Connectivity

- A (simple) undirected graph is <u>connected</u> if there exists a path between every pair of vertices.

- If a graph is not connected, then $G'(V',E')$ is a <u>connected component</u> of the graph $G(V,E)$ if V' is a <u>maximal</u> subset of vertices from V that induces a connected subgraph. (What is the meaning of <u>maximal</u>?)

- The connected components of a graph correspond to a <u>partition</u> of the set of the vertices. (What is the meaning of <u>partition</u>?)

- How to compute all the connected components?
  - Use DFS or BFS.

Biconnectivity: Generalizing Connectivity

- A tree is a <u>minimally connected</u> graph.
- Removing a vertex from a connected graph may make it disconnected.
- A graph is <u>biconnected</u> if removing a single vertex does not disconnect the graph.
- Alternatively, a graph is <u>biconnected</u> if for every pair of vertices there exists at least 2 disjoint paths between them.
- A graph is <u>k-connected</u> if for every pair of vertices there exists at least k disjoint paths between them. Alternatively, removal of any k-1 vertices does not disconnect the graph.

Biconnected Components

- If a graph is not biconnected, it can be decomposed into biconnected components.

- An <u>articulation point</u> is a vertex whose removal disconnects the graph.

- **Claim:** If a graph is not biconnected, it must have an articulation point. Proof?

- A biconnected component of a simple undirected graph G(V,E) is a <u>maximal</u> set of edges from E that induces a biconnected subgraph.
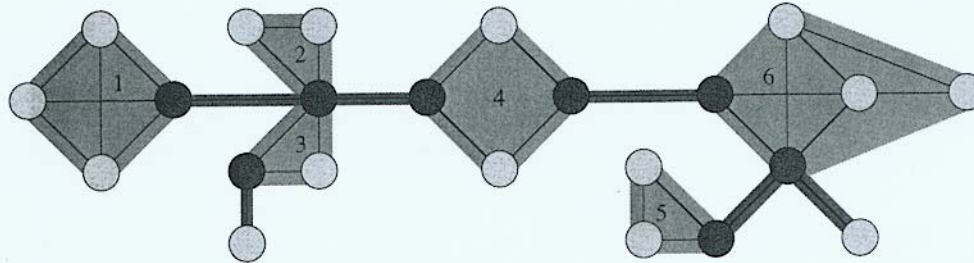
Biconnected Components

Figure 22.10 The articulation points, bridges, and biconnected components of a connected, undirected graph for use in Problem 22-2. The articulation points are the heavily shaded vertices, the bridges are the heavily shaded edges, and the biconnected components are the edges in the shaded regions, with a *bcc* numbering shown.

BCC(G, u) // Compute the biconnected components of G
　　　　// starting from vertex u
1.　Color[u] ← GRAY
2.　Low[u] ← d[u] ← Time ← Time + 1
3.　Put u on stack S
4.　**for** each v ∈ Adj[u] **do**
5.　　　Put edge (u,v) on stack S
6.　　　**if** (v ≠ π[u]) **then**
7.　　　　**if** (color[v] = WHITE) **then**
8.　　　　　π[v] ← u
9.　　　　　BCC(G, v)
10.　　　　**if** (Low[v] > d[u]) **then** // u is an articul. pt.
11.　　　　　　// Output next biconnected component
12.　　　　　　Pop S until u is reached
13.　　　　　　Push u back on S
14.　　　　　Low[u] = min { Low[u], Low[v] }
15.　　　**else** Low[u] = min { Low[u], d[v] } // back edge
16. color[u] ← BLACK
17. F[u] ← Time ← Time + 1

