Chapter 11

Introduction to Graph Theory

Take a street map of your favorite city[1] and put a bold dot • at each place where two or more streets come together or at a dead-end street. What you get is an example of what is called a (combinatorial) *graph.* Most likely, some of the streets in your favorite city are one-way streets, which permit traffic in only one direction. Put an arrow ($\rightarrow$) on each one-way street, which indicates the permitted direction of traffic flow, and a double arrow ($\leftrightarrow$) on two-way streets. You now have an example of what is called a *directed graph*, or *digraph.* Now consider the people in your favorite city. Run a string between each pair of people that like each other. You have another example of a graph. Recognizing the fact that sometimes one's fondness for another person is not always reciprocated, you may have to put arrows on your strings as you did for streets, with the result being a digraph. Now take your favorite chemical molecule,[2] made up of atoms, some of which are chemically bound to others. You've got another graph, with the bonds playing the role of the streets or strings. Finally, consider all the different types of animals, insects, and plants that inhabit your favorite city. Put an arrow from one type to another, provided the first preys on the second. This time you get a digraph. Two species may share a common prey. Putting a string between each pair that does, you get a graph which that displays competition between species.

As the preceding discussion suggests, graphs and digraphs provide mathematical models for a set of objects that are related or bound together in some way or other. The first paper on graph theory was written by the famous Swiss mathematician Leonhard Euler, in 1736, and dealt with the well-known Königsberg bridge problem. Graph theory has its historic roots in puzzles and games, but today it provides a natural and very important language and framework for investigations in many disciplines, such as networks, chemistry, psychology, social science, ecology, and genetics. Graphs are also some of the most useful models in computer science, since many questions that arise

[1]Mine is Madison, Wisconsin.

[2]Play along, and suppose you do have a favorite chemical molecule!

there can be most easily expressed, investigated, and solved by graph algorithms. We consider graphs in this chapter and digraphs in Chapter 13.

11.1 Basic Properties

A *graph* G (also called a *simple graph*) is composed of two types of objects. It has a finite set

$$V = \{a, b, c, \ldots\}$$

of elements called *vertices* (sometimes also called *nodes*) and a set E of pairs of distinct vertices called *edges*. We denote the graph whose vertex set is V and whose edge set is E by

$$G = (V, E).$$

The number n of vertices in the set V is called the *order* of the graph G. If

$$\alpha = \{x, y\} = \{y, x\}$$

is an edge of G, then we say that α *joins* x and y, and that x and y are *adjacent*; we also say that x and α are *incident*, and y and α are *incident*. We also call x and y the *vertices of the edge* α. A graph is, by definition, an abstract mathematical entity. But we can also think of a graph as a geometrical entity, by representing it with a diagram in the plane. We take one distinct point, a *vertex-point*, for each vertex x (labeling the vertex-point with the vertex) and connect two vertex-points by a simple curve[3] if and only if the corresponding vertices determine an edge α of G. We call such a curve an *edge-curve* and label it with α. In our diagrams, we must take care that *an edge-curve α passes through a vertex-point x only if x is a vertex of the edge α*, for otherwise our diagram will be ambiguous.

Example. Let a graph G of order 5 be defined by

$$V = \{a, b, c, d, e\}$$

and

$$E = \{\{a, b\}, \{b, c\}, \{c, d\}, \{d, a\}, \{e, a\}, \{e, b\}, \{e, d\}\}.$$

A geometric illustration of this graph is shown in Figure 11.1. □

[3] A non-self-intersecting curve.

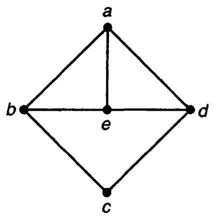

Figure 11.1

If we alter the definition of a graph to allow a pair of vertices to form more than one edge, then the resulting structure is called a *multigraph*. In a multigraph $G = (V, E)$, E is a multiset. The *multiplicity of an edge* $\alpha = \{x, y\}$ is the number of times, $m\{x, y\}$, it occurs in E. The further generalization by allowing *loops*, edges of the form $\{x, x\}$ making a vertex adjacent to itself,[4] is called a *general graph*.

A graph of order n is called *complete*, provided that each pair of distinct vertices forms an edge. Thus, in a complete graph, each vertex is adjacent to every other vertex. A complete graph of order n has $n(n-1)/2$ edges and is denoted K_n. We used this notation in our discussion of Ramsey numbers in Section 2.3.

Example. In Figure 11.2 we have drawn a multigraph of order 4 with nine edges. In Figure 11.3 we have a general graph of order 13 with 21 edges, called *GraphBuster*.[5] □

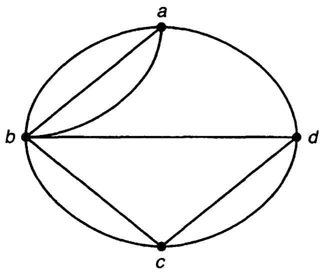

Figure 11.2

[4]Thus, a loop is a multiset consisting of one vertex with repetition number 2.
[5]"Who you gonna call?" **GraphBuster!** (a.k.a. Ghostbuster).

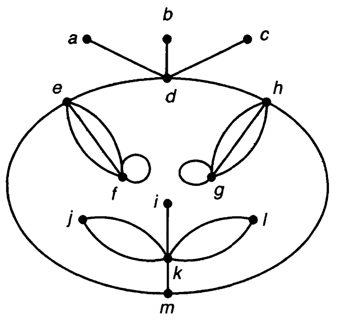

Figure 11.3

Sometimes, in drawing a geometrical representation of a graph (or multigraph or general graph), we may be forced to draw a curve that intersects another.[6]

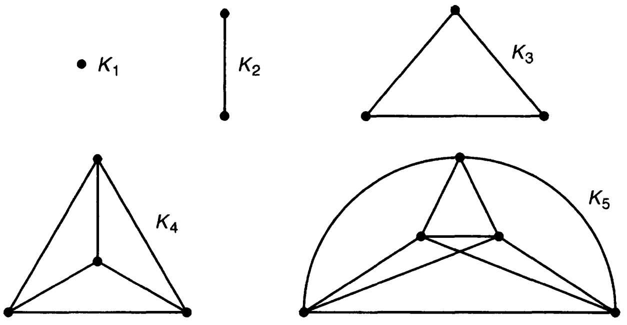

Figure 11.4

Example. The complete graphs K_1, K_2, K_3, K_4, and K_5 are drawn in Figure 11.4. It is not difficult to convince oneself that, in each drawing of K_5, there are always at least two edge-curves which intersect at a point that is not a vertex-point. Another way to draw K_5 is as a pentagon with an inscribed pentagram. □

A general graph G is called *planar*, provided that it can be represented by a drawing in the plane in the manner just described in such a way that two edge-curves intersect only at vertex-points. Such a drawing is called a *plane-graph* and is a *planar representation* of G. The drawings of K_1, K_2, K_3, and K_4 in Figure 11.4 are plane-graphs, and, consequently, those graphs are planar. The drawing of K_5 is not a

[6]But remember our rule that does not allow an edge-curve α to contain a vertex-point x unless vertex x is incident with edge α.

plane-graph, because two edge-curves intersect at a point that is not a vertex-point, and, indeed, K_5 is not planar. Planar graphs are discussed in Chapter 13.

The *degree* (*valence*) of a vertex x in a general graph G is the number $\deg(x)$ of edges that are incident with x. If $\alpha = \{x, x\}$ is a loop joining x to itself, then α contributes 2 to the degree of x.[7] To each general graph G we associate a sequence of numbers that is the list of the degrees of the graph's vertices in nonincreasing order:

$$(d_1, d_2, \ldots, d_n), \quad d_1 \geq d_2 \geq \cdots \geq d_n \geq 0.$$

We call this sequence the *degree sequence* of G.

The degree sequence of the general graph in Figure 11.3 is

$$(6, 5, 5, 5, 5, 5, 3, 2, 2, 1, 1, 1, 1).$$

The degree sequence of a complete graph K_n is

$$(n-1, n-1, \ldots, n-1), \quad ((n-1) \text{ repeated } n \text{ times}).$$

The result stated in Theorem 11.1.1 appeared in Euler's first paper on graphs.

Theorem 11.1.1 *Let G be a general graph. The sum*

$$d_1 + d_2 + \cdots + d_n$$

of the degrees of all the vertices of G is an even number, and, consequently, the number of vertices of G with odd degree is even.

Proof. Each edge of G contributes 2 to the sum of the vertex degrees, 1 to each of its two vertices, or 2 to one vertex in the case of a loop. If a sum of integers is even, then the number of odd integer summands must also be even. □

Example. At a party, a lot of handshaking takes place among the guests. Show that, at the end of the party, the number of guests who have shaken hands an odd number of times is even.

The handshaking at the party can be modeled by a multigraph. The vertices are the guests. Each time two guests shake hands we join them by a new edge. The result is a multigraph to which we can apply Theorem 11.1.1. □

Two general graphs $G = (V, E)$ and $G' = (V', E')$ are called *isomorphic*, provided that there is a one-to-one correspondence

$$\theta : V \to V'$$

between their vertex sets such that, for each pair of vertices x and y of V, there are as many edges of G joining x and y as there are edges of G' joining $\theta(x)$ and $\theta(y)$.

[7]Because both vertices of $\alpha = \{x, x\}$ equal x, α is incident "twice" with x.

The one-to-one correspondence θ is called an *isomorphism* of G and G'. The notion of isomorphism is one of "sameness." Two general graphs are isomorphic if and only if, apart from the labeling of their vertices, they are the same.[8] If G and G' are graphs, then we can express the fact that the two graphs G and G' are isomorphic by asserting that there is a one-to-one correspondence between their vertex sets V and V', such that two vertices of V are adjacent in G if and only if the corresponding vertices are adjacent in G'. This relationship holds because, in graphs, two vertices are joined by either one or zero edges.

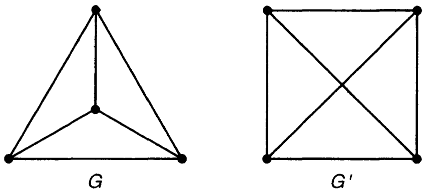

Figure 11.5

Example. Isomorphic graphs have the same order and the same number of edges, but these properties do not guarantee that two graphs are isomorphic.

First, consider the two graphs G and G' shown in Figure 11.5. These graphs are isomorphic since each is a graph of order 4 with each pair of distinct vertices adjacent, and thus each graph is a complete graph of order 4. This example illustrates the fact that a graph may be drawn in various ways (as in this example one drawing may be a plane-graph and the other not) and the actual way in which it is drawn is of no significance insofar as isomorphism is concerned. What matters is only whether two vertices are adjacent or not (or, in the case of general graphs, how many edges join each pair of vertices). □

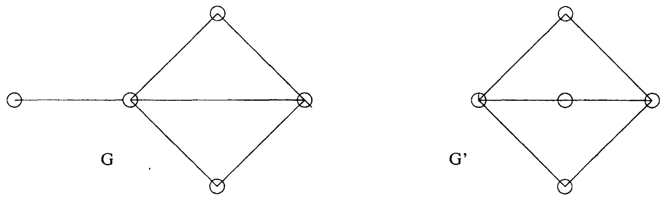

Figure 11.6

Now consider the two graphs G and G' drawn in Figure 11.6. Are these graphs isomorphic? They have the same order and they have the same number of edges. But

[8]Put another way, two general graphs are isomorphic, provided that one is the other in disguise. The one-to-one correspondence θ is the "unmasking" of G' to reveal that G' is really G: If $\theta(x) = x'$, then under the "mask" sits x.

the graph G has a vertex whose degree equals 1, while there is no vertex of G' with degree equal to 1. Such a situation cannot occur if two graphs are isomorphic. For suppose that there is an isomorphism θ between G and G'. Then, for each vertex x of G, the vertex $\theta(x)$ of G' has the same degree as x. In particular, if a number occurs as the degree of a vertex of G, then it must also occur as the degree of a vertex of G'. We conclude that G and G' are not isomorphic. More generally, the same kind of reasoning shows that isomorphic graphs must have the same degree sequence. □

Example. In this example we show that two graphs may not be isomorphic, even if they have the same degree sequence. Consider the two graphs in Figure 11.7. Each of the graphs has degree sequence equal to $(3,3,3,3,3,3)$. Yet these graphs are not isomorphic. This can be seen as follows: In the first graph, G in Figure 11.7, there are three vertices x, y, and z, the members of each pair of which are adjacent.[9] In the second graph, G' of that figure, no set of three vertices has this property. If θ were an isomorphism between the two graphs, then $\theta(x), \theta(y)$, and $\theta(z)$ would be three vertices of G', the members of each pair of which were adjacent. We conclude that G and G' are not isomorphic. □

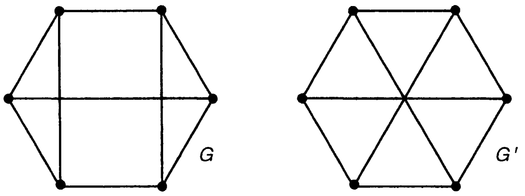

Figure 11.7

We summarize our observations in the next theorem.

Theorem 11.1.2 *Two isomorphic general graphs have the same degree sequence, but two graphs with the same degree sequence need not be isomorphic.*

In the example preceding the theorem, we used another necessary condition for two graphs to be isomorphic. Before recording it, we introduce more basic concepts.

Let $G = (V, E)$ be a general graph. A sequence of m edges of the form

$$\{x_0, x_1\}, \{x_1, x_2\}, \ldots, \{x_{m-1}, x_m\} \tag{11.1}$$

is called a *walk of length* m, and this walk *joins the vertices* x_0 and x_m. We also denote the walk (11.1) by

$$x_0 - x_1 - x_2 - \cdots - x_m. \tag{11.2}$$

[9]They form a K_3.

The walk (11.2) is *closed* or *open* depending on whether $x_0 = x_m$ or $x_0 \neq x_m$. A walk may have repeated edges.[10] If a walk has distinct edges, then it is called a *trail.*[11] If, in addition, a walk has distinct vertices (except, possibly, $x_0 = x_m$), then the walk is called a *path.* A closed path is called a *cycle.* It is easy to show, and is left as an exercise, that the edges of a trail joining vertices x_0 and x_m can be partitioned so that one part of the partition determines a path joining x_0 and x_m, and the other parts determine cycles. In particular, the edges of a closed trail can be partitioned into cycles. The length of a cycle of a graph is at least 3. In a general graph, a loop forms a cycle of length 1, and an edge $\{a, b\}$ of multiplicity $m \geq 2$ determines a cycle $\{a, b\}, \{b, a\}$ (or $a - b - a$) of length 2.

Example. Consider the general graph GraphBuster in Figure 11.3. Then we have the following statements:

(1) $a - d - b - d - c - d - h - g - h - m - k - i$ is a walk of length 11 joining vertex a and vertex i, but it is not a trail.

(2) $a - d - e - f - e - m - k - l - k - i$ is a trail of length 9 joining a and i, but it is not a path.

(3) $a - d - e - m - k - i$ is a path of length 5 joining a and i.

(4) $d - e - f - e - m - h - d$ is a closed trail of length 6, but it is not a cycle.

(5) Each of $f - f$, $e - f - e$, and $d - e - m - h - d$ is a cycle. □

A general graph G is called *connected*, provided that, for each pair of vertices x and y, there is a walk joining x and y (equivalently, a path joining x and y). Otherwise, G is *disconnected.* In a disconnected general graph there is at least one pair of vertices x and y for which there is no way to get from x to y (or from y to x) by "walking" along the edges of G. For most purposes, it suffices to consider only connected graphs. In a connected graph, $d(x, y)$ denotes the shortest length of a walk joining the vertices x and y and is called the *distance* between x and y. We define $d(x, x) = 0$ for each vertex x. It is clear that a walk joining x and y of length $d(x, y)$ is a path.

[10]This comment requires further explanation in case we are dealing with a general graph that is not a graph. In a general graph G, each edge has a multiplicity that may be greater than 1. We do not regard an edge as repeated in a walk if the number of times it occurs in the walk does not exceed its multiplicity. An edge is repeated only if the number of times it occurs in the walk is greater than the number of "copies" available in G. This is perfectly reasonable when we consider a drawing of G, for if an edge $\alpha = \{a, b\}$ has multiplicity 5, say, then in the drawing there are five *different* edge-curves joining the vertex-points a and b.

[11]Thus, in a trail the number of times an edge occurs cannot exceed its multiplicity.

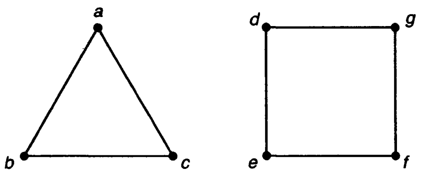

Figure 11.8

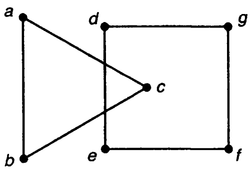

Figure 11.9

Example. The graph drawn in Figure 11.8 is disconnected. There is no walk from vertex a to vertex d. This example illustrates the fact that a disconnected graph can always (and should always!) be drawn so that the resulting geometric entity consists of two disjoint parts. Another way to draw the graph of this example is given in Figure 11.9, but it would be foolish to draw it that way. In general, we try to draw a graph in a way that reveals its structure. □

Let $G = (V, E)$ be a general graph. Let U be a subset of V and let F be a submultiset of E, such that the vertices of each edge in F belong to U. Then $G' = (U, F)$ is also a general graph called a *general subgraph* of G.[12] If F consists of all edges of G that join vertices in U, then G' is called an *induced* general subgraph of G and is denoted by G_U. In case U is the entire set V of vertices of G then G' is called *spanning*. Thus, an induced general subgraph of G is obtained by selecting some of the vertices of G and *all* of the edges of G that join them. A spanning general subgraph is obtained by taking all the vertices of G and *some* (possibly all) of the edges of G.

