CHAPTER 9

Directed Graphs

9.1 INTRODUCTION

Directed graphs are graphs in which the edges are one-way. Such graphs are frequently more useful in various dynamic systems such as digital computers or flow systems. However, this added feature makes it more difficult to determine certain properties of the graph. That is, processing such graphs may be similar to traveling in a city with many one-way streets.

This chapter gives the basic definitions and properties of directed graphs. Many of the definitions will be similar to those in the preceding chapter on (non-directed) graphs. However, for pedagogical reasons, this chapter will be mainly independent from the preceding chapter.

9.2 DIRECTED GRAPHS

A *directed graph* G or *digraph* (or simply *graph*) consists of two things:

(i) A set V whose elements are called *vertices*, *nodes*, or *points*.

(ii) A set E of *ordered* pairs (u, v) of vertices called *arcs* or *directed edges* or simply *edges*.

We will write $G(V, E)$ when we want to emphasize the two parts of G. We will also write $V(G)$ and $E(G)$ to denote, respectively, the set of vertices and the set of edges of a graph G. (If it is not explicitly stated, the context usually determines whether or not a graph G is a directed graph.)

Suppose $e = (u, v)$ is a directed edge in a digraph G. Then the following terminology is used:

(a) e *begins* at u and *ends* at v.
(b) u is the *origin* or *initial point* of e, and v is the *destination* or *terminal point* of e.
(c) v is a *successor* of u.
(d) u is *adjacent to* v, and v is *adjacent from* u.

If $u = v$, then e is called a *loop*.

The set of all successors of a vertex u is important; it is denoted and formally defined by

$$\text{succ}(u) = \{v \in V \mid \text{there exists an edge } (u, v) \in E\}$$

It is called the *successor list* or *adjacency list* of u.

A *picture* of a directed graph G is a representation of G in the plane. That is, each vertex u of G is represented by a dot (or small circle), and each (directed) edge $e = (u, v)$ is represented by an arrow or directed curve from the initial point u of e to the terminal point v. One usually presents a digraph G by its picture rather than explicitly listing its vertices and edges.

If the edges and/or vertices of a directed graph G are labeled with some type of data, then G is called a *labeled directed graph.*

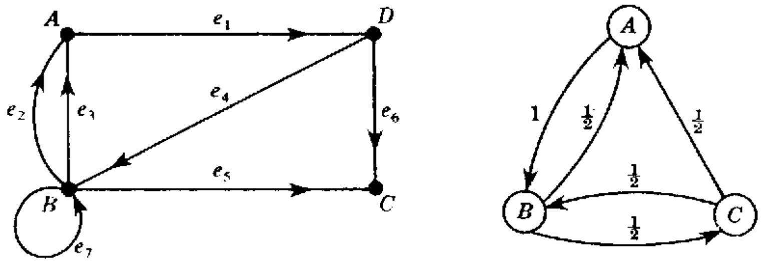
A directed graph *(V, E)* is said to be *finite* if its set V of vertices and its set E of edges are finite.

EXAMPLE 9.1

(a) Consider the directed graph G pictured in Fig. 9-1(a). It consists of four vertices, A, B, C, D, that is, $V(G) = \{A, B, C, D\}$ and the seven following edges:

$$E(G) = \{e_1, e_2, \ldots, e_7\} = \{(A, D), (B, A), (B, A), (D, B), (B, C), (D, C), (B, B)\}$$

The edges e_2 and e_3 are said to be *parallel* since they both begin at B and end at A. The edge e_7 is a *loop* since it begins and ends at B.



(a) (b)

Fig. 9-1

(b) Suppose three boys, A, B, C, are throwing a ball to each other such that A always throws the ball to B, but B and C are just as likely to throw the ball to A as they are to each other. This dynamic system is pictured in Fig. 9-1(b) where edges are labeled with the respective probabilities, that is, A throws the ball to B with probability 1, B throws the ball to A and C each with probability 1/2, and C throws the ball to A and B each with probability 1/2.

Subgraphs

Let $G = G(V, E)$ be a directed graph, and let V' be a subset of the set V of vertices of G. Suppose E' is a subset of E such that the endpoints of the edges in E' belong to V'. Then $H(V', E')$ is a directed graph, and it is called a *subgraph* of G. In particular, if E' contains all the edges in E whose endpoints belong to V', then $H(V', E')$ is called the subgraph of G *generated* or *determined* by V'. For example, for the graph $G = G(V, E)$ in Fig. 9-1(a), $H(V', E')$ is the subgraph of G determine by the vertex set V' where

$$V' = \{B, C, D\} \quad \text{and} \quad E' = \left\{e_4, e_5, e_6, e_7\right\} = \{(D, B), (B, C), (D, C), (B, B)\}$$

9.3 BASIC DEFINITIONS

This section discusses the questions of degrees of vertices, paths, and connectivity in a directed graph.

Degrees

Suppose G is a directed graph. The *outdegree* of a vertex v of G, written outdeg(v), is the number of edges beginning at v, and the *indegree* of v, written indeg(v), is the number of edges ending at v. Since each edge begins and ends at a vertex we immediately obtain the following theorem.

Theorem 9.1: The sum of the outdegrees of the vertices of a digraph G equals the sum of the indegrees of the vertices, which equals the number of edges in G.

A vertex v with zero indegree is called a *source*, and a vertex v with zero outdegree is called a *sink*.

EXAMPLE 9.2 Consider the graph G in Fig. 9-1(a). We have:

$$\begin{array}{llll} \text{outdeg}\,(A) = 1, & \text{outdeg}\,(B) = 4, & \text{outdeg}\,(C) = 0, & \text{outdeg}\,(D) = 2, \\ \text{indeg}\,(A) = 2, & \text{indeg}\,(B) = 2, & \text{indeg}\,(C) = 2, & \text{indeg}\,(D) = 1. \end{array}$$

As expected, the sum of the outdegrees equals the sum of the indegrees, which equals the number 7 of edges. The vertex C is a sink since no edge begins at C. The graph has no sources.

Paths

Let G be a directed graph. The concepts of path, simple path, trail, and cycle carry over from nondirected graphs to the directed graph G except that the directions of the edges must agree with the direction of the path. Specifically:

(i) A (*directed*) *path* P in G is an alternating sequence of vertices and directed edges, say,

$$P = \left(v_0,\ e_1,\ v_1,\ e_2,\ v_2, \ldots, e_n,\ v_n\right)$$

such that each edge e_i begins at v_{i-1} and ends at v_i. If there is no ambiguity, we denote P by its sequence of vertices or its sequence of edges.

(ii) The *length* of the path P is n, its number of edges.

(iii) A *simple path* is a path with distinct vertices. A *trail* is a path with distinct edges.

(iv) A *closed path* has the same first and last vertices.

(v) A *spanning path* contains all the vertices of G.

(vi) A *cycle* (or *circuit*) is a closed path with distinct vertices (except the first and last).

(vii) A *semipath* is the same as a path except the edge e_i may begin at v_{i-1} or v_i and end at the other vertex. *Semitrails* and *semisimple paths* are analogously defined.

A vertex v is *reachable* from a vertex u if there is a path from u to v. If v is reachable from u, then (by eliminating redundant edges) there must be a simple path from u to v.

EXAMPLE 9.3 Consider the graph G in Fig. 9-1(a).

(a) The sequence $P_1 = (D, C, B, A)$ is a semipath but not a path since (C, B) is not an edge; that is, the direction of $e_5 = (C, B)$ does not agree with the direction of P_1.

(b) The sequence $P_2 = (D, B, A)$ is a path from D to A since *(D, B)* and *(B, A)* are edges. Thus A is reachable from D.

Connectivity

There are three types of connectivity in a directed graph G:

(i) G is *strongly connected* or *strong* if, for any pair of vertices u and v in G, there is a path from u to v and a path from v to u, that is, each is reachable from the other.

(ii) G is *unilaterally connected* or *unilateral* if, for any pair of vertices u and v in G, there is a path from u to v or a path from v to u, that is, one of them is reachable from the other.

(iii) G is *weakly connected* or *weak* if there is a semipath between any pair of vertices u and v in G.

Let G' be the (nondirected) graph obtained from a directed graph G by allowing all edges in G to be nondirected. Clearly, G is weakly connected if and only if the graph G' is connected.

Observe that strongly connected implies unilaterally connected which implies weakly connected. We say that G is *strictly unilateral* if it is unilateral but not strong, and we say that G is *strictly weak* if it is weak but not unilateral.

Connectivity can be characterized in terms of spanning paths as follows:

Theorem 9.2: Let G be a finite directed graph. Then:

(i) G is strong if and only if G has a closed spanning path.

(ii) G is unilateral if and only if G has a spanning path.

(iii) G is weak if and only if G has a spanning semipath.

EXAMPLE 9.4 Consider the graph G in Fig. 9-1(a). It is weakly connected since the underlying nondirected graph is connected. There is no path from C to any other vertex, that is, C is a sink, so G is not strongly connected. However, $P = (B, A, D, C)$ is a spanning path, so G is unilaterally connected.

Graphs with sources and sinks appear in many applications (such as flow diagrams and networks). A sufficient condition for such vertices to exist follows.

Theorem 9.3: Suppose a finite directed graph G is cycle-free, that is, contains no (directed) cycles. Then G contains a source and a sink.

Proof: Let $P = (v_0, v_1, \ldots, v_n)$ be a simple path of maximum length, which exists since G is finite. Then the last vertex v_n is a sink; otherwise an edge $(v_{n,}\ u)$ will either extend P or form a cycle if $u = v_i$, for some i. Similarly, the first vertex v_0 is a source.

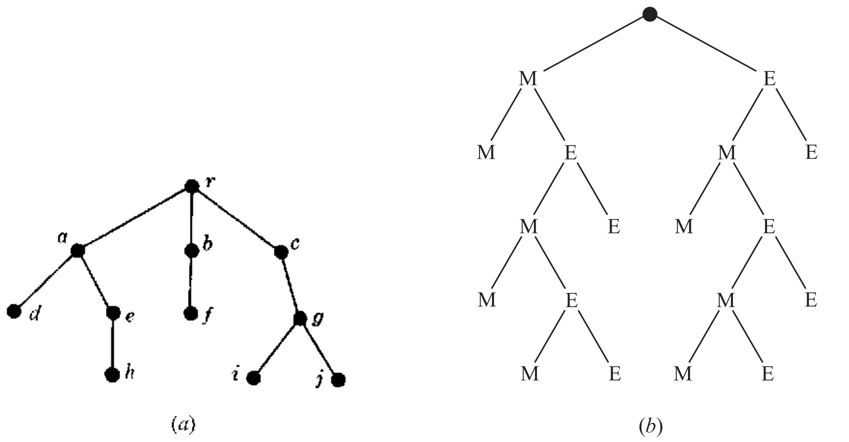
9.4 ROOTED TREES

Recall that a tree graph is a connected cycle-free graph, that is, a connected graph without any cycles. A *rooted tree* T is a tree graph with a designated vertex r called the *root* of the tree. Since there is a unique simple path from the root r to any other vertex v in T, this determines a direction to the edges of T. Thus T may be viewed as a directed graph. We note that any tree may be made into a rooted tree by simply selecting one of the vertices as the root.

Consider a rooted tree T with root r. The length of the path from the root r to any vertex v is called the *level* (or *depth*) of v, and the maximum vertex level is called the *depth* of the tree. Those vertices with degree 1, other than the root r, are called the *leaves* of T, and a directed path from a vertex to a leaf is called a *branch*.

One usually draws a picture of a rooted tree T with the root at the top of the tree. Figure 9-2(a) shows a rooted tree T with root r and 10 other vertices. The tree has five leaves, d, f, h, i, and j. Observe that: $level(a) = 1$, $level(f) = 2$, $level(j) = 3$. Furthermore, the depth of the tree is 3.

The fact that a rooted tree T gives a direction to the edges means that we can give a precedence relationship between the vertices. Specifically, we will say that a vertex u *precedes* a vertex v or that v *follows* u if there is



(*a*) (*b*)

Fig. 9-2

a (directed) path from v to u. In particular, we say that v *immediately follows* u if (u, v) is an edge, that is, if v follows u and v is adjacent to u. We note that every vertex v, other than the root, immediately follows a unique vertex, but that v can be immediately followed by more than one vertex. For example, in Fig. 9-2(*a*), the vertex j follows c but immediately follows g. Also, both i and j immediately follow g.

A rooted tree T is also a useful device to enumerate all the logical possibilities of a sequence of events where each event can occur in a finite number of ways. This is illustrated in the following example.

EXAMPLE 9.5 Suppose Marc and Erik are playing a tennis tournament such that the first person to win two games in a row or who wins a total of three games wins the tournament. Find the number of ways the tournament can proceed.

The rooted tree in Fig. 9-2(*b*) shows the various ways that the tournament could proceed. There are 10 leaves which correspond to the 10 ways that the tournament can occur:

MM, **MEMM**, **MEMEM**, **MEMEE**, **MEE**, **EMM**, **EMEMM**, **EMEME**, **EMEE**, **EE**

Specifically, the path from the root to the leaf describes who won which games in the particular tournament.

