with large integers, 279  
 Computer debugging, 872  
 Computer file systems, 750  
 Computer network, 641  
     interconnection networks, 661–663  
     local area networks, 661  
     multicasting over, 786  
     with diagnostic lines, 642  
     with multiple lines, 642  
     with multiple one-way lines, 642  
     with one-way lines, 643  
 Computer programming, 854  
 Computer representation of sets, 134  
 Computer science, 854  
 Computer time used by algorithms, 228  
 Computer virus  
     invention of term, 300  
     transmission, 441  
 Concatenation, 350, 866, 899  
 Conclusion, 70, 110  
 Conclusion of a condition statement, 6  
 Concurrent processing, 647  
 Condition  
     necessary, 6  
     sufficient, 6  
 Conditional constructions, A-12–A-13  
 Conditional probability, 453, 456–457, 494  
 Conditional statement(s), 6–9  
     contrapositive, 110  
     contrapositive of, 8  
     converse, 110  
     converse of, 8  
     for program correctness, 373–375  
     inverse of, 8  
     truth table for, 6  
 Conditions  
     don't care, 836–837  
     initial, 158, 565  
 Congruence, 240, 306  
     applications of, 287–292  
     linear, 275, 306  
 Congruence class, 241  
 Congruence modulo  $m$ , 609–610, 633  
 Congruence quadratic, 285  
 Congruent to, 240, 306  
 Conjecture, 81, 110  
      $3x + 1$ , 107  
     Frame's, 513  
     Goldbach's, 264  
     twin prime, 264  
 Conjectures about primes, 263  
 Conjunction, 4, 110, 843  
     distributive law of disjunction over, 27  
     negating, 28  
     truth table for, 4  
 Conjunction (rule of inference), 71  
 Conjunctive normal form, 820  
 Connected components of graphs, 682–685  
     strongly, 686, 736

## I-6 Index

- Connected graphs, 678–689  
  directed, 685–687  
  planar simple, 719–723  
  strongly, 685  
  undirected, 681–685  
  weakly, 686  
Connecting vertices, 651  
Connectives  
  logical, 4  
Connectivity  
  edge, 684  
  vertex, 684  
Connectivity relation, 600, 633  
Consequence of a condition statement, 6  
Consistent compound propositions, 110  
Consistent system specifications, 18  
Constant coefficients  
  linear homogenous recurrent relations with, 515–520, 565  
  linear nonhomogenous recurrent relations with, 520–524, 565  
Constant complexity, 225  
Constructing logical equivalences, 29  
Construction  
  automaton that recognizes a regular set, 881  
Construction of the real numbers, A-5  
Constructions  
  conditional, A-12–A-13  
  loop, A-14–A-15  
Constructive existence proof, 96, 110  
Contains, 116  
Context-free grammar(s), 851, 886  
  ambiguous, 901  
Context-free language, 851  
Context-sensitive grammars, 851  
Context-sensitive languages, 886  
Contingency, 25, 110  
Continuum hypothesis, 175, 186  
Contraction, edge, 664  
Contradiction, 25, 110  
  proof by, 110  
Contradiction, proof by, 86  
Contraposition, 83  
  proof by, 110  
Contrapositive, 8, 110  
  of a conditional statement, 110  
Control unit  
  Turing machine, 889  
Converse, 8, 110  
  of a conditional statement, 110  
  of directed graph, 668  
Conversion  
  between bases, 247  
  between binary and hexadecimal, 249  
  between binary and octal, 249  
Convex polygon, 338  
Cook, Stephen, 227  
Cook–Levin theorem, 227  
Cookie, 426  
Corollary, 110  
Correctness  
  of an algorithm, 193  
  of programs, 372–378  
    conditional statements for, 373–375  
    loop invariants for, 375–376  
    partial, 372  
    program verification for, 372–373  
    rules of inference for, 373  
  of recursive algorithms, 364–365  
Correspondence  
  one-to-one, 144, 186  
Countability of set of rational numbers, 172  
Countable set, 171, 186  
  cardinality of, 186  
Counterexample(s), 88, 102, 110  
Counting, 385–444, 501–571  
  basic principles of, 385–390  
  bit strings, 388  
    without consecutive 0s, 505  
  Boolean functions, 814  
  by distributing objects into boxes, 428–431  
  combinations, 409–413  
  derangements, 562, 566  
  functions, 387  
    generating functions for, 541–546  
  Internet addresses, 392  
  one-to-one functions, 387  
  onto functions, 560–562, 566  
  passwords, 391  
  paths between vertices, 688–689  
  permutations, 407–409  
  pigeonhole principle and, 399–405  
  reflexive relations, 578  
  relations, 578–579  
  subsets of finite set, 388  
  telephone numbers, 387  
  tree diagrams for, 394–395  
  variable names, 391  
  with repetition, 423  
Covariance, 494  
Covering relation, 623, 631  
Covers, 631  
CPM (Critical Path Method), 639  
C programming language, 398, 611  
Crawlers, 794  
Cricket, 97  
Criterion  
  Euler's, 286  
Critical Path Method (CPM), 639  
Crossing number, 726  
Cryptanalysis, 296, 306  
  for shift cipher, 296  
  Vigenère cipher, 305  
Cryptographic protocols, 302  
Cryptosystem, 297–306  
  definition of, 297  
  private key, 298, 306  
  public key, 298  
  RSA, 299, 306  
  shift cipher, 298  
Cunningham numbers, 262  
Cut  
  edge, 683, 684  
  set, 806  
  vertex, 683, 684  
Cycle  
  in directed graph, 599, 633  
  with  $n$  vertices, 655  
Cylinder, 831  
Czehanowski, Jan, 800  
Database  
  composite key of, 585  
  intension of, 585  
  primary key of, 585  
  records in, 584  
    relational model of, 584–586, 633  
Database query language, 588–589  
Data compression, 763  
Datagrams, 399  
Datalogy, 854  
Datatype, 117  
David Hilbert, 171  
de Bruijn sequences, 744  
Debugging, 872  
Decidable problem, 895  
Decimal expansion of a real number, 174  
Decimal expansions, 246  
Decision problems, 894, 900  
Decision trees, 760–762, 804  
Deck, standard, 402  
Decreasing function, 143  
Decryption, 295, 306  
  affine cipher, 296  
  Caesar cipher, 295  
  RSA, 300  
Decryption key  
  RSA system, 301  
Dedekind, Richard, 175  
Deductive reasoning, 312  
Deep-Blue, 769  
Deferred acceptance algorithm, 204, 343  
Definiteness  
  of an algorithm, 193  
Definition  
  recursive, 311, 344–357  
Degree  
  of  $n$ -ary relations, 584  
  of linear homogenous recurrence relations, 514  
  of region, 722  
  of vertex in undirected graph, 652  
Degree-constrained spanning tree, 806  
Degree of membership in a fuzzy set, 138  
de la Vallée-Poussin, Charles-Jean-Gustave-Nicholas, 262  
Delay machine, 861–862  
de Méré, Chevalier, 452  
De Morgan's laws, 26, 27  
  for Boolean algebra, 815, 819  
  for propositions, 31, 28–31  
  for quantifiers, 47  
  for sets, 129  
  proving by mathematical induction, 323–324  
De Morgan, Augustus, 26, 29, 729  
Dense  
  graph, 670  
  poset, 632  
Denying the hypothesis, fallacy of, 75  
Dependency notation, 846  
Depth-first search, 787–789, 804  
  applications of, 794–795  
  in directed graphs, 794–795  
Depth of combinatorial circuit, 828  
Derangements, 562–565  
  number of, 562, 566  
Derivable, 899  
  directly, 899  
Derivable from, 849  
Derivation, 849  
Derivation tree, 899  
Derivation trees, 852–854  
Descartes René, 122  
Descendants of vertex, 747, 804  
Describing a set  
  by listing its members, 116  
  by the roster method, 116  
  using set builder notation, 116  
Designing finite-state automata, 869  
Detachment  
  laws of, 71  
Deterministic finite-state automata, 872, 899  
Deviation, standard, 487  
Devil's pair, 678  
Diagnostic test results, 471–472  
Diagonalization argument, Cantor, 173, 186  
Diagonal of a polygon, 338  
Diagonal of a square matrix, 181  
Diagonal relation, 598

- Diagrams  
 Hasse, 622–626, 634  
 state, for finite-state machine, 860  
 tree, 394–395, 439  
 Venn, 118, 185
- Diameter of graph, 741
- Dice, 446
- Dictionary ordering, 435
- Die, 468, 488–489  
 dodecahedral, 496  
 octahedral, 496
- Difference, A-6  
 backward, 513  
 common, 157  
 forward, 568
- Difference equation, 513
- Difference of multisets, 138
- Difference of sets, 128, 186
- Diffe, Whitfield, 302
- Diffe–Hellman key agreement protocol, 302
- Digit  
 binary, 11
- Digital circuit(s), 20–21
- Digital signatures, 303, 306
- Digital signatures in RSA system, 303
- Digits  
 Cantor, 438
- Digraphs, 633, 643, 735  
 circuit (cycle) in, 599, 633  
 connectedness in, 685–687  
 converse of, 668  
 depth-first search in, 794–795  
 edges of, 643, 653–654  
 Euler circuit of, 694  
 loops in, 594, 633  
 paths in, 599–600, 633, 736  
 representing relations using, 594–596  
 self–converse, 740  
 vertex of, 641, 654
- Dijkstra's algorithm, 709–714, 737
- Dijkstra, Edsger Wybe, 710
- Dimes, 199
- Diophantus, 106
- Dirac's theorem, 701
- Dirac, G. A., 701
- Directed edges, 644, 653–654, 735
- Directed graphs, 633, 643, 735  
 circuit (cycle) in, 599, 633  
 connectedness in, 685–687  
 converse of, 668  
 depth-first search in, 794–795  
 edges of, 643, 653–654  
 Euler circuit of, 694  
 loops in, 594, 633  
 paths in, 599–600, 633, 736  
 representing relations using, 594–596  
 self–converse, 740  
 vertex of, 641, 654
- Directed multigraph, 643, 735  
 paths in, 679
- Directly derivable, 849, 899
- Direct proof, 82, 110
- Dirichlet, G. Lejeune, 262, 400
- Dirichlet drawer principle, 400
- Discrete logarithm, 284, 306
- Discrete logarithm problem, 284
- Discrete mathematics  
 definition of, vii, viii, xviii  
 problems studied in, xviii  
 reasons to study, xviii
- Discrete probability, 445–500  
 assigning, 453–455  
 conditional, 453, 456–457, 494  
 finite, 445–448  
 Laplace's definition of, 445  
 of collision in hashing functions, 462–463  
 of combinations of events, 449–450, 455–456
- Disjoint sets, 128
- Disjunction(s), 4, 109, 843  
 associative law of, 27  
 distributive law of, over conjunction, 27  
 negating, 28  
 truth table for, 4
- Disjunctive normal form, 35  
 for Boolean variables, 820
- Disjunctive syllogism, 71
- Disquisitiones Arithmeticae* (Gauss), 241
- Distance  
 between distinct vertices, 741  
 between spanning trees, 797
- Distinguishable  
 boxes, 428  
 objects, 428  
 strings, 888
- Distributing objects into boxes, 428–431
- Distribution  
 binomial, 458–460  
 of random variable, 460, 494  
 geometric, 484–485, 494  
 probability, 453  
 uniform, 454, 494
- Distributive lattice, 637, 817
- Distributive laws, 27  
 for Boolean algebra, 815, 818  
 for propositions, 27  
 for sets, 129
- Distributivity  
 for integers modulo  $m$ , 243
- Divide-and-conquer  
 algorithms, 527, 565  
 recurrence relations, 527–535
- Dividend, 238, 239
- Divides, 238, 306
- Divine Benevolence (Bayes), 472
- Divisibility facts, proving, by mathematical induction, 321
- Divisibility relation, 619
- Division  
 of integers, 238–240  
 trial, 258
- Division algorithm, 239, 306
- Division of a cake, 331
- Division rule, 394
- Division rule for counting, 394
- Divisor, 238, 239  
 greatest common, 265–267, 306
- DNA (deoxyribonucleic acid), 388
- DNA sequencing, 698
- Dodecahedral die, 496
- Dodgson, Charles Lutwidge (Lewis Carroll), 50
- Domain  
 of  $n$ -ary relation, 584  
 of a function, 139, 186  
 of a quantifier, 40  
 of partial function, 152  
 of relation, 584  
 restricted  
 of quantifier, 44
- Domain of definition  
 of partial function, 152
- Domain of discourse, 40, 110
- Dominating set, 739
- Domination laws, 27  
 for Boolean algebra, 815  
 for sets, 129
- Domino(s), 103, 314
- Don't care conditions, 836–837
- Double complement, law of, in Boolean algebra, 815, 819
- Double counting proofs, 412
- Double hashing, 292
- Double negation law, 27
- Drug testing, 475
- Dual  
 of a compound proposition, 35  
 of Boolean expression, 816  
 of Boolean function, 816  
 of poset, 630
- Dual graph, 727
- Duality in lattice, 639
- Duality principle, for Boolean identities, 816
- Dudeney, Henry, 504
- Dynamic programming, 507
- Ear(s), 343  
 nonoverlapping, 343
- Earth, 476
- EBNF (extended Backus–Naur form), 858
- Eccentricity of vertex, 757
- Ecology, niche overlap graph in, 648
- Edge  
 adding from a graph, 664  
 removing a vertices from, 664  
 removing from a graph, 663
- Edge chromatic number, 734
- Edge coloring, 734
- Edge connectivity, 684
- Edge contraction, 664
- Edge(s)  
 cut, 683, 684  
 directed, 644, 654, 735  
 endpoints of, 651  
 incident, 651, 735  
 multiple, 642, 643, 735  
 of directed graph, 643, 653–654  
 of directed multigraph, 643  
 of multigraph, 642  
 of pseudograph, 643  
 of simple graph, 642, 667  
 of undirected graph, 644, 654  
 undirected, 644, 735
- Edge vertex, 594
- Effectiveness  
 of an algorithm, 193
- Egyptian (unit) fraction, 380
- Einstein, Albert, 24
- Electronic mail, 472
- Element  
 image of, 186  
 pre-image of, 186
- Elementary subdivision, 724, 736
- Element of a set, 116, 185
- Elements  
 comparable, in partially ordered set, 619, 633  
 equivalent, 608  
 fixed, 494  
 greatest, of partially ordered set, 625, 634  
 incomparable, in partially ordered set, 619, 633  
 least, of partially ordered set, 625, 634  
 maximal, of partially ordered set, 624, 634  
 minimal, of partially ordered set, 624, 625, 634
- Elements of Mathematical Logic (Łukasiewicz), 782
- Ellipses (...), 116
- Empty folder, 118
- Empty set, 118, 185
- Empty string(s), 157, 186, 849

