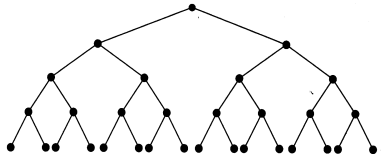
u to v in G, a contradiction. **b)** Suppose $G = (V, E)$. Let $a, b \in V$. We must show that the distance between a and b in $\overline{G}$ does not exceed 3. If $\{a, b\} \notin E$, the result follows, so assume that $\{a, b\} \in E$. Because the diameter of G is greater than or equal to 3, there exist vertices u and v such that the distance in G between u and v is greater than or equal to 3. Either u or v, or both, is not in the set $\{a, b\}$. Assume u is different from both a and b. Either $\{a, u\} \in E$ or $\{b, u\} \in E$; otherwise a, u, b is a path of length 2 in $\overline{G}$. So, without loss of generality, assume $\{a, u\} \in E$. Thus v is different from a and from b. If $\{a, v\} \in E$, then u, a, v is a path of length 2 in G, so $\{a, v\} \notin E$ and thus $\{b, v\} \in E$ (or else there would be a path a, v, b of length 2 in $\overline{G}$). Hence, $\{u, b\} \notin E$; otherwise u, b, v is a path of length 2 in G. Thus, a, v, u, b is a path of length 3 in $\overline{G}$, as desired. **41.** a, b, e, z **43.** a, c, b, d, e, z **45.** If G is planar, then because $e \leq 3v - 6$, G has at most 27 edges. (If G is not connected it has even fewer edges.) Similarly, $\overline{G}$ has at most 27 edges. But the union of G and $\overline{G}$ is K_{11}, which has 55 edges, and $55 > 27 + 27$. **47.** Suppose that G is colored with k colors and has independence number i. Because each color class must be an independent set, each color class has no more than i elements. Thus there are at most ki vertices. **49. a)** By Theorem 2 of Section 6.2, the probability of selecting exactly m edges is $C(n, m)p^m(1 - p)^{n-m}$. **b)** By Theorem 2 in Section 6.4, the expected value is np. **c)** To generate a labeled graph G, as we apply the process to pairs of vertices, the random number x chosen must be less than or equal to $1/2$ when G has an edge between that pair of vertices and greater than $1/2$ when G has no edge there. Hence, the probability of making the correct choice is $1/2$ for each edge and $1/2^{C(n,2)}$ overall. Hence, all labeled graphs are equally likely. **51.** Suppose P is monotone increasing. If the property of not having P were not retained whenever edges are removed from a simple graph, there would be a simple graph G not having P and another simple graph G' with the same vertices but with some of the edges of G missing that has P. But P is monotone increasing, so because G' has P, so does G obtained by adding edges to G'. This is a contradiction. The proof of the converse is similar.

CHAPTER 10

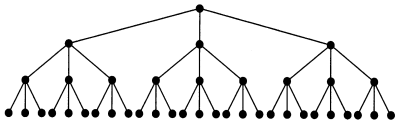
Section 10.1

1. (a), (c), (e) **3. a)** a **b)** $a, b, c, d, f, h, j, q, t$ **c)** $e, g, i, k, l, m, n, o, p, r, s, u$ **d)** q, r **e)** c **f)** p **g)** f, b, a **h)** e, f, l, m, n **5.** No **7.** Level 0: a; level 1: b, c, d; level 2: e through k (in alphabetical order); level 3: l through r; level 4: s, t; level 5: u **9. a)** The entire tree **b)** c, g, h, o, p and the four edges cg, ch, ho, hp **c)** e alone **11. a)** 1 **b)** 2 **13. a)** 3 **b)** 9 **15.** The "only if" part is Theorem 2 and the definition of a tree. Suppose G is a connected simple graph with n vertices and $n - 1$ edges. If G is not a tree, it contains, by Exercise 14, an edge whose removal produces a graph G', which is still connected. If G' is not a tree, remove an edge to produce a connected graph G''. Repeat this procedure until the result is a tree. This requires at most $n - 1$ steps because there are only $n - 1$ edges. By Theorem 2, the resulting graph has $n - 1$ edges because it has n vertices. It follows that no edges were deleted, so G was already a tree. **17.** 9999 **19.** 2000 **21.** 999 **23.** 1,000,000 dollars **25.** No such tree exists by Theorem 4 because it is impossible for $m = 2$ or $m = 84$.

27. Complete binary tree of height 4:



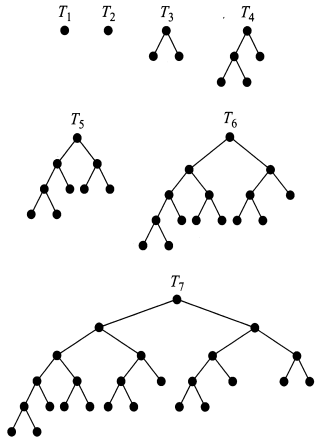
Complete 3-ary tree of height 3:



29. a) By Theorem 3 it follows that $n = mi + 1$. Because $i + l = n$, we have $l = n - i$, so $l = (mi + 1) - i = (m - 1)i + 1$. **b)** We have $n = mi + 1$ and $i + l = n$. Hence, $i = n - l$. It follows that $n = m(n - l) + 1$. Solving for n gives $n = (ml - 1)/(m - 1)$. From $i = n - l$ we obtain $i = [(ml - 1)/(m - 1)] - l = (l - 1)/(m - 1)$. **31.** $n - t$ **33. a)** 1 **b)** 3 **c)** 5 **35. a)** The parent directory **b)** A subdirectory or contained file **c)** A subdirectory or contained file in the same parent directory **d)** All directories in the path name **e)** All subdirectories and files continued in the directory or a subdirectory of this directory, and so on **f)** The length of the path to this directory or file **g)** The depth of the system, i.e., the length of the longest path **37.** Let $n = 2^k$, where k is a positive integer. If $k = 1$, there is nothing to prove because we can add two numbers with $n - 1 = 1$ processor in $\log 2 = 1$ step. Assume we can add $n = 2^k$ numbers in $\log n$ steps using a tree-connected network of $n - 1$ processors. Let $x_1, x_2, \ldots, x_{2n}$ be $2n = 2^{k+1}$ numbers that we wish to add. The tree-connected network of $2n - 1$ processors consists of the tree-connected network of $n - 1$ processors together with two new processors as children of each leaf. In one step we can use the leaves of the larger network to find $x_1 + x_2, x_3 + x_4, \ldots, x_{2n-1} + x_{2n}$, giving us n numbers, which, by the inductive hypothesis, we can add in $\log n$ steps using the rest of the network. Because we have used $\log n + 1$ steps and $\log(2n) = \log 2 + \log n = 1 + \log n$, this completes the proof. **39.** c only **41.** c and h **43.** Suppose a tree T has at least two centers. Let u and v be distinct centers, both with eccentricity e, with u and v not adjacent. Because T is connected, there is a simple path P from u to v. Let c be any other vertex on this path. Because the eccentricity of c is at least e, there is a vertex w such that the unique simple path from c to w has length at least e. Clearly, this path cannot contain both u and v or else there would be a simple circuit. In fact, this path from c to w leaves P and does not return to P once it, possibly, follows part of P toward

either u or v. Without loss of generality, assume this path does not follow P toward u. Then the path from u to c to w is simple and of length more than e, a contradiction. Hence, u and v are adjacent. Now because any two centers are adjacent, if there were more than two centers, T would contain K_3, a simple circuit, as a subgraph, which is a contradiction.

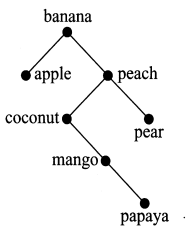
45.



47. The statement is that *every* tree with n vertices has a path of length $n - 1$, and it was shown only that there exists a tree with n vertices having a path of length $n - 1$.

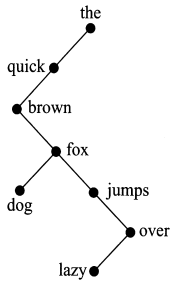
Section 10.2

1.

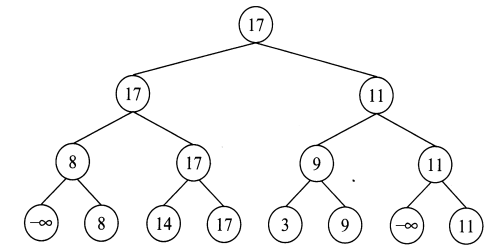


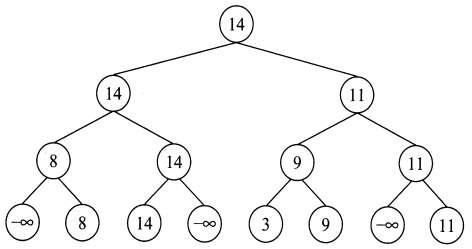
3. a) 3 **b)** 1 **c)** 4 **d)** 5

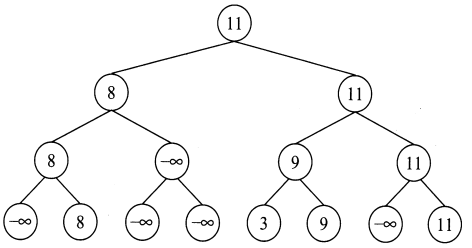
5.

