This text is Chapter 8: Graph Theory
Schaum's Outline of Discrete Mathematics

# CHAPTER 8

# Graph Theory

## 8.1 INTRODUCTION, DATA STRUCTURES

Graphs, directed graphs, trees and binary trees appear in many areas of mathematics and computer science. This and the next two chapters will cover these topics. However, in order to understand how these objects may be stored in memory and to understand algorithms on them, we need to know a little about certain data structures. We assume the reader does understand linear and two-dimensional arrays; hence we will only discuss linked lists and pointers, and stacks and queues below.

### Linked Lists and Pointers

Linked lists and pointers will be introduced by means of an example. Suppose a brokerage firm maintains a file in which each record contains a customer's name and salesman; say the file contains the following data:

| Customer | Adams | Brown | Clark | Drew | Evans | Farmer | Geller | Hiller | Infeld |
|---|---|---|---|---|---|---|---|---|---|
| Salesman | Smith | Ray | Ray | Jones | Smith | Jones | Ray | Smith | Ray |

There are two basic operations that one would want to perform on the data:

*Operation A*: Given the name of a customer, find his salesman.

*Operation B*: Given the name of a salesman, find the list of his customers.

We discuss a number of ways the data may be stored in the computer, and the ease with which one can perform the operations $A$ and $B$ on the data.

Clearly, the file could be stored in the computer by an array with two rows (or columns) of nine names. Since the customers are listed alphabetically, one could easily perform operation $A$. However, in order to perform operation $B$ one must search through the entire array.

One can easily store the data in memory using a two-dimensional array where, say, the rows correspond to an alphabetical listing of the customers and the columns correspond to an alphabetical listing of the salesmen, and where there is a 1 in the matrix indicating the salesman of a customer and there are 0's elsewhere. The main drawback of such a representation is that there may be a waste of a lot of memory because many 0's may be in the matrix. For example, if a firm has 1000 customers and 20 salesmen, one would need 20 000 memory locations for the data, but only 1000 of them would be useful.

We discuss below a way of storing the data in memory which uses linked lists and pointers. By a *linked list*, we mean a linear collection of data elements, called *nodes*, where the linear order is given by means of a field

of pointers. Figure 8-1 is a schematic diagram of a linked list with six nodes. That is, each node is divided into two parts: the first part contains the information of the element (e.g., NAME, ADDRESS, . . .), and the second part, called the *link field* or *nextpointer field*, contains the address of the next node in the list. This pointer field is indicated by an arrow drawn from one node to the next node in the list. There is also a variable pointer, called START in Fig. 8-1, which gives the address of the first node in the list. Furthermore, the pointer field of the last node contains an invalid address, called a *null pointer*, which indicates the end of the list.

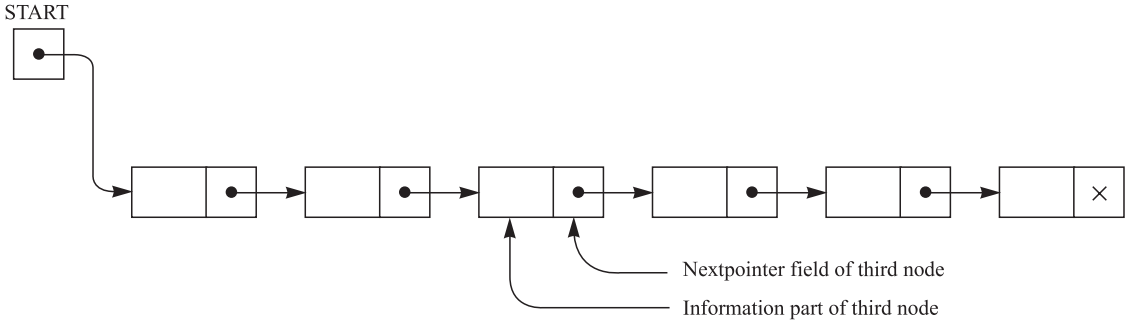



**Fig. 8-1** Linked list with 6 nodes

One main way of storing the original data pictured in Fig. 8-2, uses linked lists. Observe that there are separate (sorted alphabetically) arrays for the customers and the salesmen. Also, there is a pointer array SLSM parallel to CUSTOMER which gives the location of the salesman of a customer, hence operation *A* can be performed very easily and quickly. Furthermore, the list of customers of each salesman is a linked list as discussed above. Specifically, there is a pointer array START parallel to SALESMAN which points to the first customer of a salesman, and there is an array NEXT which points to the location of the next customer in the salesman's list (or contains a 0 to indicate the end of the list). This process is indicated by the arrows in Fig. 8-2 for the salesman Ray.

| | Customer | SLSM | NEXT |
|---|---|---|---|
| 1 | Adams | 3 | 5 |
| 2 | Brown | 2 | 3 |
| 3 | Clark | 2 | 7 |
| 4 | Drew | 1 | 6 |
| 5 | Evans | 3 | 8 |
| 6 | Farmer | 1 | 0 |
| 7 | Geller | 2 | 9 |
| 8 | Hill | 3 | 0 |
| 9 | Infeld | 2 | 0 |

| Salesman | START | |
|---|---|---|
| Jones | 4 | 1 |
| Ray | 2 | 2 |
| Smith | 1 | 3 |

**Fig. 8-2**

Operation *B* can now be performed easily and quickly; that is, one does not need to search through the list of all customers in order to obtain the list of customers of a given salesman. Figure 8-3 gives such an algorithm (which is written in pseudocode).

### Stacks, Queues, and Priority Queues

There are data structures other than arrays and linked lists which will occur in our graph algorithms. These structures, stacks, queues, and priority queues, are briefly described below.

(a) **Stack**: A *stack*, also called a *last-in first-out* (LIFO) system, is a linear list in which insertions and deletions can take place only at one end, called the "top" of the list. This structure is similar in its operation to a stack of dishes on a spring system, as pictured in Fig. 8-4(*a*). Note that new dishes are inserted only at the top of the stack and dishes can be deleted only from the top of the stack.

**Algorithm 8.1** The name of a salesman is read and the list of his customers is printed.

***Step 1.*** Read XXX.

***Step 2.*** Find $K$ such that SALESMAN[K] = XXX. [Use binary search.]

***Step 3.*** Set PTR := START[$K$]. [Initializes pointer PTR.]

***Step 4.*** Repeat while PTR $\neq$ NULL.

($a$) Print CUSTOMER[PTR].

($b$) Set PTR := NEXT[PTR]. [Update PTR.]

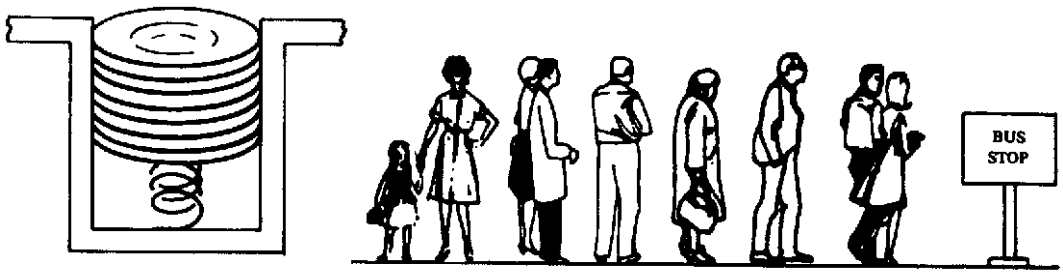
[End of loop.]

***Step 5.*** Exit.

**Fig. 8-3**

(b) **Queue**: A *queue*, also called a *first-in first-out* (FIFO) system, is a linear list in which deletions can only take place at one end of the list, the "front" of the list, and insertions can only take place at the other end of the list, the "rear" of the list. The structure operates in much the same way as a line of people waiting at a bus stop, as pictured in Fig. 8-4($b$). That is, the first person in line is the first person to board the bus, and a new person goes to the end of the line.

(c) **Priority queue**: Let $S$ be a set of elements where new elements may be periodically inserted, but where the current largest element (element with the "highest priority") is always deleted. Then $S$ is called a *priority queue*. The rules "women and children first" and "age before beauty" are examples of priority queues. Stacks and ordinary queues are special kinds of priority queues. Specifically, the element with the highest priority in a stack is the last element inserted, but the element with the highest priority in a queue is the first element inserted.




($a$) Stack of dishes. ($b$) Queue waiting for a bus.

**Fig. 8-4**

## 8.2 GRAPHS AND MULTIGRAPHS

A *graph* $G$ consists of two things:

(i) A set $V = V(G)$ whose elements are called *vertices*, *points*, or *nodes* of $G$.
(ii) A set $E = E(G)$ of unordered pairs of distinct vertices called *edges* of $G$.

We denote such a graph by $G(V, E)$ when we want to emphasize the two parts of $G$.

Vertices $u$ and $v$ are said to be *adjacent* or *neighbors* if there is an edge $e = \{u, v\}$. In such a case, $u$ and $v$ are called the *endpoints* of $e$, and $e$ is said to *connect* $u$ and $v$. Also, the edge $e$ is said to be *incident* on each of its endpoints $u$ and $v$. Graphs are pictured by diagrams in the plane in a natural way. Specifically, each vertex $v$ in $V$ is represented by a dot (or small circle), and each edge $e = \{v_1, v_2\}$ is represented by a curve which connects its endpoints $v_1$ and $v_2$ For example, Fig. 8-5($a$) represents the graph $G(V, E)$ where:

(i) $V$ consists of vertices $A$, $B$, $C$, $D$.

(ii) $E$ consists of edges $e_1 = \{A, B\}$, $e_2 = \{B, C\}$, $e_3 = \{C, D\}$, $e_4 = \{A, C\}$, $e_5 = \{B, D\}$.

In fact, we will usually denote a graph by drawing its diagram rather than explicitly listing its vertices and edges.

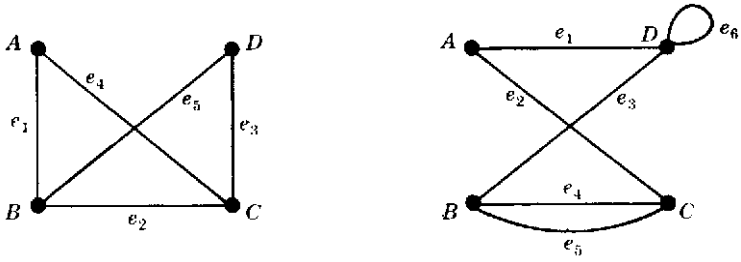



**Fig. 8-5**

### Multigraphs

Consider the diagram in Fig. 8-5($b$). The edges $e_4$ and $e_5$ are called *multiple edges* since they connect the same endpoints, and the edge $e_6$ is called a *loop* since its endpoints are the same vertex. Such a diagram is called a *multigraph*; the formal definition of a graph permits neither multiple edges nor loops. Thus a graph may be defined to be a multigraph without multiple edges or loops.

**Remark:** Some texts use the term *graph* to include multigraphs and use the term *simple graph* to mean a graph without multiple edges and loops.

### Degree of a Vertex

The *degree* of a vertex $v$ in a graph $G$, written deg $(v)$, is equal to the number of edges in $G$ which contain $v$, that is, which are incident on $v$. Since each edge is counted twice in counting the degrees of the vertices of $G$, we have the following simple but important result.

**Theorem 8.1:** The sum of the degrees of the vertices of a graph $G$ is equal to twice the number of edges in $G$.

Consider, for example, the graph in Fig. 8-5($a$). We have

$$\deg(A) = 2, \quad \deg(B) = 3, \quad \deg(C) = 3, \quad \deg(D) = 2.$$

The sum of the degrees equals 10 which, as expected, is twice the number of edges. A vertex is said to be *even* or *odd* according as its degree is an even or an odd number. Thus $A$ and $D$ are even vertices whereas $B$ and $C$ are odd vertices.

Theorem 8.1 also holds for multigraphs where a loop is counted twice toward the degree of its endpoint. For example, in Fig. 8-5($b$) we have $\deg(D) = 4$ since the edge $e_6$ is counted twice; hence $D$ is an even vertex.

A vertex of degree zero is called an *isolated* vertex.

### Finite Graphs, Trivial Graph

A multigraph is said to be *finite* if it has a finite number of vertices and a finite number of edges. Observe that a graph with a finite number of vertices must automatically have a finite number of edges and so must be finite. The finite graph with one vertex and no edges, i.e., a single point, is called the *trivial graph*. Unless otherwise specified, the multigraphs in this book shall be finite.

## 8.3 SUBGRAPHS, ISOMORPHIC AND HOMEOMORPHIC GRAPHS

This section will discuss important relationships between graphs.

### Subgraphs

Consider a graph $G = G(V, E)$. A graph $H = H(V', E')$ is called a *subgraph* of $G$ if the vertices and edges of $H$ are contained in the vertices and edges of $G$, that is, if $V' \subseteq V$ and $E' \subseteq E$. In particular:

(i) A subgraph $H(V', E')$ of $G(V, E)$ is called the subgraph *induced* by its vertices $V'$ if its edge set $E'$ contains all edges in $G$ whose endpoints belong to vertices in $H$.

(ii) If $v$ is a vertex in $G$, then $G - v$ is the subgraph of $G$ obtained by deleting $v$ from $G$ and deleting all edges in $G$ which contain $v$.

(iii) If $e$ is an edge in $G$, then $G - e$ is the subgraph of $G$ obtained by simply deleting the edge $e$ from $G$.

### Isomorphic Graphs

Graphs $G(V, E)$ and $G(V^*, E^*)$ are said to be *isomorphic* if there exists a one-to-one correspondence $f: V \to V^*$ such that $\{u, v\}$ is an edge of $G$ if and only if $\{f(u), f(v)\}$ is an edge of $G^*$. Normally, we do not distinguish between isomorphic graphs (even though their diagrams may "look different"). Figure 8-6 gives ten graphs pictured as letters. We note that $A$ and $R$ are isomorphic graphs. Also, $F$ and $T$ are isomorphic graphs, $K$ and $X$ are isomorphic graphs and $M$, $S$, $V$, and $Z$ are isomorphic graphs.

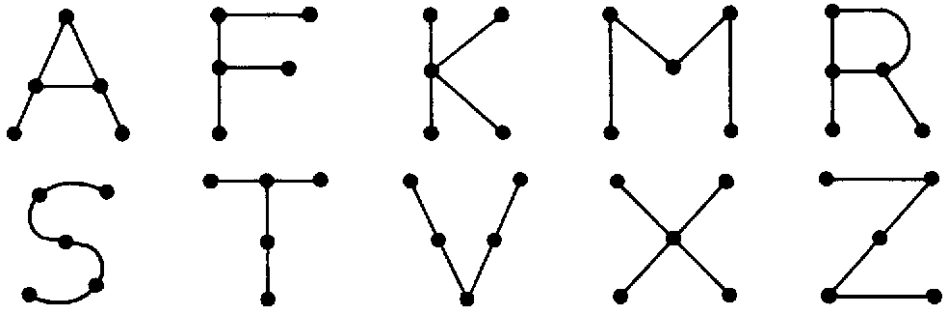


**Fig. 8-6**

### Homeomorphic Graphs

Given any graph $G$, we can obtain a new graph by dividing an edge of $G$ with additional vertices. Two graphs $G$ and $G^*$ are said to *homeomorphic* if they can be obtained from the same graph or isomorphic graphs by this method. The graphs $(a)$ and $(b)$ in Fig. 8-7 are not isomorphic, but they are homeomorphic since they can be obtained from the graph $(c)$ by adding appropriate vertices.

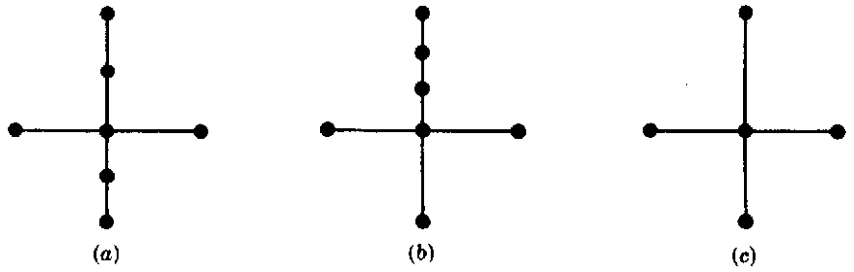



**Fig. 8-7**

## 8.4 PATHS, CONNECTIVITY

*A path* in a multigraph $G$ consists of an alternating sequence of vertices and edges of the form

$$v_0, \quad e_1, \quad v_1, \quad e_2, \quad v_2, \quad \ldots, \quad e_{n-1}, \quad v_{n-1}, \quad e_n, \quad v_n$$

where each edge $e_i$ contains the vertices $v_{i-1}$ and $v_i$ (which appear on the sides of $e_i$ in the sequence). The number $n$ of edges is called the *length* of the path. When there is no ambiguity, we denote a path by its sequence of vertices $(v_0, v_1, \ldots, v_n)$. The path is said to be *closed* if $v_0 = v_n$. Otherwise, we say the path is from $v_0$, to $v_n$ or *between* $v_0$ and $v_n$, or *connects* $v_0$ to $v_n$.

A *simple path* is a path in which all vertices are distinct. (A path in which all edges are distinct will be called a *trail*.) A *cycle* is a closed path of length 3 or more in which all vertices are distinct except $v_0 = v_n$. A cycle of length $k$ is called a *k-cycle*.

**EXAMPLE 8.1** Consider the graph $G$ in Fig. 8-8($a$). Consider the following sequences:

$$\alpha = (P_4, P_1, P_2, P_5, P_1, P_2, P_3, P_6), \quad \beta = (P_4, P_1, P_5, P_2, P_6),$$
$$\gamma = (P_4, P_1, P_5, P_2, P_3, P_5, P_6), \quad \delta = (P_4, P_1, P_5, P_3, P_6).$$

The sequence $\alpha$ is a path from $P_4$ to $P_6$; but it is not a trail since the edge $\{P_1, P_2\}$ is used twice. The sequence $\beta$ is not a path since there is no edge $\{P_2, P_6\}$. The sequence $\gamma$ is a trail since no edge is used twice; but it is not a simple path since the vertex $P_5$ is used twice. The sequence $\delta$ is a simple path from $P_4$ to $P_6$; but it is not the shortest path (with respect to length) from $P_4$ to $P_6$. The shortest path from $P_4$ to $P_6$ is the simple path $(P_4, P_5, P_6)$ which has length 2.

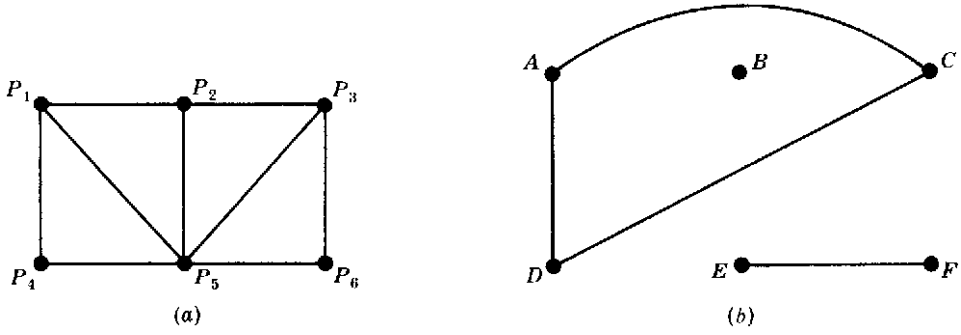



**Fig. 8-8**

By eliminating unnecessary edges, it is not difficult to see that any path from a vertex $u$ to a vertex $v$ can be replaced by a simple path from $u$ to $v$. We state this result formally.

**Theorem 8.2:** There is a path from a vertex $u$ to a vertex $v$ if and only if there exists a simple path from $u$ to $v$.

### Connectivity, Connected Components

A graph $G$ is *connected* if there is a path between any two of its vertices. The graph in Fig. 8-8($a$) is connected, but the graph in Fig. 8-8($b$) is not connected since, for example, there is no path between vertices $D$ and $E$.

Suppose $G$ is a graph. A connected subgraph $H$ of $G$ is called a *connected component* of $G$ if $H$ is not contained in any larger connected subgraph of $G$. It is intuitively clear that any graph $G$ can be partitioned into its connected components. For example, the graph $G$ in Fig. 8-8($b$) has three connected components, the subgraphs induced by the vertex sets $\{A, C, D\}$, $\{E, F\}$, and $\{B\}$.

