25.2 The Floyd-Warshall algorithm

In this section, we shall use a different dynamic-programming formulation to solve the all-pairs shortest-paths problem on a directed graph \( G = (V, E) \). The resulting algorithm, known as the **Floyd-Warshall algorithm**, runs in \( \Theta(V^3) \) time. As before, negative-weight edges may be present, but we assume that there are no negative-weight cycles. As in Section 25.1, we shall follow the dynamic-programming process to develop the algorithm. After studying the resulting algorithm, we shall present a similar method for finding the transitive closure of a directed graph.

The structure of a shortest path

In the Floyd-Warshall algorithm, we use a different characterization of the structure of a shortest path than we used in the matrix-multiplication-based all-pairs algorithms. The algorithm considers the "intermediate" vertices of a shortest path, where an **intermediate** vertex of a simple path \( p = (v_1, v_2, \ldots, v_l) \) is any vertex of \( p \) other than \( v_1 \) or \( v_l \), that is, any vertex in the set \( \{v_2, v_3, \ldots, v_{l-1} \} \).

The Floyd-Warshall algorithm is based on the following observation. Under our assumption that the vertices of \( G \) are \( V = \{1, 2, \ldots, n\} \), let us consider a subset \( \{1, 2, \ldots, k\} \) of vertices for some \( k \). For any pair of vertices \( i, j \in V \), consider all paths from \( i \) to \( j \) whose intermediate vertices are all drawn from \( \{1, 2, \ldots, k\} \), and let \( p \) be a minimum-weight path from among them. (Path \( p \) is simple.) The Floyd-Warshall algorithm exploits a relationship between path \( p \) and shortest paths from \( i \) to \( j \) with all intermediate vertices in the set \( \{1, 2, \ldots, k - 1\} \). The relationship depends on whether or not \( k \) is an intermediate vertex of path \( p \).

- If \( k \) is not an intermediate vertex of path \( p \), then all intermediate vertices of path \( p \) are in the set \( \{1, 2, \ldots, k - 1\} \). Thus, a shortest path from vertex \( i \) to vertex \( j \) with all intermediate vertices in the set \( \{1, 2, \ldots, k - 1\} \) is also a shortest path from \( i \) to \( j \) with all intermediate vertices in the set \( \{1, 2, \ldots, k\} \).

- If \( k \) is an intermediate vertex of path \( p \), then we break \( p \) down into \( i \overset{\ell_1}{\rightarrow} k \overset{\ell_2}{\rightarrow} j \) as shown in Figure 25.3. By Lemma 24.1, \( p_1 \) is a shortest path from \( i \) to \( k \) with all intermediate vertices in the set \( \{1, 2, \ldots, k\} \). Because vertex \( k \) is not an intermediate vertex of path \( p_1 \), we see that \( p_1 \) is a shortest path from \( i \) to \( k \) with all intermediate vertices in the set \( \{1, 2, \ldots, k - 1\} \). Similarly, \( p_2 \) is a shortest path from vertex \( k \) to vertex \( j \) with all intermediate vertices in the set \( \{1, 2, \ldots, k - 1\} \).
A recursive solution to the all-pairs shortest-paths problem

Based on the above observations, we define a recursive formulation of shortest path estimates that is different from the one in Section 25.1. Let \( d_{ij}^{(k)} \) be the weight of a shortest path from vertex \( i \) to vertex \( j \) for which all intermediate vertices are in the set \( \{1, 2, \ldots, k\} \). When \( k = 0 \), a path from vertex \( i \) to vertex \( j \) with no intermediate vertex numbered higher than 0 has no intermediate vertices at all. Such a path has at most one edge, and hence \( d_{ij}^{(0)} = w_{ij} \). A recursive definition following the above discussion is given by

\[
d_{ij}^{(2)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1. \\ \end{cases}
\]

Because for any path, all intermediate vertices are in the set \( \{1, 2, \ldots, n\} \), the matrix \( D^{(n)} = (d_{ij}^{(n)}) \) gives the final answer: \( d_{ij}^{(n)} = \delta(i, j) \) for all \( i, j \in V \).

Computing the shortest-path weights bottom up

Based on recurrence (25.5), 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 adjacency matrix \( W \) defined as in equation (25.1). 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. do \( d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) \)
7. return \( D^{(n)} \)
all intermediate vertices in \( 1, 2, \ldots, k - 1 \)

\[
\begin{array}{c}
\text{all intermediate vertices in } 1, 2, \ldots, k - 1 \\
\end{array}
\]

\[P_1 \rightarrow P_2 \rightarrow \cdots \rightarrow P_k\]

\(P_i\): all intermediate vertices in \( 1, 2, \ldots, k \)

**Figure 25.3** Path \( p \) is a shortest path from vertex \( i \) to vertex \( j \), and \( k \) is the highest-numbered intermediate vertex of \( p \). Path \( p_x \), the portion of path \( p \) from vertex \( i \) to vertex \( k \), has all intermediate vertices in the set \( 1, 2, \ldots, k - 1 \). The same holds for path \( p_y \) from vertex \( k \) to vertex \( j \).