## I-8 Index

- Encryption, 294, 306  
affine transformation, 296  
public key, 306  
RSA, 299
- Encryption key, 306
- Encryption  
Caesar cipher, 294
- Endpoints of edge, 651
- Engine  
Analytic, 31
- Entry of a matrix, 178
- Enumeration, 392, 439  
codeword, 506
- Equal functions, 139
- Equality  
of sets, 117, 185
- Equal matrices, 178
- Equation  
characteristic, 515  
difference, 513
- Equivalence  
proof of, 87
- Equivalence(s)  
logical, 27
- Equivalence classes, 610–611, 633  
and partitions, 612–614  
definition of, 610  
representative of, 610
- Equivalence relations, 607–614, 633  
definition of, 608
- Equivalent  
compound propositions, 8  
logically  
compound propositions, 25
- Equivalent Boolean expressions, 813
- Equivalent elements, 608
- Equivalent finite-state automata, 868, 871–872
- Eratosthenes, 259, 560  
sieve of, 259, 306, 565
- Erdős, Paul, 260, 263, 635, 636, 680
- Erdős number, 635, 680, 689
- Erdős number Project, 680
- Error  
single, 291  
transposition, 291
- Errors,  
in exhaustive proofs, 95  
in proofs by cases, 95  
in proofs by mathematical induction 328–329
- Essential prime implicant, 832, 843
- Euclid, 267
- Euclidean algorithm, 267, 347
- Euler  $\phi$ -function, 272
- Euler's criterion, 286
- Euler's formula, 720–723, 737
- Euler's formula “Eureka,” A-4
- Euler, Leonhard, 693, 695
- Euler circuits, 693–698, 736
- Euler paths, 693–698, 736
- Evaluation functions, 769
- Even, 83
- Event(s), 446  
combinations of, 449–450, 455–456  
complementary, 449  
independent, 452, 457–458, 494  
mutually independent, 497
- Exams, scheduling, 731, 732
- Exchange  
key, 302
- Exclusion rule, 349
- Exclusive or, 5, 6, 110  
truth table for, 6
- Exercises  
difficult, xix  
extremely challenging, xix  
result used in book, xix  
routine, xix
- Exhaustive proof, 93, 110  
common errors with, 95
- Existence proof(s), 96  
constructive, 96, 110  
nonconstructive, 96, 110
- Existential generalization, 76
- Existential instantiation, 76
- Existential quantification, 42, 110
- Existential quantifier, 42
- Expansion(s)  
balanced ternary, 256  
base- $b$ , 246  
binary, 246  
binary coded decimal, 836  
Cantor, 256  
decimal, 246  
hexadecimal, 246  
octal, 246
- Expected values, 477–480, 494  
in hatcheck problem, 481  
linearity of, 477–484, 494  
of inversions in permutation, 482–484
- Experiment, 446
- Exponential complexity, 225
- Exponential functions, A-7  
big- $O$  estimates for, 212
- Exponential generating function, 551
- Exponentiation  
modular, 253  
recursive, 363
- Expression(s)  
binomial, 415  
Boolean, 812–814, 843  
infix form of, 780  
postfix form of, 781  
prefix form of, 780  
regular, 879, 899
- Expressions  
logically equivalent, 110  
extended Backus–Naur form, 858
- Extended binary trees, 352
- Extended binomial coefficients, 539
- Extended binomial theorem, 540
- Extended Euclidean algorithm, 270, 273
- Extended transition function, 867
- Extension  
of transition function, 867
- Exterior of simple polygon, 338
- Facebook, 645
- Factor, 238
- Factorial complexity, 225
- Factorial function, 151
- Factorials, recursive procedure for, 361
- Factoring, 262
- Factorization  
prime, 259
- Failure, 458
- Fairy, tooth, 112
- Fallacies, 69, 75, 110  
of affirming the conclusion, 75  
of denying the hypothesis, 75
- False  
negative, 471, 472  
positive, 471, 472
- Family trees, 745
- Farmer, 692
- Fast multiplication  
of integers, 528–529  
of matrices, 529
- Female pessimal, 234
- Fermat's last theorem, 106, 349
- Fermat's little theorem, 281, 306  
proof of, 285
- Fermat, Pierre de, 106, 281, 282
- Fibonacci, 348
- Fibonacci numbers, 347  
and Huffman coding, 771  
formula for, 517  
iterative algorithm for, 366  
rabbits and, 502–503  
recursive algorithms for, 365
- Fibonacci sequence, 158
- Fibonacci trees, rooted, 757
- Field axioms, A-1
- Fields, 584
- Fields Medal, 263
- Filter, spam, 472–475
- Final assertion, 372, 378
- Final exams, scheduling, 731, 732
- Final state  
finite-state automaton, 867  
of a Turing machine, 891  
Turing machine, 891
- Final value, A-14
- Finding maximum element in a sequence  
algorithm for, 192
- Finite-state automata, 866–872  
accepting state, 867  
designing, 869  
deterministic, 872, 899  
equivalent, 868  
final state, 867  
initial state, 867  
minimization, 872  
nondeterministic, 873, 899  
set not recognized by, 885
- Finite-state machine, 847, 858, 885, 899  
for addition, 862  
input string, 861  
output string, 861  
transition function extension, 867  
transition function extension in, 867  
with output, 859, 865  
with outputs, 859, 863
- Finite-state transducer, 859
- Finite graph, 641
- Finiteness  
of an algorithm, 193
- Finite probability, 445–448
- Finite set, 121, 185
- Finite sets, 553, 565  
number of subsets, 323  
subsets of  
counting, 388  
number of, 323  
union of three, number of elements in, 554–556, 566  
union of two, number of elements in, 553, 565
- First difference, 513
- First forward difference, 568
- Fixed elements, 494
- Fixture controlled by three switches, circuit for, 825
- Flavius Josephus, 512
- Fleury's algorithm, 697, 706
- Flip-flops, 846
- Floor function, 149, 186  
properties of, 150
- Floyd's algorithm, 717

- Folder  
   empty, 118  
 Forbidden pairs, 234  
 Forests, 746  
   definition of, 803  
   minimum spanning, 802  
   spanning, 796  
 Form  
   argument, 70  
   Backus–Naur, 854, 899  
   conjunctive normal, 820  
   disjunctive normal  
     for Boolean variables, 820  
   extended Backus–Naur form, 858  
   infix, 780  
   postfix, 781  
   prefix, 780  
 Form, argument, 110  
 Formal language, 848  
 Formal power series, 538  
 Formula(s)  
   Euler's, 720–723, 737  
   for compound propositions, 354  
   for Fibonacci numbers, 517  
   of operators and operands, 351  
   Stirling's, 151  
   summation, 315  
   well-formed, 350  
 FORTRAN, 854  
 Fortune cookie, 441  
 Forward differences, 568  
 Forward reasoning, 100  
 Forward substitution  
   iteration using, 160  
 Four color theorem, 728–731, 737  
 Fraction, unit (Egyptian), 380  
 Frame's conjecture, 513  
 Free variable, 44, 110  
 Frend, Sophia, 29  
 Frequency assignments, 732  
 Friendship graph, 645  
 Frisbee, rocket-powered, 812  
 Full *m*-ary tree, 748, 752, 804  
   complete, 756  
 Full adder, 826, 843  
 Full binary trees, 352–353  
   height of, 355  
   number of vertices of, 355  
 Full subtractor, 828  
 Function(s), 139, 186  
   Addition of, 141  
   Ackermann's, 359  
   asymptotic, 218  
   as relations, 574  
   big-*O* estimates of, 209  
   bijective, 144  
   Boolean, 811–819, 843  
     dual of, 816  
     functionally complete set of operators for, 821  
     implicant of, 832  
     minimization of, 828–840, 843  
     representing, 813–821  
     self-dual, 844  
     threshold, 845  
   busy beaver, 896, 899  
   ceiling, 149, 186  
   codomain of, 139  
   codomain of a, 186  
   composition of, 146, 186  
   computable, 175, 177, 186, 896, 900  
   counting, 387  
   decreasing, 143  
   domain of, 186  
   equal, 139  
   Euler  $\phi$ , 272  
   evaluation, 769  
   exponential, A-7  
   factorial, 151  
   floor, 149, 186  
   generating, 537–548, 565  
     exponential, 551  
     for counting, 541–546  
     for proving identities, 548  
     for recurrence relations, 546–548  
     probability, 552  
   graph of, 148  
   greatest integer, 149  
   growth of, 209  
   growth of combinations of, 212–214  
   hashing, 287  
     collision in, probability of, 462–463  
   identity, 144  
   increasing, 143  
   injective, 141  
     counting, 387  
   integer-valued, 140  
   inverse, 145  
   inverse of, 186  
   invertible, 145  
   iterated, 360  
   iterated logarithm, 360  
   logarithmic, A-7–A-9  
   McCarthy, 380  
   multiplication of functions, 141  
   number-theoretic, 892  
   **mod**, 239, 306  
   one-to-one, 141, 186  
   onto, 186  
     number of, 560–562, 566  
   partial, 152, 186, 889  
   probing, 288  
   propositional, 110  
   range of, 186  
   real-valued, 140  
   recursive, 345–349, 378  
   strictly decreasing, 143  
   strictly increasing, 143  
   surjective, 143  
   threshold, 845  
   total, 152  
   transition, 859  
   Turing machines computing, 892–893  
   uncomputable, 175, 186, 896, 900  
 Functional completeness, 821  
 Functional decomposition, 846  
 Functionally complete logical operators, 35  
 Functionally complete set of operators  
   for Boolean functions, 821, 843  
 Fundamental theorem of algebra, 241  
 Fundamental theorem of arithmetic, 258, 306  
   proof of, 271  
 Fuzzy logic, 16  
 Fuzzy set(s), 138  
   complement of, 138  
   degree of membership, 138  
   intersection of, 138  
   union of, 138  
 Gödel, Escher, Bach (Hofstadter), 382  
 Gale–Shapley algorithm, 204  
 Gambling, 445  
 Game  
   obligato, 112  
 Game trees, 764–769, 804  
 Gates, logic, 21, 110, 822–827  
   AND, 21, 823, 843  
   combination of, 823  
   NAND, 828  
   NOR, 828  
   OR, 21, 823, 843  
   threshold, 845  
 Gating networks, 822–823  
   depth of, 828  
   examples of, 823–840  
   minimization of, 828–840  
 Gauss, Karl Friedrich, 241  
 Gecko, Gordon, 198  
 Gene, 389  
 Generality  
   of an algorithm, 193  
 Generalization  
   existential, 76  
   universal, 76  
 Generalized combinations, 423–427  
 Generalized induction, 356–357  
 Generalized permutations, 423  
 Generalized pigeonhole principle, 401–403, 439  
 Generating functions, 537–548, 565  
   exponential, 551  
   for counting, 541–546  
   for proving identities, 548  
   for recurrence relations, 546–548  
   probability, 552  
 Generator, power, 292  
 Gene sequencing, 389  
 Genome, 389  
 Geometric distribution, 484–485, 494  
 Geometric mean, 100, 332  
 Geometric progression, 157, 186  
   sum of terms of, 164  
 Geometric progression(s)  
   sums of, 318–319  
 Geometric series, 164  
 Geometry, computational, 338–340, 532–535  
 Giant strongly connected components (GSCC), 686  
 GIMPS (Great Internet Mersenne Prime Search, 261  
 Givens  
   in Sudoku, 32  
 Goat, 692  
 Goldbach's conjecture, 264  
 Goldbach, Christian, 264  
 Golomb's self-generating sequence, 382  
 Golomb, Solomon, 105  
 Google, 18, 794  
 Gossip problem, by mathematical induction, 332  
 Government Communications Headquarters (GCHQ), U.K., 299  
 Government Communications Headquarters (GCHQ), U.K., 299  
 Graceful trees, 807  
 Graham, Ron, 636  
 Grammar(s), 848  
   Backus–Naur form of, 854  
   context-free, 886  
   context-free (type 2), 851  
   context-sensitive, 851  
   language generated by, 850  
   language of, 850  
   monotonic, 852  
   noncontracting, 852  
   productions, 849  
   phrase-structure, 849–854, 899  
   regular, 884, 899  
   regular (type 3), 851  
   type 0, 851, 899  
   type 1, 851, 899  
   type 2, 851, 899  
   type 3, 851, 883, 899  
 Grand Hotel, Hilbert's, 171