The vertex $B$ in Fig. 8-8($b$) is called an *isolated vertex* since $B$ does not belong to any edge or, in other words, $\deg(B) = 0$. Therefore, as noted, $B$ itself forms a connected component of the graph.

**Remark:** Formally speaking, assuming any vertex $u$ is connected to itself, the relation "$u$ is connected to $v$" is an equivalence relation on the vertex set of a graph $G$ and the equivalence classes of the relation form the connected components of $G$.

### Distance and Diameter

Consider a connected graph $G$. The *distance* between vertices $u$ and $v$ in $G$, written $d(u, v)$, is the length of the shortest path between $u$ and $v$. The *diameter* of $G$, written $\text{diam}(G)$, is the maximum distance between any two points in $G$. For example, in Fig. 8-9($a$), $d(A, F) = 2$ and $\text{diam}(G) = 3$, whereas in Fig. 8-9($b$), $d(A, F) = 3$ and $\text{diam}(G) = 4$.

### Cutpoints and Bridges

Let $G$ be a connected graph. A vertex $v$ in $G$ is called a *cutpoint* if $G - v$ is disconnected. (Recall that $G - v$ is the graph obtained from $G$ by deleting $v$ and all edges containing $v$.) An edge $e$ of $G$ is called a *bridge* if $G - e$ is disconnected. (Recall that $G - e$ is the graph obtained from $G$ by simply deleting the edge $e$). In Fig. 8-9($a$), the vertex $D$ is a cutpoint and there are no bridges. In Fig. 8-9($b$), the edge $= \{D, F\}$ is a bridge. (Its endpoints $D$ and $F$ are necessarily cutpoints.)

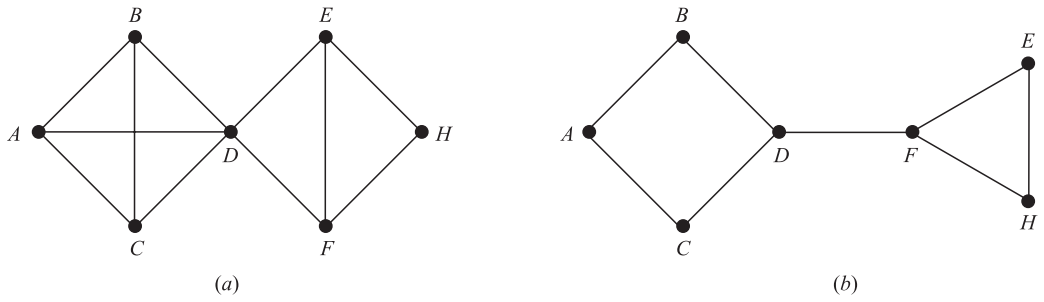
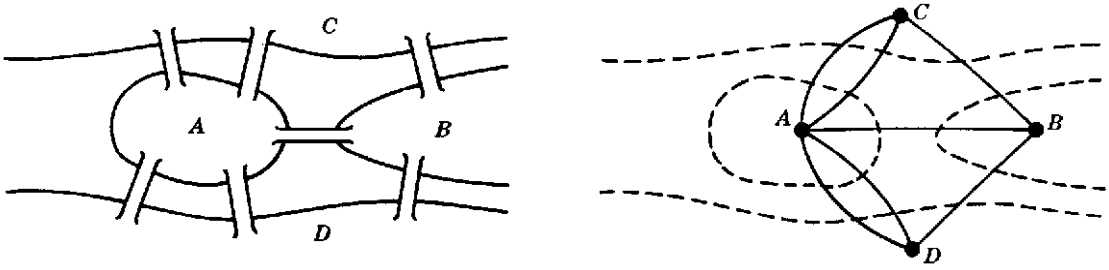



**Fig. 8-9**

## 8.5 TRAVERSABLE AND EULERIAN GRAPHS, BRIDGES OF KÖNIGSBERG

The eighteenth-century East Prussian town of Königsberg included two islands and seven bridges as shown in Fig. 8-10($a$). Question: Beginning anywhere and ending anywhere, can a person walk through town crossing all seven bridges but not crossing any bridge twice? The people of Königsberg wrote to the celebrated Swiss mathematician L. Euler about this question. Euler proved in 1736 that such a walk is impossible. He replaced the islands and the two sides of the river by points and the bridges by curves, obtaining Fig. 8-10($b$).

Observe that Fig. 8-10($b$) is a multigraph. A multigraph is said to be *traversable* if it "can be drawn without any breaks in the curve and without repeating any edges," that is, if there is a path which includes all vertices and uses each edge exactly once. Such a path must be a trail (since no edge is used twice) and will be called a *traversable trail*. Clearly a traversable multigraph must be finite and connected.




(a) Königsberg in 1736 (b) Euler's graphical representation

**Fig. 8-10**

We now show how Euler proved that the multigraph in Fig. 8-10(*b*) is not traversable and hence that the walk in Königsberg is impossible. Recall first that a vertex is even or odd according as its degree is an even or an odd number. Suppose a multigraph is traversable and that a traversable trail does not begin or end at a vertex $P$. We claim that $P$ is an even vertex. For whenever the traversable trail enters $P$ by an edge, there must always be an edge not previously used by which the trail can leave $P$. Thus the edges in the trail incident with $P$ must appear in pairs, and so $P$ is an even vertex. Therefore if a vertex $Q$ is odd, the traversable trail must begin or end at $Q$. Consequently, a multigraph with more than two odd vertices cannot be traversable. Observe that the multigraph corresponding to the Königsberg bridge problem has four odd vertices. Thus one cannot walk through Königsberg so that each bridge is crossed exactly once.

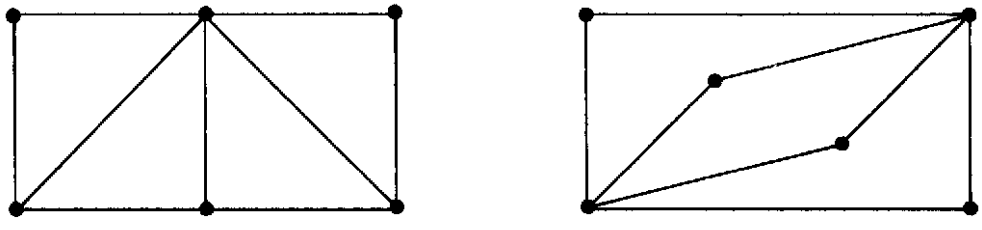
Euler actually proved the converse of the above statement, which is contained in the following theorem and corollary. (The theorem is proved in Problem 8.9.) A graph $G$ is called an *Eulerian* graph if there exists a closed traversable trail, called an *Eulerian* trail.

**Theorem 8.3 (Euler):** A finite connected graph is Eulerian if and only if each vertex has even degree.

**Corollary 8.4:** Any finite connected graph with two odd vertices is traversable. A traversable trail may begin at either odd vertex and will end at the other odd vertex.

### Hamiltonian Graphs

The above discussion of Eulerian graphs emphasized traveling edges; here we concentrate on visiting vertices. A *Hamiltonian circuit* in a graph $G$, named after the nineteenth-century Irish mathematician William Hamilton (1803–1865), is a closed path that visits every vertex in $G$ exactly once. (Such a closed path must be a cycle.) If $G$ does admit a Hamiltonian circuit, then $G$ is called a *Hamiltonian graph*. Note that an Eulerian circuit traverses every edge exactly once, but may repeat vertices, while a Hamiltonian circuit visits each vertex exactly once but may repeat edges. Figure 8-11 gives an example of a graph which is Hamiltonian but not Eulerian, and vice versa.



(a) Hamiltonian and non-Eulerian (b) Eulerian and non-Hamiltonian

**Fig. 8-11**

Although it is clear that only connected graphs can be Hamiltonian, there is no simple criterion to tell us whether or not a graph is Hamiltonian as there is for Eulerian graphs. We do have the following sufficient condition which is due to G. A. Dirac.

**Theorem 8.5:** Let $G$ be a connected graph with $n$ vertices. Then $G$ is Hamiltonian if $n \geq 3$ and $n \leq \deg(v)$ for each vertex $v$ in $G$.

## 8.6 LABELED AND WEIGHTED GRAPHS

A graph $G$ is called a *labeled graph* if its edges and/or vertices are assigned data of one kind or another. In particular, $G$ is called a *weighted graph* if each edge $e$ of $G$ is assigned a nonnegative number $w(e)$ called *the weight* or *length* of $v$. Figure 8-12 shows a weighted graph where the weight of each edge is given in the obvious way. The *weight* (or *length*) of a path in such a weighted graph $G$ is defined to be the sum of the weights of the edges in the path. One important problem in graph theory is to find a *shortest path*, that is, a path of minimum weight (length), between any two given vertices. The length of a shortest path between $P$ and $Q$ in Fig. 8-12 is 14; one such path is

$$(P, A_1, A_2, A_5, A_3, A_6, Q)$$

The reader can try to find another shortest path.

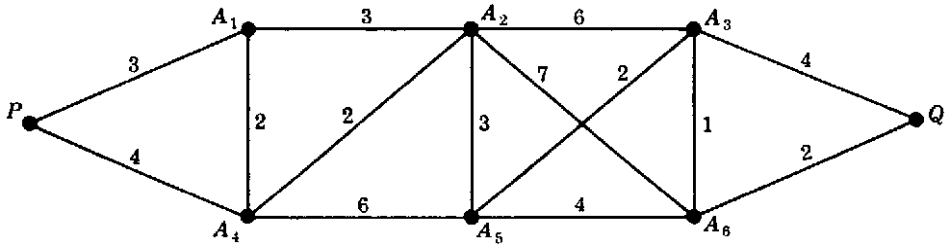



**Fig. 8-12**

## 8.7 COMPLETE, REGULAR, AND BIPARTITE GRAPHS

There are many different types of graphs. This section considers three of them: complete, regular, and bipartite graphs.

### Complete Graphs

A graph $G$ is said to be *complete* if every vertex in $G$ is connected to every other vertex in $G$. Thus a complete graph $G$ must be connected. The complete graph with $n$ vertices is denoted by $K_n$. Figure 8-13 shows the graphs $K_1$ through $K_6$.

### Regular Graphs

A graph $G$ is *regular of degree k* or *k-regular* if every vertex has degree $k$. In other words, a graph is regular if every vertex has the same degree.

The connected regular graphs of degrees 0, 1, or 2 are easily described. The connected 0-regular graph is the trivial graph with one vertex and no edges. The connected 1-regular graph is the graph with two vertices and one edge connecting them. The connected 2-regular graph with $n$ vertices is the graph which consists of a single $n$-cycle. See Fig. 8-14.

The 3-regular graphs must have an even number of vertices since the sum of the degrees of the vertices is an even number (Theorem 8.1). Figure 8-15 shows two connected 3-regular graphs with six vertices. In general, regular graphs can be quite complicated. For example, there are nineteen 3-regular graphs with ten vertices. We note that the complete graph with $n$ vertices $K_n$ is regular of degree $n - 1$.

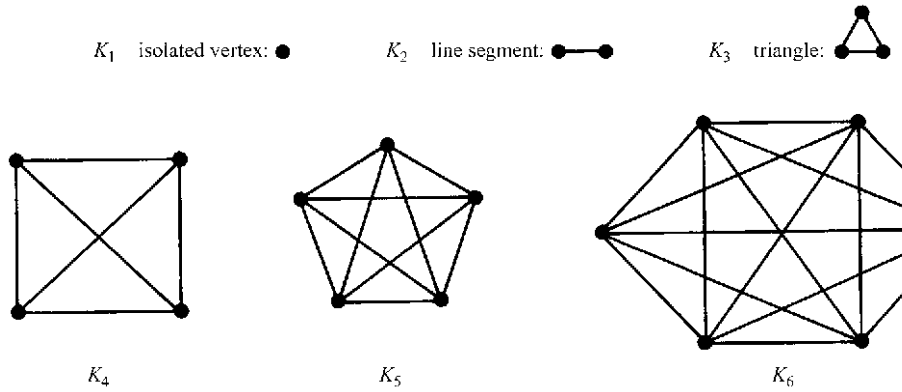



**Fig. 8-13**

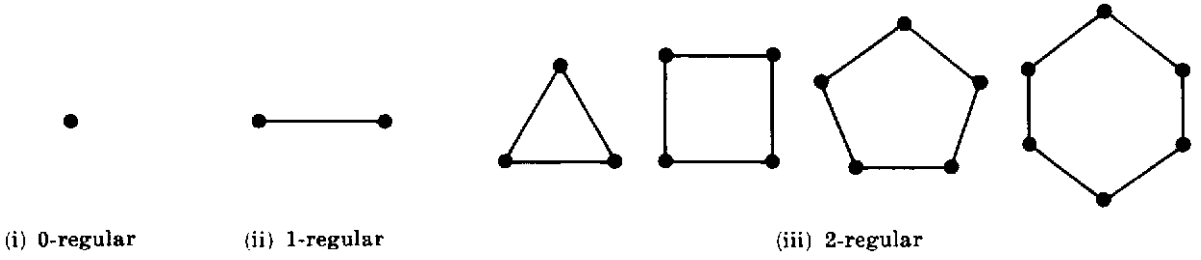



**Fig. 8-14**

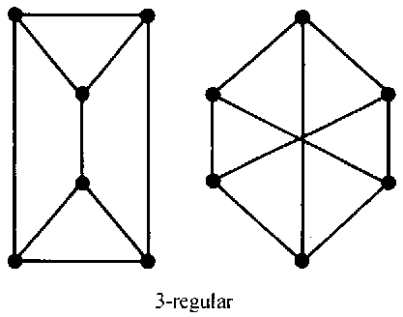



**Fig. 8-15**

### Bipartite Graphs

A graph $G$ is said to be *bipartite* if its vertices $V$ can be partitioned into two subsets $M$ and $N$ such that each edge of $G$ connects a vertex of $M$ to a vertex of $N$. By a complete bipartite graph, we mean that each vertex of $M$ is connected to each vertex of $N$; this graph is denoted by $K_{m,n}$ where $m$ is the number of vertices in $M$ and $n$ is the number of vertices in $N$, and, for standardization, we will assume $m \leq n$. Figure 8-16 shows the graphs $K_{2,3}$, $K_{3,3}$, and $K_{2,4}$, Clearly the graph $K_{m,n}$ has $mn$ edges.

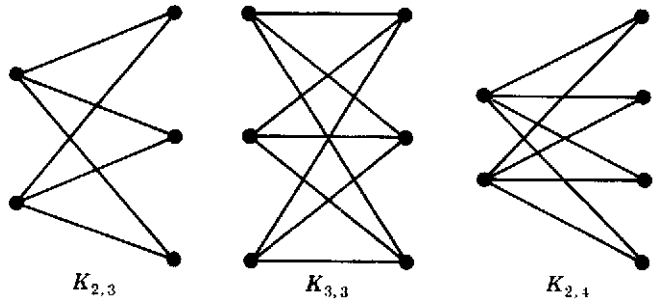



**Fig. 8-16**

## 8.8 TREE GRAPHS

A graph $T$ is called a *tree* if $T$ is connected and $T$ has no cycles. Examples of trees are shown in Fig. 8-17. *A forest* $G$ is a graph with no cycles; hence the connected components of a forest $G$ are trees. A graph without cycles is said to be *cycle-free*. The tree consisting of a single vertex with no edges is called the *degenerate tree*.

Consider a tree $T$. Clearly, there is only one simple path between two vertices of $T$; otherwise, the two paths would form a cycle. Also:

(a) Suppose there is no edge $\{u, v\}$ in $T$ and we add the edge $e = \{u, v\}$ to $T$. Then the simple path from $u$ to $v$ in $T$ and $e$ will form a cycle; hence $T$ is no longer a tree.
(b) On the other hand, suppose there is an edge $e = \{u, v\}$ in $T$, and we delete $e$ from $T$. Then $T$ is no longer connected (since there cannot be a path from $u$ to $v$); hence $T$ is no longer a tree.

The following theorem (proved in Problem 8.14) applies when our graphs are finite.

**Theorem 8.6:** Let $G$ be a graph with $n > 1$ vertices. Then the following are equivalent:

(i) $G$ is a tree.

(ii) $G$ is a cycle-free and has $n - 1$ edges.

(iii) $G$ is connected and has $n - 1$ edges.

This theorem also tells us that a finite tree $T$ with $n$ vertices must have $n - 1$ edges. For example, the tree in Fig. 8-17($a$) has 9 vertices and 8 edges, and the tree in Fig. 8-17($b$) has 13 vertices and 12 edges.

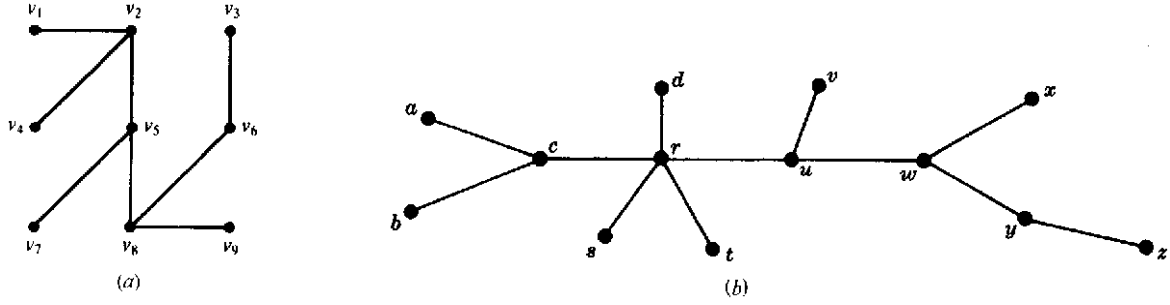



**Fig. 8-17**

### Spanning Trees

A subgraph $T$ of a connected graph $G$ is called a *spanning tree* of $G$ if $T$ is a tree and $T$ includes all the vertices of $G$. Figure 8-18 shows a connected graph $G$ and spanning trees $T_1$, $T_2$, and $T_3$ of $G$.

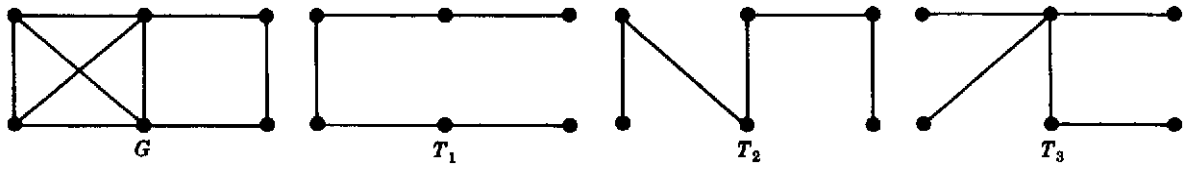



**Fig. 8-18**

### Minimum Spanning Trees

Suppose $G$ is a connected weighted graph. That is, each edge of $G$ is assigned a nonnegative number called the *weight* of the edge. Then any spanning tree $T$ of $G$ is assigned a total weight obtained by adding the weights of the edges in $T$. A *minimal spanning tree* of $G$ is a spanning tree whose total weight is as small as possible.

Algorithms 8.2 and 8.3, which appear in Fig. 8-19, enable us to find a minimal spanning tree $T$ of a connected weighted graph $G$ where $G$ has n vertices. (In which case T must have $n - 1$ vertices.)

**Algorithm 8.2:** The input is a connected weighted graph $G$ with $n$ vertices.

***Step 1.*** Arrange the edges of $G$ in the order of decreasing weights.

***Step 2.*** Proceeding sequentially, delete each edge that does not disconnect the graph until $n - 1$ edges remain.

***Step 3.*** Exit.

**Algorithm 8.3 (Kruskal):** The input is a connected weighted graph $G$ with $n$ vertices.

***Step 1.*** Arrange the edges of $G$ in order of increasing weights.

***Step 2.*** Starting only with the vertices of $G$ and proceeding sequentially, add each edge which does not result in a cycle until $n - 1$ edges are added.

***Step 3.*** Exit.

**Fig. 8-19**

The weight of a minimal spanning tree is unique, but the minimal spanning tree itself is not. Different minimal spanning trees can occur when two or more edges have the same weight. In such a case, the arrangement of the edges in Step 1 of Algorithms 8.2. or 8.3 is not unique and hence may result in different minimal spanning trees as illustrated in the following example.

