COT 6405: Analysis of Algorithms

Giri NARASIMHAN

www.cs.fiu.edu/~giri/teach/6405F19.html

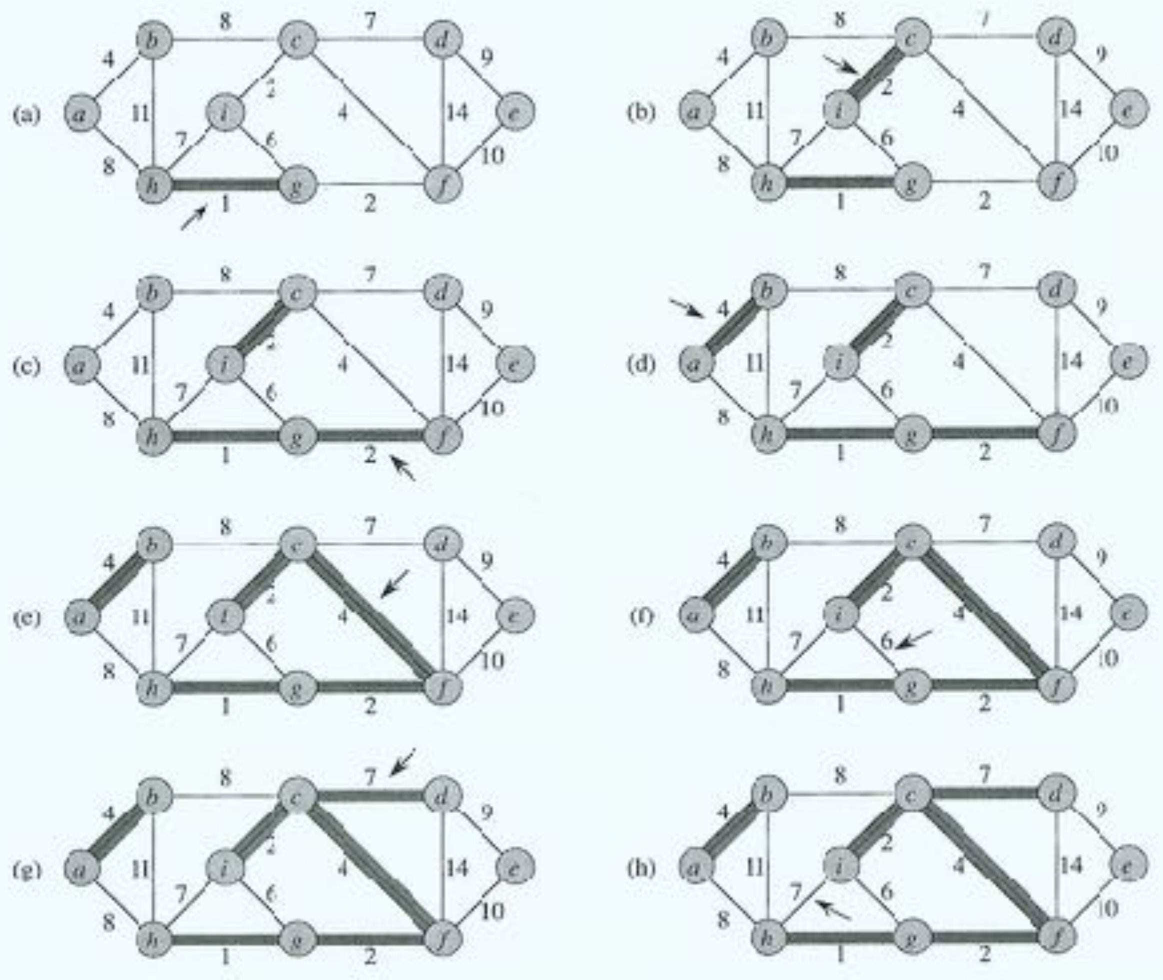
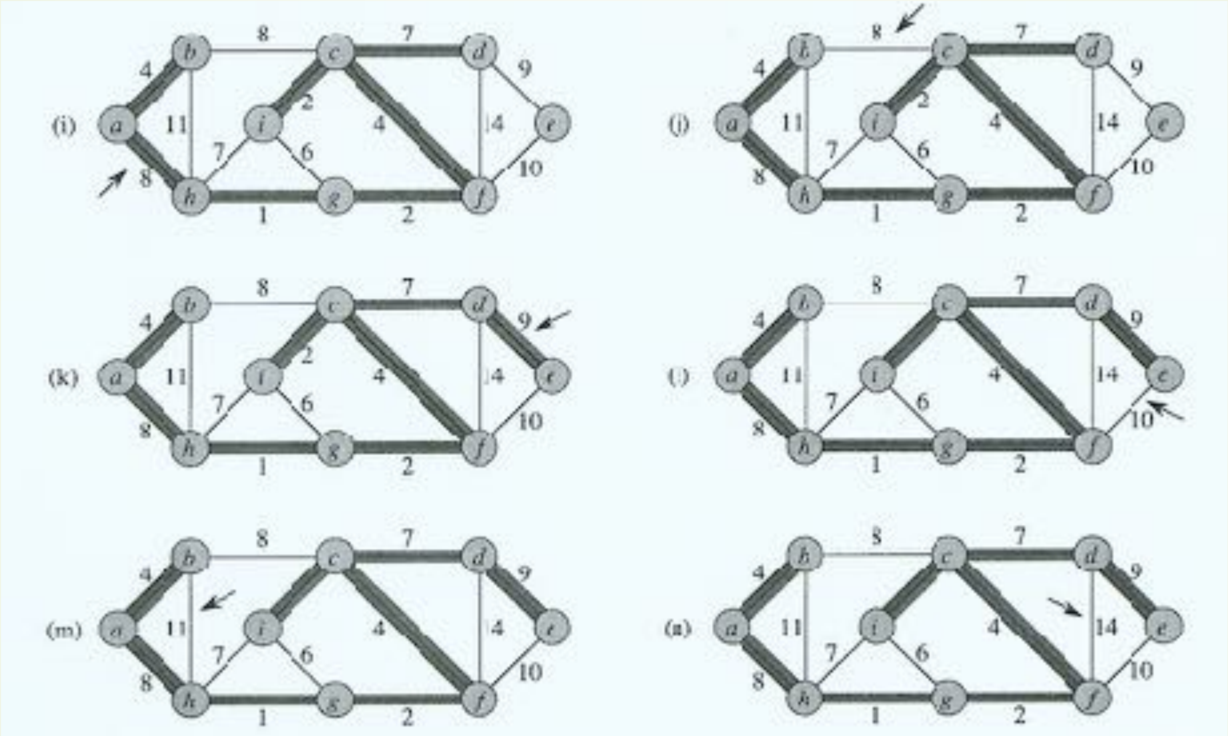