## I-10 Index

- Graph(s), 641–744  
     $k$ -connected, 684  
     $n$ -regular, 667  
    academic collaboration, 645  
    acquaintanceship, 645  
    airline route, 647  
    bandwidth of, 741  
    biconnected, 684  
    bipartite, 656–658, 736  
    book number of, 744  
    call, 646  
        connected components of, 682  
    chromatically  $k$ -critical, 734  
    chromatic number of, 728  
    citation, 646  
    collaboration, 645  
    coloring, 727–732, 736  
    complementary, 667  
    complete, 655, 736  
    complete  $m$ -partite, 738  
    complete bipartite, 658, 736  
    connected components of, 682–685, 736  
    connectedness in, 678–689  
    cut set of, 806  
    definition of, 643  
    diameter of, 741  
    directed, 633, 643, 735  
        circuit (cycle) in, 599, 633  
        connectedness in, 685–687  
        converse of, 668  
        dense, 670  
        depth-first search in, 794–795  
        edges of, 643, 653–654  
        Euler circuit of, 694  
        loops in, 594, 633  
        paths in, 599–600, 633, 736  
        representing relations using, 594–596  
        self-converse, 740  
        simple, 643  
        vertex of, 641, 654  
    directed multigraph, 644, 735  
    dual, 727  
    finite, 641  
    friendship, 645  
    Hollywood, 645  
    homeomorphic, 724–725, 736  
    independence number of, 741  
    infinite, 641  
    influence, 645  
    in roadmap modeling, 647  
    intersection, 650  
    isomorphic, 668, 671–675, 736  
        paths and, 687–688  
    matching in, 659  
    mixed, 644  
    models, 644–649  
    module dependency, 647  
    monotone decreasing property of, 742  
    monotone increasing property of, 742  
    multigraphs, 642, 644, 735  
    niche overlap, 648  
    nonplanar, 724–725  
    nonseparable, 683  
    orientable, 740  
    paths in, 678–681  
    planar, 718–725, 736  
    precedence, 647  
    protein interaction, 648  
    pseudograph, 643, 644, 735  
    radius of, 741  
    regular, 667, 736  
    representing, 668–675  
    self-complementary, 677  
        simple, 642, 654–655, 735  
        coloring of, 727  
        connected planar, 719–723  
        crossing number of, 726  
        dense, 670  
        edges of, 642, 663  
        isomorphic, 671–675, 736  
        orientation of, 740  
        paths in, 785  
        random, 742  
        self-complementary, 677  
        sparse, 670  
        thickness of, 726  
        vertices of, 642, 663  
        with spanning trees, 785–787  
    simple directed, 643  
    sparse, 670, 802  
    strongly directed connected, 685  
    subgraph of, 663, 736  
    terminology of, 651–654  
    undirected, 644, 653, 736  
        connectedness in, 681–685  
        Euler circuit of, 694  
        Euler path of, 698  
        orientation of, 740  
        paths in, 679, 736  
        underlying, 654, 736  
    union of, 664, 736  
    very large scale integration, 744  
    Web, 646–647  
        strongly connected components of, 686  
    weighted, 736  
        shortest path between, 707–714  
        traveling salesman problem with, 714–716  
    wheel, 655, 736
- Graph of a function, 148
- Gray, Frank, 703
- Gray codes, 702–703
- “Greater than or equal” relation, 618–619
- Greatest common divisor, 265–267, 306  
    as linear combination, 269
- Greatest element of poset, 625, 634
- Greatest integer function, 149
- Greatest lower bound, 634  
    of poset, 625
- Great Internet Mersenne Prime Search (GIMPS), 261
- Greedy algorithm(s), 198, 232, 235, 325, 764, 798, 804  
    definition of, 198  
    for making change, 199  
    for minimizing maximum lateness of a job, 235  
    for scheduling talks, 200
- Green, Ben, 263
- Green–Tao theorem, 263
- Group commutative, 244
- Growth of functions, 209
- GSCC (giant strongly connected components), 686
- Guarding set, 735
- Guare, John, 680
- Guidelines for mathematical induction, 328
- Guthrie, Francis, 729
- Hadamard, Jacques, 262
- Haken, Wolfgang, 728
- Half adder, 826, 843
- Half subtractor, 828
- Hall’s marriage theorem, 659
- Hall, Philip, 659, 660
- Halting problem, 201, 895, 900
- Hamilton’s “Voyage Around the World” Puzzle, 700
- Hamilton, Sir William Rowan, 698, 700
- Hamilton, William Rowan, 729
- Hamilton circuits, 698–703, 736
- Hamilton paths, 698–703, 736
- Handle, 806
- Handshaking theorem, 653
- Hanoi, tower of, 503–504
- Hardware systems, 17
- Hardy, Godfrey Harold, 97
- Hardy, Godfrey Harold, 97
- Hardy–Weinberg law, 97
- Harmonic mean, 108
- Harmonic number(s)  
    inequality of, 320–321
- Harmonic series, 321
- Hashing  
    collision in, 288  
    double, 292  
    function, 287  
        key, 287
- Hashing functions  
    collision in, probability of, 462–463
- Hasse’s algorithm, 107
- Hasse, Helmut, 623
- Hasse diagrams, 622–626, 634
- Hatcheck problem, 481, 562
- Hazard-free switching circuits, 846
- Heaps, 809
- Heawood, Percy, 728
- Height, star, 901
- Height-balanced trees, 808
- Height of full binary tree, 355
- Height of rooted tree, 753, 804
- Hellman, Martin, 302
- Hexadecimal expansions, 246
- Hexadecimal representation, 306
- Hexagon identity, 421
- Hilbert’s 23 problems, 171, 895
- Hilbert’s Grand Hotel, 171
- Hilbert’s Tenth Problem, 895
- Hilbert, David, 171, 176, 895
- HIV, 476
- Ho, Chung-Wu, 340
- Hoare, C. Anthony R., 372, 374
- Hoare triple, 372
- Hofstader, Douglas, 382
- Hollywood graph, 645  
    paths in, 680–681
- Homeomorphic graphs, 724–725, 736
- Hopper, Grace Brewster Murray, 872
- Hops, 662
- Horner’s method, 230
- Horse races, 31
- Host number (hostid), 392
- HTML, 858
- Huffman, David A., 763
- Huffman coding, 763–764, 804  
    variations of, 764
- Human genome, 389
- Husbands, jealous, 693
- Hybrid topology for local area network, 661
- Hydrocarbons, trees modeling, 750
- Hypercube, 662
- Hypothesis  
    continuum, 175, 186  
    inductive, 313
- Hypothesis of a conditional statement, 6
- Hypothetical syllogism, 71
- “Icosian Game,” 700
- Icosian Puzzle, 698
- Idempotent laws  
    for Boolean algebra, 815, 819  
    for lattices, 637  
    for propositions, 27  
    for sets, 129

- Identification number  
single error in, 291
- Identification number(s)  
for airline tickets, 293
- Identification numbers  
for money orders, 293
- Identifier, 611  
ALGOL, 855  
C language, 857
- Identifying a sequence from its initial terms, 161
- Identities  
Boolean algebra, 27  
set, 129  
set, 129  
set), 132
- Identities, combinatorial, 417–421
- Identities for Boolean algebra, 129
- Identity  
Bézout's, 270  
combinatorial, proof of, 412, 439  
hexagon, 421  
proving, generating functions for, 548  
Vandermonde's, 420–421
- Identity element(s)  
for integers modulo  $m$ , 243
- Identity elements axiom, A-1
- identity function, 144
- Identity laws, 27  
additive, A-1  
for Boolean algebra, 815, 816, 818  
for sets, 129  
multiplicative, A-1
- Identity matrix, 180, 186
- If–then construction, 8
- If then statement, 6
- If and only if, 9
- Iff, 9
- Image of an element, 139, 186
- Image of a set, 141
- Implicant, 832  
essential, 832  
prime, 832
- Implication, 6, 110
- Implicit use of biconditionals, 10
- In-degree of vertex, 654, 736
- Incidence matrices, 671, 736
- Incident edge, 651
- Inclusion–exclusion principle, 128, 392–394, 553–557, 566  
alternative form of, 558–559  
applications with, 558–564
- Inclusion relation, 619
- Inclusive or, 5
- Incomparable elements in poset, 619, 633
- Incomplete induction, 334
- Incorrect proof by mathematical induction, 757
- Increasing function, 143
- Increment  
for linear congruential method, 288
- Independence number, 741
- Independent events, 452, 453, 457–458, 494
- Independent random variables, 485–487, 494
- Independent set of vertices, 741
- Index of summation, 163
- Index registers, 732
- Indicator random variable, 492
- Indirect proofs, 83
- Indistinguishable  
boxes, 428–431  
objects, 428–431  
objects, permutations with, 428  
strings, 888
- Induction  
complete, 334  
generalized, 356–357  
incomplete, 334  
mathematical, 29, 311–329  
principle of, 377  
proofs by, 315–329  
second principle of, 334  
strong, 333–335, 378  
structural, 353–356, 378  
validity of, 326  
well-ordered, 620, 634
- Inductive definitions, 345–357  
of functions, 345–349, 378  
of sets, 349–356, 378  
of strings, 350  
of structures, 349–356
- Inductive hypothesis, 313
- Inductive loading, 333, 343, 380
- Inductive reasoning, 312
- Inductive step, 313, 377, 752, 768
- Inequality  
Bernoulli's, 330  
Bonferroni's, 467  
Boole's, 467  
Chebyshev's, 491, 495  
Markov's, 493  
of harmonic numbers, 320–321  
proving by mathematical induction, 319–320  
triangle, 108
- Inference  
for program correctness, 373  
rule of, 110  
rules of, 69–71
- Infinite graph, 641
- Infinite ladder, 311  
by strong induction, 334
- Infinite series, 167
- Infinite set, 121
- Infinitude of primes, 260
- Infix form, 780
- Infix notation, 779–782, 804
- Influence graphs, 645
- Information flow, lattice model of, 627
- Initial assertion, 372, 378
- Initial conditions, 158, 565
- Initial position  
of a Turing machine, 889
- Initial state, 859  
finite-state automaton, 867  
of a Turing machine, 889
- Initial terms  
identifying a sequence using, 161
- Initial value, A-14
- Initial vertex, 594, 654
- Injection, 141, 186
- Injective (one-to-one) function  
counting, 387
- Injective function, 141
- Inorder traversal, 773, 775, 778, 804
- Input  
to an algorithm, 193
- Input alphabet, 859
- Input string  
finite-state machine, 861
- Insertion sort, 196, 197, 232  
average-case complexity of, 483–484  
worst-case complexity, 222
- Instantiation  
existential, 76  
universal, 75
- Integer  
composite, 257, 306  
perfect, 272  
prime, 257  
signed, 855
- Integer(s)  
linear combination of, 306
- Integer-valued function, 140
- Integers  
axioms for, A-5  
division of, 238–240  
multiplication of  
fast, 528–529  
pairwise relatively prime, 306  
partition of, 359  
relatively prime, 306  
set of, 116  
squarefree, 564
- Integer sequences, 162
- Integers modulo  $m$ , 243  
additive inverses, 243  
associativity, 243  
closure, 243  
commutativity, 243  
distributivity, 243  
identity elements, 243
- Intension, of database, 585
- Interconnection networks for parallel computation, 661–663
- Interest, compound, 160
- Interior of simple polygon, 338
- Interior vertices, 603
- Internal vertices, 748, 804
- International Mathematical Olympiad (IMO), 263, 299
- International Standard Book Number (ISBN), 291
- International Standard Serial Number (ISSN), 293
- Internet, search engines on, 794
- Internet addresses, counting, 392
- Internet datagram, 399
- Internet Movie Database, 645
- Internet Protocol (IP) multicasting, 786
- Intersection  
of fuzzy sets, 138
- Intersection graph, 650
- Intersection of a collection of sets, 133
- Intersection of multisets, 138
- Intersection of sets, 127, 185
- Interval(s), 117  
closed, 117  
open, 117, 332
- Intractable  
problems, 226
- Intractable problem, 232, 897
- Invariant for graph isomorphism, 672, 736
- Invariants, loop, 375–376, 378
- Inverse, 8  
modular, 275, 306
- Inverse, multiplicative, 60
- Inverse function, 145, 186
- Inverse law  
for addition, A-2  
for multiplication, A-2
- Inverse of a square matrix, 184
- Inverse relation, 582
- Inversions, in permutation, expected number of, 482–484
- Inverter, 21, 823, 843
- invertible function, 145
- Invertible matrix, 184
- IP multicasting, 786
- Irrationality of  $\sqrt{2}$ , 342, 381
- Irrational number(s), 85, 116
- Irreflexive relation, 581
- ISBN-10, 291
- ISBN-13, 291  
check digit, 308