**EXAMPLE 8.2** Find a minimal spanning tree of the weighted graph $Q$ in Fig. 8-20($a$). Note that $Q$ has six vertices, so a minimal spanning tree will have five edges.

(a) Here we apply Algorithm 8.2.

First we order the edges by decreasing weights, and then we successively delete edges without disconnecting $Q$ until five edges remain. This yields the following data:

| Edges | $BC$ | $AF$ | $AC$ | $BE$ | $CE$ | $BF$ | $AE$ | $DF$ | $BD$ |
|---|---|---|---|---|---|---|---|---|---|
| Weight | 8 | 7 | 7 | 7 | 6 | 5 | 4 | 4 | 3 |
| Delete | Yes | Yes | Yes | No | No | Yes | | | |

Thus the minimal spanning tree of $Q$ which is obtained contains the edges

$$BE, \quad CE, \quad AE, \quad DF, \quad BD$$

The spanning tree has weight 24 and it is shown in Fig. 8-20($b$).

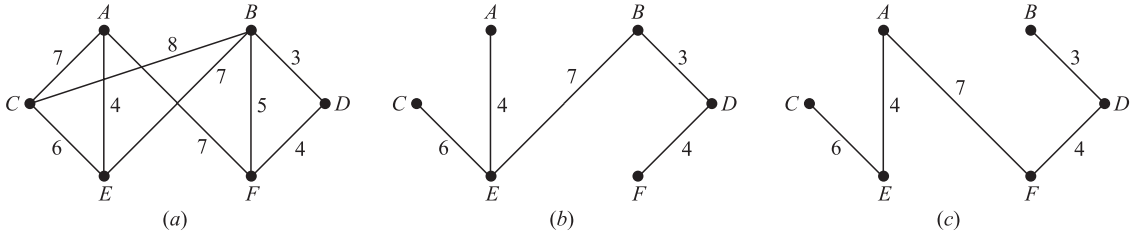



**Fig. 8-20**

(b) Here we apply Algorithm 8.3.

First we order the edges by increasing weights, and then we successively add edges without forming any cycles until five edges are included. This yields the following data:

| Edges | $BD$ | $AE$ | $DF$ | $BF$ | $CE$ | $AC$ | $AF$ | $BE$ | $BC$ |
|---|---|---|---|---|---|---|---|---|---|
| Weight | 3 | 4 | 4 | 5 | 6 | 7 | 7 | 7 | 8 |
| Add? | Yes | Yes | Yes | No | Yes | No | Yes | | |

Thus the minimal spanning tree of $Q$ which is obtained contains the edges

$$BD, \quad AE, \quad DF, \quad CE, \quad AF$$

The spanning tree appears in Fig. 8-20($c$). Observe that this spanning tree is not the same as the one obtained using Algorithm 8.2 as expected it also has weight 24.

**Remark:** The above algorithms are easily executed when the graph $G$ is relatively small as in Fig. 8-20($a$). Suppose $G$ has dozens of vertices and hundreds of edges which, say, are given by a list of pairs of vertices. Then even deciding whether $G$ is connected is not obvious; it may require some type of depth-first search (DFS) or breadth-first search (BFS) graph algorithm. Later sections and the next chapter will discuss ways of representing graphs $G$ in memory and will discuss various graph algorithms.

## 8.9 PLANAR GRAPHS

A graph or multigraph which can be drawn in the plane so that its edges do not cross is said to be *planar*. Although the complete graph with four vertices $K_4$ is usually pictured with crossing edges as in Fig. 8-21($a$), it can also be drawn with noncrossing edges as in Fig. 8-21($b$); hence $K_4$ is planar. Tree graphs form an important class of planar graphs. This section introduces our reader to these important graphs.

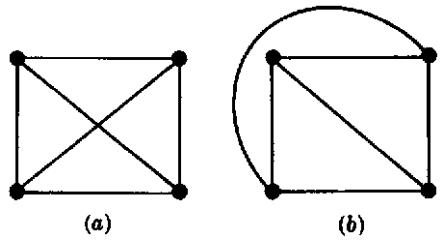



**Fig. 8-21**

### Maps, Regions

A particular planar representation of a finite planar multigraph is called a *map*. We say that the map is *connected* if the underlying multigraph is connected. A given map divides the plane into various regions. For example, the map in Fig. 8-22 with six vertices and nine edges divides the plane into five regions. Observe that four of the regions are bounded, but the fifth region, outside the diagram, is unbounded. Thus there is no loss in generality in counting the number of regions if we assume that our map is contained in some large rectangle rather than in the entire plane.

Observe that the border of each region of a map consists of edges. Sometimes the edges will form a cycle, but sometimes not. For example, in Fig. 8-22 the borders of all the regions are cycles except for $r_3$. However, if we do move counterclockwise around $r_3$ starting, say, at the vertex $C$, then we obtain the closed path

$$(C, D, E, F, E, C)$$

where the edge $\{E, F\}$ occurs twice. By the *degree* of a region $r$, written $\deg(r)$, we mean the length of the cycle or closed walk which borders $r$. We note that each edge either borders two regions or is contained in a region and will occur twice in any walk along the border of the region. Thus we have a theorem for regions which is analogous to Theorem 8.1 for vertices.

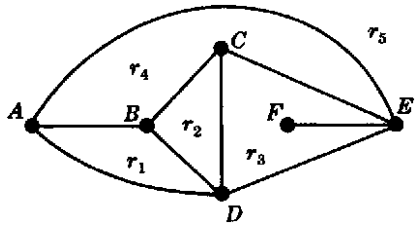



**Fig. 8-22**

**Theorem 8.7:** The sum of the degrees of the regions of a map is equal to twice the number of edges.

The degrees of the regions of Fig. 8-22 are:

$$\deg(r_1) = 3, \quad \deg(r_2) = 3, \quad \deg(r_3) = 5, \quad \deg(r_4) = 4, \quad \deg(r_5) = 3$$

The sum of the degrees is 18, which, as expected, is twice the number of edges.

For notational convenience we shall picture the vertices of a map with dots or small circles, or we shall assume that any intersections of lines or curves in the plane are vertices.

### Euler's Formula

Euler gave a formula which connects the number $V$ of vertices, the number $E$ of edges, and the number $R$ of regions of any connected map. Specifically:

**Theorem 8.8 (Euler):** $V - E + R = 2$.

(The proof of Theorem 8.8 appears in Problem 8.18.)

Observe that, in Fig. 8-22, $V = 6$, $E = 9$, and $R = 5$; and, as expected by Euler's formula.

$$V - E + R = 6 - 9 + 5 = 2$$

We emphasize that the underlying graph of a map must be connected in order for Euler's formula to hold.

Let $G$ be a connected planar multigraph with three or more vertices, so $G$ is neither $K_1$ nor $K_2$. Let $M$ be a planar representation of $G$. It is not difficult to see that (1) a region of $M$ can have degree 1 only if its border is a loop, and (2) a region of $M$ can have degree 2 only if its border consists of two multiple edges. Accordingly, if $G$ is a graph, not a multigraph, then every region of $M$ must have degree 3 or more. This comment together with Euler's formula is used to prove the following result on planar graphs.

**Theorem 8.9:** Let $G$ be a connected planar graph with $p$ vertices and $q$ edges, where $p \geq 3$. Then $q \geq 3p - 6$.

Note that the theorem is not true for $K_1$ where $p = 1$ and $q = 0$, and is not true for $K_2$ where $p = 2$ and $q - 1$.

*Proof*: Let $r$ be the number of regions in a planar representation of $G$. By Euler's formula, $p - q + r = 2$.

Now the sum of the degrees of the regions equals $2q$ by Theorem 8.7. But each region has degree 3 or more; hence $2q \geq 3r$. Thus $r \geq 2q/3$. Substituting this in Euler's formula gives

$$2 = p - q + r \leq p - q + \frac{2q}{3} \quad \text{or} \quad 2 \leq p - \frac{q}{3}$$

Multiplying the inequality by 3 gives $6 \leq 3p - q$ which gives us our result. □

### Nonplanar Graphs, Kuratowski's Theorem

We give two examples of nonplanar graphs. Consider first the *utility graph*; that is, three houses $A_1$, $A_2$, $A_3$ are to be connected to outlets for water, gas and electricity, $B_1$, $B_2$, $B_3$, as in Fig. 8-23($a$). Observe that this is the graph $K_{3,3}$ and it has $p = 6$ vertices and $q = 9$ edges. Suppose the graph is planar. By Euler's formula a planar representation has $r = 5$ regions. Observe that no three vertices are connected to each other; hence the degree of each region must be 4 or more and so the sum of the degrees of the regions must be 20 or more. By Theorem 8.7 the graph must have 10 or more edges. This contradicts the fact that the graph has $q = 9$ edges. Thus the utility graph $K_{3,3}$ is nonplanar.

Consider next the *star graph* in Fig. 8-23($b$). This is the complete graph $K_5$ on $p = 5$ vertices and has $q = 10$ edges. If the graph is planar, then by Theorem 8.9.

$$10 = q \leq 3p - 6 = 15 - 6 = 9$$

which is impossible. Thus $K_5$ is nonplanar.

For many years mathematicians tried to characterize planar and nonplanar graphs. This problem was finally solved in 1930 by the Polish mathematician K. Kuratowski. The proof of this result, stated below, lies beyond the scope of this text.

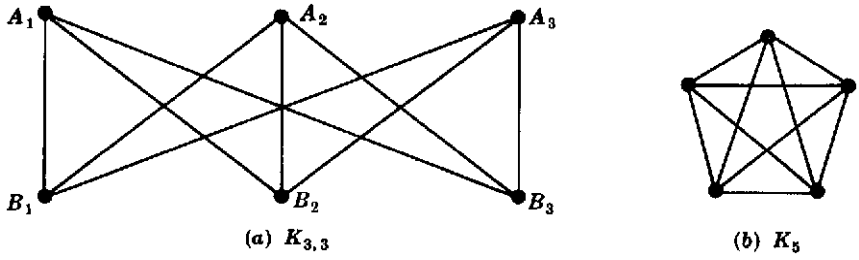



**Fig. 8-23**

**Theorem 8.10:** **(Kuratowski)** A graph is nonplanar if and only if it contains a subgraph homeomorphic to $K_{3,3}$ or $K_5$.

## 8.10 GRAPH COLORINGS

Consider a graph $G$. *A vertex coloring*, or simply a *coloring* of $G$ is an assignment of colors to the vertices of $G$ such that adjacent vertices have different colors. We say that $G$ is $n$-colorable if there exists a coloring of $G$ which uses $n$ colors. (Since the word "color" is used as a noun, we will try to avoid its use as a verb by saying,

for example, "paint" $G$ rather than "color" $G$ when we are assigning colors to the vertices of $G$.) The minimum number of colors needed to paint $G$ is called the *chromatic number* of $G$ and is denoted by $\chi(G)$.

Fig. 8-24 gives an algorithm by Welch and Powell for a coloring of a graph G. We emphasize that this algorithm does not always yield a minimal coloring of $G$.

**Algorithm 8.4 (Welch-Powell):** The input is a graph $G$.

***Step 1.*** Order the vertices of $G$ according to decreasing degrees.

***Step 2.*** Assign the first color $C_1$ to the first vertex and then, in sequential order, assign $C_1$ to each vertex which is not adjacent to a previous vertex which was assigned $C_1$.

***Step 3.*** Repeat Step 2 with a second color $C_2$ and the subsequence of noncolored vertices.

***Step 4.*** Repeat Step 3 with a third color $C_3$, then a fourth color $C_4$, and so on until all vertices are colored.

***Step 5.*** Exit.

**Fig. 8-24**

## EXAMPLE 8.3

(a) Consider the graph $G$ in Fig. 8-25. We use the Welch-Powell Algorithm 8.4 to obtain a coloring of $G$. Ordering the vertices according to decreasing degrees yields the following sequence:

$$A_5, \quad A_3, \quad A_7, \quad A_1, \quad A_2, \quad A_4, \quad A_6, \quad A_8$$

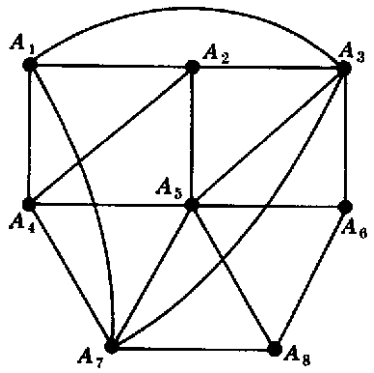



**Fig. 8-25**

The first color is assigned to vertices $A_5$ and $A_1$. The second color is assigned to vertices $A_3$, $A_4$, and $A_8$. The third color is assigned to vertices $A_7$, $A_2$, and $A_6$. All the vertices have been assigned a color, and so $G$ is 3-colorable. Observe that $G$ is not 2-colorable since vertices $A_1$, $A_2$, and $A_3$, which are connected to each other, must be assigned different colors. Accordingly, $\chi(G) = 3$.

(b) Consider the complete graph $K_n$ with $n$ vertices. Since every vertex is adjacent to every other vertex, $K_n$ requires $n$ colors in any coloring. Thus $\chi(K_n) = n$.

There is no simple way to actually determine whether an arbitrary graph is $n$-colorable. However, the following theorem (proved in Problem 8.19) gives a simple characterization of 2-colorable graphs.

**Theorem 8.11:** The following are equivalent for a graph $G$:

(i) $G$ is 2-colorable.
(ii) $G$ is bipartite.
(iii) Every cycle of $G$ has even length.

There is no limit on the number of colors that may be required for a coloring of an arbitrary graph since, for example, the complete graph $K_n$ requires $n$ colors. However, if we restrict ourselves to planar graphs, regardless of the number of vertices, five colors suffice. Specifically, in Problem 8.20 we prove:

**Theorem 8.12:** Any planar graph is 5-colorable.

Actually, since the 1850s mathematicians have conjectured that planar graphs are 4-colorable since every known planar graph is 4-colorable. Kenneth Appel and Wolfgang Haken finally proved this conjecture to be true in 1976. That is:

**Four Color Theorem (Appel and Haken):** Any planar graph is 4-colorable.

We discuss this theorem in the next subsection.

### Dual Maps and the Four Color Theorem

Consider a map $M$, say the map $M$ in Fig. 8-26($a$). In other words, $M$ is a planar representation of a planar multigraph. Two regions of $M$ are said to be *adjacent* if they have an edge in common. Thus the regions $r_2$ and $r_5$ in Fig. 8-26($a$) are adjacent, but the regions $r_3$ and $r_5$ are not. By a *coloring* of $M$ we mean an assignment of a color to each region of $M$ such that adjacent regions have different colors. A map $M$ is *n-colorable* if there exists a coloring of $M$ which uses $n$ colors. Thus the map $M$ in Fig. 8-26($a$) is 3-colorable since the regions can be assigned the following colors:

$$r_1 \text{ red}, \quad r_2 \text{ white}, \quad r_3 \text{ red}, \quad r_4 \text{ white}, \quad r_5 \text{ red}, \quad r_6 \text{ blue}$$

Observe the similarity between this discussion on coloring maps and the previous discussion on coloring graphs. In fact, using the concept of the dual map defined below, the coloring of a map can be shown to be equivalent to the vertex coloring of a planar graph.

Consider a map $M$. In each region of $M$ we choose a point, and if two regions have an edge in common then we connect the corresponding points with a curve through the common edge. These curves can be drawn so that they are noncrossing. Thus we obtain a new map $M^*$, called the *dual* of $M$, such that each vertex of $M^*$ corresponds to exactly one region of $M$. Figure 8-26($b$) shows the dual of the map of Fig. 8-26($a$). One can prove that each region of $M^*$ will contain exactly one vertex of $M$ and that each edge of $M^*$ will intersect exactly one edge of $M$ and vice versa. Thus $M$ will be the dual of the map $M^*$.

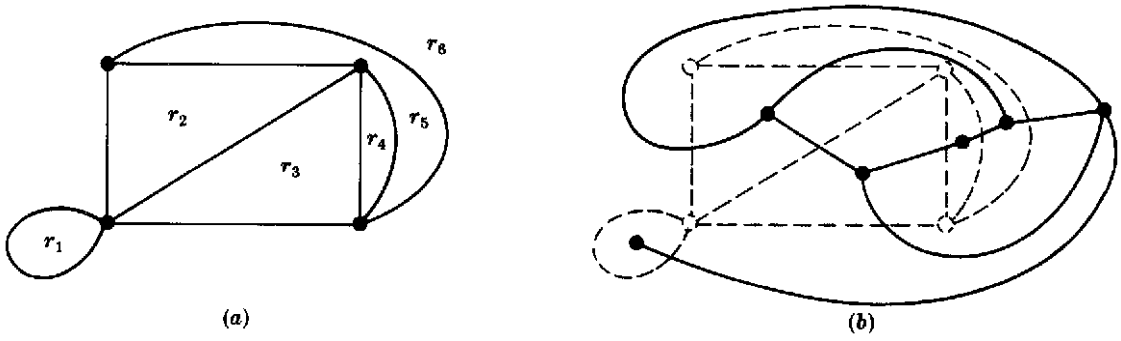



**Fig. 8-26**

Observe that any coloring of the regions of a map $M$ will correspond to a coloring of the vertices of the dual map $M^*$. Thus $M$ is $n$-colorable if and only if the planar graph of the dual map $M^*$ is vertex $n$-colorable. Thus the above theorem can be restated as follows:

**Four Color Theorem (Appel and Haken):** If the regions of any map $M$ are colored so that adjacent regions have different colors, then no more than four colors are required.

The proof of the above theorem uses computers in an essential way. Specifically, Appel and Haken first showed that if the four color theorem was false, then there must be a counterexample among one of approximately 2000 different types of planar graphs. They then showed, using the computer, that none of these types of graphs has such a counterexample. The examination of each different type of graph seems to be beyond the grasp of human beings without the use of a computer. Thus the proof, unlike most proofs in mathematics, is technology dependent; that is, it depended on the development of high-speed computers.

## 8.11 REPRESENTING GRAPHS IN COMPUTER MEMORY

There are two standard ways of maintaining a graph $G$ in the memory of a computer. One way, called the *sequential representation* of $G$, is by means of its adjacency matrix $A$. The other way, called the *linked representation* or *adjacency structure* of $G$, uses linked lists of neighbors. Matrices are usually used when the graph $G$ is dense, and linked lists are usually used when $G$ is sparse. (A graph $G$ with $m$ vertices and $n$ edges is said to be *dense* when $m = O(n^2)$ and *sparse* when $m = O(n)$ or even $O(n \log n)$.)

Regardless of the way one maintains a graph $G$ in memory, the graph $G$ is normally input into the computer by its formal definition, that is, as a collection of vertices and a collection of pairs of vertices (edges).

### Adjacency Matrix

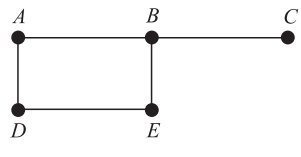
Suppose $G$ is a graph with $m$ vertices, and suppose the vertices have been ordered, say, $v_1, v_2, \ldots, v_m$. Then the *adjacency matrix* $A = [a_{ij}]$ of the graph $G$ is the $m \times m$ matrix defined by

$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j \\ 0 & \text{otherwise} \end{cases}$$

Figure 8-27($b$) contains the adjacency matrix of the graph $G$ in Fig. 8-27($a$) where the vertices are ordered $A$, $B$, $C$, $D$, $E$. Observe that each edge $\{v_i, v_j\}$ of $G$ is represented twice, by $a_{ij} = 1$ and $a_{ji} = 1$. Thus, in particular, the adjacency matrix is symmetric.

The adjacency matrix $A$ of a graph $G$ does depend on the ordering of the vertices of $G$, that is, a different ordering of the vertices yields a different adjacency matrix. However, any two such adjacency matrices are closely related in that one can be obtained from the other by simply interchanging rows and columns. On the other hand, the adjacency matrix does not depend on the order in which the edges (pairs of vertics) are input into the computer.

There are variations of the above representation. If $G$ is a multigraph, then we usually let $a_{ij}$ denote the number of edges $\{v_i, v_j\}$. Moreover, if $G$ is a weighted graph, then we may let $a_{ij}$ denote the weight of the edge $\{v_i, v_j\}$.




