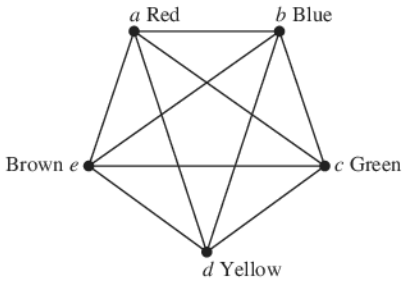


FIGURE 5 **A Coloring of K_5.**

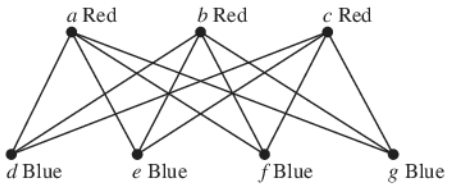


FIGURE 6 **A Coloring of $K_{3,4}$.**

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. ◀

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. ◀

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.

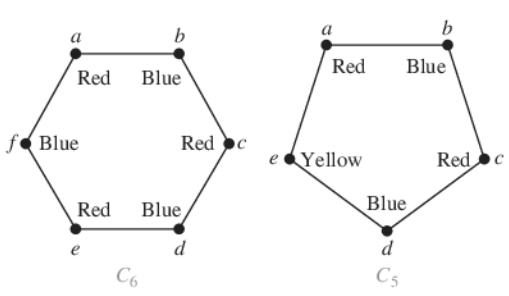


FIGURE 7 **Colorings of C_5 and C_6.**

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 nth 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 nth 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 $n \geq 3$. ◀

Links

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. ◀

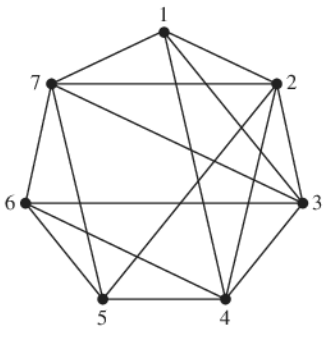
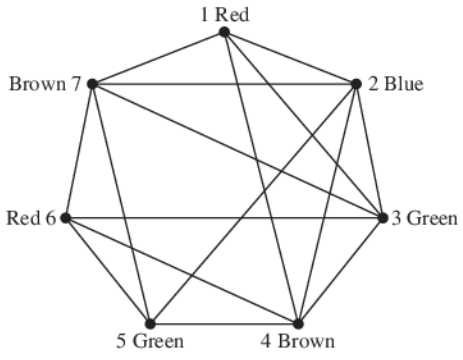


FIGURE 8 The Graph Representing the Scheduling of Final Exams.



Time Period	Courses
I	1, 6
II	2
III	3, 5
IV	4, 7

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. ◀

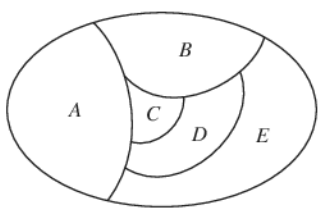
An application of graph coloring to compilers is considered in Example 7.

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. ◀

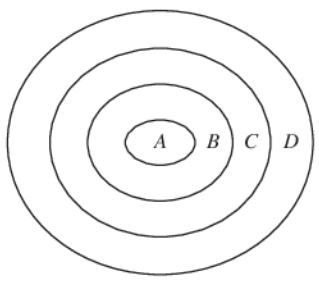
Exercises

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.

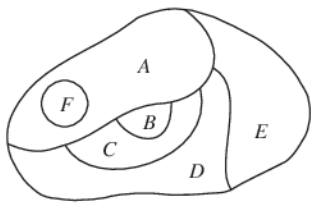
1.



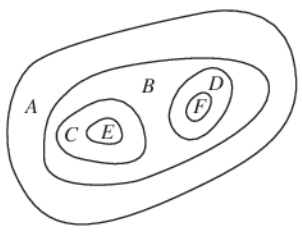
2.



3.

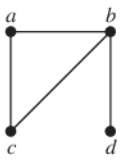


4.

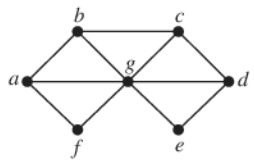


In Exercises 5–11 find the chromatic number of the given graph.

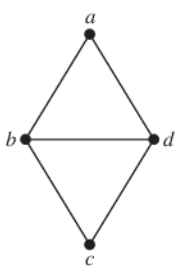
5.



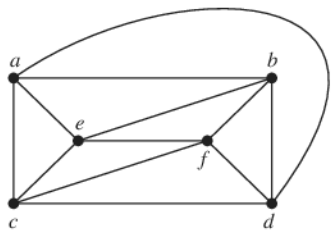
6.



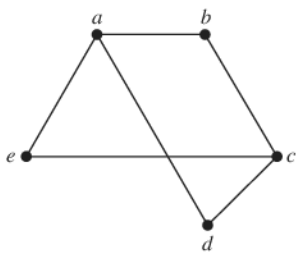
7.



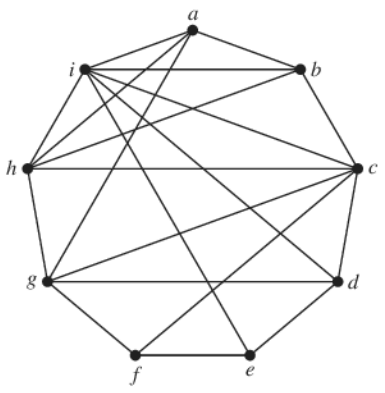
8.



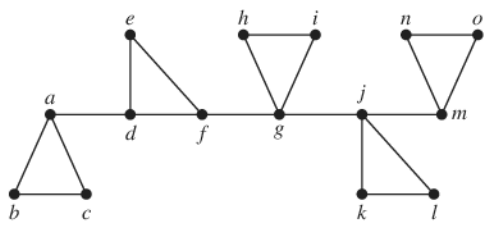
9.



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

a) C_n, where $n \geq 3$.

b) 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.

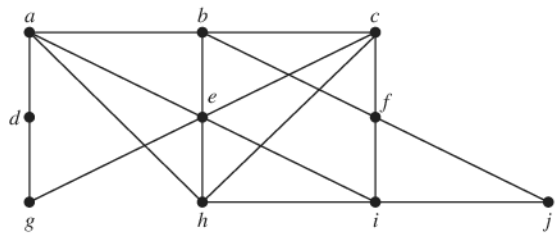
***26.** Find the edge chromatic number of K_n when n is a positive integer.

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 $v_1, v_2, v_3, \ldots, v_n$ in order of decreasing degree so that $\deg(v_1) \geq \deg(v_2) \geq \cdots \geq \deg(v_n)$. 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 $k - 1$.

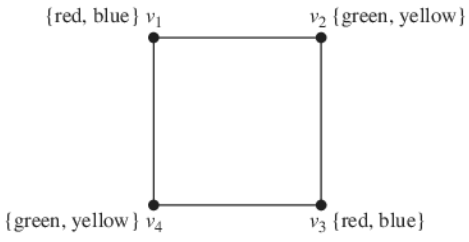
32. Show that C_n is chromatically 3-critical whenever n is an odd positive integer, $n \geq 3$.

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:

a) $\chi_2(K_3)$ **b)** $\chi_2(K_4)$ **c)** $\chi_2(W_4)$

d) $\chi_2(C_5)$ **e)** $\chi_2(K_{3,4})$ **f)** $\chi_3(K_5)$

***g)** $\chi_3(C_5)$ **h)** $\chi_3(K_{4,5})$

***37.** Let G and H be the graphs displayed in Figure 3. Find
a) $\chi_2(G)$. **b)** $\chi_2(H)$.
c) $\chi_3(G)$. **d)** $\chi_3(H)$.

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.]

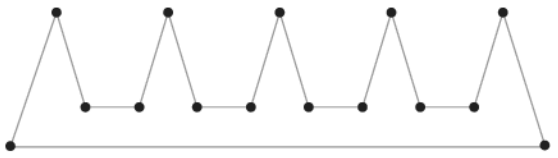
The famous Art Gallery Problem asks how many guards are needed to see all parts of an art gallery, where the gallery is the interior and boundary of a polygon with n sides. To state this problem more precisely, we need some terminology. A point x inside or on the boundary of a simple polygon P **covers** 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

$\deg^-(v)$ (the in-degree of the vertex v in a graph with directed edges): the number of edges with v as their terminal vertex

$\deg^+(v)$ (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 $v_1, v_2, \ldots, v_n$ and edges $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$

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 $e_1, e_2, \ldots, e_n$, where e_i is associated to $\{x_i, x_{i+1}\}$ for $i = 0, 1, \ldots, n$, where $x_0 = u$ and $x_{n+1} = v$

path from u to v in a graph with directed edges: a sequence of edges $e_1, e_2, \ldots, e_n$, where e_i is associated to (x_i, x_{i+1}) for $i = 0, 1, \ldots, n$, 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 weghted 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.

