5

GRAPH THEORY

5:1 BUILDING THE LAN

A college's minicomputers, terminals, and microcomputers are joined in a **Local Area Network** (**LAN** for short). The advantages include the potential to connect terminals and microcomputers with any minicomputer, the capacity to support more terminals on campus, and rapid transmission of data between terminals and minicomputers. The first step in the installation of the LAN was to link the basement of every building on campus using coaxial cable. This does not mean that each pair of buildings is joined by a cable. What it does mean is that it is possible to send electrical signals via the cable from any building on campus to any other building perhaps using one or more intermediate buildings. How should we, or the LAN designers, decide which pairs of buildings to join directly by coaxial cable?

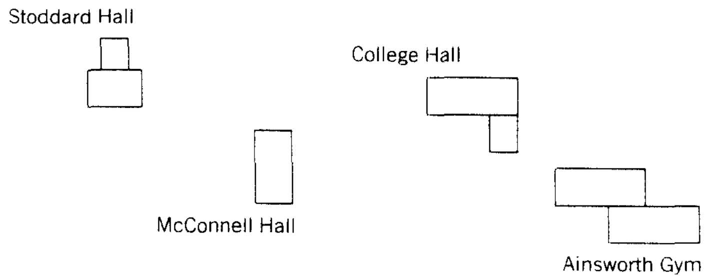


Figure 5.1 Campus map.

Example 1.1. Suppose that we want to connect College Hall (C), Ainsworth Gymnasium (A), and McConnell Hall (M) by coaxial cable. Which pairs of buildings should be directly connected? See Figure 5.1. If we connect C directly to M and M directly to A, then there is no need to connect C directly to A, since it is already possible to send electronic signals from C to A through M. There are three possible direct connections to make. Choosing any two makes the third unnecessary.

Question 1.1. Suppose that Ainsworth Gym (A), College Hall (C), McConnell Hall (M), and Stoddard Hall (S) are buildings on campus to be connected by the LAN. See Figure 5.1. How many pairs of possible direct connections are there among the four buildings? How many direct connections do you need to install so that communication (though not necessarily direct) is possible between each pair of buildings? Will any set of this many direct connections work? Answer the same questions if five buildings are to be joined.

In Example 1.1 why should any particular pair of connections be selected in preference to any other pair? In practice, such decisions are made to minimize the total installation cost.

Example 1.2. Suppose that it costs \$85,000 to install a cable between McConnell Hall and College Hall, \$78,000 to install a cable between Ainsworth and McConnell, and \$87,000 to install a cable between Ainsworth and College Hall. Which two direct cable links have minimum cost? The MC and AM links have a total cost of \$163,000; the MC and AC links have a total cost of \$172,000; and the AC and AM links have a total cost of \$165,000.

Question 1.2. Suppose that the cost of joining S with C is \$30,000, the cost of joining S with M is \$51,000, and the cost of joining S with A is \$67,000. Using the data from Example 1.2, determine which pairs of A, C, M, and S should be directly linked in order to minimize the total cost. Compare your answer with that of the preceding example.

What should be evident is that if there are only a few buildings, then these calculations can be carried out by hand and the minimum cost plan can be found. If the number of buildings to be linked is at all large, then we need a good algorithm to figure out how to do it. In the next two sections we provide the mathematical framework with which to consider this kind of problem. Then we contrast bad and good algorithms to find these special pairs of buildings.

EXERCISES FOR SECTION 1

1. Neilson Library (N) is to be included on the LAN with McConnell, Ainsworth, Stoddard, and College Hall. The cost to join Neilson with the other buildings is given by \$20,000 to join N with S, \$27,000 to join N with C, \$45,000 to join

N with M, and \$75,000 to join N with A. The other costs are the same as in Example 1.2 and Question 1.2. Which pairs of these five academic buildings should be directly linked in order to minimize the total cost?

2. Suppose that there are six buildings, say A, B, C, D, E, and F, that are to be joined in a LAN system. How many pairs of buildings are there that might be joined by cable? How many pairs will you need to join in order to establish the LAN? The estimated costs (in tens of thousands of dollars) are given in the following table. What should be the pairs of the LAN system?

	A	B	C	D	E	F
A	0	2.7	3.8	2.9	7.8	9.3
B	2.7	0	4.8	5.2	8.4	6.9
C	3.8	4.8	0	3.5	4.6	5.7
D	2.9	5.2	3.5	0	6.4	7.1
E	7.8	8.4	4.6	6.4	0	3.9
F	9.3	6.9	5.7	7.1	3.9	0

3. Suppose that you wanted to connect four buildings in a communications system so that there are two different ways to send messages from any building to any other building. How many pairwise connections would you need? Answer the same question for five buildings.

4. Suppose in Question 1.2 that we require a direct link between McConnell Hall and College Hall. Otherwise, we still wish to find the least expensive LAN connections. Which additional direct connections should we add?

5. Repeat Exercise 1 with the two constraints that McConnell Hall and College Hall must be directly linked, as must Neilson Library and Stoddard Hall.

6. Draw the 3×2 and 6×5 rectangular grid (see Section 3.1). Suppose that each line represents a street and at each street intersection there is a fire hydrant. In each grid find a subset of the streets that connects up all fire hydrants to the lower left-hand corner and whose total length is as short as possible.

5:2 GRAPHS

The LAN of the preceding section can be modeled using what mathematicians and computer scientists call a graph (not to be confused with the graph of a function). For this problem the sets that we consider are the set of buildings and the set of pairs of buildings.

Graph Theory Definitions. A **graph** consists of a finite set of **vertices** together with a finite set of edges. Each **edge** consists of a distinct pair of distinct vertices. If the edge e consists of the vertices $\{u, v\}$, we often write $e = (u, v)$ and say that u

is **joined** to v (also that v is joined to u) and that u and v are **adjacent**. We also say that both u and v are **incident** with the edge e.

Although graphs are frequently stored in a computer as lists of vertices and edges, humans have a more picturesque way to think about graphs. Typically, we shall represent the vertex set of a graph as a set of points in the plane. An edge will be represented by a line segment or an arc (not necessarily straight) joining the two vertices incident with it.

Example 2.1. Figure 5.2 exhibits two typical graphs. The vertices and edges of the graph in Figure 5.2(*b*) are labeled so that we can reinforce the above definitions. In this graph x and z are adjacent as are y and r, and so on. The edge $f = (z, y)$ is incident with both z and y, and y and w are incident with the edge h. Note that r is not adjacent to w. However, if the edges of this graph represent the direct cable connections in a Local Area Network, then w and r are connected in the sense that an electronic message could be sent from w to r going through y.

Look back at Chapter 3 and notice that we were really doing a graph theory problem there, only we didn't call it by that name. We now call the diagrams of that chapter by their usual names: **grid graphs**. Figure 5.3 shows the 3×2 grid graph.

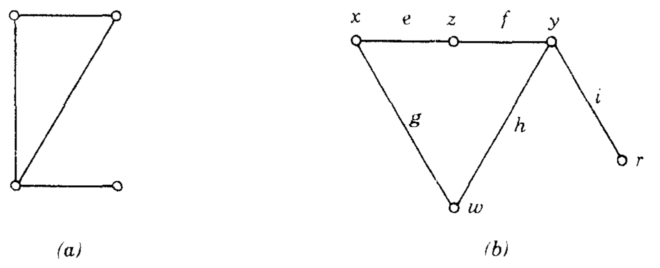


Figure 5.2

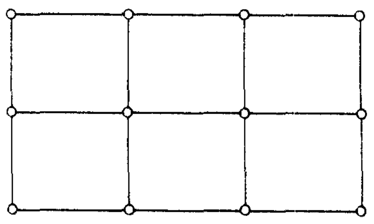


Figure 5.3

Some Notation. In a graph G the vertex set will be denoted by $V(G)$ and the number of vertices by $|V(G)| = V$ (if there is no ambiguity as to which graph we are talking about). Similarly, the edge set will be denoted by $E(G)$ and the number of edges by $|E(G)| = E$. Often the vertices of a graph will be labeled so that they can be distinguished or named according to the intended application.

Example 2.1 (again). In Figure 5.2(*b*), $V(G) = \{x, y, z, w, r\}$ and $V = 5$. Also, $E(G) = \{e, f, g, h, i\} = \{(x, z), (y, z), (x, w), (y, w), (y, r)\}$ and $E = 5$.

Question 2.1. Draw a graph with $V = 4$ and $E = 3$. Is there more than one such graph?

Another Definition. The **degree** of a vertex x is the number of vertices adjacent to x (or equivalently, the number of edges incident with x). Within a graph G we denote the degree of x by $\mathbf{deg}\,(x, G)$ or $\mathbf{deg}\,(x)$ if there is no confusion as to which graph G we refer to.

One way to think about a graph is as if it were constructed of buttons and thread: The buttons represent the vertices and the thread represents the edges. In this model the degree of a vertex is the number of strands of thread emanating from the corresponding button.

Example 2.1 (once more). In Figure 5.2(*b*), $\deg(x) = 2 = \deg(z) = \deg(w)$, while $\deg(y) = 3$ and $\deg(r) = 1$.

Question 2.2. For the graph shown in Figure 5.4 determine V, E, and the degree of each vertex. Find the sum of the degrees of all the vertices.

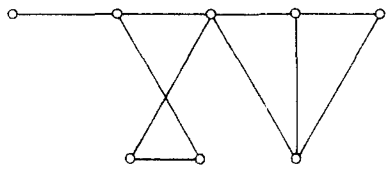


Figure 5.4

In the preceding question you might have noticed that the sum of the degrees was exactly twice the number of edges. That this happens in general is our first result.

Theorem 2.1. If $V(G) = \{x_1, \ldots, x_V\}$, then

$$\deg(x_1) + \cdots + \deg(x_V) = 2E.$$

Proof. Since the degree of a vertex is the number of edges incident with that vertex, the sum of the degrees counts the total number of times an edge is incident with a vertex. Since every edge is incident with exactly two vertices, each edge gets counted twice, once at each end. Thus the sum of the degrees equals twice the number of edges. □

Question 2.3. A vertex whose degree is odd is called **odd**. Show that in any graph there is an even number of odd vertices.

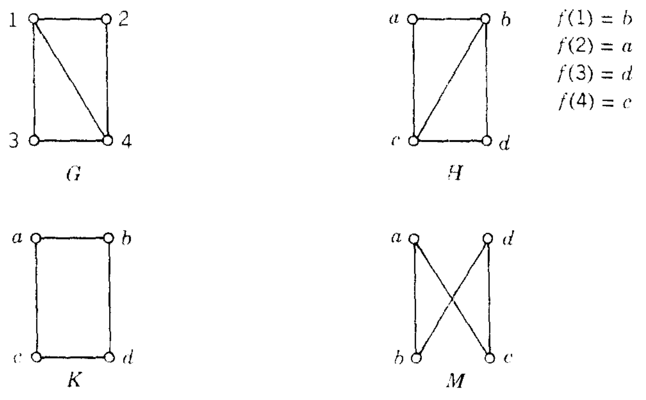
To answer the second part of Question 2.1 with assurance you need to know how to decide if two graphs are the same or different.

Definition. Two graphs, say G and H, are said to be **isomorphic** · there exists a function $f: V(G) \rightarrow V(H)$ such that
(i) f is both one-to-one and onto,
(ii) f preserves adjacencies, and
(iii) f preserves nonadjacencies.

Isomorphic is a fancy mathematical word meaning fundamentally the same. Two graphs that are not isomorphic are also called **different**. Properties (ii) and (iii) can be formalized as
(ii′) If (x, y) is in $E(G)$, then $(f(x), f(y))$ is in $E(H)$ and
(iii′) If (x, y) is not in $E(G)$, then $(f(x), f(y))$ is not in $E(H)$.

The function f that shows the correspondence between the vertices of G and the vertices of H is called an **isomorphism**. In practice, it is displayed by labeling the vertices of G and H with letters or numbers and then explicitly writing out the function values $f(x)$ for each x in $V(G)$. Or the vertices of G and H can be labeled so that if x is labeled A in G, then $f(x)$ is labeled A in H.