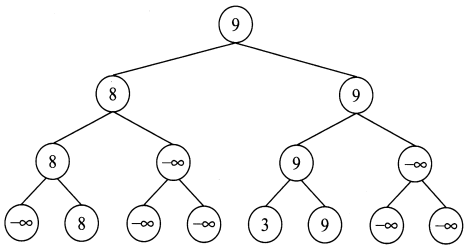


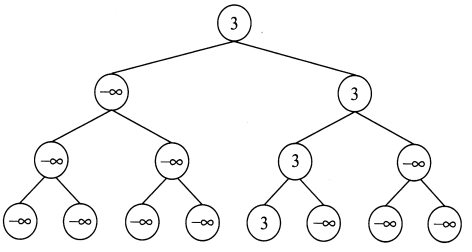
7. At least $\lceil \log_3 4 \rceil = 2$ weighings are needed, because there are only four outcomes (because it is not required to determine whether the coin is lighter or heavier). In fact, two weighings suffice. Begin by weighing coin 1 against coin 2. If they balance, weigh coin 1 against coin 3. If coin 1 and coin 3 are the same weight, coin 4 is the counterfeit coin, and if they are not the same weight, then coin 3 is the counterfeit coin. If coin 1 and coin 2 are not the same weight, again weigh coin 1 against coin 3. If they balance, coin 2 is the counterfeit coin; if they do not balance, coin 1 is the counterfeit coin. **9.** At least $\lceil \log_3 13 \rceil = 3$ weighings are needed. In fact, three weighings suffice. Start by putting coins 1, 2, and 3 on the left-hand side of the balance and coins 4, 5, and 6 on the right-hand side. If equal, apply Example 3 to coins 1, 2, 7, 8, 9, 10, 11, and 12. If unequal, apply Example 3 to 1, 2, 3, 4, 5, 6, 7, and 8. **11.** The least number is five. Call the elements a, b, c, and d. First compare a and b; then compare c and d. Without loss of generality, assume that $a < b$ and $c < d$. Next compare a and c. Whichever is smaller is the smallest element of the set. Again without loss of generality, suppose $a < c$. Finally, compare b with both c and d to completely determine the ordering. **13.** The first two steps are shown in the text. After 22 has been identified as the second largest element, we replace the leaf 22 by $-\infty$ in the tree and recalculate the winner in the path from the leaf where 22 used to be up to the root. Next, we see that 17 is the third largest element, so we repeat the process: replace the leaf 17 by $-\infty$ and recalculate. Next, we see that 14 is the fourth largest element, so we repeat the process: replace the leaf 14 by $-\infty$ and recalculate. Next, we see that 11 is the fifth largest element, so we repeat the process: replace the leaf 11 by $-\infty$ and recalculate. The process continues in this manner. We determine that 9 is the sixth largest element, 8 is the seventh largest element, and 3 is the eighth largest element. The trees produced in all steps, except the second to last, are shown here.







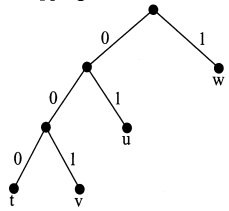




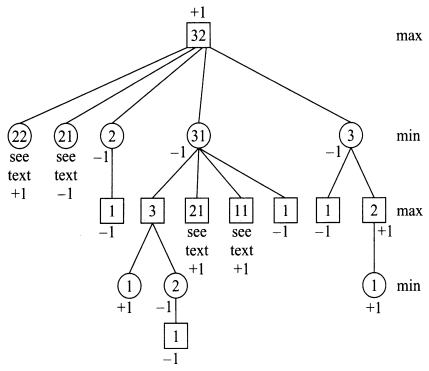
15. The value of a vertex is the list element currently there, and the label is the name (i.e., location) of the leaf responsible for that value.

procedure *tournament sort*($a_1, \ldots, a_n$)
$k := \lceil \log n \rceil$
build a binary tree of height k
for $i := 1$ **to** n
  set the value of the ith leaf to be a_i and its label to be itself
for $i := n + 1$ **to** 2^k
  set the value of the ith leaf to be $-\infty$ and its label to be itself
for $i := k - 1$ **downto** 0
  for each vertex v at level i
    set the value of v to the larger of the values of its children and its label to be the label of the child with the larger value
for $i := 1$ **to** n
begin
  $c_i :=$ value at the root
  let v be the label of the root
  set the value of v to be $-\infty$
  while the label at the root is still v
  begin
    $v := parent(v)$
    set the value of v to the larger of the values of its children and its label to be the label of the child with the larger value
  end
end {$c_1, \ldots, c_n$ is the list in nonincreasing order}

17. $k - 1$, where $n = 2^k$ **19. a)** Yes **b)** No **c)** Yes **d)** Yes **21.** *a*: 000, *e*: 001, *i*: 01, *k*: 1100, *o*: 1101, *p*: 11110, *u*: 11111 **23.** a: 11; b: 101; c: 100; d: 01; e: 00; 2.25 bits (Note: This coding depends on how ties are broken, but the average number of bits is always the same.) **25.** There are four possible answers in all, the one shown here and three more obtained from this one by swapping t and v and/or swapping u and w.

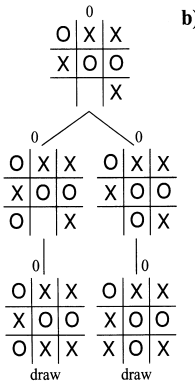


27. A:0001; B:101001; C:11001; D:00000; E:100; F:001100; G:001101; H:0101; I:0100; J:110100101; K:1101000; L:00001; M:10101; N:0110; O:0010; P:101000; Q:1101001000; R:1011; S:0111; T:111; U:00111; V:110101; W:11000; X:11010011; Y:11011; Z:1101001001 **29.** A:2; E:1; N:010; R:011; T:02; Z:00 **31.** n **33.** Because the tree is rather large, we have indicated in some places to "see text." Refer to Figure 9; the subtree rooted at these square or circle vertices is exactly the same as the corresponding subtree in Figure 9. First player wins.

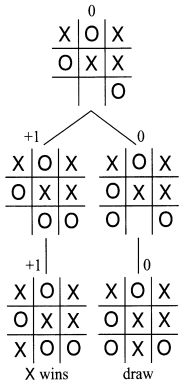


35. a) \$1 **b)** \$3 **c)** −\$3 **37.** See the figures shown next. **a)** 0 **b)** 0 **c)** 1 **d)** This position cannot have occurred in a game; this picture is impossible.