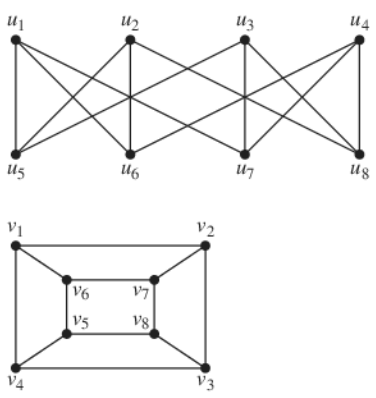
Supplementary Exercises

1. How many edges does a 50-regular graph with 100 vertices have?

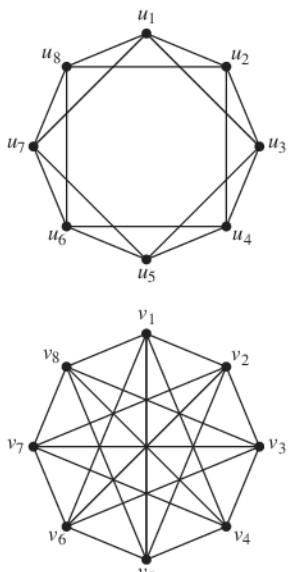
2. How many nonisomorphic subgraphs does K_3 have?

In Exercises 3–5 determine whether two given graphs are isomorphic.

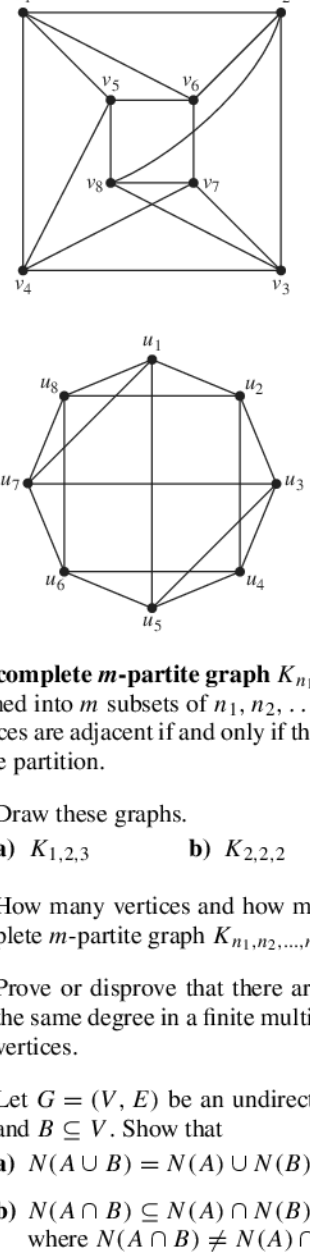
3.



4.



***5.**



The **complete m-partite graph** $K_{n_1,n_2,\ldots,n_m}$ has vertices partitioned into m subsets of $n_1, n_2, \ldots, n_m$ 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 $K_{n_1,n_2,\ldots,n_m}$ 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 $S_1, S_2, \ldots, S_n$ 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 $(a_1, a_2, \ldots, a_n)$ with the property that $a_i \in S_i$ for $i = 1, 2, \ldots, n$ 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 $S_1, S_2, \ldots, S_n$ of a set S has a SDR $(a_1, a_2, \ldots, a_n)$ if and only if $|\bigcup_{i \in I} S_i| \geq |I|$ for all subsets I of $\{1, 2, \ldots, n\}$.

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

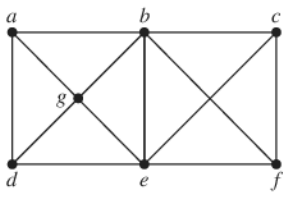
d) the protein interaction graph for a human cell

e) the graph representing the routers and communications links that make up the worldwide Internet

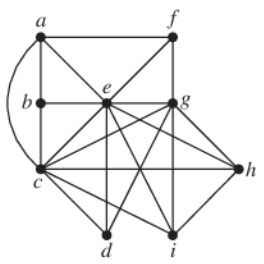
18. For each of the graphs in Exercise 17, explain whether you would expect its clustering coefficient to be closer to 0.01 or to 0.10 and why you expect this.

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.

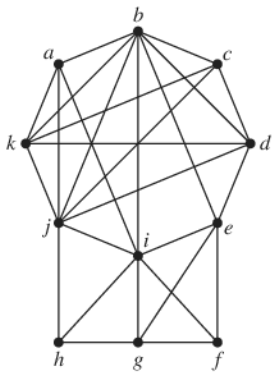
19.



20.

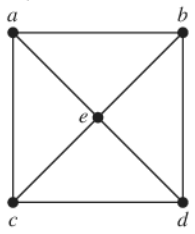


21.

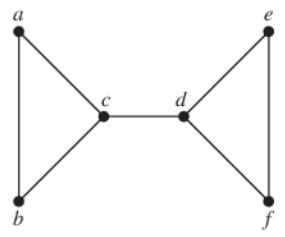


A **dominating set** of vertices in a simple graph is a set of vertices such that every other vertex is adjacent to at least one vertex of this set. A dominating set with the least number of vertices is called a **minimum dominating set**. In Exercises 22–24 find a minimum dominating set for the given graph.

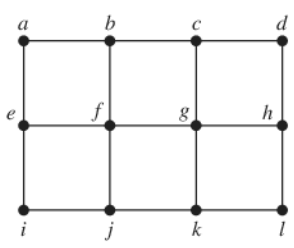
22.



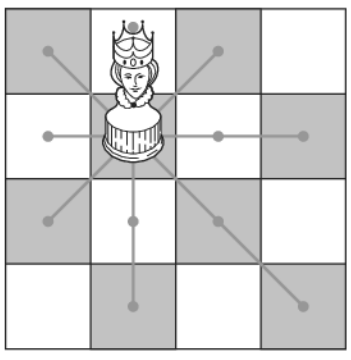
23.



24.



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.



The Squares Controlled by a Queen

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?

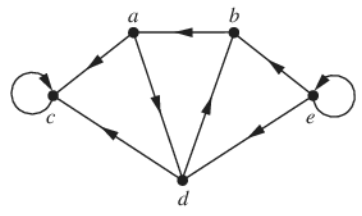
b) with chromatic number equal to four?

c) that are nonplanar?

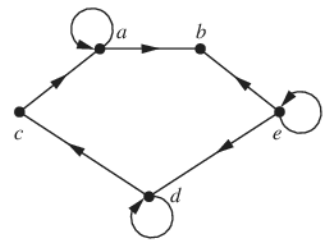
A directed graph is **self-converse** if it is isomorphic to its converse.

33. Determine whether the following graphs are self-converse.

a)



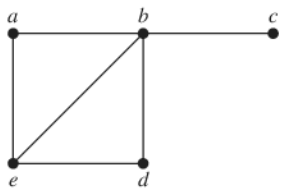
b)



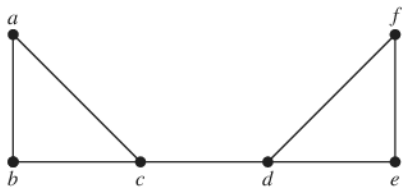
34. Show that if the directed graph G is self-converse and H is a directed graph isomorphic to G, then H is also self-converse.

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.

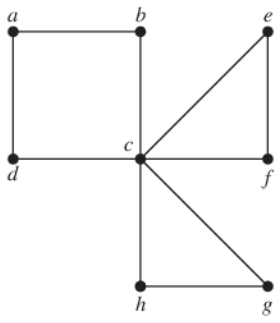
35.



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, Degore, 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 $a_1, a_2, \ldots, a_n$ of the vertices of G, of $\max\{|i - j| | a_i$ 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.

CHAPTER

11 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

Links

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.