Example. Let G be the general graph GraphBuster in Figure 11.3. In Figure 11.10, there is given

(1) A general subgraph that is neither induced nor spanning

(2) A general subgraph that is induced but not spanning

[12]If G is a graph (or multigraph), then G' is also a graph (multigraph) and is called a *subgraph* (*submultigraph*). In all definitions like this one, we shall drop the modifier *general* when we are dealing with graphs.

(3) A general subgraph (which happens to be a graph) that is spanning, but not induced. □

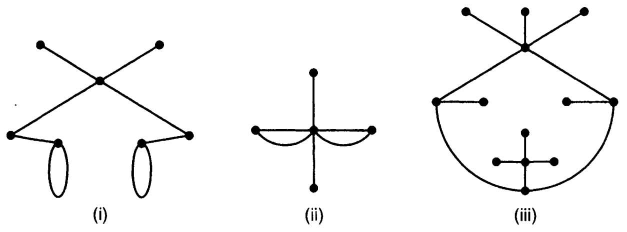

Figure 11.10

The next theorem, which states that a general graph consists of one or more connected general graphs, is clear intuitively. We leave the formal verfication for the Exercises.

Theorem 11.1.3 *Let $G = (V, E)$ be a general graph. Then the vertex set V can be uniquely partitioned into nonempty parts $V_1, V_2, \ldots, V_k$ so that the following conditions are satisfied:*

(1) *The general subgraphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2), \ldots, G_k = (V_k, E_k)$ induced by $V_1, V_2, \ldots, V_k$, respectively, are connected.*

(2) *For each $i \neq j$ and each pair of vertices x in V_i and y in V_j, there is no walk that joins x and y.*

The general graphs $G_1, G_2, \ldots, G_k$ in Theorem 11.1.3 are the *connected components* of G. Part (1) of the theorem says that the connected components are indeed connected; part (2) asserts that the connected components are *maximal* connected induced general subgraphs; that is, for each i and for each set U of vertices, such that V_i is contained in U but $V_i \neq U$, the general subgraph induced by U is disconnected.

In the next theorem we formulate additional necessary conditions in order that general graphs be isomorphic. Its proof should now be obvious, and formal verification is left for the Exercises.

Theorem 11.1.4 *Let G and G' be two general graphs. Then the following are necessary conditions for G and G' to be isomorphic:*

(1) *If G is a graph, so is G'.*

(2) *If G is connected, so is G'. More generally, G and G' have the same number of connected components.*

(3) *If G has a cycle of length equal to some integer k, then so does G'.*

(4) *If G has an (induced) general subgraph that is a complete graph K_m of order m, so does G'.*

The graphs G and G' in Figure 11.7 are not isomorphic since one has a cycle of length 3 (a subgraph isomorphic to K_3) and the other doesn't.

We conclude this section by showing that a general graph may also be described by a matrix whose entries are nonnegative integers.

Let G be a general graph of order n and let its vertices be, in some order, $a_1, a_2, \ldots, a_n$. Let A be the n-by-n array such that the entry a_{ij} in row i, column j equals the number of edges joining the vertices a_i and a_j, $(1 \leq i, j \leq n)$. We always have[13] $a_{ij} = a_{ji}$, and a_{ii} counts the number of loops at vertex a_i. The matrix A is called the *adjacency matrix* of G. In case G is a graph, then A is a matrix of 0s and 1s and the entry a_{ij} equals 1 if and only if a_i and a_j are adjacent in G.

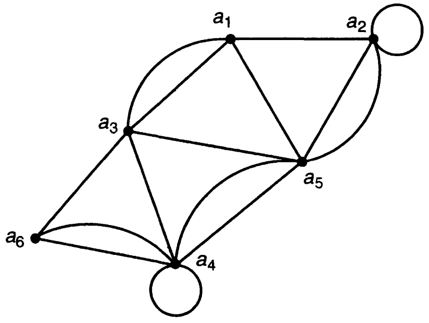

Figure 11.11

Example. Figure 11.11 shows a general graph of order 6 whose 6-by-6 adjacency matrix is

$$\begin{bmatrix} 0 & 1 & 2 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 2 & 2 \\ 1 & 2 & 1 & 2 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \end{bmatrix}.$$

We can start with either the general graph or the adjacency matrix and then construct the other. □

The adjacency matrix is uniquely determined by a general graph, apart from the ordering of its rows and columns. This is because, before we can form the adjacency matrix, we must first list the vertices in some order. Conversely, the adjacency matrix

[13]The matrix is *symmetric*.

of a general graph uniquely determines the general graph up to isomorphism; that is, any two general graphs with the same adjacency matrix are isomorphic.

11.2 Eulerian Trails

In his paper on graph theory published in 1736, Euler solved the now famous *Königsberg bridge problem*:

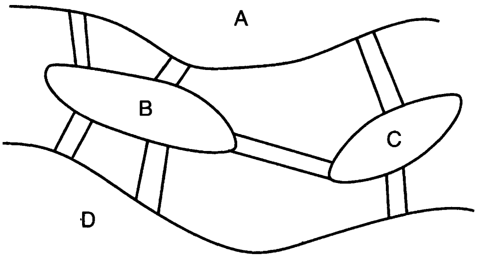

Figure 11.12

> The old city of Königsberg in East Prussia was located along the banks and on two islands of the Pregel River, with the four parts of the city connected by seven bridges as shown in Figure 11.12. On Sundays, the citizens of Königsberg would promenade about town, and the problem arose as to whether it was possible to plan a promenade so that each bridge is crossed once and only once, ending the promenade where it began.

Euler replaced the map of Königsberg with the general graph G drawn in Figure 11.13. In terms of G and the terminology we have now introduced, the problem is to determine whether there exists a closed trail that contains all the edges of G.

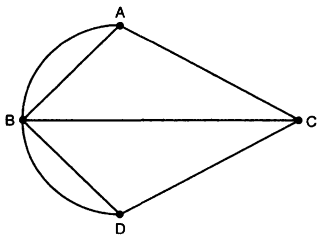

Figure 11.13

Example. Consider the plight of the mail carrier[14] who, starting at the post office, wishes to deliver the mail to the houses on the preassigned streets and then end up back at the post office at the end of the day. What the mail carrier would like is a way to deliver all the mail without having to walk over any street after having already delivered the mail on that street. Can we help the mail carrier?

Well, maybe we can and maybe we can't. But we surely should recognize his or her problem as a problem in graph theory. Let G be the general graph that can be associated with the street map of a city. (See the introductory remarks for this chapter.) Let G' be the general subgraph consisting of the vertices and edges of G that correspond to the mail carrier's assigned streets. The mail carrier desires a closed trail in G' that contains each edge of G' exactly once. Thus, we have the same mathematical problem as the citizens of Königsberg had over 200 years ago, but relative to a different general graph. □

Motivated by these problems, we make some definitions. A trail in a general graph G is called *Eulerian*, provided that it contains every edge of G. Recall that a trail in a general graph by definition contains each edge at most once, where we interpret this to mean that the number of times that an edge occurs on the trail does not exceed its multiplicity. Both the citizens of Königsberg and the mail carrier seek a closed Eulerian trail. We can easily see that the Königsberg bridge general graph in Figure 11.13 does not have a closed Eulerian trail. We reason as follows: Imagine actually promenading on a closed Eulerian trail in a general graph. Except for the first time you leave the vertex at which you begin, every time you go into a vertex you leave it (by a new edge; that is, by one that you had not yet gone over). When you finish up, you go into the beginning vertex but don't leave it. This means that the edges which are incident with a given vertex can be paired up: One edge of each pair is used to enter the vertex and one is used to leave it.[15] If the edges incident with a vertex can be paired up, that means that there must be an even number of edges at each vertex. We thus conclude that for a general graph to have a closed Eulerian trail, the degree of each vertex must be even. Since the four vertices of the general graph for the Königsberg bridge problem have odd degree, the graph does not have a closed Eulerian trail.

Theorem 11.2.2 asserts that the necessary condition for a closed Eulerian trail derived in the preceding discussion is also sufficient for a connected general graph. Before proving it, we establish a lemma, which is also of independent interest.

Lemma 11.2.1 *Let $G = (V, E)$ be a general graph and assume that the degree of each vertex is even. Then each edge of G belongs to a closed trail and hence to a cycle.*

[14]Change mail carrier to street sweeper or snowplow operator to obtain a different formulation of the same mathematical problem.

[15]If we think of starting our promenade in the "middle" of an edge, then we do not need to distinguish a beginning vertex: Each time we enter a vertex we also leave it.

Proof. We can find a closed trail containing any prescribed edge $\alpha_1 = \{x_0, x_1\}$ using the next algorithm. In this algorithm, we construct a set W of vertices and a set F of edges.

Algorithm for a closed trail

```
(1) Put $i = 1$.
(2) Put $W = \{x_0, x_1\}$.
(3) Put $F = \{\alpha_1\}$.
(4) While $x_i \neq x_0$, do the following:
    (a) Locate an edge $\alpha_{i+1} = \{x_i, x_{i+1}\}$ not in $F$.
    (b) Put $x_{i+1}$ in $W$ ($x_{i+1}$ may already be in $W$).
    (c) Put $\alpha_{i+1}$ in $F$.
    (d) Increase $i$ by 1.
```

Thus, after the initialization in (1)–(3), at each stage of the algorithm we locate a new edge[16] $\alpha_{i+1} = \{x_i, x_{i+1}\}$ incident with the most recent vertex x_i put in W, add x_{i+1} to W and α_{i+1} to F, and then increase i by 1 and repeat until we finally arrive at x_0 again.

Suppose that an edge α_{i+1} satisfying (4)(a) exists whenever $x_i \neq x_0$. Let the terminal value of i be k, giving the set $W = \{x_0, x_1, \ldots, x_k\}$ of vertices and the multiset $F = \{\alpha_1, \ldots, \alpha_k\}$ of edges. It then follows from the description of the algorithm that

$$\alpha_1, \ldots, \alpha_k \tag{11.3}$$

is a closed trail containing the initial edge α_0. Thus, we have only to show that, if $x_i \neq x_0$, then there is an edge not in F that is incident with x_i. It is here where the hypothesis of even degrees comes in.

It is readily seen that, at the end of each step (4)(d) of the algorithm, each vertex of the general graph $H = (W, F)$ has even degree, except possibly for the vertex x_0 (which starts out with odd degree 1) and the most recent new vertex x_i (whose degree has just been increased by 1). Moreover, x_0 and x_i have even degree if and only if $x_0 = x_i$. Thus, if $x_i \neq x_0$, x_i has odd degree in the general graph H. Since x_i has even degree in G, there must be an edge $\alpha_{i+1} = \{x_i, x_{i+1}\}$ not yet in F that is incident with x_i. Thus, at the end of the algorithm, $x_k = x_0$ and (11.3) is a closed trail.

The edges of a closed trail can be partitioned into cycles, and the proof of the lemma is complete. □

[16]More precisely, one whose multiplicity in F is less than that in the edge set E of our graph G.

Example. We apply the algorithm for a closed trail to the general graph G drawn in Figure 11.14. One way to carry out the algorithm[17] is illustrated in the following table, where the initial edge is $\{a, b\}$:

i	x_i	α_i	W	F
1	b	$\{a,b\}$	a,b	$\{a,b\}$
2	c	$\{b,c\}$	a,b,c	$\{a,b\},\{b,c\}$
3	d	$\{c,d\}$	a,b,c,d	$\{a,b\},\{b,c\},\{c,d\}$
4	b	$\{d,b\}$	a,b,c,d	$\{a,b\},\{b,c\},\{c,d\},\{d,b\}$
5	h	$\{b,h\}$	a,b,c,d,h	$\{a,b\},\{b,c\},\{c,d\},\{d,b\},$ $\{b,h\}$
6	a	$\{h,a\}$	a,b,c,d,h	$\{a,b\},\{b,c\},\{c,d\},\{d,b\},$ $\{h,b\},\{h,a\}$

We thus obtain the closed trail

$$\{a,b\},\{b,c\},\{c,d\},\{d,b\},\{h,b\},\{h,a\}$$

and the cycle

$$\{a,b\},\{b,h\},\{h,a\}$$

containing the edge $\{a, b\}$. □

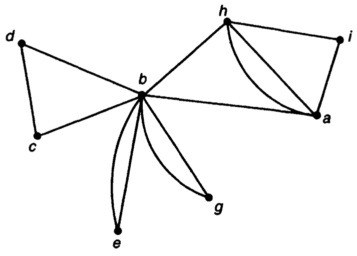

Figure 11.14

Theorem 11.2.2 *Let G be a connected general graph. Then G has a closed Eulerian trail if and only if the degree of each vertex is even.*

Proof. We have already observed that, if G has a closed Eulerian trail, then each vertex has even degree.

Now assume that every vertex of G has even degree, and let $G_1 = (V, E_1)$ be the gtraph G. We choose any edge α_1 of G_1 and apply the algorithm for a closed trail

[17]Since at each stage of the algorithm there may be more than one choice for a new edge, there will, in general, be many ways in which to carry out the algorithm.

given in the proof of Lemma 11.2.1 to obtain a closed trail γ_1 containing the edge α_1. Let $G_2 = (V, E_2)$ be the general graph obtained by removing from E_1 the edges that belong to the closed trail γ_1. All vertices have even degree in G_2. If E_2 contains at least one edge, then since we started with G_1 connected, there must be an edge α_2 of G_2 that is incident with a vertex z_1 on the closed trail γ_1. We apply the algorithm for a closed trail to G_2 and the edge α_2 and obtain a closed trail γ_2 containing the edge α_2. We now patch[18] γ_1 and γ_2 together at the vertex z_1 and obtain a closed trail $\gamma_1 \overset{z_1}{*} \gamma_2$ that includes all the edges of both γ_1 and γ_2. Let $G_3 = (V, E_3)$ be the general graph obtained by removing the edges of γ_2 from E_2. If E_3 contains at least one edge, then it contains an edge α_3 which is incident with a vertex z_2 on the closed trail $\gamma_1 \overset{z_1}{*} \gamma_2$. We apply the algorithm for a closed trail to G_3 and the edge α_3 and obtain a closed trail γ_2 containing the edge α_3. We then patch $\gamma_1 \overset{z_1}{*} \gamma_2$ and γ_3 together at vertex z_2 and obtain the closed trail $\gamma_1 \overset{z_1}{*} \gamma_2 \overset{z_2}{*} \gamma_3$, which[19] includes all the edges of γ_1, γ_2 and γ_3. We continue like this until all edges have been included in a closed trail $\gamma_1 \overset{z_1}{*} \gamma_2 \overset{z_2}{*} \cdots \overset{z_{k-1}}{*} \gamma_k$. Thus, repeated calls to our algorithm for a closed trail give an algorithm to construct a closed Eulerian trail in a connected general graph, each of whose vertices has even degree. □

Example. We continue with the preceding example and obtain a closed Eulerian trail in the general graph G of Figure 11.14, using the algorithm in the proof of Theorem 11.2.2. Since the algorithm requires us to make choices, there are several ways to carry out the algorithm. One possible result is the following:

$$\gamma_1 = a - b - c - d - b - h - a,$$
$$\gamma_2 = b - e - b, (z_1 = b),$$

$$\gamma_1 \overset{b}{*} \gamma_2 = a - b - e - b - c - d - b - h - a,$$

$$\gamma_3 = b - g - b, (z_2 = b),$$

$$\gamma_1 \overset{b}{*} \gamma_2 \overset{b}{*} \gamma_3 = a - b - g - b - e - b - c - d - b - h - a,$$

$$\gamma_4 = h - i - a - h, (z_3 = h),$$

$$\gamma_1 \overset{b}{*} \gamma_2 \overset{b}{*} \gamma_3 \overset{h}{*} \gamma_4 = a - b - g - b - e - b - c - d - b - h - i - a - h - a.$$

□

[18]We traverse γ_1 until we first come to the vertex z_1, completely traverse γ_2 ending up at vertex z_1, and then finish our traversal of γ_1.

[19]This notation is a little ambiguous. Do you know why?

Theorem 11.2.2 and its proof furnish a characterization of general graphs with a closed Eulerian trail and an algorithm for constructing a closed Eulerian trail if one exists. For an open Eulerian trail we have the next theorem.

Theorem 11.2.3 *Let G be a connected general graph. Then G has an open Eulerian trail if and only if there are exactly two vertices u and v of odd degree. Every open Eulerian trail in G joins u and v.*

Proof. First, we recall from Theorem 11.1.1 that the number of vertices of G of odd degree is even. If there is in G an open Eulerian trail, then it must join two vertices u and v of G of odd degree, and every other vertex of G must have even degree (since each time the Eulerian trail goes into a vertex x different from u and v it leaves, resulting in a pairing of the edges incident with x). Now assume that G is connected and has exactly two vertices u and v of odd degree. Let G' be the general graph obtained from G by adding a new edge $\{u, v\}$ joining u and v. Then G' is connected and each vertex now has even degree. Hence, by Theorem 11.2.2, G' has an Eulerian trail γ'. We can think of γ' as beginning at the vertex v with first edge being the new edge $\{u, v\}$ joining u and v. Removing this edge from γ' and starting at the vertex u, we obtain an open Eulerian trail γ in G joining u and v. We can apply the algorithm for a closed Eulerian trail to G' and thereby obtain an algorithm for an open Eulerian trail in G. □

The previous theorem is further generalized in the next theorem. We leave the proof for the Exercises.