Figure 23.4 The execution of Kruskal's algorithm on the graph from Figure 23.1. Shaded edges belong to the forest A being grown. The edges are considered by the algorithm in sorted order by weight. An arrow points to the edge under consideration at each step of the algorithm. If the edge joins two distinct trees in the forest, it is added to the forest, thereby merging the two trees.


(i)
(j)
(k)
(l)
(m)
(n)

Minimum Spanning Tree

```
MST-KRUSKAL(G, w)
    A ← ∅
    for each vertex v ∈ V[G]
        do MAKE-SET(v)
    sort the edges of E by nondecreasing weight w
    for each edge (u, v) ∈ E, in order by nondecreasing weigh
        do if FIND-SET(u) ≠ FIND-SET(v)
            then A ← A ∪ {(u, v)}
                UNION(u, v)
    return A
```

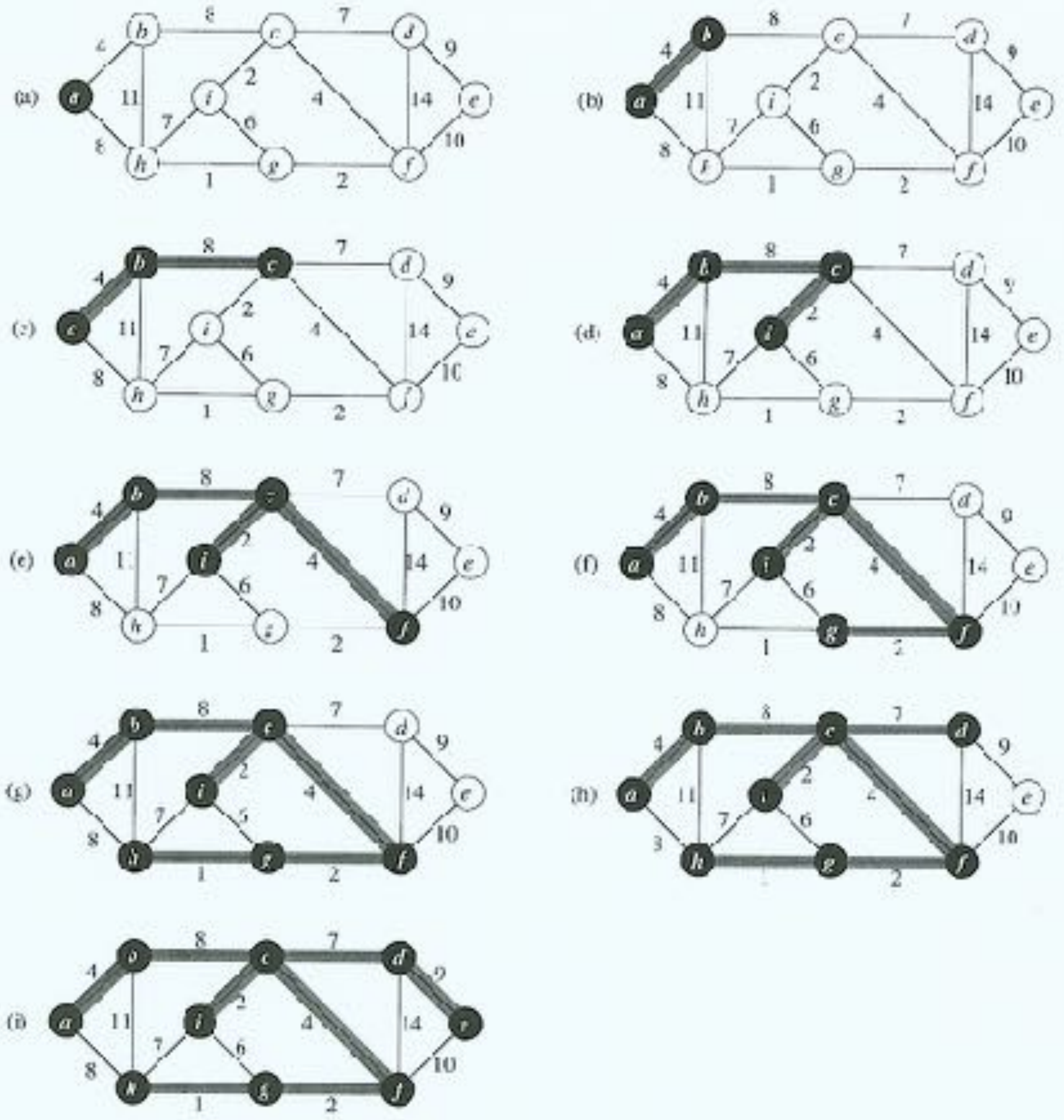


Figure 23.5 The execution of Prim's algorithm on the graph from Figure 23.1. The root vertex is a. Shaded edges are in the tree being grown, and the vertices in the tree are shown in black. At each step of the algorithm, the vertices in the tree determine a cut of the graph, and a light edge crossing the cut is added to the tree. In the second step, for example, the algorithm has a choice of adding either edge (b, c) or edge (a, h) to the tree since both are light edges crossing the cut.

```
MST-KRUSKAL(G, w)
A ← ∅
for each vertex v ∈ V[G]
    do MAKE-SET(v)
sort the edges of E by nondecreasing weight w
for each edge (u, v) ∈ E, in order by nondecreasing weigh
    do if FIND-SET(u) ≠ FIND-SET(v)
          then A ← A ∪ {(u, v)}
               UNION(u, v)
return A
```

```
MST-PRIM(G, w, r)
Q ← V[G]
for each u ∈ Q
    do key[u] ← ∞
key[r] ← 0
π[r] ← NIL
while Q ≠ ∅
      do u ← EXTRACT-MIN(Q)
         for each v ∈ Adj[u]
             do if v ∈ Q and w(u, v) < key[v]
                   then π[v] ← u
                        key[v] ← w(u, v)
```