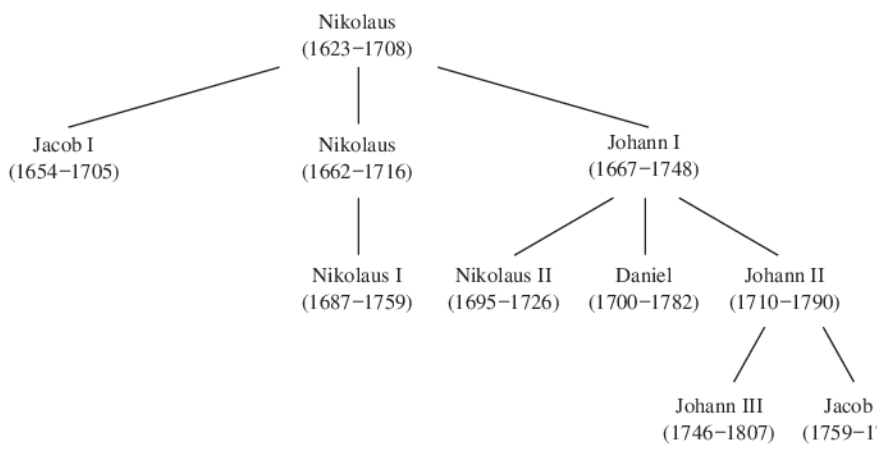


FIGURE 1 **The Bernoulli Family of Mathematicians.**

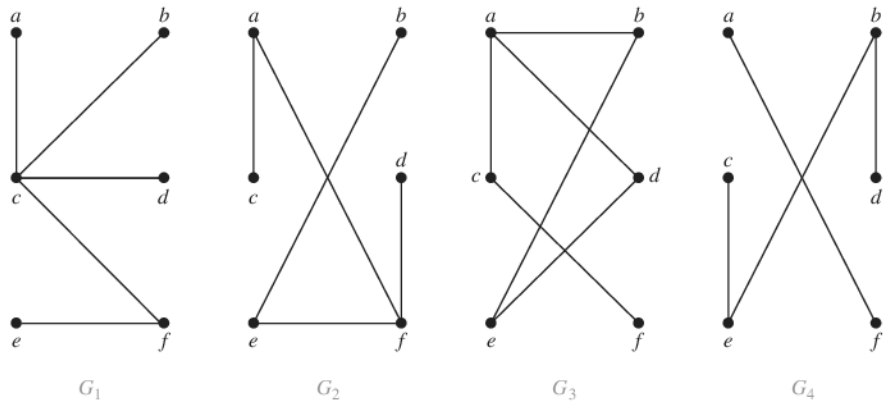


FIGURE 2 **Examples of Trees and Graphs That Are Not Trees.**

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.

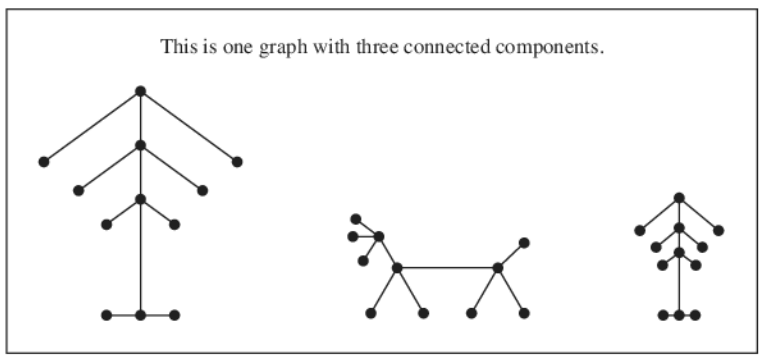


FIGURE 3 **Example of a Forest.**

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. ◁

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

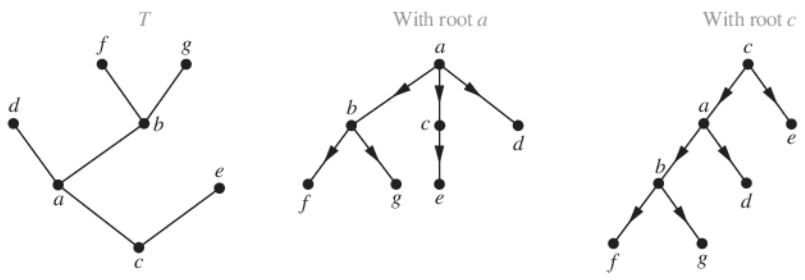


FIGURE 4 **A Tree and Rooted Trees Formed by Designating Two Different Roots.**

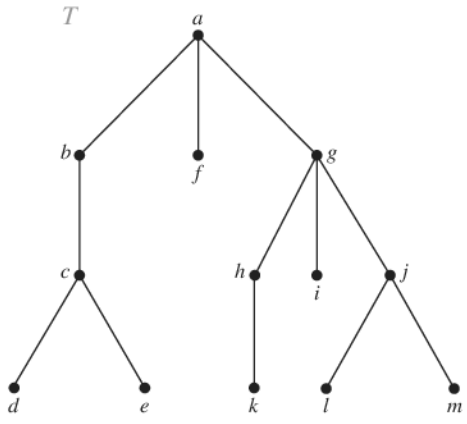


FIGURE 5 **A Rooted Tree T.**

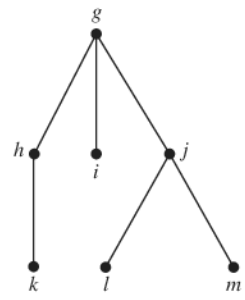


FIGURE 6 **The Subtree Rooted at g.**

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?

Extra Examples

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

Links

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?

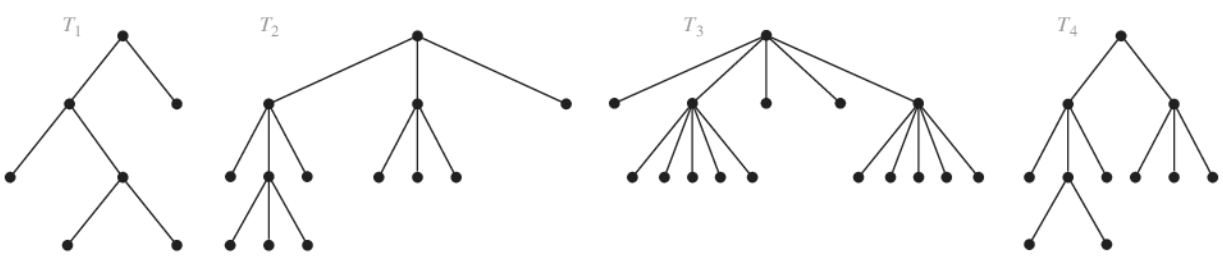


FIGURE 7 **Four Rooted Trees.**

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. ◀

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. ◀

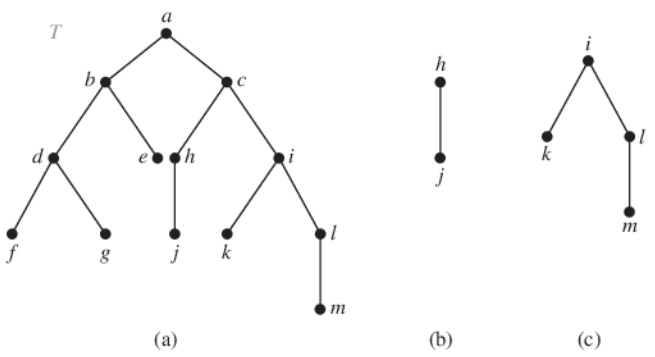


FIGURE 8 **A Binary Tree T and Left and Right Subtrees of the Vertex c.**

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.

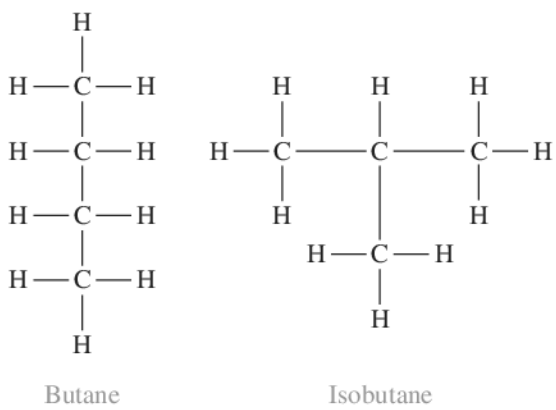


FIGURE 9 The Two Isomers of Butane.

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*.

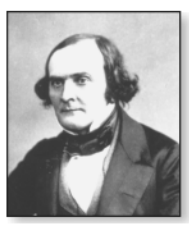
In graph models of saturated hydrocarbons, each carbon atom is represented by a vertex of degree 4, and each hydrogen atom is represented by a vertex of degree 1. There are $3n + 2$ vertices in a graph representing a compound of the form C_nH_{2n+2}. The number of edges in such a graph is half the sum 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

Links



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.