A recursive solution to the all-pairs shortest-paths problem

Based on the above observations, we define a recursive formulation of shortest-path estimates that is different from the one in Section 25.1. Let \( d_{ij}^{(0)} \) be the weight of a shortest path from vertex \( i \) to vertex \( j \) for which all intermediate vertices are in the set \( 1, 2, \ldots, k \). When \( k = 0 \), a path from vertex \( i \) to vertex \( j \) with no intermediate vertex numbered higher than \( 0 \) has no intermediate vertices at all. Such a path has at most one edge, and hence \( d_{ij}^{(0)} = w_{ij} \). A recursive definition following the above discussion is given by

\[
d_{ij}^{(k)} = \min \left( d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right) \quad \text{if } k \geq 1. \tag{25.5}
\]

Because for any path, all intermediate vertices are in the set \( 1, 2, \ldots, n \), the matrix \( D^{(n)} = (d_{ij}^{(n)}) \) gives the final answer: \( d_{ij}^{(n)} = \delta(i, j) \) for all \( i, j \in V \).

Computing the shortest-path weights bottom up

Based on recurrence (25.5), 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 (25.1). The procedure returns the matrix \( D^{(n)} \) with shortest-path weights.

**FLOYD-WARSHALL(W)**

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

**Figure 25.4** The sequence of matrices \( D^{(k)} \) and \( \pi^{(k)} \) computed by the Floyd-Warshall algorithm for the graph in Figure 25.1.

Figure 25.4 shows the matrices \( D^{(k)} \) computed by the Floyd-Warshall algorithm for the graph in Figure 25.1.

The running time of the Floyd-Warshall algorithm is determined by the triply nested for loops of lines 3-6. Because each execution of line 6 takes \( O(1) \) time, the algorithm runs in time \( O(n^3) \). As in the final algorithm in Section 25.1, 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 $\Pi$ from the $D$ matrix. This method can be implemented to run in $O(n^3)$ time (Exercise 25.1-6). Given the predecessor matrix $\Pi$, 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 $\Pi$ “on-line” just as the Floyd-Warshall algorithm computes the matrices $D^{(k)}$. Specifically, we compute a sequence of matrices $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(k)}$, 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 $\{1, 2, \ldots, k\}$.

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,

$$
\pi_{ij}^{(0)} = \left\{ \begin{array}{ll}
\text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty , \\
i & \text{if } i \neq j \text{ and } w_{ij} < \infty .
\end{array} \right.
$$

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

$$
\pi_{ij}^{(k)} = \left\{ \begin{array}{ll}
\pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} , \\
\pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} .
\end{array} \right.
$$

We leave the incorporation of the $\Pi^{(k)}$ matrix computations into the Floyd-Warshall procedure as Exercise 25.2-3. Figure 25.4 shows the sequence of $\Pi^{(k)}$ matrices that the resulting algorithm computes for the graph of Figure 25.3. The exercise also asks for the more difficult task of proving that the transitive subgraph $G_{\pi,i}$ is a shortest-paths tree with root $i$. Yet another way to renumber shortest paths is given as Exercise 25.2-7.

Transitive closure of a directed graph

Given a directed graph $G = (V, E)$ with vertex set $V = \{1, 2, \ldots, n\}$, we wish to find out whether there is a path in $G$ from $i$ to $j$ for all vertex pairs $i, j$. The transitive closure of $G$ is defined as the graph $G^* = (V, E^*)$, where
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 \( \Pi \) from the \( D \) matrix. This method
can be implemented to run in \( O(n^3) \) time (Exercise 25.1-6). Given the predecessor
matrix \( \Pi \), the **PRINT-ALL-PATHS-SHORTEST-PATH** procedure can be used to print
the vertices on a given shortest path.

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

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

\[
\pi^{(0)}_{ij} = \begin{cases} 
NIL & \text{if } i = j \text{ or } w_{ij} = \infty, \\
1 & \text{if } i \neq j \text{ and } w_{ij} < \infty.
\end{cases}
\]  
(25.6)

For \( k \geq 1 \), if we take the path \( i \leadsto k \leadsto j \), where \( k \neq j \), 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 \( \{1, 2, \ldots, k - 1\} \). Otherwise,
we choose the same predecessor of \( j \) that we chose on a shortest path from \( i \) with
all intermediate vertices in the set \( \{1, 2, \ldots, k - 1\} \). Formally, for \( k \geq 1 \),

\[
\pi^{(k)}_{ij} = \begin{cases} 
\pi^{(k-1)}_{ik} & \text{if } d^{(k-1)}_{ij} = d^{(k-1)}_{ik} + d^{(k-1)}_{kj}, \\
\pi^{(k-1)}_{kj} & \text{if } d^{(k-1)}_{ij} > d^{(k-1)}_{ik} + d^{(k-1)}_{kj}.
\end{cases}
\]  
(25.7)

We leave the incorporation of the \( \Pi^{(k)} \) matrix computations into the FLOYD-
WARSHALL procedure as Exercise 25.2-3. Figure 25.4 shows the sequence of
\( \Pi^{(k)} \) matrices that the resulting algorithm computes for the graph of Figure 25.3.
The exercise also asks for the more difficult task of proving that the predecessor
subgraph \( G_{\pi^{(k)}} \) is a shortest-paths tree with root \( i \). Yet another way to recompute
shortest paths is given as Exercise 25.2.7.