Example 2.2. The graphs G and H in Figure 5.5 are isomorphic as are the graphs K and M. Note that in the figure both G and H have four vertices and five edges while K and M both have four vertices and four edges. In general, if G and H are isomorphic graphs, then by property (i), $|V(G)| = |V(H)|$ and by properties (ii) and (iii), $|E(G)| = |E(H)|$. Furthermore, $\deg(x, G) = \deg(f(x), H)$ for every x in $V(G)$.



$f(1) = b$
$f(2) = a$
$f(3) = d$
$f(4) = c$

Figure 5.5

Example 2.3. Figure 5.6 shows the six different graphs on two and three vertices. (Note that a graph need not be connected in the sense of the LAN.)

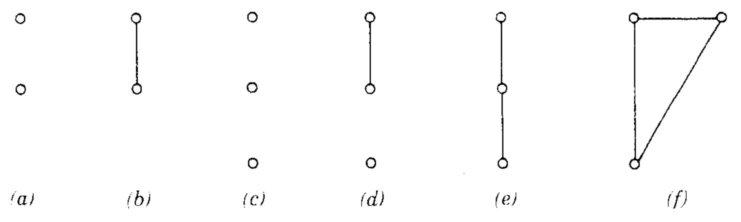


Figure 5.6

Question 2.4. Find the 11 different graphs on 4 vertices.

Question 2.5. Note that the graphs G and H shown in Figure 5.7 have identical degrees. Show that they are not isomorphic.

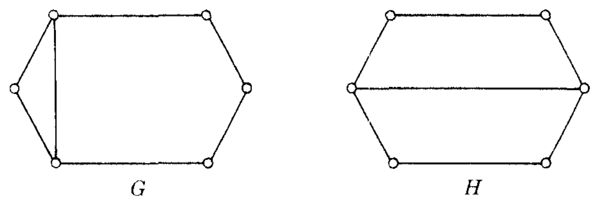


Figure 5.7

It is an open question whether there exists a good algorithm to determine if two graphs are isomorphic. This question is of considerable interest to, for instance, chemists. They model chemical molecules using graphs, where vertices represent the individual atoms and edges represent the chemical bonds. It is sometimes difficult given two molecules with identical atomic constituents to determine whether they are the same. If chemists synthesize a particular molecule in the laboratory and want to find out what is known about it, they need some method of recognizing when the chemical is discussed in the literature, that is, they want to be able to test graph isomorphism. Supplementary Exercises 6 to 9 ask you to construct and analyze straightforward graph isomorphism algorithms.

There is a notable connection between graphs and relations, as defined in Section 4.6. Given a symmetric relation $\sim$ on a finite set S, we may create a graph G with $V(G) = S$ and for $a, b \in S$ an edge $(a, b) \in E(G)$ if and only if $a \sim b$ is true. Conversely, a graph G and its edges define a relation on the elements of $V(G)$. See Exercises 24 to 27 and Supplementary Exercises 32 to 34.

We close this introductory section on graphs by defining and presenting some special classes of graphs.

Definition. A graph in which every pair of distinct vertices is joined by an edge is called **complete**. A complete graph with r vertices is also called an **r-clique** and is denoted by K_r.

Example 2.4. Figure 5.8 exhibits K_4 and K_5. Note that these graphs have 6 and 10 edges, respectively.

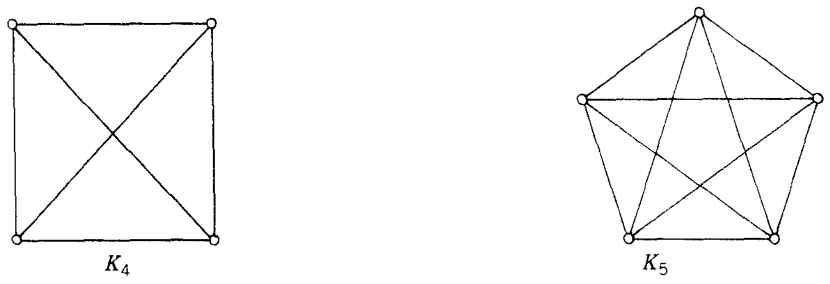


Figure 5.8

Theorem 2.2. An r-clique contains exactly $r(r-1)/2$ edges.

We present four proofs. It is worthwhile to understand all four, since they represent different ways of thinking about the problem.

Proof 1. We proceed by induction. A one-clique is a vertex without any edges, which satisfies the formula. In Figures 5.6 and 5.8 we see that $K_2, \ldots, K_5$ have the correct number of edges. Thus the base case is safely accounted for. Suppose that the theorem is true for $r = k$. We must show that it is then true for $r = k + 1$. Given a $(k+1)$-clique, pick a vertex, say x. If we erase x and all edges incident with x from our graph, we are left with a k-clique, which has $k(k-1)/2$ edges by the inductive hypothesis. In our original graph the vertex x was incident with k edges, one to each of the other vertices. Thus the total number of edges in the $(k+1)$-clique equals

$$\frac{k(k-1)}{2} + k = k\left(\frac{k-1}{2} + 1\right) = \frac{k(k+1)}{2} = \frac{(k+1)\{(k+1)-1\}}{2},$$

which is what we needed to prove. □

Proof 2. Let's draw K_r, adding one vertex at a time. When we draw the first vertex, we need no edges. When we add the second vertex, we need to draw one edge to connect the two vertices. When we add the third vertex, we must join it to the two previously created vertices, so we need to draw two more edges. In general, when we add the kth vertex, we need to draw $k-1$ new edges. Thus the total number of edges we need to draw is

$$1 + 2 + 3 + \cdots + (r - 1).$$

We have seen that this sum equals $(r - 1)\{(r - 1) + 1\}/2$ in Section 2.3. □

Proof 3. In K_r each vertex has degree $r - 1$. Thus the sum of the degrees equals $r(r - 1)$. By Theorem 2.1, this sum also equals $2E$. Thus $2E = r(r - 1)$ and $E = r(r - 1)/2$. □

Proof 4. The number of edges in K_r equals the number of 2-subsets of an r-set, which equals $\binom{r}{2}$ as we saw in Chapter 3. □

Question 2.6. How many edges are there in K_7?

Corollary 2.3. For any graph, $E \leq V(V - 1)/2$.

A graph is said to be **bipartite** if its vertex set can be partitioned into two sets, say R and B, with the property that every edge joins a vertex in R with a vertex in B. Such graphs are also called **2-colorable**, since you can think of the vertices in R as being painted red while the vertices in B are painted blue. With this painting no vertex is joined by an edge to a vertex with the same color. A bipartite graph is said to be a **complete bipartite graph** if every red vertex is joined by an edge to every blue vertex. The complete bipartite graph with p red vertices and q blue vertices is often denoted by $K_{p,q}$. (Notice that this definition cries out for an algorithm to decide if a graph is bipartite, and if it is, to construct the sets R and B; see Supplementary Exercise 10.)

Example 2.5. Each of the graphs in Figure 5.9 is bipartite. The graph in Figure 5.9(*c*) is the complete bipartite graph $K_{3,3}$.

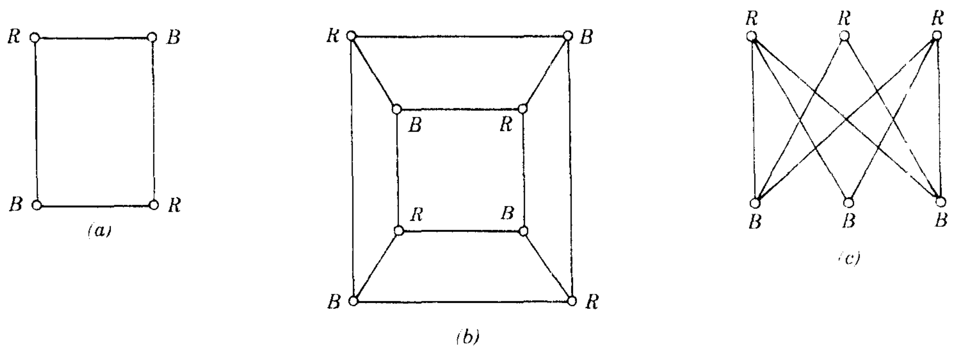


Figure 5.9

Question 2.7. Which of the graphs in Figure 5.10 are bipartite? If a graph is bipartite, color its vertices red and blue so that no edge joins two vertices of the same color. If it is not bipartite, explain why its vertices cannot be so colored.

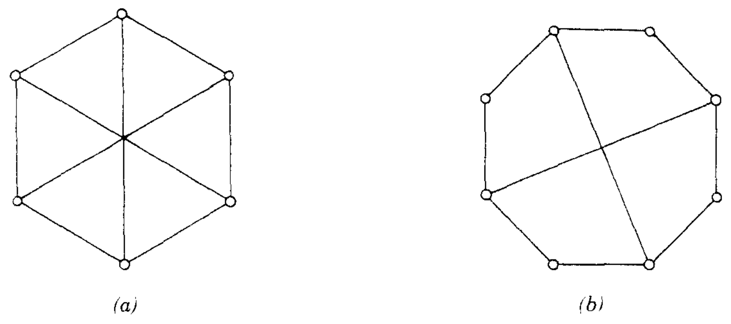
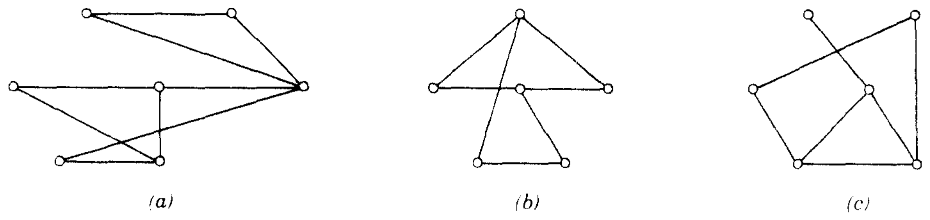


Figure 5.10

EXERCISES FOR SECTION 2

1. Draw a graph with five vertices that illustrates your LAN connections from Exercise 1.1.
2. Draw a graph with 6 vertices and 10 edges.
3. For the following graphs find V, E, and the degree of each vertex.



4. Find all the different graphs with five vertices and two edges. How about three edges? What is the maximum number of edges a graph on five vertices can have?
5. Suppose that G is a graph with V vertices. What is the largest possible degree of a vertex in G?
6. Suppose that G is a graph with V vertices and $E = V - 1$ edges. Prove that G contains a vertex of degree 0 or 1.

7. Prove Theorem 2.1. using induction on E.

8. At the beginning of a business meeting some of the participants are introduced to each other. In an introduction A is introduced to B and B is introduced to A for some pair A and B. Show that the number of individuals who have been introduced to an odd number of other individuals is even.

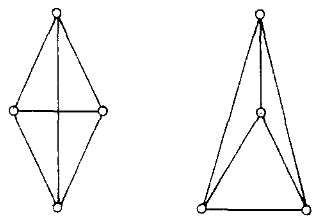
9. A graph in which every vertex has degree r is called **regular** of degree r. Find examples of graphs that are not cliques but are regular of (a) degree 1, (b) degree 2, and (c) degree 3.

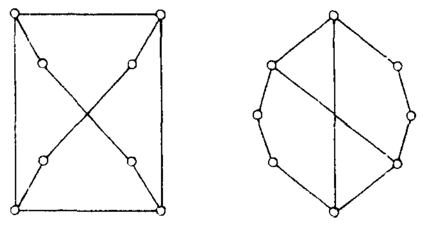
10. For each of the following sequences either find a graph whose vertices have exactly these degrees or show that such a graph cannot exist.

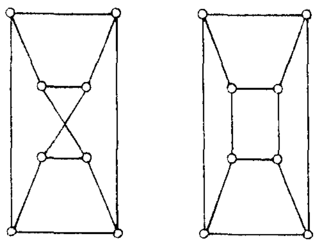
(*a*) 3, 3, 1, 1.	**(*b*)** 3, 2, 2, 1.	**(*c*)** 5, 4, 4, 2, 2, 2.
(*d*) 3, 3, 2, 2, 1, 1.	**(*e*)** 4, 1, 1, 1, 1.	**(*f*)** 2, 2, 2, 1, 1.
(*g*) 7, 3, 3, 3, 2, 2.	**(*h*)** 5, 5, 5, 2, 2, 2, 2, 1.	