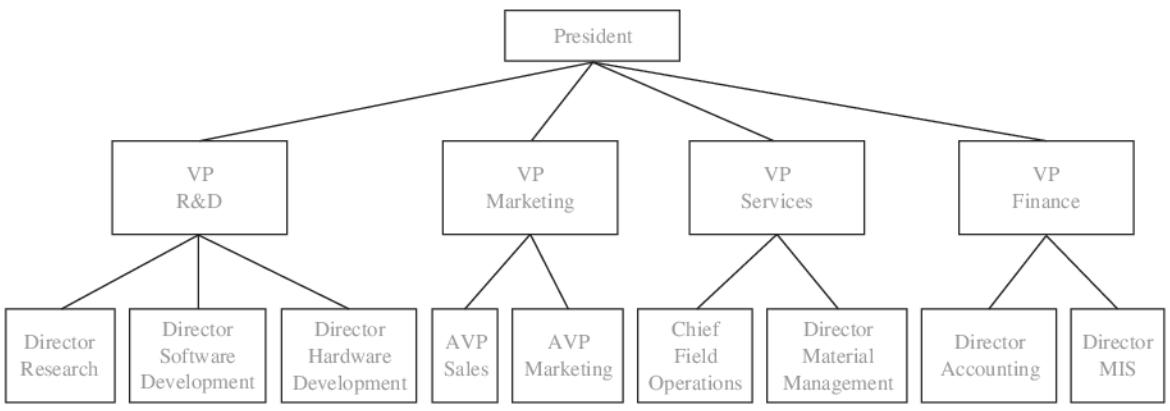


FIGURE 10 **An Organizational Tree for a Computer Company.**

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.) ◀

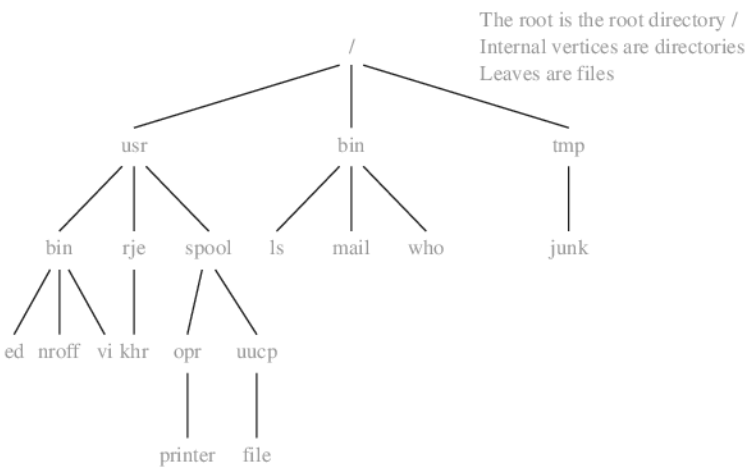


FIGURE 11 **A Computer File System.**

EXAMPLE 8 **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.

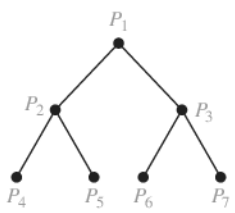


FIGURE 12 **A Tree-Connected Network of Seven Processors.**

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. ◀

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.

Links

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. ◁

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. ◁

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 $l = n-i = n-(n-1)/m = [(m-1)n+1]/m$. ◁

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?

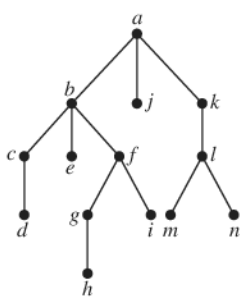


FIGURE 13 A Rooted Tree.

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. ◁

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. ◁

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. ◁

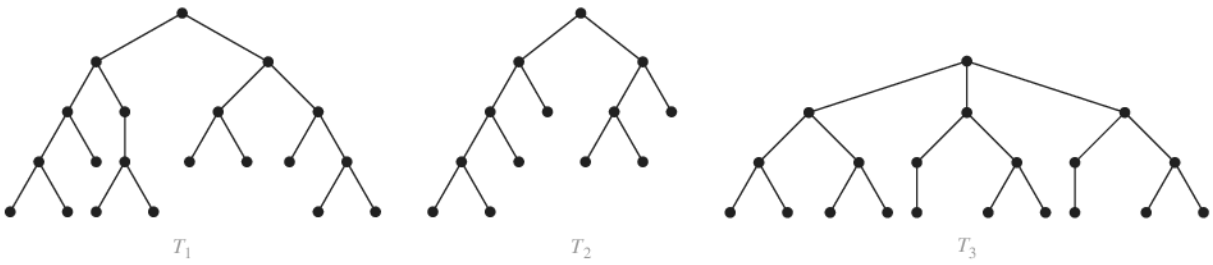


FIGURE 14 Some Rooted Trees.

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. ◁

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.

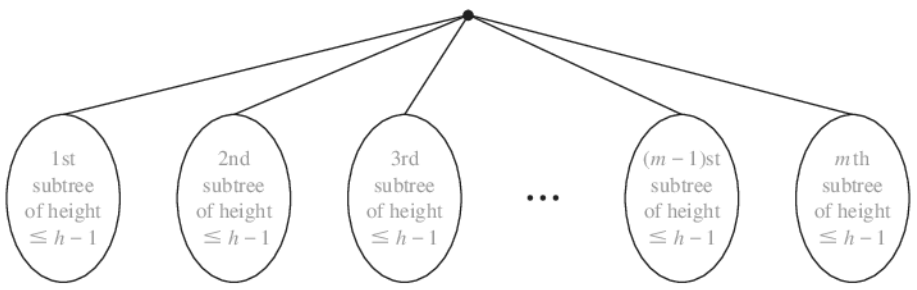
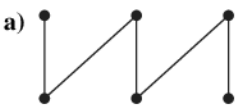


FIGURE 15 The Inductive Step of the Proof.

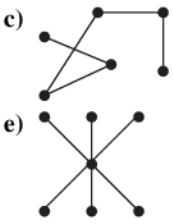
Then each leaf is at level h or $h - 1$, and because the height is h, there is at least one leaf at level h. It follows that there must be more than m^{h-1} leaves (see Exercise 30). Because $l \leq m^h$, we have $m^{h-1} < l \leq m^h$. Taking logarithms to the base m in this inequality gives $h - 1 < \log_m l \leq h$. Hence, $h = \lceil \log_m l \rceil$. ◁

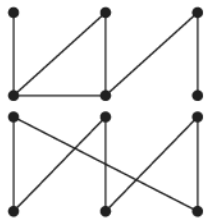
Exercises

1. Which of these graphs are trees?

a)


b)

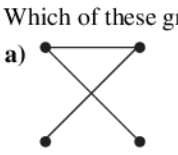

c)


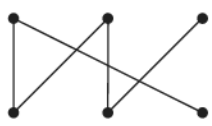
d)


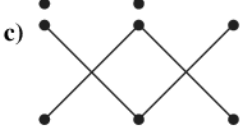
e)

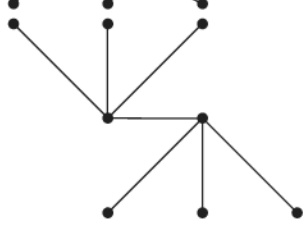
f)

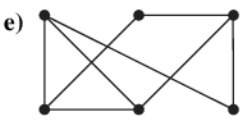
2. Which of these graphs are trees?

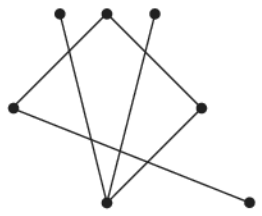
a)


b)


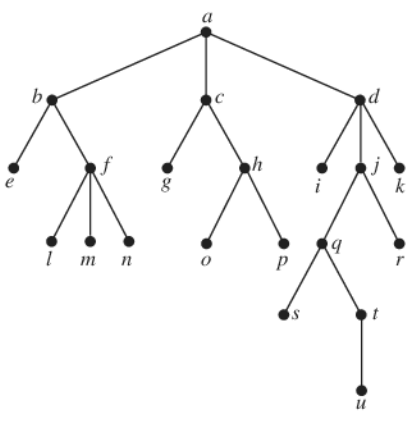
c)


d)


e)


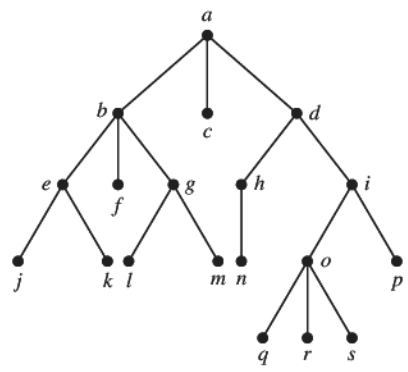
f)


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.

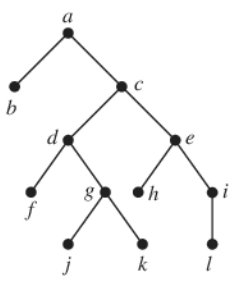
b) Show how 16 numbers can be added using the 15 processors in part (a) using four steps.

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.

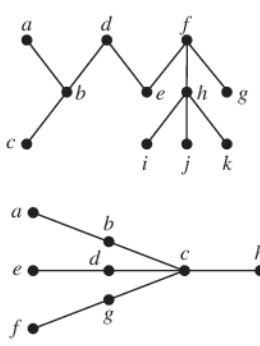
***38.** A **labeled tree** is a tree where each vertex is assigned a label. Two labeled trees are considered isomorphic when there is an isomorphism between them that preserves the labels of vertices. How many nonisomorphic trees are there with three vertices labeled with different integers from the 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 $n = 3, 4, \ldots$, 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

Links

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 *geography* 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 $label(v)$ 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.