($a$)

| | $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|
| $A$ | 0 | 1 | 0 | 1 | 0 |
| $B$ | 1 | 0 | 1 | 0 | 1 |
| $C$ | 0 | 1 | 0 | 0 | 0 |
| $D$ | 1 | 0 | 0 | 0 | 1 |
| $E$ | 0 | 1 | 0 | 1 | 0 |

($b$)

**Fig. 8-27**

### Linked Representation of a Graph $G$

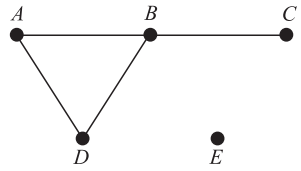
Let $G$ be a graph with $m$ vertices. The representation of $G$ in memory by its adjacency matrix $A$ has a number of major drawbacks. First of all it may be difficult to insert or delete vertices in $G$. The reason is that the size of $A$ may need to be changed and the vertices may need to be reordered, so there may be many, many changes in the matrix $A$. Furthermore, suppose the number of edges is $O(m)$ or even $O(m \log m)$, that is, suppose $G$ is sparse. Then the matrix $A$ will contain many zeros; hence a great deal of memory space will be wasted. Accordingly, when $G$ is sparse, $G$ is usually represented in memory by some type of *linked representation*, also called an *adjacency structure*, which is described below by means of an example.

Consider the graph $G$ in Fig. 8-28($a$). Observe that $G$ may be equivalently defined by the table in Fig. 8-28($b$) which shows each vertex in $G$ followed by its *adjacency list*, i.e., its list of adjacent vertices (*neighbors*). Here the symbol $\varnothing$ denotes an empty list. This table may also be presented in the compact form

$$G = [A{:}B, D; \quad B{:}A, C, D; \quad C{:}B; \quad D{:}A, B; \quad E{:}\varnothing]$$

where a colon ":" separates a vertex from its list of neighbors, and a semicolon ";" separates the different lists.

**Remark:** Observe that each edge of a graph $G$ is represented twice in an adjacency structure; that is, any edge, say $\{A, B\}$, is represented by $B$ in the adjacency list of $A$, and also by $A$ in the adjacency list of $B$. The graph $G$ in Fig. 8-28($a$) has four edges, and so there must be 8 vertices in the adjacency lists. On the other hand, each vertex in an adjacency list corresponds to a unique edge in the graph $G$.




($a$)

| Vertex | Adjacency list |
|---|---|
| $A$ | $B, D$ |
| $B$ | $A, C, D$ |
| $C$ | $B$ |
| $D$ | $A, B$ |
| $E$ | $\varnothing$ |

($b$)

**Fig. 8-28**

The *linked representation* of a graph $G$, which maintains $G$ in memory by using its adjacency lists, will normally contain two files (or sets of records), one called the Vertex File and the other called the Edge File, as follows.

(a) ***Vertex File*:** The Vertex File will contain the list of vertices of the graph $G$ usually maintained by an array or by a linked list. Each record of the Vertex File will have the form


Here VERTEX will be the name of the vertex, NEXT-V points to the next vertex in the list of vertices in the Vertex File when the vertices are maintained by a linked list, and **PTR** will point to the first element in the adjacency list of the vertex appearing in the Edge File. The shaded area indicates that there may be other information in the record corresponding to the vertex.

(b) ***Edge File*:** The Edge File contains the edges of the graph $G$. Specifically, the Edge File will contain all the adjacency lists of $G$ where each list is maintained in memory by a linked list. Each record of the Edge File will correspond to a vertex in an adjacency list and hence, indirectly, to an edge of $G$. The record will usually have the form

Here:

(1) EDGE will be the name of the edge (if it has one).

(2) ADJ points to the location of the vertex in the Vertex File.

(3) NEXT points to the location of the next vertex in the adjacency list.

We emphasize that each edge is represented twice in the Edge File, but each record of the file corresponds to a unique edge. The shaded area indicates that there may be other information in the record corresponding to the edge.

Figure 8-29 shows how the graph $G$ in Fig. 8-28($a$) may appear in memory. Here the vertices of $G$ are maintained in memory by a linked list using the variable START to point to the first vertex. (Alternatively, one could use a linear array for the list of vertices, and then NEXT-V would not be required.) Note that the field EDGE is not needed here since the edges have no name. Figure 8-29 also shows, with the arrows, the adjacency list [$D$, $C$, $A$] of the vertex $B$.

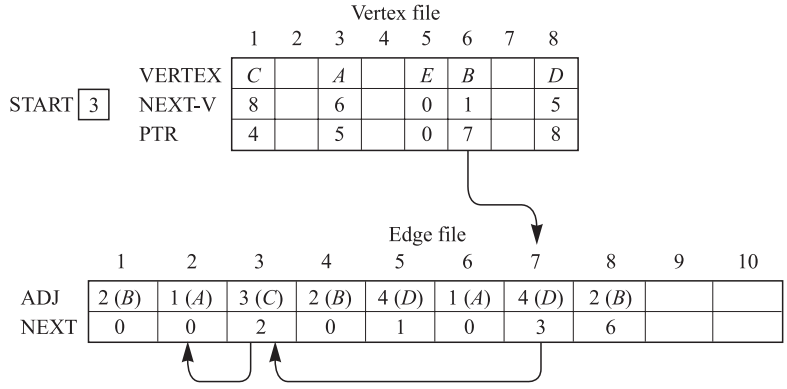
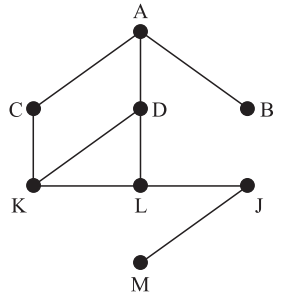



**Fig. 8-29**

## 8.12 GRAPH ALGORITHMS

This section discusses two important graph algorithms which systematically examine the vertices and edges of a graph $G$. One is called a *depth-first search* (DFS) and the other is called a *breadth-first search* (BFS). Other graph algorithms will be discussed in the next chapter in connection with directed graphs. Any particular graph algorithm may depend on the way $G$ is maintained in memory. Here we assume $G$ is maintained in memory by its adjacency structure. Our test graph $G$ with its adjacency structure appears in Fig. 8-30 where we assume the vertices are ordered alphabetically.




(*a*)

| Vertex | Adjacency list |
|---|---|
| A | B, C, D |
| B | A |
| C | A, K |
| D | A, K, L |
| J | L, M |
| K | C, D, L |
| L | D, J, K |
| M | J |

(*b*)

**Fig. 8-30**

During the execution of our algorithms, each vertex (node) $N$ of $G$ will be in one of three states, called the *status* of $N$, as follows:

STATUS = 1: (Ready state) The initial state of the vertex $N$.

STATUS = 2: (Waiting state) The vertex $N$ is on a (waiting) list, waiting to be processed.

STATUS = 3: (Processed state) The vertex $N$ has been processed.

The waiting list for the depth-first seach (DFS) will be a (modified) STACK (which we write horizontally with the top of STACK on the left), whereas the waiting list for the breadth-first search (BFS) will be a QUEUE.

### Depth-first Search

The general idea behind a depth-first search beginning at a starting vertex $A$ is as follows. First we process the starting vertex $A$. Then we process each vertex $N$ along a path $P$ which begins at $A$; that is, we process a neighbor of $A$, then a neighbor of $A$, and so on. After coming to a "dead end," that is to a vertex with no unprocessed neighbor, we backtrack on the path $P$ until we can continue along another path $P'$. And so on. The backtracking is accomplished by using a STACK to hold the initial vertices of future possible paths. We also need a field STATUS which tells us the current status of any vertex so that no vertex is processed more than once.

The depth-first search (DFS) algorithm appears in Fig. 8-31. The algorithm will process only those vertices which are connected to the starting vertex $A$, that is, the connected component including $A$. Suppose one wants to process all the vertices in the graph $G$. Then the algorithm must be modified so that it begins again with another vertex (which we call $B$) that is still in the ready state (STATE = 1). This vertex $B$ can be obtained by traversing through the list of vertices.

**Remark:** The structure STACK in the above algorithm is not technically a stack since, in Step 5($b$), we allow a vertex $J$ to be deleted and then inserted in the front of the stack. (Although it is the same vertex $J$, it usually represents a different edge in the adjacency structure.) If we do not delete $J$ in Step 5($b$), then we obtain an alternate form of DFS.

**Algorithm 8.5 (Depth-first Search):** This algorithm executes a depth-first search on a graph $G$ beginning with a starting vertex $A$.

***Step 1.*** Initialize all vertices to the ready state (STATUS = 1).

***Step 2.*** Push the starting vertex $A$ onto STACK and change the status of $A$ to the waiting state (STATUS = 2).

***Step 3.*** Repeat Steps 4 and 5 until STACK is empty.

***Step 4.*** Pop the top vertex $N$ of STACK. Process $N$, and set STATUS ($N$) = 3, the processed state.

***Step 5.*** Examine each neighbor $J$ of $N$.

($a$) If STATUS ($J$) = 1 (ready state), push $J$ onto STACK and reset STATUS ($J$) = 2 (waiting state).

($b$) If STATUS ($J$) = 2 (waiting state), delete the previous $J$ from the STACK and push the current $J$ onto STACK.

($c$) If STATUS ($J$) = 3 (processed state), ignore the vertex $J$.

[End of Step 3 loop.]

***Step 6.*** Exit.

**Fig. 8-31**

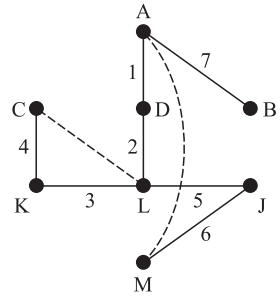
**EXAMPLE 8.4** Suppose the DFS Algorithm 8.5 in Fig. 8-31 is applied to the graph in Fig. 8-30. The vertices will be processed in the following order:

$$A, \quad D, \quad L, \quad K, \quad C, \quad J, \quad M, \quad B$$

Specifically, Fig. 8-32($a$) shows the sequence of vertices being processed and the sequence of waiting lists in STACK. (Note that after vertex $A$ is processed, its neighbors, $B$, $C$, and $D$ are added to STACK in the order first $B$, then $C$, and finally $D$; hence $D$ is on the top of the STACK and $D$ is the next vertex to be processed.) Each vertex, excluding $A$, comes from an adjacency list and hence corresponds to an edge of the graph. These edges form a spanning tree of $G$ which is pictured in Fig. 8-32($b$). The numbers indicate the order that the edges are added to the spanning tree, and the dashed lines indicate backtracking.

| STACK | Vertex |
|---|---|
| A | A |
| D, C, B | D |
| L, K, C, B | L |
| K, J, ~~K~~, C, B | K |
| C, J, ~~C~~, B | C |
| J, B | J |
| M, B | M |
| B | B |
| ∅ | |

($a$)




($b$)

**Fig. 8-32**

### Breadth-first Search

The general idea behind a breadth-first search beginning at a starting vertex $A$ is as follows. First we process the starting vertex $A$. Then we process all the neighbors of $A$. Then we process all the neighbors of neighbors of $A$. And so on. Naturally we need to keep track of the neighbors of a vertex, and we need to guarantee that no vertex is processed twice. This is accomplished by using a QUEUE to hold vertices that are waiting to be processed, and by a field STATUS which tells us the current status of a vertex.

The breadth-first search (BFS) algorithm appears in Fig. 8-33, Again the algorithm will process only those vertices which are connected to the starting vertex $A$, that is, the connected component including $A$. Suppose one wants to process all the vertices in the graph $G$. Then the algorithm must be modified so that it begins again with another vertex (which we call $B$) that is still in the ready state (STATUS $= 1$). This vertex $B$ can be obtained by traversing through the list of vertices.

**EXAMPLE 8.5** Suppose the breadth-first search (BFS) Algorithm 8.6 in Fig. 8-33 is applied to the graph in Fig. 8-30. The vertices will be processed in the following order:

$$A, \quad B, \quad C, \quad D, \quad K, \quad L, \quad J, \quad M$$

Specifically, Fig. 8-34($a$) shows the sequence of waiting lists in QUEUE and the sequence of vertices being processed (Note that after vertex $A$ is processed, its neighbors, $B$, $C$, and $D$ are added to QUEUE in the order first $B$, then $C$, and finally $D$; hence $B$ is on the front of the QUEUE and so B is the next vertex to be processed.) Again, each vertex, excluding $A$, comes from an adjacency list and hence corresponds to an edge of the graph. These edges form a spanning tree of $G$ which is pictured in Fig. 8-34($b$). Again, the numbers indicate the order that the edges are added to the spanning tree. Observe that this spanning tree is different from the one in Fig. 8.32($b$) which came from a depth-first search.

**Algorithm 8.6 (Breadth-first Search):** This algorithm executes a breadth-first search on a graph $G$ beginning with a starting vertex $A$.

***Step 1.*** Initialize all vertices to the ready state (STATUS = 1).

***Step 2.*** Put the starting vertex $A$ in QUEUE and change the status of $A$ to the waiting state (STATUS = 2).

***Step 3.*** Repeat Steps 4 and 5 until QUEUE is empty.

***Step 4.*** Remove the front vertex $N$ of QUEUE. Process $N$, and set STATUS $(N) = 3$, the processed state.

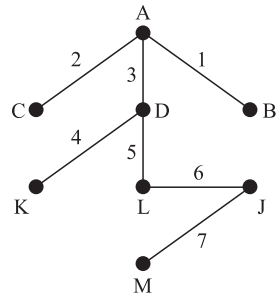
***Step 5.*** Examine each neighbor $J$ of $N$.
($a$) If STATUS $(J) = 1$ (ready state), add $J$ to the rear of QUEUE and reset STATUS $(J) = 2$ (waiting state).
($b$) If STATUS $(J) = 2$ (waiting state) or STATUS $(J) = 3$ (processed state), ignore the vertex $J$.
[End of Step 3 loop.]

***Step 6.*** Exit.

**Fig. 8-33**

| QUEUE | Vertex |
|---|---|
| A | A |
| D, C, B | B |
| D, C | C |
| D | D |
| L, K | K |
| L | L |
| J | J |
| M | M |
| ∅ | |

($a$)




($b$)

**Fig. 8-34**

## 8.13 TRAVELING-SALESMAN PROBLEM

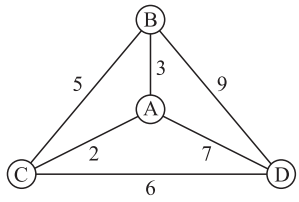
Let $G$ be a complete weighted graph. (We view the vertices of $G$ as cities, and the weighted edges of $G$ as the distances between the cities.) The "traveling-salesman" problem refers to finding a Hamiltonian circuit for $G$ of minimum weight.

First we note the following theorem, proved in Problem 8.33:

**Theorem 8.13:** The complete graph $K_n$ with $n \geq 3$ vertices has $H = (n-1)!/2$ Hamiltonian circuits (where we do not distinguish between a circuit and its reverse).

Consider the complete weighted graph $G$ in Fig. 8-35($a$). It has four vertices, $A$, $B$, $C$, $D$. By Theorem 8.13 it has $H = 3!/2 = 3$ Hamiltonian circuits. Assuming the circuits begin at the vertex $A$, the following are the three circuits and their weights:

$$\begin{aligned} |ABCDA| &= 3+5+6+7 = 21 \\ |ACDBA| &= 2+6+9+3 = 20 \\ |ACBDA| &= 2+5+9+7 = 23 \end{aligned}$$




(*a*)

| | P | Q | R | S | T |
|---|---|---|---|---|---|
| P | | 18 | 22 | 15 | 20 |
| Q | 18 | | 11 | 12 | 22 |
| R | 22 | 11 | | 16 | 10 |
| S | 15 | 12 | 16 | | 13 |
| T | 20 | 22 | 10 | 13 | |

(*b*)

**Fig. 8-35**

Thus *ACDBA* with weight 20 is the Hamiltonian circuit of minimum weight.

We solved the "traveling-salesman problem" for the weighted complete graph in Fig. 8-35(*a*) by listing and finding the weights of its three possible Hamiltonian circuits. However, for a graph with many vertices, this may be impractical or even impossible. For example, a complete graph with 15 vertices has over 40 million Hamiltonian circuits. Accordingly, for circuits with many vertices, a strategy of some kind is needed to solve or give an approximate solution to the traveling-salesman problem. We discuss one of the simplest algorithms here.

### Nearest-Neighbor Algorithm

The nearest-neighbor algorithm, starting at a given vertex, chooses the edge with the least weight to the next possible vertex, that is, to the "closest" vertex. This strategy is continued at each successive vertex until a Hamiltonian circuit is completed.

**EXAMPLE 8.6** Let $G$ be the weighted graph given by the table in Fig. 8-35(*b*). That is, $G$ has the vertices $P, Q, \ldots, T$, and the distance from $P$ to $Q$ is 18, from $P$ to $R$ is 22, and so on until the distance from $T$ to $S$ is 13. We apply the nearest-neighbor algorithm to $G$ starting at: (a) $P$, (b) $Q$.

(a) Starting at $P$, the first row of the table shows us that the closest vertex to $P$ is $S$ with distance 15. The fourth row shows that the closest vertex to $S$ is $Q$ with distance 12. The closest vertex to $Q$ is $R$ with distance 11. From $R$, there is no choice but to go to $T$ with distance 10. Finally, from $T$, there is no choice but to go back to $P$ with distance 20. Accordingly, the nearest-neighbor algorithm beginning at $P$ yields the following weighted Hamiltonian circuit:

$$|PSQRTP| = 15 + 12 + 11 + 10 + 20 = 68$$

(b) Starting at $Q$, the closest vertex is $R$ with distance 11; from $R$ the closest is T with distance 10; and from $T$ the closest is $S$ with distance 13. From $S$ we must go to $P$ with distance 15; and finally from $P$ we must go back to $Q$ with distance 18. Accordingly, the nearest-neighbor algorithm beginning at $Q$ yields the following weighted Hamiltonian circuit:

$$|QRTSPQ| = 11 + 10 + 13 + 15 + 18 = 67$$

The idea behind the nearest-neighbor algorithm is to minimize the total weight by minimizing the weight at each step. Although this may seem reasonable, Example 8.6 shows that we may not get a Hamiltonian circuit of minimum weight; that is, it cannot be both 68 and 67. Only by checking all $H = (n - 1)!/2 = 12$ Hamiltonian circuits of $G$ will we really know the one with minimum weight. In fact, the nearest-neighbor algorithm beginning at $A$ in Fig. 8-35(*a*) yields the circuit *ACBDA* which has the maximum weight. However, the nearest-neighbor algorithm usually gives a Hamiltonian circuit which is relatively close to the one with minimum weight.

# Solved Problems

## GRAPH TERMINOLOGY

**8.1.** Consider the graph $G$ in Fig. 8-36($a$).

($a$) Describe $G$ formally, that is, find the set $V(G)$ of vertices of $G$ and the set $E(G)$ of edges of $G$.

($b$) Find the degree of each vertex and verify Theorem 8.1 for this graph.

($a$) There are five vertices so $V(G) = \{A, B, C, D, E\}$. There are seven pairs $\{x, y\}$ of vertices where the vertex $x$ is connected with the vertex $y$, hence

$$E(G) = [\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, E\}, \{C, D\}, \{C, E\}]$$

($b$) The degree of a vertex is equal to the number of edges to which it belongs; e.g., $\deg(A) = 3$ since $A$ belongs to the three edges $\{A, B\}$, $\{A, C\}$, $\{A, D\}$. Similarly,

$$\deg(B) = 3,\ \deg(C) = 4,\ \deg(D) = 2,\ \deg(E) = 2$$

The sum of the degrees is $3 + 3 + 4 + 2 + 2 = 14$ which does equal twice the number of edges.