Transitive closure of a directed graph

Given a directed graph \( G = (V, E) \) with vertex set \( V = \{1, 2, \ldots, n\} \), 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

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

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 \( d_{ij} < n \). 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 \( \lor \) (logical OR) and \( \land \) (logical AND) for the arithmetic
operations \( + \) and \( \times \) in the Floyd-Warshall algorithm. For \( i, j, k = 1, 2, \ldots, n \),
we define \( t^{(k)}_{ij} \) to be \( i \) if there exists a path in graph \( G \) from vertex \( i \) to vertex \( j \) with
all intermediate vertices in the set \( \{1, 2, \ldots, k\} \), and 0 otherwise. We construct
the transitive closure \( G^* = (V, E^*) \) by putting edge \((i, j) \) in \( E^* \) if and only if

\[
t^{(n)}_{ij} = 1. \text{ A recursive definition of } t^{(k)}_{ij}, \text{ analogous to recurrence (25.5), is}
\]

\[
t^{(0)}_{ij} = \begin{cases} 
0 & \text{if } i \neq j \text{ and } (i, j) \notin E, \\
1 & \text{if } i = j \text{ or } (i, j) \in E,
\end{cases}
\]

and for \( k \geq 1 \),

\[
t^{(k)}_{ij} = t^{(k-1)}_{ij} \lor (t^{(k-1)}_{ik} \land t^{(k-1)}_{kj}).
\]  
(25.8)

As in the Floyd-Warshall algorithm, we compute the matrices \( T^{(k)} = \{t^{(k)}_{ij}\} \)
in order of increasing \( k \).

**TRANSITIVE-CLOSURE**

1. \( n \leftarrow |V[G]| \)
2. for \( i \leftarrow 1 \) to \( n \)
3. do for \( j \leftarrow 1 \) to \( n \)
4. \( \) do for \( i \leftarrow 1 \) to \( n \)
5. \( \) do for \( j \leftarrow 1 \) to \( n \)
6. \( \) do for \( i \leftarrow 1 \) to \( n \)
7. \( \) do for \( j \leftarrow 1 \) to \( n \)
8. \( \) do for \( i \leftarrow 1 \) to \( n \)
9. \( \) do for \( j \leftarrow 1 \) to \( n \)
10. \( \) do for \( i \leftarrow 1 \) to \( n \)
11. return \( T^{(n)} \)

Figure 25.5 shows the matrices \( T^{(k)} \) computed by the **TRANSITIVE-CLOSURE**
procedure on a sample graph. The **TRANSITIVE-CLOSURE** procedure, like the Floyd-
Warshall algorithm, runs in \( \Theta(n^3) \) time. On some computers, though, logical
operations on single-bit values execute faster than arithmetic operations on integer
words of data. Moreover, because the direct transitive-closure algorithm uses only
boolean values rather than integer values, its space requirement is less than the

Floyd-Warshall algorithm's by a factor corresponding to the size of a word of computer storage.

Exercises

25.2-1
Run the Floyd-Warshall algorithm on the weighted, directed graph of Figure 25.5. Show the matrix \( D^{(k)} \) that results for each iteration of the outer loop.

25.2-2
Show how to compute the transitive closure using the technique of Section 25.1.

25.2-3
Modify the FLOYD-WARSHALL procedure to include computation of the \( \Pi^{(k)} \) matrices according to equations (25.6) and (25.7). Prove rigorously that for all \( i \neq j \), the predecessor subgraph \( G_{\pi,i} \) is a shortest-paths tree with root \( i \). (Hint: To show that \( G_{\pi,i} \) is acyclic, first show that \( \pi_{ij}^{(k)} = i \) implies \( d_{ij}^{(k)} \geq d_{ii}^{(k)} + wij \), according to the definition of \( \pi_{ij}^{(k)} \). Then, adapt the proof of Lemma 24.16.)

25.2-4
As it appears above, the Floyd-Warshall algorithm requires \( \Theta(n^3) \) space, since it computes \( d_{ij}^{(k)} \) for \( i, j, k = 1, 2, \ldots, n \). Show that the following procedure, which simply drops all the superscripts, is correct, and thus only \( \Theta(n^2) \) space is required.