Theorem 11.2.4 *Let G be a connected general graph and suppose that the number of vertices of G with odd degree is $m > 0$. Then the edges of G can be partitioned into $m/2$ open trails. It is impossible to partition the edges of G into fewer than $m/2$ open trails.*

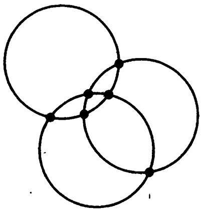

Figure 11.15

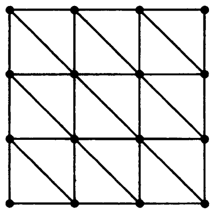

Figure 11.16

Example. Consider the graphs drawn in Figures 11.15, 11.16, and 11.17. Is it possible to trace these plane graphs with a pencil without removing the pencil from the paper?

To trace a plane graph without removing our pencil from the paper, it is necessary and sufficient that there is an Eulerian trail, either open or closed. The vertices of the graph drawn in Figure 11.15 all have degree equal to 4 and hence, by Theorem 11.2.2, the graph is traceable. The graph drawn in Figure 11.16 has two vertices of odd degree and hence, by Theorem 11.2.3, has an open Eulerian trail joining the two vertices of odd degree. The graph drawn in Figure 11.17 has four vertices of odd degree and hence, by Theorem 11.2.3, is not traceable. However it follows from Theorem 11.2.4 that this graph can be traced if we are allowed to lift the pencil once from the paper. The proof of Theorem 11.2.2 contains an algorithm to trace a plane graph when a tracing exists. □

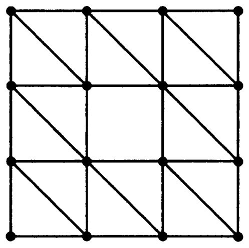

Figure 11.17

By Theorem 11.2.4, if a general graph G has $m > 0$ vertices of odd degree, then the edges can be partitioned into $m/2$ open trails, each trail joining two vertices of odd degree. If we want to trace out G as discussed in the previous example, then it is necessary to lift the pencil only $(m/2) - 1$ times. In tracing out G, lifting the pencil is no great hardship, but if G represents the route of a mail carrier (as discussed in the example at the beginning of this section) who has to deliver mail on foot on each of the streets corresponding to the edges of G, then what's the mail carrier to do? Fly? If the mail carrier's route does not contain a closed Eulerian trail, then in order for all the mail to be delivered and for the mail carrier to return to the post office, the mail carrier will have to walk over some streets more than once. How can we minimize the number of streets that the mail carrier will have to walk over after already having delivered the mail at the houses on those streets? This problem is known as the *Chinese postman problem.*[20] A precise formulation is the following:

> *Chinese postman problem:* Let G be a connected general graph. Find a closed walk of shortest length which uses each edge of G at least once.[21]

We close this section with a simple observation concerning the solution of the Chinese postman problem.

Theorem 11.2.5 *Let G be a connected general graph having K edges. Then there is a closed walk in G of length $2K$ in which the number of times an edge is used equals twice its multiplicity.*

Proof. Let G^* be the general graph obtained from G by doubling the multiplicity of each edge of G. Then G^* is a connected graph with $2K$ edges. Moreover, each vertex of G^* has even degree (twice its degree in G). Applying Theorem 11.2.2 to G^*, we see that G^* has a closed Eulerian trail. This closed trail in G^* is a closed walk in G of the required type. □

Example. Consider a graph G with vertices $1, 2, \ldots, n$ and edges $\{1,2\}, \{2,3\}, \ldots, \{n-1, n\}$. Thus, the edges of G form a path joining vertex 1 to vertex n. Any closed walk in G that includes each edge must include each edge at least twice. Thus, if the post office is at vertex k, our Chinese postman can do no better than to walk to vertex 1, retrace his steps back to the post office, walk to vertex n, and retrace his steps back to the post office. The length of such a walk is $2(n-1)$, that is, twice the number of edges. The graph G is a simple instance of a tree. Trees are studied in Sections 11.5 and 11.7. For a tree, the smallest length of a closed walk that includes each edge at least once equals twice the number of edges. (See Exercise 78.) □

While the Chinese postman problem, as we have phrased it, may be interesting from a purely mathematical point of view, it is not very practical. This is because we have not taken into account the length of the streets. Some streets may be very long, while others are very short. If the mail carrier has to repeat some streets, obviously the shorter ones are to be preferred. To make the problem practical, we should attach a nonnegative *weight* to each edge and then measure a walk not by its length (the number of its edges) but by its total weight (the sum of the weights of its edges,

[20]Not because it has particular relevance to China, but because it was introduced by the Chinese mathematician M. K. Kwan in a paper, Graphic Programming Using Odd or Even Points, *Chinese Math.*, 1 (1962), 273–277.

[21]A solution to this problem is given in J. Edmonds and E. L. Johnson, Matching, Euler Tours and the Chinese Postman, *Math. Programming*, 5 (1973), 88–124.

counting the weight of an edge the number of times that it is used in the walk). The practical Chinese postman problem is to determine a walk of smallest weight which includes each edge at least once. This problem has also been solved satisfactorily from an algorithmic point of view.[22]

11.3 Hamilton Paths and Cycles

In the nineteenth century, Sir William Rowan Hamilton invented a puzzle whose object was to determine a route on the sides of a dodecahedron[23] that started at some corner and returned there after having visited every other corner exactly once. The corners and sides of a dodecahedron determine a graph with 20 vertices and 30 edges, which is drawn in Figure 11.18. There are many readily discovered solutions to Hamilton's puzzle.[24]

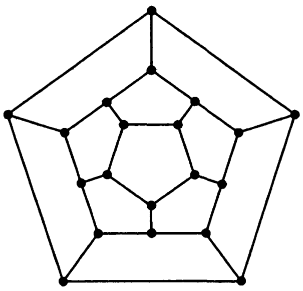

Figure 11.18

Hamilton's puzzle can be formulated for any graph:

> Given a graph G, can one determine a route along the edges of G that begins at some vertex and then returns there after having visited every other vertex exactly once?

Today, a solution to Hamilton's puzzle for a graph G is called a Hamilton cycle. More precisely, a *Hamilton cycle* of a graph G of order n is a cycle of G of length n. Hence, a Hamilton cycle in the graph G of order n is a cycle

$$x_1 - x_2 - \cdots - x_n - x_1$$

[22]Ibid.

[23]The dodecahedron is one of the regular solids. It is bounded by 12 regular pentagons which come together at 30 sides, determining 20 corner points.

[24]And this perhaps explains why Hamilton's puzzle was not a great commercial success.

of length n, where $x_1, x_2, \ldots, x_n$ are the n vertices of G in some order. A *Hamilton path* in G joining vertices a and b is a path

$$a = x_1 - x_2 - \cdots - x_n = b$$

of length $n-1$ of G. Thus, a Hamilton path in G is given by a permutation of the n vertices of G in which consecutive vertices are joined by an edge of G. The Hamilton path joins the first vertex of the permutation to the last. The edges of a Hamilton path and of a Hamilton cycle are necessarily distinct.

We can also consider Hamilton paths and cycles in general graphs, but higher multiplicities of edges have no impact on the existence and nonexistence of Hamilton paths and cycles. Whether or not there is a Hamilton path or Hamilton cycle is determined solely by which pairs of vertices are joined by an edge and not by the multiplicity of an edge joining a pair of vertices. For this reason. *we consider only graphs, and not general graphs, in this section.*

Example. A complete graph K_n of order $n \geq 3$ has a Hamilton cycle. In fact, since each pair of distinct vertices of K_n forms an edge, each permutation of the n vertices of K_n is a Hamilton path. Since the first vertex and last vertex are joined by an edge, each Hamilton path can be extended to a Hamilton cycle. We thus see that K_n has $n!$ Hamilton paths and $(n-1)!$ Hamilton cycles (corresponding to circular permutations of length n). □

Example. For each of the two graphs drawn in Figure 11.19, determine whether there is a Hamilton path or cycle.

First, consider the graph on the left. Then $a - b - c - d - f - e - a$ is a Hamilton cycle, and thus $a - b - c - d - f - e$ is a Hamilton path. Another Hamilton path is $a - b - c - d - e - f$, but this Hamilton path cannot be extended to a Hamilton cycle since a and f are not joined by an edge.

Now consider the "dumbbell" graph on the right. A Hamilton path is $a-b-c-d-e-f$, but this graph does not have a Hamilton cycle. The reason is that a Hamilton cycle is closed, and thus would have to cross the "bar" of the dumbbell twice, but this is not allowed in a Hamilton cycle. □

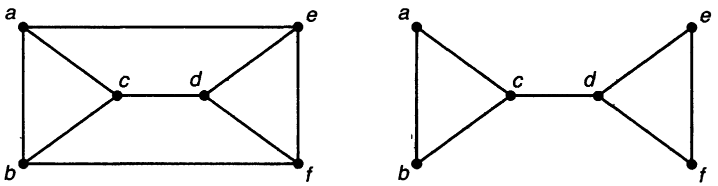

Figure 11.19

At first glance, the question of the existence of a Hamilton cycle in a graph seems similar to the question of the existence of a closed Eulerian trail in a graph. For the latter, we seek a closed trail that includes every edge exactly once. For the former, we seek a closed path that includes every vertex exactly once. Beyond this superficial resemblance, the two questions are very much different. In Theorem 11.1.1 an easily verifiable characterization of (general) graphs with a closed Eulerian trail is given, and we have a satisfactory algorithm for constructing one when those conditions are met. No such characterization exists for graphs with a Hamilton cycle, nor is there a satisfactory algorithm for constructing a Hamilton cycle in a graph, should one exist. The problem of the existence and construction of Hamilton cycles (and paths) in graphs has been investigated quite extensively and continues as a major research question in graph theory.

So if we cannot characterize graphs with Hamilton cycles (that is, find conditions which are both necessary and sufficient for their existence in a graph), we must be content to find conditions that are sufficient for their existence (that is, guarantee a Hamilton cycle) and, separately, conditions that are necessary for their existence (so if they are not met, guarantee that there is no Hamilton cycle). One obvious necessary condition for a Hamilton cycle is that the graph must be connected. Another less obvious condition was hinted at in our analysis of the dumbbell graph in Figure 11.19.

An edge of a connected graph is called a *bridge*, provided its removal from the graph leaves a disconnected graph. In a certain sense, a connected graph with a bridge is just barely connected: Remove the bridge and the graph "breaks apart." The bar of the dumbbell graph in Figure 11.19 is a bridge.

Theorem 11.3.1 *A connected graph of order $n \geq 3$ with a bridge does not have a Hamilton cycle.*[25]

Proof. Suppose that $\alpha = \{x, y\}$ is a bridge of a connected graph G. Let G' be the graph obtained from G by removing the edge α but not any vertices. Since G is connected, G' has two connected components.[26] Suppose G has a Hamilton cycle γ. Then γ would, say, begin in one of the components in G'; would eventually cross over to the other, via α; and then would have to cross back to the first, also via α. But then γ is not a Hamilton cycle since it would include the edge α twice (in fact, G cannot even have an Eulerian cycle). □

We now discuss a simple sufficent condition for a Hamilton cycle in a graph which is due to Ore.[27]

Let G be a graph of order n, and consider the following property which may or may not be satisfied in G:

[25] Although it might have a Hamilton path.

[26] If G' had more than two connected components, then putting the edge α back could only combine two of these components and the resulting graph (namely, G) would be disconnected, contrary to assumption.

[27] O. Ore, A Note on Hamilton Circuits, *Amer. Math. Monthly*, 67 (1960), 55.

Ore property: For all pairs of distinct vertices x and y that are not adjacent,

$$\deg(x) + \deg(y) \geq n.$$

What are the implications for a graph which satisfies the Ore property? A graph all of whose vertices have "large" degree[28] must have a lot of edges, and these edges are distributed somewhat uniformly throughout the graph. We would hope that such a graph would have a Hamilton cycle.[29] Suppose, for instance, that G is a graph with $n = 50$ vertices which satisfies the Ore property. If G had a vertex x of small degree, say, 4, this would imply that there are 45 vertices different from x that are not adjacent to x. By the Ore property, each of these 45 vertices has degree at least 46. Thus, the Ore property implies either that all vertices have large degree or that there are some vertices of small degree and *very* many vertices of *very* large degree. Therefore, the Ore property compensates for the possible presence of vertices of small degree (which might keep a graph from having a Hamilton cycle) by forcing there to be a lot of vertices of high degree (which might help a graph to have a Hamilton cycle).

Theorem 11.3.2 *Let G be a graph of order $n \geq 3$ that satisfies the Ore property. Then G has a Hamilton cycle.*

Proof. Suppose that G is not connected. We then show that G cannot satisfy the Ore property. Since G is not connected, its vertices can be partitioned into two parts, U and W, in such a way that there are no edges joining a vertex in U with a vertex in W. Let r be the number of vertices in U and let s be the number of verticed in W. Then $r + s = n$, and each vertex in U has degree at most $r - 1$, and each vertex in W has degree at most $s - 1$. Let x be any vertex in U and let y be any vertex in W. Then x and y are not adjacent, but the sum of their degrees is, at most,

$$(r - 1) + (s - 1) = r + s - 2 = n - 2,$$

and this contradicts the Ore property. We conclude that if G satisfies the Ore property, then G must be connected.

To complete the proof of the theorem, we describe an algorithm[30] for constructing a Hamilton cycle in a graph that satisfies the Ore property.

Algorithm for a Hamilton cycle

(1) Start with any vertex and, by attaching adjacent vertices at either end, construct a longer and longer path until it is not possible to make it any longer. Let the path be

$$\gamma : y_1 - y_2 - \cdots - y_m.$$

[28]This will be made precise in Corollary 11.3.3.

[29]If having a lot of edges well distributed over the graph did not guarantee a Hamilton cycle, what chance would we ever have of finding a condition that would?

[30]The algorithm is implicit in Ore's proof of Theorem 11.3.2 and was explicitly formulated by M. O. Albertson.

(2) Check to see if y_1 and y_m are adjacent.

(i) If y_1 and y_m are not adjacent, go to (3). Else y_1 and y_m are adjacent, and go to (ii).

(ii) If $m = n$, then stop and output the Hamilton cycle

$$y_1 - y_2 - \cdots - y_m - y_1.$$

Else, y_1 and y_m are adjacent and $m < n$, and go to (iii).

(iii) Locate a vertex z not on γ and a vertex y_k on γ such that z is adjacent to y_k. Replace γ with the path of length $m+1$ given by

$$z - y_k - \cdots - y_m - y_1 \cdots - y_{k-1},$$

and go back to (2).

(3) Locate a vertex y_k with $1 < k < m$ such that y_1 and y_k are adjacent and y_{k-1} and y_m are adjacent. Replace γ with the path

$$y_1 - \cdots - y_{k-1} - y_m - \cdots - y_k.$$

The two ends of this path, namely, y_1 and y_k, are adjacent, and go back to (2)(ii).

To prove that the algorithm does construct a Hamilton cycle when the Ore property holds, we have to show that in (2)(iii) we can locate the specified vertex z, and in (3) we can locate the specified vertex y_k.

First, consider (2)(iii). We have $m < n$. Since we have already shown that the Ore property implies that G is connected, there must be some vertex z not on the cycle γ which is adjacent to one of the vertices $y_1, \ldots, y_m$.

Now consider (3). We know that y_1 and y_m are not adjacent. Let the degree of y_1 be r and let the degree of y_m be s. By the Ore property, we have $r + s \geq n$. Since γ is a longest path from step (1), y_1 is adjacent to only vertices on γ and hence to r of the vertices $y_2, \ldots, y_{m-1}$. Similarly, y_m is adjacent to s of the vertices $y_2, \ldots, y_{m-1}$. Each of the r vertices joined to y_1 is preceded in the path γ by some vertex, and one of these must be adjacent to y_m. For, if not, then y_m is adjacent to at most $(m-1) - r$ vertices and hence $s \leq m - 1 - r$. This means that

$$r + s \leq m - 1 \leq n - 1,$$

contrary to the Ore property. Thus, there is a vertex y_k such that y_1 is adjacent to y_k and y_m is adjacent to y_{k-1}. Hence, the algorithm stops after having constructed a Hamilton cycle in G. □

One way to guarantee the Ore property in a graph is to assume that all vertices have degree equal to or greater than half the order of the graph. This results in a theorem of Dirac,[31], which was proved in 1952 before Theorem 11.3.2 but now is a consequence of it.

Corollary 11.3.3 *A graph of order $n \geq 3$, in which each vertex has degree at least $n/2$, has a Hamilton cycle.*

A proof with algorithm similar to that given for Theorem 11.3.2 can be constructed for the next theorem in which a sufficient condition is given for a Hamilton path in a graph. We leave the proof as an exercise.

Theorem 11.3.4 *A graph of order n, in which the sum of the degrees of each pair of nonadjacent vertices is at least $n-1$, has a Hamilton path.*

Example. *The traveling salesperson problem.* Consider a salesperson who is planning a business trip that takes him (or her) to certain cities in which he has customers and then brings him back home to the city from whence he started. Between some of the pairs of cities he has to visit, there is direct air service; between others there is not. Can he plan the trip so that he flies into each city to be visited exactly once?

