

8.2 Marshall's and Floyd's Algorithms

In this section, we look at two well-known algorithms: Marshall's algorithm for computing the transitive closure of a directed graph and Floyd's algorithm for the all-pairs shortest-paths problem. These algorithms are based on essentially the same idea, which we can interpret as an application of the dynamic programming technique.

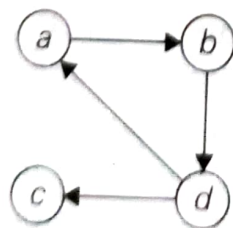
Marshall's Algorithm

Recall that the adjacency matrix $A = \{a_{ij}\}$ of a directed graph is the boolean matrix that has 1 in its i th row and j th column if and only if there is a directed edge from the i th vertex to the j th vertex. We may also be interested in a matrix containing the information about the existence of directed paths of arbitrary lengths between vertices of a given graph.

DEFINITION The transitive closure of a directed graph with n vertices can be defined as the n -by- n boolean matrix $T = \{t_{ij}\}$, in which the element in the i th row ($1 \leq i \leq n$) and the j th column ($1 \leq j \leq n$) is 1 if there exists a nontrivial directed path (i.e., a directed path of a positive length) from the i th vertex to the j th vertex; otherwise, t_{ij} is 0.

An example of a digraph, its adjacency matrix, and its transitive closure is given in Figure 8.2.

We can generate the transitive closure of a digraph with the help of depth-first search or breadth-first search. Performing either traversal starting at the i th vertex gives the information about the vertices reachable from the i th vertex and hence the columns that contain ones in the i th row of the transitive closure. Thus, doing such a traversal for every vertex as a starting point yields the transitive closure in its entirety.



(a)

$$A = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

(b)

$$T = \begin{matrix} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{matrix}$$

(c)

FIGURE 8.2 (a) Digraph. (b) Its adjacency matrix. (c) Its transitive closure.

Since this method traverses the same digraph several times, we should hope that a better algorithm can be found. Indeed, such an algorithm exists. It is called **Warshall's algorithm** after S. Warshall [War62]. Warshall's algorithm constructs the transitive closure of a given digraph with n vertices through a series of n -by- n boolean matrices:

$$R^{(0)}, \dots, R^{(k-1)}, R^{(k)}, \dots, R^{(n)}. \quad (8.5)$$

Each of these matrices provides certain information about directed paths in the digraph. Specifically, the element $r_{ij}^{(k)}$ in the i th row and j th column of matrix $R^{(k)}$ ($k = 0, 1, \dots, n$) is equal to 1 if and only if there exists a directed path (of a positive length) from the i th vertex to the j th vertex with each intermediate vertex, if any, numbered not higher than k . Thus, the series starts with $R^{(0)}$, which does not allow any intermediate vertices in its paths; hence, $R^{(0)}$ is nothing else but the adjacency matrix of the digraph. (Recall that the adjacency matrix contains the information about one-edge paths, i.e., paths with no intermediate vertices.) $R^{(1)}$ contains the information about paths that can use the first vertex as intermediate; thus, with more freedom, so to speak, it may contain more ones than $R^{(0)}$. In general, each subsequent matrix in series (8.5) has one more vertex to use as intermediate for its paths than its predecessor and hence may, but does not have to, contain more ones. The last matrix in the series, $R^{(n)}$, reflects paths that can use all n vertices of the digraph as intermediate and hence is nothing else but the digraph's transitive closure.

* The central point of the algorithm is that we can compute all the elements of each matrix $R^{(k)}$ from its immediate predecessor $R^{(k-1)}$ in series (8.5). Let $r_{ij}^{(k)}$, the element in the i th row and j th column of matrix $R^{(k)}$, be equal to 1. This means that there exists a path from the i th vertex v_i to the j th vertex v_j with each intermediate vertex numbered not higher than k :

$$v_i, \text{ a list of intermediate vertices each numbered not higher than } k, v_j. \quad (8.6)$$