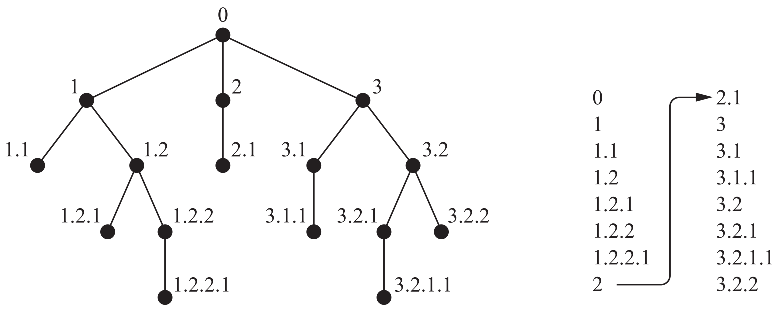
Ordered Rooted Trees

Consider a rooted tree T in which the edges leaving each vertex are ordered. Then we have the concept of an *ordered rooted tree*. One can systematically label (or *address*) the vertices of such a tree as follows: We first assign 0 to the root r. We next assign $1, 2, 3, \ldots$ to the vertices immediately following r according as the edges were ordered. We then label the remaining vertices in the following way. If a is the label of a vertex v, then $a.1, a.2, \ldots$ are assigned to the vertices immediately following v according as the edges were ordered. We illustrate this address system in Fig. 9-3(*a*), where edges are pictured from left to right according to their order. Observe that the number of decimal points in any label is one less than the level of the vertex. We will refer to this labeling system as the *universal address system* for an ordered rooted tree.

The universal address system gives us an important way of linearly describing (or storing) an ordered rooted tree. Specifically, given addresses a and b, we let $a < b$ if $b = a.c$, (that is, a is an *initial segment* of b), or if there exist positive integers m and n with $m < n$ such that

$$a = \text{r.m.s} \quad \text{and} \quad b = \text{r.n.t}$$

This order is called the *lexicographic order* since it is similar to the way words are arranged in a dictionary. For example, the addresses in Fig. 9-3(*a*) are linearly ordered as pictured in Fig. 9-3(*b*). This lexicographic order is



(*a*) (*b*)

Fig. 9-3

identical to the order obtained by moving down the leftmost branch of the tree, then the next branch to the right, then the second branch to the right, and so on.

9.5 SEQUENTIAL REPRESENTATION OF DIRECTED GRAPHS

There are two main ways of maintaining a directed 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* of G, is by means of linked lists of neighbors. This section covers the first representation. The linked representation will be covered in Section 9.7.

Suppose a graph G has m vertices (nodes) and n edges. We say G is *dense* if $m = O(n^2)$ and *sparse* if $m = O(n)$ or even if $m = O(n \log n)$. The matrix representation of G is usually used when G is dense, and linked lists are usually used when G is sparse. 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 edges (pairs of vertices).

Remark: In order to avoid special cases of our results, we assume, unless otherwise stated, that $m > 1$ where m is the number of vertices in our graph G. Therefore, G is not connected if G has no edges.

Digraphs and Relations, Adjacency Matrix

Let $G(V, E)$ be a *simple* directed graph, that is, a graph without parallel edges. Then E is simply a subset of $V \times V$, and hence E is a relation on V. Conversely, if R is a relation on a set V, then $G(V, R)$ is a simple directed graph. Thus the concepts of relations on a set and simple directed graphs are one and the same. In fact, in Chapter 2, we already introduced the directed graph corresponding to a relation on a set.

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

$$a_{ij} = \begin{cases} 1 & \text{if there is an edge } (v_i, v_j) \\ 0 & \text{otherwise} \end{cases}$$

Such a matrix A, which contains entries of only 0 or 1, is called a *bit matrix* or a *Boolean matrix*. (Although the adjacency matrix of an undirected graph is symmetric, this is not true here for a directed graph.)

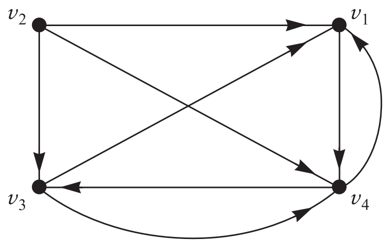
The adjacency matrix A of the graph G does depend on the ordering of the vertices of G. However, the matrices resulting from two different orderings are closely related in that one can be obtained from the other by simply interchanging rows and columns. Unless otherwise stated, we assume that the vertices of our matrix have a fixed ordering.

Remark: The adjacency matrix $A = [a_{ij}]$ may be extended to directed graphs with parallel edges by setting:

$$a_{ij} = \text{the number of edges beginning at } v_i \text{ and ending at } v_j$$

Then the entries of A will be nonnegative integers. Conversely, every $m \times m$ matrix A with nonnegative integer entries uniquely defines a directed graph with m vertices.

EXAMPLE 9.6 Let G be the directed graph in Fig. 9-4(a) with vertices v_1, v_2, v_3, v_4. Then the adjacency matrix A of G appears in Fig. 9-4(b). Note that the number of 1's in A is equal to the number (eight) of edges.



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

(a) (b)

Fig. 9-4

Consider the powers A, A^2, A^3,... of the adjacency matrix $A = [a_{ij}]$ of a graph G. Let

$$a_K(i, j) = \text{the } ij \text{ entry in the matrix } A^K$$

Note that $a_1(i, j) = a_{ij}$ gives the number of paths of length 1 from vertex v_i to vertex v_j. One can show that $a_2(i, j)$ gives the number of paths of length 2 from v_i to v_j. In fact, we prove in Problem 9.17 the following general result.

Proposition 9.4: Let A be the adjacency matrix of a graph G. Then $a_K(i, j)$, the ij entry in the matrix A^K, gives the number of paths of length K from v_i to v_j.

EXAMPLE 9.7 Consider again the graph G and its adjacency matrix A appearing in Fig. 9-4. The powers A^2, A^3, and A^4 of A follow:

$$A^2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 2 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 2 \end{bmatrix}, \quad A^3 = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 3 & 0 & 2 & 3 \\ 2 & 0 & 1 & 2 \\ 2 & 0 & 2 & 1 \end{bmatrix}, \quad A^4 = \begin{bmatrix} 2 & 0 & 2 & 1 \\ 5 & 0 & 3 & 5 \\ 3 & 0 & 2 & 3 \\ 3 & 0 & 1 & 4 \end{bmatrix}$$

Observe that a_2 (4, 1) = 1, so there is a path of length 2 from v_4 to v_1. Also, a_3(2, 3) = 2, so there are two paths of length 3 from v_2 to v_3; and a_4(2, 4) = 5, so there are five paths of length 4 from v_2 to v_4.

Remark: Let A be the adjacency matrix of a graph G, and let B_r be the matrix defined by:

$$B_r = A + A^2 + A^3 + \cdots + A^r$$

Then the ij entry of the matrix B_r gives the number of paths of length r or less from vertex v_i to vertex v_j.

Path Matrix

Let $G = G(V, E)$ be a simple directed graph with m vertices $v_1, v_2, \ldots, v_m$. The *path matrix* or *reachability matrix* of G is the m-square matrix $P = [p_{ij}]$ defined as follows:

$$p_{ij} = \begin{cases} 1 & \text{if there is a path from to } v_i \text{ to } v_j \\ 0 & \text{otherwise} \end{cases}$$

(The path matrix P may be viewed as the transitive closure of the relation E on V.)

Suppose now that there is a path from vertex v_i to vertex v_j in a graph G with m vertices. Then there must be a simple path from v_i to v_j when $v_i \neq v_j$, or there must be a cycle from v_i to v_j when $v_i = v_j$. Since G has m vertices, such a simple path must have length $m - 1$ or less, or such a cycle must have length m or less. This means that there is a nonzero ij entry in the matrix B_m (defined above) where A is the adjacency matrix of G. Accordingly, the path matrix P and B_m have the same nonzero entries. We state this result formally.

Proposition 9.5: Let A be the adjacency matrix of a graph G with m vertices. Then the path matrix P and B_m have the same nonzero entries where

$$B_m = A + A^2 + A^3 + \cdots + A^m$$

Recall that a directed graph G is said to be *strongly connected* if, for any pair of vertices u and v in G, there is a path from u to v and from v to u. Accordingly, G is strongly connected if and only if the path matrix P of G has no zero entries. This fact together with Proposition 9.5 gives the following result.

Proposition 9.6: Let A be the adjacency matrix of a graph G with m vertices. Then G is strongly connected if and only if B_m has no zero entries where

$$B_m = A + A^2 + A^3 + \cdots + A^m$$

EXAMPLE 9.8 Consider the graph G and its adjacency matrix A appearing in Fig. 9-4. Here G has $m = 4$ vertices. Adding the matrix A and matrices A^2, A^3, A^4 in Example 9.7, we obtain the following matrix B_4 and also path (reachability) matrix P by replacing the nonzero entries in B_4 by 1:

$$B_4 = \begin{bmatrix} 4 & 0 & 3 & 4 \\ 11 & 0 & 7 & 11 \\ 7 & 0 & 4 & 7 \\ 7 & 0 & 4 & 7 \end{bmatrix} \quad \text{and} \quad P = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Examining the matrix B_4 or P, we see zero entries; hence G is not strongly connected. In particular, we see that the vertex v_2 is not reachable from any of the other vertices.

Remark: The adjacency matrix A and the path matrix P of a graph G may be viewed as logical (Boolean) matrices where 0 represents "false" and 1 represents "true." Thus the logical operations of $\wedge$ (AND) and $\vee$ (OR) may be applied to the entries of A and P where these operations, used in the next section, are defined in Fig. 9-5.

$\wedge$	0	1
0	0	0
1	0	1

(a) AND.

$\vee$	0	1
0	0	1
1	1	1

(b) OR.

Fig. 9-5

Transitive Closure and the Path Matrix

Let R be a relation on a finite set V with m elements. As noted above, the relation R may be identified with the simple directed graph $G = G\ (V,\ R)$. We note that the composition relation $R^2 = R \times R$ consists of all pairs $(u,\ v)$ such that there is a path of length 2 from u to v. Similarly:

$$R^K = \{(u,\ v) |\ \text{there is a path of length } K \text{ from } u \text{ to } v\}.$$

The transitive closure R^* of the relation R on V may now be viewed as the set of ordered pairs (u, v) such that there is a path from u to v in the graph G. Furthermore, by the above discussion, we need only look at simple paths of length $m - 1$ or less and cycles of length m or less. Accordingly, we have the following result which characterizes the transitive closure R^* of R.

Theorem 9.7: Let R be a relation on a set V with m elements. Then:

(i) $R^* = R \cup R^2 \cup \ldots \cup R^m$ is the transitive closure of R.

(ii) The path matrix P of $G(V, R)$ is the adjacency matrix of $G'(V, R^*)$.

9.6 WARSHALL'S ALGORITHM, SHORTEST PATHS

Let G be a directed graph with m vertices, $v_1, v_2, \ldots, v_m$. Suppose we want to find the path matrix P of the graph G. Warshall gave an algorithm which is much more efficient than calculating the powers of the adjacency matrix A. This algorithm is defined in this section, and a similar algorithm is used to find shortest paths in G when G is weighted.

Warshall's Algorithm

First we define m-square Boolean matrices $P_0, P_1, \ldots, P_m$ where $P_k[i, j]$ denotes the ij entry of the matrix P_k:

$$P_k[i, j] = \begin{cases} 1 & \text{if there is a simple path from } v_i \text{ to } v_j \text{ which does not use any} \\ & \quad \text{other vertices except possibly } v_1, v_2, \ldots, v_{k,} \\ 0 & \text{otherwise.} \end{cases}$$

For example,

$$P_3[i, j] = 1 \quad \text{if there is a simple path from } v_i \text{ to } v_j \text{ which does not use any other vertices except possibly } v_1, v_2, v_3.$$

Observe that the first matrix $P_0 = A$, the adjacency matrix of G. Furthermore, since G has only m vertices, the last matrix $P_m = P$, the path matrix of G.

Warshall observed that $P_k[i, j] = 1$ can occur only if one of the following two cases occurs:

(1) There is a simple path from v_i to v_j which does not use any other vertices except possibly $v_1, v_2, \ldots, v_{k-1}$; hence

$$P_{k-1}[i, j] = 1$$

(2) There is a simple path from v_i to v_k and a simple path from v_k to v_j where each simple path does not use any other vertices except possibly $v_1, v_2, \ldots, v_{k-1}$; hence

$$P_{k-1}[i, k] = 1 \quad \text{and} \quad P_{k-1}[k, j] = 1$$

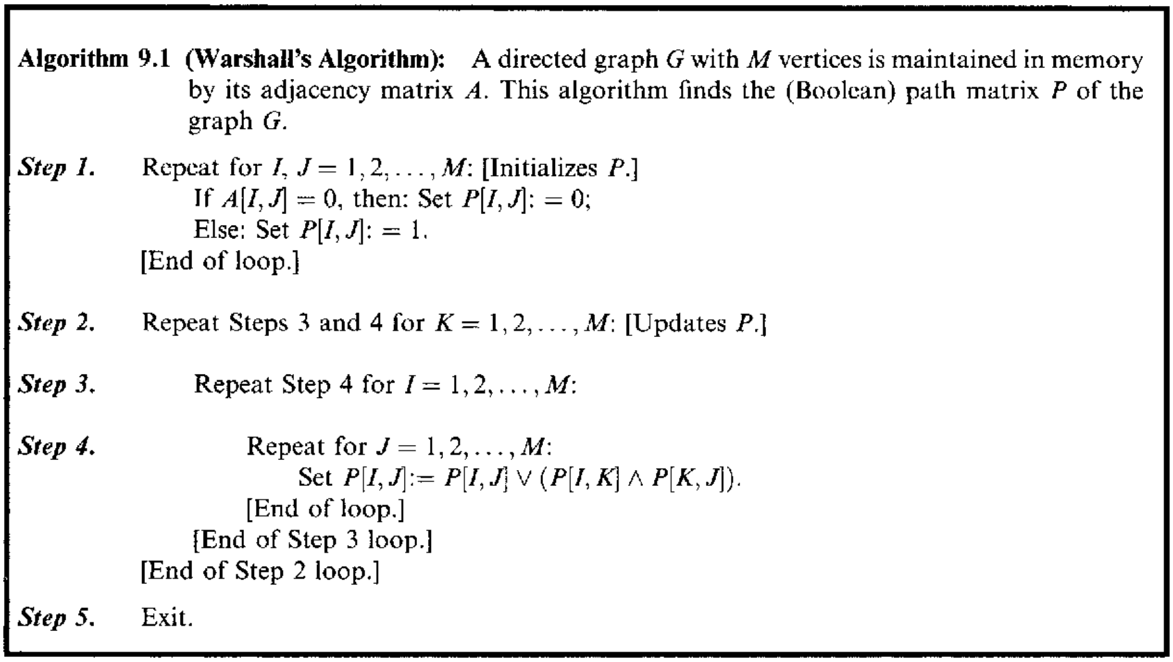
These two cases are pictured as follows:

$$(1)\ v_i \to \cdots \to v_j; \qquad (2)\ v_i \to \cdots \to v_k \to \cdots v_j$$

where $\to \cdots \to$ denotes part of a simple path which does not use any other vertices except possibly $v_1, v_2, \ldots, v_{k-1}$. Accordingly, the elements of P_k can be obtained by:

$$P_k[i, j] = P_{k-1}[i, j] \lor (P_{k-1}[i, k] \land P_{k-1}[k, j])$$

where we use the logical operations of a $\land$ (AND) and $\lor$ (OR). In other words we can obtain each entry in the matrix P_k by looking at only three entries in the matrix P_{k-1}. Warshall's algorithm appears in Fig. 9-6.



