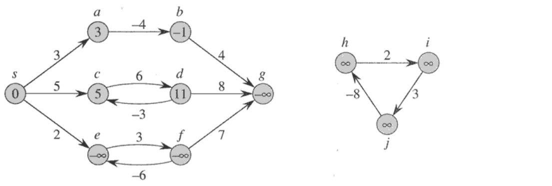


Figure 24.1 Negative edge weights in a directed graph. Shown within each vertex is its shortest-path weight from source s. Because vertices e and f form a negative-weight cycle reachable from s, they have shortest-path weights of $-\infty$. Because vertex g is reachable from a vertex whose shortest-path weight is $-\infty$, it, too, has a shortest-path weight of $-\infty$. Vertices such as h, i, and j are not reachable from s, and so their shortest-path weights are ∞, even though they lie on a negative-weight cycle.

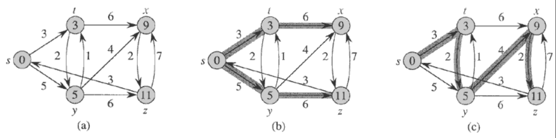


Figure 24.2 **(a)** A weighted, directed graph with shortest-path weights from source s. **(b)** The shaded edges form a shortest-paths tree rooted at the source s. **(c)** Another shortest-paths tree with the same root.

INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex $v \in V[G]$
  do $d[v] \leftarrow \infty$
    $\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$

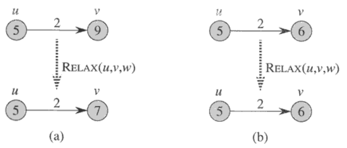


Figure 24.3 Relaxation of an edge (u, v) with weight $w(u, v) = 2$. The shortest-path estimate of each vertex is shown within the vertex. **(a)** Because $d[v] > d[u] + w(u, v)$ prior to relaxation, the value of $d[v]$ decreases. **(b)** Here, $d[v] \le d[u] + w(u, v)$ before the relaxation step, and so $d[v]$ is unchanged by relaxation.

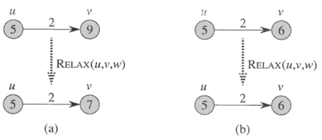


Figure 24.3 Relaxation of an edge (u, v) with weight $w(u, v) = 2$. The shortest-path estimate of each vertex is shown within the vertex. **(a)** Because $d[v] > d[u] + w(u, v)$ prior to relaxation, the value of $d[v]$ decreases. **(b)** Here, $d[v] \le d[u] + w(u, v)$ before the relaxation step, and so $d[v]$ is unchanged by relaxation.

RELAX(u, v, w)

if $d[v] > d[u] + w(u, v)$
  then $d[v] \leftarrow d[u] + w(u, v)$
    $\pi[v] \leftarrow u$

BELLMAN-FORD(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)
for $i \leftarrow 1$ **to** $|V[G]| - 1$
    do for each edge $(u, v) \in E[G]$
        do RELAX(u, v, w)
for each edge $(u, v) \in E[G]$
    do if $d[v] > d[u] + w(u, v)$
        then return FALSE
return TRUE

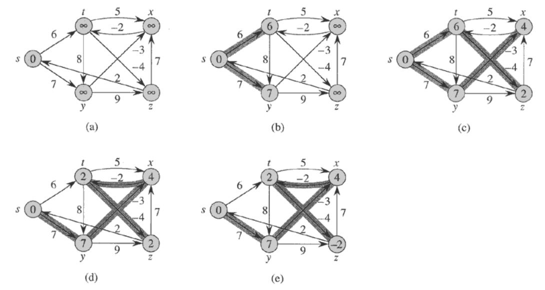


Figure 24.4 The execution of the Bellman-Ford algorithm. The source is vertex s. The d values are shown within the vertices, and shaded edges indicate predecessor values: if edge (u, v) is shaded, then $\pi[v] = u$. In this particular example, each pass relaxes the edges in the order $(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$. **(a)** The situation just before the first pass over the edges. **(b)–(e)** The situation after each successive pass over the edges. The d and π values in part (e) are the final values. The Bellman-Ford algorithm returns TRUE in this example.