Proof of Correctness: MST Algorithms

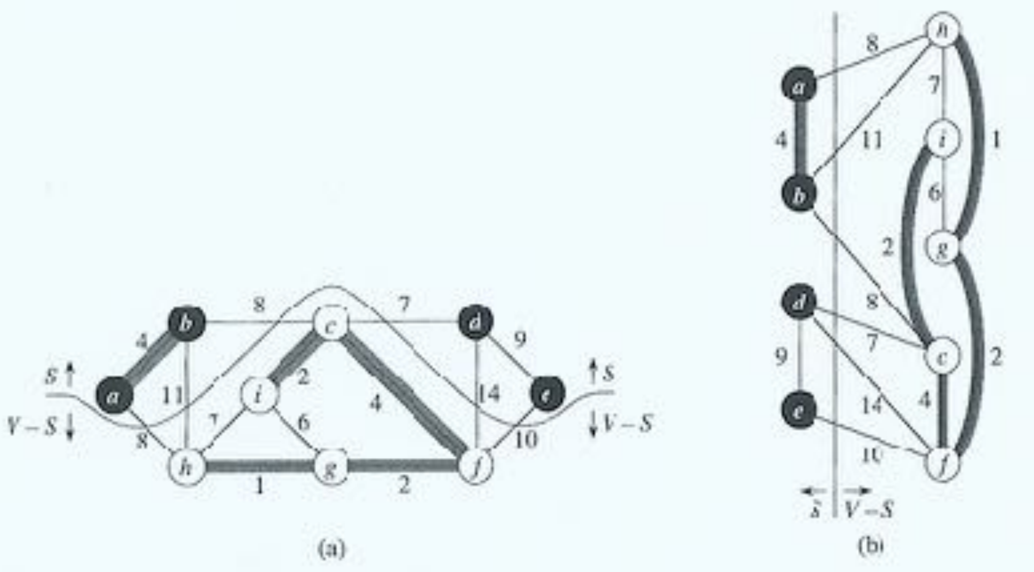


Figure 23.2 Two ways of viewing a cut $(S, V - S)$ of the graph from Figure 23.1. **(a)** The vertices in the set S are shown in black, and those in $V - S$ are shown in white. The edges crossing the cut are those connecting white vertices with black vertices. The edge (d, c) is the unique light edge crossing the cut. A subset A of the edges is shaded; note that the cut $(S, V - S)$ respects A, since no edge of A crosses the cut. **(b)** The same graph with the vertices in the set S on the left and the vertices in the set $V - S$ on the right. An edge crosses the cut if it connects a vertex on the left with a vertex on the right.