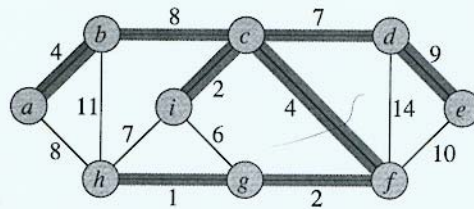# BCC Example
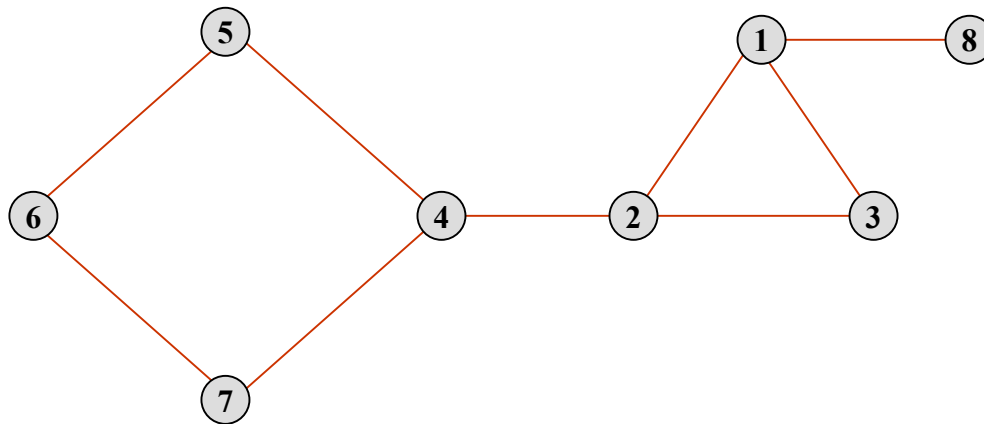
Minimum Spanning Tree