Algorithm 9.1 (Warshall's Algorithm): A directed graph G with M vertices is maintained in memory by its adjacency matrix A. This algorithm finds the (Boolean) path matrix P of the graph G.

```
Step 1.  Repeat for $I, J = 1, 2, \ldots, M$: [Initializes $P$.]
             If $A[I, J] = 0$, then: Set $P[I, J] := 0$;
             Else: Set $P[I, J] := 1$.
         [End of loop.]

Step 2.  Repeat Steps 3 and 4 for $K = 1, 2, \ldots, M$: [Updates $P$.]

Step 3.      Repeat Step 4 for $I = 1, 2, \ldots, M$:

Step 4.          Repeat for $J = 1, 2, \ldots, M$:
                     Set $P[I, J] := P[I, J] \vee (P[I, K] \wedge P[K, J])$.
                 [End of loop.]
             [End of Step 3 loop.]
         [End of Step 2 loop.]

Step 5.  Exit.
```

Fig. 9-6

Shortest-path Algorithm

Let G be a simple directed graph with m vertices, $v_1, v_2, \ldots, v_m$. Suppose G is weighted; that is, suppose each edge e of G is assigned a nonnegative number $w(e)$ called the *weight* or *length* of e. Then G may be maintained in memory by its weight matrix $W = [w_{ij}]$ defined as follows:

$$w_{ij} = \begin{cases} w(e) & \text{if there is an edge } e \text{ from } v_i \text{ to } v_j \\ 0 & \text{if there is no edge from } v_i \text{ to } v_j \end{cases}$$

The path matrix P tells us whether or not there are paths between the vertices. Now we want to find a matrix Q which tells us the lengths of the shortest paths between the vertices or, more exactly, a matrix $Q = [q_{ij}]$ where

$$q_{ij} = \text{length of the shortest path from } v_i \text{ to } v_j$$

Next we describe a modification of Warshall's algorithm which efficiently finds us the matrix Q.

Here we define a sequence of matrices $Q_0, Q_1, \ldots, Q_m$ (analogous to the above matrices $P_0, P_1, \ldots, P_m$) where $Q_k[i, j]$, the ij entry of Q_k, is defined as follows:

$Q_k[i, j]$ = the smaller of the length of the preceding path from v_i to v_j or the sum of the lengths of the preceding paths from v_i to v_k and from v_k to v_j.

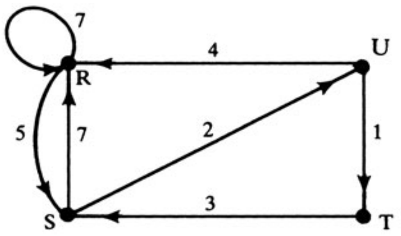
More exactly,

$$Q_k[i, j] = \text{MIN}(Q_{k-1}[i, j],\ Q_{k-1}[i, k] + Q_{k-1}[k, j])$$

The initial matrix Q_0 is the same as the weight matrix W except that each 0 in w is replaced by ∞ (or a very, very large number). The final matrix Q_m will be the desired matrix Q.

EXAMPLE 9.9 Figure 9-7 shows a weighted graph G and its weight matrix W where we assume that $v_1 = R$, $v_2 = S$, $v_3 = T$, $v_4 = U$.

Suppose we apply the modified Warshall's algorithm to our weighted graph G in Fig. 9-7. We will obtain the matrices Q_0, Q_1, Q_3, and Q_4 in Fig. 9-8. (To the right of each matrix Q_k in Fig. 9-8, we show the matrix of



$$W = \begin{bmatrix} 7 & 5 & 0 & 0 \\ 7 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 1 & 0 \end{bmatrix}$$

Fig. 9-7

paths which correspond to the lengths in the matrix Q_k.) The entries in the matrix Q_0 are the same as the weight matrix W except each 0 in W is replaced by ∞ (a very large number). We indicate how the circled entries are obtained:

$$\begin{aligned}
Q_1[4,2] &= \mathrm{MIN}\,(Q_0[4,2],\ \ Q_0[4,1] + Q_0[1,2]) = \mathrm{MIN}(\infty, 4+5) = 9 \\
Q_2[1,3] &= \mathrm{MIN}\,(Q_1[1,3],\ \ Q_1[1,2] + Q_1[2,3]) = \mathrm{MIN}(\infty, 5+\infty) = \infty \\
Q_3[4,2] &= \mathrm{MIN}\,(Q_2[4,2],\ \ Q_2[4,3] + Q_2[3,2]) = \mathrm{MIN}(9, 3+1) = 4 \\
Q_4[3,1] &= \mathrm{MIN}\,(Q_3[3,1],\ \ Q_3[3,4] + Q_3[4,1]) = \mathrm{MIN}(10, 5+4) = 9
\end{aligned}$$

The last matrix $Q_4 = Q$, the desired shortest-path matrix.

$$Q_0 = \begin{bmatrix} 7 & 5 & \infty & \infty \\ 7 & \infty & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & \infty & 1 & \infty \end{bmatrix} \qquad \begin{bmatrix} \mathrm{RR} & \mathrm{RS} & - & - \\ \mathrm{SR} & - & - & \mathrm{SU} \\ - & \mathrm{TS} & - & - \\ \mathrm{UR} & - & \mathrm{UT} & - \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} 7 & 5 & \infty & \infty \\ 7 & 12 & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & \enclose{circle}{9} & 1 & \infty \end{bmatrix} \qquad \begin{bmatrix} \mathrm{RR} & \mathrm{RS} & - & - \\ \mathrm{SR} & \mathrm{SRS} & - & \mathrm{SU} \\ - & \mathrm{TS} & - & - \\ \mathrm{UR} & \mathrm{URS} & \mathrm{UT} & - \end{bmatrix}$$

$$Q_2 = \begin{bmatrix} 7 & 5 & \enclose{circle}{\infty} & 7 \\ 7 & 12 & \infty & 2 \\ 10 & 3 & \infty & 5 \\ 4 & 9 & 1 & 11 \end{bmatrix} \qquad \begin{bmatrix} \mathrm{RR} & \mathrm{RS} & - & \mathrm{RSU} \\ \mathrm{SR} & \mathrm{SRS} & - & \mathrm{SU} \\ \mathrm{TSR} & \mathrm{TS} & - & \mathrm{TSU} \\ \mathrm{UR} & \mathrm{URS} & \mathrm{UT} & \mathrm{URS} \end{bmatrix}$$

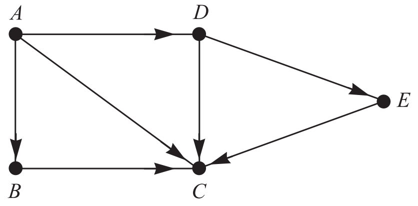
$$Q_3 = \begin{bmatrix} 7 & 5 & \infty & 7 \\ 7 & 12 & \infty & 2 \\ 10 & 3 & \infty & 5 \\ 4 & \enclose{circle}{4} & 1 & 6 \end{bmatrix} \qquad \begin{bmatrix} \mathrm{RR} & \mathrm{RS} & - & \mathrm{RSU} \\ \mathrm{SR} & \mathrm{SRS} & - & \mathrm{SU} \\ \mathrm{TSR} & \mathrm{TS} & - & \mathrm{TSU} \\ \mathrm{UR} & \mathrm{UTS} & \mathrm{UT} & \mathrm{UTSU} \end{bmatrix}$$

$$Q_4 = \begin{bmatrix} 7 & 5 & 8 & 7 \\ 7 & 11 & 3 & 2 \\ \enclose{circle}{9} & 3 & 6 & 5 \\ 4 & 4 & 1 & 6 \end{bmatrix} \qquad \begin{bmatrix} \mathrm{RR} & \mathrm{RS} & \mathrm{RSUT} & \mathrm{RSU} \\ \mathrm{SR} & \mathrm{SURS} & \mathrm{SUT} & \mathrm{SU} \\ \mathrm{TSUR} & \mathrm{TS} & \mathrm{TSUT} & \mathrm{TSU} \\ \mathrm{UR} & \mathrm{UTS} & \mathrm{UT} & \mathrm{UTSU} \end{bmatrix}$$

Fig. 9-8

9.7 LINKED REPRESENTATION OF DIRECTED GRAPHS

Let G be a directed graph with m vertices. Suppose the number of edges of G is $O(m)$ or even $O(m \log m)$, that is, suppose G is sparse. Then the adjacency matrix A of G 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.



(a) Graph G

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

(b) Adjacency lists of G

Fig. 9-9

Consider the directed graph G in Fig. 9-9(a). Observe that G may be equivalently defined by the table in Fig. 9-9(b), which shows each vertex in G followed by its *adjacency list*, also called its *successors* or *neighbors*. Here the symbol $\varnothing$ denotes an empty list. Observe that each edge of G corresponds to a unique vertex in an adjacency list and vice versa. Here G has seven edges and there are seven vertices in the adjacency lists. This table may also be presented in the following compact form where a colon ":" separates a vertex from its list of neighbors, and a semicolon ";" separates the different lists:

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

The *linked representation* of a directed graph G maintains G in memory by using linked lists for its adjacency lists. Specifically, the linked representation will normally contain two files (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

VERTEX	NEXT-V	PTR	▨

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, 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 G and also contains all the adjacency lists of G where each list is maintained in memory by a linked list. Each record of the Edge File will represent a unique edge in G and hence will correspond to a unique vertex in an adjacency list. The record will usually have the form

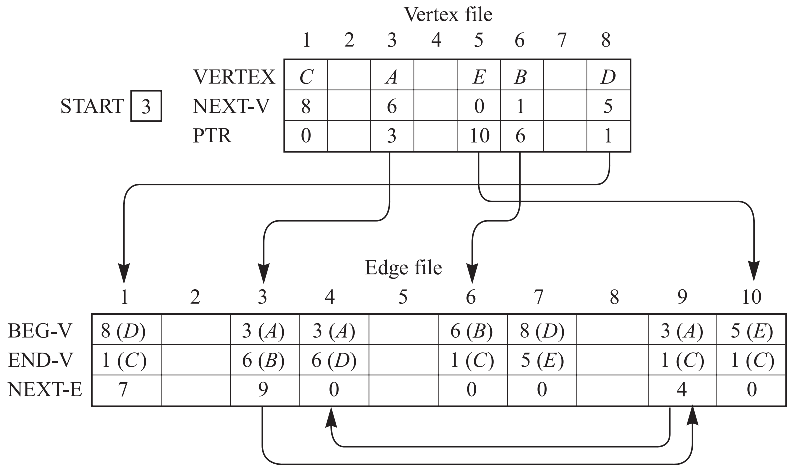
EDGE	BEG-V	END-V	NEXT-E	▨

Here:

(1) EDGE will be the name of the edge (if it has a name).
(2) BEG-V- points to location in the Vertex File of the initial (beginning) vertex of the edge.
(3) END-V points to the location in the Vertex File of the terminal (ending) vertex of the edge. The adjacency lists appear in this field.
(4) NEXT-E points to the location in the Edge File of the next vertex in the adjacency list.

We emphasize that the adjacency lists consist of terminal vertices and hence are maintained by the END-V field. The shaded area indicates that there may be other information in the record corresponding to the edge. We note that the order of the vertices in any adjacency list does depend on the order in which the edges (pairs of vertices) appear in the input.

Figure 9-10 shows how the graph G in Fig. 9-9(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.) The choice of eight locations for the Vertex File and 10 locations for the Edge File is arbitrary. The additional space in the files will be used if additional vertices or edges are inserted in the graph. Figure 9-10 also shows, with arrows, the adjacency list [B, C, D] of the vertex A.