## I-12 Index

- ISBN check digit, 291  
Isobutane, 750  
Isolated vertex, 652, 736  
Isomorphic graphs, 668, 671–675, 736  
  paths and, 687–688  
Iterated function, 360  
Iterated logarithm, 360  
Iteration, 160, 360  
  using to solve recurrence relations, 159  
Iteration using backward substitution, 160  
Iteration using forward substitution, 160  
Iterative algorithm, for Fibonacci numbers, 366  
Iterative procedure, for factorials, 366  
Iwaniec, Henryk, 264
- Jacobean rebellion, 151  
Jacquard loom, 31  
Jarník, Vojtěch, 798  
Java, 855  
Jealous husband problem, 693  
Jersey crags, 163  
Jewish–Roman wars, 512  
Jigsaw puzzle, 342  
Job(s)  
  assignment of, 565, 658  
  lateness of, 235  
  slackness of, 235  
Join, in lattice, 637  
Join of *n*-ary relations, 588  
Join of zero-one matrices, 181  
Joint authorship, 645  
Jordan curve theorem, 338  
Josephus, Flavius, 512  
Josephus problem, 512  
Jug, 109, 693
- k*-connected graph, 684  
*k*-tuple graph coloring, 734  
K-maps, 830–836, 843  
Königsberg, 171  
Königsberg bridge problem, 693–694, 696, 697  
Kakutani's problem, 107  
Kaliningrad, Russia, 693  
Karnaugh, 830–836  
Karnaugh, Maurice, 830  
Karnaugh maps, 830–836, 843  
Kayal, Neeraj, 262  
Kempe, Alfred Bray, 728  
Key, 295  
  composite, 585  
  encryption, 306  
  for Caesar cipher, 295  
  hashing function, 287  
  primary, 585–586  
  public, 299  
Key exchange, 302  
Key exchange protocol, 305, 306  
Keystream  
  autokey cipher, 309  
King Hermeas, 2  
Kissing problem, 163  
Kleene's theorem, 880, 900  
Kleene, Stephen Cole, 867, 878  
Kleene closure, 866, 899  
Knapsack problem, 235, 568  
Kneiphof Island, 693  
Knight's tour, 707  
  reentrant, 707  
Knights, knaves, and normals puzzles, 112  
Knights, knaves, and spies puzzles, 112  
Knights and knaves puzzles, 19  
Knuth, Donald, 196, 206  
Knuth, Donald E., 208
- Kruskal's algorithm, 799–801, 804  
Kruskal, Joseph Bernard, 799, 800  
Kuratowski's theorem, 724–725, 737  
Kuratowski, Kazimierz, 724
- Löb's paradox, 112  
Labeled tree, 756  
Ladder, infinite, 311–334  
Lady Byron, Annabella Millbanke, 31  
Lagarias, Jeffrey, 107  
Lamé's theorem, 347  
Lamé, Gabriel, 349  
Landau, Edmund, 206, 207  
Landau symbol, 206  
Language, 847–899  
  context-free, 851  
  context-sensitive, 886  
  formal, 848  
  generated by grammar, 850  
  natural, 847–848  
  of a grammar, 850  
  recognized  
    by nondeterministic finite-state automaton, 873  
  recognized by an automaton, 899  
  recognized by finite-state automata, 868  
  regular, 851  
Language recognizer, 863  
Laplace's definition of probability, 446  
Laplace, Pierre Simon, 445, 447, 472  
Large integers  
  computer arithmetic with, 279  
Large numbers, law of, 459  
Lateness of a job, 235  
Lattice model of information flow, 627  
Lattice point, 380  
Lattices, 626–627, 634  
  absorption laws for, 637  
  associative laws for, 637  
  bounded, 637  
  commutative laws for, 637  
  complemented, 637  
  distributive, 637  
  duality in, 639  
  idempotent laws for, 637  
  join in, 637  
  meet in, 637  
  modular, 639
- Law  
  complement, 129  
  complementation, 129  
  Hardy–Weinberg, 97
- Law(s)  
  absorption  
    for Boolean algebra, 815, 816  
    for lattices, 637  
    for propositions, 27  
  associative  
    for addition, A-1  
    for Boolean algebra, 815, 818  
    for lattices, 637  
    for multiplication, A-1  
    for propositions, 27  
  closure  
    for addition, A-1  
    for multiplication, A-1  
  commutative  
    for addition, A-1  
    for Boolean algebra, 815, 818  
    for lattices, 637  
    for multiplication, A-1  
    for propositions, 27  
  compatibility  
    additive, A-2  
    multiplicative, A-2  
  completeness, A-2  
  De Morgan's  
    for Boolean algebra, 815, 819  
    for propositions, 28  
    proving by mathematical induction, 323–324  
  distributive, A-2  
    for Boolean algebra, 815, 818  
    for propositions, 27  
  domination  
    for Boolean algebra, 815  
  idempotent  
    for Boolean algebra, 815, 819  
    for lattices, 637  
    for propositions, 27  
  identity  
    additive, A-1  
    for Boolean algebra, 815, 816, 818  
    multiplicative, A-1  
  inverse  
    for addition, A-2  
    for multiplication, A-2  
  of detachment, 71  
  of double complement, in Boolean algebra, 815, 818  
  of large numbers, 459  
  of mathematical induction, A-5  
  transitivity, A-2  
  trichotomy, A-2  
  used to prove basic facts, A-3–A-5
- Law of total expectation, 492  
Laws  
  De Morgan's, 26  
Laws of propositional logic, 27  
  absorption, 27  
  associative, 27  
  commutative, 27  
  De Morgan's, 27  
  distributive, 27  
  domination, 27  
  double negation, 27  
  identity, 27  
Laws of Thought, The (Boole), 472, 811  
Leaf, 746, 804  
Least common multiple, 266, 306  
Least element of poset, 625, 634  
Least upper bound, 634, A-2  
  of poset, 625  
Left child of vertex, 749  
Left subtree, 749  
Lemma, 81, 110  
  pumping, 888  
Length of bit string, 505  
Length of path  
  in directed graph, 599  
  in weighted graph, 708  
Length of string  
  recursive definition of, 350  
“Less than or equals” relation, 619  
Letters of English  
  relative frequency, 296  
Level of vertex, 753, 804  
Level order of vertex, 806  
Levin, Leonid, 227  
Lewis Carroll (C. L. Dodgson), 50  
Lexicographic ordering, 356, 435, 620–622  
Liber Abaci (Fibonacci), 348  
Library sort, 196  
Light fixture controlled by three switches, circuit for, 825  
Limit, definition of, 61

- Linear array, 662  
 Linear bounded automata, 886  
 Linear combination of integers, 269, 306  
 Linear complexity, 225  
 Linear congruence, 275, 306  
     systems of, 277–279  
 Linear congruence(s)  
     solving, 277  
 Linear congruential method, 288  
     increment, 288  
     modulus, 288  
     multiplier, 288  
     seed, 288  
 Linear homogenous recurrence relations, 514–520, 565  
 Linearithmic complexity, 225  
 Linearity of expectations, 477–484, 494  
 Linearly ordered set, 619, 633  
 Linear nonhomogenous recurrence relations, 520–524, 565  
 Linear ordering, 619, 633  
 Linear probing function, 288  
 Linear search  
     average-case time complexity, 221  
 Linear search algorithm, 194, 232  
     average-case complexity of, 482–484  
     recursive, 363  
     time complexity of, 220  
 LISP, 855  
 List of 23 problems  
     David Hilbert, 176  
 Lists, merging two, 368  
 Literal, 820, 843  
 Little- $\omega$  notation, 218  
 Littlewood, John E., 97  
 Load balancing problem, 235  
 Loading, inductive, 333, 343, 380  
 Load of a processor, 235  
 Loan, 169  
 Lobsters, 524  
 Local area networks, 661  
 Logarithm  
     discrete, 284  
 Logarithm, iterated, 360  
 Logarithm discrete, 306  
 Logarithmic complexity, 225  
 Logarithmic function, A-7–A-9  
 logarithms  
     big- $O$  estimates of, 212  
 Logic, 1  
     fuzzy, 16  
     predicate, 37  
     propositional, 3  
 Logical connectives, 4  
 Logical equivalences, 27  
     constructing, 29  
 Logical Expression(s)  
     translating English sentences into, 62–63  
 Logical expressions  
     translating English sentences into, 16–17  
     translating mathematical statements into, 60, 61  
 Logically equivalent compound propositions, 25, 110  
 Logically equivalent expressions, 110  
 Logically equivalent statements, 45  
 Logical operators, 109  
     functionally complete, 35  
     precedence of, 11  
 Logic circuit, 20, 110  
 Logic gates, 21, 110, 822–827  
     AND, 21, 823, 843  
     combination of, 823  
     NAND, 828  
     NOR, 828  
     OR, 21, 823, 843  
     threshold, 845  
 Logic programming, 51  
 Logic puzzles, 19–20  
 Long-distance telephone network, 646  
 Longest common subsequence problem, 568  
 Loom, Jacquard, 31  
 Loop constructions, A-14–A-15  
 Loop invariants, 375–376, 378  
 Loops  
     in directed graphs, 594, 633  
     nested, 58  
     within loops, A-15  
 Lord Byron, 31  
 Lottery, 447  
     Mega Millions, 496  
     Powerball, 496  
 Lovelace, Countess of (Augusta Ada), 29, 31  
 Lower bound  
     of lattice, 637  
     of poset, 625, 634  
 Lower limit of a summation, 163  
 Lucas, Edouard, 503  
 Lucas, François Édouard, 162  
 Lucas numbers, 379, 525  
 Lucas sequence, 162  
 Lucky numbers, 570  
 Łukasiewicz, Jan, 780, 782  
 $m$ -ary tree, 748, 804  
     complete, 756  
     full, 748, 752  
     height of, 754–755  
 $m$ -tuple, 586  
 Ménages, probl'eme des, 571  
 Machine  
     finite-state, 899  
     Mealy, 863, 899  
     Moore, 863, 865  
     Turing, 899  
 machine  
     unit-delay, 861  
 Machine(s)  
     delay, 861–862  
     finite-state, 847, 858–859  
        with no output, 865  
        with outputs, 859–863  
     Turing, 847, 886, 888, 889, 893  
        computing functions with, 892–893  
        definition of, 889  
        nondeterministic, 893  
        sets recognized by, 891–892  
        types of, 893  
 Machine minimization, 872  
     vending, 858–859  
 MAD Magazine, 208  
 Magic tricks, 20  
 Majority voting, circuit for, 825  
 Makespan, 235  
 Making change  
     greedy algorithm, 199  
 Male optimal, 234  
 Mappings, 139  
 Maps  
     coloring of, 727  
 Markov's inequality, 493  
 markup languages, 858  
 Massachusetts Institute of Technology (M.I.T.), 299  
 Master theorem, 532  
 Matching, 659  
     maximum, 659  
     stable, 204  
     string, 231  
 Matchings with forbidden pairs, 234  
 Mathematical induction, 29, 311–329  
     Axiom of, A-5  
     errors in, 328–329  
     generalized, 356–357  
     guidelines for, 328  
     incorrect proof by, 757  
     inductive loading with, 333  
     principle of, 377  
     proofs by, 315–329  
        errors in, 328–329  
        of divisibility facts, 321  
        of inequalities, 319–320  
        of results about algorithms, 324  
        of results about sets, 323–324  
        of summation formulae, 315  
     second principle of, 334  
     strong, 333–335, 378  
     structural, 353–356, 378  
     template for, 328  
     template for proofs, 329  
     validity of, 328  
 Matrix (matrices)  
     addition, 178  
     adjacency, 669–671, 736  
        counting paths between vertices by, 688–689  
     entry of, 178  
     equal, 178  
     identity, 180, 186  
     incidence, 671, 736  
     invertible,  
        multiplication  
         algorithm for, 222  
         complexity of, 223  
        noncommutativity of, 179  
        fast, 529  
        representing relations using, 591–594  
     row of a, 178  
     sparse, 670  
     square, 178  
     symmetric, 181, 186  
     transpose of, 181, 186  
     upper triangular, 231  
     zero-one, 181, 186  
        of transitive closure, 602–603  
        representing relations using, 591–594  
     Warshall's algorithm and, 604–606  
 Matrix-chain multiplication, 223  
 Matrix-chain multiplication problem, 513  
 Matrix product, 179  
 Maurolico, Francesco, 313  
 Maximal element of poset, 624, 634  
 Maximum, of sequence, 528  
 Maximum element  
     algorithm for finding, 193  
     in finite sequence, 192  
 Maximum matching, 659  
 Maximum satisfiability problems, 498  
 Maximum spanning tree, 802  
 Maxterm, 822  
 McCarthy  
     function, 380  
 McCarthy, John, 381  
 McCluskey, Edward J., 837  
 Mealy, G. H., 863  
 Mealy machine, 863, 899  
 Mean, 203  
     arithmetic, 100, 332  
     deviation from, 491  
     geometric, 100, 332  
     harmonic, 108  
     quadratic, 108  
 Median, 203  
 Meet, in lattice, 637  
 Meet of zero-one matrices, 181

## I-14 Index

- Mega Millions, Lottery, 496  
Meigu, Guan, 698  
Member of a set, 185  
Membership table, 186  
Members of a set, 116  
memoization, 510  
Merge sort, 196, 367–370, 378, 528  
    complexity of, 532  
    recursive, 368  
Merging two lists, 368  
Mersenne, Marin, 261  
Mersenne primes, 261, 306  
Mesh network, 662–663  
Mesh of trees, 809  
metacharacters, 858  
Metafont, 208  
Method  
    middle-square, 292  
    roster, 116  
Method(s)  
    Critical Path, 639  
    Horner's, 230  
    probabilistic, 465–466, 494  
    Quine–McCluskey, 830, 837–840  
Method(s) of proof, 82  
    by cases, 92, 95  
    by contradiction, 86  
    by contraposition, 83  
    by exhaustion, 93  
    direct, 82  
    exhaustive, 93  
    proofs of equivalence, 87  
    trivial, 84  
    vacuous, 84  
Method roster, 185  
Middle-square method, 292  
Millennium Prize problems, 227  
Miller's test, 286  
Miller's test for base  $b$ , 286  
Minimal element of poset, 624, 634  
Minimization  
    of Boolean functions, 828–840, 843  
    of combinational circuits, 828–840  
Minimization of a finite-state machine, 872  
Minimizing maximum lateness  
    greedy algorithm for, 235  
Minimum, of sequence, 528  
Minimum dominating set, 739  
Minimum spanning forest, 802  
Minimum spanning trees, 797–802, 804  
Minmax strategy, 767, 804  
Minterm, 820, 843  
Mistakes in proofs, 89–90  
Mixed graph, 644  
**mod** function, 239, 306  
Mode, 203  
Modeling  
    computation, 847–897  
    with graphs, 644–649  
    with recurrence relations, 502–507  
    with trees, 749–752  
Modular arithmetic, 240–306  
Modular exponentiation, 253  
    algorithm for, 253  
    recursive, 363  
Modular inverse, 275, 306  
Modular lattice, 639  
Modular properties, in Boolean algebra, 819  
Module dependency graphs, 647  
Modulus  
    for linear congruential method, 288  
Modus ponens, 71  
    universal, 77  
Modus tollens, 71  
    universal, 77  
Mohammed's scimitars, 697  
Molecules, trees modeling, 750  
Money orders  
    identification numbers for, 293  
Monoalphabetic cipher, 297  
Monotone decreasing property of graph, 742  
Monotone increasing property of graph, 742  
Monotonic grammars, 852  
Monte Carlo algorithms, 463–465  
Montmort, Pierre Raymond de, 563, 571  
Monty Hall Three Door Puzzle, 450, 452, 476, 499  
Monty Python, 472  
Moore, E. F., 863  
Moore machine, 863, 865  
Moth, 872  
Motorized pogo stick, 812  
Mr. Fix-It, 263  
Muddy children puzzle, 19  
Multicasting, 786  
Multigraphs, 642, 644, 735  
    Euler circuit of, 697  
    Euler path of, 697  
    undirected, 644  
Multinomial coefficient, 434  
Multinomial theorem, 434  
Multiple, 238  
    least common, 266, 306  
Multiple edges, 642–644, 735  
Multiple output circuit, 826  
Multiplexer, 828  
Multiplication  
    of function, 141  
    matrix-chain, 223  
    of integers  
        fast, 528–529  
    of matrices  
        fast, 529  
Multiplication algorithm, 251  
Multiplicative Compatibility Law, A-2  
Multiplicative inverse, 60  
Multiplicities of elements in a multiset, 138  
Multiplier  
    for linear congruential method, 288  
Multiset, 138  
    multiplicities of elements, 138  
Multisets  
    difference of, 138  
    intersection of, 138  
Mutually independent events, 497  
Mutually independent trials, 458  
 $n$ -ary relations, 583–589, 633  
    domain of, 584  
    operations on, 586–588  
 $n$ -cubes, 655  
 $n$ -queens problem, 792–793  
 $n$ -regular graph, 667  
 $n$ -tuple  
    ordered, 122  
 $n$ -tuples, 584–585  
Naive set theory, 118  
Namagiri, 98  
NAND, 821, 843  
NAND gate, 828  
Natural language, 847–848  
Natural numbers  
    set of, 116  
Naur, Peter, 854  
NAUTY, 674  
Naval Ordnance Laboratory, 872  
Navy WAVES, 872  
Necessary and sufficient conditions, 9  
Necessary condition, 6  
    expressing a conditional statement using, 6  
Necessary for, 6  
Negating conjunctions, 28  
Negating disjunctions, 28  
Negating Quantified Expressions, 46  
Negation, 109  
    of a proposition, 3  
    of nested quantifiers, 63–64  
    truth table for, 4  
Negation operator, 4  
Negative  
    false, 471  
    true, 471  
Neighbors in graphs, 651  
Neptune, 476  
Nested loops, 58  
Nested quantifiers, 57–64  
    negating, 63–64  
Network(s)  
    computer, 641  
        interconnection networks, 661–663  
        local area networks, 661  
        multicasting over, 786  
        with diagnostic lines, 642  
        with multiple lines, 642  
        with multiple one-way lines, 642  
        with one-way lines, 643  
    gating, 822–823  
        depth of, 828  
        examples of, 823–825  
        minimization of, 828–840  
    social, 644  
    tree-connected, 751–752  
Network number (netid), 392  
Newton–Pepys problem, 500  
Niche overlap graph, 648  
Nickels, 199  
Nim, 766, 768  
Nobel Prize, 119  
Nodes, 594, 641  
Noncommutativity of matrix multiplication, 179  
Nonconformists, 472  
Nonconstructive existence proof, 96, 110  
Noncontracting grammar, 851  
Nondeterministic finite-state automaton, 873, 899  
    equivalent finite-state automaton, 874  
    language recognized by, 873  
Nondeterministic polynomial-time problems, 227  
    class of, 900  
item Nondeterministic Turing machine, 893, 899  
Nonoverlapping ears, 343  
Nonplanar graphs, 724–725  
Nonregular set, 891  
    recognition by Turing machine, 891  
Nonresidue  
    quadratic, 286  
Nonseparable graph, 683  
Nonterminals, 849  
NOR, 821, 843  
NOR gate, 828  
Normal form  
    disjunction, 35  
    prenex, 68  
NOT, 18