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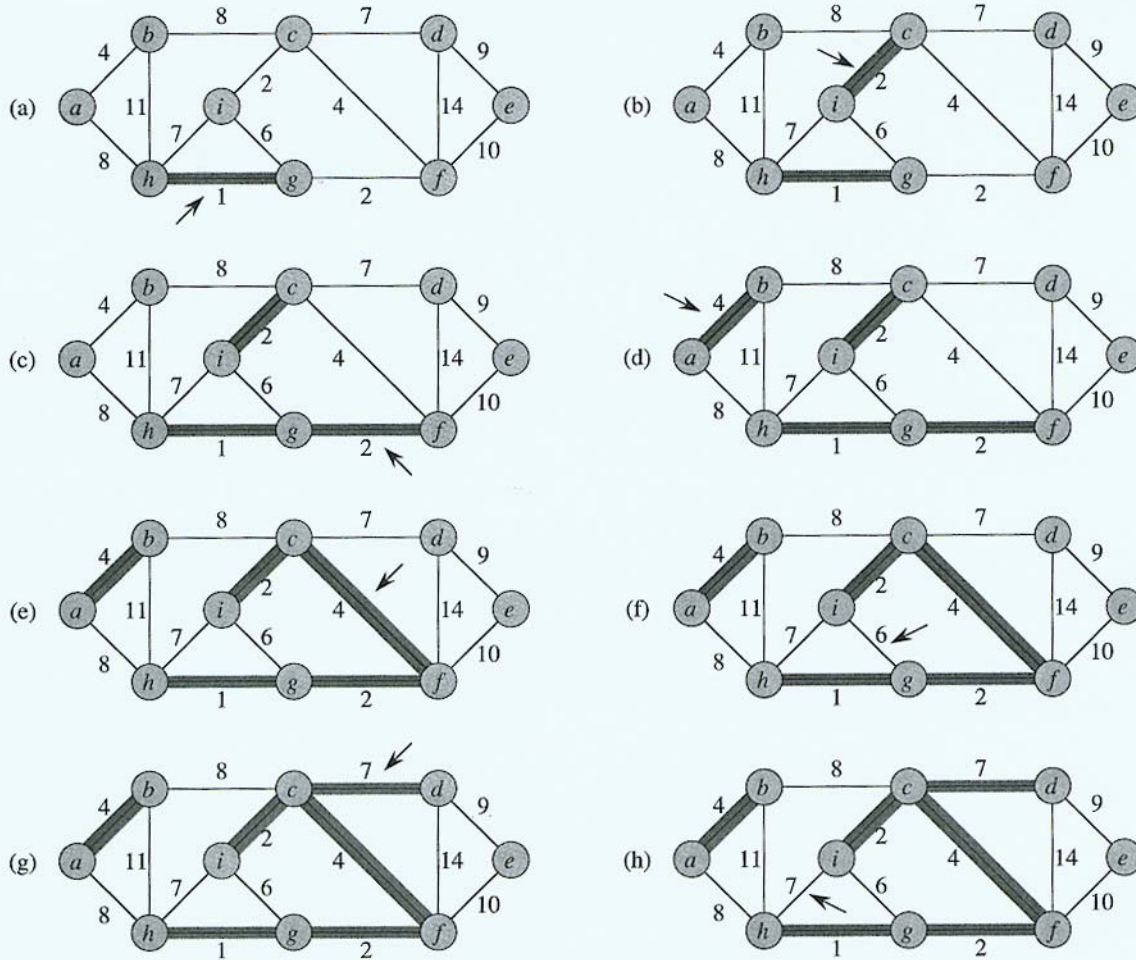
Figure 23.4 The execution of Kruskal's algorithm on the graph from Figure 23.1. Shaded edges belong to the forest A being grown. The edges are considered by the algorithm in sorted order by weight. An arrow points to the edge under consideration at each step of the algorithm. If the edge joins two distinct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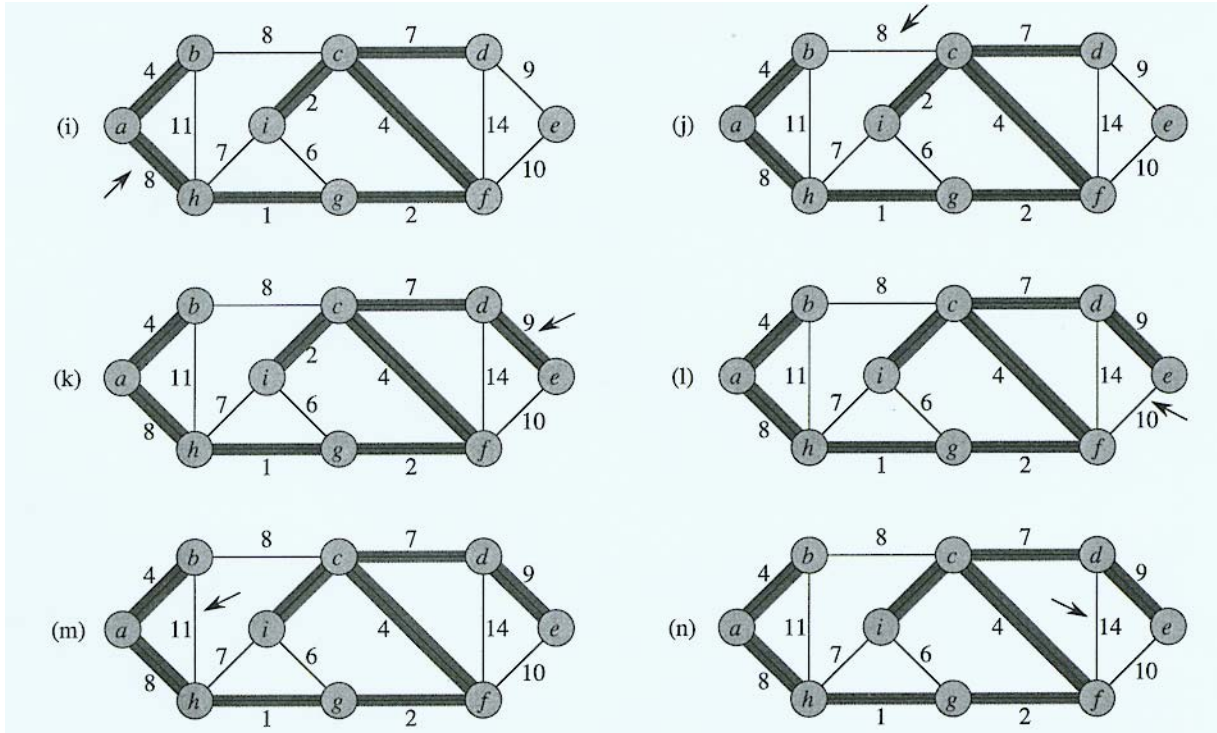
Minimum Spanning Tree