Vertex file

START 3

	1	2	3	4	5	6	7	8
VERTEX	C		A		E	B		D
NEXT-V	8		6		0	1		5
PTR	0		3		10	6		1

Edge file

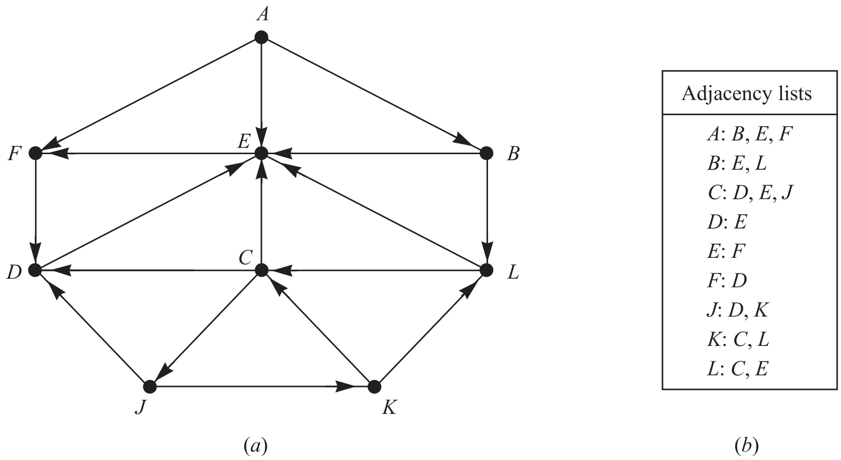
	1	2	3	4	5	6	7	8	9	10
BEG-V	8 (D)		3 (A)	3 (A)		6 (B)	8 (D)		3 (A)	5 (E)
END-V	1 (C)		6 (B)	6 (D)		1 (C)	5 (E)		1 (C)	1 (C)
NEXT-E	7		9	0		0	0		4	0

Fig. 9-10

9.8 GRAPH ALGORITHMS: DEPTH-FIRST AND BREADTH-FIRST SEARCHES

This section discusses two important graph algorithms for a given graph G. 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. 9-11.

Many applications of graphs require one to systematically examine the vertices and edges of a graph G. There are two standard ways that this is done. One way is called a *depth-first search* (DFS) and the other is called a *breadth-first search* (BFS). (These algorithms are essentially identical to the analogous algorithms for undirected graphs in Chapter 8.)



Adjacency lists
A: B, E, F
B: E, L
C: D, E, J
D: E
E: F
F: D
J: D, K
K: C, L
L: C, E

(a) (b)

Fig. 9-11

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 search 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 will be a QUEUE.

(a) **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 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 algorithm appears in Fig. 9-12.

```
Algorithm 9.2 (Depth-first Search): This algorithm executes a depth-first search on a directed
        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. 9-12

Algorithm 9.2 will process only those vertices which are reachable from a starting vertex 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 that is still in the ready state (STATE = 1). This new vertex, say B, can be obtained by traversing through the list of vertices.

Remark: The structure STACK in Algorithm 9.2 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 will represent a different edge.) If we do not delete the previous J in Step 5(b), then we obtain an alternative traversal algorithm.

EXAMPLE 9.10 Consider our test graph G in Fig. 9-11. Suppose we want to find and print all the vertices reachable from the vertex J (including J itself). One way to do this is to use a depth-first search of G starting at the vertex J.

Applying Algorithm 9.2, the vertices will be processed and printed in the following order:

$$J,\quad K,\quad L,\quad E,\quad F,\quad D,\quad C$$

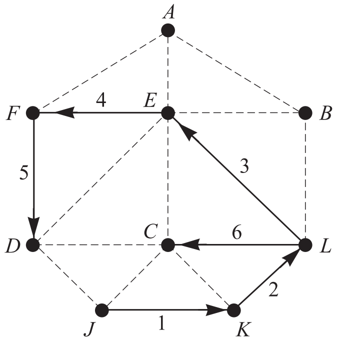
Specifically, Fig. 9-13(a) shows the sequence of waiting lists in STACK and the vertices being processed. (The slash / indicates that vertex is deleted from the waiting list.) We emphasize that each vertex, excluding J, comes from an adjacency list and hence it is the terminal vertex of a unique edge of the graph. We have indicated the edge by labeling the terminal vertex with the initial vertex of the edge as a subscript. For example,

$$D_j$$

means D is in the adjacency list of J, and hence D is the terminal vertex of an edge beginning at J. These edges form a rooted tree T with J as the root, which is pictured in Fig. 9-13(b). (The numbers indicate the order the edges are added to the tree.) This tree T spans the subgraph G' of G consisting of vertices reachable from J.

STACK	Vertex
J	J
K_J, D_J	K_J
L_K, C_K, D_J	L_K
E_L, C_L, $\cancel{C_K}$, D_J	E_L
F_E, C_L, D_J	F_E
D_F, C_L, $\cancel{D_J}$	D_F
C_L	C_L
∅	

(a)



(b)

Fig. 9-13

(b) **Breath-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 algorithm appears in Fig. 9-14.

Algorithm 9.3 will process only those vertices which are reachable from a starting vertex 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 that is still in the ready state (STATE $= 1$). This new vertex, say B, can be obtained by traversing through the list of vertices.

EXAMPLE 9.11 Consider our test graph G in Fig. 9-11. Suppose G represents the daily flights between cities, and suppose we want to fly from city A to city J with the minimum number of stops. That is, we want to find a shortest path P from A to J (where each edge has weight 1). One way to do this is to use a breadth-first search of G starting at the vertex A, and stop as soon as J is encountered.

Figure 9-15(a) shows the sequence of waiting lists in QUEUE and the vertices being processed up to the time vertex J is encountered. We then work backwards from J to obtain the following desired path which is pictured in Fig. 9-15(b):

$$J_C \leftarrow C_L \leftarrow L_B \leftarrow B_A \leftarrow A \quad \text{or} \quad A \rightarrow B \rightarrow L \rightarrow C \rightarrow J$$

Thus a flight from city A to city J will make three intermediate stops at B, L, and C. Note that the path does not include all of the vertices processed by the algorithm.

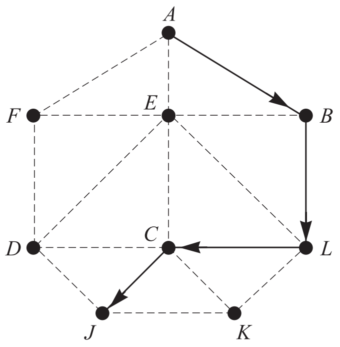
Algorithm 9.3 (Breadth-first Search): This algorithm executes a breadth-first search on a directed graph G beginning with a starting vertex A.

```
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 first 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. 9-14

QUEUE	Vertex
A	A
F_A, E_A, B_A	B_A
L_B, F_A, E_A	E_A
L_B, F_A	F_A
D_F, L_B	L_B
C_L, D_F	D_F
C_L	C_L
J_C	J_C

(*a*)



(*b*)

Fig. 9-15

9.9 DIRECTED CYCLE-FREE GRAPHS, TOPOLOGICAL SORT

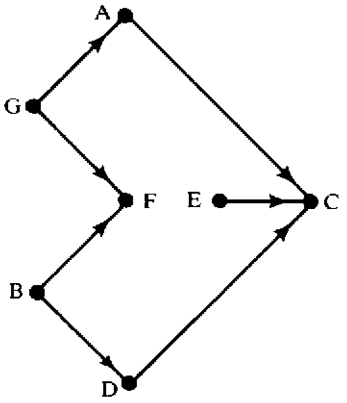
Let S be a directed graph with the following two properties:

(1) Each vertex v_i of S represents a task.

(2) Each (directed) edge $(u,\ v)$ of S means that task u must be completed before beginning task v.

We note that such a graph S cannot contain a cycle, such as $P\ =\ (u, v, w, u)$, since, otherwise, we would have to complete u before beginning v, complete v before beginning w, and complete w before beginning u. That is, we cannot begin any of the three tasks in the cycle.

Such a graph S, which represents tasks and a prerequisite relation and which cannot have any cycles, is said to be *cycle-free* or *acyclic*. A directed acyclic (cycle-free) graph is called a *dag* for short. Figure 9-16 is an example of such a graph.



(a)

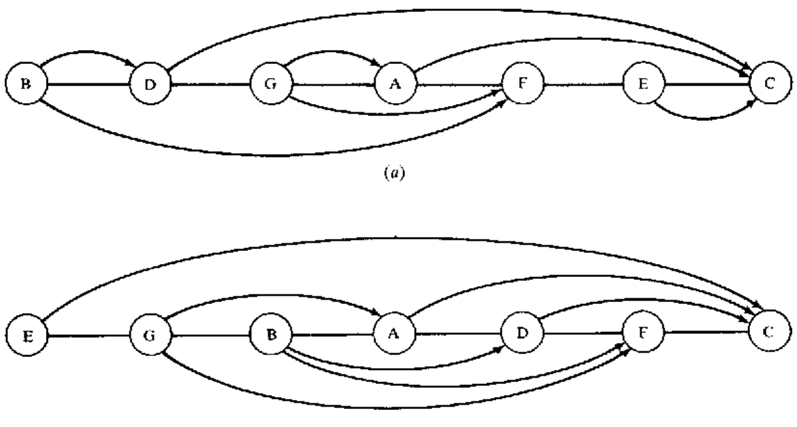
Adjacency lists	
A:	C
B:	D, F
C:	
D:	C
E:	C
F:	
G:	A, F

(b)

Fig. 9-16

A fundamental operation on a dag S is to process the vertices one after the other so that the vertex u is always processed before vertex v whenever (u, v) is an edge. Such a linear ordering T of the vertices of S, which may not be unique, is called a *topological sort.*

Figure 9-17 shows two topological sorts of the graph S in Fig. 9-16. We have included the edges of S in Fig. 9-17 to show that they agree with the direction of the linear ordering.



(a)

(b)

Fig. 9-17 Two topological sorts

The following is the main theoretical result in this section.

Theorem 9.8: Let S be a finite directed cycle-free graph. Then there exists a topological sort T of the graph S.

Note that the theorem states only that a topological sort exists. We now give an algorithm which will find a topological sort. The main idea of the algorithm is that any vertex (node) N with zero indegree may be chosen as the first element in the sort T. The algorithm essentially repeats the following two steps until S is empty:

(1) Find a vertex N with zero indegree.
(2) Delete N and its edges from the graph S.

We use an auxiliary QUEUE to temporarily hold all vertices with zero degree. The algorithm appears in Fig. 9-18.

```
Algorithm 9.4: The algorithm finds a topological sort $T$ of a directed cycle-free graph $S$.
Step 1. Find the indegree INDEG($N$) of each vertex $N$ of $S$.
Step 2. Insert in QUEUE all vertices with zero degree.
Step 3. Repeat Steps 4 and 5 until QUEUE is empty.
Step 4.         Remove and process the front vertex $N$ of QUEUE.
Step 5.         Repeat for each neighbor $M$ of the vertex $N$.
                (a) Set INDEG($M$) : = INDEG($M$) − 1.
                    [This deletes the edge from $N$ to $M$.]
                (b) If INDEG($M$) = 0, add $M$ to QUEUE.
                [End of loop.]
        [End of Step 3 loop.]
Step 6. Exit.
```

Fig. 9-18

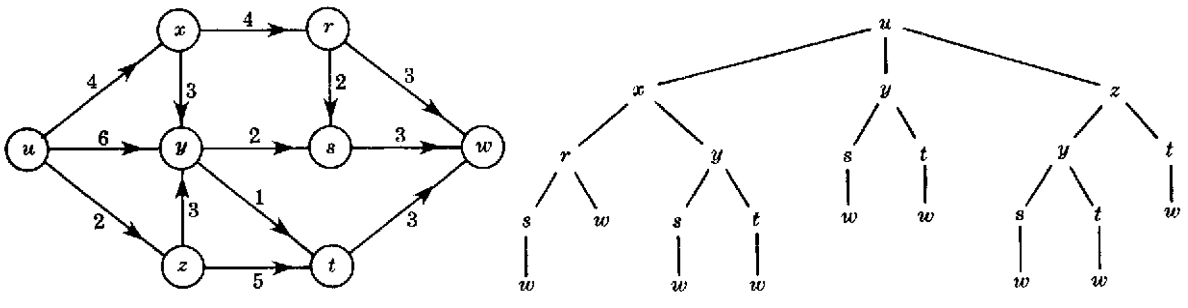
EXAMPLE 9.12 Suppose Algorithm 9.4 is applied to the graph S in Fig. 9-16. We obtain the following sequence of the elements of QUEUE and sequence of vertices being processed:

QUEUE	*GEB*	*DGE*	*DG*	*FAD*	*FA*	*CF*	*C*	∅
Vertex	*B*	*E*	*G*	*D*	*A*	*F*	*C*	

Thus the vertices are processed in the order: B, E, G, D, A, F.

9.10 PRUNING ALGORITHM FOR SHORTEST PATH

Let G be a weighted directed cycle-free graph. We seek the shortest path between two vertices, say u and w. We assume G is finite so at each step there is a finite number of moves. Since G is cycle-free, all paths between u and w can be given by a rooted tree with u as the root. Figure 9-19(b) enumerates all the paths between u and w in the graph in Fig. 9-19(a).



(a) (b)

Fig. 9-19

One way to find the shortest path between u and w is simply to compute the lengths of all the paths of the corresponding rooted tree. On the other hand, suppose two partial paths lead to an intermediate vertex v. From then on, we need only consider the shorter partial path; that is, we prune the tree at the vertex corresponding to the longer partial path. This pruning algorithm is described below.

Pruning Algorithm

This algorithm finds the shortest path between a vertex u and a vertex w in a weighted directed cycle-free graph G. The algorithm has the following properties:

(a) During the algorithm each vertex v' of G is assigned two things:

(1) A number $\ell(v')$ denoting the current minimal length of a path from u to v'.

(2) A path $p(v')$ from u to v' of length $\ell(v')$.

(b) Initially, we set $\ell(u) = 0$ and $p(u) = u$. Every other vertex v is initially assigned $\ell(v) = \infty$ and $p(v) = \varnothing$.

(c) Each step of the algorithm examines an edge $e = (v', v)$ from v' to v with, say, length k. We calculate $\ell(v') + k$.

(1) Suppose $\ell(v') + k < \ell(v)$. Then we have found a shorter path from u to v. Thus we update:

$$\ell(v) = \ell(v') + k \quad \text{and} \quad p(v) = p(v')v$$

(This is always true when $\ell(v) = \infty$, that is, when we first enter the vertex v.)