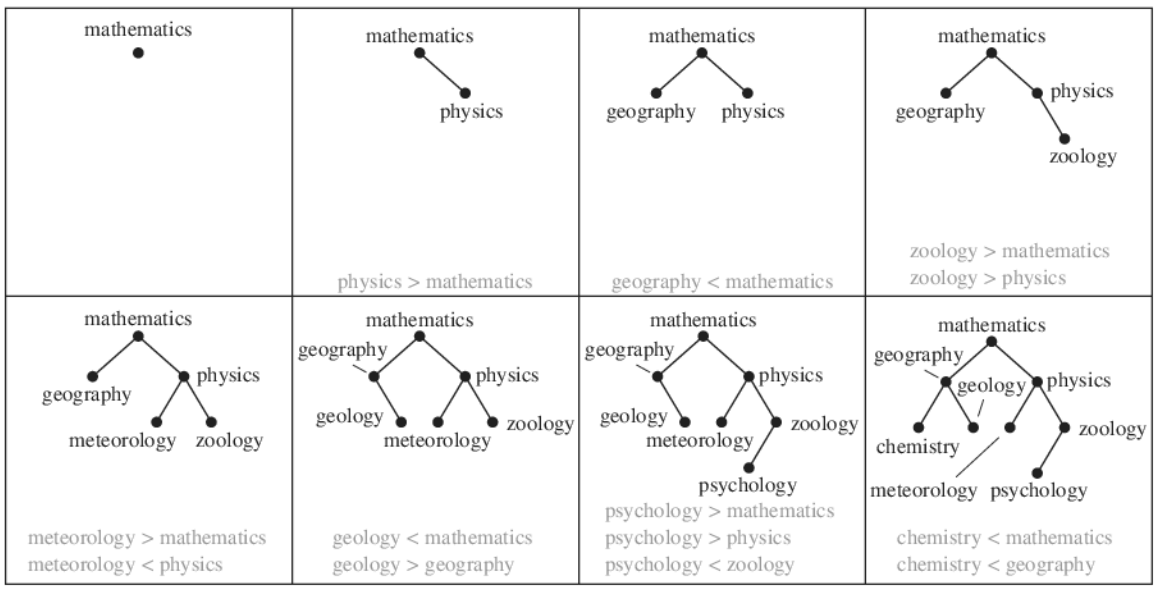


FIGURE 1 **Constructing a Binary Search Tree.**

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.

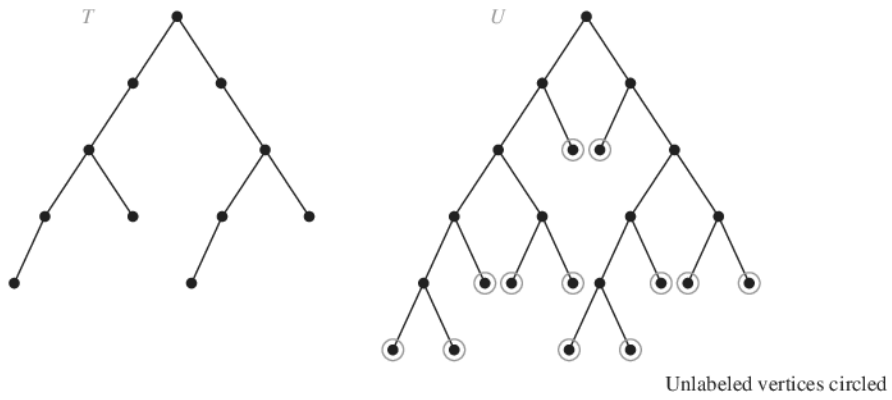


FIGURE 2 **Adding Unlabeled Vertices to Make a Binary Search Tree Full.**

Decision Trees

Links

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.

Extra Examples

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.

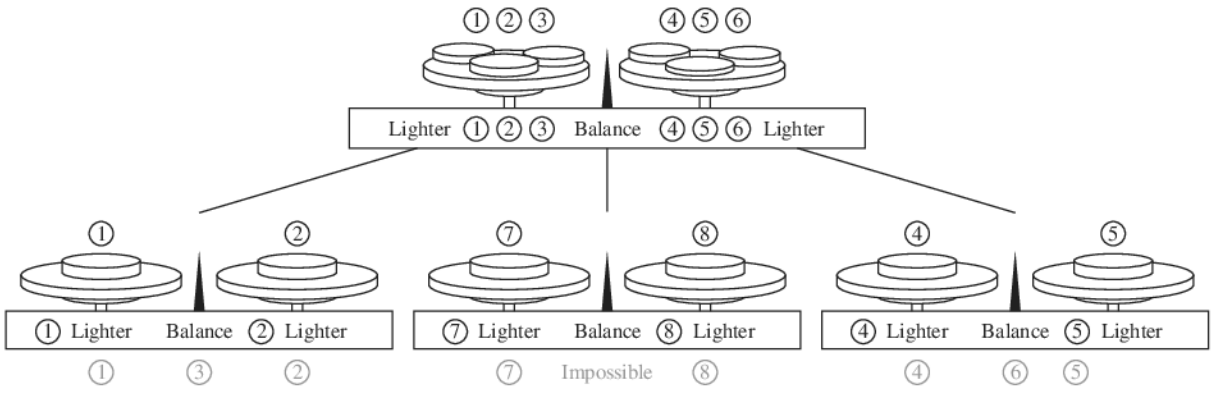


FIGURE 3 **A Decision Tree for Locating a Counterfeit Coin. The counterfeit coin is shown in color below each final weighing.**

EXAMPLE 4 We display in Figure 4 a decision tree that orders the elements of the list a, b, c. ◀

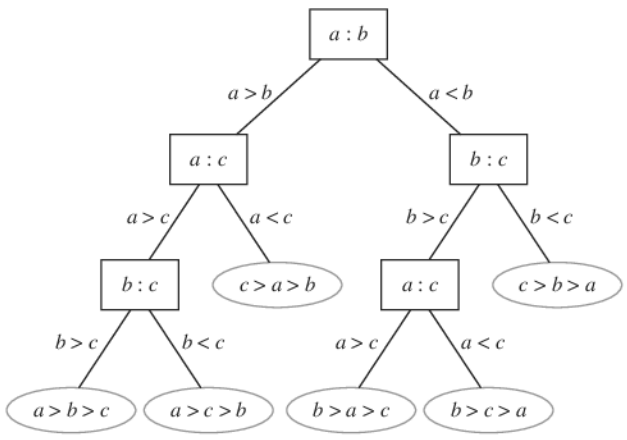


FIGURE 4 **A Decision Tree for Sorting Three Distinct Elements.**

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.

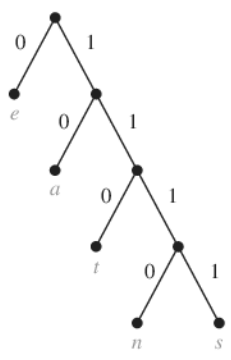


FIGURE 5 A Binary Tree with a Prefix Code.

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 11111011100 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.

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.

Links 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.

Demo

Links

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, $i = 1, \ldots, n$)
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?

Extra Examples

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.

Huffman coding is used in JPEG image coding

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.