Let the number of cities to be visited, including his home city, be n. We let these cities be the vertices of a graph G of order n, in which there is an edge between two cities, provided that there is direct air service between them. Then what the salesperson seeks is a Hamilton cycle in G. If the graph G has the Ore property, then we know from Theorem 11.3.2 that there is a Hamilton cycle and, from its proof, a good way to construct one. But, in general, there is no good algorithm known which will construct a Hamilton cycle for the salesperson or which will tell him that no Hamilton cycle exists. The problem as formulated is not the real problem that a traveling salesperson faces. This is because distances between the cities he has to visit will in general vary, and what he would like is a Hamilton cycle in which the total distance travelled is as small as possible.[32] □

11.4 Bipartite Multigraphs

Let $G = (V, E)$ be a multigraph. Then G is called *bipartite*, provided that the vertex set V may be partitioned into two subsets X and Y so that each edge of G has one vertex in X and one vertex in Y. A pair X, Y with this property is called a *bipartition*

[31] G. A. Dirac, Some Theorems on Abstract Graphs, *Proc. London Math. Soc.*, 2 (1952), 69–81.

[32] On the other hand, he may want a Hamilton cycle that minimizes the total cost of his trip. Mathematically, there is no difference since, rather than attaching a weight to each edge that represents the distance between the cities it joins, we attach a weight that represents costs. In both cases we want a Hamilton cycle in which the sum of the weights attached to the edges of the cycle is minimum.

of G (and of its vertex set V). Two vertices in the same part of the bipartition are not adjacent. We usually picture a bipartite multigraph so that the vertices in X are on the left (and so are often called *left vertices*) and the vertices in Y are on the right (and so are often called *right vertices*).[33] Note that a bipartite multigraph does not have any loops. A multigraph that is a subgraph of a bipartite multigraph or is isomorphic to a bipartite multigraph is also bipartite.

Example. A bipartite multigraph with bipartition X, Y, where $X = \{a, b, c, d\}$ and $Y = \{u, v, w\}$, is shown in Figure 11.20. □

Example. Consider the graph G shown in Figure 11.21. Although it is not apparent from the drawing, G is a bipartite graph. This is because we may also draw G as in Figure 11.22, which reveals that G has a bipartition $X = \{a, c, g, h, j, k\}, Y = \{b, d, e, f, i\}$. □

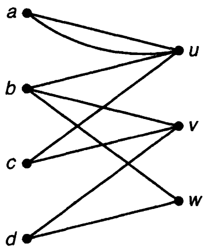

Figure 11.20

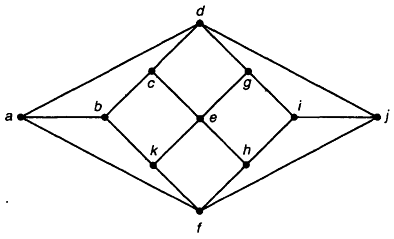

Figure 11.21

The previous example demonstrates that a drawing of a bipartite graph or a listing of its edges may not directly reveal the bipartite property. A description of the edges of a graph may reveal a bipartition of its vertices.

[33]Of course, *left* and *right* are interchangeable.

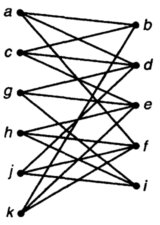

Figure 11.22

Example. Let G be the graph whose vertices are the integers from 1 to 20, with two integers joined by an edge if and only if their difference is an odd integer. We partition the vertices of G into the even integers and the odd integers. Since the difference between two odd integers is even and so is the difference between two even integers, two integers are adjacent in G if and only if one is odd and one is even. Thus, G is a bipartite graph with bipartition $X = \{1, 3, \ldots, 17, 19\}, Y = \{2, 4, \ldots, 18, 20\}$. □

A bipartite graph[34] G with bipartition X, Y is called *complete*, provided that each vertex in X is adjacent to each vertex in Y. Accordingly, if X contains m vertices and Y contains n vertices, then G has $m \times n$ edges. A complete bipartite graph with m left vertices and n right vertices is denoted by $K_{m,n}$. The graph G in the previous example is a $K_{10,10}$.

Since the bipartiteness of a multigraph may not be apparent from the way it is presented, we would like to have some alternative way to recognize bipartite multigraphs.

Theorem 11.4.1 *A multigraph is bipartite if and only if each of its cycles has even length.*

Proof. First, assume that G is a bipartite multigraph with bipartition X, Y. The vertices of a walk of G must alternate between X and Y. Since a cycle is closed, this implies that a cycle contains as many left vertices as it does right vertices and hence has even length.

Now suppose that each cycle of G has even length. First, assume that G is connected. Let x be any vertex of G. Let X be the set consisting of those vertices whose distance to x is even and let Y be the set consisting of those vertices whose distance to x is odd. Since G is assumed to be connected, X, Y is a partition of the vertices of G. We show that X, Y is a bipartition; that is, that no two vertices in X, respectively Y, are adjacent. Suppose, to the contrary, that there exists an edge $\{a, b\}$ where a and b are both in X. Let

$$\alpha : x - \cdots - a \text{ and } \beta : x - \cdots - b \tag{11.4}$$

[34]Not bipartite multigraph.

be walks of shortest length from x to a and x to b, respectively. Since the first vertex of each of these walks is x, there is a vertex z that is the last common vertex of these two walks. Thus, the walks in (11.4) are of the form

$$\alpha : x - \cdots - z - \cdots - a \text{ and } \beta : x - \cdots - z - \cdots - b. \tag{11.5}$$

We break each of these walks into two smaller walks:

$$\alpha_1 : x - \cdots - a \text{ and } \alpha_2 : z - \cdots - a,$$

and

$$\beta_1 : x - \cdots - z \text{ and } \beta_2 : z - \cdots - b.$$

The walks α_2 and β_2 have no vertices in common other than z. Since the walks α from x to a and β from x to b in (11.5) are shortest walks, the walks α_1 and β_1 must have the same length; if, for instance, α_1 had smaller length than β_1, then we could combine α_1 with β_2 and produce a walk from x to b of length smaller than that of β, a contradiction. Therefore, the two walks α_2 and β_2 are both of odd length or both of even length. The edge $\{a, b\}$ now implies the existence of a cycle

$$z - \cdots - a - b - \cdots - z$$

of odd length, contrary to hypothesis. Thus, there cannot be an edge joining two vertices in X, and, similarly, we show that there can be no edge joining two vertices in Y. Hence G is bipartite.

If G is not connected, then we apply the preceding argument to each connected component of G and conclude that each component is bipartite. But this implies that G is bipartite as well. □

In Section 11.7 we give a simple algorithm for determining the distances from a specified vertex x of a connected graph to every other vertex. Referring to the proof of Theorem 11.4.1, this will determine a bipartition of G if G is bipartite.

Example. Let n be a positive integer. We consider the set of all n-tuples of 0s and 1s as the vertices of a graph Q_n with two vertices joined by an edge if and only if they differ in exactly one coordinate. If $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ are joined by an edge, then the number of 1s in y is either one more or one less than the number of 1s in x. Let X consist of those n-tuples that have an even number of 1s; let Y consist of those n-tuples that have an odd number of 1s. Then two distinct vertices in X differ in at least two coordinates and hence are not adjacent. Similarly, two distinct vertices in Y are not adjacent. Hence, Q_n is a bipartite graph with bipartition X, Y.

Q_n is the graph of vertices and edges of an n-dimensional cube. The graphs Q_2 and Q_3 are shown in Figures 4.2–4.3, however, in a way that does not automatically reveal their bipartite nature; the drawings given in Figure 11.23 do. The reflected Gray code constructed in Section 4.3 is a Hamilton cycle in the graph Q_n. Thus, the search for a method to generate all the subsets of an n-element set with consecutive subsets differing as little as possible (one new element in or one old element out) is the same as the search for a Hamilton cycle (or path) in the n-cube graph Q_n. □

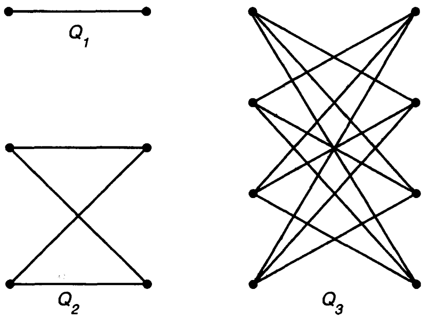

Figure 11.23

Example. Consider an n-by-n chessboard. Define a graph B_n whose vertices are the 64 squares of the board, where two squares are joined by an edge if and only if they have a common side.[35] Equivalently, two squares are adjacent if and only if they can be simultaneously covered by a domino. If we think of the squares of the board as alternately colored black and white, then we see that no two black squares are adjacent and no two white squares are adjacent. Thus, the usual coloring of a chessboard determines a bipartition of the vertices into its black squares and white squares, respectively, and hence the graph is bipartite. This graph is the *domino bipartite graph* of the board, and we may associate such a graph with any board with forbidden positions. We refer to Exercise 3 of Chapter 1, which asked whether it is possible to walk from one corner of an 8-by-8 board to the opposite corner, passing through each square exactly once. We now recognize this problem as asking whether the graph B_8 has a Hamilton path. Now B_8 is a bipartite graph with 32 white (or left) vertices and 32 black (or right) vertices. The desired Hamilton path starts and ends at vertices of the same color, say, black. Since B_8 is bipartite, the colors of the vertices in a path must alternate. Thus, it is impossible to include all the vertices in a Hamilton path from one corner to its opposite corner, since such a path must include one more black square than white square.

In a similar way, with any m-by-n board with forbidden positions, we may associate a domino bipartite graph whose vertices are the free positions of the board. □

Reasoning similar to that used in the preceding example establishes the following elementary result.

[35]That is, two squares are adjacent as vertices of B_n if and only if they are adjacent squares on the board.

Theorem 11.4.2 *Let G be a bipartite graph with bipartition X, Y. If $|X| \neq |Y|$, then G does not have a Hamilton cycle. If $|X| = |Y|$, then G does not have a Hamilton path that begins at a vertex in X and ends at a vertex in X. If X and Y differ by at least 2, then G does not have a Hamilton path. If $|X| = |Y| + 1$, then G does not have a Hamilton path that begins at X and ends at Y, or vice versa.* □

Notice that Theorem 11.4.2 has no positive conclusion. Each assertion in it only rules out the possibility of a Hamilton cycle or Hamilton path.

We close this section by discussing another old recreational problem[36] which, in modern language, also asks for a Hamilton cycle in a certain graph.

Example. *The knight's tour problem.* Consider an 8-by-8 chessboard and the chess piece known as a *knight*. A knight moves from its current location by moving two squares vertically and one square horizontally or one square vertically and two squares horizontally. Is it possible to place the knight on the board so that, with legal moves, the knight lands in each square exactly once? Such a tour is called a *knight's tour*, and we can ask for a knight's tour which has the property that the move from the last square to the first square is also a legal knight's move. A knight's tour with this property is called *reentrant*.

A solution of the problem, due to Euler, is

58	43	60	37	52	41	62	35
49	46	57	42	61	36	53	40
44	59	48	51	38	55	34	63
47	50	45	56	33	64	39	54
22	7	32	1	24	13	18	15
31	2	23	6	19	16	27	12
8	21	4	29	10	25	14	17
3	30	9	20	5	28	11	26

where the numbers indicate the order in which the squares are visited by the knight. In particular, square 1 is the initial position of the knight, and square 64 is the last. Since the move from square 1 to square 64 is a legal knight's move, this special tour is reentrant. Note that, in this tour, the knight first visits all the squares on the lower half of the board before entering the upper half.

The problem of the knight's tour can be considered on any m-by-n board, and we recognize it as a problem of the existence of a Hamilton path in a graph. Consider the squares of an m-by-n board to be the vertices of a graph $\mathcal{K}_{m,n}$ in which two squares are joined by an edge if and only if the move from one to the other is a legal knight's move. A Hamilton path in $\mathcal{K}_{m,n}$ represents a knight's tour on the m-by-n board, and a Hamilton cycle represents a reentrant tour. Considering the squares of the board to

[36]This problem was apparently first posed and solved by Indian chess players around 200 B.C.

be alternately colored black and white, as usual, we see that a knight always moves from a square of one color to a square of the other color. Thus, the graph $\mathcal{K}_{m,n}$ is a bipartite graph of order $m \times n$. If m and n are both odd, then there is one more square of one color than the other and hence, by Theorem 11.4.2, a reentrant knight's tour cannot exist. If at least one of m and n is even, then there is an equal number of black and white squares, and hence a reentrant tour possibly exists.

On a 1-by-n board, a knight cannot move at all. On a 2-by-n board, each of the four corner squares is accesssible by a knight from only one square. This means that in the graph $\mathcal{K}_{m,n}$, the corner squares each have degree equal to 1, and hence a knight's tour is impossible. What about a 3-by-3 board? On such a board the square in the middle is accessible by a knight from no other square. Hence, in the graph $\mathcal{K}_{3,3}$ the middle square has degree 0, and no tour is possible. Do not despair, for here is a nonreentrant tour, for a knight on a 3-by-4 board:

1	4	7	10
12	9	2	5
3	6	11	8

The labeling of the squares from 1 to n^2, using a knight's tour on an n-by-n board, results in a square array of numbers in which each of the numbers from 1 to n^2 appears exactly once. A person interested in magic squares[37] might ask whether there are knight's tours that result in magic squares, *magic knight's tours.*[38] It is known that magic knight's tours are not possible if n is odd, and that magic knight's tours exist if $n = 4k$ with $k > 2$. It has now been verified by exhaustive computer search that there does not exist a magic knight's tour on an 8-by-8 board. There exist many knight's tours that are *semimagic* in the sense that the integers in each row and in each column, but not the diagonals, add to the same number. An old example[39] is

1	30	47	52	5	28	43	54
48	51	2	29	44	53	6	27
31	46	49	4	25	8	55	42
50	3	32	45	56	41	26	7
33	62	15	20	9	24	39	58
16	19	34	61	40	57	10	23
63	14	17	36	21	12	59	38
18	35	64	13	60	37	22	11

.

□

[37]See Section 1.3.
[38]See H. E. Dudeney, *Amusements in Mathematics*, Dover Publishing Co., New York, 1958.
[39]W. Beverley, *Philos. Mag.*, p. 102, April 1848.

11.5 Trees

Suppose we want to build a connected graph of order n, using the smallest number of edges that we can "get away with."[40] One simple method of construction is to select one vertex and join it by an edge to each of the other $n-1$ vertices. The result is a complete bipartite graph $K_{1,n-1}$, called a *star*. The star $K_{1,n-1}$ is connected and has $n-1$ edges. If we remove any edge from it, we obtain a disconnected graph with a vertex meeting no edges. Another simple method of construction is to join the n vertices in a path. The resulting graph also is connected and has $n-1$ edges, and if we remove any edge, we obtain a disconnected graph. Can we construct a connected graph with n vertices that has fewer than $n-1$ edges?

Suppose we have a connected graph G of order n. Let's think of putting in the edges of G one by one. Thus, we start with n vertices and no edges and hence with a graph with n connected components. Each time we put in an edge we can decrease the number of connected components by, at most, 1: If the new edge joins two vertices that were already in the same component, then the number of components stays the same; if the new edge joins two vertices that were in different components, then those two components become one and all others are unaltered. Since we start with n components, and an edge can decrease the number of components by at most 1, we require at least $n-1$ edges to reduce the number of components to 1; that is, to get a connected graph. So we have proved the next elementary result.

Theorem 11.5.1 *A connected graph of order n has at least $n-1$ edges. Moreover, for each positive integer n, there exist connected graphs with exactly $n-1$ edges. Removing any edge from a connected graph of order n with exactly $n-1$ edges leaves a disconnected graph, and hence each edge is a bridge.* □

A *tree* is defined to be a connected graph that becomes disconnected upon the removal of any edge. Thus, a tree is a connected graph, each of whose edges is a bridge: Each edge is essential for the connectedness of the graph. We now prove that a *connected* graph can be shown to be a tree, simply by counting the number of its edges.

Theorem 11.5.2 *A connected graph of order $n \geq 1$ is a tree if and only if it has exactly $n-1$ edges.*

Proof. By Theorem 11.5.1 a connected graph of order n with exactly $n-1$ edges is a tree, since each of its edges is a bridge. Conversely, we prove by induction on n that a tree G of order n has exactly $n-1$ edges. If $n=1$, then G has no edges, and the conclusion is vacuously true. Assume that $n \geq 2$. Let α be any edge of G and

[40]For example, connect n cities by roads, using the fewest number of roads, in such a way that it is possible to get from each city to every other one.

let G' be the graph obtained from G by removing α. Since α is a bridge, G' has two connected components, G'_1 and G'_2, consisting of k and l vertices, respectively, where k and l are positive integers with $k + l = n$. Each edge of G'_1 is a bridge of G'_1, for, otherwise, its removal from G would clearly leave a connected graph, contrary to our assumption that G is a tree. Similarly, each edge of G'_2 is a bridge of G'_2. Thus, G'_1 and G'_2 are trees, and, by the induction hypothesis, G'_1 has $k - 1$ edges, and G'_2 has $l - 1$. Hence, G has $(k - 1) + (l - 1) + 1 = n - 1$ edges, as desired. □

Another characterization of a tree is given in the next theorem, but first we prove a lemma.