(2) Otherwise, we do not change $\ell(v)$ and $p(v)$.

If no other unexamined edges enter v, we will say that $p(v)$ has been determined.

(d) The algorithm ends when $p(w)$ has been determined.

Remark: The edge $e = (v', v)$ in (c) can only be chosen if v' has been previously visited, that is, if $p(v')$ is not empty. Furthermore, it is usually best to examine an edge which begins at a vertex v' whose path $p(v')$ has been determined.

EXAMPLE 9.13 We apply the pruning algorithm to the graph G in Fig. 9-19(a).

From u: The successive vertices are x, y, and z, which are all entered for the first time. Thus:

(1) set $\ell(x) = 4$, $p(x) = ux$.

(2) set $\ell(y) = 6$, $p(y) = uy$.

(3) set $\ell(z) = 2$, $p(z) = uz$.

Note that $p(x)$ and $p(z)$ have been determined.

From x: The successive vertices are r, entered for the first time, and y. Thus:

(1) Set $\ell(r) = 4 + 4 = 8$ and $p(r) = p(x)r = uxr$.

(2) We calculate:

$$\ell(x) + k = 4 + 3 = 7 \quad \text{which is not less than} \quad \ell(y) = 6.$$

Thus we leave $\ell(y)$ and $p(y)$ alone.

Note that $p(r)$ has been determined.

From z: The successive vertices are t, entered for the first time, and y. Thus:

(1) Set $\ell(t) = \ell(z) + k = 2 + 5 = 7$ and $p(t) = p(z)t = urt$.

(2) We calculate:

$$\ell(z) + k = 2 + 3 = 5 \quad \text{which is less than} \quad \ell(y) = 6.$$

We have found a shorter path to y, and so we update $\ell(y)$ and $p(y)$; set:

$$\ell(y) = \ell(z) + k = 5 \quad \text{and} \quad p(y) = p(z)y = uzy$$

Now $p(y)$ has been determined.

From *y*: The successive vertices are s, entered for the first time, and t. Thus:

(1) Set $\ell(s) = \ell(y) + k = 5 + 2 = 7$ and $p(s) = p(y)s = uzys$.

(2) We calculate:

$$\ell(y) + k = 5 + 1 = 6 \quad \text{which is less than} \quad \ell(t) = 7.$$

Thus we change $\ell(t)$ and $p(t)$ to read:

$$\ell(t) = \ell(y) + 1 = 6 \quad \text{and} \quad p(t) = p(y)t = uzyt.$$

Now $p(t)$ has been determined.

From *r*: The successive vertices are w, entered for the first time, and s. Thus:

(1) Set $\ell(w) = \ell(r) + 3 = 11$ and $p(w) = p(r)w = uxrw$.

(2) We calculate:

$$\ell(r) + k = 8 + 2 = 10 \quad \text{which is less than} \quad \ell(s) = 7.$$

Thus we leave $\ell(s)$ and $p(s)$ alone.

Note that $p(s)$ has been determined.

From *s*: The successive vertex is w. We calculate:

$$\ell(s) + k = 7 + 3 = 10 \quad \text{which is less than} \quad \ell(w) = 11.$$

Thus we change, $\ell(w)$ and $p(w)$ to read:

$$\ell(w) = \ell(s) + 3 = 10 \quad \text{and} \quad p(w) = p(s)w = uzysw.$$

From *t*: The successive vertex is w. We calculate:

$$\ell(t) + k = 6 + 3 = 9 \quad \text{which is less than} \quad \ell(w) = 10.$$

Thus we undate $\ell(w)$ and $p(w)$ as follows:

$$\ell(w) = \ell(t) + 3 = 9 \quad \text{and} \quad p(w) = p(t) = uzytw$$

Now $p(w)$ has been determined.

The algorithm is finished since $p(w)$ has been determined. Thus $p(w) = uzytw$ is the shortest path from u to w and $\ell(w) = 9$.

The edges which were examined in the above example form the rooted tree in Fig. 9-20. This is the tree in Fig. 9-19(*b*) which has been pruned at vertices belonging to longer partial paths. Observe that only 13 of the original 23 edges of the tree had to be examined.

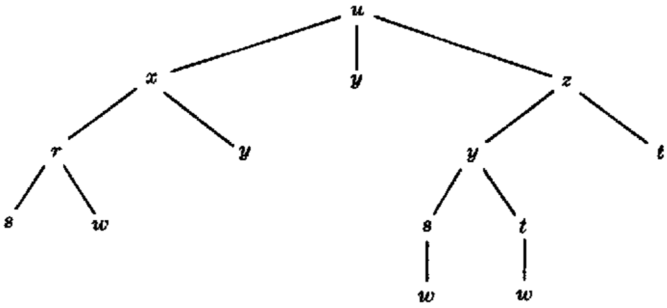


Fig. 9-20

Solved Problems

GRAPH TERMINOLOGY

9.1. Let G be the directed graph in Fig. 9-21(a).

(a) Describe G formally.
(b) Find all simple paths from X to Z.
(c) Find all simple paths from Y to Z.
(d) Find all cycles in G.
(e) Is G unilaterally connected?
(f) Is G strongly connected?

(a) The vertex set V has four vertices and the edge set E has seven (directed) edges as follows:

$$V = \{X, Y, Z, W\} \quad \text{and} \quad E = \{(X, Y), (X, Z), (X, W), (Y, W), (Z, Y), (Z, W), (W, Z)\}$$

(b) There are three simple paths from X to Z, which are (X, Z), (X, W, Z), and (X, Y, W, Z).

(c) There is only one simple path from Y to Z, which is (Y, W, Z).

(d) There is only one cycle in G, which is (Y, W, Z, Y).

(e) G is unilaterally connected since (X, Y, W, Z) is a spanning path.

(f) G is not strongly connected since there is no closed spanning path.

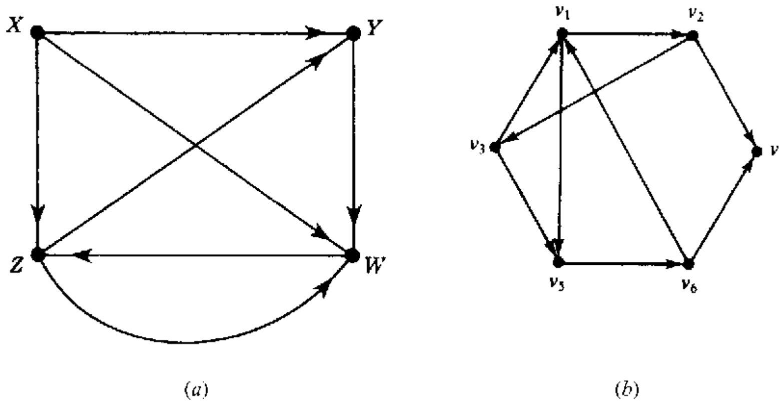


Fig. 9-21

9.2. Let G be the directed graph in Fig. 9-21(a).

(a) Find the indegree and outdegree of each vertex of G.
(b) Find the successor list of each vertex of G.
(c) Are there any sources or sinks?
(d) Find the subgraph H of G determined by the vertex set $V' = X, Y, Z$.

(a) Count the number of edges ending and beginning at a vertex v to obtain, respectively, indeg(v) and outdeg(v). This yields the data:

$$\begin{array}{cccc} \text{indeg}(X) = 0, & \text{indeg}(Y) = 2, & \text{indeg}(Z) = 2, & \text{indeg}(W) = 3 \\ \text{outdeg}(X) = 3, & \text{outdeg}(Y) = 1, & \text{outdeg}(Z) = 2, & \text{outdeg}(W) = 1 \end{array}$$

(As expected, the sum of the indegrees and the sum of the outdegrees each equal 7, the number of edges.)

(b) Add vertex v to the successor list succ(u) of u for each edge (u, v) in G. This yields:

$$\text{succ}(X) = [Y, Z, W], \quad \text{succ}(Y) = [W], \quad \text{succ}(Z) = [Y, W], \quad \text{succ}(W) = [Z]$$

(*c*) X is a source no edge enters X, that is, $\text{indeg}(X) = 0$. There are no sinks since every vertex is the initial point of an edge, that is, has nonzero outdegree.

(*d*) Let E' consists of all edges of G whose endpoints lie in V'. This yield $E' = \{(X, Y), (X, Z), (Z, Y)\}$. Then $H = H(V', E')$

9.3 Let G be the directed graph in Fig. 9-21(*b*).

(*a*) Find two simple paths from v_1 to v_6. Is $\alpha = (v_1, v_2, v_4, v_6)$ such a simple path?

(*b*) Find all cycles in G which include v_3.

(*c*) Is G unilaterally connected? Strongly connected?

(*d*) Find the successor list of each vertex of G.

(*e*) Are there any sources in G? Any sinks?

(*a*) A simple path is a path where all vertices are distinct. Thus (v_1, v_5, v_6) and (v_1,v_2,v_3,v_5,v_6) are two simple paths from v_1 to v_6. The sequence is not even a path since the edge joining v_4 to v_6 does not begin at v_4.

(*b*) There are two such cycles: (v_3, v_1, v_2, v_3) and $(v_3, v_5, v_6, v_1, v_2, v_3)$.

(*c*) G is unilaterally connected since $(v_1, v_2, v_3, v_5, v_6, v_4)$ is a spanning path. G is not strongly connected since there is no closed spanning path.

(*d*) Add vertex v to the successor list $\text{succ}(u)$ of u for each edge (u, v) in G. This yields:

$$\text{succ}(v_1) = [v_2, v_5], \quad \text{succ}(v_2) = [v_3, v_4], \quad \text{succ}(v_3) = [v_1, v_5]$$
$$\text{succ}(v_4) = \varnothing, \quad \text{succ}(v_5) = [v_6], \quad \text{succ}(v_6) = [v_1, v_4]$$

(As expected, the number of successors equals 9, which is the number of edges.)

(*e*) There are no sources since every vertex is the endpoint of some edge. Only v_4 is a sink since no edge begins at v_4, that is, $\text{succ}(v_4) = \varnothing$, the empty set.

9.4. Let G be the directed graph with vertex set $V(G) = (a, b, c, d, e, f, g\}$ and edge set:

$$E(G) = \{(a, a), (b, e), (a, e), (e, b), (g, c), (a, e), (d, f), (d, b), (g, g)\}$$

(*a*) Identify any loops or parallel edges.

(*b*) Are there any sources in G?

(*c*) Are there any sinks in G?

(*d*) Find the subgraph H of G determined by the vertex set $V' = \{a, b, c, d\}$.

(*a*) A loop is an edge with the same initial and terminal points; hence (a, a) and (g, g) are loops. Two edges are parallel if they have the same initial and terminal points. Thus (a, e) and (a, e) are parallel edges.

(*b*) The vertex d is a source since no edge ends in d, that is, d does not appear as the second element in any edge. There are no other sources.

(*c*) Both c and f are sinks since no edge begins at c or at f, that is, neither c nor f appear as the first element in any edge. There are no other sinks.

(*d*) Let E' consist of all edges of G whose endpoints lie in $V' = \{a, b, c, d\}$. This yields $E' = \{(a, a), (d, b)\}$. Then $H = H(V', E')$.

ROOTED TREES, ORDERED ROOTED TREES

9.5 Let T be the rooted tree in Fig. 9-22.

(*a*) Identify the path α from the root R to each of the following vertices, and find the level number n of the vertex: (i) H; (ii) F; (iii) M.

(*b*) Find the siblings of E.

(*c*) Find the leaves of T.

(*a*) List the vertices while proceeding from R down the tree to the vertex. The number of vertices, other than R, is the level number:

$$\text{(i)}\, \alpha = (R, A, C, H),\ n = 3; \quad \text{(ii)}\, \alpha = (R, B, F),\ n = 2; \quad \text{(iii)}\, \alpha = (R, B, G, L, M),\ n = 4.$$

(*b*) The siblings of E are F and G since they all have the same parent B.

(*c*) The leaves are vertices with no children, that is, H, D, I, J, K, M, N.

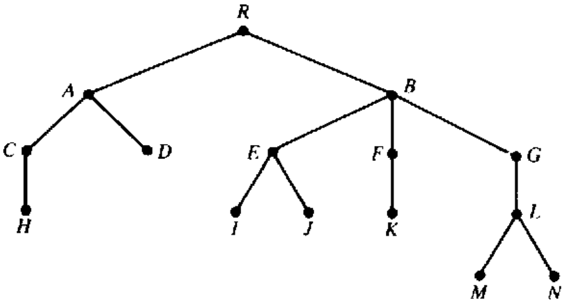


Fig. 9-22

9.6. Let T be the ordered rooted tree in Fig. 9-23 whose vertices are labeled using the universal address system. Find the lexicographic order of the addresses of the tree T.

An ordered rooted tree T is usually drawn so the edges are ordered from left to right as in Fig. 9-23. The lexicographic order can be obtained by reading down the leftmost branch, then the second branch from the left, and so forth.