Game Trees

Links

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

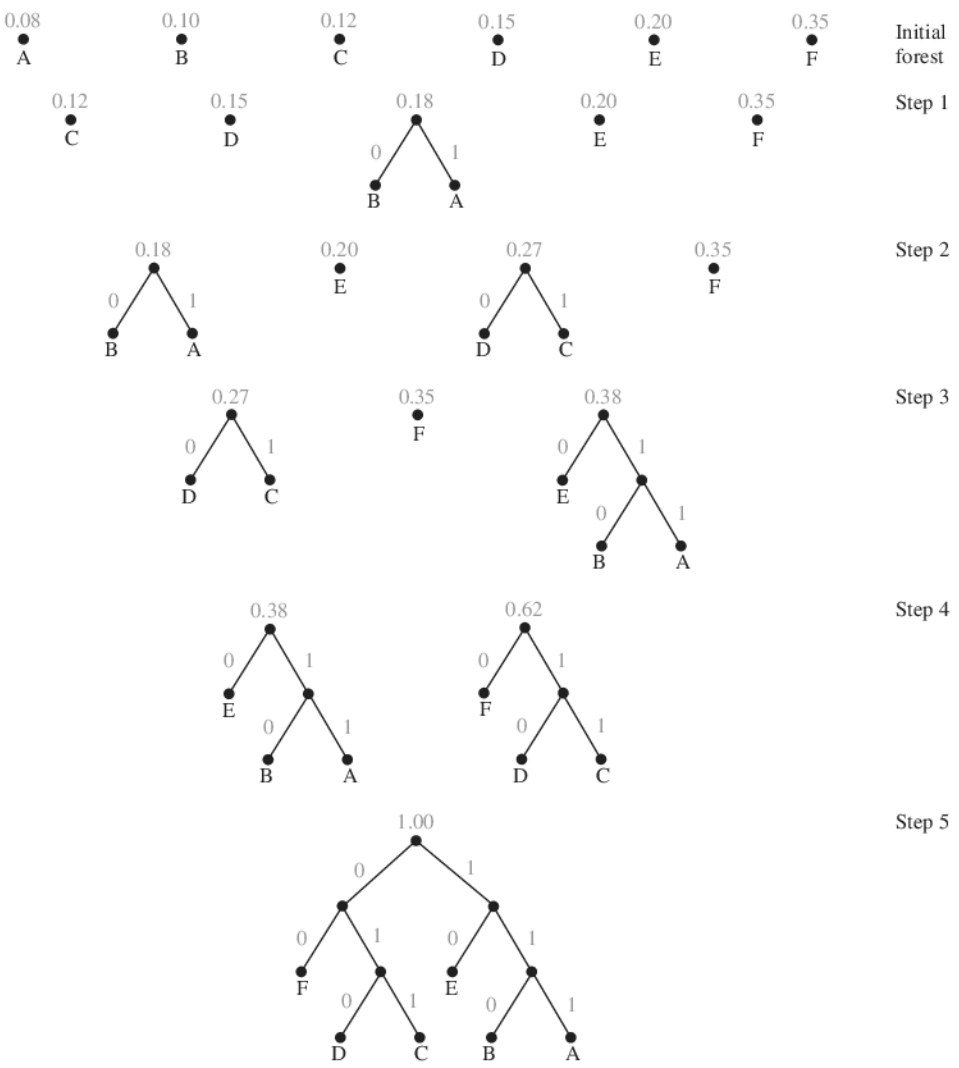


FIGURE 6 **Huffman Coding of Symbols in Example 4.**

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.

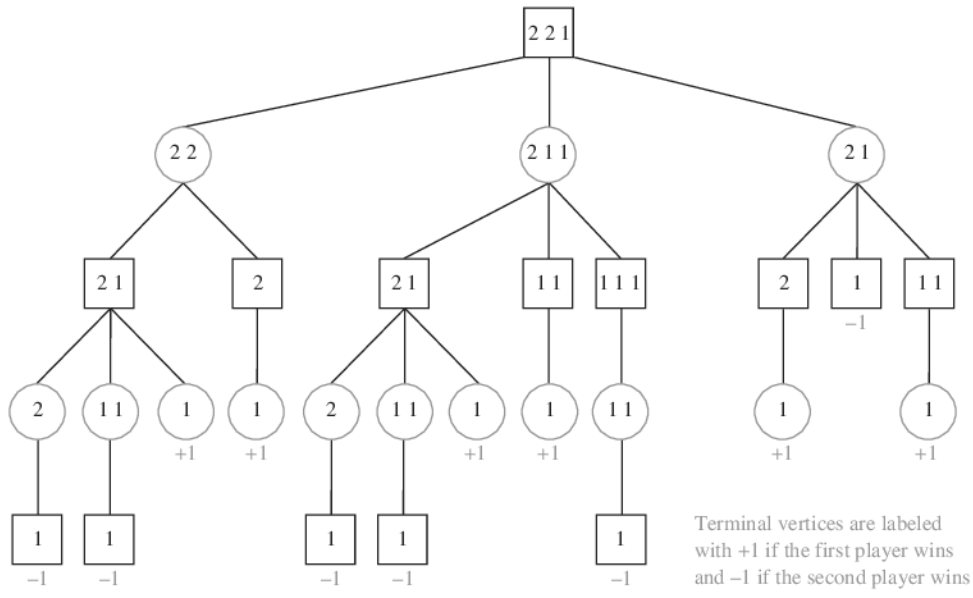


FIGURE 7 **The Game Tree for a Game of Nim.**

EXAMPLE 6 **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. ◀

Although nim is an ancient game, Charles Bouton coined its modern name in 1901 after an archaic English word meaning "to steal."

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. ◀

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.

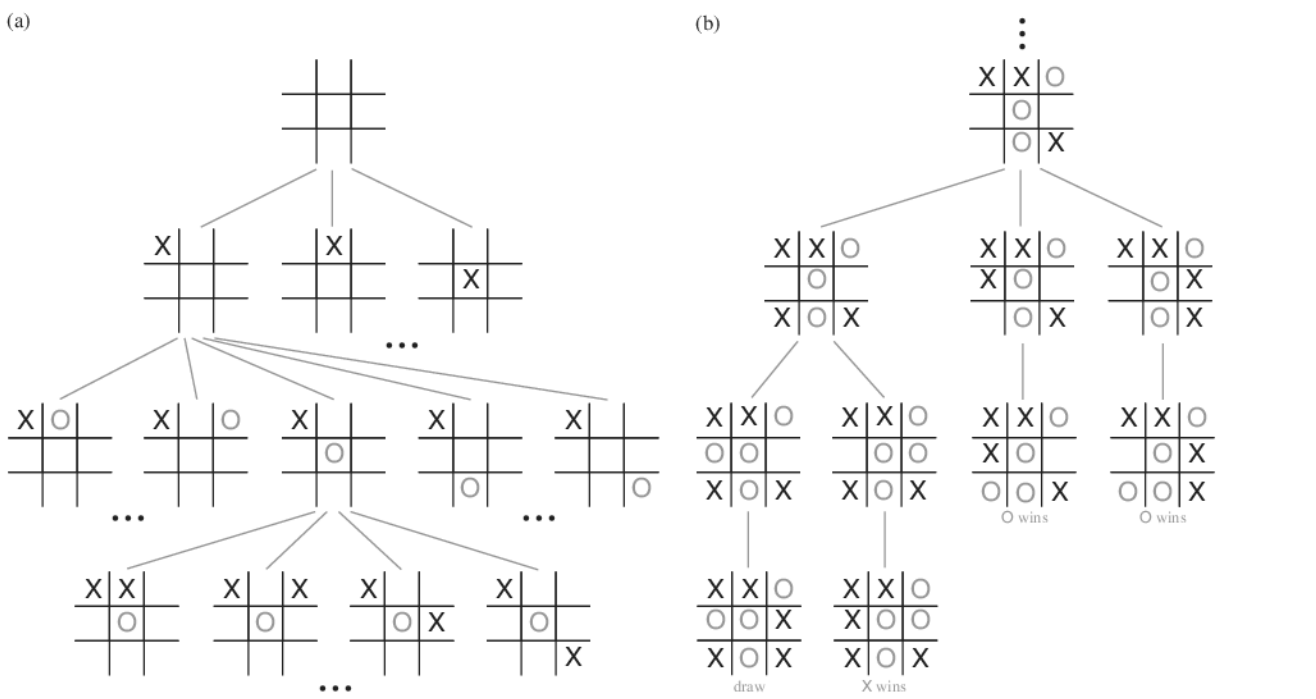


FIGURE 8 **Some of the Game Tree for Tic-Tac-Toe.**

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.

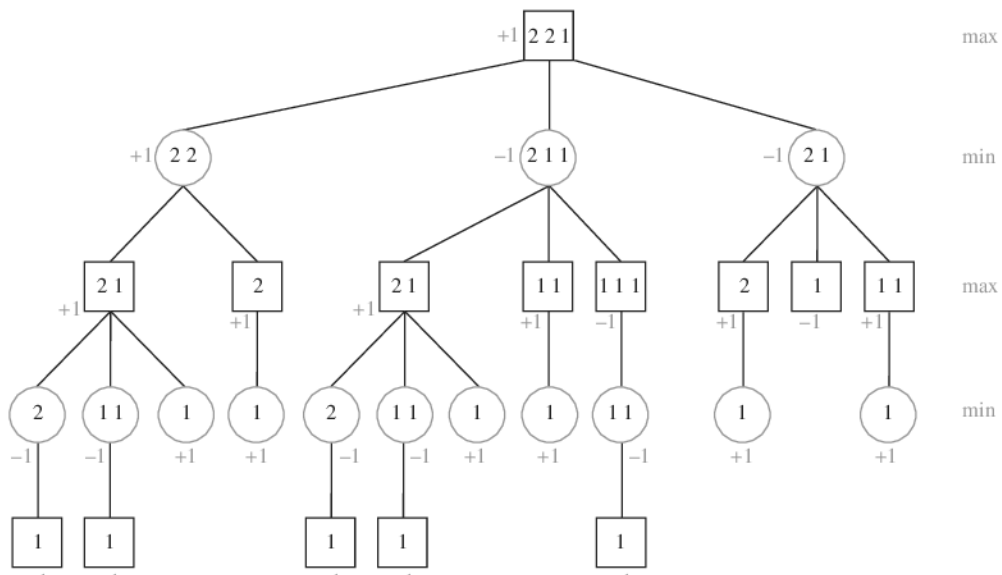


FIGURE 9 **Showing the Values of Vertices in the Game of Nim.**

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.