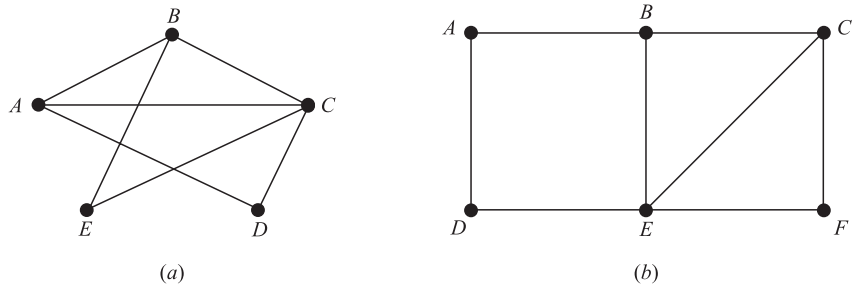



**Fig. 8-36**

**8.2.** Consider the graph $G$ in Fig. 8-36($b$). Find:

($a$) all simple paths from $A$ to $F$;
($b$) all trails from $A$ to $F$;
($c$) $d(A, F)$, the distance from $A$ to $F$;
($d$) $\text{diam}(G)$, the diameter of $G$;
($e$) all cycles which include vertex $A$;
($f$) all cycles in $G$.

($a$) A simple path from $A$ to $F$ is a path such that no vertex, and hence no edge, is repeated. There are seven such paths, four beginning with the edges $\{A, B\}$ and three beginning with the ege $\{A, D\}$:

$$(A, B, C, F), \quad (A, B, C, E, F), \quad (A, B, E, F), \quad (A, B, E, C, F),$$
$$(A, D, E, F), \quad (A, D, E, B, C, F), \quad (A, D, E, C, F).$$

($b$) A trail from $A$ to $F$ is a path such that no edge is repeated. There are nine such trails, the seven simple paths from ($a$) together with

$$(A, D, E, B, C, E, F) \quad \text{and} \quad (A, D, E, C, B, E, F).$$

($c$) There is a path, e.g., $(A, B, C, F)$, from $A$ to $F$ of length 3 and no shorter path from $A$ to $F$; hence $d(A, F) = 3$.

($d$) The distance between any two vertices is not greater than 3, and the distance from $A$ to $F$ is 3; hence $\text{diam}(G) = 3$.

($e$) A cycle is a closed path in which no vertex is repeated (except the first and last). There are three cycles which include vertex $A$:

$$(A, B, E, D, A), \quad (A, B, C, E, D, A), \quad (A, B, C, F, E, D, A).$$

($f$) There are six cycles in $G$; the three in ($e$) and

$$(B, C, E, B), \quad (C, F, E, C), \quad (B, C, F, E, B).$$

**8.3.** Consider the multigraphs in Fig. 8-37.

(*a*) Which of them are connected? If a graph is not connected, find its connected components.

(*b*) Which are cycle-free (without cycles)?

(*c*) Which are loop-free (without loops)?

(*d*) Which are (simple) graphs?

(*a*) Only (1) and (3) are connected, (2) is disconnected; its connected components are $\{A, D, E\}$ and $\{B, C\}$. (4) is disconnected; its connected components are $\{A, B, E\}$ and $\{C, D\}$.

(*b*) Only (1) and (4) are cycle-free. (2) has the cycle $(A, D, E, A)$, and (3) has the cycle $(A, B, E, A)$.

(*c*) Only (4) has a loop which is $\{B, B\}$.

(*d*) Only (1) and (2) are graphs. Multigraph (3) has multiple edges $\{A, E\}$ and $\{A, E\}$; and (4) has both multiple edges $\{C, D\}$ and $\{C, D\}$ and a loop $\{B, B\}$.

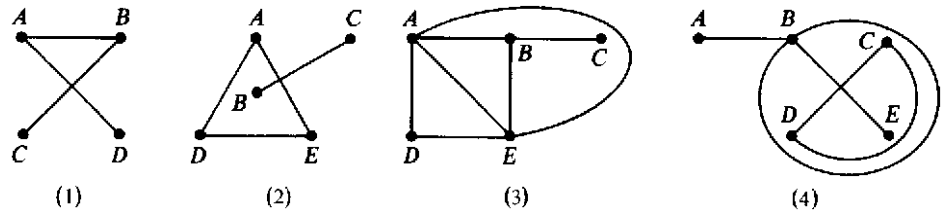



**Fig. 8-37**

**8.4.** Let $G$ be the graph in Fig. 8-38(*a*). Find:

(*a*) all simple paths from $A$ to $C$;
(*b*) all cycles;
(*c*) subgraph $H$ generated by $V' = \{B, C, X, Y\}$;
(*d*) $G - Y$;
(*e*) all cut points;
(*f*) all bridges.

(*a*) There are two simple paths from $A$ to $C$: $(A, X, Y, C)$ and $(A, X, B, Y, C)$.

(*b*) There is only one cycle: $(B, X, Y, B)$.

(*c*) As pictured in Fig. 8-38(*b*), $H$ consists of the vertices $V'$ and the set $E'$ of all edges whose endpoints belong to $V'$, that is, $E' = [\{B, X\}, \{X, Y\}, \{B, Y\}, \{C, Y\}]$.

(*d*) Delete vertex $Y$ from $G$ and all edges which contain $Y$ to obtain the graph $G - Y$ in Fig. 8-38(*c*). (Note $Y$ is a cutpoint since $G - Y$ is disconnected.)

(*e*) Vertices $A$, $X$, and $Y$ are cut points.

(*f*) An edge $e$ is a bridge if $G - e$ is disconnected. Thus there are three bridges: $\{A, Z\}$, $\{A, X\}$, and $\{C, Y\}$.

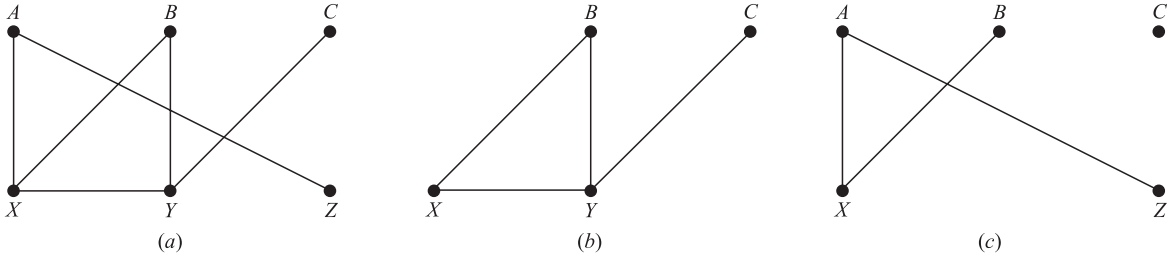



**Fig. 8-38**

**8.5.** Consider the graph $G$ in Fig. 8-36($b$). Find the subgraphs obtained when each vertex is deleted. Does $G$ have any cut points?

When we delete a vertex from $G$, we also have to delete all edges which contain the vertex. The six graphs obtained by deleting each of the vertices of $G$ are shown in Fig. 8-39. All six graphs are connected; hence no vertex is a cut point.

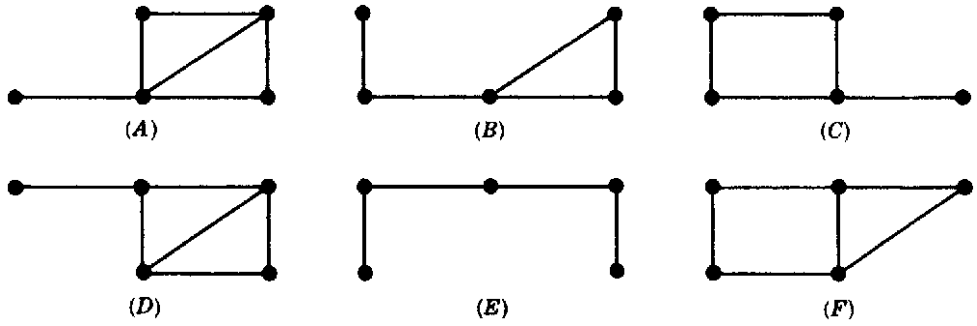



**Fig. 8-39**

**8.6.** Show that the six graphs obtained in Problem 8.5 are distinct, that is, no two of them are isomorphic. Also show that ($B$) and ($C$) are homeomorphic.

The degrees of the five vertices of any graph cannot be paired off with the degrees of any other graph, except for ($B$) and ($C$). Hence none of the graphs is isomorphic except possibly ($B$) and ($C$).

However if we delete the vertex of degree 3 in ($B$) and ($C$), we obtain distinct subgraphs. Thus ($B$) and ($C$) are also nonisomorphic; hence all six graphs are distinct. However, ($B$) and ($C$) are homeomorphic since they can be obtained from isomorphic graphs by adding appropriate vertices.

## TRAVERSABLE GRAPHS, EULER AND HAMILTONIAN CIRCUITS

**8.7.** Consider each graph in Fig. 8-40. Which of them are traversable, that is, have Euler paths?

Which are Eulerian, that is, have an Euler circuit? For those that do not, explain why.

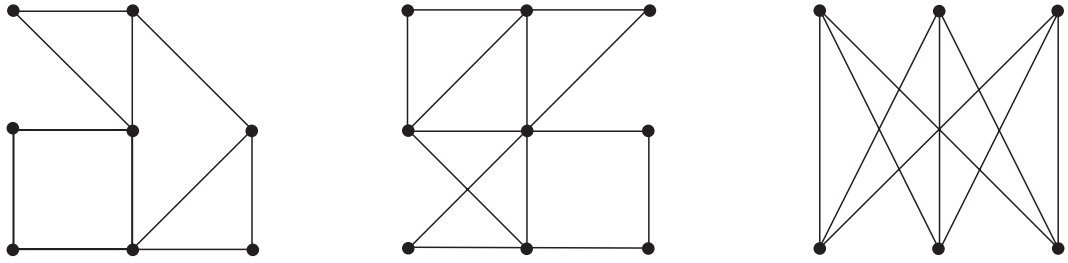



**Fig. 8-40**

$G$ is traversable (has an Euler path) if only 0 or 2 vertices have odd degree, and $G$ is Eulerian (has an Euler circuit) if all vertices are of even degree (Theorem 8.3).

($a$) Traversable, since there are two odd vertices. The traversable path must begin at one of the odd vertices and will end at the other.

($b$) Traversable, since all vertices are even. Thus $G$ has an Euler circuit.

($c$) Since six vertices have odd degrees, $G$ is not traversable.

**8.8.** Which of the graphs in Fig. 8-40 have a Hamiltonian circuit? If not, why not?

Graphs $(a)$ and $(c)$ have Hamiltonian circuits. (The reader should be able to easily find one of them.) However, graph $(b)$ has no Hamiltonian circuit. For if $\alpha$ is a Hamiltonian circuit, then $\alpha$ must connect the middle vertex with the lower right vertex, then proceed along the bottom row to the lower right vertex, then vertically to the middle right, but then is forced back to the central vertex before visiting the remaining vertices.

**8.9.** Prove Theorem 8.3 (Euler): A finite connected graph $G$ is Eulerian if and only if each vertex has even degree.

Suppose $G$ is Eulerian and $T$ is a closed Eulerian trail. For any vertex $v$ of $G$, the trail $T$ enters and leaves $v$ the same number of times without repeating any edge. Hence $v$ has even degree.

Suppose conversely that each vertex of $G$ has even degree. We construct an Eulerian trail. We begin a trail $T_1$ at any edge $e$. We extend $T_1$ by adding one edge after the other. If $T_1$ is not closed at any step, say, $T_1$ begins at $u$ but ends at $v \neq u$, then only an odd number of the edges incident on $v$ appear in $T_1$; hence we can extend $T_1$ by another edge incident on $v$. Thus we can continue to extend $T_1$ until $T_1$ returns to its initial vertex $u$, i.e., until $T_1$ is closed. If $T_1$ includes all the edges of $G$, then $T_1$ is our Eulerian trail.

Suppose $T_1$ does not include all edges of $G$. Consider the graph $H$ obtained by deleting all edges of $T_1$ from $G$. $H$ may not be connected, but each vertex of $H$ has even degree since $T_1$ contains an even number of the edges incident on any vertex. Since $G$ is connected, there is an edge $e'$ of $H$ which has an endpoint $u'$ in $T_1$. We construct a trail $T_2$ in $H$ beginning at $u'$ and using $e'$. Since all vertices in $H$ have even degree, we can continue to extend $T_2$ in $H$ until $T_2$ returns to $u'$ as pictured in Fig. 8-41. We can clearly put $T_1$ and $T_2$ together to form a larger closed trail in $G$. We continue this process until all the edges of $G$ are used. We finally obtain an Eulerian trail, and so $G$ is Eulerian.

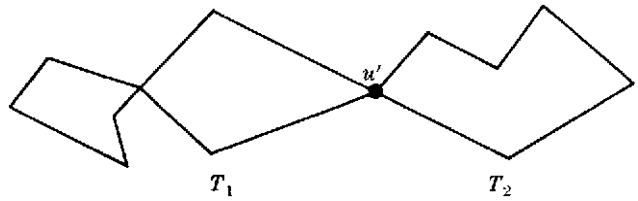



**Fig. 8-41**

## TREES, SPANNING TREES

**8.10.** Draw all trees with exactly six vertices.

There are six such trees which are exhibited in Fig. 8-42. The first tree has diameter 5, the next two diameter 4, the next two diameter 3, and the last one diameter 2. Any other tree with 6 nodes is isormorphic to one of these trees.

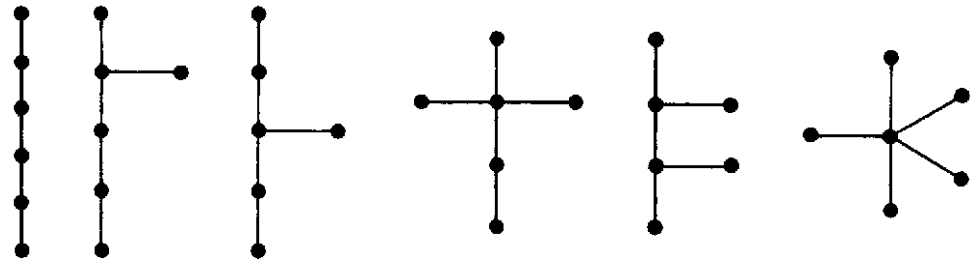


**Fig. 8-42**

**8.11.** Find all spanning trees of the graph $G$ shown in Fig. 8-43$(a)$.

There are eight such spanning trees as shown in Fig. 8-43$(b)$. Each spanning tree must have $4 - 1 = 3$ edges since $G$ has four vertices. Thus each tree can be obtained by deleting two of the five edges of $G$. This can be done in 10 ways,

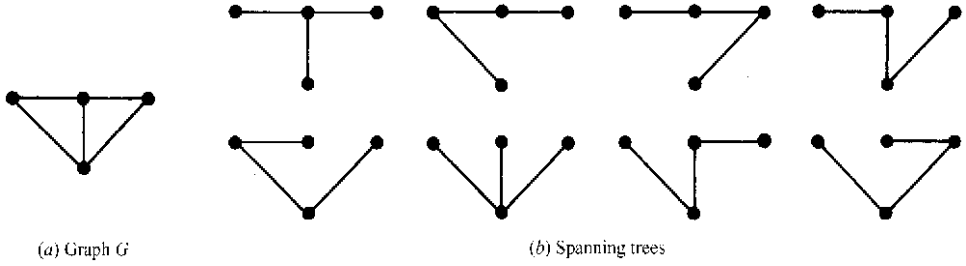



**Fig. 8-43**

except that two of the ways lead to disconnected graphs. Hence the above eight spanning trees are all the spanning trees of $G$.

**8.12.** Find a minimal spanning tree $T$ for the weighted graph $G$ in Fig. 8-44($a$).

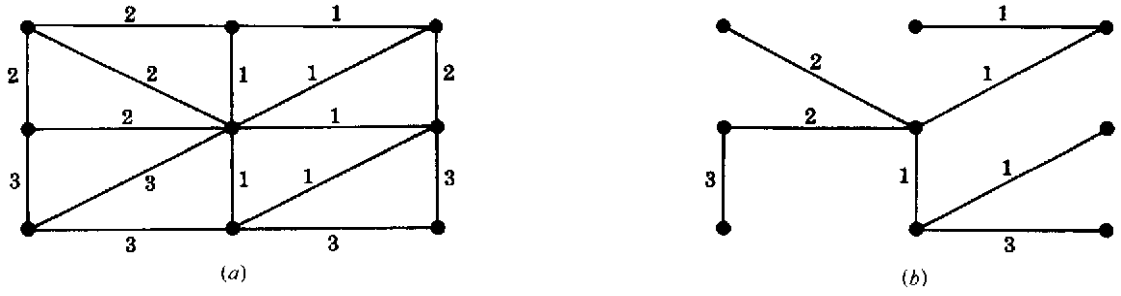



**Fig. 8-44**

Since $G$ has $n = 9$ vertices, $T$ must have $n - 1 = 8$ edges. Apply Algorithm 8.2, that is, keep deleting edges with maximum length and without disconnecting the graph until only $n-1 = 8$ edges remain. Alternatively, apply Algorithm 8.3, that is, beginning with the nine vertices, keep adding edges with minimum length and without forming any circle until $n - 1 = 8$ edges are added. Both methods give a minimum spanning tree such as that shown in Fig. 8-44($b$).

**8.13.** Let $G$ be a graph with more than one vertex. Prove the following are equivalent.

(i) $G$ is a tree.

(ii) Each pair of vertices is connected by exactly one simple path.

(iii) $G$ is connected; but $G - e$ is disconnected for any edge $e$ of $G$.

(iv) $G$ is cycle-free, but if any edge is added to $G$ then the resulting graph has exactly one cycle.

(i) *implies* (ii) Let $u$ and $v$ be two vertices in $G$. Since $G$ is a tree, $G$ is connected so there is at least one path between $u$ and $v$. By Problem 8.37 there can only be one simple path between $u$ and $v$, otherwise $G$ will contain a cycle.

(ii) *implies* (iii) Suppose we delete an edge $e = \{u, v\}$ from $G$. Note $e$ is a path from $u$ to $v$. Suppose the resulting graph $G - e$ has a path $P$ from $u$ to $v$. Then $P$ and $e$ are two distinct paths from $u$ to $v$, which contradicts the hypothesis. Thus there is no path between $u$ and $v$ in $G - e$, so $G - e$ is disconnected.

(iii) *implies* (iv) Suppose $G$ contains a cycle $C$ which contains an edge $e = \{u, v\}$. By hypothesis, $G$ is connected but $G' = G - e$ is disconnected, with $u$ and $v$ belonging to different components of $G'$ (Problem 8.41) This contradicts the fact that $u$ and $v$ are connected by the path $P = C - e$ which lies in $G'$. Hence $G$ is cycle-free. Now let $x$ and $y$ be vertices of $G$ and let $H$ be the graph obtained by adjoining the edge $e = \{x, y\}$ to $G$. Since $G$ is connected, there is a path $P$ from $x$ to $y$ in $G$; hence $C = Pe$ forms a cycle in $H$. Suppose $H$ contains another cycle $C'$. Since $G$ is cycle-free, $C'$ must contain the edge $e$, say $C' = P'e$. Then $P$ and $P'$ are two simple paths in $G$ from $x$ to $y$. (See Fig. 8-45.) By Problem 8.37, $G$ contains a cycle, which contradicts the fact that $G$ is cycle-free. Hence $H$ contains only one cycle.

(iv) *implies* (i) Since adding any edge $e = \{x, y\}$ to $G$ produces a cycle, the vertices $x$ and $y$ must already be connected in $G$. Hence $G$ is connected and by hypothesis $G$ is cycle-free; that is, $G$ is a tree.

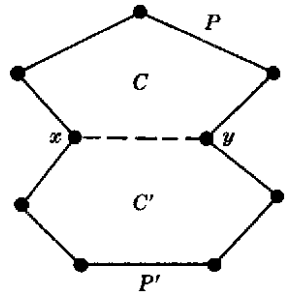



**Fig. 8-45**

**8.14.** Prove Theorem 8.6: Let $G$ be a finite graph with $n \geq 1$ vertices. Then the following are equivalent. (i) $G$ is a tree, (ii) $G$ is a cycle-free and has $n - 1$ edges, (iii) $G$ is connected and has $n - 1$ edges.

The proof is by induction on $n$. The theorem is certainly true for the graph with only one vertex and hence no edges. That is, the theorem holds for $n = 1$. We now assume that $n > 1$ and that the theorem holds for graphs with less than $n$ vertices.