$\text{MST-KRUSKAL}(G, w)$

1. $A \leftarrow \emptyset$
2.   **for** each vertex $v \in V[G]$
3.   **do** $\text{MAKE-SET}(v)$
4.   sort the edges of E by nondecreasing weight w
5.   **for** each edge $(u, v) \in E$, in order by nondecreasing weigh
6.   **do if** $\text{FIND-SET}(u) \neq \text{FIND-SET}(v)$
7.   **then** $A \leftarrow A \cup \{(u, v)\}$
8. $\text{UNION}(u, v)$
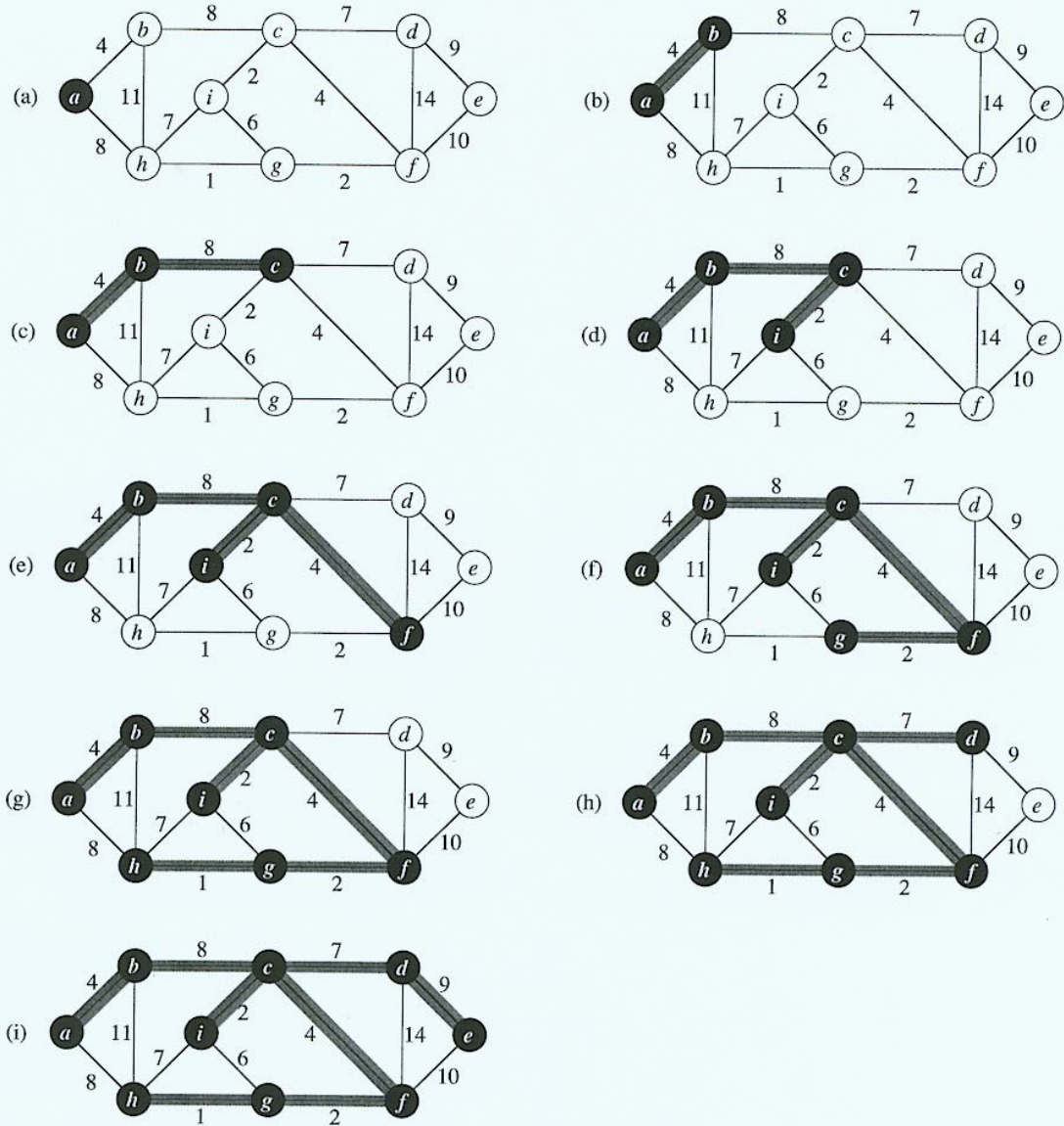9.   **return** A