- Notation  
 big-*O*, 205, 232  
 big-Omega ( $\Omega$ ), 214, 232  
 big-Theta, 215  
 big-Theta ( $\Theta$ ), 232  
 dependency, 846  
 for products  
     well-formed formula in, 784  
 infix, 779–782, 804  
 little-*o*, 218  
 Polish, 780, 804  
 postfix, 779–782, 804, 858  
 prefix, 779–782, 804  
 product, 186  
 reverse Polish, 781, 804, 858  
 set builder, 116, 185  
 summation, 162, 186  
 NOT gate, 21  
 Noun, 848  
 Noun phrase, 848  
*Nova*, 107  
 NP, class of nondeterministic polynomial-time problems, 900  
 NP-complete problems, 227, 715, 830, 900  
 Null quantification, 56  
 Null set, 185  
 Null string, 849  
 Number(s)  
     Bacon, 680  
     Bell, 618  
     cardinal, 121  
     Carmichael, 283, 306  
     Catalan, 507  
     chromatic, 728, 737  
         edge, 734  
     crossing, 726  
     computable, 886  
     Cunningham, 262  
     Erdős, 635, 680, 689  
     Fibonacci, 347, 365–367, 771  
         formula for, 517  
         iterative algorithm for, 366  
         rabbits and, 502–503  
         recursive algorithms for, 365  
     harmonic  
         inequality of, 320–321  
     independence, 741  
     irrational, 85, 116  
     large, law of, 459  
     Lucas, 379, 525  
     lucky, 570  
     natural, 116  
     pseudorandom, 288  
     Ramsey, 404  
     rational, 85, 116  
     real, A-1–A-5  
     Stirling, of the first kind, 431  
         signless, 443  
     Stirling, of the second kind, 430  
     Ten Most Wanted, 262  
     Ulam, 188  
 Number-theoretic functions, 892  
 Numbering plan, 387  
 Number theory, 237  
     definition of, 237  
 Object, 118  
 Object(s)  
     distinguishable, 428  
         and distinguishable boxes, 428–429  
         and indistinguishable boxes, 428–431  
     indistinguishable, 428  
         and indistinguishable boxes, 428–431  
         unlabeled, 428  
 Obligato game, 112  
 Octahedral die, 496  
 Octal expansions, 246, 306  
 Octal representation, 246, 306  
 Odd, 83  
 Odd pie fights, 325  
 Odlyzko, Andrew, 163  
 Odometer, 307  
 One's complement representations, 256  
 One-to-one (injective) function  
     counting, 387  
 One-to-one correspondence, 144, 186  
 One-to-one function, 141, 143, 186  
*On-Line Encyclopedia of Integer Sequences (OEIS)*, 162  
 Only if, expressing conditional statement using, 7  
 Onto (surjective) function, 143, 186  
     number of, 560–562, 566  
 Open interval, 117, 332  
 Open problems, 106, 263–264  
 Operands, well-formed formula of, 351  
 Operation(s)  
     bit, 11, 110  
     bitwise, 110  
     set, 127  
 Operations  
     on  $n$ -ary relations, 586–588  
 Operator(s)  
     bitwise, 12  
     logical, 109  
     negation, 4  
     logical, 109  
     selection, 586  
     well-formed formula of, 351  
 Opium, 475  
 Optimal algorithm, 230  
 Optimal for suitors, stable assignment, 343  
 Optimal solution, 198  
 Optimization problems, 198  
 OR, 18  
 Or  
     exclusive, 5, 6  
     inclusive, 5  
 Oracle of Bacon, 680  
 Order, 207  
     of quantifiers, 58  
 Ordered  $n$ -tuple, 122  
 Ordered rooted tree, 749, 804, 806  
 Ordering  
     dictionary, 435  
     lexicographic, 356, 435, 620–622  
     linear, 619  
     partial, 618–629, 633  
     quasi-ordering, 637  
     total, 619, 633  
 Ordered pair  
     defined using sets, 126  
 Ordinary generating function, 537  
 Ore's theorem, 701, 707  
 Ore, O., 701  
 Organizational tree, 750  
 OR gate, 21, 823, 843  
 Orientable graphs, 740  
 Orientation of undirected graph, 740  
 Out-degree of vertex, 654, 736  
 Output  
     to an algorithm, 193  
 Output alphabet, 859  
 Outputs  
     finite-state machines with, 859–863  
     finite-state machines without, 865  
 Output string  
     finite-state machine, 861  
 $P(n, r)$ , 439  
 P, class of polynomial-time problems, 900  
 P=NP problem, 897  
 Pair, devil's, 678  
 Pairs, forbidden, 234  
 Pairwise relatively prime integers, 306  
 Palindrome(s), 202, 397, 857  
     set of, 888  
 Paradigm  
     Algorithmic, 224, 232  
 Paradox, 118, 185  
     barber, 16  
     Löb's, 112  
     Russell's, 126  
     St. Petersburg, 497  
 Parallel algorithms, 661  
 Parallel edges, 642  
 Parallel processing, 229, 661  
     tree-connected, 751–752  
 Parentheses, balanced strings of, 382  
 Parent of vertex, 747, 803  
 Parent relation, 580  
 Parity  
     same, 83  
 Parity check bit, 290  
 Parse tree, 852–854, 899  
 Parsing  
     bottom-up, 853  
     top-down, 853  
 Partial correctness, 372  
 Partial function, 152, 186, 889  
     codomain of, 152  
     domain of, 152  
     domain of definition, 152  
     undefined values, 152  
 Partially ordered set, 618  
     antichain in, 637  
     chain in, 637  
     comparable elements in, 622, 633  
     dense, 632  
     dual of, 630  
     greatest element of, 625, 634  
     Hasse diagram of, 622–624, 634  
     incomparable elements in, 622, 633  
     least element of, 625, 634  
     lower bound of, 625, 634  
     maximal element of, 624, 634  
     minimal element of, 624, 625, 634  
     upper bound of, 625, 634  
     well-founded, 632  
 Partial orderings, 618–629, 633  
     compatible total ordering from, 634  
 Partition, 552  
     of positive integer, 359, 431  
     of set, 612–614, 633  
     refinement of, 617  
 Partner  
     valid, 234  
 Pascal's identity, 418–419, 439  
 Pascal's triangle, 419, 439  
 Pascal, Blaise, 282, 419, 445, 452  
 Passwords, 66, 391

## I-16 Index

- Paths, 678–681  
 and graph isomorphism, 687–688  
 counting between vertices, 688–689  
 Euler, 693–698, 736  
 Hamilton, 698–703, 736  
 in acquaintanceship graphs, 680  
 in collaboration graphs, 680  
 in directed graphs, 599–600, 633, 736  
 in directed multigraphs, 679  
 in Hollywood graph, 680–681  
 in simple graphs, 679  
 in undirected graphs, 679  
 length of, in weighted graph, 708  
 of length  $n$ , 680  
 shortest, 707–716, 736  
 terminology of, 679
- Payoff, 765
- Pearl Harbor, 872
- Pecking order, 741
- Peirce, Charles Sanders, 38
- Peirce arrow, 36
- Pelc, A., 536
- Pendant vertex, 652, 736
- Pennies, 199
- Perfect integer, 272
- Perfect power, 93
- Perfect square, 83
- Peripatetics, 2
- Permutation, circular, 415
- Permutations, 407–409, 439  
 generating, 434–436  
 generating random, 499  
 inversions in, expected number of, 482–484  
 with indistinguishable objects, 427–428  
 with repetition, 423
- PERT (Program Evaluation and Review Technique), 639
- Petersen, Julius Peter Christian, 706
- Phrase-structure grammar, 849, 899
- Phrase-structure grammars, 849–854
- Pick's theorem, 342
- Pie fights, odd, 325
- Pigeonhole principle, 86, 399–405, 439  
 applications with, 403–405  
 generalized, 401–403, 439
- Planar graphs, 718–725, 736
- Plato, 2
- Plato's Academy, 2, 259
- PNF (prenex normal form, 68
- Pogo stick, motorized, 812
- Pointer, position of, digital representation of, 702–703
- Poisonous snakes, 763
- Poker hands, 411
- Polish notation, 780, 804
- Polygon, 338  
 convex, 338  
 diagonal of, 338  
 exterior of, 338  
 interior of, 338  
 nonconvex, 338  
 sides of, 338  
 simple, 338  
 vertices of, 338  
 with nonoverlapping ears, 343
- Polynomial-time algorithm for primality, 262
- Polynomial-time problems  
 class of, 900
- Polynomial complexity, 225
- Polynomials  
 big- $O$  estimates for, 212  
 big- $O$  estimates of, 209  
 big-Theta estimates for, 216
- Polynomials, rook, 571
- Polyominoes, 105, 326
- Ponens  
 modus, 71
- Population of the world, 168
- Poset, 618, 633  
 antichain in, 637  
 chain in, 637  
 comparable elements in, 619, 633  
 dense, 632  
 dual of, 630  
 greatest element of, 625, 634  
 Hasse diagram of, 622–624, 634  
 incomparable elements in, 619, 633  
 least element of, 625, 634  
 lower bound of, 625, 634  
 maximal element of, 624, 634  
 minimal element of, 624, 625, 634  
 upper bound of, 625, 634  
 well-founded, 632
- Positive  
 false, 471  
 true, 471
- Positive integers  
 axioms for, A-5
- Postcondition, 372
- Postconditions, 39
- Postfix form, 781
- Postfix notation, 779–782, 804, 858
- Postorder traversal, 773, 776–778, 804
- Postulates, 81
- Potrzebie System of Weights and Measures, 208
- Powerball, Lottery, 496
- Power generator, 292
- Powers  
 of relation, 580–581, 633
- Power series, 538–541  
 formal, 538
- Power set, 121, 185
- Pre-image of an element, 139, 186
- Precedence  
 of logical operators, 11  
 of quantifiers, 44
- Precedence graphs, 647
- Precondition(s), 39, 372
- Predicate, 110  
 truth set of, 125
- Predicate calculus, 40
- Predicate logic, 37
- Prefix codes, 762–764, 804
- Prefix form, 780
- Prefix notation, 779–782, 804  
 well-formed formula in, 784
- Premise of a condition statement, 6, 110
- Premises of an argument, 69, 70
- Prenex normal form (PNF), 68
- Preorder traversal, 773–775, 778, 804
- Prim's algorithm, 798–799, 804
- Prim, Robert Clay, 798, 800
- Primality testing, 464–465
- Primary key, 585–586, 633
- Prime, 257, 306  
 Mersenne, 306  
 primitive root of a, 306  
 probability positive integer less than  $n$  is, 262
- Prime(s)  
 arithmetic progression of, 263  
 conjectures about, 263  
 distribution of, 261  
 in arithmetic progressions, 262  
 infinitude of, 260  
 Mersenne, 261, 306  
 of form  $n^2 + 1$ , 264
- primitive root of a, 306  
 probability positive integer less than  $n$  is, 262  
 twin, 264
- Prime factorization, 259
- Prime implicant, 832, 843  
 essential, 832
- Prime number theorem, 262, 636
- Primitive root of a prime, 284, 306
- Prince of Mathematics, 241
- Princess of Parallelograms, 31
- Principia Mathematica (Whitehead and Russell), 34
- Principia Mathematica, 119
- Principle(s)  
 duality, for Boolean identities, 816  
 of buoyancy, A-4  
 of counting, 385–390  
 of inclusion–exclusion, 392–394, 553–557, 566  
 alternative form of, 559  
 applications with, 558–564  
 of mathematical induction, 377  
 of well-founded induction, 635  
 of well-ordered induction, 620  
 pigeonhole, 86, 399–405, 439  
 applications with, 403–405  
 generalized, 401–403, 439
- Principle of inclusion–exclusion, 128
- Private key  
 cryptosystem, 298, 306  
 RSA system, 301
- Prize, Nobel, 119
- Probabilistic algorithms, 445, 463–465, 494
- Probabilistic method, 465–466, 494
- Probabilistic primality testing, 464–465
- Probabilistic reasoning, 450
- Probability, discrete, 445–500  
 assigning, 453–455  
 conditional, 453, 456–457, 494  
 finite, 445–448  
 in medical test results, 471–472  
 Laplace's definition of, 446  
 of collision in hashing functions, 462–463  
 of combinations of events, 449–450, 455–456
- Probability distribution, 453
- Probability generating function, 552
- Probability positive integer less than  $n$  is prime, 262
- Probability theory, 445, 452–466
- Probing function, 288
- Problem(s)  
 n-queens, 792–793  
 art gallery, 735  
 birthday, 461–463  
 bridge, 693–694, 696, 697  
 celebrity, 332  
 Chinese postman, 698  
 class of NP-complete, 897  
 class P, 896  
 closest-pair, 532–535  
 Collatz, 107  
 decidable, 895  
 decision, 894, 900  
 discrete logarithm, 284  
 halting, 201, 895, 900  
 hatcheck, 481, 562  
 Hilbert's 23, 895  
 Hilbert's Tenth Problem, 895  
 intractable, 226, 232, 897  
 jealous husband, 693  
 Josephus, 512  
 Kakutani's, 107  
 kissing, 163  
 knapsack, 235, 568  
 load balancing, 235  
 longest common subsequence, 568  
 matrix-chain multiplication, 513  
 maximum satisfiability, 498