(i) *implies* (ii) Suppose $G$ is a tree. Then $G$ is cycle-free, so we only need to show that $G$ has $n - 1$ edges. By Problem 8.38, $G$ has a vertex of degree 1. Deleting this vertex and its edge, we obtain a tree $T$ which has $n - 1$ vertices. The theorem holds for $T$, so $T$ has $n - 2$ edges. Hence $G$ has $n - 1$ edges.

(ii) *implies* (iii) Suppose $G$ is cycle-free and has $n - 1$ edges. We only need show that $G$ is connected. Suppose $G$ is disconnected and has $k$ components, $T_1, \ldots, T_k$, which are trees since each is connected and cycle-free. Say $T_i$ has $n_i$ vertices. Note $n_i < n$. Hence the theorem holds for $T_i$, so $T_i$ has $n_i - 1$ edges. Thus

$$n = n_1 + n_2 + \cdots + n_k$$

and

$$n - 1 = (n_1 - 1) + (n_2 - 1) + \cdots + (n_k - 1) = n_1 + n_2 + \cdots + n_k - k = n - k$$

Hence $k = 1$. But this contradicts the assumption that $G$ is disconnected and has $k > 1$ components. Hence $G$ is connected.

(iii) *implies* (i) Suppose $G$ is connected and has $n - 1$ edges. We only need to show that $G$ is cycle-free. Suppose $G$ has a cycle containing an edge $e$. Deleting $e$ we obtain the graph $H = G - e$ which is also connected. But $H$ has $n$ vertices and $n - 2$ edges, and this contradicts Problem 8.39. Thus $G$ is cycle-free and hence is a tree.

## PLANAR GRAPHS

**8.15.** Draw a planar representation, if possible, of the graphs (*a*), (*b*), and (*c*) in Fig. 8-46.

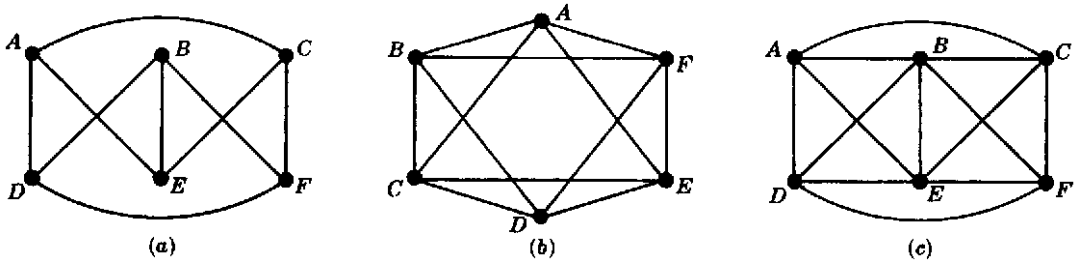



**Fig. 8-46**

(*a*) Redrawing the positions of $B$ and $E$, we get a planar representation of the graph as in Fig. 8-47(*a*).

(*b*) This is not the star graph $K_5$. This has a planar representation as in Fig. 8-47(*b*).

(*c*) This graph is non-planar. The utility graph $K_{3,3}$ is a subgraph as shown in Fig. 8-47(*c*) where we have redrawn the positions of $C$ and $F$.

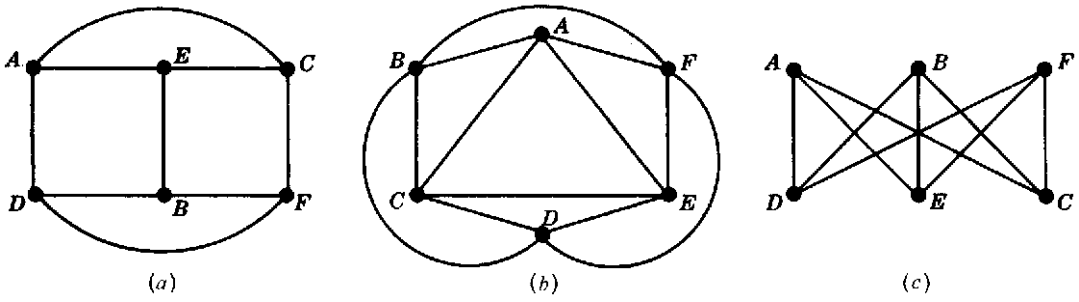



**Fig. 8-47**

**8.16.** Count the number $V$ of vertices, the number $E$ of edges, and the number $R$ of regions of each map in Fig. 8-48; and verify Euler's formula. Also find the degree $d$ of the outside region.

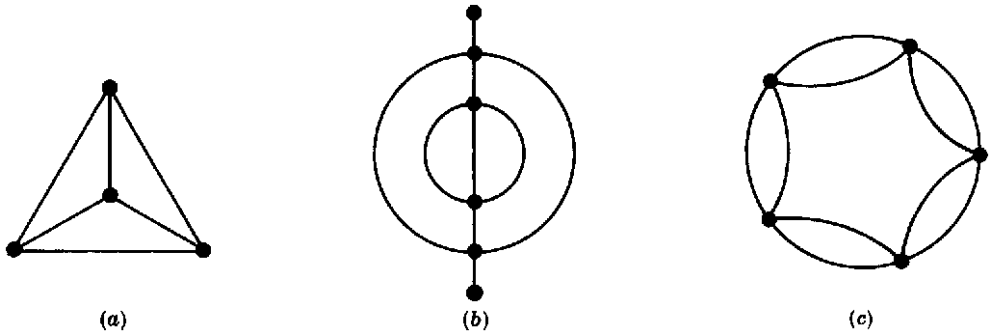



**Fig. 8-48**

(*a*) $V = 4$, $E = 6$, $R = 4$. Hence $V - E + R = 4 - 6 + 4 = 2$. Also $d = 3$.

(*b*) $V = 6$, $E = 9$, $R = 5$; so $V - E + R = 6 - 9 + 5 = 2$. Here $d = 6$ since two edges are counted twice.

(*c*) $V = 5$, $E = 10$, $R = 7$. Hence $V - E + R = 5 - 10 + 7 = 2$. Here $d = 5$.

**8.17.** Find the minimum number $n$ of colors required to paint each map in Fig. 8-48.

(*a*) $n = 4$; (*b*) $n = 3$; (*c*) $n = 2$.

**8.18.** Prove Theorem 8.8 (Euler): $V - E + R = 2$.

Suppose the connected map $M$ consists of a single vertex $P$ as in Fig. 8-49(*a*). Then $V = 1$, $E = 0$, and $R = 1$. Hence $V - E + R = 2$. Otherwise $M$ can be built up from a single vertex by the following two constructions:

(1) Add a new vertex $Q_2$ and connect it to an existing vertex $Q_1$ by an edge which does not cross any existing edge as in Fig. 8-49(*b*).

(2) Connect two existing vertices $Q_1$ and $Q_2$ by an edge $e$ which does not cross any existing edge as in Fig. 8-49(*c*).

Neither operation changes the value of $V - E + R$. Hence $M$ has the same value of $V - E + R$ as the map consisting of a single vertex, that is, $V - E + R = 2$. Thus the theorem is proved.

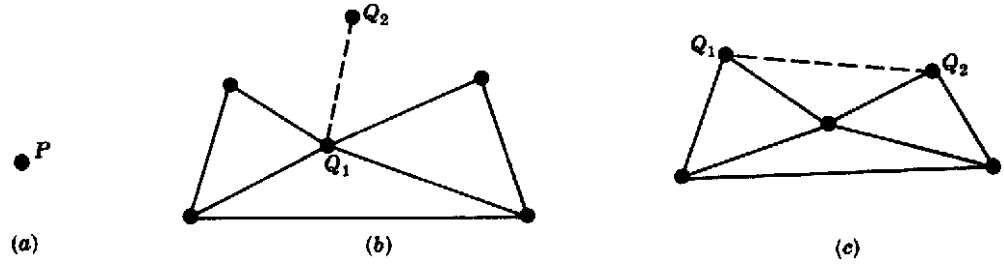



**Fig. 8-49**

**8.19.** Prove Theorem 8.11: The following are equivalent for a graph $G$: (i) $G$ is 2-colorable. (ii) $G$ is bipartite. (iii) Every cycle of $G$ has even length.

(i) *implies* (ii). Suppose $G$ is 2-colorable. Let $M$ be the set of vertices painted the first color, and let $N$ be the set of vertices painted the second color. Then $M$ and $N$ form a bipartite partition of the vertices of $G$ since neither the vertices of $M$ nor the vertices of $N$ can be adjacent to each other since they are of the same color.

(ii) *implies* (iii). Suppose $G$ is bipartite and $M$ and $N$ form a bipartite partition of the vertices of $G$. If a cycle begins at a vertex $u$ of, say, $M$, then it will go to a vertex of $N$, and then to a vertex of $M$, and then to $N$ and so on. Hence when the cycle returns to $u$ it must be of even length. That is, every cycle of $G$ will have even length.

(iii) *implies* (i). Lastly, suppose every cycle of $G$ has even length. We pick a vertex in each connected component and paint it the first color, say red. We then successively paint all the vertices as follows: If a vertex is painted red, then any vertex adjacent to it will be painted the second color, say blue. If a vertex is painted blue, then any vertex adjacent to it will be painted red. Since every cycle has even length, no adjacent vertices will be painted the same color. Hence $G$ is 2-colorable, and the theorem is proved.

**8.20.** Prove Theorem 8.12: A planar graph $G$ is 5-colorable.

The proof is by induction on the number $p$ of vertices of $G$. If $p \leq 5$, then the theorem obviously holds. Suppose $p > 5$, and the theorem holds for graphs with less than $p$ vertices. By the preceding problem, $G$ has a vertex $v$ such that $\deg(v) \leq 5$. By induction, the subgraph $G - v$ is 5-colorable. Assume one such coloring. If the vertices adjacent to $v$ use less than the five colors, than we simply paint $v$ with one of the remaining colors and obtain a 5-coloring of $G$. We are still left with the case that $v$ is adjacent to five vertices which are painted different colors. Say the vertices, moving counterclockwise about $v$, are $v_1, \ldots, v_5$ and are painted respectively by the colors $c_1, \ldots, c_5$. (See Fig. 8-50(*a*).)

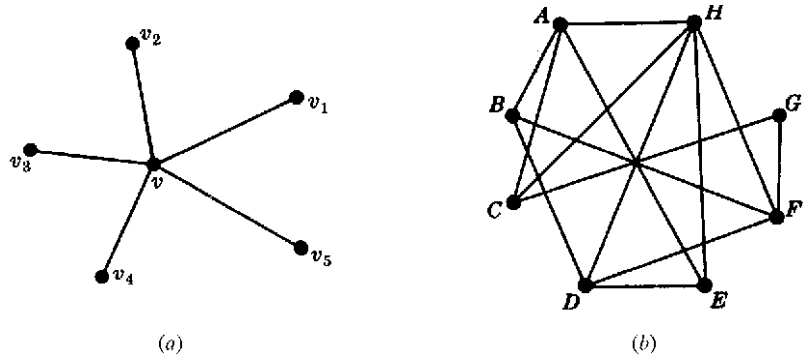



**Fig. 8-50**

Consider now the subgraph $H$ of $G$ generated by the vertices painted $c_1$ and $c_3$. Note $H$ includes $v_1$ and $v_3$. If $v_1$ and $v_3$ belong to different components of $H$, then we can interchange the colors $c_1$ and $c_3$ in the component containing $v_1$ without destroying the coloring of $G - v$. Then $v_1$ and $v_3$ are painted by $c_3$, $c_1$ can be chosen to paint $v$, and we have a 5-coloring of $G$. On the other hand, suppose $v_1$ and $v_3$ are in the same component of $H$. Then there is a path $P$ from $v_1$ to $v_3$ whose vertices are painted either $c_1$ or $c_3$. The path $P$ together with the edges $\{v, v_1\}$ and $\{v, v_3\}$ form

a cycle $C$ which encloses either $v_2$ or $v_4$. Consider now the subgraph $K$ generated by the vertices painted $c_3$ or $c_4$. Since $C$ encloses $v_2$ or $v_4$, but not both, the vertices $v_2$ and $v_4$ belong to different components of $K$. Thus we can interchange the colors $c_2$ and $c_4$ in the component containing $v_2$ without destroying the coloring of $G - v$. Then $v_2$ and $v_4$ are painted by $c_4$, and we can choose $c_2$ to paint $v$ and obtain a 5-coloring of $G$. Thus $G$ is 5-colorable and the theorem is proved.

**8.21.** Use the Welch-Powell Algorithm 8.4 (Fig. 8-24) to paint the graph in Fig. 8-50($b$).

First order the vertices according to decreasing degrees to obtain the sequence

$$H, \quad A, \quad D, \quad F, \quad B, \quad C, \quad E, \quad G$$

Proceeding sequentially, we use the first color to paint the vertices $H$, $B$, and then $G$. (We cannot paint $A$, $D$, or $F$ the first color since each is connected to $H$, and we cannot paint $C$ or $E$ the first color since each is connected to either $H$ or $B$.) Proceeding sequentially with the unpainted vertices, we use the second color to paint the vertices $A$ and $D$. The remaining vertices $F$, $C$, and $E$ can be painted with the third color. Thus the chromatic number $n$ cannot be greater than 3. However, in any coloring, $H$, $D$, and $E$ must be painted different colors since they are connected to each other. Hence $n = 3$.

**8.22.** Let $G$ be a finite connected planar graph with at least three vertices. Show that $G$ has at least one vertex of degree 5 or less.

Let $p$ be the number of vertices and $q$ the number of edges of $G$, and suppose $\deg(u) \geq 6$ for each vertex $u$ of $G$. But $2q$ equals the sum of the degrees of the vertices of $G$ (Theorem 8.1); so $2q \geq 6p$. Therefore

$$q \geq 3p > 3p - 6$$

This contradicts Theorem 8.9. Thus some vertex of $G$ has degree 5 or less.

## SEQUENTIAL REPRESENTATION OF GRAPHS

**8.23.** Find the adjacency matrix $A = [a_{ij}]$ of each graph $G$ in Fig. 8-51.

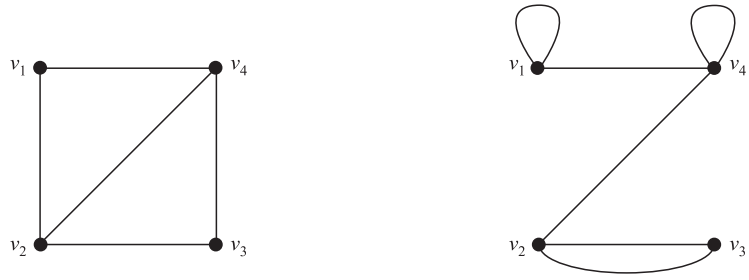



**Fig. 8-51**

Set $a_{ij} = n$ if there are $n$ edges $\{v_i, v_j\}$ and $a_{ij} = 0$ otherwise. Hence:

$$(a)\ A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}; \quad (b)\ A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 0 & 2 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

(Since ($a$) has no multiple edges and no loops, the entries in $A$ are either 0 or 1, and are 0 on the diagonal.)

**8.24.** Draw the graph $G$ corresponding to each adjacency matrix:

$$(a)\ A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix}; \quad (b)\ A = \begin{bmatrix} 1 & 3 & 0 & 0 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & 2 & 2 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

(*a*) Since $A$ is a 5-square matrix, $G$ has five vertices, say, $v_1, v_2 \ldots, v_5$. Draw an edge from $v_i$ to $v_j$ when $a_{ij} = 1$. The graph appears in Fig. 8-52(*a*).

(*b*) Since $A$ is a 4-square matrix, $G$ has four vertices, say, $v_1, \ldots, v_4$. Draw $n$ edges from $v_i$ to $v_j$ when $a_{ij} = n$. Also, draw $n$ loops at $v_i$ when $a_i = n$. The graph appears in Fig. 8-52(*b*).

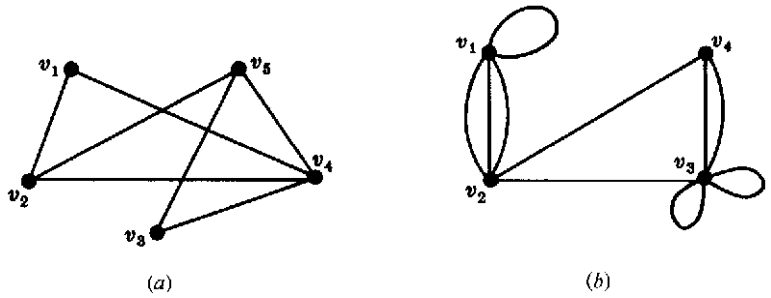
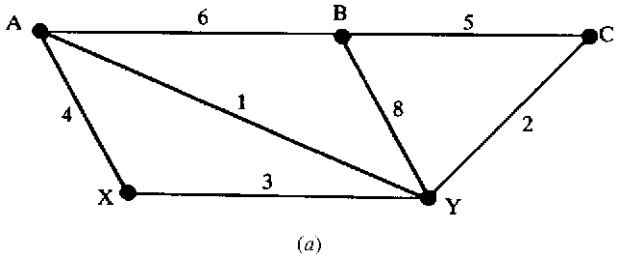



**Fig. 8-52**

**8.25.** Find the weight matrix $W = [w_{ij}]$ of the weighted graph $G$ in Fig. 8-53(*a*) where the vertices are stored in the array DATA as follows: DATA: $A$, $B$, $C$, $X$, $Y$.




$$W = \begin{bmatrix} 0 & 6 & 0 & 4 & 1 \\ 6 & 0 & 5 & 0 & 8 \\ 0 & 5 & 0 & 0 & 2 \\ 4 & 0 & 0 & 0 & 3 \\ 1 & 8 & 2 & 3 & 0 \end{bmatrix}$$

(*b*)

**Fig. 8-53**

The vertices are numbered according to the way they are stored in the array DATA; so $v_1 = A$, $v_2 = B, \ldots,$ $v_5 = Y$. Then set $W_{ij} = w$, where $w$ is the weight of the edge from $v_i$ to $v_j$. This yields the matrix $W$ in Fig. 8-53(*b*).

## LINKED REPRESENTATION OF GRAPHS

**8.26.** A graph $G$ with vertices $A, B, \ldots, F$ is stored in memory using a linked representation with a vertex file and an edge file as in Fig. 8-54.

(*a*) List the vertices in the order they appear in memory.

(*b*) Find the adjacency list adj($v$) of each vertex $v$ of $G$.

(*a*) Since START = 4, the list begins with the vertex $D$. The NEXT-V tells us to go to 1($B$), then 3($F$), then 5($A$), then 8($E$), and then 7($C$); that is,

$$D, \quad B, \quad F, \quad A, \quad E, \quad C$$

(*b*) Here adj($D$) = [5($A$), 1($B$), 8($E$)]. Specifically, PTR[4($D$)] = 7 and ADJ[7] = 5($A$) tells us that adj($D$) begins with $A$. Then NEXT[7] = 3 and ADJ[3] = 1($B$) tells us that $B$ is the next vertex in adj($D$). Then NEXT[3] = 10 and ADJ[10] = 8($E$) tells us that $E$ in the next vertex in adj($D$). However, NEXT[10] = 0 tells us that there are no more neighbors of $D$. Similarly.

$$\text{adj}(B) = [A, D], \quad \text{adj}(F) = [E], \quad \text{adj}(A) = [B, D], \quad \text{adj}(E) = [C, D, F], \quad \text{adj}(C) = [E]$$

In other words, the following is the adjacency structure of $G$:

$$G = [A{:}B, D; \quad B{:}A, D; \quad C{:}E; \quad D{:}A, B, E; \quad E{:}C, D, F; \quad F{:}E]$$

Vertex file

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| | VERTEX | *B* | | *F* | *D* | *A* | | *C* | *E* |
| START 4 | NEXT-V | 3 | | 5 | 1 | 8 | | 0 | 7 |
| | PTR | 9 | | 4 | 7 | 6 | | 5 | 12 |

Edge file

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADJ | 4 | 4 | 1 | 8 | 8 | 1 | 5 | 3 | 5 | 8 | 4 | 7 | | |
| NEXT | 8 | 0 | 10 | 0 | 0 | 2 | 3 | 0 | 11 | 0 | 0 | 1 | | |

**Fig. 8-54**

**8.27.** Draw the diagram of the graph $G$ whose linked representation appears in Fig. 8-54.

Use the vertex list obtained in Problem 8.26(a) and the adjacency lists obtained in Problem 8.26(b) to draw the graph $G$ in Fig. 8-55.