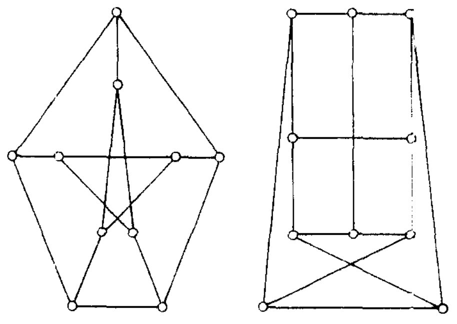
11. **(*a*)** Find all different graphs with 6 vertices and 15 edges.
(*b*) Find all different graphs with 6 vertices and 14 edges.
(*c*) Find all different graphs with 6 vertices and 13 edges.

12. Which of the following pairs of graphs are isomorphic? For each isomorphic pair, exhibit the isomorphism.


(*a*)


(*b*)

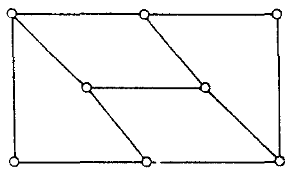

(*c*)

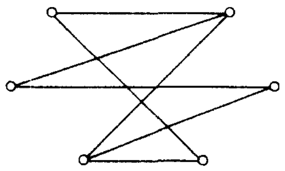

(*d*)

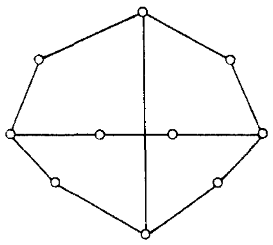
13. Find two nonisomorphic graphs with six vertices both of which are regular of degree 3.

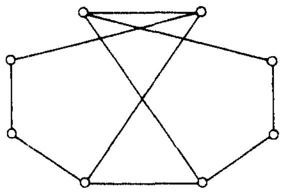
14. Explain why a regular graph with V vertices and E edges must have all vertices of degree $2E/V$. (See Exercise 9.)

15. Which of the following graphs are bipartite? Label the vertices of the bipartite graphs with R and B so that no edge joins two vertices with the same label, and explain why the others cannot be so labeled.


(a)


(b)


(c)


(d)

16. Draw K_6 and K_7. What is the smallest value of n such that K_n has at least 1000 edges?

17. Is there a bipartite graph with 10 vertices that is regular of degree 3? If so, find one; if not, explain why not.

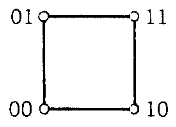
18. Exhibit all the different graphs with 6 vertices and 3 edges, and 6 vertices and 12 edges.

19. Find a formula for the number of edges in $K_{p,q}$. Prove your formula in two different ways, one of which must be by induction.

20. If G is a bipartite graph on 12 vertices, what is the largest number of edges that G might have?

21. If G is a bipartite graph on V vertices, what is the largest number of edges that G might have?

22. The **generalized cubes** Q_n are graphs defined as follows: The vertices of Q_n consist of all binary sequences of length n, that is, $V(Q_n) = \{0, 1\}^n$. Two vertices in Q_n are joined by an edge precisely if the corresponding binary sequences differ in exactly one entry. Thus the vertices of Q_1 are 0 and 1 and there is an edge joining them. The graph Q_2 follows. Draw Q_3 and Q_4. How many vertices and how many edges does Q_n have?



23. Show that for all n, Q_n {defined in Exercise 22} is bipartite.
24. Let $S = \{0, 1, 2, \ldots, 9\}$ and suppose that for $x, y \in S$, $x \sim y$ is true if and only if x and y are both even or both odd. Draw the corresponding graph.
25. Let $G = K_{3,3}$. Specify the corresponding relation on $V(G) = \{1, 2, 3, 4, 5, 6\}$. Do the same for $G = K_6$.
26. Are the relations in Exercise 25 reflexive or transitive?
27. If G is a graph, explain why the corresponding relation $\sim$ is necessarily a symmetric relation.

5:3 TREES AND THE LAN

In our model of cable connections for the LAN system we wanted to be able to send an electrical signal from any building to any other building. This property corresponds with the notion of connectivity. Here is an informal description. With the button and thread image from the previous section, a graph is connected if whenever you pick up a button and walk out of the room, the entire graph comes with you. If only some of the graph comes with you, the part that comes with you is called a (connected) component.

Example 3.1. In Figure 5.11, G is connected while H has three components.

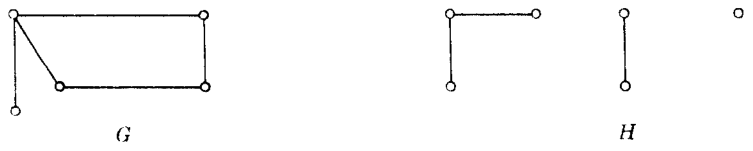


Figure 5.11

Even though this informal definition of connectivity captures the spirit of the concept correctly, we need a less "seamy" definition to enable us to prove theorems about connected graphs.

Definition. Given a graph G with vertices x and y, a **path** from x to y of length k is a sequence of k distinct edges, $e_1, e_2, \ldots, e_k$, such that

$$\begin{aligned} e_1 &= (x, x_1),\\ e_2 &= (x_1, x_2),\\ &\ldots\\ e_j &= (x_{j-1}, x_j),\\ &\ldots \end{aligned}$$

and finally,

$$e_k = (x_{k-1}, y).$$

Frequently, we just list the vertices that are incident with the edges of the path, like $x, x_1, x_2, \ldots, x_{j-1}, x_j, \ldots, x_{k-1}, y$. We may say that there is a path of length zero from a vertex x to itself. A path from x to itself of length k is called a ***k*-cycle**.

Example 3.2. In the graph of Figure 5.12 the vertices a, b, c, d, and e form a 5-cycle. There is a path from a to z of length four using the vertices b, c, and r as well as a path from a to z of length three using the vertices e and d. In the latter path $e_1 = (a, e)$, $e_2 = (e, d)$, and $e_3 = (d, z)$.

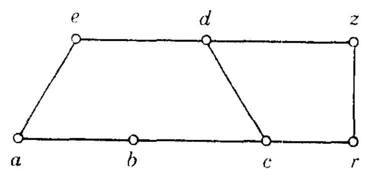


Figure 5.12

Question 3.1. In the graph of Figure 5.13, find a path of length five from a to b, a path of length three from z to r, and a 4-cycle through b.

Recall that we used this terminology when discussing grids in Chapter 3. For example, in the 6×5 grid graph we were searching for paths of length 11 from (the vertex) M to (the vertex) P.

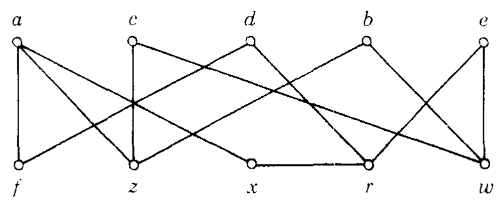


Figure 5.13

Definition. The graph that consists of a path of length k from one vertex to another vertex with no repeated vertices is called $\boldsymbol{P_k}$. The graph that is a k-cycle with no repeated vertices is called $\boldsymbol{C_k}$.

Example 3.3. The graphs P_3 and C_5 are shown in Figure 5.14.

Figure 5.14

We use the concept of path to define connectivity precisely.

Definition. A graph G is said to be **connected** if for every pair of vertices x and y, there exists a path from x to y. The **distance** between x and y, denoted $\boldsymbol{d(x, y)}$, is the smallest number of edges in a path from x to y. Given a vertex x within a graph G, the **component** of G containing x consists of the set of vertices and the set of edges in G that are a part of some path beginning at x.

We emphasize that this does not mean that in a connected graph every pair of vertices is joined by an edge.

If we construct a graph to model the LAN, the vertices will represent the buildings of the campus and the edges will represent the pairs of buildings that are directly joined by a coaxial cable. We want this graph to be connected. On the other hand, we don't want to include unnecessary connections. These properties suggest the following definitions.

Definition. A **forest** is a graph with no cycles. (Such a graph is also called **acyclic**.) A connected graph with no cycles is called a **tree**.

Trees are the most widely applicable type of graph. We shall explore some of their properties in the remainder of this section and use them to settle the LAN question.

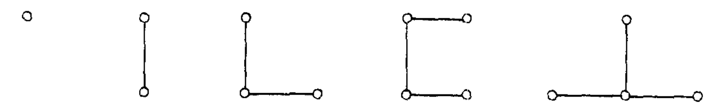


Figure 5.15

Example 3.4. Figure 5.15 exhibits all the different trees on fewer than 5 vertices. The union of these graphs forms a forest with 5 components and 14 vertices. Figure 5.16 shows a tree whose bark is worse than its bite.

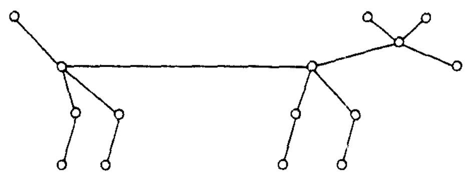


Figure 5.16

Question 3.2. Find the three different trees on five vertices.

Question 3.3. How many edges does a tree on six vertices contain? Does the answer depend upon the tree or is the answer the same for all trees on six vertices?

Theorem 3.1. If T is a tree with V vertices and E edges, then

$$E = V - 1.$$

Proof. We proceed by induction on the number of vertices. You can consult Figure 5.15 to see that the result is true for $V = 1$, 2, 3, and 4. We assume that the result is true for all trees with fewer than V vertices. Consider an arbitrary tree T with V vertices and an edge $e = (x, y)$. What happens if we remove the edge e from T? We illustrate in Figure 5.17.

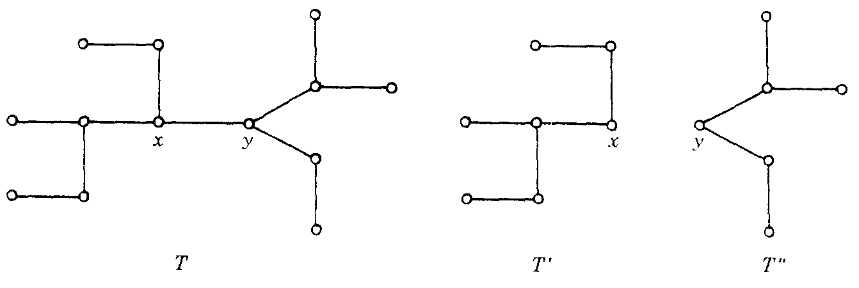


Figure 5.17

The resulting graph is a forest made up of two trees, one containing the vertex x and the other containing the vertex y. Let T' denote the tree containing x and T'' denote the tree containing y. Suppose that the number of vertices in T' is V' and the number of edges in T' is E'. Similarly, the number of vertices in T'' is V'' and the number of edges in T'' is E''. Since both T' and T'' are trees that have fewer vertices than T, we can assume by the inductive hypothesis that

$$E' = V' - 1 \qquad \text{and} \qquad E'' = V'' - 1.$$

The number of edges in T is one more than $E' + E''$ (since e is in T) while the number of vertices in T equals $V' + V''$. Thus

$$\begin{aligned} E &= E' + E'' + 1 \\ &= (V' - 1) + (V'' - 1) + 1 \\ &= V' + V'' - 1 \\ &= V - 1. \end{aligned}$$ □

Question 3.4. If you want to join every pair of buildings on a campus with 40 buildings, how many different coaxial cables do you need? What is the minimum number of cables that must be installed to sustain a LAN system connecting these 40 buildings?

Question 3.5. (a) Let C denote the number of component trees in the forest shown in Figure 5.15. Show that $E = V - C$. (b) More generally, show that if F is any forest with V vertices, E edges, and C component trees, then $E = V - C$. [*Hint:* Suppose that for $i = 1, \ldots, C$, the ith component tree contains V_i vertices and E_i edges.] (See also Exercises 24 and 25.)