Two situations regarding this path are possible. In the first, the list of its intermediate vertices does not contain the k th vertex. Then this path from v_i to v_j has intermediate vertices numbered not higher than $k-1$, and therefore $r_{ij}^{(k-1)}$ is equal to 1 as well. The second possibility is that path (8.6) does contain the k th vertex v_k among the intermediate vertices. Without loss of generality, we may assume that v_k occurs only once in that list. (If it is not the case, we can create a new path from v_i to v_j with this property by simply eliminating all the vertices between the first and last occurrences of v_k in it.) With this caveat, path (8.6) can be rewritten as follows:

$$v_i, \text{ vertices numbered } \leq k-1, v_k, \text{ vertices numbered } \leq k-1, v_j.$$

The first part of this representation means that there exists a path from v_i to v_k with each intermediate vertex numbered not higher than $k-1$ (hence $r_{ik}^{(k-1)} = 1$), and the second part means that there exists a path from v_k to v_j with each intermediate vertex numbered not higher than $k-1$ (hence $r_{kj}^{(k-1)} = 1$).

What we have just proved is that if $r_{ij}^{(k)} = 1$, then either $r_{ij}^{(k-1)} = 1$ or both $r_{ik}^{(k-1)} = 1$ and $r_{kj}^{(k-1)} = 1$. It is easy to see that the converse of this assertion is also true. Thus, we have the following formula for generating the elements of matrix $R^{(k)}$ from the elements of matrix $R^{(k-1)}$:

$$r_{ij}^{(k)} = r_{ij}^{(k-1)} \text{ or } (r_{ik}^{(k-1)} \text{ and } r_{kj}^{(k-1)}) \quad (8.7)$$

Formula (8.7) is at the heart of Warshall's algorithm. This formula implies the following rule for generating elements of matrix $R^{(k)}$ from elements of matrix $R^{(k-1)}$, which is particularly convenient for applying Warshall's algorithm by hand:

- If an element r_{ij} is 1 in $R^{(k-1)}$, it remains 1 in $R^{(k)}$.
- If an element r_{ij} is 0 in $R^{(k-1)}$, it has to be changed to 1 in $R^{(k)}$ if and only if the element in its row i and column k and the element in its column j and row k are both 1's in $R^{(k-1)}$. (This rule is illustrated in Figure 8.3.)

As an example, the application of Warshall's algorithm to the digraph in Figure 8.2 is shown in Figure 8.4.

Here is a pseudocode of Warshall's algorithm.

ALGORITHM Warshall($A[1..n, 1..n]$)

//Implements Warshall's algorithm for computing the transitive closure

//Input: The adjacency matrix A of a digraph with n vertices

//Output: The transitive closure of the digraph

$R^{(0)} \leftarrow A$

for $k \leftarrow 1$ to n do

 for $i \leftarrow 1$ to n do

 for $j \leftarrow 1$ to n do

$R^{(k)}[i, j] \leftarrow R^{(k-1)}[i, j] \text{ or } (R^{(k-1)}[i, k] \text{ and } R^{(k-1)}[k, j])$

return $R^{(n)}$

Several observations need to be made about Warshall's algorithm. First, it is remarkably succinct, is it not? Still, its time efficiency is only in $\Theta(n^3)$. In fact, for sparse graphs represented by their adjacency lists, the traversal-based algorithm mentioned at the beginning of this section has a better asymptotic efficiency than