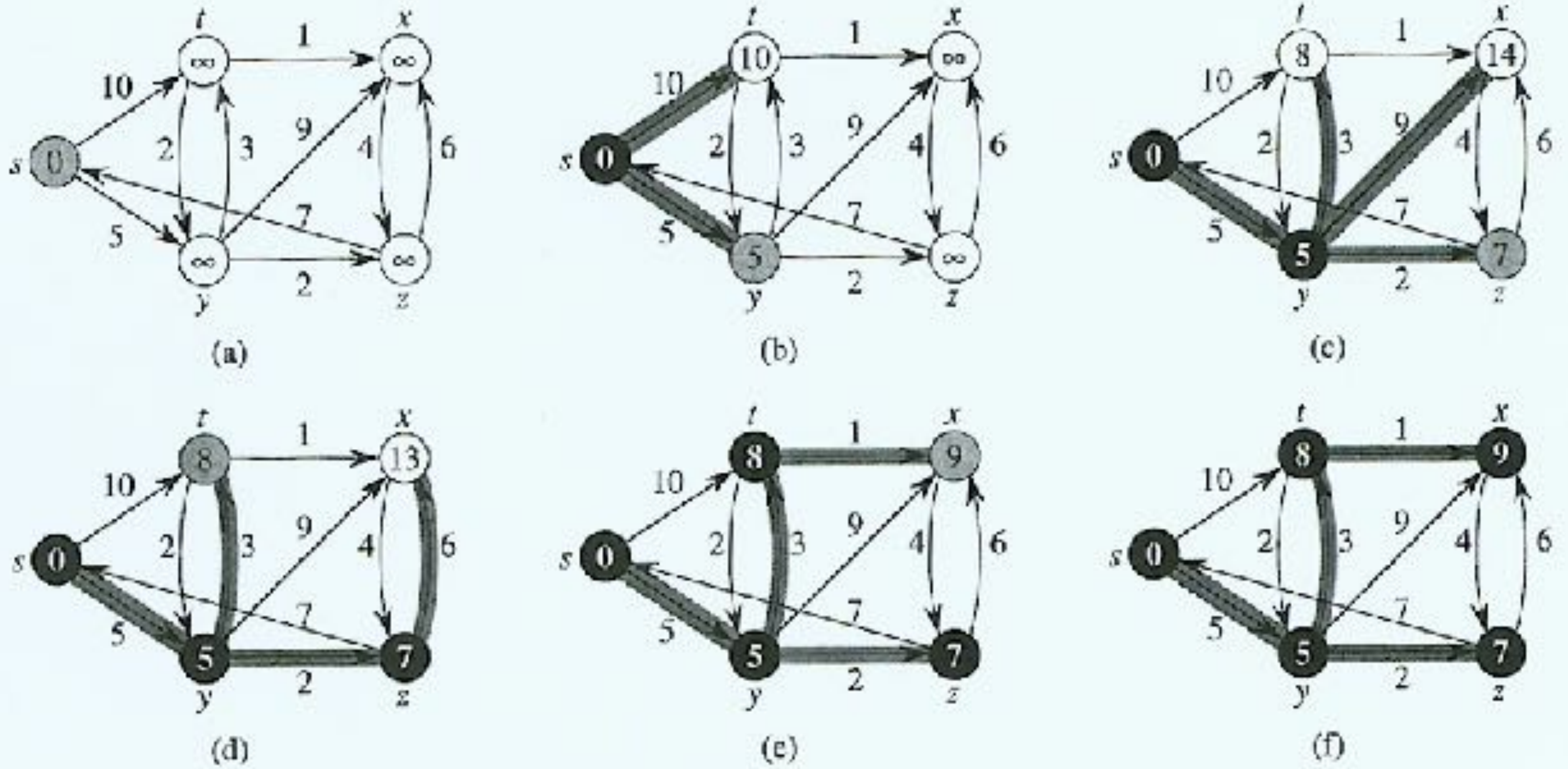


Figure 24.6 The execution of Dijkstra's algorithm. The source s is the leftmost vertex. The shortest-path estimates are shown within the vertices, and shaded edges indicate predecessor values. Black vertices are in the set S, and white vertices are in the min-priority queue $Q = V - S$. **(a)** The situation just before the first iteration of the **while** loop of lines 4–8. The shaded vertex has the minimum d value and is chosen as vertex u in line 5. **(b)–(f)** The situation after each successive iteration of the **while loop.** The shaded vertex in each part is chosen as vertex u in line 5 of the next iteration. The d and π values shown in part (f) are the final values.

Dijkstra's SSSP Algorithm

```
DIJKSTRA(G, w, s)
1.   // INITIALIZE-SINGLE-SOURCE(G, s)
        for each vertex v ∈ V[G]
              do d[v] ← ∞
                    π[v] ← NIL
        d[s] ← 0
2.   S ← ∅
3.   Q ← V[G]
4.   while Q ≠ ∅
5.           do u ← EXTRACT-MIN(Q)
6.                 S ← S ∪ {u}
7.                 for each v ∈ Adj[u]
8.                       do // RELAX(u, v, w)
                              if d[v] > d[u] + w(u, v)
                                 then d[v] ← d[u] + w(u, v)
                                      π[v] ← u
```

```
DIJKSTRA(G, w, s)
1.  // INITIALIZE-SINGLE-SOURCE(G, s)
        for each vertex v ∈ V[G]
            do d[v] ← ∞
               π[v] ← NIL
        d[s] ← 0
2.  S ← ∅
3.  Q ← V[G]
4.  while Q ≠ ∅
5.      do u ← EXTRACT-MIN(Q)
6.         S ← S ∪ {u}
7.         for each v ∈ Adj[u]
8.             do // RELAX(u, v, w)
                  if d[v] > d[u] + w(u, v)
                     then d[v] ← d[u] + w(u, v)