That a tree on V vertices has exactly $V - 1$ edges is fundamental. Indeed this property can be exchanged with either of the two defining properties of a tree. This is shown in the next theorem and the questions that follow it.

Theorem 3.2. If G is an acyclic graph with V vertices and $V - 1$ edges, then G is a tree.

Proof. If G is acyclic, then G is by definition a forest. By the result in Question 3.5 a forest with V vertices, E edges, and C component trees necessarily has

$$E = V - C.$$

Solving for C and substituting for E yields

$$\begin{aligned} C &= V - E \\ &= V - (V - 1) = 1. \end{aligned}$$

Thus $C = 1$ and G is a tree. $\square$

Question 3.6. Show that if G is a connected graph containing a cycle and if e is any edge of the cycle, then $G - e$ is connected. (By $G - e$ we mean the graph obtained from G by erasing e, that is, removing e from the edge set while leaving e's incident vertices in the vertex set.) (*Hint:* Pick a pair of vertices in G, say x and y, and describe a path from x to y in $G - e$.)

Question 3.7. Show that a connected graph with V vertices and $V - 1$ edges is a tree. (*Hint:* Use the result of Question 3.6.)

We now have most of the mathematical machinery necessary to construct an algorithm to pick the pairs of buildings that ought to be joined directly by coaxial cables in the LAN system. In our model of the campus buildings we wanted enough cables to be installed so that the graph that represents the cable connections is a tree. Recall the two defining properties of trees: They are connected and acyclic. We need the connectivity because that is just the property that mimics the "real world requirement" that electrical signals can be sent between any pair of buildings. If the cable graph had a cycle, then by Question 3.6, it would remain connected if some cable in the cycle were removed. Consequently, in seeking a minimum cost connected graph we are inexorably led to a tree (provided that no edge has a negative cost, i.e., no one is willing to pay us to install an extra cable connection). We need just a few more definitions to be able to finish the task.

Definitions. Suppose that $V(G)$ and $E(G)$ denote the vertex and edge sets of a graph G. If H is a graph with the properties that
(1) $V(H) \subseteq V(G)$,
(2) $E(H) \subseteq E(G)$, and
(3) every edge of $E(H)$ has both its incident vertices in $V(H)$,
then H is called a **subgraph** of G. If $V(H) = V(G)$, then H is called a **spanning subgraph** of G. If in addition H is a tree, then H is called a **spanning tree** of G.

Example 3.5. Let G be the graph shown in Figure 5.18. If $V(H) = \{1, 2, 3\}$ and $E(H) = \{(1, 2)\}$, then H is a subgraph. If $V(H) = \{1, 3, 4\}$ and $E(H) = \{(1, 2)\}$, then H is not a subgraph. If $V(H) = \{1, 2, 3, 4, 5\}$ and $E(H) = \{(1, 2), (2, 3), (3, 4), (4, 5), (1, 5)\}$, then H is a spanning subgraph. If $V(H) = \{1, 2, 3, 4, 5\}$ and $E(H) = \{(1, 2), (2, 3), (3, 4), (4, 5)\}$, then H is a spanning tree.

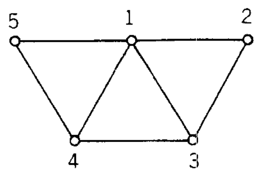


Figure 5.18

Question 3.8. Find a spanning tree of the graph in Figure 5.19.

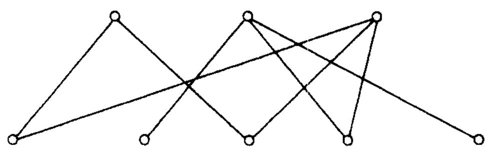


Figure 5.19

Does every graph have a spanning tree and how can we find a spanning tree in a graph with one? In Exercise 13 you are asked to prove that a graph contains a spanning tree if and only if it is connected. If a graph is disconnected, it contains a spanning tree for each connected component; such a collection of trees is called a **spanning forest**. In Exercise 19 we present an algorithm SPTREE that produces a spanning tree of a connected graph. The algorithm SPTREE actually does more than promised and produces a spanning forest of a disconnected graph.

We return to the LAN problem. We improve our graph theory model to incorporate the costs of installing the cables.

Weighty Definitions. Let R^+ denote the set of positive real numbers. A graph G together with a function $w: E(G) \to R^+$ is called a **weighted** (or an **edge-weighted**) **graph**. If e is in $E(G)$, then $w(e)$ will be the weight of e. If $F \subseteq E(G)$, then $\mathbf{w(F)}$, the **weight of** $\boldsymbol{F}$, is defined to be the sum of the weights of the edges in F.

Typically, the numbers assigned to the edges will represent costs, capacities, lengths, or some parameter of real-world interest.

Example 3.6. Given the weighted graph G shown in Figure 5.20, the graph H is a (weighted) spanning tree of G whose total weight is 37.

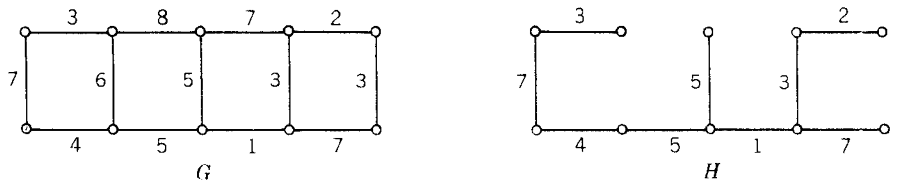


Figure 5.20

Question 3.9. Given the weighted graph shown in Figure 5.21, find all the spanning trees of this graph and the weight of each.

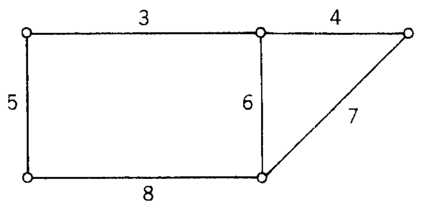


Figure 5.21

Our LAN problem can now be formalized as follows. Given the weighted graph G with vertices representing the buildings, edges representing the possible direct cable connections, and edge weights representing the installation costs of the corresponding cables: find H, a minimum weight spanning tree of the graph G. It is important to distinguish the edges of G that represent the possible direct-cable connections from those of H that represent the actual direct-cable connections. We shall examine two different algorithmic solutions to this question. First we state the problem succinctly.

Problem. Given a weighted graph G, find a spanning tree H with minimum total weight.

The idea of our initial, naive algorithm is that every subset of $V - 1$ edges of a graph that forms a connected or an acyclic subgraph gives a spanning tree by Theorem 3.2 and Question 3.6.

Algorithm BADMINTREE

```
STEP 1.  Input the weighted graph $G$
STEP 2.  Use algorithm JSET from Chapter 3 to find all subsets of the edges
         of $G$ that contain exactly $V - 1$ edges
STEP 3.  If a $(V - 1)$-subset of $E(G)$ forms a tree, compute its weight
STEP 4.  If there are no trees in step 3, output the fact that there is no spanning
         tree; otherwise, select and output a spanning tree of minimum weight;
         then stop.
```

Question 3.10. Run the algorithm BADMINTREE on the weighted graph in Figure 5.22.

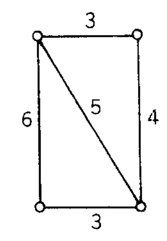


Figure 5.22

The name of this algorithm gives away the quality of its performance. We know that JSET performs at least $\binom{E}{j}$ steps to find all j-subsets of a set with E elements; here it's the set of E edges whose $(V-1)$-subsets we list. Each edge represents the potential installation of a cable. Since, in general, we cannot eliminate any of these possibilities, by Theorem 2.3, $E = V(V-1)/2$. Thus the number of steps is at least

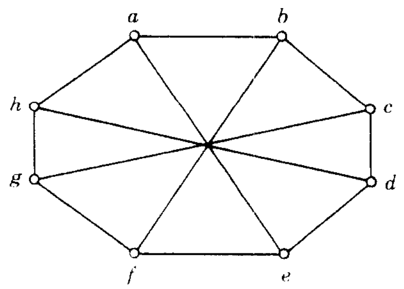
$$\binom{V(V-1)/2}{(V-1)}. \qquad (*)$$

Question 3.11. Evaluate (*) for $V = 3, 4, 5, 6, 7$.

This binomial coefficient evidently grows rapidly. Exercise 23 asks you to find lower bounds. In the next section we shall find a much more efficient way to solve the LAN Problem.

EXERCISES FOR SECTION 3

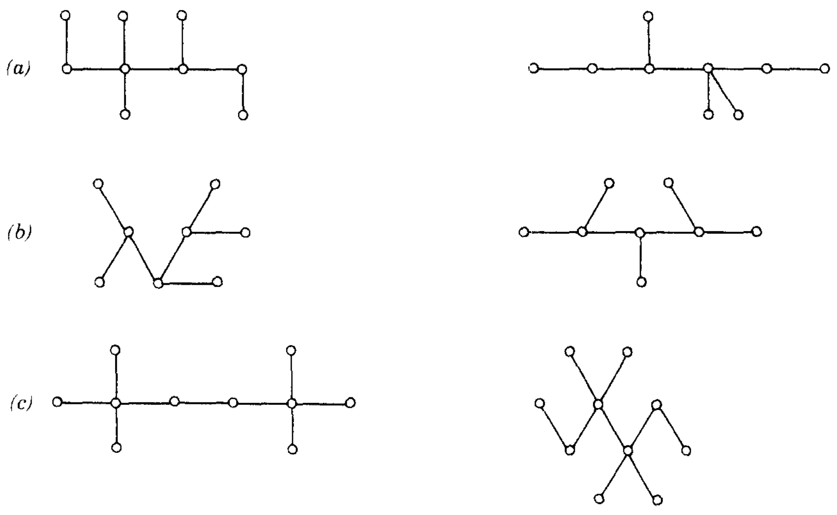
1. Find all trees with six vertices. Find a graph with six vertices and five edges that is not a tree. Can you find such a graph that is also acyclic?
2. For what values of n is C_n bipartite?
3. Show that trees are bipartite.
4. How many different 5-cycles are there in the following graph? Before answering, specify what it means for two 5-cycles to be different.



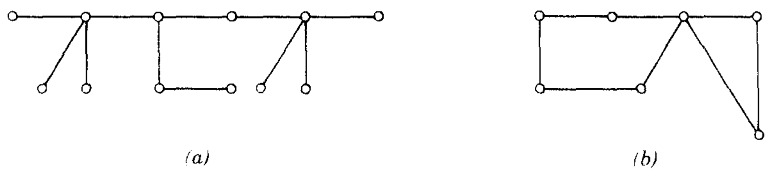
5. Give an example of a graph G and a subgraph H of G that is not a spanning subgraph. Give an example of a spanning subgraph of G that is not a spanning tree.

6. The **average degree** of a graph is the sum of the degrees of all the vertices of the graph divided by the number of vertices. Show that the average degree of a forest is less than 2.

7. Which of the following pairs of trees are isomorphic?



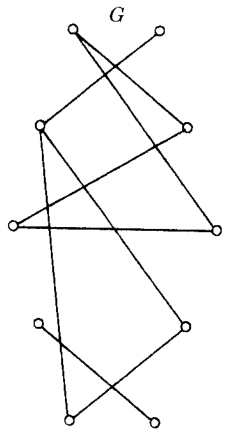
8. Find the longest path and the longest cycle that is a subgraph of the following graphs and give their lengths. Then find the largest value of j and k such that P_j and C_k are subgraphs of the given graph.



9. Let x and y be two vertices in the r-clique, K_r. Explain why there are paths of length $1, 2, \ldots,$ and $(r-1)$ joining x and y. For each $r > 3$, describe a graph other than a clique that contains r vertices, some pair of which are joined by paths of all possible lengths.