Lemma 11.5.3 *Let G be a connected graph and let $\alpha = \{x, y\}$ be an edge of G. Then α is a bridge if and only if there does not exist a cycle of G containing α.*

Proof. First suppose that α is a bridge. Then G consists of two connected graphs held together by α, and there can be no cycle containing α.[41] Now suppose that α is not a bridge. Then removing α from G leaves a connected graph G'. Hence, there is in G', and hence in G, a path

$$x - \cdots - y$$

that joins x and y and that does not contain the edge α. Then

$$x - \cdots - y - x$$

is a cycle containing the edge α. □

Theorem 11.5.4 *Let G be a connected graph of order n. Then G is a tree if and only if G does not have any cycles.*

Proof. We know that each edge of a tree is a bridge and hence, by Lemma 11.5.3, is not contained in any cycle. Thus, if G is a tree, then G does not have any cycle. Now suppose that G does not have any cycles. Since there are no cycles, it follows from Lemma 11.5.3, again, that each edge of G is a bridge and hence that G is a tree. □

Theorem 11.5.4 implies another characterization of trees.

Theorem 11.5.5 *A graph G is a tree if and only if every pair of distinct vertices x and y is joined by a unique path. This path is necessarily a shortest path joining x and y; that is, a path of length $d(x, y)$.*

Proof. First, suppose that G is a tree. Since G is connected, each pair of distinct vertices is joined by some path. If some pair of vertices is joined by two different paths, then it is easy to see that G contains a cycle,[42] contradicting Theorem 11.5.4.

[41] Keep in mind that the edges of a cycle are all different.

[42] Suppose that there are two different paths γ_1 and γ_2 from x to y. Both parts start at x and, since they are different, break apart at some vertex u. Since both paths end at y, they must come back together for the first time at some vertex v. We then have a cycle: Proceed from u to v along γ_1 and then from v to u along γ_2 in the opposite direction.

Now suppose that each pair of distinct vertices of G is joined by a unique path. Then G is connected. Since each pair of vertices of a cycle is joined by two different paths, G cannot have any cycles and, once again by Theorem 11.5.4, G is a tree. □

Let G be a graph. A *pendent vertex* of G is a vertex whose degree is equal to 1. Thus, a pendent vertex is incident with exactly one edge, and any edge incident with a pendent vertex is called a *pendent edge.*

Example. The graph G of order $n = 7$, shown in Figure 11.24, has three pendent vertices, namely, a, b, and g, and three pendent edges. This graph is not a tree. This is because the edge $\{c, d\}$ is not a bridge, or because it has $7 > 6$ edges (cf. Theorem 11.5.2), or because it has a cycle (cf. Theorem 11.5.4). □

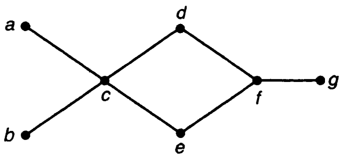

Figure 11.24

Theorem 11.5.6 *Let G be a tree of order $n \geq 2$. Then G has at least two pendent vertices.*

Proof. Let the degrees of the vertices of G be $d_1, d_2, \ldots, d_n$. Since G has $n-1$ edges, it follows from Theorem 11.1.1 that

$$d_1 + d_2 + \cdots + d_n = 2(n-1).$$

If at most one of the d_i equals 1, we have

$$d_1 + d_2 + \cdots + d_n \geq 1 + 2(n-1),$$

a contradiction. Hence, at least two of the d_i equal 1; that is, there are at least two pendent vertices. □

Example. What is the smallest and largest number of pendent vertices a tree G of order $n \geq 2$ can have?

Each of the two vertices of a tree of order 2 is pendent. Now let $n \geq 3$. If all the vertices of a tree were pendent, then the tree would not be connected (in fact, n would have to be even and no two edges would be incident). A star $K_{1,n-1}$ has $n-1$ pendent vertices, and hence $n-1$ is the largest number of pendent vertices a tree of order $n \geq 3$ can have. A tree whose edges are arranged in a path has exactly two pendent vertices. Thus, by Theorem 11.5.6, 2 is the smallest number of pendent vertices for a tree of order $n \geq 2$. □

Example. *How to grow trees.* By Theorem 11.5.6, a tree has a pendent vertex and hence a pendent edge. If we remove an edge from a tree, G, then we get a graph with two connected components each of which is also a tree. If the edge removed is pendent, then one of the smaller trees consists of a single vertex, and the other is a tree G' of order $n-1$. Conversely, if we have a tree G' of order $n-1$, then, selecting a new vertex u and joining it by an edge $\{u, x\}$ to a vertex x of G', we get a tree G in which u is a pendent vertex. This implies that *every* tree can be constructed as follows: Start with a single vertex and iteratively choose a new vertex, and put in a new edge joining the new vertex to any old vertex. A tree of order 5 is constructed in Figure 11.25 in this way. □

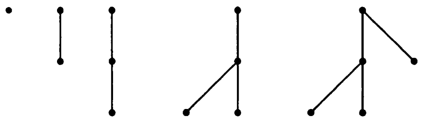

Figure 11.25

Using the method of construction of the previous example, it is not difficult to now show that the number t_n of nonisomorphic trees of order n satisfies $t_1 = 1, t_2 = 1, t_3 = 1, t_4 = 2, t_5 = 3,$ and $t_6 = 6$. The different trees with six vertices are shown in Figure 11.26.

We have defined a tree to be a connected graph, each of whose edges is a bridge. Thus, if a connected graph G is not a tree, then it has a nonbridge; that is, an edge whose removal does not disconnect the graph. If we iteratively remove nonbridge edges until every edge is a bridge of the remaining graph, we get a tree with the same set of vertices as G and some of its edges; that is, we get a spanning subgraph that is a tree. A tree that is a spanning subgraph of a graph G is called a *spanning tree* of G.

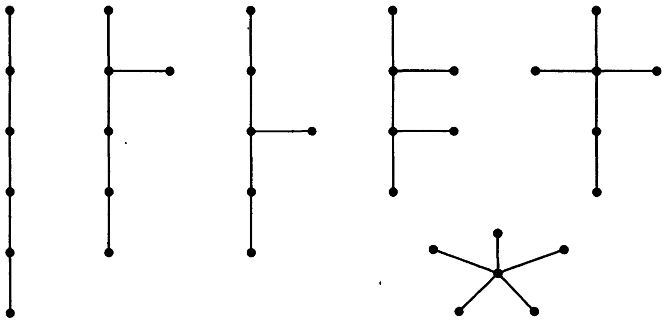

Figure 11.26

Theorem 11.5.7 *Every connected graph has a spanning tree.*

Proof. The algorithmic proof is contained in the preceding paragraph. We give a more precise formulation of the algorithm. Recall from Lemma 11.3.1 that an edge of a connected graph is a bridge if and only if it is not contained in any cycle.

Algorithm for a spanning tree

Let $G = (V, E)$ be a connected graph of order n.

(i) Set F equal to E.

(ii) While there is an edge α of F such that α is not a bridge of the graph $T = (V, F)$, remove α from F.

The terminal graph $T = (V, F)$ is a spanning tree of G.

As we have argued, the terminal graph $T = (V, F)$ is connected and does not have any bridges; hence, it is a tree. □

We remark that our restriction to graphs in Theorem 11.5.7 is not essential. If G is a general graph, then we can immediately remove all loops, and all but one copy of each edge in G, and then apply Theorem 11.5.7 and the algorithm in its proof. Thus, every connected general graph has a spanning tree as well.

Example. Let G be the connected graph of order 7, shown on the left in Figure 11.27. This graph has exactly one bridge, namely the edge $\{2, 3\}$; hence, we can begin the algorithm for a spanning tree by removing any other edge, say the edge $\{1, 2\}$. The edges $\{1, 4\}, \{4, 5\}, \{2, 5\}$, and $\{2, 3\}$ are now bridges and can no longer be removed. Removing the edge $\{6, 7\}$ leaves the spanning tree shown on the right. □

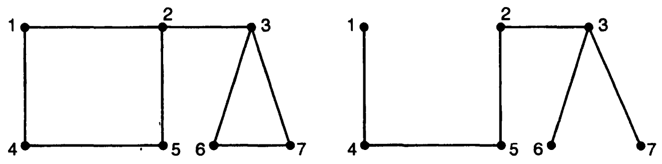

Figure 11.27

We conclude this section with two properties of spanning trees that will be used in subsequent sections of this chapter.

Theorem 11.5.8 *Let T be a spanning tree of a connected graph G. Let $\alpha = \{a, b\}$ be an edge of G that is not an edge of T. Then there is an edge β of T such that the graph T' obtained from T by inserting α and deleting β is also a spanning tree of G.*

Proof. Let the graph G, and hence the graph T, have n vertices. First, consider the graph T' obtained from T by inserting the given edge α. Since T' is not a tree, it has, by Theorem 11.5.4, a cycle γ which necessarily contains at least one edge of T. By Lemma 11.3.1, each edge of γ is not a bridge of T'. Let β be any edge of γ other than α. Removing β from T' results in a graph with n vertices and $n - 1$ edges that is connected and hence is a tree. □

Theorem 11.5.9 *Let T_1 and T_2 be spanning trees of a connected graph G. Let β be an edge of T_1. Then there is an edge α of T_2 such that the graph obtained from T_1 by inserting α and deleting β is a spanning tree of G.*

Proof. We first remark on the difference between Theorems 11.5.8 and 11.5.9. In Theorem 11.5.8 we are given a spanning tree and some edge α not in it, and we want to put α in T and take out *any* edge β of T as long as the result is a spanning tree. In Theorem 11.5.9 we are given a spanning tree T_1 and we want to take out a *specific* edge β of T_1 and put in any edge of T_2 as long as the result is a spanning tree.

To prove the theorem, first remove the edge β from the spanning tree T_1 of G. The result is a graph with two connected components T_1' and T_2'' (both of which must be trees). Since T_2 is also a spanning tree of G, T_2 is connected with the same set of vertices as T_1, and hence there must be some edge α of T_2 that joins a vertex of T_1' and a vertex of T_2''. The graph obtained from T_1 by inserting the edge α and removing the edge β is a connected graph with $n - 1$ edges; hence, it is a tree. (We note that if β is not an edge of T_2, then α is not an edge of T_1, for otherwise we would get a connected graph of order n with fewer than $n - 1$ edges.) □

It is natural for us to ask for the number of spanning trees of a connected graph. The number of spanning trees of any connected graph can be computed by an algebraic formula,[43] but such a formula is beyond the scope of this book.

Example. The number of spanning trees of the graph of order 4 shown in Figure 11.28 (a cycle of length 4) is 4. Each of these spanning trees is a path of length 3, as drawn in the figure. Consequently, all are isomorphic. □

A famous formula of Cayley asserts that the number of spanning trees of a complete graph K_n is n^{n-2}, a surprisingly simple formula. As illustrated in the preceding example, many of these trees may be isomorphic to each other. Thus, while each tree of order n occurs as a spanning tree of K_n, it may occur many times (with different labels on its vertices). Thus, n^{n-2} does not represent the number of nonisomorphic trees of order n. The latter number is a more complicated function of n.

[43]It is the absolute value of the determinant of any submatrix of order $n - 1$ of the Laplacian matrix of a graph.

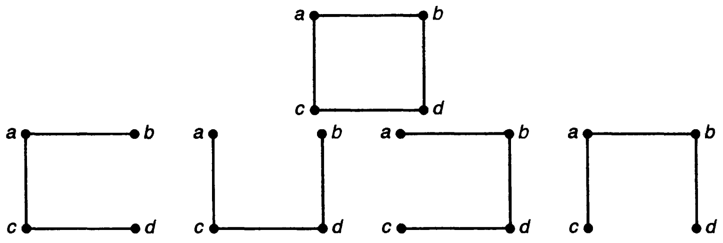

Figure 11.28

11.6 The Shannon Switching Game

We discuss in this section a game that can be played on any multigraph. It was invented by C. Shannon[44] and its elegant solution was found by A. Lehman.[45] The remainder of this book is independent of this section.

Shannon's game is played by two people, called here the *positive player* P and the *negative player* N, who alternate turns.[46] Let $G = (V, E)$ be a multigraph in which two of its vertices u and v have been distinguished. Thus, the "gameboard" consists of a multigraph with two distinguished vertices. The goal of the positive player is to construct a path between the distinguished vertices u and v. The goal of the negative player is to deny the positive player his goal, that is, to destroy all paths between u and v. The play of the game proceeds as follows: When it is N's turn, N destroys some edge of G by putting a negative sign $-$ on it.[47] When it is P's turn, P puts a positive sign $+$ on some edge of G, which now cannot be destroyed by N. Play proceeds until one of the players achieves his or her goal:

(1) There is a path between u and v that has only $+$ signs on its edges. In this case, *the positive player has won.*

(2) Every path in G between u and v contains a $-$ sign on at least one of its edges; that is, N has destroyed all paths between u and v. In this case *the negative player has won.*

[44]Clause Shannon, 1916–2001, laid the foundation of modern communication theory while working at Bell Labs.

[45]A. Lehman, A Solution of the Shannon switching Game, *J. Society Industrial and Applied Mathematics*, 12 (1964), 687–725. Our description of the game and its solution is based on Section 3 of the author's article, Networks and the Shannon Switching Game, *Delta*, 4 (1974), 1-23.

[46]We could call the positive player the *constructive player* and the negative player the *destructive player.*

[47]If the game is played by drawing G on paper with a pencil, then N can destroy an edge by erasing the edge.

It is evident that, after all edges of the multigraph G have been played (that is, have either a + or a − on them), exactly one of the players will have won. In particular, the game never ends in a draw. If G is not connected and u and v lie in different connected components of G, then we can immediately declare N the winner.[48]

We consider the following questions:

(1) Does there exist a strategy for P to follow which will guarantee him or her a win, *no matter how well N plays*? If so, determine such a winning strategy for P.

(2) Does there exist a strategy for N to follow which will guarantee him or her a win, *no matter how well P plays*? If so, determine such a winning strategy for N.

The answers to these questions may sometimes depend on whether the positive or negative player has the first move.

Example. First, consider the multigraph on the left in Figure 11.29, with distinguished vertices u and v as shown. In this game the positive player P wins whether he or she plays first or second. This is because a + on either edge determines a path between u and v. Now consider the middle graph in Figure 11.29. In this game the negative player N wins, whether he or she plays first or second. This is because a − on either of the two edges destroys all paths between u and v. Finally, consider the right graph in Figure 11.29. In this game, whichever player goes first, and thereby claims the only edge of the graph, is the winner. □

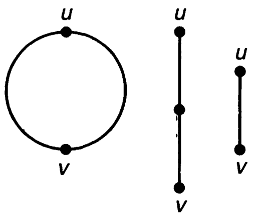

Figure 11.29

Motivated by the preceding example, we make the following definitions: A game is called a *positive game* provided that the positive player has a winning strategy whether he or she plays first or second. A game is called a *negative game* provided that the negative player has a winning strategy whether he or she plays first or second. A game is called a *neutral game* provided that the player who plays first has a winning

[48]And P should be embarrassed for getting involved in a game in which it was impossible for him or her to win.

strategy. We note that, if the positive player has a winning strategy when playing second, then he or she also has a winning strategy playing first. This is because the positive player can ignore his or her first move[49] and play according to the winning strategy as the second player. If the strategy calls for the positive player to put a + on an edge that already has one, he or she then has a "free move" and can put a + on any available edge. Similarly, if the negative player has a winning strategy playing second, then he or she has a winning strategy playing first.

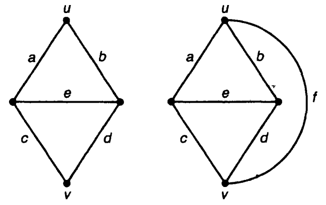

Figure 11.30

Example. Consider the game determined by the left graph in Figure 11.30, with distinguished vertices u and v as shown. Assume that P has first move and puts a + on edge e. We pair up the remaining edges by pairing a with b and c with d. If P counters a move by N on an edge, by a move on the other edge of its pair, then P is guaranteed a win. Thus, P can win this game, provided he or she has first move. Now assume that N has first move and puts a $-$ on edge e. We now pair up the remaining edges by pairing a with c and b with d. If N counters a move by P on an edge by a move on the other edge of its pair, then N is guaranteed a win. Hence, N can win this game, provided he or she has first move. We conclude that the game determined by Figure 11.30 is a neutral game.

Now suppose that we add a new edge f, which joins the distinguished vertices u and v, resulting in the graph shown on the right in Figure 11.30. Suppose the negative player makes the first move in this new game. If N does not put a $-$ on the new edge f, then the positive player can put a + on that edge, thereby winning the game. If N does put a $-$ on f, then the rest of the game is the same as the previous game, with P making the first move, and hence P can win. Thus, P has a winning strategy as second player, and this game is a positive game. □

The principle illustrated in the previous example holds in general.

Theorem 11.6.1 *A neutral game is converted into a positive game if a new edge joining the distinguished vertices u and v is added to the multigraph of the game.*

A characterization of positive games is given in the next theorem. Recall that, if $G = (V, E)$ is a multigraph and U is a subset of the vertex set V, then G_U denotes the

[49]But the negative player cannot.

multisubgraph of G induced by U—that is, the multigraph with vertex set U whose edges are all the edges of G that join two vertices in U. Put another way, G_U is obtained from G by deleting all vertices in $\overline{U} = V - U$ and all edges that are incident with at least one vertex in $\overline{U}$.

Theorem 11.6.2 *The game determined by a multigraph $G = (V, E)$ with distinguished vertices u and v is a positive game if and only if there is a subset U containing u and v of the vertex set V such that the induced multisubgraph G_U has two spanning trees, T_1 and T_2, with no common edges.*

Otherwise stated, a game is a positive game if and only if there are two trees T_1 and T_2 in G such that T_1 and T_2 have the same set of vertices, both u and v are vertices of T_1 and T_2, and T_1 and T_2 have no edges in common. The game determined by the right graph in Figure 11.30 was shown to be a positive game. For T_1 and T_2, we can take the two trees in Figure 11.31. In this case T_1 and T_2 are spanning trees of G (that is, $U = V$), but this need not always be so. It is possible that the set U contain only some of the vertices of V.