                          π[v] ← u
```

```
MST-PRIM(G, w, r)
Q ← V[G]
for each u ∈ Q
    do key[u] ← ∞
key[r] ← 0
π[r] ← NIL
while Q ≠ ∅
    do u ← EXTRACT-MIN(Q)
       for each v ∈ Adj[u]
           do if v ∈ Q and w(u, v) < key[v]
                 then π[v] ← u
                      key[v] ← w(u, v)
```

Analysis of Dijkstra's Algorithm

- O(n) calls to INSERT, EXTRACT-MIN
- O(m) calls to DECREASE-KEY

Approach	Insert	Dec-Key	Extract-Min	Total
PQ in Arrays	$O(1)$	$O(1)$	$O(n)$	$O(n^2)$
Heaps	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O((m+n)\log n)$
Fibonacci Heaps	$O(1)$*	$O(1)$*	$O(\log n)$*	$O(m + n \log n)$*

* Amortized Time Complexity

SSSP Algorithms

- Dijkstra's algorithm (only non-negative edges allowed)
  - Best: O(m + n log n)
- Bellman-Ford algorithm (allows non-negative edges, but less efficient)
  - O(mn) time complexity

All Pairs Shortest Path Algorithm

- Invoke Dijkstra's SSSP algorithm n times.
- Or use dynamic programming. How?

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

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

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

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

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

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

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 14.38
Worst-case running times of various graph algorithms

TYPE OF GRAPH PROBLEM	RUNNING TIME	COMMENTS
Unweighted	$O(\|E\|)$	Breadth-first search
Weighted, no negative edges	$O(\|E\| \log \|V\|)$	Dijkstra's algorithm
Weighted, negative edges	$O(\|E\| \cdot \|V\|)$	Bellman–Ford algorithm
Weighted, acyclic	$O(\|E\|)$	Uses topological sort