10. Identify the components in the following graphs.

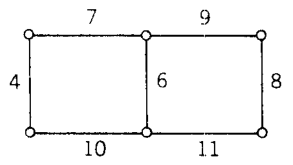
(*a*)



(*b*) H where $V(H) = \{1, \ldots, 10\}$ and $E(H) = \{(1,2), (3,4), (5,6), (7,8), (9,10), (3,5), (5,8), (1,9), (4,10), (6,9)\}$

11. If G is a connected graph and x, y, and z are vertices, is it always true that $d(x,y) + d(y,z) \geq d(x,z)$? Give a proof or counterexample.

12. Given the weighted graph shown here, find all of the spanning trees of this graph and the weight of each.



13. (*a*) Prove that a graph is connected if and only if it has a spanning tree.

(*b*) Prove that every graph contains a spanning forest that consists of a spanning tree of each component.

14. Determine whether the following are true or false. Prove each true statement and give a counterexample for each false statement.

(*a*) A graph with $E \geq V$ is connected.

(*b*) A graph with $E \geq V$ contains a cycle.

(*c*) A graph with $E \leq V - 2$ is not connected.

(*d*) A graph with $E \leq V - 2$ is acyclic.

(*e*) A graph with two components has at most $V - 2$ edges.

(*f*) A graph with $E = V - 2$ has at least two components.

(*g*) A graph with $E = V - 2$ has exactly two components.

(*h*) If G is a connected graph containing a cycle, then the removal of any edge of the graph leaves the graph connected.

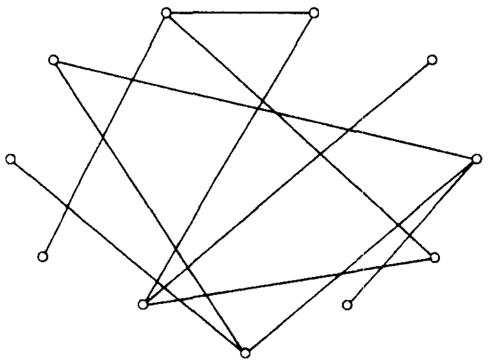
(*i*) Every spanning subgraph H of a connected graph G has $|E(H)| = |V(H)| - 1$.

(*j*) A graph with $E = V + 1$ contains at least two cycles.

(*k*) A graph with $E = V + 1$ contains exactly two cycles.

15. Are two trees with the same sequence of degrees necessarily isomorphic? Give a proof or a counterexample.

16. Find a spanning forest of the following graph.



17. A subgraph H of a graph G is called an **induced subgraph** if whenever x and y in $V(H)$ are joined by an edge e in G, then e is also in H. Determine which of the following subgraphs of G are induced:

$$V(G) = \{1,2,3,4,5,6\} \qquad E(G) = \{(1,2),(1,4),(1,5),(2,3),(2,5),(2,6),(3,6),(4,5),(5,6)\}$$

(*a*) $V(H) = \{1,2,3\}$ $\quad E(H) = \{(1,2),(2,3)\}$

(*b*) $V(H) = \{1,2,4,5\}$ $\quad E(H) = \{(1,2),(1,4),(2,5),(4,5)\}$

(*c*) $V(H) = \{1,3,4,6\}$ $\quad E(H) = \{(1,4),(3,6)\}$

18. Suppose that G is a connected graph and T is a spanning tree of G. When is it the case that T is an induced subgraph of G?

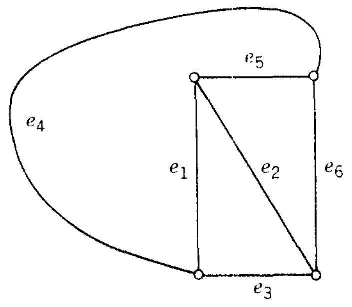
19. Here is an algorithm that finds (if possible) a spanning tree of the input graph.

Algorithm SPTREE

```
STEP 1. Input the graph $G$ with $V$ vertices and edge list $e_1, e_2, \ldots, e_E$
STEP 2. For $j = 1$ to $E$ do
        STEP 3. Add edge $e_j$ to the spanning tree $T$ if it creates no cycle
                with the edges already in $T$
```

STEP 4. If T contains $V - 1$ edges, then output T as the desired spanning tree and stop; otherwise, declare that G contains no spanning tree and stop.

Run the algorithm SPTREE on the following graph.



20. Prove that SPTREE stores a spanning tree in T if and only if G is connected.

21. Prove that SPTREE stores a spanning forest in T in all cases. Modify SPTREE so that it always outputs T and determines the number of connected components of G. (*Hint:* See Question 3.5.)

22. Show that in a tree, every pair of vertices is joined by a unique path. Is the converse true, that is, if G is a graph in which every pair of vertices is joined by a unique path, then is G a tree? Give a proof or a counterexample.

23. Find N such that if $V > N$, then the binomial coefficient given in (*) is greater than 2^V. Show that, in any case, the binomial coefficient is greater than $\left(\frac{V-1}{2}\right)^{V-1}$.

24. (An alternate solution to Question 3.5). Using induction on the number of components, show that if F is a forest with V vertices, E edges, and C component trees, then $E = V - C$.

25. (Another alternate solution to Question 3.5). Given a forest F with V vertices, E edges, and C component trees labeled $T_1, \ldots, T_C$, for $i = 1, \ldots, C - 1$, add an edge from some vertex of T_i to some vertex of T_{i+1}. Use the resulting graph to prove that $E = V - C$.

5:4 A GOOD MINIMUM-WEIGHT SPANNING TREE ALGORITHM

We now present an algorithm that is dramatically better than BADMINTREE. It is universally known as Kruskal's algorithm. Kruskal did write the first paper developing this particular algorithm in 1956. However, there are earlier algorithms that correctly and efficiently find minimum-weight spanning trees. The earliest known such algorithm is due to Otakar Borůvka, who as an electrical engineer, was working on the problem of the electrification of Southern Moravia about 60 years ago.

We are seeking a spanning tree of small weight. A lightweight spanning tree contains lightweight edges. Thus we build our tree using the lightest possible edge at each stage. It is plausible that such a strategy might produce a reasonably light tree. It is surprising (as we discuss in the next section) that this strategy is guaranteed to produce a minimum-weight spanning tree. The following algorithm works on the graph G assuming that its edge list is arranged so that the weights are **increasing** [i.e., for all $1 \le i < j \le E$, $w(e_i) \le w(e_j)$].

Algorithm KRUSKAL

```
STEP 1.  Input the weighted graph $G$ {Assume that $G$ has $V$ vertices and $E$
         edges and that the edge list of $G$ is in increasing order, by weight.}
STEP 2.  Set $j := 1$ {$j$ will index the edges of $G$.}
STEP 3.  Set $T$ to be empty {$T$ will contain the edges of the minimum-weight
         spanning tree.}
STEP 4.  Set $k := 0$ {$k$ records $|E(T)|$.}
STEP 5.  Repeat
         Begin
         STEP 6.  If $T + e_j$ is acyclic, then do {add $e_j$ to tree}
                  Begin
                  STEP 7.  $T := T + e_j$
                  STEP 8.  $k := k + 1$
                  End {step 6}
         STEP 9.  $j := j + 1$
         End {step 5}
         Until either $k = V - 1$ or $j > E$
STEP 10. If $k = V - 1$ report success, output $T$, and stop. Otherwise, report
         failure and stop.
```

COMMENT. We use "+," as in $T + e_j$, to denote set theoretic union.

Theorem 4.1. If G is weighted graph, then KRUSKAL outputs a minimum-weight spanning tree if and only if G is connected.

Example 4.1. We run KRUSKAL on G shown in Figure 5.23 to obtain the minimum-weight spanning tree T.

Question 4.1. Run KRUSKAL on the weighted graphs shown in Figure 5.24.

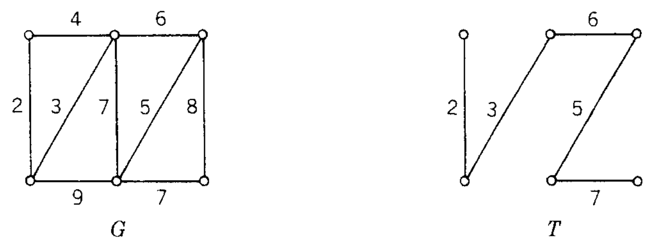


Figure 5.23

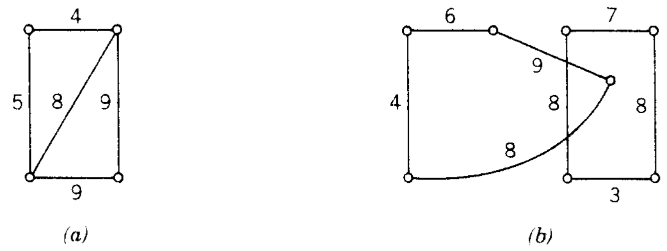


Figure 5.24

Proof of Theorem 4.1. First let's see that if KRUSKAL reports success, then the graph in T is a tree. In step 7 the edge e_j is added to the set T precisely when $T + e_j$ is acyclic. Thus the subgraph formed by T must always be acyclic. If KRUSKAL returns success, then T has $V - 1$ edges. By Theorem 3.2 we know that an acyclic graph with $V - 1$ edges is necessarily a tree. Consequently, if KRUSKAL returns success, then the edges stored in T are the edges of a tree.

Next let's see that if KRUSKAL returns failure, then G does not contain a spanning tree. For KRUSKAL to return failure, after examining all the edges of the graph, the set T does not contain $V - 1$ edges but still forms an acyclic subgraph. Thus T is a forest. By Question 3.5 if the number of edges in T is $V - C$, then C equals the number of components of T. Since T contains fewer than $V - 1$ edges, T contains more than one component. Suppose that Q is a component of T. If the whole graph G were connected, then there would be an edge in G, say e, that joins a vertex of Q with a vertex of another component of T. When KRUSKAL examined the edge e, it would have found that $T + e$ was acyclic and thus T would have contained e. Since e did not make it into T, G must not have been connected. Thus KRUSKAL reports a failure when G is disconnected.

A more difficult thing to verify is that the tree returned by KRUSKAL is a minimum-weight spanning tree. We begin with a lemma.

Lemma 4.2. Suppose that T_1 and T_2 are two different spanning trees of a connected graph G and that c is an edge of T_1 but not of T_2. Then there is an edge d of T_2 but not of T_1 such that $T_2 + c - d$ is a spanning tree of G.

Proof. Suppose that T_1, T_2, and c are as given in the lemma. Consider the subgraph whose edges are $T_2 + c$. Since T_2 is a spanning tree, $T_2 + c$ is a connected, spanning subgraph of G with V edges. Thus $T_2 + c$ must contain a cycle C (using the contrapositive of Theorem 3.1). C is not contained in T_1, since T_1 is acyclic. Thus there is an edge d of C that is in T_2 but not in T_1. Consider $T_2 + c - d$, a subgraph with $(V - 1)$ edges. Since d is an edge in a cycle of $T_2 + c$, its removal does not disconnect the subgraph by Question 3.6. Thus $T_2 + c - d$ is a connected subgraph with $(V - 1)$ edges. By Question 3.7 it is a spanning tree of G. □

Question 4.2. In the graph G with spanning trees T_1 and T_2 given in Figure 5.25, find an edge c and the corresponding edge d whose existence is guaranteed by Lemma 4.2.