- Millennium Prize, 227  
 Newton–Pepys, 500  
 NP, 227, 896  
 NP-complete, 227, 715, 830, 900  
 P, 227  
 P=NP, 897  
 P versus NP, 227  
 open, 106, 263–264  
 optimization, 198  
 problem, two children, 498  
 satisfiability, 34, 227  
**Satisfiability**  
 solving, 34  
 scheduling greedy algorithm for final exams, 731, 732  
 searching, 194  
 shortest-path, 707–716, 736  
 solvable, 226, 232, 895, 900  
 Syracuse, 107  
 tiling, 895  
 tractable, 226, 232, 897  
 traveling salesman, 714–716, 736  
 traveling salesperson, 702, 709  
 two children, 498  
 Ulam’s, 107  
 undecidable, 895  
 unsolvable, 226, 232, 895, 900  
 utilities-and-houses, 718  
 yes-or-no, 894  
*Probl’eme de rencontres*, 563, 571  
*Probl’eme des ménages*, 571  
 Procedure statements, A-11  
**Processing**  
 concurrent, 647  
 parallel, 229, 661  
 tree-connected, 751–752  
**Product**  
 Boolean, 811–812, 821, 843  
 matrix, 179  
 Product-of-sums expansion, 820  
 Production, 899  
 Productions, 849  
 of grammar, 849  
 Product notation, 186  
 Product rule, 386  
 Program correctness, 372–378  
 conditional statements for, 373–375  
 loop invariants for, 375–376  
 partial, 372  
 program verification for, 372–373  
 rules of inference for, 373  
 Program Evaluation and Review Technique (PERT), 639  
**Programming**  
 logic, 51  
 Programming, dynamic, 507  
 Programming languages, 398, 611  
 Program verification, 372–373  
**Progression**  
 arithmetic, 186  
 geometric, 157, 186  
 sums of, 318–319  
 Projection of  $n$ -ary relations, 586, 633  
 Prolog, 51, 74  
 Proof(s), 81, 110  
 adapting, 101  
 by cases, 92–110  
     common errors with, 95  
 by contradiction, 86, 110  
 by contraposition, 83, 110  
 by exhaustion, 93  
 by mathematical induction, 315–329  
 by structural induction, 353  
 combinatorial, 412, 439  
 constructive existence, 96  
 direct, 82, 1180  
 existence, 96  
 indirect, 83  
 mistakes in, 89  
 nonconstructive existence, 96  
 of equivalence, 87  
 of recursive algorithms, 364–367  
 trivial, 84, 110  
 uniqueness, 99, 110  
 vacuous, 84, 110  
 Proof strategy, 85, 100–106  
**Proper subset**, 185  
**Properties**  
 of algorithms, 193  
**Property**  
 Archimedean, A-5  
 Completeness, A-2  
 Well-ordering, 340–341, 378, A-5  
**Proposition(s)**, 2, 81, 109  
 compound, 3, 25, 27, 109  
     negation of, 3, 109  
 Propositional equivalences, 129  
 Propositional calculus, 3  
 Propositional function, 110  
 Propositional logic, 3  
     applications of, 16  
     argument, 70  
     rules of inference for, 71  
 Propositional variable(s), 2, 109  
 Protein interaction graph, 648  
 Protestant Nonconformists, 472  
**Protocol(s)**  
 cryptographic, 302  
 Diffie–Helman key agreement, 302  
 key exchange, 305, 306  
 Pseudocode, 192, A-11–A-15  
 Pseudograph, 643, 644, 735  
 Pseudoprime, 282, 306  
 strong, 286  
 to the base  $b$ , 282  
 Pseudorandom numbers, 288  
     generated by linear congruential method, 288  
     middle-square generator, 292  
     power generator for, 292  
     pure multiplicative generator of, 289  
**Public key**  
 cryptosystem, 298  
 encryption, 306  
 RSA system, 299  
**Pumping lemma**, 888  
 Pure multiplicative generator, 289  
 Pushdown automaton, 886  
**Puzzle**  
 “Voyage Around the World,”, 700  
 Birthday Problem, 461–463  
 Icosian, 698  
 jigsaw, 342  
 knights, knaves, and normals, 112  
 knights, knaves, and spies, 112  
 knights and knaves, 19  
 the lady or the tiger, 24  
 logic(, 19  
 logic), 20  
 Monty Hall Three Door, 450, 452, 476, 499  
 muddy children, 19  
 Reve’s, 504  
 river crossing, 692  
 Sudoku, 32  
 Sun-Tsu’s, 277  
 Tower of Hanoi, 503–504  
 zebra, 24  
 P versus NP problem, 227  
 Pythagorean triples, 106  
 Panini, 854  
 Quadratic congruence, 285  
 Quadratic mean, 108  
 Quadratic nonresidue, 286  
 Quadratic residue, 286  
 Quad trees, 809  
 Quality control, 463  
 Quantification, 40  
     as loops, 58  
     existential, 42, 110  
     null, 56  
     universal, 110  
 Quantification, universal, 40  
**Quantified Expressions**  
 negating, 46  
**Quantified statement(s)**  
 restricted domain, 124  
 rules of inference for, 75  
 rules of inference for), 77  
**Quantifier**  
 existential, 42  
 scope of, 45, 110  
 uniqueness, 44  
 universal, 40  
**Quantifiers**  
 De Morgan’s law, 47  
 nested  
     negating, 63–64  
     nested(, 57  
     nested), 64  
 order of, 58  
 precedence of, 44  
 use in system specifications, 50  
 Quarters, 199  
 Quasi-ordering, 637  
 Queen of mathematics, 241  
 Queen of the sciences, 241  
 Queens on chessboard, 740  
 Question, begging the, 90, 110  
 Quick sort, 196, 371  
 Quine, Willard van Orman, 837, 839  
 Quine–McCluskey method, 830, 837–840  
 Quotient, 238, 239, A-6  
 Quotient automaton, 878  
 r-combination, 410, 439  
 r-permutation, 407, 439  
 Rabbits, 502–503  
 Races, horse, 31  
 Radius of graph, 741  
 Rado, Tibor, 899  
 Ramanujan, Srinivasa, 97, 98  
 Ramaré, O., 264  
 Ramsey, Frank Plumpton, 404  
 Ramsey number, 404  
 Ramsey theory, 404  
 Random permutation, generating, 499  
 Random simple graph, 742  
 Random variables, 454, 460–461  
     covariance of, 494  
     definition of, 494  
     distribution of, 460, 494  
         geometric, 484–485  
         expected values of, 477–480, 491, 494  
     independent, 485–487, 494  
     indicator, 492  
     standard deviation of, 487  
     variance of, 477–480, 494  
 Range of a function, 139, 186  
 Ratio  
     common, 157

## I-18 Index

- Rational number(s), 85, 116  
  countability of, 172
- Reachable, 807
- Reachable state, 901
- Real-valued function, 140
- Real number  
  decimal expansion of a, 174
- Real numbers  
  constructing, A-5  
  least upper bound, A-2  
  set of, 116  
  upper bound, A-2
- Reasoning  
  circular, 90, 110  
  deductive, 312  
  forward, 100  
  inductive, 312  
  probabilistic, 450
- Recognized language, 868
- Recognized strings, 868
- Recognizer  
  language, 863
- Rencontres*, 563, 571
- Records, 584
- Recurrence relation(s), 158, 186, 501–510  
  associated homogenous, 520  
  definition of, 158, 565  
  divide-and-conquer, 527–535  
  initial condition for, 158  
  linear homogenous, 514–520, 565  
  linear nonhomogenous, 520–524, 565  
  modeling with, 502–507  
  simultaneous, 526  
  solution of, 158  
  solving, 514–524  
    generating functions for, 546–548  
    solving using iteration, 159
- Recursion, 344
- Recursive algorithms, 360–370, 378  
  correctness of, 364  
  for binary search, 363  
  for computing  $a^n$ , 361  
  for computing greatest common divisor, 362  
  for factorials, 361  
  for Fibonacci numbers, 365  
  for linear search, 363  
  for modular exponentiation, 363  
  proving correct, 364–367  
  trace of, 361, 362
- Recursive definitions, 311, 345–357  
  of extended binary trees, 352  
  of factorials, 361  
  of functions, 345–349, 378  
  of a sequence, 158  
  of sets, 349–356, 378  
  of strings, 349  
  of structures, 349–356
- Recursive merge sort, 368
- Recursive modular exponentiation, 363
- Recursive sequential search algorithm, 363
- Recursive step, 349
- Reentrant knight's tour, 707
- Refinement, of partition, 617
- Reflexive closure of relation, 598, 634
- Reflexive relation, 576, 633  
  representing  
    using digraphs, 594–596  
    using matrices, 591
- Regions of planar representation graphs, 719, 736
- Regular expression, 899
- Regular expressions, 879
- Regular grammar(s), 851, 884, 899, 900
- Regular graph, 667
- Regular language, 851
- Regular set(s), 878–879, 884, 899, 900  
  constructing automaton to recognize, 881
- Relation, 124  
  recurrence, 158, 186
- Relation(s), 573–639  
   $n$ -ary, 583–589, 633  
    domain of, 584  
    operations on, 586–588  
    “greater than or equal.”, 618–619  
    “less than or equals.”, 619  
  antisymmetric, 577–578, 633  
  asymmetric, 582  
  binary, 573, 633  
  circular, 635  
  closures of, 597–606, 633  
    reflexive, 598, 633  
    symmetric, 598, 634  
    transitive, 597, 600–603, 634  
  combining, 579–580  
  complementary, 582  
  composite of, 580, 633  
  connectivity, 600, 633  
  counting, 578–579  
  covering, 623, 631  
  diagonal, 598  
  divide-and-conquer recurrence, 527  
  divisibility, 619  
  domains of, 584  
  equivalence, 607–614, 633  
  functions as, 574  
  inclusion, 619  
  inverse, 582, 633  
  irreflexive, 581  
  on set, 575–576  
  parent, 580  
  paths in, 599–600  
  powers of, 580–633  
  properties of, 576–579  
  reflexive, 576, 633  
  representing  
    using digraphs, 594–596  
    using matrices, 591–594  
  symmetric, 577–578, 634  
  transitive, 578–580, 633
- Relational database model, 584–586, 633
- Relation on a set, 124
- Relative frequency  
  letters of English, 296
- Relatively prime integers, 306
- Remainder, 239, 306
- Removing an edge from a graph, 663
- Removing an edge from a vertices, 664
- Repetition  
  combinations with, 424–427  
  permutations with, 423
- Replacement  
  sampling with, 448  
  sampling without, 448
- Representation  
  base  $b$ , 306  
  binary, 306  
  octal, 306  
  unary, 892
- Representation hexadecimal, 306
- Representations  
  one's complement, 256  
  two's complement, 256
- Representative, of equivalence class, 610
- Residue  
  quadratic, 286
- Resolution, 74
- Resolution (rule of inference), 71
- Restricted domain  
  of quantified statement, 124  
  quantifier with, 44
- return statement, A-15
- Reve's puzzle, 504
- Reverse-delete algorithm, 803
- Reverse Polish notation, 781, 804  
  reverse Polish notation, 858
- Right child of vertex, 749
- Right subtree, 749
- Right triominoes, 326, 333
- Ring commutative, 244
- Ring topology for local area network, 661
- River crossing puzzle, 692
- Rivest  
  Ronald, 300  
Rivest, Ronald, 299
- RNA (ribonucleic acid), 388
- RNA chain sequencing, 443
- Roadmaps, 647
- Rock climbing, 163
- Rocket-powered Frisbee, 812
- Rook polynomials, 571
- Root  
  primitive, 284
- Rooted Fibonacci trees, 757
- Rooted spanning tree, 797
- Rooted trees, 351, 747–749  
   $S_k$ -tree, 806  
  B-tree of degree  $k$ , 805  
  balanced, 753, 804  
  binomial, 805  
  decision trees, 760–762  
  definition of, 803  
  height of, 753, 804  
  level order of vertices of, 806  
  ordered, 749, 804, 806
- Roots, characteristic, 515
- Roster method, 116, 185
- Round-robin tournaments, 341, 649
- Routing transit number (RTN), 308  
  check digit, 308
- Row of a matrix, 178
- Roy, Bernard, 603
- Roy–Warshall algorithm, 603–606
- RSA system, 299  
  cryptosystem, 299, 301, 306  
  decryption, 300  
  decryption key, 301  
  digital signatures, 303  
  encryption 299  
  private key, 301  
  public key, 301
- Rule  
  division, 394  
  product, 386  
  subtraction, 393  
  sum, 386, 389, 439
- Rule(s) of inference, 69–71, 110  
  addition, 71  
  building arguments using, 73  
  conjunction, 71  
  disjunctive syllogism, 71  
  for program correctness, 373  
  for propositional logic, 71  
  for quantified statements, 75–77  
  hypothetical syllogism, 71  
  modus ponens, 71  
  modus tollens, 71  
  resolution, 71, 74  
  simplification, 71
- Run, 492
- Russell's paradox, 126
- Russell, Bertrand, 118, 119