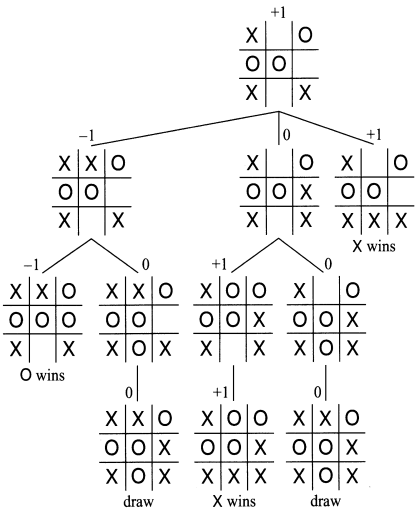
a)



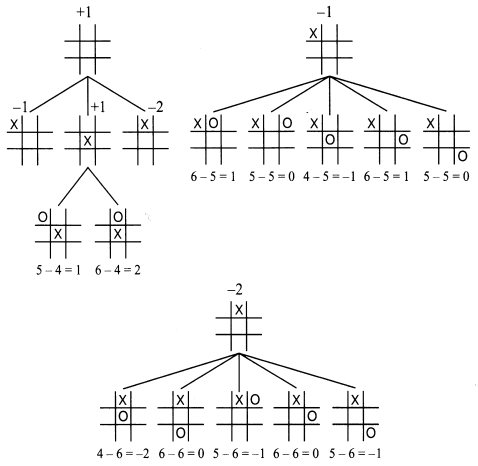
b)



c)

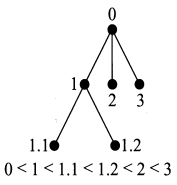


39. Proof by strong induction: *Basis step:* When there are $n = 2$ stones in each pile, if first player takes two stones from a pile, then second player takes one stone from the remaining pile and wins. If first player takes one stone from a pile, then second player takes two stones from the other pile and wins. *Inductive step:* Assume inductive hypothesis that second player can always win if the game starts with two piles of j stones for all $2 \leq j \leq k$, where $k \geq 2$, and consider a game with two piles containing $k + 1$ stones each. If first player takes all the stones from one of the piles, then second player takes all but one stone from the remaining pile and wins. If first player takes all but one stone from one of the piles, then second player takes all the stones from the other pile and wins. Otherwise first player leaves j stones in one pile, where $2 \leq j \leq k$, and $k + 1$ stones in the other pile. Second player takes the same number of stones from the larger pile, also leaving j stones there. At this point the game consists of two piles of j stones each. By the inductive hypothesis, the second player in that game, who is also the second player in our actual game, can win, and the proof by strong induction is complete. **41.** 7; 49 **43.** Value of tree is 1. Note: The second and third trees are the subtrees of the two children of the root in the first tree whose subtrees are not shown because of space limitations. They should be thought of as spliced into the first picture.

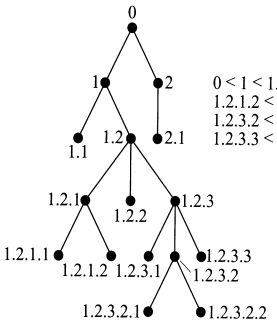


Section 10.3

1.



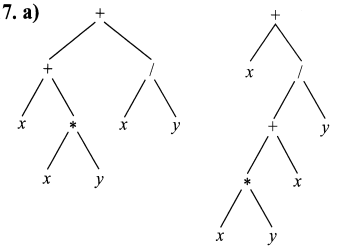
3.



$0 < 1 < 1.1 < 1.2 < 1.2.1 < 1.2.1.1 < 1.2.1.2 < 1.2.2 < 1.2.3 < 1.2.3.1 < 1.2.3.2 < 1.2.3.2.1 < 1.2.3.2.2 < 1.2.3.3 < 2 < 2.1$

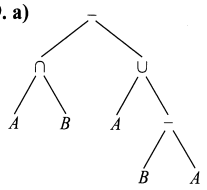
5. No **7.** a, b, d, e, f, g, c **9.** $a, b, e, k, l, m, f, g, n, r, s, c, d, h, o, i, j, p, q$ **11.** $d, b, i, e, m, j, n, o, a, f, c, g, k, h, p, l$ **13.** d, f, g, e, b, c, a **15.** $k, l, m, e, f, r, s, n, g, b, c, o, h, i, p, q, j, d, a$

17. a)



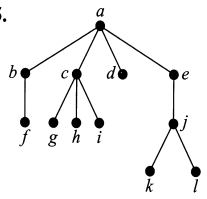
b) $++x*xy/xy, +x/+*xyxy$ **c)** $xxy*+xy/+, xxy*x+y/+$ **d)** $((x+(x*y))+(x/y)), (x+(((x*y)+x)/y))$

19. a)



b) $-\cap AB \cup A-BA$ **c)** $AB\cap AB A-\cup-$ **d)** $((A\cap B)-(A\cup(B-A)))$ **21.** 14 **23. a)** 1 **b)** 1 **c)** 4 **d)** 2205

25.