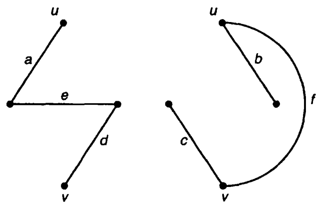

Figure 11.31

We shall not give a complete proof of Theorem 11.6.2. Rather, we shall show only how to use the pair of trees T_1 and T_2 to devise a winning strategy for the positive player P when the negative player N makes the first move. After each sequence of play, consisting of a move by the negative player followed by a move by the positive player, we shall construct a new pair of spanning trees of G_U that have one more edge in common than the previous pair. Initially, we have the spanning trees T_1 and T_2 of G_U with no edges in common, and we now label these trees as

$$T_1^{(0)} = T_1 \text{ and } T_2^{(0)} = T_2.$$

The first sequence of play

Player N goes first and puts a $-$ on some edge β. We consider two cases:

Case 1: β is an edge of one of the trees $T_1^{(0)}$ and $T_2^{(0)}$, say, the tree $T_1^{(0)}$.

Since $T_1^{(0)}$ and $T_2^{(0)}$ are spanning trees of G_U, it follows from Theorem 11.5.9 that there is an edge α of $T_2^{(0)}$ such that the graph obtained from $T_1^{(0)}$ by inserting α and deleting β is a spanning tree $T_1^{(1)}$ of G_U. Our instructions to P are to put a $+$ on the edge α. We let $T_2^{(1)} = T_2^{(0)}$. The trees $T_1^{(1)}$ and $T_2^{(1)}$ have exactly one edge in common, namely, the edge α with a $+$ on it.

Case 2: β is neither an edge of $T_1^{(0)}$ nor an edge of $T_2^{(0)}$.

Our instructions to P are now to place a $+$ on any edge α of $T_1^{(0)}$ or of $T_2^{(0)}$, say, an edge α of $T_1^{(0)}$.[50] Since $T_2^{(0)}$ is a spanning tree of G_U and α is an edge of G_U, it follows from Theorem 11.5.9 that there is an edge γ of $T_2^{(0)}$ such that the graph obtained from $T_2^{(0)}$ by inserting α and deleting γ is a spanning tree $T_2^{(1)}$ of G_U. We let $T_1^{(1)} = T_1^{(0)}$. The trees $T_1^{(1)}$ and $T_2^{(1)}$ have only the edge α with a $+$ in common.

We conclude that, at the end of the first sequence of play, there are two spanning trees, $T_1^{(1)}$ and $T_2^{(1)}$, of G_U that have exactly one edge in common, namely, the edge with a $+$ on it that was played by P.

The second sequence of play

Player N puts a $-$ on a second edge δ of G, and we seek a countermove for P. The determination of an edge ρ on which P should put a $+$ is very much like that in the first sequence of play, and we shall be briefer in our description:

Case 1: δ is an edge of one of the two trees $T_1^{(1)}$ and $T_2^{(1)}$, say, the tree $T_2^{(1)}$.

There is an edge ρ of $T_1^{(1)}$ such that the graph $T_1^{(2)}$ obtained from $T_1^{(1)}$ by inserting the edge δ and deleting the edge ρ is a spanning tree of G_U. Our instructions to P are to place a $+$ on the edge ρ. We let $T_2^{(2)} = T_2^{(1)}$.

Case 2: δ is neither an edge of $T_1^{(1)}$ nor of $T_2^{(1)}$.

Our instructions to P are to place a $+$ on any available edge[51] of $T_1^{(1)}$ and $T_2^{(1)}$, say, an edge ρ of $T_1^{(1)}$. There exists an edge ϵ of $T_2^{(1)}$ such that the graph $T_2^{(2)}$ obtained from $T_2^{(1)}$ by inserting the edge ρ and deleting the edge ϵ is a spanning tree of G_U. We let $T_1^{(2)} = T_1^{(1)}$.

[50]In this case, N has "wasted" his or her move and P gets a "free" move anywhere on one of the trees $T_1^{(0)}$ and $T_2^{(0)}$.

[51]That is, an edge that has not yet been "signed."

We conclude that, at the end of the second sequence of play, there are two spanning trees, $T_1^{(2)}$ and $T_2^{(2)}$, of G_U that have exactly two edges in common, namely, the two edges with a + on them that were played by P.

The description of the remainder of the strategy for P is very similar to that given for the first and second sequences of play. At the end of the kth sequence of play, there are two spanning trees, $T_1^{(k)}$ and $T_2^{(k)}$ of G_U, which have exactly k edges in common, namely, the k edges with a + on them that have been played up to this point by P. Let the number of vertices in U be m. Then, at the end of the $(m-1)$st sequence of play, the spanning trees $T_1^{(m-1)}$ and $T_2^{(m-1)}$ of G_U have exactly $m-1$ edges in common. Since a tree with m vertices has only $m-1$ edges, this means that $T_1^{(m-1)}$ is the same tree as $T_2^{(m-1)}$, and thus the edges with a + on them are the edges of a spanning tree of G_U. Because u and v belong to U, there is a path of edges with a + on them joining the distinguished vertices u and v. We therefore conclude that, had the positive player P followed our instructions, then, at the end of the $(m-1)$st sequence of play, if not before, he or she would have put + signs on a set of edges that contains a path joining u and v and thus would have won the game. Our instructions to P are thus a winning strategy. □

Theorem 11.6.2 can be used to classify neutral and negative games as follows: Let $G = (V, E)$ be a multigraph with distinguished vertices u and v. Let G^* be the multigraph obtained from G by inserting a new edge joining u and v. Then the following conclusions can be drawn:

1. The game played with G, u, and v is a neutral game if and only if it is not a positive game, but the game played with G^*, u, and v is a positive game.

2. The game played with G, u, and v is a negative game if and only if neither the game played with G, u, and v nor the game played with G^*, u, and v is a positive games.

Thus, by Theorem 11.6.2, the game played with G, u, and v is a neutral game if and only if G does not contain two disjoint trees with the same set of vertices including u and v, but by inserting a new edge joining u and v we are able to find two such trees. The game played with G, u, and v is a negative game if and only if, even with the new edge joining u and v, two such trees do not exist. In a neutral game G, the positive player can win when he or shegoes first by pretending that the game is being played with G^* with N going first and that N's first move was to put a $-$ on the new edge joining u and v. In general, there is no easily describable winning strategy for negative games in which N goes second or for neutral games in which N goes first.

11.7 More on Trees

In the proof of Theorem 11.5.7, we have given an algorithm for obtaining a spanning tree of a connected graph. Reviewing this algorithm, we see that it is more "destructive" than it is constructive: Iteratively, we locate an edge that is in a cycle—a nonbridge edge—of the current graph and remove or "destroy" it. Implicit in this algorithm is the assumption that we have some subalgorithm for locating a nonbridge edge. In Section 11.5, described a procedure that will construct any tree with n vertices, equivalently, any spanning tree of the complete graph K_n of order n. This procedure can be refined to apply to any graph[52] to grow all of its spanning trees. We formalize the resulting algorithm now. It need not be assumed that the initial graph G is connected. A byproduct of the algorithm is an algorithm to determine whether or not a graph is connected.

Algorithm to grow a spanning tree

```
Let $G = (V, E)$ be a graph of order $n$ and let $u$ be any vertex.
(1) Put $U = \{u\}$ and $F = \emptyset$.
(2) While there exists a vertex $x$ in $U$ and a vertex $y$ not in $U$ such that $\alpha = \{x, y\}$
    is an edge of $G$,
      (i) Put the vertex $y$ in $U$.
      (ii) Put the edge $\alpha$ in $F$.
(3) Put $T = (U, F)$.
```

In step (2) there will, in general, be many choices for the vertices x and y, and thus we have considerable latitude in carrying out the algorithm. Two special and important rules for choosing x and y are described after the next theorem.

Theorem 11.7.1 *Let $G = (V, E)$ be a graph. Then G is connected if and only if the graph $T = (U, F)$ constructed by carrying out the preceding algorithm is a spanning tree of G.*

Proof. If T is a spanning tree of G, then surely G is connected. Now assume that G is connected. Initially, T has one vertex and no edges and is therefore connected. Each application of (2) adds one new vertex to U and one new edge to F, which joins the new vertex to an old vertex. It then follows inductively that, at each stage of

[52]There is no loss in generality in considering only graphs in this section. If we have a general graph, we can immediately remove all loops and all but one copy of each edge and apply the results and algorithms of this section to the resulting graph.

the algorithm, the current $T = (U, F)$ is connected with $|F| = |U| - 1$, and hence T is a tree. Suppose that, upon termination of the algorithm, we have $U \neq V$. Since G is connected, there must be an edge from some vertex in U to some vertex not in U, contradicting the assumption that the algorithm has terminated. Thus, upon termination, we have $U = V$, and $T = (U, F)$ is a spanning tree of G. □

It should be clear that each spanning tree of a connected graph can be constructed by making the right choices for x and y in carrying out the algorithm for growing a spanning tree. We now describe one way to make choices that results in a spanning tree with a special property. The resulting algorithm is described next, and it constructs what is called a *breadth-first spanning tree* rooted at a prescribed vertex, the initial vertex u in the set U. A connected graph G has, in general, many breadth-first spanning trees T rooted at a vertex u. Their common feature is that the distance between u and x in G is the same as the distance between u and x in T for each vertex x. For convenience, we call a breadth-first spanning tree a *BFS-tree*. In the algorithm, we attach two numbers to each vertex x. One of these is called its *breadth-first number*, denoted $bf(x)$. The breadth-first numbers represent the order in which vertices are put into the BFS-tree. The other number represents the distance between the root u and x in the BFS-tree, and is denoted by $D(x)$.[53]

BF-algorithm to grow a BFS-tree rooted at u

Let $G = (V, E)$ be a graph of order n and let u be any vertex.

(1) Put $i = 1$, $U = \{u\}$, $D(u) = 0$, $bf(u) = 1$, $F = \emptyset$, and $T = (U, F)$.

(2) If there is no edge in G that joins a vertex x in U to a vertex y not in U, then stop. Otherwise, determine an edge $\alpha = \{x, y\}$ with x in U and y not in U such that x has the smallest breadth-first number $bf(x)$, and do the following:

   (i) Put $bf(y) = i + 1$.
   (ii) Put $D(y) = D(x) + 1$.
   (iii) Put the vertex y into U.
   (iv) Put the edge $\alpha = \{x, y\}$ into F.
   (v) Put $T = (U, F)$.
   (vi) Increase i by 1 and go back to (2).

Theorem 11.7.2 *Let $G = (V, E)$ be a graph and let u be any vertex of G. Then G is connected if and only if the graph $T = (U, F)$ constructed by carrying out the BF-algorithm is a spanning tree of G. If G is connected, then, for each vertex y of G, the distance in G between u and y equals $D(y)$; and this is the same as the distance between u and y in T.*

[53]The number $D(x)$ depends on the choice of root u, but otherwise depends only on the graph G and not on the particular BFS-tree rooted at u. The number $bf(x)$ does depend on the BFS-tree.

Proof. The BF-algorithm is a special way of carrying out the general algorithm for growing a spanning tree. It thus follows from Theorem 11.7.1 that G is connected if and only if the terminal graph $T = (U, F)$ is a spanning tree.

Now assume that G is connected so that at the termination of the algorithm $T = (U, F)$ is a spanning tree of G. It should be clear from the algorithm that $D(y)$ equals the distance between u and y in the tree T. Trivially, $D(u) = 0$ is the distance between u and itself in G. Suppose that there is some vertex y such that $D(y) = l$ is greater than the distance k between u and y in G. We may assume that k is the smallest number with this property. Then there is a path

$$\gamma: \quad u = x_0 - x_1 - \cdots - x_{k-1} - x_k = y$$

in G joining u and y whose length k satisfies

$$k < l = D(y).$$

The distance between u and the vertex x_{k-1} of γ is, at most, $k - 1$ and hence, by the minimality of k, $D(x_{k-1}) \leq k - 1$. Since $y = x_k$ is adjacent to x_{k-1}, it follows from the BF-algorithm that we would put $D(y) = k$ unless $D(y)$ had already been assigned a smaller number. Hence, $D(y) \leq k < l$, a contradiction. Therefore, the function D gives the distance in G (and in T) from u to each vertex. □

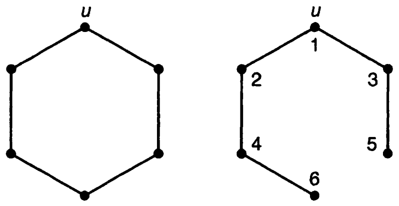

Figure 11.32

Example. Each BFS-tree of a complete graph K_n is a star $K_{1,n-1}$. A BFS-tree of the cycle of length 6 on the left in Figure 11.32 is the tree on the right in that figure. A BFS-trees of the graph Q_3 of vertices and edges of a three-dimensional cube is shown in Figure 11.33. (Recall from Section 11.4 that the vertices of this graph are the 3-tuples of 0s and 1s and that two vertices are adjacent if and only if they differ in exactly one coordinate.) In each case, the breadth-first numbers are noted next to the vertices of the tree. The distances $D(x)$ are readily determined. □

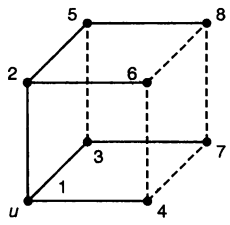

Figure 11.33

A breadth-first spanning tree rooted at u of a connected graph G is a spanning tree that is as "broad" as possible; each vertex is as close to the root as G will allow. The algorithm for a BFS-tree can be regarded as a systematic way to search (or list) all the vertices of G without repetition. According to this algorithm, one visits the vertices closest to the root first (breadth takes precedence over depth). We now describe a way to carry out the algorithm to grow a tree that produces a spanning tree that is as deep as possible. A spanning tree produced by this algorithm is called a *depth-first spanning tree*, abbreviated as *DFS-tree*, rooted at a vertex u. In this case, depth takes precedence over breadth. In the algorithm, we attach a number to each vertex x, called its *depth-first number* and denoted by $df(x)$. The depth-first algorithm is also known as *backtracking*. In backtracking we proceed in the forward direction as long as we are able; when it is no longer possible to advance, then we backtrack to the first vertex from which we can go forward.

DF-algorithm to grow a DFS-tree rooted at u

Let $G = (V, E)$ be a graph of order n and let u be any vertex.

(1) Put $i = 1$, $U = \{u\}$, $df(u) = 1$, $F = \emptyset$, and $T = (U, F)$.

(2) If there is no edge in G that joins a vertex x in U to a vertex y not in U, then stop. Otherwise, determine an edge $\alpha = \{x, y\}$ with x in U and y not in U such that x has the largest depth-first number $df(x)$, and do the following:

  (i) Put $df(y) = i + 1$.

  (ii) Put the vertex y into U.

  (iii) Put the edge $\alpha = \{x, y\}$ into F.

  (iv) Put $T = (U, F)$.

  (vi) Increase i by 1 and go back to (2).

Theorem 11.7.3 *Let $G = (V, E)$ be a graph and let u be any vertex of G. Then G is connected if and only if the graph $T = (U, F)$, constructed by carrying out the preceding DF-algorithm, is a spanning tree of G.*

Proof. The DF-algorithm is a special way of carrying out the general algorithm for growing a spanning tree. It thus follows from Theorem 11.7.1 that G is connected if and only if the constructed graph $T = (U, F)$ is a spanning tree. □

Example. Each DFS-tree of a complete graph K_n is a path. A DFS-tree of a cycle of any length is also a path. A DFS-tree of the graph Q_3 of vertices and edges of a three-dimensional cube is shown in Figure 11.34. In each case, the depth-first numbers are noted next to the vertices of the tree. □

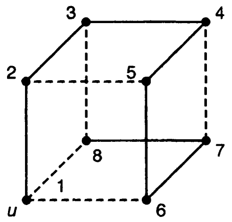

Figure 11.34

Example. If G is a tree, then each BFS-tree and DFS-tree of G is G itself, with its vertices ordered in the order they are visited. In this case, we often speak of a *breadth-first search* of G and a *depth-first search* of G. The tree G may represent a data structure for a computer file in which information is stored at places corresponding to the vertices of G. To find a particular piece of information, we need to "search" each vertex of the tree until we find the desired information. Both a breadth-first search and a depth-first search provide an algorithm for searching each vertex at most once. If we think of a tree as a system of roads connecting various cities, then a depth-first search of G can be visualized as a walk along the edges, in which each vertex is visited at least once.[54] Starting at the root u, we walk in the forward direction as long as possible and go backward only until we locate a vertex from which we can again go forward. Such a walk is illustrated in Figure 11.35, where we have returned to the root u (so our walk is a closed walk in which we traverse each edge exactly twice). □

According to Theorem 11.7.2, the number $D(x)$ computed by the breadth-first algorithm starting with a vertex u equals the distance from u to x in a connected graph. However, in graphs that model various physical situations, some edges are more "costly" than others. An edge might represent a road connecting two cities, and the physical distance between these cities should be taken into account if the graph is to provide an accurate model. An edge might also represent a potential new road between two cities, and the cost of constructing that road must be considered. These

[54]But we search each vertex only the first time it is visited.

two situations motivate us to consider graphs in which a weight is attached to each edge.[55]

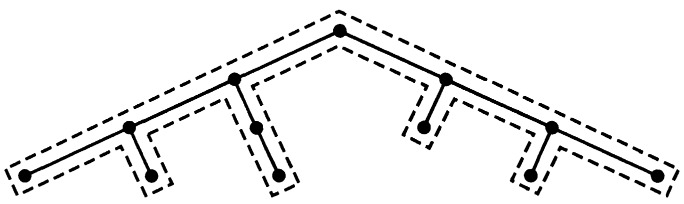

Figure 11.35

Let $G = (V, E)$ be a graph in which to each edge $\alpha = \{x, y\}$ there is associated a nonnegative number $c(\alpha) = c\{x, y\}$, called its *weight*. We call G a *weighted graph* with weight function c. The *weight of a walk*

$$\gamma : \{x_0, x_1\}, \{x_1, x_2\}, \ldots, \{x_{k-1}, x_k\}$$

in G is defined to be

$$c(\gamma) = c\{x_0, x_1\} + c\{x_1, x_2\} + \cdots + c\{x_{k-1}, x_k\},$$

the sum of the weights of the edges of γ. The *weighted distance* $d_c(x, y)$ between a pair of vertices x and y of G is the smallest weight of all the walks joining x and y. If there is no walk joining x and y, then we define $d_c(x, y) = \infty$. We also define $d_c(x, x) = 0$ for each vertex x. Since all weights are nonnegative, if $d_c(x, y) \neq \infty$, then there is a path of weight $d_c(x, y)$ joining the pair of distinct vertices x and y. Starting with a vertex u in a connected graph G, we show how to compute $d_c(u, x)$ for each vertex x and construct a spanning tree rooted at u such that the weighted distance between u and each vertex x equals $d_c(u, x)$. We call such a spanning tree a *distance tree for* u. The algorithm presented next is usually called *Dijkstra's algorithm*[56] and can be regarded as a weighted generalization of the BF-algorithm.