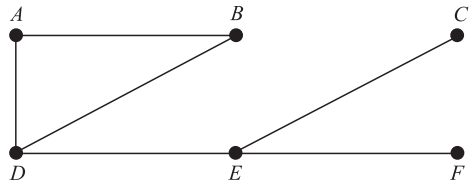



**Fig. 8-55**

**8.28.** Exhibit the adjacency structure ($AS$) of the graph $G$ in: (*a*) Fig. 8-56(*a*), (*b*) Fig. 8-56(*b*).

The adjacency structure of a graph $G$ consists of the adjacency lists of the vertices where we use a colon ":" to separate a vertex from its adjacency list, and a semicolon ";" to separate the different lists. Thus:

(*a*) $G = [A{:}B, C, D; \quad B{:}A, C, E; \quad C{:}A, B, D, E; \quad D{:}A, C; \quad E{:}B, C]$

(*b*) $G = [A{:}B, D; \quad B{:}A, C, E; \quad C{:}B, E, F; \quad D{:}A, E; \quad E{:}B, C, D, F; \quad F{:}C, E]$

## GRAPH ALGORITHMS

**8.29.** Consider the graph $G$ in Fig. 8-56(*a*) (where the vertices are ordered alphabetically).

(*a*) Find the adjacency structure of $G$.

(*b*) Find the order in which the vertices of $G$ are processed using a DFS (depth-first search) algorithm beginning at vertex $A$.

(*a*) List the neighbors of each vertex as follows:

$$G = [A{:}B, C, D; \quad B{:}A, J; \quad C{:}A; \quad D{:}A, K; \quad J{:}B, K, M; \quad K{:}D, J, L; \quad L{:}K, M; \quad M{:}J, L]$$

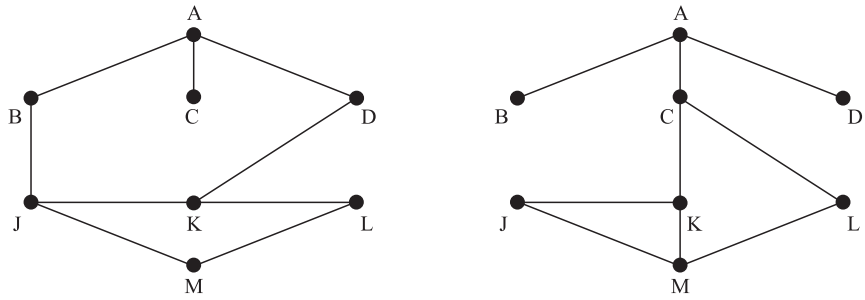



**Fig. 8-56**

(*b*) During the DFS algorithm, the first vertex $N$ in STACK is processed and the neighbors of $N$ (which have not been previously processed) are then pushed onto STACK. Initially, the beginning vertex $A$ is pushed onto STACK. The following shows the sequence of waiting lists in STACK and the vertices being processed:

| STACK | $A$ | $DCB$ | $KCB$ | $LJCB$ | $MJCB$ | $JCB$ | $CB$ | $B$ | $\varnothing$ |
|---|---|---|---|---|---|---|---|---|---|
| Vertex | $A$ | $D$ | $K$ | $L$ | $M$ | $J$ | $C$ | $B$ | |

In other words the vertices are processed in the order: $A, D, K, L, M, J, C, B$.

**8.30.** Repeat Problem 8.29 for the graph $G$ in Fig. 8-56(*b*).

(*a*) List the neighbors of each vertex as follows:

$$G = [A{:}B, C, D;\quad B{:}A;\quad C{:}A, K, L;\quad D{:}A;\quad J{:}K, M;\quad K{:}C, J, M;\quad L{:}C, M;\quad M{:}J, K, L]$$

(*b*) The following shows the sequence of waiting lists in STACK and the vertices being processed:

| STACK | $A$ | $DCB$ | $CB$ | $LKB$ | $MKB$ | $KJB$ | $JB$ | $B$ | $\varnothing$ |
|---|---|---|---|---|---|---|---|---|---|
| Vertex | $A$ | $D$ | $C$ | $L$ | $M$ | $K$ | $J$ | $B$ | |

In other words the vertices are processed in the order: $A, D, C, L, M, K, J, B$.

**8.31.** Beginning at vertex $A$ and using a BFS (breadth-first search) algorithm, find the order the vertices are processed for the graph $G$: (*a*) in Fig. 8-56(*a*), (*b*) in Fig. 8-56(*b*).

(*a*) The adjacency structure of $G$ appears in Problem 8.29. During the BFS algorithm, the first vertex $N$ in QUEUE is processed and the neighbors of $N$ (which have not appeared previously) are then added onto QUEUE. Initially, the beginning vertex $A$ is assigned to QUEUE. The following shows the sequence of waiting lists in QUEUE and the vertices being processed:

| QUEUE | $A$ | $DCB$ | $JDC$ | $JD$ | $KJ$ | $MK$ | $LM$ | $L$ | $\varnothing$ |
|---|---|---|---|---|---|---|---|---|---|
| Vertex | $A$ | $B$ | $C$ | $D$ | $J$ | $K$ | $M$ | $L$ | |

In other words the vertices are processed in the order: $A, B, C, D, J, K, M, L$.

(*b*) The adjacency structure of $G$ appears in Problem 8.30. The following shows the sequence of waiting lists in QUEUE and the vertices being processed:

| QUEUE | $A$ | $DCB$ | $DC$ | $LKD$ | $LK$ | $MJL$ | $MJ$ | $M$ | $\varnothing$ |
|---|---|---|---|---|---|---|---|---|---|
| Vertex | $A$ | $B$ | $C$ | $D$ | $K$ | $L$ | $J$ | $M$ | |

In other words the vertices are processed in the order: $A, B, C, D, K, L, J, M$.

## TRAVELING–SALESMAN PROBLEM

**8.32.** Apply the nearest-neighbor algorithm to the complete weighted graph $G$ in Fig. 8-57 beginning at: (*a*) vertex $A$; (*b*) vertex $D$.

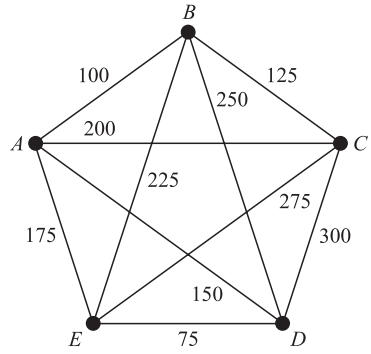



**Fig. 8-57**

(*a*) Starting at $A$, the closest vertex is $B$ with distance 100; from $B$, the closest is $C$ with distance 125; and from $C$ the closest is $E$ with distance 275. From $E$ we must go to $D$ with distance 75; and finally from $D$ we must go back to $A$ with distance 150. Accordingly, the nearest-neighbor algorithm beginning at $A$ yields the following weighted Hamiltonian circuit:

$$|ABCEDA| = 100 + 125 + 275 + 75 + 150 = 725$$

(*b*) Starting at $D$, we must go to $E$, then $A$, then $B$ then $C$ and finally back to $D$. Accordingly, the nearest-neighbor algorithm beginning at $D$ yields the following weighted Hamiltonian circuit:

$$|DEABCD| = 75 + 175 + 100 + 125 + 300 = 775$$

**8.33.** Prove Theorem 8.13. The complete graph $K_n$ with $n \geq 3$ vertices has $H = (n-1)!/2$ Hamiltonian circuits.

The counting convention for Hamiltonian circuits enables us to designate any vertex in a circuit as the starting point. From the starting point, we can go to any $n-1$ vertices, and from there to any one of $n-2$ vertices, and so on until arriving at the last vertex and then returning to the starting point. By the basic counting principle, there are a total of $(n-1)(n-2)\cdots 2 \cdot 1 = (n-1)!$ circuits that can be formed from a stating point. For $n \geq 3$, any circuit can be paired with one in the opposite direction which determines the same Hamiltonian circuit. Accordingly, there are a total of $H = (n-1)!/2$ Hamiltonian circuits.

# Supplementary Problems

## GRAPH TERMINOLOGY

**8.34.** Consider the graph $G$ in Fig. 8-58. Find:

(a) degree of each vertex (and verify Theorem 8.1);

(b) all simple paths from $A$ to $L$;

(c) all trails (distinct edges) from $B$ to $C$;

(d) $d(A, C)$, distance from $A$ to $C$;

(e) diam$(G)$, the diameter of $G$.

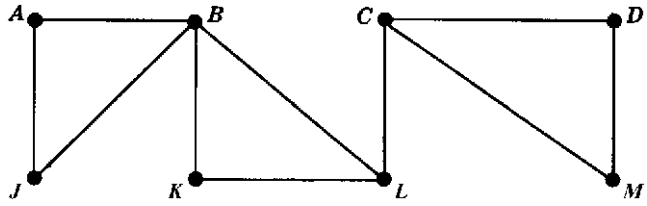



**Fig. 8-58**

**8.35.** Consider the graph in Fig. 8-58. Find (if any): (a) all cycles; (b) all cut points; (c) all bridges.

**8.36.** Consider the graph in Fig. 8-58. Find the subgraph $H = H(V', E')$ of $G$ where $V'$ equals:

(a) $\{B, C, D, J, K\}$ (b) $\{A, C, J, L, M\}$ (c) $\{B, D, J, M\}$ (d) $\{C, K, L, M\}$

Which of them are isomorphic and which homeomorphic?

**8.37.** Suppose a graph $G$ contains two distinct paths from a vertex $u$ to a vertex $v$. Show that $G$ has a cycle.

**8.38.** Suppose $G$ is a finite cycle-free graph with at least one edge. Show that $G$ has at least two vertices of degree 1.

**8.39.** Show that a connected graph $G$ with $n$ vertices must have at least $n - 1$ edges.

**8.40.** Find the number of connected graphs with four vertices. (Draw them.)

**8.41.** Let $G$ be a connected graph. Prove:

(a) If $G$ contains a cycle $C$ which contains an edge $e$, then $G - e$ is still connected.

(b) If $e = \{u, v\}$ is an edge such that $G - e$ is disconnected, then $u$ and $v$ belong to different components of $G - e$.

**8.42.** Suppose $G$ has $V$ vertices and $E$ edges. Let $M$ and $m$ denote, respectively, the maximum and minimum of the degrees of the vertices in $G$. Show that $m \leq 2E/V \leq M$.

**8.43.** Consider the following two steps on a graph $G$: (1) Delete an edge. (2) Delete a vertex and all edges containing that vertex. Show that every subgraph $H$ of a finite graph $G$ can be obtained by a sequence consisting of these two steps.

## TRAVERSABLE GRAPHS, EULER AND HAMILTONIAN CIRCUITS

**8.44.** Consider the graphs $K_5$, $K_{3,3}$ and $K_{2,3}$ in Fig. 8-59. Find an Euler (traversable) path or an Euler circuit of each graph, if it exists. If it does not, why not?

**8.45.** Consider each graph in Fig. 8-59. Find a Hamiltonian path or a Hamiltonian circuit, if it exists. If it does not, why not?

**8.46.** Show that $K_n$ has $H = (n - 1)!/2$ Hamiltonian circuits. In particular, find the number of Hamiltonian circuits for the graph $K_5$ in Fig. 8-59$(a)$.

**8.47.** Suppose $G$ and $G^*$ are homeomorphic graphs. Show that $G$ is traversable (Eulerian) if and only if $G^*$ is traversable (Eulerian).

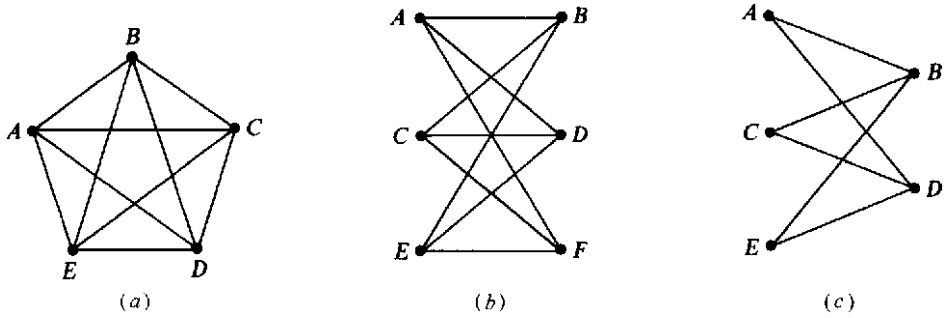



**Fig. 8-59**

## SPECIAL GRAPHS

**8.48.** Draw two 3-regular graphs with: (a) eight vertices; (b) nine vertices.

**8.49.** Consider the complete graph $K_n$.

(a) Find the diameter of $K_n$.

(b) Find the number $m$ of edges in $K_n$.

(c) Find the degree of each vertex in $K_n$.

(d) Find those values of $n$ for which $K_n$ is: (i) traversable; (ii) regular.

**8.50.** Consider the complete graph $K_{m,n}$.

(a) Find the diameter of $K_{m,n}$.

(b) Find the number $E$ of edges in $K_{m,n}$.

(c) Find those $K_{m,n}$ which are traversable.

(d) Which of the graphs $K_{m,n}$ are isomorphic and which homeomorphic?

**8.51.** The *n-cube*, denoted by $\mathbf{Q}_n$, is the graph whose vertices are the $2^n$ bit strings of length $n$, and where two vertices are adjacent if they differ in only one position. Figure 8-60(*a*) and (*b*) show the $n$-cubes $\mathbf{Q}_2$ and $\mathbf{Q}_3$.

(a) Find the diameter of $\mathbf{Q}_n$.

(b) Find the number $m$ of edges in $\mathbf{Q}_n$.

(c) Find the degree of each vertex in $\mathbf{Q}_n$.

(d) Find those values of $n$ for which $\mathbf{Q}_n$ is traversable.

(e) Find a Hamiltonian circuit (called a *Gray code*) for (i) $\mathbf{Q}_3$; (ii) $\mathbf{Q}_4$.

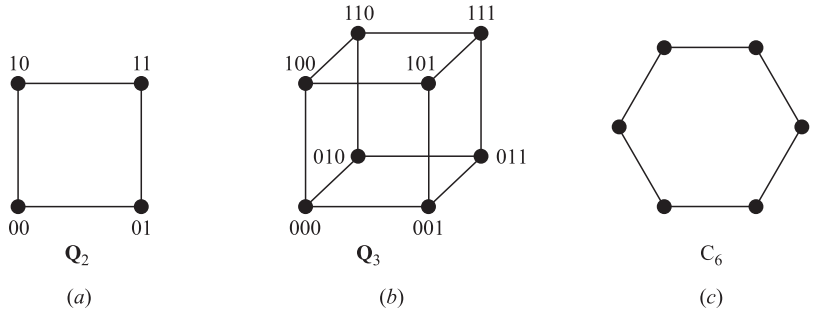



**Fig. 8-60**

**8.52.** The *n-cycle*, denoted by $C_n$, is the graph which consists of only a single cycle of length $n$. Figure 8-60($c$) shows the 6-cycle $C_6$. (a) Find the number of vertices and edges in $C_n$. (b) Find the diameter of $C_n$.

**8.53.** Describe those connected graphs which are both bipartite and regular.

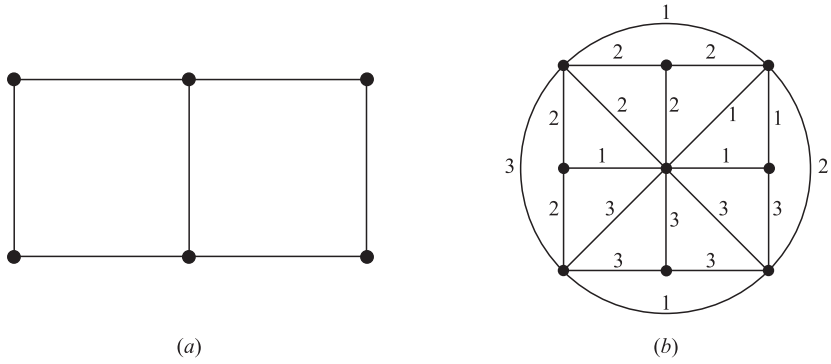
## TREES

**8.54.** Draw all trees with five or fewer vertices.

**8.55.** Find the number of trees with seven vertices.

**8.56.** Find the number of spanning trees in Fig. 8-61($a$).

**8.57.** Find the weight of a minimum spanning tree in Fig. 8-61($b$)




($a$) ($b$)

**Fig. 8-61**

**8.58.** Show that any tree is a bipartite graph.

**8.59.** Which complete bipartite graphs $K_{m,n}$ are trees.

## PLANAR GRAPHS, MAPS, COLORINGS

**8.60.** Draw a planar representation of each graph $G$ in Fig. 8-62, if possible; otherwise show that it has a subgraph homeomorphic to $K_5$ or $K_{3,3}$.

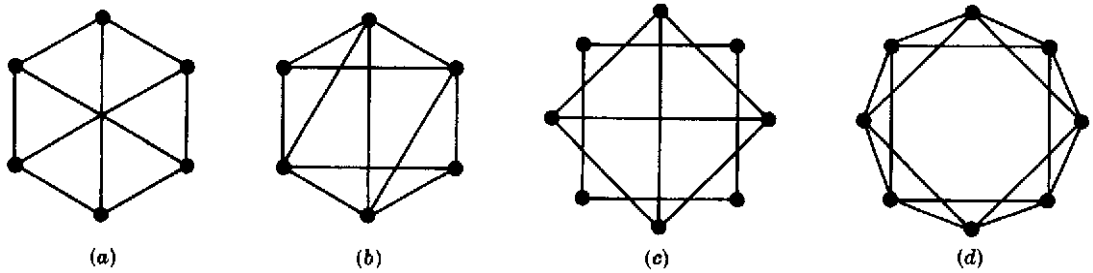



**Fig. 8-62**

**8.61.** Show that the 3-cube $Q_3$ (Fig. 8-60($b$)) is planar.

**8.62.** For the map in Fig. 8-63, find the degree of each region and verify that the sum of the degrees of the regions is equal to twice the number of edges.

**8.63.** Count the number $V$ of vertices, the number $E$ of edges, and the number $R$ of regions of each of the maps in Fig. 8-64, and verify Euler's formula.