- $S_k$ -tree, 806  
 Same parity, 83  
 Samos, Michael, 533  
 Sample space, 446  
 Sampling  
     without replacement, 448  
     with replacement, 448  
 Sandwich, 858  
 Sanskrit, 854  
 Satisfiability, 30  
 Satisfiability problem, 227  
     solving, 34  
     modeling Sudoku puzzle as, 32  
 Satisfiability problem, maximum, 498  
 Satisfiable compound proposition, 30, 110  
 Saturated hydrocarbons, trees modeling, 750  
 Saxena, Nitin, 262  
 Scheduling problems  
     final exams, 731, 732  
     greedy algorithm for, 324–325  
     software projects, 633  
     talks, 200  
     tasks, 629  
 Schröder, Ernst, 175  
 Schröder–Bernstein theorem, 174  
 Scimitars, Mohammed's, 697  
 Scope of a quantifier, 45, 110  
 Screw, Archimedes, A-4  
 Search  
     binary, 194  
     linear, 194  
     sequential, 194  
 Search engines, 794  
 Searches  
     Boolean, 18  
 Searching algorithms, 194–196, 232  
     binary, 528  
     breadth-first, 789–791, 804  
     depth-first, 787–789, 804  
         applications with, 794–795  
         in directed graphs, 794–795  
     linear  
         average-case complexity of, 482–484  
         recursive, 363  
     recursive binary, 363  
     recursive linear, 363  
     recursive sequential, 363  
 Searching problems, 194  
 Search trees, binary, 757–759, 804  
 Searcing  
     web pages, 18  
 Second principle of mathematical induction, 334  
 Secret key  
     exchange, 302  
 Seed  
     for linear congruential method, 288  
 Sees, 735  
 Selection operator, 586, 633  
 Selection sort, 196, 203  
 Self-complementary graph, 677  
 Self-converse directed graph, 740  
 Self-dual, 844  
 Self-generating sequences, 382  
 Semantics, 847  
 Sentence, 848, 849  
 Separating set, 684  
 Sequence(s), 156, 186  
     de Bruijn, 744  
     Fibonacci, 158  
     finding maximum and minimum of, 528  
     generating functions for, 537  
     integer, 162  
     Lucas, 162  
     recursive definition of, 158  
     self-generating, 382  
     strictly decreasing, 403  
     strictly increasing, 403  
     unimodal, 568  
 Sequencing  
     RNA chains, 443  
 Sequential search algorithm, 194  
 Serial algorithms, 661  
 Series  
     geometric, 164  
     harmonic, 321  
     infinite, 167  
     power, 538–541  
         formal, 538  
 Set(s), 185  
     Cartesian product of, 123  
     complement of, 128, 129, 186  
     computer representation of, 134  
     countable, 171, 186  
     cut, 806  
     difference of, 128, 186  
     disjoint, 128  
     dominating, 739  
     element of, 185  
     elements of a, 116  
     empty, 118, 185  
     finite, 121, 185, 553, 565  
         combinations of, 437–438  
         counting of subsets, 388  
         number of subsets of, 323  
         union of three, number of elements in, 554–556,  
             566  
         union of two, number of elements in, 553, 565  
     fuzzy, 138  
     guarding, 735  
     image of, 141  
     infinite, 121  
     linearly ordered, 619, 633  
     member of, 185  
     members of a, 116  
     nonregular, 891  
     not recognized by finite-set automata, 885  
     null, 185  
     of complex numbers, 116  
     of irrational numbers, 116  
     of natural numbers, 116  
     of palindromes, 888  
     of rational numbers, 116  
     of real numbers, 116  
     partially ordered, 618, 633  
         antichain in, 637  
         chain in, 637  
         comparable elements in, 619, 633  
         dense, 632  
         dual of, 630  
         greatest element of, 625, 634  
         Hasse diagram of, 622–624, 634  
         incomparable elements in, 619, 622, 633  
         least element of, 625, 634  
         lower bound of, 625, 634  
         maximal element of, 624, 634  
         minimal element of, 624, 625, 634  
         upper bound of, 625, 634  
         well-founded, 632  
     partition of, 612–613, 633  
     power, 121, 185  
     proofs of facts about, by mathematical induction,  
         321  
     recognized by Turing machines, 891–892  
     recursively defined, 349–356, 378  
     regular, 878, 879, 884, 899  
     relation on, 575–576  
     relation on, 124  
     representing with a bit string, 134  
     separating, 684  
     singleton, 118  
     successor of, 137  
     symmetric difference of, 186  
     totally ordered, 619, 633  
     uncountable, 186  
     union of, 127, 185  
     universal, 118, 185  
     well-ordered, 381, 620, 633  
 Set builder notation, 116, 185  
 Set description  
     by listing its members, 116  
     by the roster method, 116  
     using set builder notation, 116  
 Set equality, 117, 185  
 Set identities, 129–132  
     absorption laws, 129  
     associative laws for, 129  
     Cartesian product of, 123  
     commutative laws for, 129  
     De Morgan's laws for, 129  
     difference of, 128, 186  
     distributive laws for, 129  
     domination laws for, 129  
     idempotent laws for, 129  
     identity laws for, 129  
     intersection of, 127, 185  
 Set of real numbers  
     cardinality of, 186  
     uncountability of, 173  
 Set operations, 127  
 Sex, 458  
 Shamir, Adi, 299, 300  
 Shannon, Claude, 20  
 Shannon, Claude Elwood, 811, 812  
 Sheffer, Henry Maurice, 34  
 Sheffer stroke, 34, 36  
 Shift cipher, 295, 306  
     cryptanalysis, 296  
 Shift ciphers  
     cryptosystem, 298  
 Shifting, 252  
 Shifting index of summation, 164  
 Shift register, 846  
 Shortest-path algorithm, 709–714  
 Shortest-path problems, 707–716, 736  
 Showing two sets are equal, 121  
 Sibling of vertex, 747, 803  
 Sides of a polygon, 338  
 Sieve of Eratosthenes, 259, 306, 560, 565  
 Signature  
     digital, 306  
 Signatures  
     digital, 303  
 Signed integer, 855  
 Simple circuit, 679  
 Simple directed graph, 643  
 Simple graphs, 642, 654–658, 735  
     coloring of, 727  
     connected planar, 719–723  
     crossing number of, 726  
     dense, 670  
     edges of, 642, 663  
     isomorphic, 671–675, 736  
     orientation of, 740  
     paths in, 679  
     random, 742  
     self-complementary, 677  
     sparse, 670  
     thickness of, 726  
     vertices of, 642, 663  
     with spanning trees, 785–787

- Simple polygon, 338  
 triangulation of, 338
- Simplification (rule of inference), 71
- Single-elimination tournament, 649
- Single error in an identification number, 291
- Singleton set, 118
- Sink, 901
- Six Degrees of Separation* (Guare), 680
- Slackness of a job, 235
- Sloane  
 Neil, 162
- Sloane, Neil, 163
- Smullyan, Raymond, 19, 20
- Snakes, poisonous, 763
- Sneakers*, 300
- Soccer players, 475
- Social networks, 644
- Socks, 405
- Software  
 origin of term, 11
- Software systems, 17
- Sollin's algorithm, 803
- Solution  
 optimal, 198
- Solution of a recurrence relation, 158
- Solvable problem, 232, 895, 900
- Solvable problems, 226
- Solving linear congruences, 277
- Solving satisfiability problems, 34
- Solving Sudoku puzzles, 32
- Solving systems of linear congruences, 277–279  
 by back substitution, 279
- Solving using generating functions  
 counting problems, 541–546  
 recurrence relations, 546–548
- Sort  
 binary insertion, 203  
 binary insertion sort, 196  
 bubble, 196, 232  
 bidirectional, 233  
 insertion, 196, 197, 232  
   average-case complexity of, 483–484
- library, 196
- Merge, 196
- merge, 367–370, 378, 528  
   complexity of, 532  
   recursive, 368
- quick, 196, 371
- selection, 196, 203
- tournament, 196, 770
- Sorting, 196–198, 232
- Sorting algorithms, 196, 198  
 topological, 627–629, 634
- Space, sample, 446
- Space Complexity, 232
- Space complexity, 219
- Space probe, 476
- Spam, 472–475
- Spam filters, Bayesian, 472–475
- Spanning forest, 796  
 minimum, 802
- Spanning trees, 785–795, 804  
 building  
   by breadth-first search, 789–791  
   by depth-first search, 787–789  
 definition of, 785
- degree-constrained, 806
- distance between, 797
- in IP multicasting, 786
- maximum, 802
- minimum, 797–802, 804
- rooted, 797
- Sparse graphs, 670, 802
- Sparse matrix, 670
- Specification  
 of a Turing machine, 889
- Specifications  
 system, 17
- Specifying  
 algorithms, 192
- Spiders, Web, 794
- Spies, 112
- SQL, 588–589, 855
- Square  
 perfect, 83
- Squarefree integer, 563
- Square Matrix  
 diagonal of, 181  
 inverse of, 184
- Square zero-one matrix  
 Boolean power of, 183
- St. Petersburg Paradox, 497
- Stable assignment, 343
- Stable matching, 204
- Stable matching problem  
 variations on, 234
- Standard deck, 402
- Standard deviation, 487
- Star Height, 901
- Star topology for local area network, 661
- Start symbol, 849
- State  
 accepting  
   finite-state automaton, 867
- final  
   finite-state automaton, 867
- initial, 859  
   finite-state automaton, 867
- reachable, 901
- transient, 901
- State diagram  
 for finite-state machine, 860
- Strategy  
 proof, 102
- Statement  
 conditional, XX  
 biconditional, 9
- Statement(s), A-11  
**if then**, 6  
**procedure**, A-11  
 assignment, A-11–A-12  
 blocks of, A-14–A-15  
 conditional, 6–9  
 for program correctness, 373–375
- Statement, **return**, A-15
- Statements  
 logically equivalent, 45
- Statement variables, 2
- States, 859
- State table  
 for finite-state machine, 860
- Stephen Cook, 227
- Steroids, 475
- Stirling's formula, 151
- Stirling, James, 151
- Stirling numbers of the first kind, 431  
 signless, 443
- Stirling numbers of the second kind, 430
- Strategies  
 minmax, 767, 804  
 proof, 100, 106
- Strategy  
 proof, 85
- Strictly decreasing function, 143
- Strictly decreasing sequence, 403
- Strictly increasing function, 143
- Strictly increasing sequence, 403
- String, 186  
 bit, 110  
 empty, 186, 849  
 null, 849  
 recognized  
   by Turing machine, 891
- String(s)  
 bit, 12
- String matching, 231
- Strings, 157  
 concatenation, 866  
 concatenation of, 350  
 counting, 388  
   without consecutive 0s, 505  
 decoding, 762–763  
 distinguishable, 888  
 generating next largest, 437  
 indistinguishable, 888  
 length of, 505  
   recursive definition of, 350  
   lexicographic ordering of, 620  
 of parentheses, balanced, 382  
 recognized (accepted), 868  
 recursively defined, 349  
 ternary, 511
- Stroke  
 Sheffer, 34
- Stroke, Sheffer, 36
- Strong induction, 333–335, 378
- Strongly connected components of graphs, 686, 736
- Strongly connected graphs, 685
- Strong pseudoprime, 286
- Structural induction, 353–356, 378  
 proof by, 353
- Structured query language, 588–589
- Structures, recursively defined, 349–356
- Stuarts, 151
- Subgraph, 663, 739
- Subsequence, 403
- Subset, 119, 185  
 proper, 185
- Subsets  
 of finite set counting, 388  
   number of, 323  
   sums of, 793
- Subtraction rule, 393
- Subtractors  
 full, 828  
 half, 828
- Subtree, 746, 748, 803
- Success, 458
- Successor  
 of integer, A-5
- Successor of a set, 137
- Sudoku  
 modeling as a satisfiability problem, 32
- Sudoku puzzle, 32
- Sufficient  
 necessary and, 9
- Sufficient condition, 6  
 expressing a conditional statement using, 6
- Sufficient for, 6
- Suitees, 204
- Suitors, 204, 343  
 optimal for, 343
- Sum(s)  
 Boolean, 811–812, 821, 843  
 of first  $n$  positive integers, 315  
 of first  $n$  positive integers, 317  
 of geometric progressions, 318–319  
 of subsets, 793
- Sum-of-products expansions, 820–821, 843  
 simplifying, 828–840