Algorithm for a distance tree for u

Let $G = (V, E)$ be a weighted graph of order n and let u be any vertex.

(1) Put $U = \{u\}$, $D(u) = 0$, $F = \emptyset$, and $T = (U, F)$.

[55]The physical significance of the weight is irrelevant for the mathematical problems that we solve. However, the fact that weight may have relevant physical significance leads to important applications of the mathematical results obtained.

[56]E. W. Dijkstra, A Note on Two Problems in Connection with Graphs, *Numerische Math.*, 1 (1959), 285–292.

(2) If there is no edge in G that joins a vertex x in U to a vertex y not in U, then stop. Otherwise, determine an edge $\alpha = \{x, y\}$ with x in U and y not in U such that $D(x) + c\{x, y\}$ is as small as possible, and do the following:

(i) Put the vertex y into U.

(ii) Put the edge $\alpha = \{x, y\}$ into F.

(iii) Put $D(y) = D(x) + c\{x, y\}$ and go back to (2).

Theorem 11.7.4 *Let $G = (V, E)$ be a weighted graph and let u be any vertex of G. Then G is connected if and only if the graph $T = (U, F)$ obtained by carrying out the preceding algorithm is a spanning tree of G. If G is connected, then for each vertex y of G, the weighted distance between u and y equals $D(y)$, and this is the same as the weighted distance between u and y in the weighted tree T.*

Proof. The algorithm for a distance tree is a special way of carrying out our general algorithm for growing a spanning tree. It thus follows from Theorem 11.7.1 that G is connected if and only if the constructed graph $T = (U, F)$ is a spanning tree; that is, if and only if the terminal value of U is V.

Now, assume that G is connected, so that at the termination of the algorithm, $U = V$, and $T = (U, F)$ is a spanning tree of G. It is clear from the algorithm that $D(y)$ equals the distance between u and y in the tree T. Trivially, $D(u) = 0$ is the distance between u and itself in G. Suppose, to the contrary, that there is some vertex y such that $D(y)$ is greater than the distance d between u and y in G. We may assume that y is the first vertex put in U with this property. There is a path

$$\gamma: \quad u = x_0 - x_1 - \cdots - x_k = y$$

in G joining u and y whose weight is $d < D(y)$. Let x_j be the last vertex of γ which is put into U before y. (Since u is the first vertex put into U, the vertex x_j exists.) It follows from our choice of y that $D(x_j)$ equals the weighted distance from u to x_j in G. The subpath

$$\gamma': \quad u = x_0 - x_1 - \cdots - x_j - x_{j+1}$$

of γ has weight

$$D(x_j) + c\{x_j, x_{j+1}\} \leq d < D(y).$$

Hence, by the algorithm, x_{j+1} is put into U before y, contradicting our choice of x_j. This contradiction implies that $D(y)$ is the weighted distance between u and y for all vertices y. $\square$

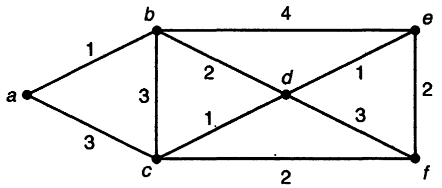

Figure 11.36

Example. Let G be the weighted graph in Figure 11.36, where the numbers next to an edge denote its weight. If we carry out the algorithm for a distance tree with $u = a$, we obtain the tree drawn in Figure 11.37, with the vertices and edges selected in the following order:

$$\text{vertices: } a,\ b,\ d,\ c,\ e,\ f,$$

$$\text{edges: } \{a,b\},\ \{b,d\},\ \{a,c\},\ \{d,e\},\ \{c,f\}.$$

□

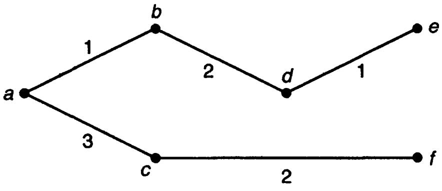

Figure 11.37

We conclude this section by discussing another practical problem, called the *minimum connector problem*. Its practicality is illustrated in the next example.

Example. There are n cities $A_1, A_2, \ldots, A_n$, and it is desired to connect some of them by highways so that each city is accesssible from any other. The cost of constructing a direct highway between city A_i and city A_j is estimated to be $c\{A_i, A_j\}$. Determine which cities should be directly connected by highways to minimize the total construction costs.

Since we are to minimize the total construction costs, a solution of the problem corresponds to a tree[57] with vertices $A_1, A_2, \ldots, A_n$, in which there is an edge joining cities A_i and A_j if and only if we put a direct highway between A_i and A_j. Indeed, if we consider the complete graph K_n with the n vertices $A_1, A_2, \ldots, A_n$, whose edges are weighted by the construction costs in the problem, then we seek a spanning tree

[57]If we did not have a tree, we could eliminate one or more of the highways without destroying the accessibility feature and thereby reduce costs.

the sum of whose edge weights is as small as possible. In what follows, we give two algorithms to solve the "minimum weight spanning tree problem" for any weighted connected graph. □

Let $G = (V, E)$ be a weighted connected graph with weight function c. We define the *weight of a subgraph* H of G to be

$$c(H) = \sum_{\{\alpha \text{ an edge of } H\}} c(\alpha),$$

the sum of the weights of the edges of H. A spanning tree of G that has the smallest weight of all spanning trees of G is a *minimum weight spanning tree.* If all the edges of G have the same weight, then every spanning tree of G is a minimum weight spanning tree. Given any connected graph, by appropriately assigning weights to its edges, we can make any spanning tree the unique minimum weight spanning tree. We now describe an algorithm known as Kruskal's algorithm.[58] This algorithm is also known as a *greedy algorithm*, since, at each stage, we choose an edge of smallest weight consistent with the fact that, upon termination, the chosen edges are to be the edges of a spanning tree. Consistency is simply the idea that we should never choose edges which can be used to create a cycle.

Greedy algorithm for a minimum weight spanning tree

Let $G = (V, E)$ be a weighted connected graph with weight function c.

(1) Put $F = \emptyset$.

(2) While there exists an edge α not in F such that $F \cup \{\alpha\}$ does not contain the edges of a cycle of G, determine such an edge α of minimum weight and put α in F.

(3) Put $T = (V, F)$.

Theorem 11.7.5 *Let $G = (V, E)$ be a weighted connected graph with weight function c. Then the preceding greedy algorithm constructs a minimum weight spanning tree $T = (V, F)$ of G.*

Proof. In the greedy algorithm, we begin with $n = |V|$ vertices and no edges (initially $F = \emptyset$), and hence with a spanning graph (V, F) with n connected components. Choosing an edge α that does not create a cycle means that α joins vertices in different components of (V, F), and hence putting α in F decreases the number of connected

[58]J. B. Kruskal, Jr., On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem, *Proc. Amer. Math. Soc.*, 7 (1956), 48–50.

components by 1. On termination, we have $n-1$ edges in F, and hence $T=(V,F)$ is a spanning tree. We now show that T is a minimum weight spanning tree.

Let the $n-1$ edges of F be $\alpha_1, \alpha_2, \ldots, \alpha_{n-1}$ in the order that they are put in F. Let $T^*=(V,F^*)$ be a minimum weight spanning tree, which has the largest number of edges in common with T. Thus, no minimum weight spanning tree has more edges in common with F than F^* does. If we can show that $F^*=F$, then it follows that T is a minimum weight spanning tree. Suppose, to the contrary, that $F^* \neq F$. Let α_k be the first edge of F that is not in F^*. Thus, the edges $\alpha_1, \ldots, \alpha_{k-1}$ all belong to F^*. By Theorem 11.5.8, there is an edge β of T^* such that the graph T^{**}, obtained from T^* by inserting α_k and deleting β, is a spanning tree of G. The edge β is an edge of the cycle that is created by inserting the edge α_k into T^*; since T is a tree, at least one of the edges of the cycle does not belong to T, and we choose such an edge β. We have

$$c(T^{**}) = c(T^*) - c(\beta) + c(\alpha_k). \tag{11.6}$$

Since T^* is a minimum weight spanning tree, we conclude that

$$c(\alpha_k) \geq c(\beta). \tag{11.7}$$

Because $L=\{\alpha_1, \ldots, \alpha_{k-1}, \beta\}$ is a subset of the edges of T^*, no cycle has all its edges contained in L. Hence, in determining the kth edge to be put in F in carrying out the greedy algorithm, β is a possible choice. It thus follows from (11.7) that

$$c(\alpha_k) = c(\beta)$$

and from Theorem 11.7.5 that T^{**} is also a minimum weight spanning tree. Since T^{**} has one more edge[59] in common with T than T^* has, we contradict our choice of T^*; the proof of the theorem is complete. □

Example. Let G be the weighted graph of order 7, shown in Figure 11.38, where the numbers next to the edges are their weights. In applying the greedy algorithm to determine a minimum weight spanning tree of G, we often have more than one good choice for the next edge. One way to carry out the greedy algorithm for the weighted graph in Figure 11.38 is to choose, in order, the edges

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

The weight of the resulting spanning tree T is

$$C(T) = 1+1+2+3+4+4 = 15.$$

Note that the algorithm does not grow the tree T in the sense that we have previously used that term. □

[59]The edge α_k.

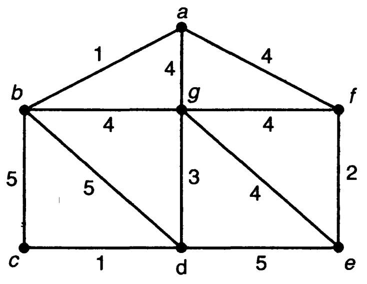

Figure 11.38

The best way to carry out the greedy algorithm is to arrange the edges in a sequence from smallest to largest weight and then iteratively select the first edge[60] that does not create a cycle. A disadvantage of the greedy algorithm is that one has to be able to recognize when a new edge creates a cycle and thus cannot be chosen. Prim[61] modified the greedy algorithm by showing how to grow a minimum weight spanning tree, thereby making it unnecessary to deal with cycles.

Prim's algorithm for a minimum weight spanning tree

Let $G = (V, E)$ be a weighted connected graph with weight function c and let u be any vertex of G.

(1) Put $i = 1$, $U_1 = \{u\}$, $F_1 = \emptyset$ and $T_1 = (U_1, F_1)$.

(2) For $i = 1, 2, \ldots, n-1$, do the following:

  (i) Locate an edge $\alpha_i = \{x, y\}$ of smallest weight such that x is in U_i and y is not in U_i.

  (ii) Put $U_{i+1} = U_i \cup \{y\}$, $F_{i+1} = F_i \cup \{\alpha_{i+1}\}$ and $T_{i+1} = (U_{i+1}, F_{i+1})$.

  (iii) Increase i to $i + 1$.

(3) Output $T_{n-1} = (U_{n-1}, F_{n-1})$. (Here $U_{n-1} = V$.)

Theorem 11.7.6 *Let $G = (V, E)$ be a weighted graph with weight function c. Then Prim's algorithm constructs a minimum weight spanning tree $T = (V, F)$ of G.*

[60]This is the greedy feature of the algorithm.

[61]R. C. Prim: Shortest Connection Networks and Some Generalizations, *Bell Systems Tech. J.*, 36 (1957), 1389–1401.

Proof. The proof is similar to the proof of Theorem 11.7.5. We use the same notation as in that proof, and we shall also be brief. At the end of each stage of the algorithm, we have grown a tree on a subset of the vertices of G. The theorem asserts that the tree $T = T_{n-1} = (V, F_{n-1})$ at termination of the algorithm, is a minimum weight spanning tree. Of all the minimum weight spanning trees of G, let $T^* = (V, F^*)$ be one for which the edges $\alpha_1, \ldots, \alpha_{k-1}$ are in T^* and k is largest. Suppose that $k \neq n$, that is, that $T^* \neq T$. Then α_k is not in F^* where α_k joins a vertex in U_k to a vertex in its complement $\overline{U_k}$. Since T^* is a spanning tree, there is an edge β of T^* that joins a vertex in U_k to a vertex in $\overline{U_k}$ such that inserting α_k in T^* and deleting β gives a spanning tree T^{**}. We have $c(\beta) \leq c(\alpha_k)$. Since α_k has the smallest weight of all edges with one vertex in U_k and the other in $\overline{U_k}$, it follows that $c(\beta) = c(\alpha_k)$ and T^{**} is a minimum weight spanning tree with one more edge in common with T. □

Example. We apply Prim's algorithm to the weighted graph G in Figure 11.38, with the initial vertex equal to a. One way of carrying out the algorithm results in the edges (in the order they are chosen)

$$\{a,b\}, \{a,f\}, \{f,e\}, \{e,g\}, \{g,d\}, \{d,c\},$$

which gives a spanning tree of weight 15. The advantage of Prim's algorithm over the greedy algorithm is clear in that, at each stage, we have only to determine an edge of smallest weight which joins a vertex that has already been reached to a vertex not yet reached. In the algorithm, cycles are automatically avoided in contrast to the greedy algorithm in which cycles must be explicitly avoided. □

11.8 Exercises

1. How many nonisomorphic graphs of order 1 are there? of order 2? of order 3? Explain why the answer to each of the preceding questions is ∞ for general graphs.

2. Determine each of the 11 nonisomorphic graphs of order 4, and give a planar representation of each.

3. Does there exist a graph of order 5 whose degree sequence equals $(4, 4, 3, 2, 2)$?

4. Does there exist a graph of order 5 whose degree sequence equals $(4, 4, 4, 2, 2)$? a multigraph?

5. Use the pigeonhole principle to prove that a graph of order $n \geq 2$ always has two vertices of the same degree. Does the same conclusion hold for multigraphs?

6. Let $(d_1, d_2, \ldots, d_n)$ be a sequence of n nonnegative even integers. Prove that there exists a general graph with this sequence as its degree sequence.

7. Let $(d_1, d_2, \ldots, d_n)$ be a sequence of n nonnegative integers whose sum $d_1 + d_2 + \cdots + d_n$ is even. Prove that there exists a general graph with this sequence as its degree sequence. Devise an algorithm to construct such a general graph.

8. Let G be a graph with degree sequence $(d_1, d_2, \ldots, d_n)$. Prove that, for each k with $0 < k < n$,
$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min\{k, d_i\}.$$

9. Draw a connected graph whose degree sequence equals
$$(5, 4, 3, 3, 3, 3, 3, 2, 2).$$

10. Prove that any two connected graphs of order n with degree sequence $(2, 2, \ldots, 2)$ are isomorphic.

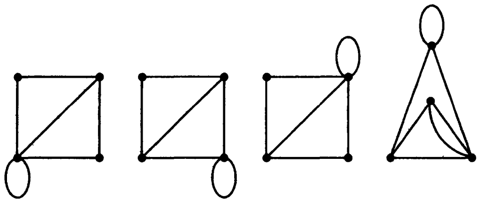

Figure 11.39

11. Determine which pairs of the general graphs in Figure 11.39 are isomorphic and, if isomorphic, find an isomorphism.

12. Determine which pairs of the graphs in Figure 11.40 are isomorphic, and for those that are isomorphic, find an isomorphism.

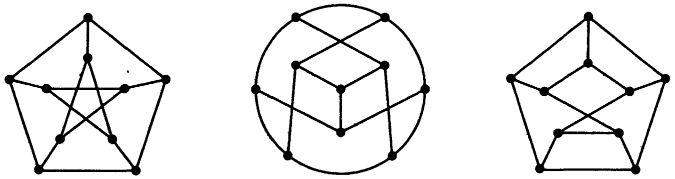

Figure 11.40

13. Prove that, if two vertices of a general graph are joined by a walk, then they are joined by a path.

14. Let x and y be vertices of a general graph, and suppose that there is a closed walk containing both x and y. Must there be a closed trail containing both x and y?