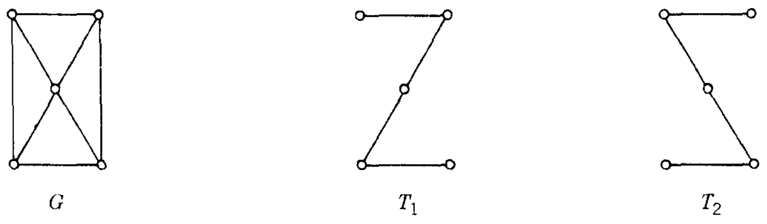


Figure 5.25

To finish the proof that if KRUSKAL reports success, then the spanning tree in T is of minimum weight, we assume that the edge weights of G are all distinct. (You can complete the general proof in Supplementary Exercise 31.) The proof will be by contradiction. We negate the conclusion we are seeking to prove and assume that T is not a minimum-weight spanning tree. Thus there exists a minimum-weight spanning tree, say F, with $w(F) < w(T)$. Since $T \neq F$, there exists an edge in T that is not in F. Let e denote the lightest-weight such edge. We apply Lemma 4.2 with $T_1 = T$, $T_2 = F$, and $c = e$. Thus there exists an edge, say f, such that f is in F but not in T. By the lemma $F' = F + e - f$ is a spanning tree. If $w(e) < w(f)$, then $w(F') < w(F)$, contradicting the assumption that F is a minimum-weight spanning tree. On the other hand, suppose that $w(f) < w(e)$. Since e was the lightest-weight edge in T, but not in F, all the edges selected by KRUSKAL before e are also in F. Thus f would have been added to T instead of e, another contradiction. We conclude that T is a minimum-weight spanning tree. □

There are two steps within KRUSKAL that should provoke comment. The first is the requirement that the edges of G be input in increasing order. There is a straightforward way to do this. Specifically, we could examine the edge list of

G to find an edge with the smallest weight and list it first. We could then find a second smallest edge and list it second, and so on. We shall see in the next chapter that this procedure would perform $O(E^2)$ comparisons to list the E edges of G in increasing order. This method is analogous to the algorithm MAX, presented in Exercise 2.4.12. We shall also see that there are more efficient ways to do this sorting.

The second difficulty with KRUSKAL as presented above occurs in step 6. Specifically, how should we decide, given an acyclic set of edges T and an edge e_j, whether $T + e_j$ is acyclic? In small examples we can obviously "eyeball" the set of edges for cycles. One way to check for cycles in larger and more general examples is to keep track of the connected components of T at each stage of its creation. If e_j joins the two vertices x and y, then the addition of e_j creates a cycle if and only if x and y are in the same component of T.

Example 4.1 (continued). Initially, T is empty, and we consider every vertex to be a separate component of T. First we added the edges of weights 2 and 3 because not only are they the lightest-weight edges, but also they join vertices in different components of T. Now T has a component consisting of these two edges and three vertices as well as three additional components, each consisting of an isolated vertex. The next edge of weight 4 is rejected because it joins two vertices in a component, whereas the edge of weight 5 joins two vertices in different components and is accepted in T.

It is not hard to estimate the complexity of KRUSKAL. The principal operations in this and most graph theory algorithms are comparisons. In KRUSKAL we first compare edge weights so that the edges are rearranged in increasing order. As mentioned previously, ordering the edges might take $O(E^2)$ comparisons. The loop at step 5 is repeated no more than E times. Testing $T + e_j$ for cycles requires that we keep track of the components of T at each stage, so with one comparison we can tell whether e_j joins two vertices in the same component. However when e_j does not form a cycle and is added to T, we need to update the components of T because the addition of e_j causes two components to be joined into one. This updating can be done with at most V comparisons. (For more details, see Exercises 11 to 13.) Thus there are $O(V)$ comparisons done within the loop beginning at step 5 and no more than $O(E\,V)$ comparisons after the ordering of the edges. Thus the total number of comparisons and assignments is $O(E^2) + O(E\,V)$.

A careful analysis of a more efficient implementation could achieve the result that the algorithm including an efficient sorting routine in step 1 is $O(E\log(E))$. Notice that this bound and the previous one can also be expressed in terms of (only) V, since by Corollary 2.3, $E \leq V(V-1)/2 = O(V^2)$.

Question 4.3. Express $O(E^2) + O(E\,V)$ and $O(E\log(E))$ as $O(f(V))$, where f is a function of V but not of E.

A formal analysis of any graph algorithm must consider how to input the graph G as a string of zeros and ones. One convenient method uses what is called the adjacency matrix of a graph. A **matrix** is a rectangular array of entries, usually numbers; an $r \times s$ matrix consists of $r \cdot s$ entries arranged in r rows and s columns. There are exactly s entries in every row and exactly r entries in every column.

Example 4.2. Here is a 2×3 matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}.$$

Suppose that G is a graph with V vertices that are labeled $1, \ldots, V$. We define $A(G)$, the **adjacency matrix of G**, to be the $V \times V$ matrix that has a one in the ith row and jth column if the vertex labeled i is adjacent to the vertex labeled j. All other entries of $A(G)$ equal zero.

Example 4.3. Here is a graph given first by its edge list and then by its adjacency matrix:

Edge List	*Adjacency Matrix*					
{1,2}	0	1	1	1	1	1
{1,3}	1	0	1	0	0	0
{1,4}	1	1	0	0	0	0
{1,5}	1	0	0	0	0	0
{1,6}	1	0	0	0	0	0
{2,3}	1	0	0	0	0	0

Question 4.4. Find the adjacency matrix of each graph in Figure 5.26.

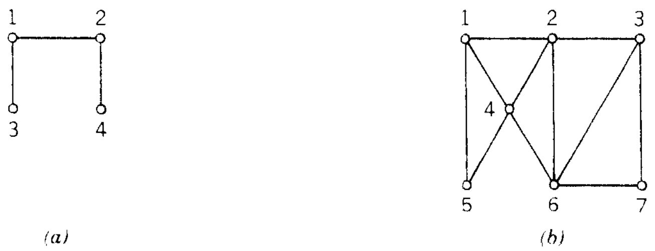


Figure 5.26

Question 4.5. Draw the graphs whose adjacency matrices are as follows.

(a) $\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$ (b) $\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

This form of representing a graph has advantages and disadvantages. Notice that it gives us a way to input a graph into an algorithm as a string of V^2 zeros and ones obtained by laying out the matrix, row by row, as one long string.

Example 4.4. Here is the string of $V^2 = 36$ zeros and ones that represents the graph of Example 4.3.

011111101000110000100000100000100000

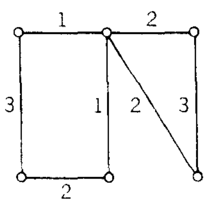
Thus the number of bits needed to input the adjacency matrix of a graph with V vertices is given by $B = V^2$.

Question 4.6. Suppose that $f(n)$ counts the number of comparisons made in the worst case of a graph algorithm and suppose that $f(n) = 0(n^k)$ for some positive integer k. If $B = n^2$, find a big oh bound on the number of comparisons made in terms of B.

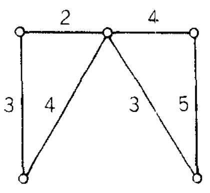
The result of Question 4.6 indicates that if we determine the complexity of a graph theory algorithm to be bounded by a polynomial in V, then it is also bounded by a polynomial in B and hence is a good algorithm. The converse is also true, that if an algorithm requires an exponential number of steps in terms of V, then it also requires a nonpolynomial number in terms of B and is a bad algorithm. (See Supplementary Exercise 30.) In particular, KRUSKAL is a good algorithm and BADMINTREE is not.

EXERCISES FOR SECTION 4

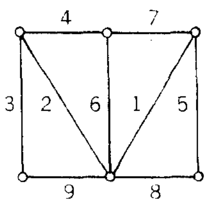
1. Run KRUSKAL on the following weighted graphs.



(a)



(b)



(c)

2. Suppose that G is the weighted graph with $V = 7$ and $E = 10$ whose edges are (in lexicographic order) $(1,2)$, $(1,5)$, $(1,6)$, $(2,3)$, $(2,6)$, $(2,7)$, $(3,4)$, $(4,5)$, $(5,6)$ and $(6,7)$. The weights are given by 1, 1, 2, 2, 1, 2, 3, 2, 1, and 3, respectively. Run KRUSKAL on this graph. Find all minimum-weight spanning trees of this graph.

3. Is the following variation on Lemma 4.2 true or false? Suppose that H_1 and H_2 are two different spanning subgraphs of G that are themselves connected graphs. Suppose that e is an edge of H_1 but not of H_2. Then there exists an edge f of H_2 but not H_1 such that $H_2 + e - f$ is a connected spanning subgraph of G. Explain.

4. Find adjacency matrices for the graphs in Exercises 1 and 2. You will have to label the vertices of the graphs from 1 to V.

5. Suppose that a graph G has adjacency matrix

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}.$$

Without drawing G determine the number of edges of G and the degree of each vertex. Describe, in general, how to obtain the degrees of the vertices from the adjacency matrix.

6. Suppose that G is a weighted graph with V vertices. Find a way to describe G including the weights as a $V \times V$ matrix.

7. Which of the following is the adjacency matrix of a graph? Explain.

(a) $\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$ (b) $\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$ (c) $\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$

8. Find an example of a weighted graph G whose edge weights include negative numbers and with the property that a minimum-weight connected spanning subgraph is not a tree.

9. Suppose that in Exercise 2 we require that edges e_7 and e_{10} be included in a spanning tree, but otherwise the spanning tree should be as light-weight as possible. Describe informally how to select the other edges of the spanning tree.

10. Describe an algorithm that, upon input of a weighted graph G and a designated subset S of $E(G)$, finds a minimum-weight subgraph of G that is a connected and spanning subgraph that contains all the edges of S.

11. Here are more details on how to test algorithmically for cycles in KRUSKAL. Initially, in step 4.5, we define the component number of vertex i, denoted $cn(i)$, to be equal to itself, i.

```
STEP 4.5.  For $i = 1$ to $V$ do
               $cn(i) := i$
```

Then in the new step 6, to test an edge $e_j = (x_j, y_j)$ we compare $cn(x_j)$ and $cn(y_j)$. The edge e_j forms a cycle if and only if these component numbers are equal. If they are not equal, we add e_j to T and reset the component numbers of the new component. This is accomplished by the Procedure Renumber.

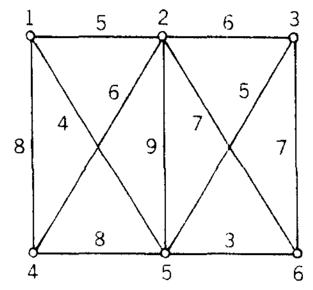
```
STEP 6.  {Suppose $e_j = (x_j, y_j)$.}
         If $cn(x_j) \neq cn(y_j)$, then do
         Begin
         STEP 7.  $T := T + e_j$
         STEP 8.  $k := k + 1$
         STEP 8.5.  Call Procedure Renumber $(x_j, y_j)$
         End {step 6}
```

Here is the procedure:
Procedure Renumber (a, b)

```
STEP 1.  Set $bigcn := \max(cn(a), cn(b))$,
         set $smallcn := \min(cn(a), cn(b))$
STEP 2.  For $i = 1$ to $V$ do
         STEP 3.  If $cn(i) = bigcn$, then
                      $cn(i) := smallcn$
STEP 4.  Return
```

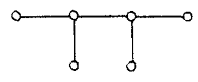
Run KRUSKAL with these additional steps and this procedure on the example in Exercise 2. Keep track of the component numbers at each vertex. Do you get the same spanning tree?

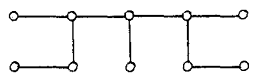
12. Run the extended version of KRUSKAL as given in Exercise 11 on the following graph. Keep track of the component numbers at each vertex.

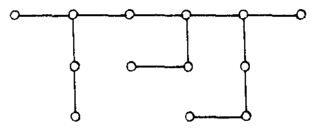


13. Explain why step 6 of the extended KRUSKAL now performs $V + 3 = O(V)$ comparisons and why step 5 performs $O(E\,V)$ comparisons.

14. Within a tree, the **eccentricity** of a vertex x is defined to be the number of edges in a longest path that begins at x. (Or equivalently, it is the maximum value of $d(x, v)$ taken over all vertices v.) For each of the following trees find the eccentricity of each vertex.


(a)


(b)


(c)

15. The **center** of a tree is the set of vertices whose eccentricities are as small as possible. Find the center of each of the trees in Exercise 14.

16. Given a tree T with more than one edge, let $p(T)$ denote the tree obtained from T by erasing all of the leaves of T and their incident edges. For each tree from Exercise 14 find $p(T)$. {Within a tree a vertex of degree one is called a **leaf**. We could say that $p(T)$ is obtained from T by **pruning** all of T's leaves.}

17. If you know the eccentricities of every vertex in a tree T, what can you say (with proof) about the eccentricities of the vertices in $p(T)$ [where $p(T)$ is as defined in Exercise 16]?