DIJKSTRA(G, w, s)

1 INITIALIZE-SINGLE-SOURCE(G, s)
2 $S \leftarrow \emptyset$
3 $Q \leftarrow V[G]$
4 **while** $Q \neq \emptyset$
5   **do** $u \leftarrow$ EXTRACT-MIN(Q)
6         $S \leftarrow S \cup \{u\}$
7   **for** each vertex $v \in Adj[u]$
8   **do** RELAX(u, v, w)

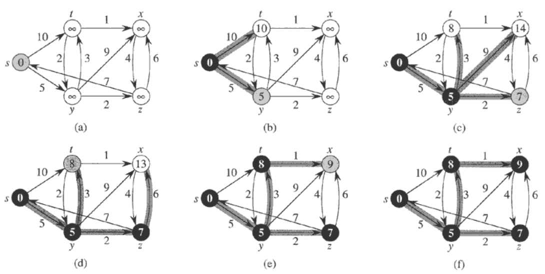


Figure 24.6 The execution of Dijkstra's algorithm. The source s is the leftmost vertex. The shortest-path estimates are shown within the vertices, and shaded edges indicate predecessor values. Black vertices are in the set S, and white vertices are in the min-priority queue $Q = V - S$. **(a)** The situation 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.

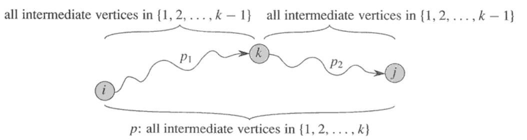


Figure 25.3 Path p is a shortest path from vertex i to vertex j, and k is the highest-numbered intermediate vertex of p. Path p_1, the portion of path p from vertex i to vertex k, has all intermediate vertices in the set $\{1, 2, \ldots, k-1\}$. The same holds for path p_2 from vertex k to vertex j.

FLOYD-WARSHALL(W)

$n \leftarrow rows[W]$
$D^{(0)} \leftarrow W$
for $k \leftarrow 1$ **to** n
    do for $i \leftarrow 1$ **to** n
        do for $j \leftarrow 1$ **to** n
            do $d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
return $D^{(n)}$

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

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

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

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

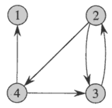
$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1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

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

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.

TRANSITIVE-CLOSURE(G)

$n \leftarrow |V[G]|$
for $i \leftarrow 1$ **to** n
  do for $j \leftarrow 1$ **to** n
    do if $i = j$ or $(i, j) \in E[G]$
      then $t_{ij}^{(0)} \leftarrow 1$
      else $t_{ij}^{(0)} \leftarrow 0$
for $k \leftarrow 1$ **to** n
  do for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
      do $t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)$
return $T^{(n)}$



$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Figure 25.5 A directed graph and the matrices $T^{(k)}$ computed by the transitive-closure algorithm.

FLOYD-WARSHALL$'(W)$

$n \leftarrow rows[W]$
$D \leftarrow W$
for $k \leftarrow 1$ **to** n
  do for $i \leftarrow 1$ **to** n
    do for $j \leftarrow 1$ **to** n
      do $d_{ij} \leftarrow \min(d_{ij}, d_{ik} + d_{kj})$
return D

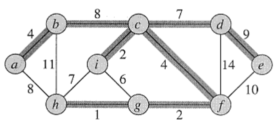


Figure 23.1 A minimum spanning tree for a connected graph. The weights on edges are shown, and the edges in a minimum spanning tree are shaded. The total weight of the tree shown is 37. This minimum spanning tree is not unique: removing the edge (b, c) and replacing it with the edge (a, h) yields another spanning tree with weight 37.

GENERIC-MST(G, w)

$A \leftarrow \emptyset$
while A does not form a spanning tree
  do find an edge (u, v) that is safe for A
    $A \leftarrow A \cup \{(u, v)\}$
return A

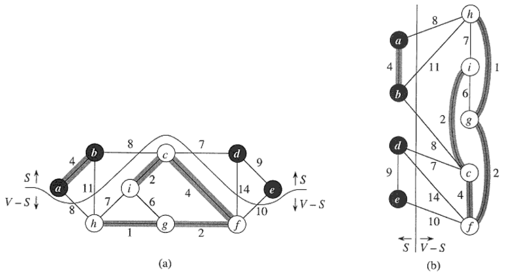


Figure 23.2 Two ways of viewing a cut $(S, V-S)$ of the graph from Figure 23.1. **(a)** The vertices in the set S are shown in black, and those in $V-S$ are shown in white. The edges crossing the cut are those connecting white vertices with black vertices. The edge (d, c) is the unique light edge crossing 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.