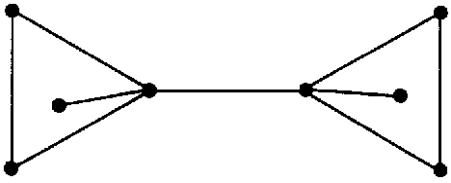


**Fig. 8-63**

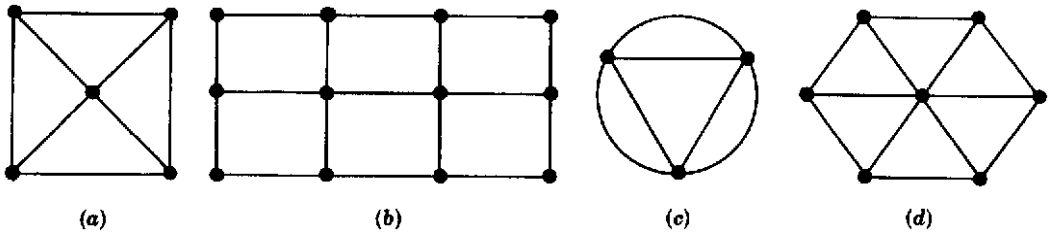



**Fig. 8-64**

**8.64.** Find the minimum number of colors needed to paint the regions of each map in Fig. 8-64.

**8.65.** Draw the map which is dual to each map in Fig. 8-64.

**8.66.** Use the Welch-Powell algorithm to paint each graph in Fig. 8-65. Find the chromatic number $n$ of the graph.

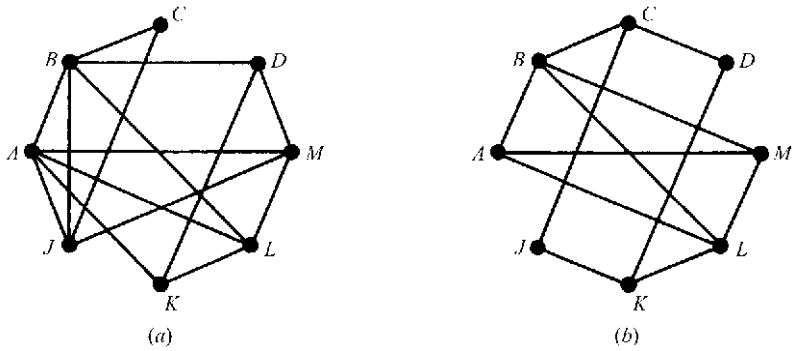



**Fig. 8-65**

## SEQUENTIAL REPRESENTATION OF GRAPHS

**8.67.** Find the adjacency matrix $A$ of each multigraph in Fig. 8-66.

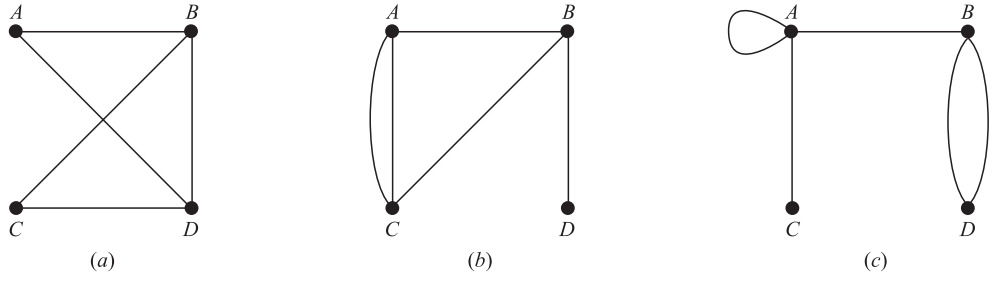



**Fig. 8-66**

**8.68.** Draw the multigraph $G$ corresponding to each of the following adjacency matrices:

$$\text{(a) } A = \begin{bmatrix} 0 & 2 & 0 & 1 \\ 2 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}; \quad \text{(b) } A = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 2 \\ 2 & 0 & 2 & 2 \end{bmatrix}$$

**8.69.** Suppose a graph $G$ is bipartite. Show that one can order the vertices of $G$ so that its adjacency matrix $A$ has the form: $A = \begin{bmatrix} 0 & B \\ C & 0 \end{bmatrix}$

## LINKED REPRESENTATION OF GRAPHS

**8.70.** Suppose a graph $G$ is stored in memory as in Fig. 8-67.

Vertex file

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| | VERTEX | *C* | | *F* | *E* | *A* | | *B* | *D* |
| START 7 | NEXT-V | 0 | | 5 | 1 | 8 | | 3 | 4 |
| | PTR | 2 | | 11 | 6 | 12 | | 4 | 1 |

Edge file

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADJ | 7 | 7 | 4 | 5 | | 7 | 1 | | 8 | 3 | 1 | 7 |
| NEXT | 0 | 10 | 0 | 7 | | 0 | 9 | | 3 | 0 | 0 | 0 |

**Fig. 8-67**

(a) List the vertices in the order in which they appear in memory.

(b) Find the adjacency structure of $G$, that is, find the adjacency list adj($v$) of each vertex $v$ of $G$.

**8.71.** Exhibit the adjacency structure (AS) for each graph $G$ in Fig. 8-59.

**8.72.** Figure 8-68($a$) shows a graph $G$ representing six cities $A$, $B$, ..., $F$ connected by seven highways numbered 22, 33, ..., 88. Show how $G$ may be maintained in memory using a linked representation with sorted arrays for the cities and for the numbered highways. (Note that VERTEX is a sorted array and so the field NEXT-V is not needed.)

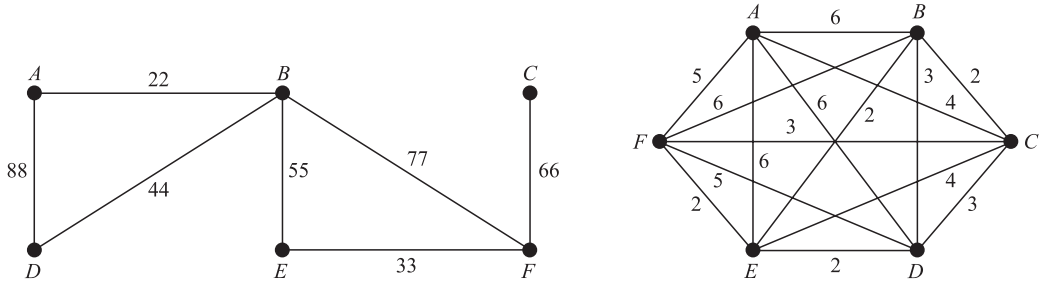



**Fig. 8-68**

## TRAVELING–SALESMAN PROBLEM

**8.73.** Apply the nearest-neighbor algorithm to the complete weighted graph $G$ in Fig. 8-68($b$) beginning at: (a) vertex $A$; (b) vertex $B$.

**8.74.** Consider the complete weighted graph $G$ in Fig. 8-57 with 5 vertices.

(a) Beginning at vertex $A$, list the $H = (n-1)!/2 = 12$ Hamiltonian circuits of $G$, and find the weight of each of them.

(b) Find a Hamiltonian circuit of minimal weight.

## GRAPH ALGORITHMS

**8.75.** Consider the graph $G$ in Fig. 8-57 (where the vertices are ordered alphabetically).

(a) Find the adjacency structure (AS) of $G$.

(b) Using the DFS (depth-first search) Algorithm 8.5 on $G$ and beginning at vertex $C$, find the STACK sequence and the order in which the vertices are processed.

(c) Repeat (b) beginning at vertex $K$.

**8.76.** Using the BFS (breadth-first search) Algorithm 8.6 on the graph $G$ in Fig. 8-57, find the QUEUE sequence and the order the vertices are processed beginning at: (a) vertex $C$; (b) vertex $K$.

**8.77.** Repeat Problem 8.75 for the graph $G$ in Fig. 8-65($a$).

**8.78.** Repeat Problem 8.76 for the graph $G$ in Fig. 8-65($a$).

**8.79.** Repeat Problem 8.75 for the graph $G$ in Fig. 8-65($b$).

**8.80.** Repeat Problem 8.76 for the graph $G$ in Fig. 8-65($b$).

### Answers to Supplementary Problems

**8.34.** (a) 2, 4, 3, 2, 2, 2, 3, 2; (b) $ABL$, $ABKL$, $AJBL$, $AJBKL$; (c) $BLC$, $BKLC$, $BAJBLC$, $BAJBKLC$; (d) 3; (e) 4.

**8.35.** (a) $AJBA$, $BKLB$, $CDMC$; (b) $B, C, L$; (c) only $\{C, L\}$.

**8.36.** (a) $E' = \{BJ, BK, CD\}$; (b) $E' = \{AJ, CM, LC\}$; (c) $E' = \{BJ, DM\}$; (d) $E' = \{KL, LC, CM\}$, Also, (a) and (b) are isomorphic, and (a), (b), and (c) are homeomorphic.

**8.38.** *Hint*: Consider a maximal simple path $\alpha$, and show that its endpoints have degree 1.

**8.40.** There are five of them, as shown in Fig. 8-69.

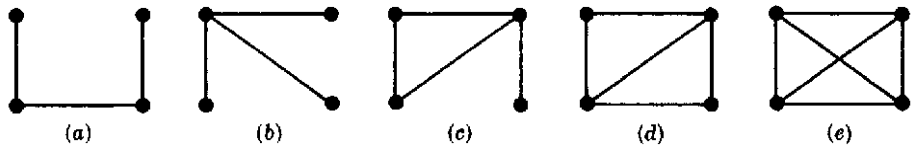



**Fig. 8-69**

**8.42.** *Hint*: Use Theorem 8.1.

**8.43.** First delete all edges in $G$ not in $H$, then delete all vertrices in $G$ not in $H$.

**8.44.** (a) Eulerian since all vertices are even: $ABCDEACEBDA$. (b) None, since four vertices are odd. (c) Euler path beginning at $B$ and ending at $D$ (or vice versa): $BADCBED$.

**8.45.** (a) $ABCDEA$; (b) $ABCDEFA$; (c) none, since $B$ or $D$ must be visited twice in any closed path including all vertices.

**8.46.** $(5-1)!/2 = 12$.

**8.47.** *Hint*: Adding a vertex by dividing an edge does not change the degree of the original vertices and simply adds a vertex of even degree.

**8.48.** (a) The two 3-regular graphs in Fig. 8-70 are not isomorphic: (b) has a 5-cycle, but (a) does not. (b) There are none. The sum of the degrees of an $r$-regular graph with $s$ vertices equals $rs$, and $rs$ must be even.

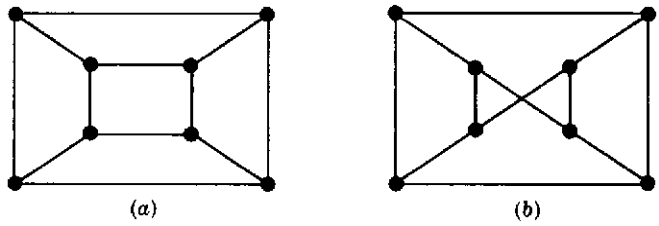



**Fig. 8-70**

**8.49.** (a) $\text{diam}(K_1) = 0$; all others have diameter 1; (b) $m = C(n,2) = n(n-1)/2$; (c) $n-1$; (d) (i) $n = 2$ and $n$ odd; (ii) all $n$.

**8.50.** (a) $\text{diam}(K_{1,1}) = 1$; all others have diameter 2; (b) $E = mn$; (c) $K_{1,1}$, $K_{1,2}$, and all $K_{m,n}$ where $m$ and $n$ are even; (d) none are isomorphic; only $K_{1,1}$ and $K_{1,2}$ are homeomorphic.

**8.51.** (a) $n$; (b) $n2^{n-1}$; (c) $n$; (d) $n = 1$, even; (e) consider the $4 \times 16$ matrix:

$$M = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

which shows how $\mathbf{Q}_4$ (the columns of $M$) is obtained from $\mathbf{Q}_3$. That is, the upper left $3 \times 8$ submatrix of $M$ is $\mathbf{Q}_3$, the upper right $3 \times 8$ submatrix of $M$ is $\mathbf{Q}_3$ written in reverse, and the last row consists of eight 0s followed by eight 1s.

**8.52.** (a) $n$ and $n$; (b) $n/2$ when $n$ is even, $(n+1)/2$ when $n$ is odd.

**8.53.** $K_{m,m}$ is bipartite and $m$-regular. Also, beginning with $K_{m,m}$, delete $m$ disjoint edges to obtain a bipartite graph which is $(m-1)$-regular, delete another $m$ disjoint edges to obtain a bipartite graph which is $(m-2)$-regular, and so on. These graphs may be disconnected, but their connected components have the desired properties.

**8.54.** There are eight such trees, as shown in Fig. 8-71. The graph with one vertex and no edges is called the *trivial tree*.

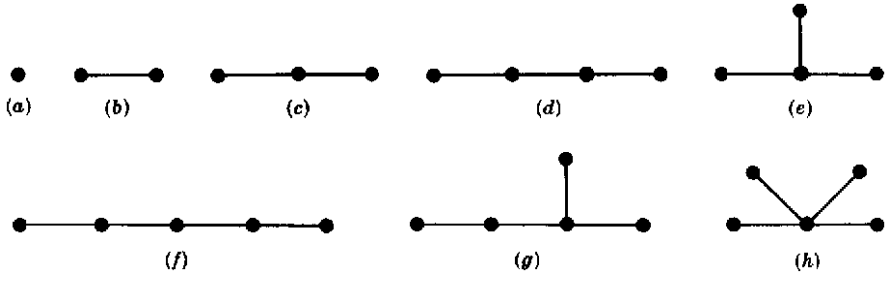



**Fig. 8-71**

**8.55.** 10

**8.56.** 15

**8.57.** $1 + 1 + 1 + 1 + 1 + 2 + 2 + 3 = 12$.

**8.59.** $m = 1$.

**8.60.** Only (a) is nonplanar, and $K_{3,3}$ is a subgraph.

**8.61.** Figure 8-70($a$) is a planar representation of $Q3$.

**8.62.** The outside region has degree 8, and the other two regions have degree 5.

**8.63.** (a) 5, 8, 5; (b) 12, 17, 7; (c) 3, 6, 5; (d) 7, 12, 7.

**8.64.** (a) 3; (b) 3; (c) 2; (d)3.

**8.65.** See Fig. 8-72.

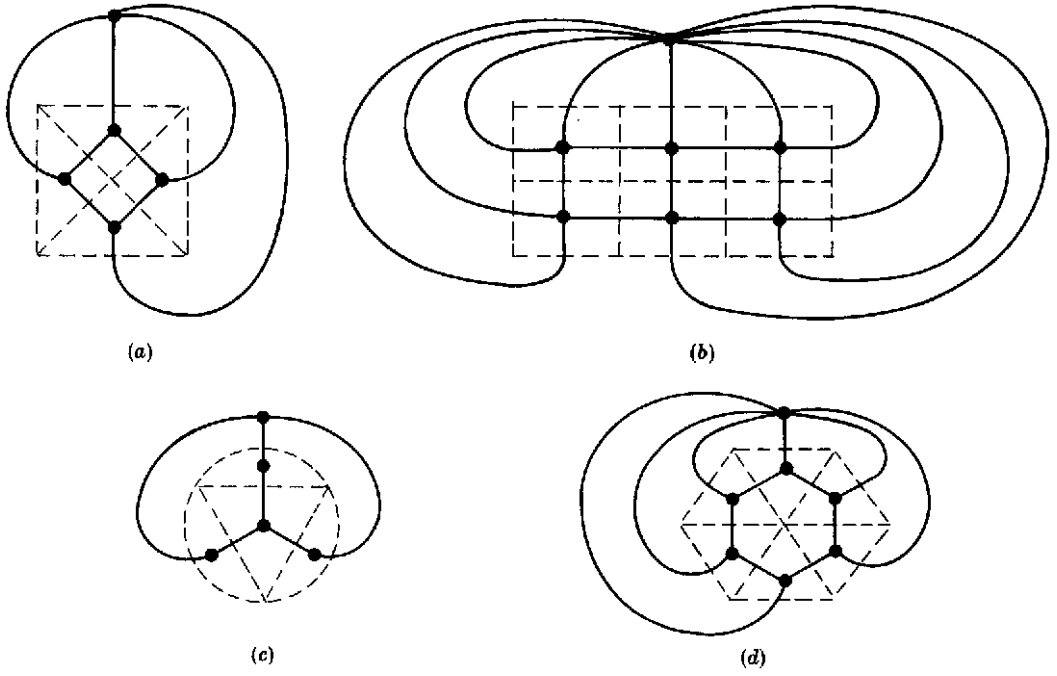



**Fig. 8-72**

**8.66.** (a) $n = 3$; (b) $n = 4$.

**8.67.** ($a$) $\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$; ($b$) $\begin{bmatrix} 0 & 1 & 2 & 0 \\ 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$; ($c$) $\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$

**8.68.** See Fig. 8-73.

**8.69.** Let $M$ and $N$ be the two disjoint sets of vertices determining the bipartite graph $G$. Order the vertioes in $M$ first and then those in $N$.

**8.70.** (a) $B, F, A, D, E, C$.

(b) $G = [A{:}B;\ B{:}A, C, D, E;\ C{:}F;\ D{:}B;\ E{:}B;\ F{:}C]$.

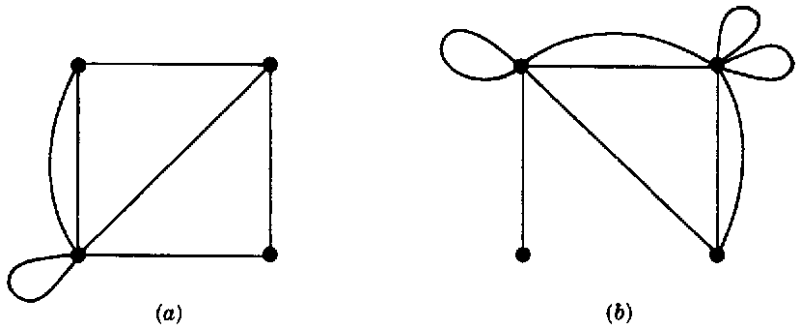



**Fig. 8-73**

**8.71.** (a) Each vertex is adjacent to the other four vertices.

(b) $G = [A{:}B, D, F;\ B{:}A, C, E;\ C{:}B, D, F;\ D{:}A, C, E;\ E{:}B, D, F;\ F{:}A, C, E]$.

(c) $G = [A{:}B, D;\ B{:}A, C, E;\ C{:}B, D;\ D{:}A, C, E;\ E{:}B, D]$.

**8.72.** See Fig. 8-74.

Vertex file

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| VERTEX | *A* | *B* | *C* | *D* | *E* | *F* | | |
| PTR | 1 | 2 | 9 | 14 | 8 | 12 | | |

Edge file

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NUMBER | 22 | 22 | 33 | 33 | 44 | 44 | 55 | 55 | 66 | 66 | 77 | 77 | 88 | 88 | |
| ADJ | 2 | 1 | 6 | 5 | 4 | 2 | 5 | 2 | 6 | 3 | 6 | 2 | 4 | 1 | |
| NEXT | 13 | 5 | 0 | 0 | 7 | 0 | 11 | 3 | 0 | 4 | 0 | 10 | 0 | 6 | |

**Fig. 8-74**

**8.73.** (a) $|ACBEDFA| = 20$ or $|ACBEFDA| = 21$; (b) $|BCFEDAB| = 21$ or $|BCDEFAB| = 20$

**8.74.** (a) $|ABCDEA| = 775$, $|ABCEDA| = 725$, $|ABDCEA| = 1100$, $|ABDECA| = 900$, $|ABECDA| = 1050$, $|ABEDCA| = 900$, $|ACBDEA| = 825$, $|ACBEDA| = 775$, $|ACDBEA| = 1150$, $|ACEBDA| = 1100$, $|ADBCEA| = 975$;

(b) $|ABCEDA| = 725$

**8.75.** (a) $G = [A{:}BJ;\ B{:}AJKL;\ C{:}DLM;\ D{:}CM;\ J{:}AB;\ K{:}BL;\ L{:}BCK;\ M{:}CD]$

(b) [STACK : $C, MLD, DL, L, KB, B, J, A$], $CMDLKBJA$

(c) [STACK : $K, LB, CB, MDB, DB, B, JA, A$], $KLCMDBJA$

**8.76.** (a) [QUEUE : $C, MLD, ML, L, KB, JAK, JA, J$], $CDMLBKAJ$

(b) [QUEUE : $K, LB, JAL, CJA, CJ, C, MD, M$], $KBLAJCDM$

**8.77.** (a) $G = [A{:}BMJKL;\ B{:}ACDJL;\ C{:}BJ;\ D{:}BKM;\ J{:}ABCM;\ K{:}ADL;\ L{:}ABKM;\ M{:}ADJL]$

(b) [STACK : $C, JB, MBA, LDAB, KBAD, DAB, AB, B$], $CJMLKDAB$

(c) [STACK : $K, LDA, MBAD, JDAB, CBAD, BAD, AD, D$], $KLMJCBAD$

**8.78.** (a) [QUEUE : $C$, $JB$, $LDAJ$, $MLDA$, $KMLD$, $KML$, $KM$, $K$], $CBJADLMK$

(b) [QUEUE : $K$, $LDA$, $JMBLD$, $JMBL$, $CJMB$, $CJM$, $CJ$, $C$], $KADLBMJC$

**8.79.** (a) $G = [A{:}BLM;\ B{:}ACLM;\ C{:}BDJ;\ D{:}CK;\ J{:}CK;\ K{:}DJL;\ L{:}ABKM;\ M{:}ABL]$

(b) [STACK : $C$, $JDB$, $KDB$, $LDB$, $MBAD$, $BAD$, $AD$, $D$], $CJKLMBAD$

(c) [STACK : $K$, $LJD$, $MBAJD$, $BAJD$, $CAJD$, $JDA$, $DA$, $A$], $KLMBCJDA$

**8.80.** (a) [QUEUE : $C$, $JDB$, $MLAJD$, $KMLAJ$, $KMLA$, $KML$, $KM$, $K$], $CBDJALMK$

(b) [QUEUE : $K$, $LJD$, $CLJ$, $CL$, $MBAC$, $MBA$, $MB$, $M$], $KDJLCABM$