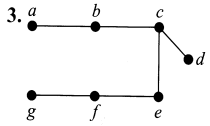


27. Use mathematical induction. The result is trivial for a list with one element. Assume the result is true for a list with n elements. For the inductive step, start at the end. Find the sequence of vertices at the end of the list starting with the last leaf, ending with the root, each vertex being the last child of the one following it. Remove this leaf and apply the inductive hypothesis. **29.** c, d, b, f, g, h, e, a in each case **31.** Proof by mathematical induction. Let $S(X)$ and $O(X)$ represent the number of symbols and number of operators in the well-formed formula X, respectively. The statement is true for well-formed formulae of length 1, because they have 1 symbol and 0 operators. Assume the statement is true for all well-formed formulae of length less than n. A well-formed formula of length n must be of the form $*XY$, where $*$ is an operator and X and Y are well-formed formulae of length less than n. Then by the inductive hypothesis $S(*XY) = S(X) + S(Y) = [O(X) + 1] + [O(Y) + 1] = O(X) + O(Y) + 2$. Because $O(*XY) = 1 + O(X) + O(Y)$, it follows that $S(*XY) = O(*XY) + 1$. **33.** $xy+zx\circ+x\circ, xyz++yx++, xyxy\circ\circ xy\circ\circ z\circ+, xz\times zz+\circ, yyyy\circ\circ\circ, zx+yz+\circ$, for instance

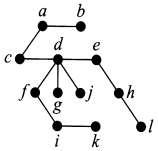
Section 10.4

1. $m - n + 1$

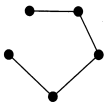
3.



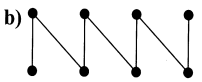
5.



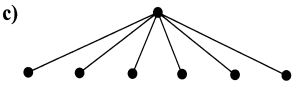
7. a)



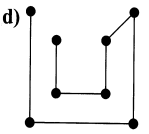
b)



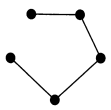
c)



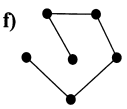
d)



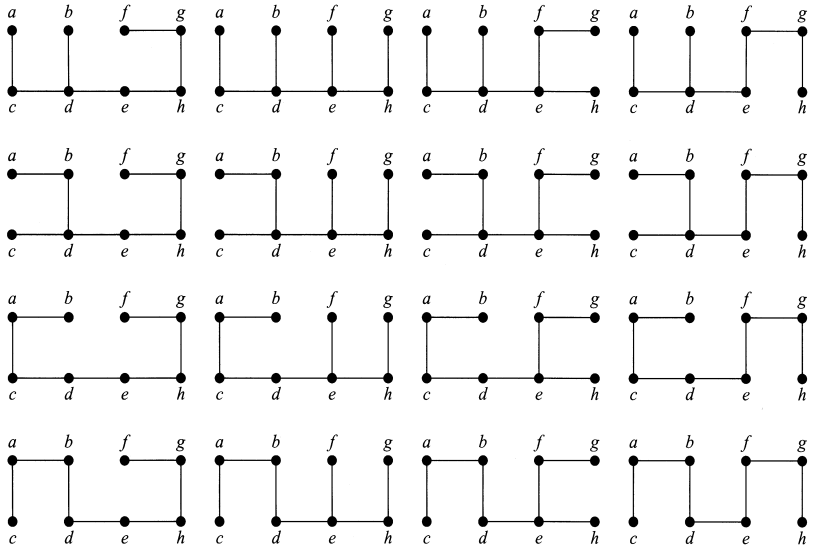
e)



f)

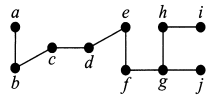


9.

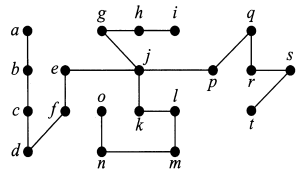


11. a) 3 **b)** 16 **c)** 4 **d)** 5

13.

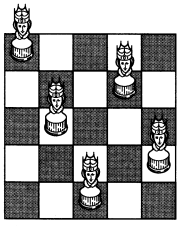


15.

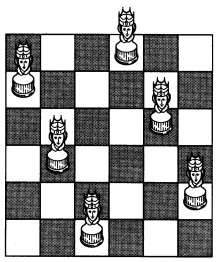


17. a) A path of length 6 **b)** A path of length 5 **c)** A path of length 6 **d)** Depends on order chosen to visit the vertices; may be a path of length 7 **19.** With breadth-first search, the initial vertex is the middle vertex, and the n spokes are added to the tree as this vertex is processed. Thus, the resulting tree is $K_{1,n}$. With depth-first search, we start at the vertex in the middle of the wheel and visit a neighbor—one of the vertices on the rim. From there we move to an adjacent vertex on the rim, and so on all the way around until we have reached every vertex. Thus, the resulting spanning tree is a path of length n. **21.** With breadth-first search, we fan out from a vertex of degree m to all the vertices of degree n as the first step. Next, a vertex of degree n is processed, and the edges from it to all the remaining vertices of degree m are added. The result is a $K_{1,n-1}$ and a $K_{1,m-1}$ with their centers joined by an edge. With depth-first search, we travel back and forth from one partite set to the other until we can go no further. If $m = n$ or $m = n - 1$, then we get a path of length $m + n - 1$. Otherwise, the path ends while some vertices in the larger partite set have not been visited, so we back up one link in the path to a vertex v and then successively visit the remaining vertices in that set from v. The result is a path with extra pendant edges coming out of one end of the path. **23.** A possible set of flights to discontinue are: Boston–New York, Detroit–Boston, Boston–Washington, New York–Washington, New York–Chicago, Atlanta–Washington, Atlanta–Dallas, Atlanta–Los Angeles, Atlanta–St. Louis, St. Louis–Dallas, St. Louis–Detroit, St. Louis–Denver, Dallas–San Diego, Dallas–Los Angeles, Dallas–San Francisco, San Diego–Los Angeles, Los Angeles–San Francisco, San Francisco–Seattle. **25.** Trees **27.** Proof by induction on the length of the path: If the path has length 0, then the result is trivial. If the length is 1, then u is adjacent to v, so u is at level 1 in the breadth-first spanning tree. Assume that the result is true for paths of length l. If the length of a path is $l + 1$, let u' be the next-to-last vertex in a shortest path from v to u. By the inductive hypothesis, u' is at level l in the breadth-first spanning tree. If u were at a level not exceeding l, then clearly the length of the shortest path from v to u would also not exceed l. So u has not been added to the breadth-first spanning tree yet after the vertices of level l have been added. Because u is adjacent to u', it will be added at level $l + 1$ (although the edge connecting u' and u is not necessarily added). **29. a)** No solution

b)