Reading down the leftmost branch of T we obtain:

0, 1, 1.1, 1.1.1

The next branch is 1.2, 1.2.1, 1.2.1.1, so we add this branch to the list to obtain

0, 1, 1.1, 1.1.1, 1.2 1.2.1, 1.2.1.1

Proceeding in this manner, we finally obtain

0, 1, 1.1, 1.1.1, 1.2, 1.2.1, 1.2.1.1, 1.2.2, 1.3, 2, 2.1, 2.2.1

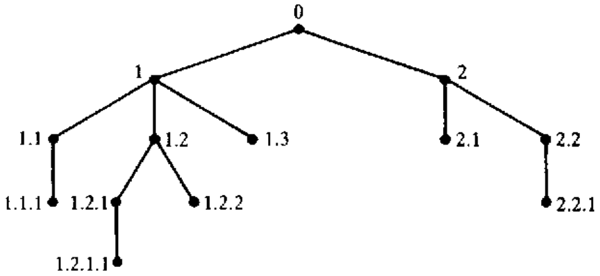


Fig. 9-23

SEQUENTIAL REPRESENTATION OF GRAPHS

9.7. Consider the graph G in Fig. 9-21(*a*), and suppose the vertices are stored in memery in the array:

DATA: X, Y, Z, W

(*a*) Find the adjacency matrix A of the graph G and the powers A^2, A^3, A^4.

(*b*) Find the path matrix P of G using the powers of A. Is G strongly connected?

(a) The vertices are normally ordered according to the way they appear in memory; that is, we assume $v_1 = X$, $v_2 = Y$, $v_3 = Z$, $v_4 = W$. The adjacency matrix $A = [a_{ij}]$ is obtained by setting $a_{ij} = 1$ if there is an edge from v_i to v_j; and 0 otherwise. The matrix A and its powers follow:

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad A^2 = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}, \quad A^3 = \begin{bmatrix} 0 & 1 & 2 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad A^4 = \begin{bmatrix} 0 & 2 & 2 & 3 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

(*b*) Since G has 4 vertices, we need only find the matrix $B_4 = A + A^2 + A^3 + A^4$, and then the path matrix $P = [p_{ij}]$ is obtained by setting $p_{ij} = 1$ whenever there is a nonzero entry in the matrix B_4, and 0 otherwise. The matrices B_4 and P follow:

$$B_4 = \begin{bmatrix} 0 & 5 & 6 & 8 \\ 0 & 1 & 2 & 3 \\ 0 & 3 & 3 & 5 \\ 0 & 2 & 3 & 5 \end{bmatrix} \quad \text{and} \quad P = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

The path matrix P shows there is no path from any node to v_1. Hence G is not strongly connected.

9.8. Consider the adjacency matrix A of the graph G in Fig. 9-19(*a*) obtained in Problem 9.7. Find the path matrix P of G using Warshall's algorithm rather than the powers of A.

Initially set $P_0 = A$. Then, P_1, P_2, P_3, P_4 are obtained recursively by setting

$$P_k[i, j] = P_{k-1}[i, j] \vee (P_{k-1}[i, k] \wedge P_{k-1}[k, j])$$

where P_k[i, j] denotes the *ij*-entry in the matrix $P_{k.}$ That is, by setting

$$P_k[i, j] = 1 \quad \text{if} \quad P_{k-1}[i, j] = 1 \quad \text{or if both} \quad P_{k-1}[i, k] = 1 \quad \text{and} \quad P_{k-1}[k, j] = 1$$

Then matrices P_1, P_2, P_3, P_4 follow:

$$P_1 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad P_3 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}, \quad P_4 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

Observe that $P_1 = P_2 = A$. The changes in P_3 occur for the following reasons:

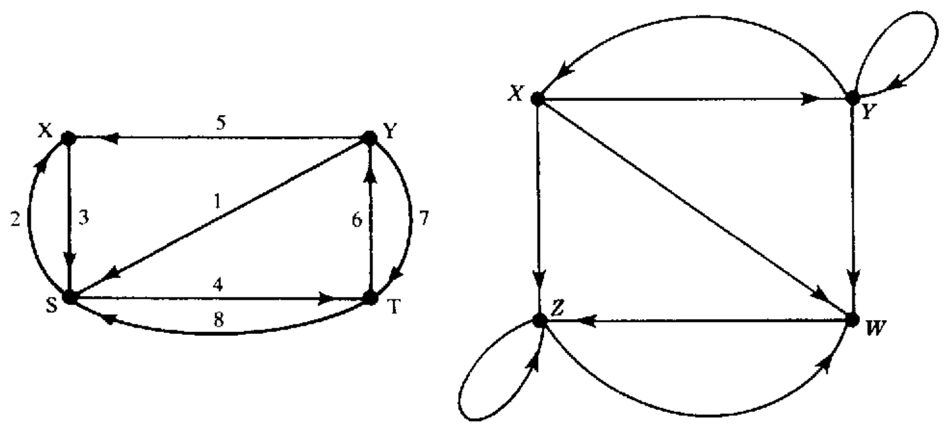
$$P_3[4, 2] = 1 \quad \text{because} \quad P_2[4, 3] = 1 \text{ and } P_2[3, 2] = 1$$
$$P_3[4, 4] = 1 \quad \text{because} \quad P_2[4, 3] = 1 \text{ and } P_2[3, 4] = 1$$

9.9. Draw a picture of the weighted graph G which is maintained in memory by the following vertex array DATA and weight matrix W:

$$\text{DATA: } X, Y, S, T; \quad W = \begin{bmatrix} 0 & 0 & 3 & 0 \\ 5 & 0 & 1 & 7 \\ 2 & 0 & 0 & 4 \\ 0 & 6 & 8 & 0 \end{bmatrix}$$

The picture appears in Fig. 9-24(*a*). The vertices are labeled by the entries in DATA.

Assuming $v_1 = X$, $v_2 = Y$, $v_3 = S$, $v_4 = T$, the order the vertices appear in the array DATA, we draw an edge from v_i to v_j with weight w_{ij} when $w_{ij} \neq 0$.



(a) (b)

Fig. 9-24

LINKED REPRESENTATION OF GRAPHS

9.10. Let G be the graph presented by the following table:

$$G = [X : Y, Z, W; \quad Y : X, Y, W; \quad Z : Z, W; \quad W : Z]$$

(a) Find the number of vertices and edges in G.

(b) Draw the graph of G.

(c) Are there any sources or sinks?

(a) The table tells us that there are four vertices, X, Y, Z, W. The outdegrees of the vertices are 3, 3, 2, 1, respectively. Thus there are $3 + 3 + 2 + 1 = 9$ edges.

(b) Using the adjacency lists, draw the graph in Fig. 9-24(b).

(c) No vertex has zero outdegree, so there are no sinks. Also, no vertex has zero indegree, that is, each vertex is a successor; hence there are no sources.

9.11. A weighted graph G with six vertices, $A, B, \ldots, F$, is stored in memory using a linked representation with a vertex file and an edge file as in Fig. 9-25(a).

Vertex file

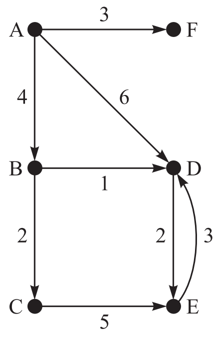
START 3

	1	2	3	4	5	6	7	8
VERTEX	*D*		*B*	*F*	*A*		*C*	*E*
NEXT-V	7		1	5	0		8	4
PTR	9		3	0	6		10	1

Edge file

	1	2	3	4	5	6	7	8	9	10
BEG-V	8	5	3		5	5	3		1	7
END-V	1	4	7		3	1	1		8	8
NEXT-E	0	5	7		0	2	0		0	0
WEIGHT	3	4	2		6	3	1		2	5

(a)



(b)

Fig. 9-25

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

(b) Find the successor list succ(v) of each vertex v.

(c) Draw the graph of G.

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

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

(*b*) Here $\text{succ}(A) = [1(D),\ 4(F),\ 3(B)] = [D, F, B]$. Specifically, $\text{PTR}[5(A)] = 6$ and $\text{END-V}[6] = 1(D)$ tells us that $\text{succ}(A)$ begins with D. Then $\text{NEXT-E}[6] = 2$ and $\text{END-V}[2] = 4(F)$ tells us that F is the next vertex in $\text{succ}(A)$. Then $\text{NEXT-E}[2] = 5$ and $\text{END-V}[5] = 3(B)$ tells us that B is the next vertex in $\text{succ}(A)$. However, $\text{NEXT-E}[5] = 0$ tells us that there are no more successors of A. Similarly,

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

Furthermore $\text{succ}(F) = \varnothing$, since $\text{PTR}[4(F)] = 0$. In other words,

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

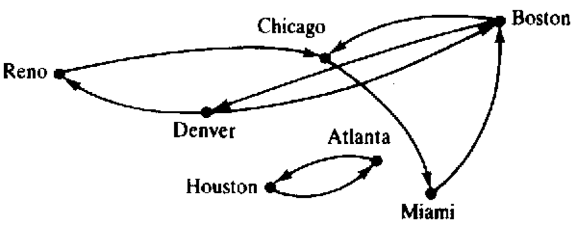
(*c*) Use the successor lists obtained in (b) and the weights of the edges in the Edge File in Fig. 9-25(*a*) to draw the graph in Fig. 9-25(*b*).

9.12. Suppose Friendly Airways has nine daily flights as follows:

103	Atlanta to Houston	203	Boston to Denver	305	Chicago to Miami
106	Houston to Atlanta	204	Denver to Boston	308	Miami to Boston
201	Boston to Chicago	301	Denver to Reno	401	Reno to Chicago

Describe the data by means of a labeled directed graph G.

The data are described by the graph in Fig. 9-26(*a*) (where the flight numbers have been omitted for notational convenience.)



(*a*)

Vertex file

	1	2	3	4	5	6	7	8
CITY	*A*	*B*	*C*	*D*	*H*	*M*	*R*	
PTR	1	3	7	5	2	8	9	

Edge file

	1	2	3	4	5	6	7	8	9	10
NUMBER	103	106	201	203	204	301	305	308	402	
ORIG	1	5	2	2	4	4	3	6	7	
DEST	5	1	3	4	2	7	6	2	3	
NEXT-E	0	0	4	0	6	0	0	0	0	

(*b*)

Fig. 9-26

9.13. Describe how the graph in Problem 9.12 may appear in memory using a linked representation where the cities and flights appear in linear sorted arrays.

See Fig. 9-26(*b*) (where A, B, ... denote, respectively, Atlanta, Boston,. . .). There is no need for a START variable since the cities form an array, not a linked list. We also use ORIG (origin) and DEST (destination) instead of BEG-V and END-V.

9.14. Clearly, the data in Problem 9.12 may be efficiently stored in a file where each record contains only three fields:

Flight Number, City of Origin, City of Destination

However, when there are many, many flights, such a representation does not easily answer the following natural questions:

(i) Is there a direct flight from city X to city Y?
(ii) Can one fly from city X to city Y?
(iii) What is the most direct route (minimum number of stops) from city X to city Y?

Show how the answer, say, to (ii) may be more readily available if the data is stored in memory using the linked representation of the graph as in Fig. 9-26(*b*).

One way to answer (ii) is to use a breadth-first or depth-first search algorithm to decide whether city Y is reachable from city X. Such algorithms require the adjacency lists, which can easily be obtained from the linked representation of a graph, but not from the above representation which uses only three fields.

MISCELLANEOUS PROBLEMS

9.15. Let $A = \begin{bmatrix} 0 & 2 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 2 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$ be the adjacency matrix of a multigraph G. Draw a picture of G.

Since A is a 4×4 matrix, G has four vertices, v_1, v_2, v_3, v_4. For each entry a_{ij} in A, draw a_{ij} arcs (directed edges) from vertex v_i to vertex v_j to obtain the graph in Fig. 9-27(a).

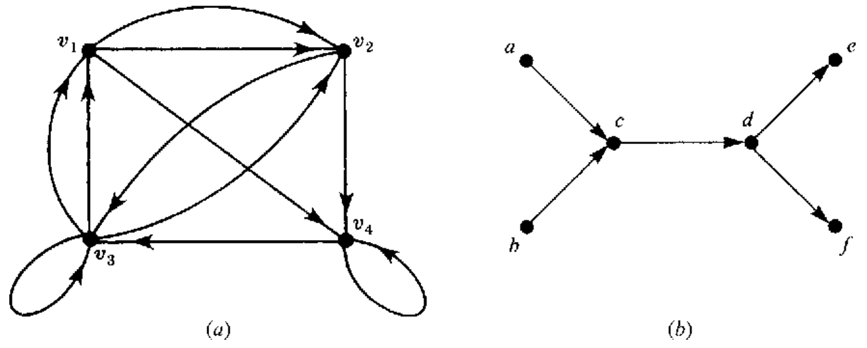


Fig. 9-27

9.16. Let S be the cycle-free graph in Fig. 9-27(b). Find all possible topological sorts of S.

There are four possible topological sorts of S. Specifically, each sort T must begin with either a or b, must end with e or f, and c and d must be the third and fourth elements, respectively. The four sorts follow:

$$T_1 = [a, b, c, d, e, f], \qquad T_2 = [b, a, c, d, e, f]$$
$$T_3 = [a, b, c, d, f, e], \qquad T_4 = [b, a, c, d, f, e]$$

9.17. Prove Proposition 9.4: Let A be the adjacency matrix of a graph G. Then $a_K[i, j]$, the ij-entry in the matrix A^K, gives the number of paths of length K from v_i to v_j.

The proof is by induction on K. A path of length 1 from v_i to v_j is precisely an edge (v_i, v_j). By definition of the adjacency matrix A, $a_1[i, j] = a_{ij}$ gives the number of edges from v_i to v_j. Hence the proposition is true for $K = 1$.

Suppose $K > 1$. (Assume G has m nodes.) Since $A^K = A^{K-1}A$,

$$a_K[i, j] = \sum_{s=1}^{m} a_{K-1}[i, s]\, a_1[s, j]$$

By induction, $a_{K-1}[i, s]$ gives the number of paths of length $K - 1$ from v_i to v_s and $a_1[s, j]$ gives the number of paths of length 1 from v_s to v_j. Thus $a_{K-1}[i, s]a_1[s, j]$ gives the number of paths of length K from v_i to v_j where v_s is the next-to-last vertex. Thus all paths of length K from v_i to v_j can be obtained by summing up the product $a_{K-1}[i, s]\, a_1[s, j]$ for all s. Accordingly, $a_K[i, j]$ is the number of paths of length K from v_i to v_j. Thus the proposition is proved.