18. If T is a tree with more than one edge, show that the center of T equals the center of $p(T)$.

19. Prove that every tree has a center that consists of either one or two vertices.

20. Construct an algorithm that will, upon input of a tree T, find the center of T.

21. Let G be a weighted and connected graph. For x and y in $V(G)$ we define the distance from x to y, $d(x, y)$, to be the length of the shortest path from x to y, where by shortest path we mean that the sum of the edge weights along that path is a minimum among all paths from x to y. Find $d(x, y)$ for each pair of distinct vertices in the graph in Exercise 2.

22. Suppose that T is a minimum-weight spanning tree in G, a weighted and connected graph. Then for x and y in $V(G)$, there is a unique path in T from x to y. We define the tree distance, $dT(x, y)$, to be the sum of the edge weights on that path. Find examples where $dT(x, y) = d(x, y)$, where $d(x, y)$ is as defined in Exercise 21. Then find examples where $dT(x, y) \neq d(x, y)$.

5:5 AN ODE TO GREED

The problem of finding a minimum-weight spanning tree of a graph is typical of a large number of problems in discrete mathematics. In a more general context there is a set of objects with positive numbers assigned to them. The subsets of these objects are partitioned into **desirable subsets** and **undesirable subsets**. We assume that if S is a desirable subset and T is a subset of S, then T is desirable. This property is known as the **hereditary** property. In the particular tree problem of this chapter the objects are the weighted edges. The desirable subsets are those that when considered as subgraphs are acyclic. The undesirable subsets are those that contain cycles. The property of being acyclic is hereditary.

The problem is to find a maximal desirable subset of the objects whose total value is a minimum (or in some cases a maximum). The word **maximal** means that the subset cannot be extended to a larger desirable subset. So a maximal desirable subset of a given set S is first of all a subset of S, second of all it is desirable, and finally it is not properly contained in any desirable subset of S. In the context of the tree problem, a maximal desirable subset of the edges of a graph G is a subset of the edges that is acyclic and not contained in any larger acyclic subgraph. If G is connected, a maximal desirable subset is just a spanning tree. If G is not connected, a maximal desirable subset is a spanning forest, composed of spanning trees of each connected component.

Problem. Given a set of weighted objects E and a partition of the subsets of E into desirable subsets and undesirable subsets such that the property of being desirable is hereditary, find a minimum-weight maximal desirable subset of E.

Algorithm GREEDYMIN

```
STEP 1. Order the objects of $E$ in order of increasing weight; assume $E$ contains $m$ objects $e_1, \ldots, e_m$
STEP 2. Set $j := 1$ {$j$ will index the objects.}
STEP 3. Set $T$ to be empty {$T$ will contain the desirable subset being created.}
STEP 4. Repeat
            Begin
            STEP 5. If $T + e_j$ is desirable, set $T := T + e_j$
            STEP 6. $j := j + 1$
            End
        Until $j > m$
STEP 7. Output $T$ and stop.
```

This algorithm is called **greedy** because at each stage it tries to do as well as it can without regard to what will happen at future steps. Notice that if E is the set of weighted edges in a graph and if desirability is defined as being acyclic, then GREEDYMIN is identical with KRUSKAL. {Actually, KRUSKAL contains an additional stopping criterion that was possible because we knew exactly how many edges a tree has.}

Question 5.1. Construct GREEDYMAX, a greedy algorithm to find a maximum-weight, maximal desirable subset of E. If E is the set of weighted edges in a graph and if desirability is defined as being acyclic, does GREEDYMAX produce a maximum-weight spanning tree?

Example 5.1. Suppose that we greedily attempt to find not a minimum-weight spanning tree but a minimum-weight spanning path. Specifically, we implement GREEDYMIN with desirability defined as follows. S is said to be desirable if S is contained in some path P_{V-1} within the graph G on V vertices. We show in Figure 5.27 a weighted graph G whose greedily chosen path is heavier than the minimum-weight spanning path.

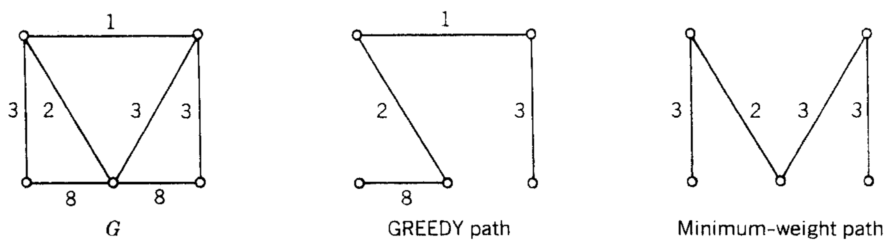


Figure 5.27

This example illustrates the important fact that being greedy does not always produce a best answer, that is, greed does not always pay. In fact, researchers in the field of combinatorial optimization are actively seeking an understanding of just how bad an answer GREEDY will produce for specific applications.

Question 5.2. Suppose that G is a complete weighted graph on V vertices. Further suppose that you wanted to find a minimum-weight cycle C_V as a subgraph of G. Formulate a greedy algorithm to "solve" this problem. Find an example where your algorithm fails to produce the minimum-weight cycle.

This last question is not just whimsy. A variation of this is known as the **Traveling Salesrepresentative Problem**. Suppose that the vertices of G represent a collection of cities and the weight on each edge represents the cost of flying between the two cities. Then an economy-minded salesrepresentative might wish to visit all the cities in a cyclic tour but wants a tour of minimum cost. In fact, no good algorithm is known to solve this problem, or the graph theoretical version in Question 5.2. It is also not known that the problem requires an exponential algorithm. In fact, the problem is computationally equivalent to the Satisfiability Problem introduced in Section 1.10. This area is an active and important one in computer science, operations research, and combinatorics. In Chapter 8 we shall use KRUSKAL to give an approximate solution to this problem.

EXERCISES FOR SECTION 5

1. Here is a new algorithm:

```
STEP 1. Input the weighted graph $G$ with edges $e_1, e_2, \ldots, e_E$ with $w(e_1) \geq$
        $w(e_2) \geq \cdots \geq w(e_E) > 0$
STEP 2. For $j = 1$ to $E$ do
        STEP 3. If $G - e_j$ is connected, set $G := G - e_j$
STEP 4. Output $G$ and stop.
```

   Run this algorithm on the graphs of Exercises 4.1 and 4.2.

2. Describe, in general, for any weighted graph G, the output of the algorithm in Exercise 1. Is this a greedy algorithm?

3. A graph is said to be **unicyclic** if it contains exactly one cycle. Suppose that, given a weighted graph G, we wanted to find a minimum-weight, connected, unicyclic subgraph of G. Does greed pay?

4. Suppose you attempt to find a minimum-weight path using a greedy algorithm with the following criterion of desirability: S is said to be desirable if its edges form a path. Is this GREEDYMIN different from the GREEDYMIN in Example 5.1? If not prove that the two are the same. If they are different, determine whether this GREEDYMIN produces a minimum-weight path.

5. Recall that a property P is called hereditary if whenever S has property P and T is a subset of S, then T has property P. Decide which of the following properties are hereditary:
   (*a*) P is the property that the subset is nonempty.
   (*b*) P is the property that the subset contains an even number of elements.
   (*c*) P is the property that the subset S satisfies $|S| = \lfloor n/2 \rfloor$, where the universe has n elements.
   (*d*) P is the property that S is such that $|S| < n/2$, where the universe has n elements.
   (*e*) P is the property that S does not contain a fixed element x.
   (*f*) P is the property that S contains a fixed element x.
   (*g*) P is the property that S contains at most one of the two elements x and y.

6. Call a subgraph of G desirable if by itself it is a connected graph. In this instance is desirability hereditary? Describe the maximal desirable subgraphs of G.

7. We list some properties that a graph G might have. In each instance if H is a subgraph of G, does H necessarily have the specified property? (That is, is the property hereditary?)
   (*a*) The maximum degree of a vertex in G is less than 7.
   (*b*) G is bipartite.

(*c*) G is a forest.
(*d*) G contains a cycle.
(*e*) G is a complete graph.

8. A graph G is said to be **triangle-free** if G does not contain a 3-clique as a subgraph. Show that being triangle-free is hereditary.

9. Some subgraphs of a graph are induced subgraphs (see Exercise 3.17 for the definition.) Is this a hereditary property?

10. Suppose that you wanted to find a maximum-weight, triangle-free subgraph. Does greed pay?

11. Let G be a graph with $V(G) = \{A, B, C, D, E, F, H\}$. Suppose that G is complete and its edges (in lexicographic order) have weights 1, 4, 14, 4, 15, 21, 2, 3, 2, 3, 3, 1, 3, 5, 2, 2, 2, 5, 2, 17, 1. Find a minimum-weight spanning cycle C_7 that begins and ends at A. What is the cycle that the greedy algorithm produces?

12. Call a subset of $E(G)$ desirable if it is contained in a spanning cycle of G. Show that with this definition of desirable GREEDY will not produce a minimum-weight spanning cycle.

5:6 GRAPHICAL HIGHLIGHTS

Graph theory is a rapidly expanding mathematical discipline. It is important in its own right, as the mathematical basis of many applications, and as a fertile ground for logical and algorithmic thinking. Like the number theory of the previous chapter, graph theory is accessible and concrete. Pictures of graphs make small examples workable; computer programs make large examples tractable. Examples lead to conjectures and ideas for proofs and counterexamples. In fact, this is the effective learning process for both students and research mathematicians.

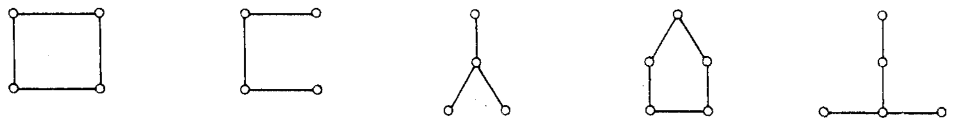
It may seem as if this chapter contains an overwhelming number of definitions. Each is there for a reason relevant to our work. Most definitions are needed immediately to understand Kruskal's algorithm. Others are needed for wide-ranging applications that mathematics and computer science students will meet. A tree may be thought of as the basic underlying structure on which the rest of a connected graph hangs. Trees also arise as a structure used for information storage. These so-called data structures, when formed as a tree, allow for quick retrieval of stored information. For example, most computer operating systems allow for directories, subdirectories, and so on, that are organized by the vertices of a tree.

Our modeling of the Local Area Network with graphs is a true-to-life depiction of how the problems of linking computers and terminals are now being attacked. In fact, computer scientists and electrical engineers regularly look to graphs and their properties to aid them in network design. Telephone companies use graphs, for example, to design systems of switching stations. Their goals are to have short distances between vertices while still having each vertex of small degree. Kruskal's algorithm and other minimum-weight spanning tree algorithms are used in a whole spectrum of applications dealing with transportation systems, commodity flows, and efficient robot manufacturing, as well as the cable connection problem we've seen here. The tree minimizing problem of this chapter provides an introduction to the area of combinatorial optimization. In this field it is now well understood when greedy algorithms work. On the other hand, the search for effective methods to find a minimum-weight spanning cycle in a weighted graph is one of the central problems of mathematics and computer science. Of seemingly intermediate difficulty, the graph isomorphism problem has so far resisted satisfactory solution, yet workers in the field expect this problem to be solved in the near future.

In Chapter 8 we consider more graph theory, both abstract and applied to optimization problems. We also present some approximation algorithms, that is, algorithms that work efficiently but only produce a near-optimal answer.

SUPPLEMENTARY EXERCISES FOR CHAPTER 5

1. Given a graph G, define G^c, the **complement** of G, to be the graph that has $V(G^c) = V(G)$ and $E(G^c) = \{(x, y): (x, y) \notin E(G)\}$. Find the complement of each of the following graphs.