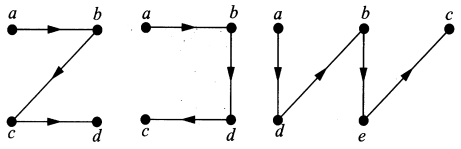


c)

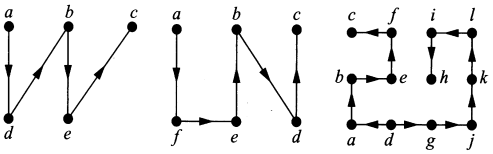


31. Start at a vertex and proceed along a path without repeating vertices as long as possible, allowing the return to the start after all vertices have been visited. When it is impossible to continue along a path, backtrack and try another extension of the current path. **33.** Take the union of the spanning trees of the connected components of G. They are disjoint, so the result is a forest. **35.** $m - n + c$ **37.** Use depth-first search on each component. **39.** If an edge uv is not followed while we are processing vertex u during the depth-first search process, then it must be the case that the vertex v had already been visited. There are two cases. If vertex v was visited after we started processing u, then, because we are not finished processing u yet, v must appear in the subtree rooted at u (and hence, must be a descendant of u). On the other hand, if the processing of v had already begun before we started processing u, then why wasn't this edge followed at that time? It must be that we had not finished processing v, in other words, that we are still forming the subtree rooted at v, so u is a descendant of v, and hence, v is an ancestor of u. **41.** Certainly these two procedures produce the identical spanning trees if the graph we are working with is a tree itself, because in this case there is only one spanning tree (the whole graph). This is the only case in which that happens, however. If the original graph has any other edges, then by Exercise 39 they must be back edges and hence, join a vertex to an ancestor or descendant, whereas by Exercise 40, they must connect vertices at the same level or at levels that differ by 1. Clearly these two possibilities are mutually exclusive. Therefore there can be no edges other than tree edges if the two spanning trees are to be the same. **43.** Because the edges not in the spanning tree are not followed in the process, we can ignore them. Thus we can assume that the graph was a rooted tree to begin with. The basis step is trivial (there is only one vertex), so we assume the inductive hypothesis that breadth-first search applied to trees with n vertices have their vertices visited in order of their level in the tree and consider a tree T with $n + 1$ vertices. The last vertex to be visited during breadth-first search of this tree, say v, is the one that was added last to the list of vertices waiting to be processed. It was added when its parent, say u, was being processed. We must show that v is at the lowest (bottom-most, i.e., numerically greatest) level of the tree. Suppose not; say vertex x, whose parent is vertex w, is at a lower level. Then w is at a lower level than u. Clearly v must be a leaf, because any child of v could not have been seen before v is seen. Consider the tree T' obtained from T by deleting v. By the inductive hypothesis, the vertices in T' must be processed in order of their level in T' (which is the same as their level in T, and the absence of v in T' has no effect on the rest of the algorithm). Therefore u must have been processed before w, and therefore v would have joined the waiting list before x did, a contradiction. Therefore v is at the bottom-most level of the tree, and the proof is complete. **45.** We modify the pseudocode given in Algorithm 2 by initializing m to be 0 at the beginning of the algorithm, and adding the statements "$m := m + 1$" and "assign m to vertex v" after the statement that removes vertex v from L. **47.** If a directed edge uv is not followed while we are processing its tail u during the depth-first search process, then it must be the case that its head v had already been visited. There are three cases. If vertex v was visited after we started processing u, then, because we are not finished processing u yet, v must appear in the subtree rooted at u (and hence, must be a descendant of u), so we have a forward edge. Otherwise, the processing of v must have already begun before we started processing u. If it had not yet finished (i.e., we are still forming the subtree rooted at v), then u is a descendant of v, and hence, v is an ancestor of u (we have a back edge). Finally, if the processing of v had already finished, then by definition we have a cross edge. **49.** Let T be the spanning tree constructed in Figure 3 and T_1, T_2, T_3, and T_4 the spanning trees in Figure 4. Denote by $d(T', T'')$ the distance between T' and T''. Then $d(T, T_1) = 6$, $d(T, T_2) = 4$, $d(T, T_3) = 4$, $d(T, T_4) = 2$, $d(T_1, T_2) = 4$, $d(T_1, T_3) = 4$, $d(T_1, T_4) = 6$, $d(T_2, T_3) = 4$, $d(T_2, T_4) = 2$, and $d(T_3, T_4) = 4$. **51.** Suppose $e_1 = \{u, v\}$ is as specified. Then $T_2 \cup \{e_1\}$ contains a simple circuit C containing e_1. The graph $T_1 - \{e_1\}$ has two connected components; the endpoints of e_1 are in different components. Travel C from u in the direction opposite to e_1 until you come to the first vertex in the same component as v. The edge just crossed is e_2. Clearly, $T_2 \cup \{e_1\} - \{e_2\}$ is a tree, because e_2 was on C. Also $T_1 - \{e_1\} \cup \{e_2\}$ is a tree, because e_2 reunited the two components.