Links

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.

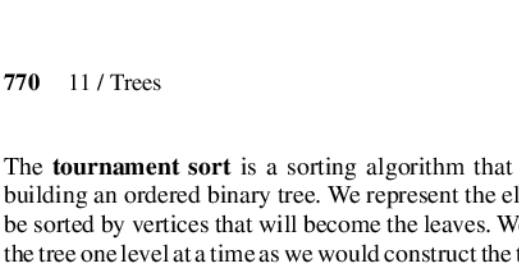
***10.** One of four coins may be counterfeit. If it is counterfeit, it may be lighter or heavier than the others. How many weighings are needed, using a balance scale, to determine whether there is a counterfeit coin, and if there is, whether it is lighter or heavier than the others? Describe an algorithm 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.

(a)



(b)

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.

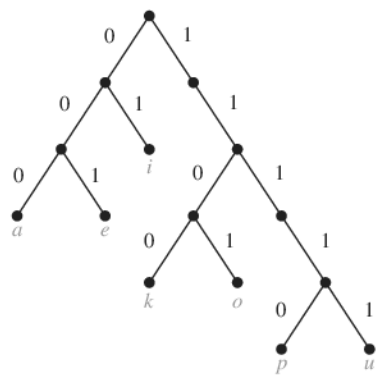
17. How many comparisons does the tournament sort use to find the second largest, the third largest, and so on, up to the $(n-1)$st largest (or second smallest) element?

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.

Letter	*Frequency*	*Letter*	*Frequency*
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.

a) Construct a Huffman code for these three symbols.

b) 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.

c) 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 $x_1, x_2, \ldots, x_n, x_{n+1}$ appearing 1, $f_1, f_2, \ldots, f_n$ times in a symbol string, respectively, where f_j is the jth 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

a) two piles with one and three stones, respectively.

b) two piles with two and four stones, respectively.

c) 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.

a)

O	X	X
X	O	O
		X

b)

X	O	X
O	X	X
		O

c)

X		O
O	O	
X		X

d)

	O	X
	X	O
X	O	

38. Suppose that the first four moves of a tic-tac-toe game are as shown. Does the first player (whose moves are marked by Xs) have a strategy that will always win?

a)

X	O	X
	O	

b)

X	O	X
	O	

c)

	O	X
	X	
O		

d)

X		
X	O	
O		

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

a) piles with four and five stones, respectively.

b) piles with two, three, and four stones, respectively.

c) piles with one, two, three, and four stones, respectively.

d) piles with two, two, three, three, and five 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

Links

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 $1, 2, 3, \ldots, k$.
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 $A.1, A.2, \ldots, A.k_v$.

Following this procedure, a vertex v at level n, for $n \geq 1$, is labeled $x_1.x_2.\ldots.x_n$, where the unique path from the root to v goes through the x_1st vertex at level 1, the x_2nd 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 $x_1.x_2.\ldots.x_n$ is less than the vertex labeled $y_1.y_2.\ldots.y_m$ if there is an i, $0 \leq i \leq n$, with $x_1 = y_1, x_2 = y_2, \ldots, x_{i-1} = y_{i-1}$, and $x_i < y_i$; or if $n < m$ and $x_i = y_i$ for $i = 1, 2, \ldots, n$.

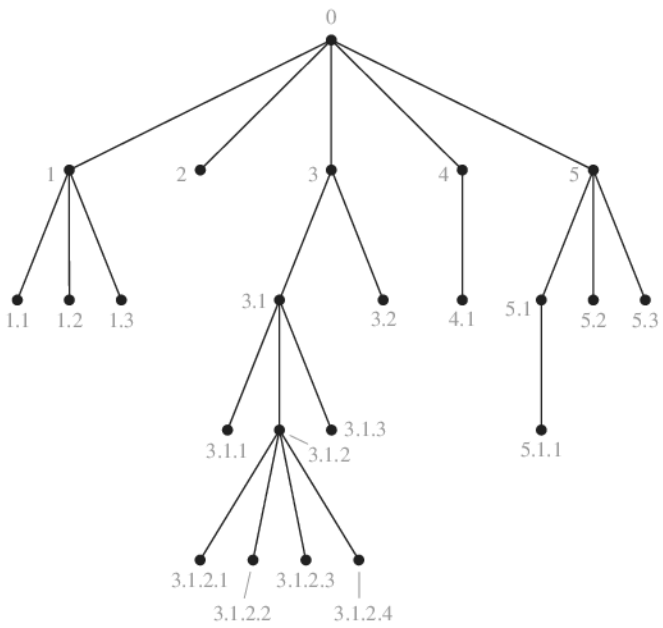


FIGURE 1 **The Universal Address System of an Ordered Rooted Tree.**

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

$$0 < 1 < 1.1 < 1.2 < 1.3 < 2 < 3 < 3.1 < 3.1.1 < 3.1.2 < 3.1.2.1 < 3.1.2.2 < 3.1.2.3 < 3.1.2.4 < 3.1.3 < 3.2 < 4 < 4.1 < 5 < 5.1 < 5.1.1 < 5.2 < 5.3$$ ◀

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 $T_1, T_2, \ldots, T_n$ 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.

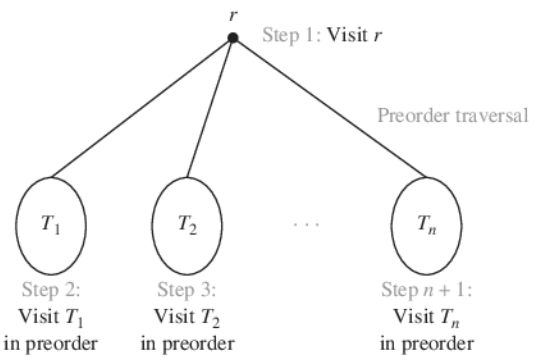


FIGURE 2 Preorder Traversal.

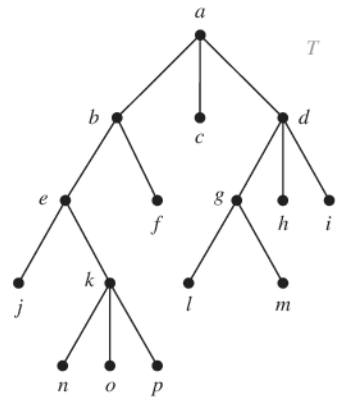


FIGURE 3 The Ordered Rooted Tree *T*.

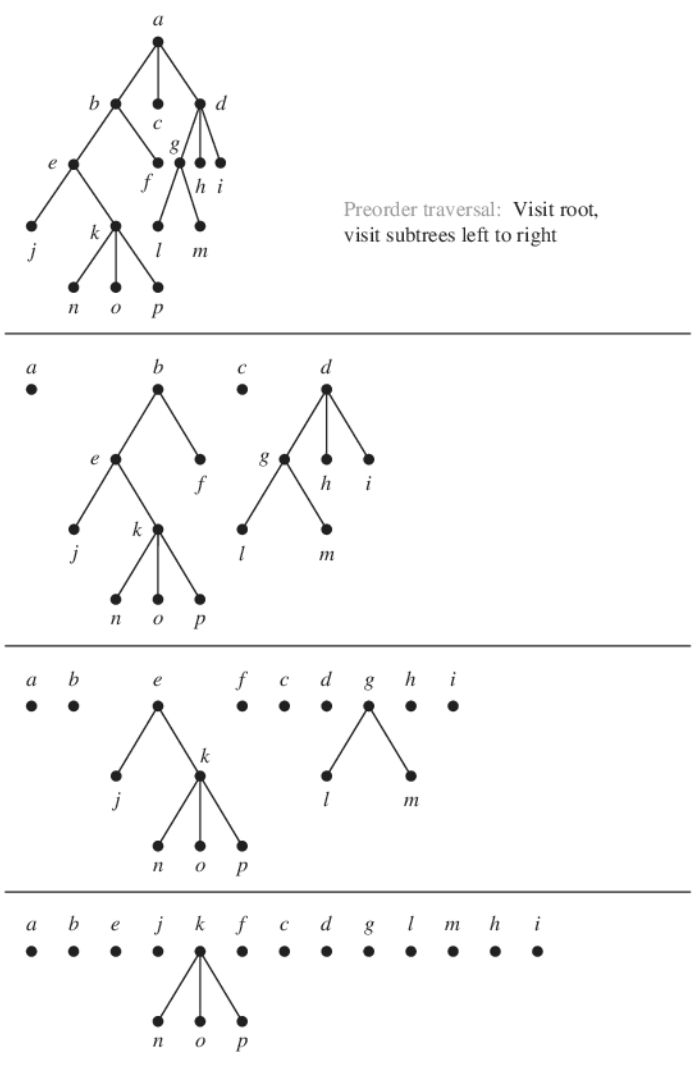


FIGURE 4 The Preorder Traversal of *T*.