Supplementary Problems

GRAPH TERMINOLOGY

9.18. Consider the graph G in Fig. 9-28(a).

(a) Find the indegree and outdegree of each vertex.

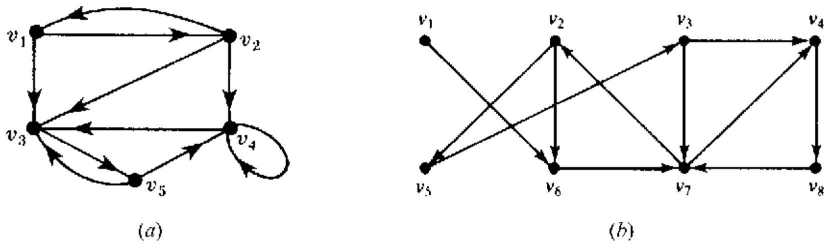
(b) Are there any sources or sinks?

(c) Find all simple paths from v_1 to v_4.

(d) Find all cycles in G.

(e) Find all paths of length 3 or less from v_1 to v_3.

(f) Is G unilaterally or strongly connected?



(a) (b)

Fig. 9-28

9.19. Consider the graph G in Fig. 9-28(b).

(a) Are there any sources or sinks?

(b) Find all simple paths from v_1 to v_4.

(c) Find a non-simple path from v_1 to v_4.

(d) Find all cycles in G which include v_4.

9.20. Consider the graph G in Fig. 9-28(b).

(a) Find: $\text{succ}(v_1)$, $\text{succ}(v_3)$, $\text{succ}(v_5)$, $\text{succ}(v_7)$.

(b) Find the subgraph H of G generated by: (i) $\{v_1, v_3, v_5, v_6\}$; (ii) $\{v_2, v_3, v_6, v_7\}$.

9.21. Let G be the graph with vertex set $V(G) = \{A, B, C, D, E\}$ and edge set

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

(a) Express G by its adjacency table.

(b) Does G have any loops or parallel edges?

(c) Find all simple paths from D to E.

(d) Find all cycles in G.

(e) Is G unilaterally or strongly connected?

(f) Find the number of subgraphs of G with vertices C, D, E.

(g) Find the subgraph H of G generated by C, D, E.

9.22. Let G be the graph with vertex set $V(G) = \{a, b, c, d, e\}$ and the following successor lists:

$$\text{succ}(a) = [b, c], \quad \text{succ}(b) = \varnothing, \quad \text{succ}(c) = [d, e], \quad \text{succ}(d) = [a, b, e], \quad \text{succ}(e) = \varnothing$$

(a) List the edges of G. (b) Is G weakly, unilaterally, or strongly connected?

9.23. Let G be the graph in Fig. 9-29(a).

(a) Express G by its adjacency table.

(b) Does G have any sources or sinks?

(c) Find all simple paths from A to E.

(d) Find all cycles in G.

(e) Find a spanning path of G.

(f) Is G strongly connected?

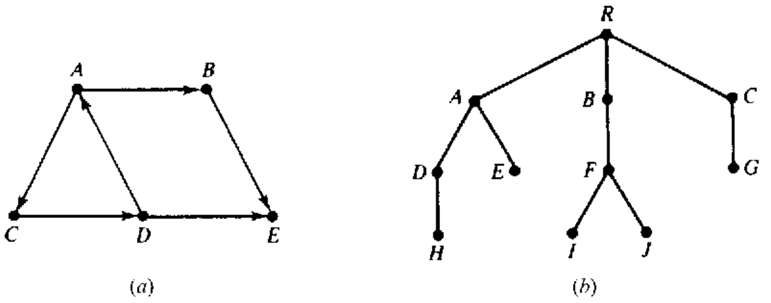


Fig. 9-29

ROOTED TREES, ORDERED ROOTED TREES

9.24. Let T be the rooted tree in Fig. 9-29(b).

(a) Identify the path α from the root R to each of the following vertices, and find the level number of the vertex: (i) D; (ii) J; (iii) G.

(b) Find the leaves of T.

(c) Assuming T is an ordered rooted tree, find the universal address of each leaf of T.

9.25. The following addresses are in random order:

2.1.1, 3.1, 2.1, 1, 2.2.1.2, 0, 3.2, 2.2, 1.1, 2, 3.1.1, 2.2.1, 3, 2.2.1.1

(a) Place the addresses in lexicographic order.

(b) Draw the corresponding rooted tree.

SEQUENTIAL REPRESENTATION OF GRAPHS

9.26. Let G be the graph in Fig. 9-30(a).

(a) Find the adjacency matrix A and the path matrix P for G.

(b) For all $k > 0$, find n_k where n_k denotes the number of paths of length k from v_1 to v_4.

(c) Is G weakly, unilaterally, or strongly connected?

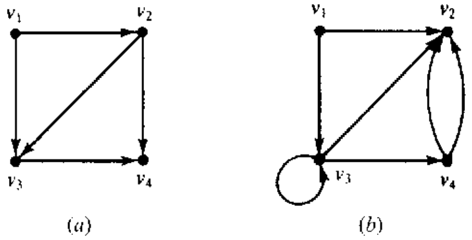


Fig. 9-30

9.27. Repeat Problem 9.26 for the graph G in Fig. 9-30(b).

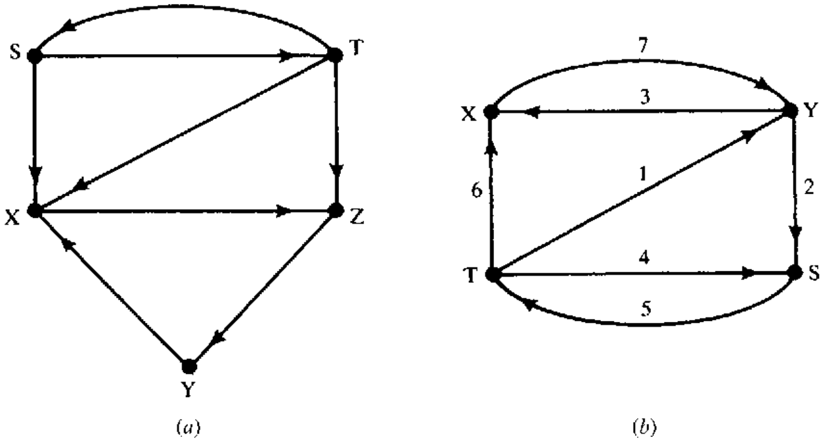
9.28. Let P be the path matrix for a graph G. Describe P when G is: (a) strongly connected; (b) unilaterally connected.

9.29. Let G be the graph in Fig. 9-31(a) where the vertices are maintained in memory by the array DATA: X, Y, Z, S, T. (a) Find the adjacency matrix A and the path matrix P of G. (b) Find all cycles in G. (c) Is G unilaterally connected? Strongly connected?

9.30. Let G be the weighted graph in Fig. 9-31(b) where the vertices are maintained in memory by the array DATA: X, Y, S, T.

(a) Find the weight matrix W of G.

(b) Find the matrix Q of shortest paths using Warshall's algorithm.



(a)

(b)

Fig. 9-31

LINKED REPRESENTATION OF GRAPHS

9.31. A weighted graph G with six vertices, A, B, ..., F, is stored in memory using a linked representation with a vertex file and an edge file as in Fig. 9-32.

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

(b) Find the successor list succ(v) of each vertex v in G.

(c) Draw a picture of G.

Vertex file

START $\boxed{7}$

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

Edge file

	1	2	3	4	5	6	7	8	9	10	11	12
BEG-V	5	5	7		3	7	1		4	1	4	7
END-V	8	7	5		1	1	5		8	4	3	8
NEXT-E	0	1	12		0	0	10		11	0	0	6
WEIGHT	5	2	1		3	2	4		1	3	4	1

Fig. 9-32

9.32. Let G be the graph presented by the table: $G = [A:B, C;\ \ B:C, D;\ \ C:C;\ \ D:B;\ \ E:\varnothing]$.

(a) Find the number of vertices and edges in G.

(b) Draw a picture of G.

(c) Are there any sources or sinks?

(d) Is G weakly, unilaterally, or strongly connected?

9.33. Repeat Problem 9.32 for the table: $G = [A:D;\ \ B:C;\ \ C:E;\ \ D:B, D, E;\ \ E:A]$.

9.34. Repeat Problem 9.32 for the table: $G = [A:B, C, D, K;\ \ B:J;\ \ C:\varnothing; D:\varnothing;\ \ J:B, D, L;\ \ K:D, L;\ \ L:D]$.

9.35. Suppose Friendly Airways has eight daily flights serving the seven cities: Atlanta, Boston, Chicago, Denver, Houston, Philadelphia, Washington. Suppose the data on the flights are stored in memory as in Fig. 9-33; that is, using a linked representation where the cities and flights appear in linear sorted arrays. Draw a labeled directed graph G describing the data.

Vertex file

	1	2	3	4	5	6	7	8
CITY	*A*	*B*	*C*	*D*	*H*	*P*	*W*	
PTR	1	2	3	8	9	5	7	

Edge file

	1	2	3	4	5	6	7	8	9	10
NUMBER	101	102	201	202	203	301	302	401	402	
ORIG	1	2	3	1	6	6	7	4	5	
DEST	2	3	6	7	3	1	6	5	4	
NEXT-E	4	0	0	0	6	0	0	0	0	

Fig. 9-33

9.36. Using the data in Fig. 9-33, write a procedure with input CITY X and CITY Y which finds the flight number of a direct flight from city X to city Y, if it exists. Test the procedure using:

(a) $X =$ Atlanta, $Y =$ Philadelphia; (c) $X =$ Houston, $Y =$ Chicago;

(b) $X =$ Philadelphia, $Y =$ Atlanta; (d) $X =$ Washington, $Y =$ Chicago.

9.37. Using the data in Fig. 9-33, write a procedure with input CITY X and CITY Y which finds the most direct route (minimum number of stops) from city X to city Y, if it exists. Test the procedure using the input in Problem 9.36.

MISCELLANEOUS PROBLEMS

9.38. Use the pruning algorithm to find the shortest path from s to t in Fig. 9-34.

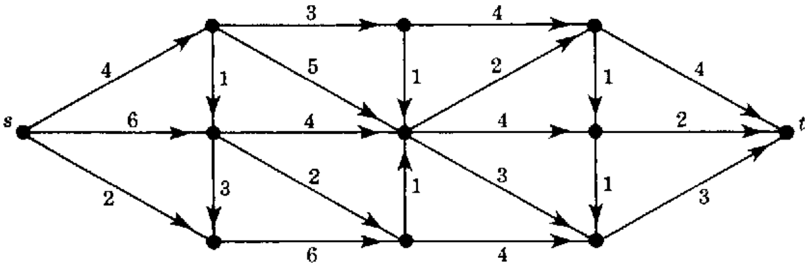


Fig. 9-34

9.39. Find a topological sort T of each of the following graphs:

(a) $G = [A : Z;\quad B : T;\quad C : B;\quad D : \emptyset;\quad X : D;\quad Y : X;\quad Z : B,\ X;\quad S : C,\ Z;\quad T : \emptyset]$

(b) $G = [A : X, Z;\quad B : A;\quad C : S, T;\quad D : Y;\quad X : S, T;\quad Y : B;\quad Z : \emptyset;\quad S : Y;\quad T : \emptyset]$

(c) $G = [A : C, S;\quad B : T, Z;\quad C : \emptyset;\quad D : Z;\quad X : A;\quad Y : A;\quad Z : X,\ Y;\quad S : \emptyset;\quad T : Y]$

9.40. Draw a labeled graph G that represents the following situation. Three sisters, Barbara, Rose, and Susan, each regularly telephone their mother Gertrude, though Gertrude calls only Rose. Susan will not call Rose, though Rose continues to call Susan. Barbara and Susan will call each other, and Barbara and Rose will call each other.

9.41. Let R be the relation (directed graph) on $V = \{2, 3, 4, 9, 15\}$ defined by "x is less than and relatively prime to y." (a) Draw the diagram of the graph R. (b) Is R weakly connected? Unilaterally connected? Strongly connected?

9.42. A directed graph G is *complete* if, for each pair of distinct vertices u and v, either $(u,\ v)$ is an arc or $(v,\ u)$ is an arc. Show that a finite complete directed graph G has a path which includes all vertices. (This obviously holds for non-directed complete graphs.) Thus G is unilaterally connected.

9.43. Suppose a graph G is input by means of an integer M, representing the vertices $1, 2, \ldots, M$, and a list of N ordered pairs of integers representing the edges of G. Write a procedure for each of the following:

(a) Finds the $M \times M$ adjacency matrix A of the graph G.

(b) Uses A and Warshall's algorithm to find the path matrix P of G.

Test the above procedure using the following data:

(i) $M = 5$; $N = 8$; (3, 4), (5, 3), (2, 4), (1, 5), (3, 2), (4, 2), (3, 1), (5, 1)

(ii) $M = 6$; $N = 10$; (1, 6), (2, 1), (2, 3), (3, 5), (4, 5), (4, 2), (2, 6), (5, 3), (4, 3), (6, 4)

9.44. Suppose a graph G is input by means of an integer M, representing the vertices $1, 2, \ldots, M$, and a list of N ordered triples (a_i, b_i, w_i) of integers such that (a_i, b_i) is an edge of G and w_i is its weight. Write a procedure for each of the following:

(a) Finds the $M \times M$ weight matrix W of the graph G.

(b) Uses W and Warshall's algorithm to find the matrix Q of shortest paths between the vertices of G.