53. Exercise 18: Exercise 19: Exercise 20:



Exercise 21: Exercise 22: Exercise 23:

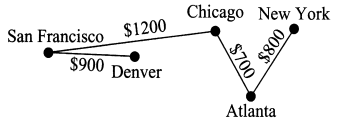


55. First construct an Euler circuit in the directed graph. Then delete from this circuit every edge that goes to a vertex previously visited. **57.** According to Exercise 56, a directed graph contains a circuit if and only if there are any back edges. We can detect back edges as follows. Add a marker on each vertex v to indicate what its status is: not yet seen (the initial situation), seen (i.e., put into T) but not yet finished (i.e., *visit*(v) has not yet terminated), or finished (i.e., *visit*(v) has terminated). A few extra lines in Algorithm 1 will accomplish this bookkeeping. Then to determine whether a directed graph has a circuit, we just have to check when looking at edge uv whether the status of v is "seen." If that ever happens, then we know there is a circuit; if not, then there is no circuit.

Section 10.5

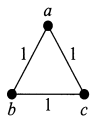
1. Deep Springs–Oasis, Oasis–Dyer, Oasis–Silverspeak, Silverspeak–Goldfield, Lida–Gold Point, Gold Point–Beatty, Lida–Goldfield, Goldfield–Tonopah, Tonopah–Manhattan, Tonopah–Warm Springs **3.** $\{e, f\}$, $\{c, f\}$, $\{e, h\}$, $\{h, i\}$, $\{b, c\}$, $\{b, d\}$, $\{a, d\}$, $\{g, h\}$

5.



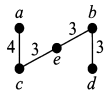
7. $\{e, f\}$, $\{a, d\}$, $\{h, i\}$, $\{b, d\}$, $\{c, f\}$, $\{e, h\}$, $\{b, c\}$, $\{g, h\}$

9.

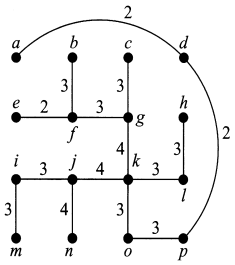


11. Instead of choosing minimum-weight edges at each stage, choose maximum-weight edges at each stage with the same properties.

13.

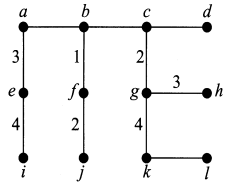


15.



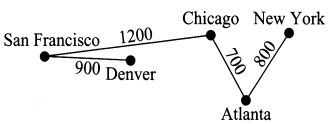
17. First find a minimum spanning tree T of the graph G with n edges. Then for $i = 1$ to $n - 1$, delete only the ith edge of T from G and find a minimum spanning tree of the remaining graph. Pick the one of these $n - 1$ trees with the shortest length. **19.** If all edges have different weights, then a contradiction is obtained in the proof that Prim's algorithm works when an edge e_{k+1} is added to T and an edge e is deleted, instead of possibly producing another spanning tree.

21.

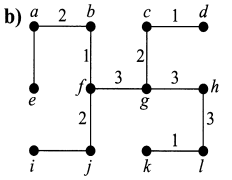


23. Same as Kruskal's algorithm, except start with $T :=$ this set of edges and iterate from $i = 1$ to $i = n - 1 - s$, where s is the number of edges you start with.

25. a)



b)

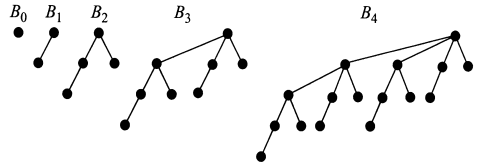


27. By Exercise 24, at each stage of Sollin's algorithm a forest results. Hence, after $n - 1$ edges are chosen, a tree results. It remains to show that this tree is a minimum spanning tree. Let T be a minimum spanning tree with as many edges in common with Sollin's tree S as possible. If $T \neq S$, then there is an edge $e \in S - T$ added at some stage in the algorithm, where prior to that stage all edges in S are also in T. $T \cup \{e\}$ contains a unique simple circuit. Find an edge $e' \in S - T$ and an edge $e'' \in T - S$ on this circuit and "adjacent" when viewing the trees of this stage as "supervertices." Then by the algorithm, $w(e') \leq w(e'')$. So replace T by $T - \{e''\} \cup \{e'\}$ to produce a minimum spanning tree closer to S than T was. **29.** Each of the r trees is joined to at least one other tree by a new edge. Hence, there are at most $r/2$ trees in the result (each new tree contains two or more old trees). To accomplish this, we need to add $r - (r/2) = r/2$ edges. Because the number of edges added is integral, it is at least $\lceil r/2 \rceil$. **31.** If $k \geq \log n$, then $n/2^k \leq 1$, so $\lceil n/2^k \rceil = 1$, so by Exercise 30 the algorithm is finished after at most $\log n$ iterations.