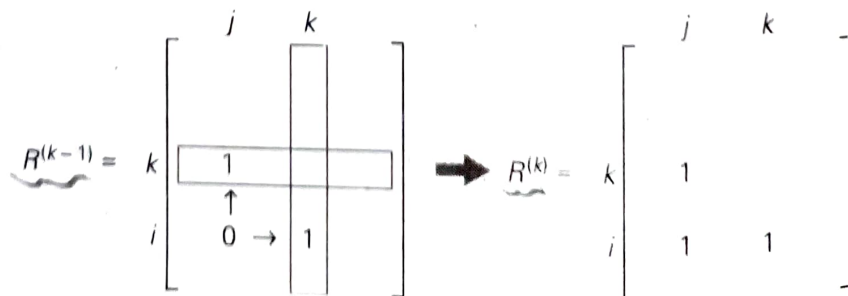


FIGURE 8.3 Rule for changing zeros in Warshall's algorithm

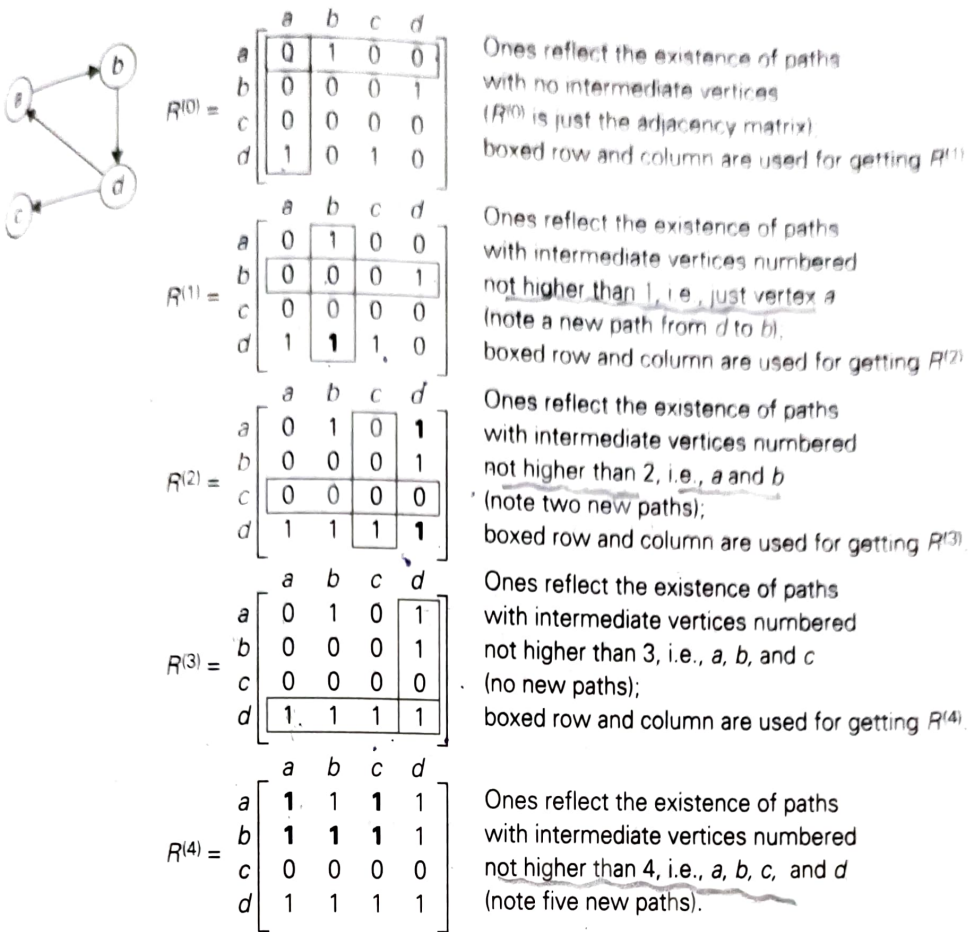


FIGURE 8.4 Application of Warshall's algorithm to the digraph shown. New ones are in bold.

Warshall's algorithm (why?). We can speed up the above implementation of Warshall's algorithm for some inputs by restructuring its innermost loop (see Problem 4 in the exercises). Another way to make the algorithm run faster is to treat matrix rows as bit strings and employ the bitwise *or* operation available in most modern computer languages.

As to the space efficiency of Warshall's algorithm, the situation is similar to that of the two earlier examples in this chapter: computing a Fibonacci number and computing a binomial coefficient. Although we used separate matrices for recording intermediate results of the algorithm, this is, in fact, unnecessary. (Problem 3 in the exercises asks you to find a way of avoiding this wasteful use of the computer memory.) Finally, we shall see below how the underlying idea of Warshall's algorithm can be applied to the more general problem of finding lengths of shortest paths in weighted graphs.