Test the above procedure using the following data:

(i) $M = 4$; $N = 7$; (1, 2, 5), (2, 4, 2), (3, 2, 3), (1, 1, 7), (4, 1, 4), (4, 3, 1)

(ii) $M = 5$; $N = 8$; (3, 5, 3), (4, 1, 2), (5, 2, 2), (1, 5, 5), (1, 3, 1), (2, 4, 1), (3, 4, 4), (5, 4, 4)

9.45. Consider the graph G in Fig. 9-11. Show the sequence of waiting lists in STACK and the sequence of vertices processed while doing a depth-first search (DFS) of G beginning at the vertex: (a) B; (b) E; (c) K.

9.46. Consider the graph in Fig. 9-11. As in Example 9.11, find the shortest path from K to F using a breadth-first search of G. In particular, show the sequence of waiting lists in QUEUE during the search.

Answers to Supplementary Problems

Notation: $M = [R_1;\ R_2;\ \ldots;\ R_n\ ]$ denotes a matrix wih rows $R_1, R_2, \ldots, R_n$.

9.18. (a) Indegrees: 1, 1, 4, 3, 1; outdegrees: 2, 3, 1, 2, 2.

(b) None.

(c) $(v_1, v_2, v_4), (v_1, v_3, v_5, v_4), (v_1, v_2, v_3, v_5, v_4)$

(d) (v_3, v_5, v_4, v_3)

(e) $(v_1, v_3), (v_1, v_2, v_3), (v_1, v_2, v_4, v_3), (v_1, v_2, v_1, v_3)$ (v_1, v_3, v_5, v_7)

(f) Unilaterally, but not strongly.

9.19. (a) Sources: v_1

(b) $(v_1, v_6, v_7, v_4), (v_1, v_6, v_7, v_2, v_5, v_3, v_4)$

(c) $(v_1, v_6, v_7, v_2, v_6, v_7, v_4)$

(d) $(v_4, v_8, v_7, v_4), (v_4, v_8, v_7, v_2, v_5, v_3, v_4)$

9.20. (a) succ(1) = [6], succ(3) = [4, 7], succ(5) = [3], succ(7) = [2, 4].

(b) (i) (1, 6), (5, 3); (ii) (2, 6), (6, 7), (7, 2), (3, 7).

9.21. (a) $G = [A : D;\ B : C;\ C : E;\ D : B, D, E;\ E : A]$

(b) Loop: (D, D)

(c) $(D,\ E), (D,\ B,\ C,\ E)$

(d) $(A,\ D,\ E,\ A), (A,\ D,\ B,\ C,\ E,\ A)$

(e) Unilaterally, and strongly.

(f) And (g) H has three edges: $(C,\ E)$, $(D,\ E)$, $(D,\ D)$. There are $8 = 2^3$ ways of choosing some of the three edges; and each choice gives a subgraph.

9.22. (a) $(a,\ b), (a,\ c), (c,\ d), (c,\ e), (d,\ a), (d,\ b), (d,\ e)$

(b) Since b and e are sinks, there is no path from b to e or from e to b, so G is neither unilaterally nor strongly connected. G is weakly connected since, cc, a, b, d, e is a spanning semipath.

9.23. (a) $G = [A : B,\ C : B : E;\ C : D;\ E : \varnothing]$; (b) Sink: E; (c) $(A,\ B,\ E), (A,\ C,\ D,\ E)$; (d) $(A,\ C,\ D,\ A)$; (e) $C,\ D,\ A,\ B,\ E)$; (f) No.

9.24. (a) (i) $(R,\ A,\ D)$, 2; (ii) $(R,\ B,\ F,\ J)$, 3; (iii) $(R,\ C,\ G)$, 2.

(b) $H,\ E,\ I,\ J,\ G$

(c) H : 1.1.1, E : 1.2, I : 2.1.1, J : 2.1.2, G : 3.1

9.25. (a) 0, 1, 1.1, 2, 2.1, 2.1.1, 2.2, 2.2.1, 2.2.1.1, 2.2.1.2, 3, 3.1, 3.1.1, 3.2. (b) Fig. 9-35(a).

9.26. (a) $A =$ [0, 1, 1, 0; 0, 0, 1, 1; 0, 0, 0, 1; 0, 0, 0, 0]; $P =$ [0, 1, 1, 1; 0, 0, 1, 1; 0, 0, 0, 1; 0, 0, 0, 0];

(b) 0, 2, 1, 0, 0,. . .; (c) Weakly and unilaterally.

9.27. (a) $A =$ [0, 1, 1, 0; 0, 0, 0, 0; 0, 1, 1, 1; 0, 2, 0, 0]; $P =$ [0, 1, 1, 1; 0, 0, 0, 0; 0, 1, 1, 1; 0, 1, 0, 0];

(b) 0, 1, 1, 1,. . .; (c) Weakly and unilaterally.

9.28. Let $P = [p_{ij}]$. For $i \neq j$: (a) $p_{ij} \neq 0$; (b) either $p_{ij} \neq 0$ or $p_{ji} \neq 0$.

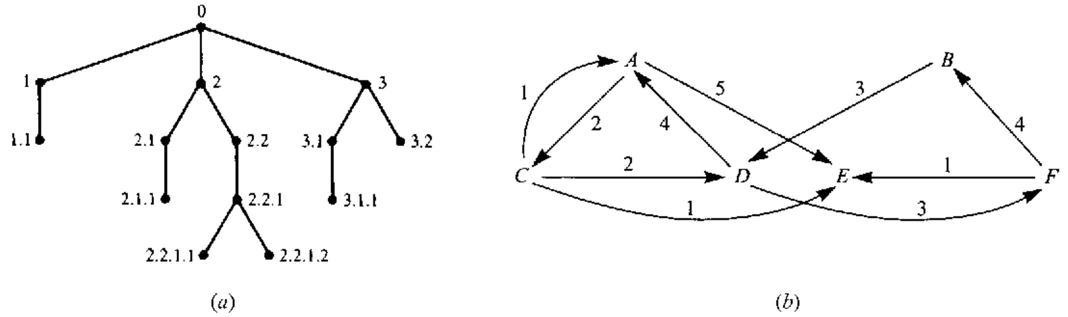


Fig. 9-35

9.29. (a) $A = [0, 0, 1, 0, 0; 1, 0, 0, 0, 0; 0, 1, 0, 0, 0; 1, 0, 0, 0, 1; 1, 0, 1, 1, 0]$;
$P = [1, 1, 1, 0, 0; 1, 1, 1, 0, 0; 1, 1, 1, 0, 0; 1, 1, 1, 1, 1; 1, 1, 1, 1, 1; 1, 1, 1, 1, 1]$
(b) $(X,\ Z,\ Y,\ X)$; (S, T, S) (c) Unilaterally.

9.30. (a) $A = [0, 7, 0, 0; 3, 0, 2, 0; 0, 0, 0, 5; 6, 1, 4, 0]$
(b) $Q =$ [*XYX*, *XY*, *XYS*, *XYST*; *YX*, *YSTY*, *YS*, *YST*; *STYX*, *STY*, *STYS*, *ST*; *TX*, *TY*, *TYS*, *TYST*]

9.31. (a) $C,\ F,\ D,\ B,\ E,\ A$; (b) $[A : C,\ E;\ B : D;\ C : D,\ E,\ A;\ D : A,\ F;\ E : \varnothing;\ F : B,\ E]$; (c) See Fig. 9-35(*b*).

9.32. (a) 5, 6; (b) source: *A*; (c) See Fig. 9-36(*a*); none.

9.33. (a) 5, 1; (b) none; (c) See Fig. 9-36(*b*); (d) all three.

9.34. (a) 7, 11; (b) source: *A*; sinks: *C*, *D*; (c) See Fig. 9-36(*c*); (d) only weakly.

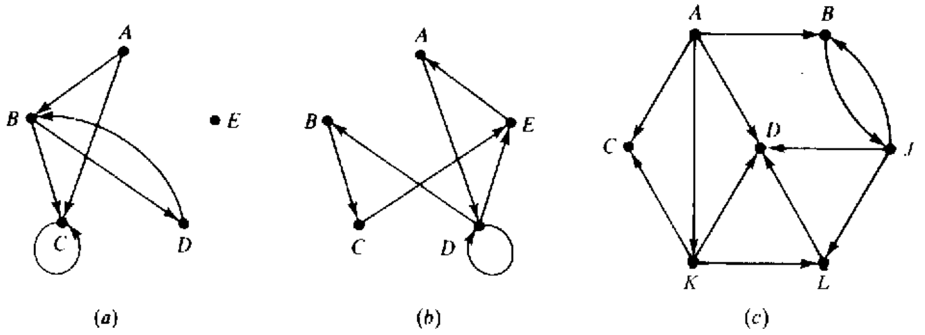


Fig. 9-36

9.35. See Fig. 9-37.

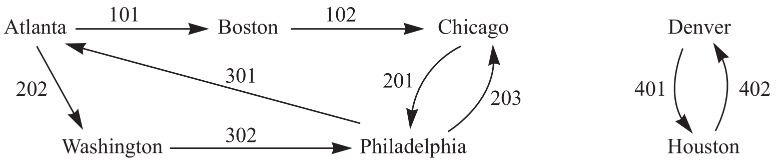


Fig. 9-37

9.36. (a) No; (b) yes; (c) no; (d) no.

9.37. (a) *AWP*; (b) *PA*; (c) none; (d) *WPC*.

9.38. $(s, 4, 1, 2, 1, 2, 1, 2, t)$

9.39. ***Hint*:** First draw the graph. (a) *ASYCZBXTD*; (b) none, the graph is not cycle-free, e.g., *YBAXSY* is a cycle; (c) *BTYXACSDZ*.

9.40. See Fig. 9-38(*a*).

9.41. (a) See Fig. 9-38(*b*). (b) Only weakly connected.

9.42. ***Hint*:** Suppose $(\alpha = v_1, \ldots, v_m)$ is a longest path in G and does not include vertex u. If $(u_1 v_1)$ is an arc, then $\beta = (u, \alpha)$ extends α. Hence (v_1, u) is an arc. If (u, v_2) is also an arc, then $\beta = (v_1, u, v_2, \ldots, v_m)$ extends α; hence (v_2, u) is an arc. Similarly $(v_3, u), \ldots, (v_m, u)$ are arcs. Hence $\beta = (\alpha,\ u)$ extends α. This contradicts the maximality of α.

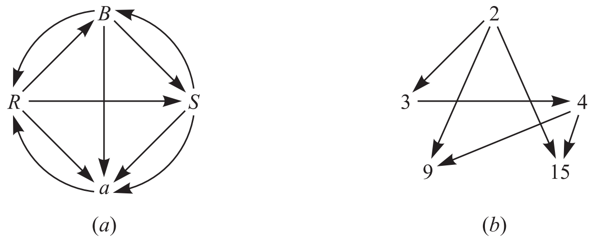


Fig. 9-38

9.43. (i) $A = [0, 0, 0, 0, 0; 0, 0, 0, 1, 0; 1, 1, 0, 1, 0; 0, 1, 0, 0, 0; 1, 0, 1, 0, 0]$
$P = [1, 1, 1, 1, 1; 0, 1, 0, 1, 0; 1, 1, 1, 1, 1; 0, 1, 0, 1, 0; 1, 1, 1, 1, 1]$
(ii) $A = [0, 0, 0, 0, 0, 1; 1, 0, 1, 0, 0, 1; 0, 0, 0, 0, 1, 0; 0, 1, 1, 0, 1, 0; 0, 0, 1, 0, 0, 0; 0, 0, 0, 1, 0, 0]$
$P = [1, 1, 1, 1, 1, 1; 1, 1, 1, 1, 1, 1; 0, 0, 1, 0, 1, 0; 1, 1, 1, 1, 1, 1; 0, 0, 1, 0, 1, 0; 1, 1, 1, 1, 1, 1]$

9.44. (i) $W = [7, 5, 0, 0; 0, 0, 0, 2; 0, 3, 0, 0; 4, 0, 1, 0]$; $Q = [AA, AB, ABCD, ABD; BDA, BDCB, BDC, BD; CBDA, CB, CBDC, CBD; DA, DCB, DC, DCBD]$, where A, B, C, D are the vertices.
(ii) $W = [0, 0, 1, 0, 5; 0, 0, 0, 1, 0; 0, 0, 0, 4, 3; 2, 0, 0, 0, 0; 0, 2, 0, 4, 0]$; $Q = [ACDA, ACEB, AC, ACD, ACE; BDA, BDACEB, BDAC, BD, BDACE; CDA, CEB, CDAC, CD, CE; DA, DACEB, DAC, DACD, DACEB; EDA, EB, EDAC, ED, EDACE]$ where A, B, C, D, E are the vertices.

9.45. (a) STACK: $B, L_B\,E_B, E_L\,C_L\,E_B, F_E\,C_L, D_F\,C_L, C_L, J_C, K_J, \varnothing$; Vertex: B, L_B, E_L, F_E, D_F, C_L, J_C, K_J
(b) STACK: E, F_E, D_F, $\varnothing$; Vertex: E, F_E, D_F
(c) STACK: K, $L_K C_K$, $E_L C_L$,C_K, C_L, $D_F C_L$, $C_L J_C$, $\varnothing$; Vertex: K, L_K, E_L, F_E, D_F, C_L, J_C

9.46. QUEUE: K, $L_K C_K$, $J_C E_C D_C L_K$, $J_C E_C D_C$, $J_C E_C$, F_E; Vertex: K, C_K, L_K, D_C, E_C, J_C, F_E; Minimal Path: $F_E \leftarrow E_C \leftarrow C_K \leftarrow$ or $K \rightarrow C_K \rightarrow E_C \rightarrow F_E$.