Supplementary Exercises

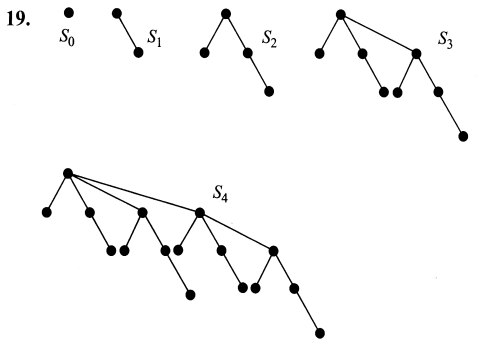
1. Suppose T is a tree. Then clearly T has no simple circuits. If we add an edge e connecting two nonadjacent vertices u and v, then obviously a simple circuit is formed, because when e is added to T the resulting graph has too many edges to be a tree. The only simple circuit formed is made up of the edge e together with the unique path P in T from v to u. Suppose T satisfies the given conditions. All that is needed is to show that T is connected, because there are no simple circuits in the graph. Assume that T is not connected. Then let u and v be in separate connected components. Adding $e = \{u, v\}$ does not satisfy the conditions. **3.** Suppose that a tree T has n vertices of degrees $d_1, d_2, \ldots, d_n$, respectively. Because $2e = \sum_{i=1}^{n} d_i$ and $e = n - 1$, we have $2(n - 1) = \sum_{i=1}^{n} d_i$. Because each $d_i \geq 1$, it follows that $2(n - 1) = n + \sum_{i=1}^{n}(d_i - 1)$, or that $n - 2 = \sum_{i=1}^{n}(d_i - 1)$. Hence, at most $n - 2$ of the terms of this sum can be 1 or more. Hence, at least two of them are 0. It follows that $d_i = 1$ for at least two values of i. **5.** $2n - 2$ **7.** T has no circuits, so it cannot have a subgraph homeomorphic to $K_{3,3}$ or K_5. **9.** Color each connected component separately. For each of these connected components, first root the tree, then color all vertices at even levels red and all vertices at odd levels blue. **11.** Upper bound: k^h; lower bound: $2\lceil k/2 \rceil^{h-1}$

13.



15. Because B_{k+1} is formed from two copies of B_k, one shifted down one level, the height increases by 1 as k increases by 1. Because B_0 had height 0, it follows by induction that B_k has height k. **17.** Because the root of B_{k+1} is the root of B_k with one additional child (namely the root of the other B_k), the degree of the root increases by 1 as k increases by 1. Because B_0 had a root with degree 0, it follows by induction that B_k has a root with degree k.

19.

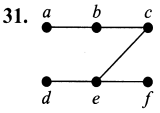


21. Use mathematical induction. The result is trivial for $k = 0$. Suppose it is true for $k - 1$. T_{k-1} is the parent tree for T. By induction, the child tree for T can be obtained from $T_0, \ldots, T_{k-2}$ in the manner stated. The final connection of r_{k-2} to r_{k-1} is as stated in the definition of S_k-tree.

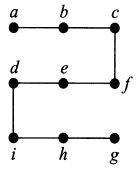
23. **procedure** *level*(T: ordered rooted tree with root r)
queue := sequence consisting of just the root r
while *queue* contains at least one term
begin
v := first vertex in queue
list v
remove v from queue and put children of v onto the end of queue
end

25. Build the tree by inserting a root for the address 0, and then inserting a subtree for each vertex labeled i, for i a positive integer, built up from subtrees for each vertex labeled $i.j$ for j a positive integer, and so on. **27. a)** Yes **b)** No **c)** Yes **29.** The resulting graph has no edge that is in more than one simple circuit of the type described. Hence, it is a cactus.

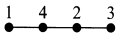
31.



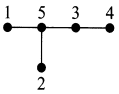
33.



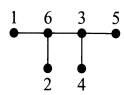
35. a)



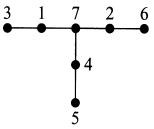
b)



c)



d)



37. 6 **39. a)** 0 for 00, 11 for 01, 100 for 10, 101 for 11 (exact coding depends on how ties were broken, but all versions are equivalent); $0.645n$ for string of length n **b)** 0 for 000, 100 for 001, 101 for 010, 110 for 100, 11100 for 011, 11101 for 101, 11110 for 110, 11111 for 111 (exact coding depends on how ties were broken, but all versions are equivalent); $0.532\overline{6}n$ for string of length n **41.** Let G' be the graph obtained by deleting from G the vertex v and all edges incident to v. A minimum spanning tree of G can be obtained by taking an edge of minimal weight incident to v together with a minimum spanning tree of G'. **43.** Suppose that edge e is the edge of least weight incident to vertex v, and suppose that T is a spanning tree that does not include e. Add e to T, and delete from the simple circuit formed thereby the other edge of the circuit that contains v. The result will be a spanning tree of strictly smaller weight (because the deleted edge has weight greater than the weight of e). This is a contradiction, so T must include e.

CHAPTER 11

Section 11.1

1. a) 1 **b)** 1 **c)** 0 **d)** 0 **3. a)** $(1 \cdot 1) + (\overline{0 \cdot 1} + 0) = 1 + (\overline{0} + 0) = 1 + (1 + 0) = 1 + 1 = 1$
b) $(\mathbf{T} \wedge \mathbf{T}) \vee (\neg(\mathbf{F} \wedge \mathbf{T}) \vee \mathbf{F}) \equiv \mathbf{T}$

5. a)

x	y	z	$\bar{x}y$
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	1
0	0	1	0
0	0	0	0

b)

x	y	z	$x + yz$
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

c)

x	y	z	$x\bar{y} + \overline{xyz}$
1	1	1	0
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	1
0	1	0	1
0	0	1	1
0	0	0	1

d)

x	y	z	$x(yz + \bar{y}\bar{z})$
1	1	1	1
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0