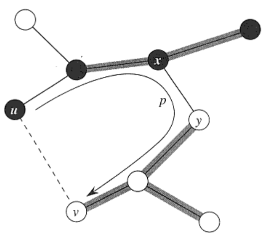


Figure 23.3 The proof of Theorem 23.1. The vertices in S are black, and the vertices in $V-S$ are white. The edges in the minimum spanning tree T are shown, but the edges in the graph G are not. The edges in A are shaded, and (u, v) is a light edge crossing the cut $(S, V-S)$. The edge (x, y) is an edge on the unique path p from u to v in T. A minimum spanning tree T' that contains (u, v) is formed by removing the edge (x, y) from T and adding the edge (u, v).

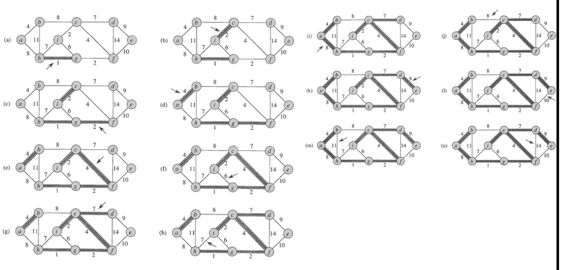


Figure 23.4 The execution of Kruskal's algorithm on the graph from Figure 23.1. Shaded edges belong to the forest A being grown. The edges are considered by the algorithm in sorted order by weight. An arrow points to the edge under consideration at each step of the algorithm. If the edge joins two distinct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

MST-KRUSKAL(G, w)

$A \leftarrow \emptyset$
for each vertex $v \in V[G]$
  do MAKE-SET(v)
sort the edges of E into nondecreasing order by weight w
for each edge $(u, v) \in E$, taken in nondecreasing order by weight
  do if FIND-SET$(u) \neq$ FIND-SET(v)
    then $A \leftarrow A \cup \{(u, v)\}$
      UNION(u, v)
return A

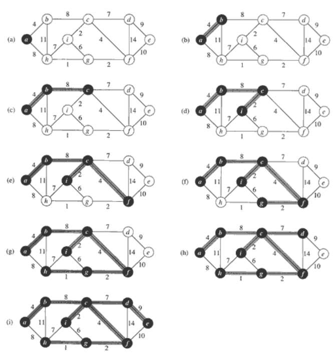


Figure 23.5 The execution of Prim's algorithm on the graph from Figure 23.1. The root vertex is a. Shaded edges are in the tree being grown, and the vertices in the tree are shown in black. At each step of the algorithm, the vertices in the tree determine a cut of the graph, and a light edge crossing 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-PRIM(G, w, r)

for each $u \in V[G]$
  do $key[u] \leftarrow \infty$
    $\pi[u] \leftarrow$ NIL
$key[r] \leftarrow 0$
$Q \leftarrow V[G]$
while $Q \neq \emptyset$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    for each $v \in Adj[u]$
      do if $v \in Q$ and $w(u, v) < key[v]$
        then $\pi[v] \leftarrow u$
          $key[v] \leftarrow w(u, v)$

```
MST-REDUCE(G, orig, c, T)
for each v ∈ V[G]
    do mark[v] ← FALSE
       MAKE-SET(v)
for each u ∈ V[G]
    do if mark[u] = FALSE
          then choose v ∈ Adj[u] such that c[u, v] is minimized
               UNION(u, v)
               T ← T ∪ {orig[u, v]}
               mark[u] ← mark[v] ← TRUE
V[G′] ← {FIND-SET(v) : v ∈ V[G]}
E[G′] ← ∅
for each (x, y) ∈ E[G]
    do u ← FIND-SET(x)
       v ← FIND-SET(y)
       if (u, v) ∉ E[G′]
          then E[G′] ← E[G′] ∪ {(u, v)}
               orig′[u, v] ← orig[x, y]
               c′[u, v] ← c[x, y]
          else if c[x, y] < c′[u, v]
                  then orig′[u, v] ← orig[x, y]
                       c′[u, v] ← c[x, y]
construct adjacency lists Adj for G′
return G′, orig′, c′, and T
```