- Summation  
 index of, 163  
 lower limit, 163  
 notation, 162, 186  
 shifting index of, 164  
 upper limit, 163
- Summation formulae  
 proving by mathematical induction, 315–318
- Sum of multisets, 138
- Sum of terms of a geometric progression, 164
- Sum rule, 386, 389, 439
- Sun-Tsu, 277
- Superman, 80
- Surjection, 143, 186
- Surjective (onto) function  
 number of, 560–562, 566
- Surjective function, 143
- Switching circuits, hazard-free, 846
- Symbol(s), 849  
 start, 849  
 terminal, 849
- Symbol, Landau, 206
- Symbolic Logic* (Venn), 120
- Symmetric closure of relation, 598, 634
- Symmetric difference  
 of two sets, 186
- Symmetric matrix, 181, 186
- Symmetric relation, 577–578, 633  
 representing  
 using digraphs, 595  
 using matrices, 591–592
- Syntax, 847
- System  
 RSA, 299
- Systems  
 hardware, 17  
 software, 17
- Systems of linear congruence, 277–279
- System specifications, 17, 50  
 consistent, 18
- T-shirts, 395
- Table  
 truth, 4, 109
- Table(s)  
 state  
 for finite-state machine, 860
- Table, circular, 394
- Table membership, 186
- Tao, Terence, 263
- Tape  
 Turing machine, 889
- Tautology, 25, 110
- Tee-shirts, 395
- Telephone call graph, 682
- Telephone calls, 646
- Telephone lines  
 computer network with diagnostic, 642  
 computer network with multiple, 642  
 computer network with multiple one-way, 642  
 computer network with one-way, 643
- Telephone network, 646, 682
- Telephone number, 646
- Telephone numbering plan, 387
- Template for proofs by mathematical induction, 329
- “Ten Most Wanted” numbers, 262
- Terminals, 849
- Terminal vertex, 594, 654
- Terms of a sequence  
 closed formula for, 159
- Ternary search algorithm, 203
- Ternary string, 511
- Test  
 Miller’s, 286  
 primality, 464–465  
 probabilistic primality, 464–465
- Tetromino, 109
- TeX, 208
- The Art of Computer Programming* (Donald Knuth), 196
- THE BOOK, 260
- The Elements*, 260
- The Elements*(Euclid), 267
- The lady or the tiger puzzle, 24
- Theorem(s), 81, 110  
 alternate names for, 81  
 Archimedean property, A-5  
 art gallery, 735  
 Bayes’, 468–475  
 Bézout’s, 269, 306  
 binomial, 415–418, 439  
 Cantor’s, 177  
 Chinese remainder, 277, 306  
 Cook-Levin, 227  
 Dirac’s, 701  
 extended binomial, 540  
 Fermat’s last, 106  
 Fermat’s little, 281, 306  
 four color, 728–731, 737  
 fundamental theorem of arithmetic, 258, 336–337  
 Green-Tao, 263  
 Hall’s marriage, 659  
 handshaking, 653  
 Jordan curve, 338  
 Kleene’s, 880, 900  
 Kuratowski’s, 724–725, 737  
 Lamé’s, 347  
 master, 532  
 methods of proof, 82  
 multinomial, 434  
 Ore’s, 701, 707  
 Pick’s, 342  
 prime number, 262  
 proving, automated, 114  
 Schröder-Bernstein, 174  
 Wilson’s, 285
- Theory, naïve set, 118
- Theory, Ramsey, 404
- Thesis, Church-Turing, 893
- Thickness of a graph, 726
- Threshold function, 845
- Threshold gate, 845
- Threshold value, 845
- Thue, Axel, 849
- Tic-tac-toe, 766
- Tiger, 24
- Tiling(s), 103
- Tiling of checkerboard, 326
- Time complexity, 219, 232  
 average-case time, 232  
 average case, 220  
 of algorithm for finding maximum, 219  
 of linear search algorithm, 220  
 worst case, 220, 232
- Tooth fairy, 112
- Top-down parsing, 853
- Topological sorting, 627–629, 634
- Topology for local area network  
 hybrid, 661  
 ring, 661  
 star, 661
- Torus, 726
- Total expectation, law of, 492
- Total function, 152
- Totally ordered set, 619, 633
- Total ordering, 619, 633  
 compatible, 628, 634
- Tournament, 741  
 round-robin, 468, 649  
 single-elimination, 649
- Tournament sort, 196, 770
- Tower of Hanoi, 503–504
- Trace of recursive algorithm, 361, 362
- Tractable problem(s), 226, 232, 897
- Trail, 679
- Transducer(s)  
 finite-state, 859
- Transformation  
 affine, 296
- Transformations, 139
- Transient state, 901
- Transition function, 859  
 extended, 867  
 extending, 867  
 extension, 867
- Transition rule(s)  
 Turing machine, 889
- Transitive closure of relation, 597, 600–603, 634  
 computing, 603–606
- Transitive relation, 578–580, 633  
 representing, using digraphs, 596
- Transitivity law, A-2
- Translating  
 English sentences to logical expressions, 16–17,  
 62–63  
 mathematical statements to logical expressions, 60,  
 61  
 nested quantifiers into English, 61–62  
 logical statements into English, 61–62
- Transpose of a matrix, 181, 186
- Transposition cipher, 297  
 decryption, 297  
 encryption, 297
- Transposition error, 291
- “Traveler’s Dodecahedron,” 698, 700
- Traveling salesperson problem, 702, 709, 714–716,  
 736
- Traversal of tree, 772–782, 804  
 inorder, 773, 775, 778, 804  
 level-order, 806  
 postorder, 773, 776–778, 804  
 preorder, 773–775, 778, 804
- Tree  
 derivation, 899  
 parse, 899
- Tree(s), 745–802  
 m-ary, 748, 804  
 complete, 756  
 height of, 754–755  
 applications of, 757–769  
 as models, 749–752  
 AVL, 808  
 binary, 352–353, 748, 749  
 extended, 352  
 full, 352–353  
 binary search, 757–759, 804  
 binomial, 805  
 caterpillar, 807  
 decision, 760–762, 804  
 definition of, 746  
 derivation, 852–854  
 extended binary, 352  
 family, 745  
 full m-ary, 748, 752, 804  
 full binary, 352  
 game, 764–769  
 graceful, 807  
 height-balanced, 808  
 labeled, 756  
 mesh of, 809

- Tree(s)—*Cont.*
- parse, 852
  - properties of, 752–755
  - quad, 809
  - rooted, 351, 747–749
    - $S_k$ -tree, 806
    - B-tree of degree  $k$ , 805
    - balanced, 753, 804
    - binomial, 805
    - decision trees, 760–762
    - definition of, 803
    - height of, 753, 804
    - level order of vertices of, 806
    - ordered, 749, 804, 806
    - rooted Fibonacci, 757
    - spanning, 785–795, 804
      - definition of, 785
      - degree-constrained, 806
      - distance between, 797
      - in IP multicasting, 786
      - maximum, 802
      - minimum, 797–802, 804
      - rooted, 797
  - Tree-connected network, 751–752
  - Tree-connected parallel processors, 751–752
  - Tree diagrams, 394–395, 439
  - Tree edges, 788
  - Tree traversal, 772–782, 804
    - inorder, 773, 775, 778, 804
    - postorder, 773, 776–778, 804
    - preorder, 773–775, 778, 804
  - Trial division, 258
  - Triangle inequality, 108
  - Triangle, Pascal's, 419, 439
  - Triangulation, 338
  - Trichotomy Law, A-2
  - Tricks, magic, 20
  - Triomino(es), 105, 326, 333
    - Right, 105, 333
    - straight, 105
  - Trivial proof, 84, 110
  - True negative, 471
  - True positive, 471
  - Truth set
    - of a predicate, 125
  - Truth table(s), 4, 109
    - for biconditional statements, 9
    - for conditional statement, 6
    - for conjunction, 4
    - for disjunction, 4
    - for exclusive or, 6
    - for logical equivalences, 27
    - for negation, 4
    - for XOR, 6
    - of compound propositions, 10
  - Truth value, 3, 109
    - of implication, 6
  - Tukey, John Wilder, 11
    - coining words, 11
  - Turing, Alan Mathison, 226, 886
  - Turing Award, 854
  - Turing machine, 847, 886, 888, 889, 893, 899
    - computing functions with, 892–893
    - control unit, 889
    - definition of, 889
    - final state, 891
    - halts, 891
    - in computational complexity, 894
    - initial position, 889
    - initial state, 889
    - nondeterministic, 893, 899
    - recognition of nonregular set, 891
    - sets recognized by, 891, 892
  - specification, 889
  - string recognition by, 891
  - tape, 889
  - transition rules, 889
  - types of, 893
  - Twin prime conjecture, 264
  - Twin primes, 264
  - Two's complement representations, 256
  - Two-dimensional array, 662–663
  - Two children problem, 498
  - Type, 117
  - Type 0 grammar, 851, 899
  - Type 1 grammar, 851, 899
  - Type 2 grammar, 851, 899
  - Type 3 grammar, 851, 883, 899
  - U. S. Coast Survey, 38
  - Ulam's problem, 536
  - Ulam, Stanislaw, 536
  - Ulam numbers, 188
  - Unary representations, 892
  - Uncomputable function, 175, 186, 896, 900
  - Uncountability
    - set of real numbers, 173
  - Uncountable set, 186
  - Undecidable problem, 895
  - Undefined values
    - of partial function, 152
  - Underlying undirected graph, 654, 736
  - Undirected edges, 644, 735
    - of simple graph, 643
  - Undirected graphs, 644, 653, 735
    - connectedness in, 681–685
    - Euler circuit of, 694
    - Euler path of, 698
    - orientation of, 740
    - paths in, 679, 736
    - underlying, 654, 736
  - Unicasting, 786
  - Unicorns, 112
  - Unicycle, 812
  - Uniform distribution, 454, 494
  - Unimodal sequence, 568
  - Union
    - of graphs, 664
    - of three finite sets, number of elements in, 554–556, 566
    - of two finite sets, number of elements in, 553, 565
  - Union of
    - fuzzy sets, 138
  - Union of a collection of sets, 133
  - Union of multisets, 138
  - Union of sets, 127, 185
  - Uniqueness proof, 110
  - Uniqueness quantifier, 44
  - Uniqueness
    - proof, 99
  - Unit (Egyptian) fraction, 380
  - Unit-delay machine, 861–862
  - unit-delay machine, 861
  - United States Coast Survey, 38
  - Unit property, in Boolean algebra, 815
  - UNIVAC, 872
  - Universal quantification, 110
  - Universal address system, 772–773
  - Universal generalization, 76
  - Universal instantiation, 75
  - Universal modus ponens, 77
  - Universal modus tollens, 77
  - Universal product code (UPC), 290
  - Universal quantification, 40
  - Universal quantifier, 40
  - Universal set, 118, 185
  - Universe of discourse, 40, 110
  - Unlabeled
    - boxes, 428
    - objects, 428
  - Unless, 6
    - expressing conditional statement using, 7
  - Unsatisfiable compound proposition, 30
  - Unsolvable problem, 232, 895, 900
  - Unsolvable problems, 226
  - UPC check digit, 290
  - Upper bound, A-2
    - of lattice, 637
    - of poset, 625, 634
  - Upper limit of a summation, 163
  - Upper triangular matrix, 231
  - Utilities-and-houses problem, 718
  - Uzbekistan, 192
  - Vacuous proof, 84, 110
  - Valid argument, 69, 110
  - Valid argument form, 70, 110
  - Valid partner, 234
  - Value
    - truth, 3, 109
  - Value(s)
    - expected, 477–480, 491, 494
      - in hatcheck problem, 495
      - linearity of, 477–484, 494
      - of inversions in permutation, 482–484
    - final, A-14
    - initial, A-14
    - of tree, 767
    - of vertex in game tree, 767–769, 804
    - threshold, 845
  - Vandermonde's identity, 420–421
  - Vandermonde, Alexandre-Théophile, 420
  - Variable(s)
    - Boolean, 11, 110, 814, 820, 843
    - bound, 44, 110
    - free, 44, 110
    - propositional, 2, 109
    - random, 454, 460–461
      - covariance of, 494
      - definition of, 494
      - distribution of, 460, 494
      - expected values of geometric, 477–480
      - geometric, 484–485
      - geometric expected values of, 491, 494
      - independent, 485–487, 494
      - indicator, 492
      - standard deviation of, 487
      - variance of, 477–480, 494
    - statement, 2
  - Variance, 477, 487–490, 494
  - Veitch, E. W., 830
  - Vending machine, 858–859
  - Venn, John, 118, 120
  - Venn diagrams, 118, 185
  - Verb, 848
  - Verb phrase, 848
  - Vertex (vertices), 594
    - adjacent, 651, 654, 735
    - ancestor of, 747, 803
    - basis, 691
    - child of, 747, 749, 803
    - connecting, 651
    - connectivity, 684
    - counting paths between, 688–689
    - cut, 683, 684
    - degree of, 652
    - degree of, in undirected graph, 652
    - descendant of, 747, 804
    - distance between, 741
    - eccentricity of, 757
    - end, 654

- Vertex (vertices)—*Cont.*  
 in-degree of, 654, 736  
 independent set of, 741  
 initial, 594, 654  
 interior, 603  
 internal, 748, 804  
 isolated, 652, 736  
 level of, 753, 804  
 level order of, 806  
 number of, of full binary tree, 355  
 of directed graph, 643, 654  
 of directed multigraph, 644  
 of multigraph, 642  
 of polygon, 338  
 of pseudograph, 643  
 of simple graph, 642, 668  
 of undirected graph, 644, 652, 654  
 out-degree of, 654, 736  
 parent of, 747, 803  
 pendant, 652, 736  
 sibling of, 747, 803  
 terminal, 594, 654  
 value of, in game tree, 767–769, 804  
 Vertex set, 338  
 bipartition of, 656  
 Very large scale integration (VLSI) graphs, 744  
 Vigenère cipher, 304  
 cryptanalysis, 305  
 Vocabulary, 849, 899  
 "Voyage Around the World" Puzzle, 700  
 Walk, 679  
 closed, 679  
*Wall Street*, 198  
 Warshall's algorithm, 603–606  
 Warshall, Stephen, 603, 604  
 WAVES, Navy, 872  
 Weakly connected graphs, 686  
 Web crawlers, 646  
 Web graph, 646–647, 794  
 strongly connected components of, 686  
 Web page(s), 646, 686, 794  
 searching, 18  
 Web spiders, 794  
 Weighted graphs, 736  
 minimum spanning tree for, 798–802  
 shortest path between, 707–714  
 traveling salesperson problem with, 714–716  
 Well-formed expressions, 354  
 Well-formed formula, 350  
 for compound propositions, 354  
 in prefix notation, 784  
 of operators and operands, 351  
 structural induction and, 354  
 Well-founded induction, principle of, 635  
 Well-founded poset, 632  
 Well-ordered induction, 634  
 principle of, 620  
 Well-ordered set, 381, 620, 633  
 Well-ordering property, 340–341, 378, A-5  
*WFF'N PROOF, The Game of Modern Logic* (Allen), 80, 114  
 Wheels, 655, 736  
 Wiles, Andrew, 107  
 Williamson, Malcolm, 302  
 Wilson's theorem, 285  
 Winning strategy  
 for chomp, 98  
 Without loss of Generality (WLOG), 95, 110  
 Witness(es)  
 to big-*O* relationship, 205, 232  
 to an existence proof, 96  
 Witnesses to a big-*O* relationship, 232  
 WLOG (without loss of generality), 95  
 Word, 849  
*Word and Object* (Quine), 839  
 World's record, for twin primes, 264  
 World Cup soccer tournament, 415  
 World population, 168  
 World Wide Web graph, 646–647  
 Worst-case complexity  
 insertion sort, 222  
 of binary search, 220  
 of bubble sort, 221  
 Worst-case time complexity, 232  
 Worst case  
 time complexity, 220  
 XML, 855, 858  
 XOR, 110  
 Yahoo, 794  
 Yes-or-no problems, 894  
 Zebra puzzle, 24  
 Zermelo–Fraenel axioms, 176  
 Zero-one matrices  
 Boolean product of, 182  
 join of, 181  
 meet of, 181  
 of transitive closure, 602–603  
 representing relations using, 591–594  
 Zero-one matrix, 181  
 Zero-one matrix, 186  
 Zero property, in Boolean algebra, 815  
 Ziegler's Giant Bar, 208  
 Zodiac, signs of, 441