Floyd's Algorithm for the All-Pairs Shortest-Paths Problem

Given a weighted connected graph (undirected or directed), the *all-pairs shortest-paths problem* asks to find the distances (the lengths of the shortest paths) from

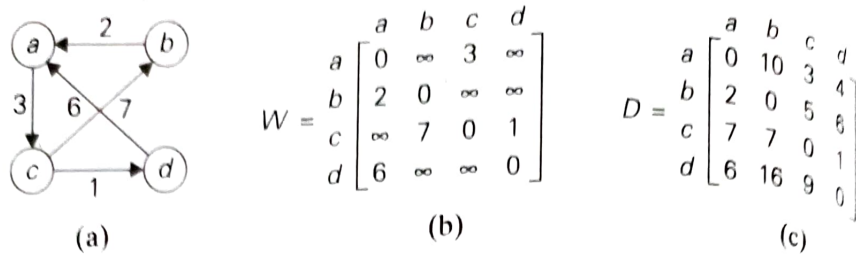


FIGURE 8.5 (a) Digraph. (b) Its weight matrix. (c) Its distance matrix.

each vertex to all other vertices. It is convenient to record the lengths of shortest paths in an n -by- n matrix D called the **distance matrix**: the element d_{ij} in the i th row and the j th column of this matrix indicates the length of the shortest path from the i th vertex to the j th vertex ($1 \leq i, j \leq n$). For an example, Figure 8.5.

We can generate the distance matrix with an algorithm that is very similar to Warshall's algorithm. It is called **Floyd's algorithm**, after its inventor R. Floyd [Flo62]. It is applicable to both undirected and directed weighted graphs providing that they do not contain a cycle of a negative length. (Of course, in the case of a directed graph, by a path or cycle we mean a directed path or a directed cycle.)

Floyd's algorithm computes the distance matrix of a weighted graph with n vertices through a series of n -by- n matrices:

$$D^{(0)}, \dots, D^{(k-1)}, D^{(k)}, \dots, D^{(n)}. \tag{8.8}$$

Each of these matrices contains the lengths of shortest paths with certain constraints on the paths considered for the matrix in question. Specifically, the element $d_{ij}^{(k)}$ in the i th row and the j th column of matrix $D^{(k)}$ ($k = 0, 1, \dots, n$) is equal to the length of the shortest path among all paths from the i th vertex to the j th vertex with each intermediate vertex, if any, numbered not higher than k . In particular, the series starts with $D^{(0)}$, which does not allow any intermediate vertices in its paths; hence, $D^{(0)}$ is nothing but the weight matrix of the graph. The last matrix in the series, $D^{(n)}$, contains the lengths of the shortest paths among all paths that can use all n vertices as intermediate and hence is nothing but the distance matrix being sought.

As in Warshall's algorithm, we can compute all the elements of each matrix $D^{(k)}$ from its immediate predecessor $D^{(k-1)}$ in series (8.8). Let $d_{ij}^{(k)}$ be the element in the i th row and the j th column of matrix $D^{(k)}$. This means that $d_{ij}^{(k)}$ is equal to the length of the shortest path among all paths from the i th vertex v_i to the j th vertex v_j with their intermediate vertices numbered not higher than k :

v_i , a list of intermediate vertices each numbered not higher than k , v_j .

We can partition all such paths into two disjoint subsets: those that do not use the k th vertex v_k as intermediate and those that do. Since the paths of the first subset

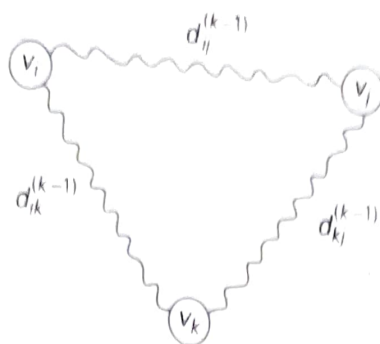


FIGURE 8.6 Underlying idea of Floyd's algorithm

have their intermediate vertices numbered not higher than $k - 1$, the shortest of them is, by definition of our matrices, of length $d_{ij}^{(k-1)}$.

What is the length of the shortest path in the second subset? If the graph does not contain a cycle of a negative length, we can limit our attention only to the paths in the second subset that use vertex v_k as their intermediate vertex exactly once (because visiting v_k more than once can only increase the path's length). All such paths have the following form:

v_i , vertices numbered $\leq k - 1$, v_k , vertices numbered $\leq k - 1$, v_j .