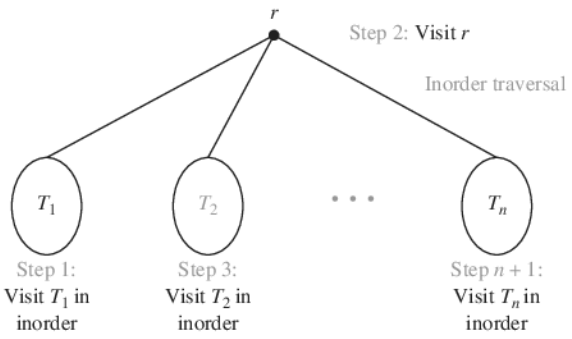


FIGURE 5 **Inorder Traversal.**

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 $a, b, e, j, k, n, o, p, f, c, d, g, l, m, h, i$. ◀

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 $T_1, T_2, \ldots, T_n$ 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, ..., 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?

Extra Examples

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 $j, e, n, k, o, p, b, f, a, c, l, g, m, d, h, i$. ◀

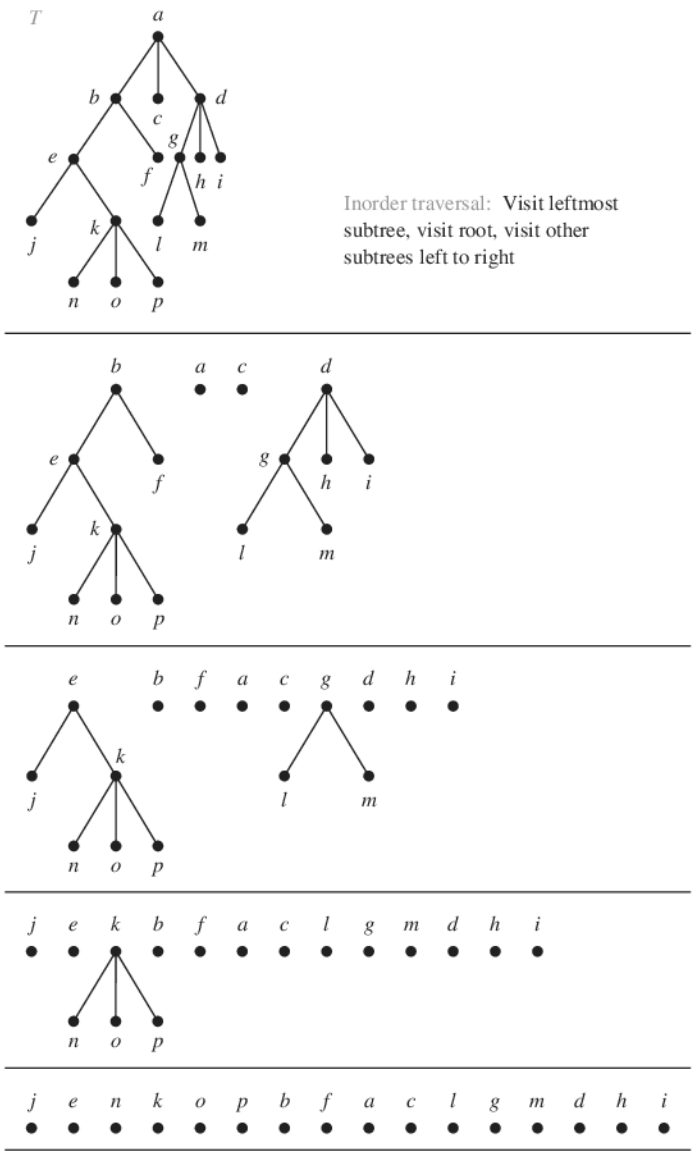


FIGURE 6 **The Inorder Traversal of T.**

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 $T_1, T_2, \ldots, T_n$ are the subtrees at r from left to right. The *postorder traversal* begins by traversing T_1 in postorder, then T_2 in postorder, ..., 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.

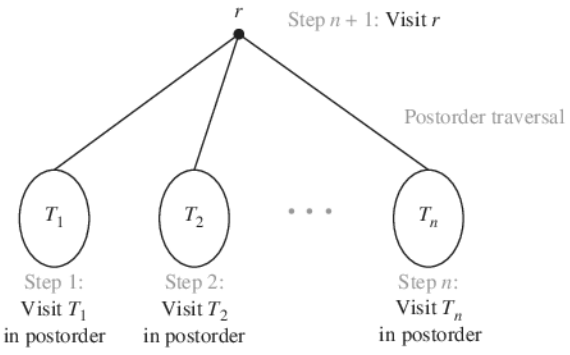


FIGURE 7 Postorder Traversal.

EXAMPLE 4 In which order does a postorder traversal visit the vertices of the ordered rooted tree T shown in Figure 3?

Extra Examples

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($T(c)$)

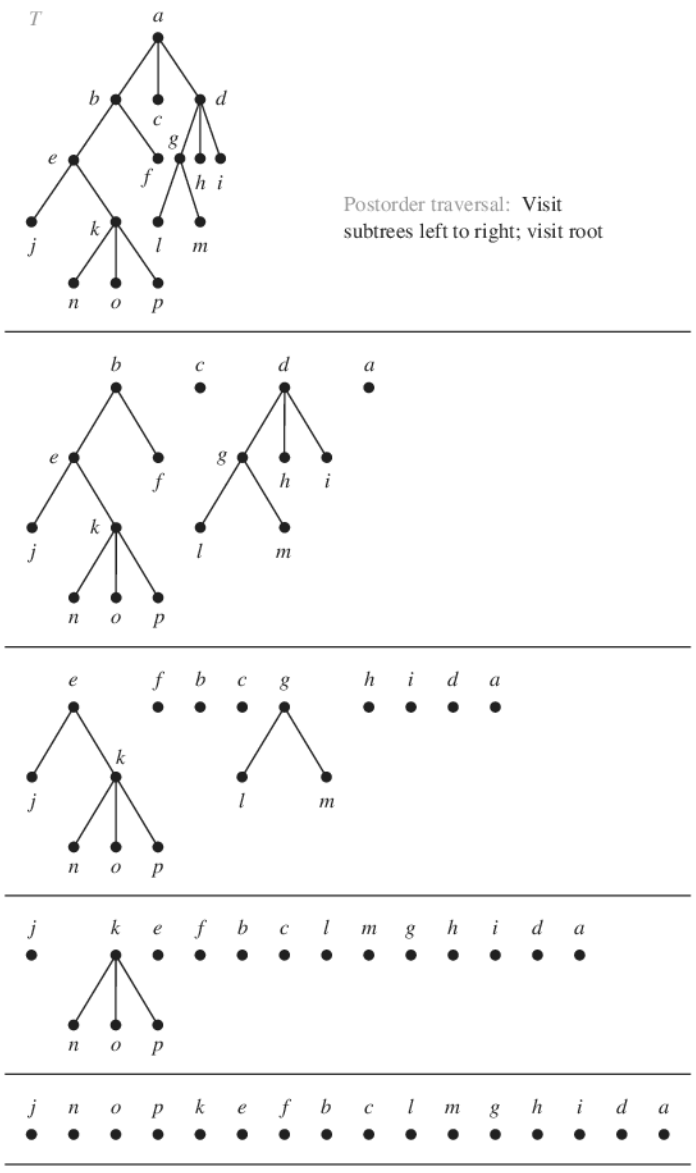


FIGURE 8 **The Postorder Traversal of T.**

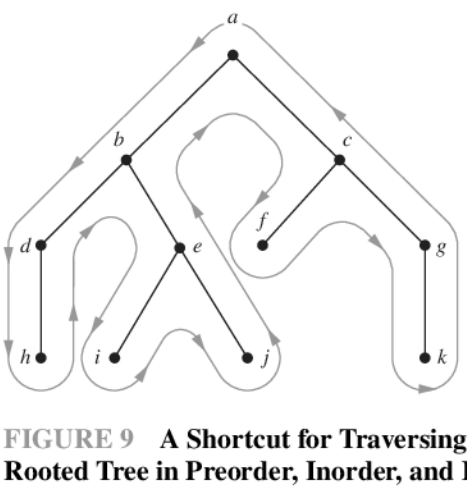


FIGURE 9 **A Shortcut for Traversing an Ordered Rooted Tree in Preorder, Inorder, and Postorder.**

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 $(x + y) \uparrow 2$