(a) *(b)* *(c)* *(d)* *(e)*

2. If G is isomorphic to H, is G^c isomorphic to H^c? Give a proof or counterexample.
3. If $d_1, d_2, \ldots, d_V$ are the degrees of G, a graph on V vertices, what are the degrees of G^c?
4. What is the largest clique contained in the complement of Q_3? Q_4? (For a definition of Q_n see Exercise 2.19.)
5. We define the **diameter** of a graph to be the maximum value of $d(x, y)$ among all pairs of vertices x and y. Show that if G has diameter 4 or more, then G^c has diameter 2 or less.
6. Suppose that G is a graph with V vertices and E edges and with vertices labeled $\{1, 2, \ldots, V\}$. Then we can list the edges in lexicographic order, as defined in Section 3.3: If each edge e_i is given as a pair of vertices (x_i, y_i), then the edges are numbered and listed in the order $e_1, e_2, \ldots, e_E$ subject to the restrictions that for all i and j with $1 \le i, j \le E$,
   (1) $x_i \le y_i$;
   (2) $i < j$ implies that $x_i \le x_j$; and
   (3) $i < j$ and $x_i = x_j$ implies that $y_i \le y_j$.

(*a*) Explain why the edge list $e_1 = (1,2)$, $e_2 = (1,3)$ and $e_3 = (3,4)$ is in lexicographic order, but the list $f_1 = (1,2)$, $f_2 = (2,4)$, and $f_3 = (2,3)$ is not in lexicographic order.

(*b*) The following edge list is not in lexicographic order. Rearrange it so that conditions (1), (2), and (3) are met:

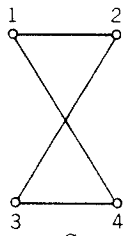
$$\begin{array}{ll} e_1 = (1,2) & e_4 = (5,4) \\ e_2 = (1,3) & e_5 = (5,1) \\ e_3 = (2,3) & e_6 = (4,1). \end{array}$$

7. Here is an algorithm to solve the so-called Labeled Graph Isomorphism Problem.

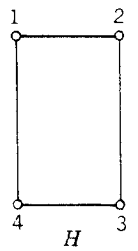
Algorithm LABGPHISO

```
STEP 1. Input $G$ and $H$ with edges in lexicographic order; let $e_i$ denote the
        $i$th edge of $G$ and $f_i$ the $i$th edge of $H$ {Assume $|V(G)| = |V(H)| = V$;
        the vertices are labeled with $1, 2, \ldots, V$, and $|E(G)| = |E(H)| = E$.}
STEP 2. For $j = 1$ to $E$ do {$j$ indexes the edges}
        STEP 3. If $e_j \neq f_j$ {as ordered pairs}, then output "no" and stop.
STEP 4. Output "yes" and stop.
```

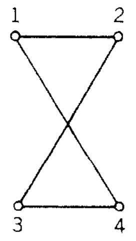
(*a*) Run LABGPHISO on the labeled graphs G and H in the following figure. (Make sure your edge lists are in lexicographic order.)



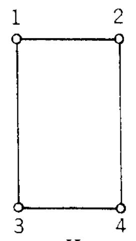
G



H



G



H

(*b*) Explain why the number of comparisons made in LABGPHISO is at most $2E = O(E)$.

8. Here is an idea for solving the more difficult problem of determining graph isomorphism for unlabeled graphs: Fix a labeling of the vertices of G with $1, 2, \ldots, V$. Then run through all permutations of labels of H, and for each permutation run the algorithm LABGPHISO. Design an algorithm GPHISO that uses these ideas. (You may use the algorithms PERM from Chapter 3 and LABGPHISO within your algorithm, without repeating it in its entirety.) Is your algorithm good or bad?

9. Here is an idea to try to speed up the algorithm GPHISO. As seen in Example 2.2 vertices with the same labels must have the same degrees. Thus, for example, if S is the set of all vertices of degree 3 of G and T all vertices of degree 3 of H, then we need to check only $|T|!$ permutations of the labels of T. (The same is true for each degree of vertices in G and H.) Use this idea to redesign GPHISO and then analyze whether this speeds up the algorithm in some or all cases.

10. Suppose that we are given a graph G with $V(G) = \{x_1, \ldots, x_V\}$ and edge list $E(G) = \{(x_1, x_2), \ldots\}$. Suppose that we want to decide if G is bipartite. Begin by placing x_1 in R. Next place each of x_1's neighbors in B, and so on. Construct a precise algorithm BIPARTITE. How many comparisons does BIPARTITE make?

11. What is the maximum number of edges a graph on V vertices can have and still not be connected?

12. For each of the following sequences, either draw a tree whose vertices have these degrees or show that such a tree cannot exist.
(*a*) $\langle 4, 1, 1, 1, 1 \rangle$.
(*b*) $\langle 6, 2, 2, 1, 1, 1, 1, 1 \rangle$.
(*c*) $\langle 5, 2, 2, 1, 1, 1, 1, 1 \rangle$.
(*d*) $\langle 4, 3, 2, 1, 1, 1, 1, 1 \rangle$.
(*e*) $\langle 3, 2, 2, 2, 2, 2, 2, 2, 1 \rangle$.
(*f*) $\langle 3, 3, 3, 1, 1, 1, 1, 1 \rangle$.

13. Give an algorithm that will, given a sequence of positive integers $d_1, \ldots, d_V$ with $d_1 + \cdots + d_V = 2V - 2$, construct a tree whose vertices have the given sequence as its sequence of degrees.

14. Show that in any gathering of people, some pair of people have the same number of acquaintances. (*Hint:* Assume that if A knows B, then B knows A. Think of the graph that could represent acquaintances and try a proof by contradiction.)

15. Prove that if d equals the maximum degree of a vertex in a tree T, then T contains at least d vertices of degree 1.

16. Find all graphs G such that both G and G^c are trees.

17. Let G be a connected graph with edge weights any real numbers. For vertices u and v of G, prove that there is a shortest path between u and v if and only if no path from u to v contains a cycle, the sum of whose edge weights is negative.

18. Here is the idea of Borůvka's original minimum-weight spanning tree algorithm.

```
STEP 1. Input $G$, a connected weighted graph with $n$ vertices
STEP 2. Set $T$ equal to the $n$ vertices of $G$
STEP 3. Repeat
        STEP 4. For each component $C$ of $T$ do
                STEP 5. Select a minimum-weight edge joining a vertex of
                        $C$ with a vertex of $G - C$ and add it to $T$
        Until $T$ is a spanning tree of G
STEP 6. Output $T$ and stop.
```

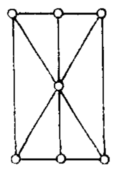
Prove that this algorithm produces a minimum-weight spanning tree of G. Compare this algorithm with KRUSKAL. Find examples where it produces the same and where it produces different minimum-weight spanning trees. What is its complexity?

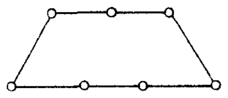
19. Here is another version of a minimum-weight spanning tree algorithm due to Prim.

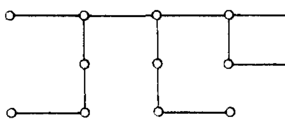
```
STEP 1. Input $G$, a weighted connected graph with $n$ vertices
STEP 2. Set $T = \{v\}$, where $v$ is a vertex of $G$
STEP 3. For $i = 1$ to $n - 1$ do
        STEP 4. Select a minimum-weight edge $e$ joining a vertex $x$ not in $T$
                with a vertex in $T$; set $T = T + e + x$
STEP 4. Output $T$ and stop.
```

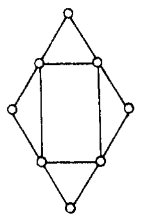
Prove that this algorithm produces a minimum-weight spanning tree of G. Compare the algorithm with KRUSKAL and with Borůvka's algorithm of Exercise 18. What is the complexity of this algorithm?

20. A subset I of the vertex set of a graph G is said to be **independent** if no two vertices in I are joined by an edge in G. The **independence number** of a graph G, denoted by $\boldsymbol{\alpha(G)}$, is defined to be the maximum number of vertices in an independent set in G. Find $\alpha(G)$ for each of the following graphs:

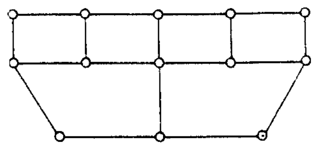

(a)


(b)


(c)



(d)



(e)

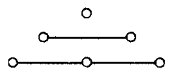
21. Show that if F is a forest, then $\alpha(F) \geq V/2$. Find an example of a forest with $V = 10$ and $\alpha = 5$.

22. Here is an algorithm to find an independent set.

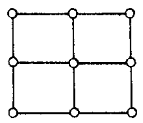
Algorithm IND

```
STEP 1. Input $G$; set $I$ to be empty
STEP 2. While $G$ is nonempty do
        Begin
        STEP 3. Find a vertex $x$ with $\deg(x, G)$ minimum
        STEP 4. Set $I := I + \{x\}$
        STEP 5. Set $G := G - \{x\} - \{y: (x, y) \in E(G)\}$
        End {step 2}
STEP 6. Output $I$ and stop.
```

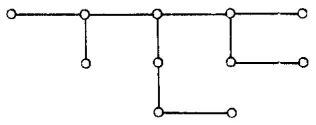
Run IND on each of the following graphs.



(a)



(b)



(c)

23. Prove that Algorithm IND works, that is, it finds an independent set in a graph G.

24. Find an example where IND does not find a maximum independent set.

25. Show that if the input to Algorithm IND is a forest, then the output I will be a maximum independent set.

26. Let A be the $V \times V$ adjacency matrix of a graph. What information does the matrix A^2, the product of A with itself, contain about the graph?

27. Let A be the adjacency matrix of a graph G and let i and j be in $V(G)$. Then prove that the least integer k such that A^k contains a positive entry in the (i,j)th position equals $d(i,j)$, the distance between i and j.

28. Suppose that G is a **regular graph** (i.e., for some fixed constant r, every vertex has degree r). Then the degree of each vertex is the average of the degrees of all adjacent vertices. Prove the converse: Suppose that for every vertex v of a connected graph G

$$\deg(v) = \frac{\deg(x_1) + \cdots + \deg(x_r)}{\deg(v)},$$

where the $x_1, \ldots, x_r$ are all of the vertices adjacent to v. Then prove that G is regular.

29. Suppose that f is a function with domain $V(G)$ and target the real numbers for some connected graph G, and that f satisfies the following property: For every vertex v of $V(G)$

$$f(v) = \frac{f(x_1) + \cdots + f(x_j)}{\deg(v)},$$

where the sum is taken over all vertices x adjacent to v. Then prove that f is a constant function, that is, $f(v) = c$ for some constant c for all v in $V(G)$. Is the result true if G is not connected?

30. Suppose that the number of comparisons made in a graph algorithm A is given by $g(V)$ and that $g(V) > r^V$ for some positive constant r. If $B = V^2$, then show that $r^{(B^{1/2})}$ is a lower bound on the number of comparisons made in the algorithm A. Prove that the function $h(B) = r^{(B^{1/2})}$ is not $O(p(B))$ for any polynomial p and that $h(B) = O(s^B)$ for $s > 1$. What can you conclude about whether or not A is a good algorithm in terms of the input size B?

31. **(a)** Specify where the proof of Theorem 4.1 fails if the edge weights are not all distinct.

(b) Prove Theorem 4.1 in the case that edge weights are not all distinct. (*Hints:* Assume that the edges of any tree produced by KRUSKAL are numbered in the order in which they were selected. Further suppose that F is a minimum-weight spanning tree that has the greatest initial agreement with T. Then complete the proof along the lines of the proof of Theorem 4.1.)

32. Suppose that G is a graph and $\sim$ the corresponding relation on $V(G)$ (as defined in Section 2). For what graphs G is $\sim$ symmetric? Transitive?

33. The **transitive closure** of a graph G is defined to be the graph G' with $V(G') = V(G)$, $E(G) \subseteq E(G')$, and additional edges of $E(G')$ given by: Whenever (a, b) and $(b, c) \in E(G)$, then $(a, c) \in E(G')$. Explain why the corresponding relation $\sim$ defined on $V(G)$ is a transitive relation.

34. Characterize all graphs G such that there is an equivalence relation $\sim$ on a set S whose corresponding graph is G.