In other words, each of the paths is made up of a path from v_i to v_k with each intermediate vertex numbered not higher than $k - 1$ and a path from v_k to v_j with each intermediate vertex numbered not higher than $k - 1$. The situation is depicted symbolically in Figure 8.6.

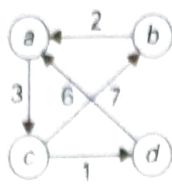
Since the length of the shortest path from v_i to v_k among the paths that use intermediate vertices numbered not higher than $k - 1$ is equal to $d_{ik}^{(k-1)}$ and the length of the shortest path from v_k to v_j among the paths that use intermediate vertices numbered not higher than $k - 1$ is equal to $d_{kj}^{(k-1)}$, the length of the shortest path among the paths that use the k th vertex is equal to $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$. Taking into account the lengths of the shortest paths in both subsets leads to the following recurrence:

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} \quad \text{for } k \geq 1, \quad d_{ij}^{(0)} = w_{ij}. \quad (8.10)$$

To put it another way, the element in the i th row and the j th column of the current distance matrix $D^{(k-1)}$ is replaced by the sum of the elements in the same row i and the k th column and in the same column j and the k th column if and only if the latter sum is smaller than its current value.

The application of Floyd's algorithm to the graph in Figure 8.5 is illustrated in Figure 8.7.

Here is a pseudocode of Floyd's algorithm. It takes advantage of the fact that the next matrix in sequence (8.8) can be written over its predecessor.



$$D^{(0)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & \infty & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & \infty & 0 \end{array}$$

Lengths of the shortest paths with no intermediate vertices ($D^{(0)}$ is simply the weight matrix)

$$D^{(1)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & 5 & \infty \\ c & \infty & 7 & 0 & 1 \\ d & 6 & \infty & 9 & 0 \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 1, i.e. just *a* (note two new shortest paths from *b* to *c* and from *d* to *c*)

$$D^{(2)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & \infty & 3 & \infty \\ b & 2 & 0 & 5 & \infty \\ c & 9 & 7 & 0 & 1 \\ d & 6 & \infty & 9 & 0 \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 2, i.e. *a* and *b* (note a new shortest path from *c* to *a*)

$$D^{(3)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 10 & 3 & 4 \\ b & 2 & 0 & 5 & 6 \\ c & 9 & 7 & 0 & 1 \\ d & 6 & 16 & 9 & 0 \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 3, i.e. *a*, *b*, and *c* (note four new shortest paths from *a* to *d*, from *b* to *d*, and from *d* to *a*)

$$D^{(4)} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 10 & 3 & 4 \\ b & 2 & 0 & 5 & 6 \\ c & 7 & 7 & 0 & 1 \\ d & 6 & 16 & 9 & 0 \end{array}$$

Lengths of the shortest paths with intermediate vertices numbered not higher than 4, i.e. *a*, *b*, *c*, and *d* (note a new shortest path from *c* to *a*)

FIGURE 8.7 Application of Floyd's algorithm to the graph shown. Updated elements are shown in bold.

ALGORITHM *Floyd*($W[1..n, 1..n]$)

//Implements Floyd's algorithm for the all-pairs shortest-paths problem

//Input: The weight matrix W of a graph with no negative-length cycle

//Output: The distance matrix of the shortest paths' lengths

$D \leftarrow W$ //is not necessary if W can be overwritten

for $k \leftarrow 1$ **to** n **do**

for $i \leftarrow 1$ **to** n **do**

for $j \leftarrow 1$ **to** n **do**

$D[i, j] \leftarrow \min\{D[i, j], D[i, k] + D[k, j]\}$

return D

Obviously, the time efficiency of Floyd's algorithm is cubic—as is the time efficiency of Warshall's algorithm. In the next chapter, we examine Dijkstra's algorithm—another method for finding shortest paths.

We finish this section with an important general comment. It deals with a general principle that underlines dynamic programming algorithms for optimization problems. Richard Bellman called it the *principle of optimality*. In terms