



$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad D^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

Figure 26.1 A directed graph and the sequence of matrices $D^{(m)}$ computed by SLOW-ALL-PAIRS-SHORTEST-PATHS. The reader may verify that $D^{(5)} = D^{(4)} \cdot W$ is equal to $D^{(4)}$, and thus $D^{(m)} = D^{(4)}$ for all $m \geq 4$.

FASTER-ALL-PAIRS-SHORTEST-PATHS(W)

```

1   $n \leftarrow \text{rows}[W]$ 
2   $D^{(1)} \leftarrow W$ 
3   $m \leftarrow 1$ 
4  while  $n - 1 > m$ 
5      do  $D^{(2m)} \leftarrow \text{EXTEND-SHORTEST-PATHS}(D^{(m)}, D^{(m)})$ 
6           $m \leftarrow 2m$ 
7  return  $D^{(m)}$ 

```

In each iteration of the **while** loop of lines 4–6, we compute $D^{(2m)} = (D^{(m)})^2$, starting with $m = 1$. At the end of each iteration, we double the value of m . The final iteration computes $D^{(n-1)}$ by actually computing $D^{(2m)}$ for some $n - 1 \leq 2m < 2n - 2$. By equation (26.3), $D^{(2m)} = D^{(n-1)}$. The next time the test in line 4 is performed, m has been doubled, so now $n - 1 < m$. the test fails, and the procedure returns the last matrix it

Computing the shortest-path weights bottom up

Based on recurrence (26.5), the following bottom-up procedure can be used to compute the values $d_{ij}^{(k)}$ in order of increasing values of k . Its input is an $n \times n$ matrix W defined as in equation (26.1). The procedure returns the matrix $D^{(n)}$ of shortest-path weights.

FLOYD-WARSHALL(W)

```
1  $n \leftarrow \text{rows}[W]$ 
2  $D^{(0)} \leftarrow W$ 
3 for  $k \leftarrow 1$  to  $n$ 
4     do for  $i \leftarrow 1$  to  $n$ 
5         do for  $j \leftarrow 1$  to  $n$ 
6              $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$ 
7 return  $D^{(n)}$ 
```

Figure 26.4 shows a directed graph and the matrices $D^{(k)}$ computed by the Floyd-Warshall algorithm.

The running time of the Floyd-Warshall algorithm is determined by the triply nested **for** loops of lines 3–6. Each execution of line 6 takes $O(1)$ time. The algorithm thus runs in time $\Theta(n^3)$. As in the final algorithm in Section 26.1, the code is tight, with no elaborate data structures, and so the constant hidden in the Θ -notation is small. Thus, the Floyd-Warshall algorithm is quite practical for even moderate-sized input graphs.

Constructing a shortest path

There are a variety of different methods for constructing shortest paths in the Floyd-Warshall algorithm. One way is to compute the matrix D of shortest-path weights and then construct the predecessor matrix Π from the D matrix. This method can be implemented to run in $O(n^3)$ time (Exercise 26.1-5). Given the predecessor matrix Π , the PRINT-ALL-PAIRS-SHORTEST-PATH procedure can be used to print the vertices on a given shortest path.

We can compute the predecessor matrix Π “on-line” just as the Floyd-Warshall algorithm computes the matrices $D^{(k)}$. Specifically, we compute a sequence of matrices $\Pi^{(0)}, \Pi^{(1)}, \dots, \Pi^{(n)}$, where $\Pi = \Pi^{(n)}$ and $\pi_{ij}^{(k)}$ is defined to be the predecessor of vertex j on a shortest path from vertex i with all intermediate vertices in the set $\{1, 2, \dots, k\}$.

We can give a recursive formulation of $\pi_{ij}^{(k)}$. When $k = 0$, a shortest path from i to j has no intermediate vertices at all. Thus,

) gives the final answer— $d_{ij}^{(n)} = \delta(i, j)$ for all intermediate vertices are in the set $\{1, 2, \dots, n\}$.

th weights bottom up

), the following bottom-up procedure can be used in order of increasing values of k . Its input is as in equation (26.1). The procedure returns k -path weights.

1 to n

$$\min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

ed graph and the matrices $D^{(k)}$ computed by the

e Floyd-Warshall algorithm is determined by the lines 3–6. Each execution of line 6 takes $O(1)$ runs in time $\Theta(n^3)$. As in the final algorithm in tight, with no elaborate data structures, and so Θ -notation is small. Thus, the Floyd-Warshall al for even moderate-sized input graphs.

th

fferent methods for constructing shortest paths orithm. One way is to compute the matrix D of then construct the predecessor matrix Π from od can be implemented to run in $O(n^3)$ time the predecessor matrix Π , the PRINT-ALL-PAIRS-re can be used to print the vertices on a given

edecessor matrix Π “on-line” just as the Floyd-ites the matrices $D^{(k)}$. Specifically, we compute a $\Pi^{(1)}, \dots, \Pi^{(n)}$, where $\Pi = \Pi^{(n)}$ and $\pi_{ij}^{(k)}$ is defined

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty. \end{cases} \quad (26.6)$$

For $k \geq 1$, if we take the path $i \rightsquigarrow k \rightsquigarrow j$, then the predecessor of j we choose is the same as the predecessor of j we chose on a shortest path from k with all intermediate vertices in the set $\{1, 2, \dots, k-1\}$. Otherwise, we choose the same predecessor of j that we chose on a shortest path from i with all intermediate vertices in the set $\{1, 2, \dots, k-1\}$. Formally, for $k \geq 1$,

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases} \quad (26.7)$$

We leave the incorporation of the $\Pi^{(k)}$ matrix computations into the FLOYD-WARSHALL procedure as Exercise 26.2-3. Figure 26.4 shows the sequence of $\Pi^{(k)}$ matrices that the resulting algorithm computes for the graph of Figure 26.1. The exercise also asks for the more difficult task of proving that the predecessor subgraph $G_{\pi,i}$ is a shortest-paths tree with root i . Yet another way to reconstruct shortest paths is given as Exercise 26.2-6.

Transitive closure of a directed graph

Given a directed graph $G = (V, E)$ with vertex set $V = \{1, 2, \dots, n\}$, we may wish to find out whether there is a path in G from i to j for all vertex pairs $i, j \in V$. The **transitive closure** of G is defined as the graph $G^* = (V, E^*)$, where

$$E^* = \{(i, j) : \text{there is a path from vertex } i \text{ to vertex } j \text{ in } G\}.$$

One way to compute the transitive closure of a graph in $\Theta(n^3)$ time is to assign a weight of 1 to each edge of E and run the Floyd-Warshall algorithm. If there is a path from vertex i to vertex j , we get $d_{ij} < \infty$. Otherwise, we get $d_{ij} = \infty$.

There is another, similar way to compute the transitive closure of G in $\Theta(n^3)$ time that can save time and space in practice. This method involves substitution of the logical operations \vee and \wedge for the arithmetic operations \min and $+$ in the Floyd-Warshall algorithm. For $i, j, k = 1, 2, \dots, n$, we define $t_{ij}^{(k)}$ to be 1 if there exists a path in graph G from vertex i to vertex j with all intermediate vertices in the set $\{1, 2, \dots, k\}$, and 0 otherwise. We construct the transitive closure $G^* = (V, E^*)$ by putting edge (i, j) into E^* if and only if $t_{ij}^{(n)} = 1$. A recursive definition of $t_{ij}^{(k)}$, analogous to