15. Let x and y be vertices of a general graph, and suppose that there is a closed trail containing both x and y. Must there be a cycle containing both x and y?

16. Let G be a connected graph of order 6 with degree sequence $(2,2,2,2,2,2)$.

    (a) Determine all the nonisomorphic induced subgraphs of G.

    (b) Determine all the nonisomorphic spanning subgraphs of G.

    (b) Determine all the nonisomorphic subgraphs of order 6 of G.

17. First, prove that any two multigraphs G of order 3 with degree sequence $(4,4,4)$ are isomorphic. Then

    (a) Determine all the nonisomorphic induced subgraphs of G.

    (b) Determine all the nonisomorphic spanning subgraphs of G.

    (b) Determine all the nonisomorphic subgraphs of order 3 of G.

18. Let γ be a trail joining vertices x and y in a general graph. Prove that the edges of γ can be partitioned so that one part of the partition determines a path joining x and y and the other parts determine cycles.

19. Let G be a general graph and let G' be the graph obtained from G by deleting all loops and all but one copy of each edge with multiplicity greater than 1. Prove that G is connected if and only if G' is connected. Also prove that G is planar if and only if G' is planar.

20. Prove that a graph of order n with at least

$$\frac{(n-1)(n-2)}{2}+1$$

edges must be connected. Give an example of a disconnected graph of order n with one fewer edge.

21. Let G be a general graph with exactly two vertices x and y of odd degree. Let G^* be the general graph obtained by putting a new edge $\{x,y\}$ joining x and y. Prove that G is connected if and only if G^* is connected.

22. (This and the following two exercises prove Theorem 11.1.3.) Let $G=(V,E)$ be a general graph. If x and y are in V, define $x \sim y$ to mean that either $x=y$ or there is a walk joining x and y. Prove that, for all vertices x, y, and z, we have

    (a) $x \sim x$.

(b) $x \sim y$ if and only if $y \sim x$.

(c) if $x \sim y$ and $y \sim z$, then $x \sim z$.

23. (Continuation of Exercise 22.) For each vertex x, let

$$C(x) = \{z : x \sim z\}.$$

Prove the following:

(i) For all vertices x and y, either $C(x) = C(y)$ or else $C(x) \cap C(y) = \emptyset$. In other words two of the sets $C(x)$ and $C(y)$ cannot intersect unless they are equal.

(ii) If $C(x) \cap C(y) = \emptyset$, then there does not exist an edge joining a vertex in $C(x)$ to a vertex in $C(y)$.

24. (Continuation of Exercise 23.) Let $V_1, V_2, \ldots, V_k$ be the different sets that occur among the $C(x)$'s. Prove the following:

(i) $V_1, V_2, \ldots, V_k$ form a partition of the vertex set V of G.

(ii) The general subgraphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2), \ldots, G_k = (V_k, E_k)$ of G induced by $V_1, V_2, \ldots, V_k$, respectively, are connected.

The induced subgraphs $G_1, G_2, \ldots, G_k$ are the *connected components* of G.

25. Prove Theorem 11.1.4.

26. Determine the adjacency matrices of the first and second general graphs in Figure 11.39.

27. Determine the adjacency matrices of the first and second graphs in Figure 11.40.

28. Let A and B be two n-by-n matrices of numbers whose entries are denoted by a_{ij} and b_{ij}, $(1 \leq i, j \leq n)$, respectively. Define the product $A \times B$ to be the n-by-n matrix C whose entry c_{ij} in row i and column j is given by

$$c_{ij} = \sum_{p=1}^{n} a_{ip}b_{pj}, \quad (1 \leq i, j \leq n).$$

If k is a positive integer, define

$$A^k = A \times A \times \cdots \times A \quad (k\ A's).$$

Now let A denote the adjacency matrix of a general graph of order n with vertices $a_1, a_2, \ldots, a_n$. Prove that the entry in row i, column j of A^k equals the number of walks of length k in G joining vertices a_i and a_j.

29. Determine if the multigraphs in Figure 11.41 have Eulerian trails (closed or open). In case there is an Eulerian trail, use the algorithms prsented in this chapter to construct one.

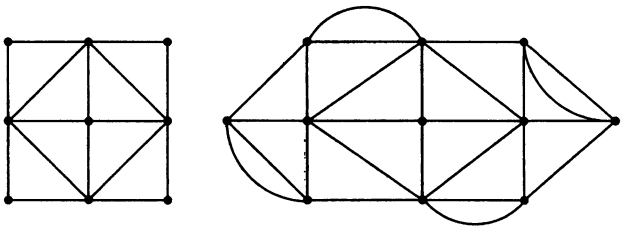

Figure 11.41

30. Which complete graphs K_n have closed Eulerian trails? open Eulerian trails?

31. Prove Theorem 11.2.4.

32. What is the fewest number of open trails into which the edges of GraphBuster can be partitioned?

33. Show how, removing pencil from paper the fewest number of times, to trace the plane graphs in Figures 11.15, 11.16, and 11.17.

34. Determine all nonisomorphic graphs of order at most 6 that have a closed Eulerian trail.

35. Show how, removing pencil from paper the fewest number of times, to trace out the graph of the regular dodecahedron shown in Figure 11.18.

36. Let G be a connected graph. Let γ be a closed walk that contains each edge of G at least once. Let G^* be the multigraph obtained from G by increasing the multiplicity of each edge from 1 to the number of times it occurs in γ. Prove that γ is a closed Eulerian trail in G^*. Conversely, suppose we increase the multiplicity of some of the edges of G and obtain a multigraph with m edges, each of whose vertices has even degree. Prove that there is a closed walk in G of length m which contains each edge of G at least once. This exercise shows that the Chinese postman problem for G is equivalent to determining the smallest number of copies of the edges of G that need to be inserted so as to obtain a multigraph all of whose vertices have even degree.

37. Solve the Chinese postman problem for the complete graph K_6.

38. Solve the Chinese postman problem for the graph obtained from K_6 by removing any edge.

39. Call a graph *cubic* if each vertex has degree equal to 3. The complete graph K_4 is the smallest example of a cubic graph. Find an example of a connected, cubic graph that does not have a Hamilton path.

40. * Let G be a graph of order n having at least
$$\frac{(n-1)(n-2)}{2}+2$$
edges. Prove that G has a Hamilton cycle. Exhibit a graph of order n with one fewer edge that does not have a Hamilton cycle.

41. Let $n \geq 3$ be an integer. Let G_n be the graph whose vertices are the $n!$ permutations of $\{1, 2, \ldots, n\}$, wherein two permutations are joined by an edge if and only if one can be obtained from the other by the interchange of two numbers (an arbitrary transposition). Deduce from the results of Section 4.1 that G_n has a Hamilton cycle.

42. Prove Theorem 11.3.4.

43. Devise an algorithm analogous to our algorithm for a Hamilton cycle that constructs a Hamilton path in graphs satisfying the condition given in Theorem 11.3.4.

44. Which complete bipartite graphs $K_{m,n}$ have Hamilton cycles? Which have Hamilton paths?

45. Prove that a multigraph is bipartite if and only if each of its connected components is bipartite.

46. Prove that $K_{m,n}$ is isomorphic to $K_{n,m}$.

47. Prove that a bipartite multigraph with an odd number of vertices does not have a Hamilton cycle.

48. Is GraphBuster a bipartite graph? If so, find a bipartition of its vertices. What if we delete the loops?

49. Let $V = \{1, 2, \ldots, 20\}$ be the set of the first 20 positive integers. Consider the graphs whose vertex set is V and whose edge sets are defined below. For each graph, investigate whether the graph (i) is connected (if not connected, determine the connected components), (ii) is bipartite, (iii) has an Eulerian trail, and (iv) has a Hamilton path.

    (a) $\{a, b\}$ is an edge if and only if $a + b$ is even.

    (b) $\{a, b\}$ is an edge if and only if $a + b$ is odd.

(c) $\{a, b\}$ is an edge if and only if $a \times b$ is even.

(d) $\{a, b\}$ is an edge if and only if $a \times b$ is odd.

(e) $\{a, b\}$ is an edge if and only if $a \times b$ is a perfect square.

(f) $\{a, b\}$ is an edge if and only if $a - b$ is divisible by 3.

50. What is the smallest number of edges that can be removed from K_5 to leave a bipartite graph?

51. Find a knight's tour on the boards of the following sizes:

    (a) 5-by-5
    (b) 6-by-6
    (c) 7-by-7

52. * Prove that there does not exist a knight's tour on a 4-by-4 board.

53. Prove that a graph is a tree if and only if it does not contain any cycles, but the insertion of any new edge always creates exactly one cycle.

54. Which trees have an Eulerian path?

55. Which trees have a Hamilton path?

56. Grow all the nonisomorphic trees of order 7.

57. Let $(d_1, d_2, \ldots, d_n)$ be a sequence of integers.

    (a) Prove that there is a tree of order n with this degree sequence if and only if $d_1, d_2, \ldots, d_n$ are positive integers with sum $d_1 + d_2 + \cdots + d_n = 2(n-1)$.
    (b) Write an algorithm that, starting with a sequence $(d_1, d_2, \ldots, d_n)$ of positive integers, either constructs a tree with this degree sequence or concludes that none is possible.

58. A *forest* is a graph each of whose connected components is a tree. In particular, a tree is a forest. Prove that a graph is a forest if and only if it does not have any cycles.

59. Prove that the removal of an edge from a tree leaves a forest of two trees.

60. Let G be a forest of k trees. What is the fewest number of edges that can be inserted in G in order to obtain a tree?

61. Determine a spanning tree for GraphBuster.

62. Prove that, if a tree has a vertex of degree p, then it has at least p pendent vertices.

63. Determine a spanning tree for each of the graphs in Figures 11.15 through 11.17.

64. For each integer $n \geq 3$ and for each integer k with $2 \leq k \leq n - 1$, construct a tree of order n with exactly k pendent vertices.

65. Use the algorithm for a spanning tree in Section 11.5 to construct a spanning tree of the graph of the dodecahedron.

66. How many cycles does a connected graph of order n with n edges have?

67. Let G be a graph of order n that is not necessarily connected. A forest is defined in Exercise 58. A *spanning forest* of G is a forest consisting of a spanning tree of each of the connected components of G. Modify the algorithm for a spanning tree given in Section 11.5 so that it constructs a spanning forest of G.

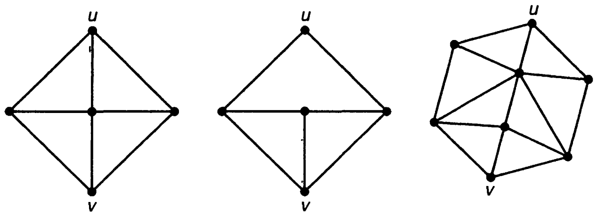

Figure 11.42

68. Determine whether the Shannon switching games played on the graphs in Figure 11.42 are positive, negative, or neutral games.

69. Let G be a connected multigraph. An *edge-cut* of G is a set F of edges whose removal disconnects G. An edge-cut F is *minimal*, provided that no subset of F other than F itself is an edge-cut. Prove that a bridge is always a minimal edge-cut, and conclude that the only minimal edge-cuts of a tree are the sets consisting of a single edge.

70. Let G be a connected multigraph having a vertex of degree k. Prove that G has a minimal edge-cut F with $|F| \leq k$.

71. Let F be a minimal edge-cut of a connected multigraph $G = (V, E)$. Prove that there exists a subset U of V such that F is precisely the set of edges that join a vertex in U to a vertex in the complement $\overline{U}$ of U.

72. (Continuation of Exercise 71.) Prove that a spanning tree of a connected multigraph contains at least one edge of every edge-cut.

73. Use the algorithm for growing a spanning tree in Section 11.7 in order to grow a spanning tree of GraphBuster. (Note: GraphBuster is a general graph and has loops and edges of multiplicity greater than 1. The loops can be ignored and only one copy of each edge need be considered.)

74. Use the algorithm for growing a spanning tree in order to grow a spanning tree of the graph of the regular dodecahedron.

75. Apply the BF-algorithm of Section 11.7 to determine a BFS-tree for the following:

    (a) The graph of the regular dodecahedron (any root)

    (b) GraphBuster (any root)

    (c) A graph of order n whose edges are arranged in a cycle (any root)

    (d) A complete graph K_n (any root)

    (e) A complete bipartite graph $K_{m,n}$ (a left-vertex root and a right-vertex root)

    In each case, determine the breadth-first numbers and the distance of each vertex from the root chosen.

76. Apply the DF-algorithm of Section 11.7 to determine a DFS-tree for (a), (b), (c), (d), and (e) as in Exercise 75. In each case, determine the depth-first numbers.

77. Let G be a graph that has a Hamilton path which joins two vertices u and v. Is the Hamilton path a DFS-tree rooted at u for G? Could there be other DFS-trees?

78. (Solution of the Chinese postman problem for trees.) Let G be a tree of order n. Prove that the length of a shortest closed walk that includes each edge of G at least once is $2(n-1)$. Show how the depth-first algorithm finds a walk of length $2(n-1)$ that includes each edge exactly twice.

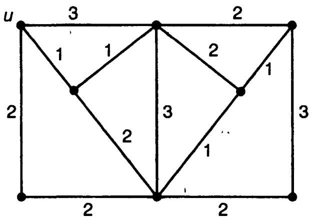

Figure 11.43

79. Use Dijkstra's algorithm in order to construct a distance tree for u for the weighted graph in Figure 11.43, with specified vertex u as shown.

80. Consider the complete graph K_n with labeled vertices $1, 2, \ldots, n$, in which the edge joining vertices i and j is weighted by $c\{i, j\} = i + j$ for all $i \neq j$. Use Dijkstra's algorithm to construct a distance tree rooted at vertex $u = 1$ for

    (a) K_4
    (b) K_6
    (c) K_8

81. Consider the complete graph K_n with labeled vertices $1, 2, \ldots, n$, with the weight function $c\{i, j\} = |i - j|$ for all $i \neq j$. Use Dijkstra's algorithm to construct a distance tree rooted at vertex $u = 1$ for

    (a) K_4
    (b) K_6
    (c) K_8

82. Consider the complete graph K_n whose edges are weighted as in Exercise 80. Apply the greedy algorithm to determine a minimum weight spanning tree for

    (a) K_4
    (b) K_6
    (c) K_8

83. Consider the complete graph K_n whose edges are weighted as in Exercise 81. Apply the greedy algorithm to determine a minimum weight spanning tree for

    (a) K_4
    (b) K_6
    (c) K_8

84. Same as Exercise 82, using Prim's algorithm in place of the greedy algorithm.

85. Same as Exercise 83, using Prim's algorithm in place of the greedy algorithm.

86. Let G be a weighted connected graph in which all edge weights are different. Prove that there is exactly one spanning tree of minimum weight.

87. Define a *caterpillar* to be a tree T that has a path γ such that every edge of T is either an edge of γ or has one of its vertices on γ.

    (a) Verify that all trees with six or fewer vertices are caterpillars.

(b) Let T_7 be the tree on seven vertices consisting of three paths of length 2 meeting at a central vertex c. Prove that T_7 is the only tree on 7 vertices that is not a caterpillar.

(c) Prove that a tree is a caterpillar if and only if it does not contain T_7 as a spanning subgraph.

88. Let $d_1, d_2, \ldots, d_n$ be positive integers. Prove that there is a caterpillar with degree sequence $(d_1, d_2, \ldots, d_n)$ if and only if $d_1 + d_2 + \cdots + d_n = 2(n-1)$. Compare with Exercise 57.

89. A *graceful labeling* of a graph G with vertex set V and with m edges is an injective function $g : V \rightarrow \{0, 1, 2, \ldots, m\}$ such that the labels $|g(x) - g(y)|$ corresponding to the m edges $\{x, y\}$ of G are $1, 2, \ldots, m$ in some order. It has been *conjectured* by Kotzig and Ringel (1964) that every tree has a graceful labeling. Find a graceful labeling of the tree T_7 in the previous exercise, any path, and the graph $K_{1,n}$.

90. Verify that cycles of lengths 5 and 6 cannot be gracefully labeled. Then find graceful labelings of cycles of lengths 7 and 8.

91. Let G be a graph with n vertices $x_1, x_2, \ldots, x_n$. Let r_i be the largest of the distances of x_i to the other vertices of G. Then

$$d(G) = \max\{r_1, r_2, \ldots, r_n\} \text{ and } r(G) = \min\{r_1, r_2, \ldots, r_n\}$$

are called. respectively, the *diameter* and *radius* of G. The *center* of G is the subgraph of G induced by the set of those vertices x_i for which $r_i = r(G)$, Prove the following assertions:

(a) Determine the radius, diameter, and center of the complete bipartite graph $K_{m,n}$.

(b) Determine the radius, diameter, and center of a cycle graph C_n.

(c) Determine the radius, diameter, and center of a path with n vertices.

(d) Determine the radius, diameter, and center of the graph Q_n corresponding to the vertices and edges of an n-dimensional cube.

92. Prove the following assertions.

(a) The center of a tree T is either a single vertex or two vertices joined by an edge. (*Hint*: Use induction on the number n of vertices.)

(b) Let G be a graph, and let $\overline{G}$ be the *complement graph* obtained from G by putting an edge between two vertices of G provided there isn't one in G and removing all edges of G. Prove that if $d(G) \geq 3$, then $d(\overline{G}) \leq 3$.