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)

1. $A \leftarrow \emptyset$
2. **for** each vertex $v \in V[G]$
3.   **do** MAKE-SET(v)
4. sort the edges of E by nondecreasing weight w
5. **for** each edge $(u, v) \in E$, in order by nondecreasing weight
6.   **do if** FIND-SET$(u) \neq$ FIND-SET(v)
7.   **then** $A \leftarrow A \cup \{(u, v)\}$
8.   UNION(u, v)
9. **return** A

MST-PRIM(G, w, r)

1. $Q \leftarrow V[G]$
2. **for** each $u \in Q$
3.   **do** $key[u] \leftarrow \infty$
4. $key[r] \leftarrow 0$
5. $\pi[r] \leftarrow$ NIL
6. **while** $Q \neq \emptyset$
7.   **do** $u \leftarrow$ EXTRACT-MIN(Q)
8.   **for** each $v \in Adj[u]$
9.   **do if** $v \in Q$ and $w(u, v) < key[v]$
10.   **then** $\pi[v] \leftarrow u$
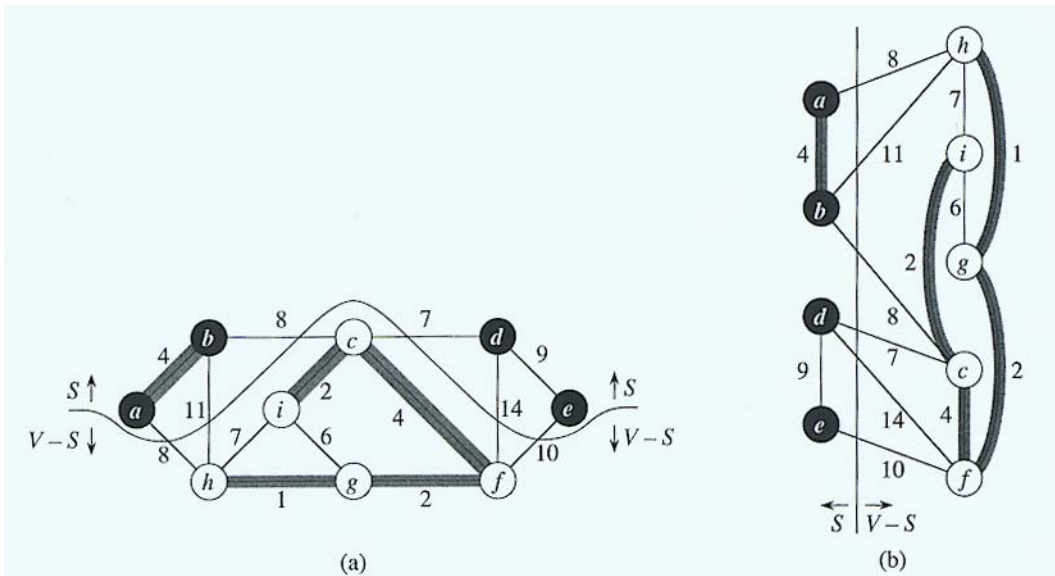11. $key[v] \leftarrow w(u, v)$

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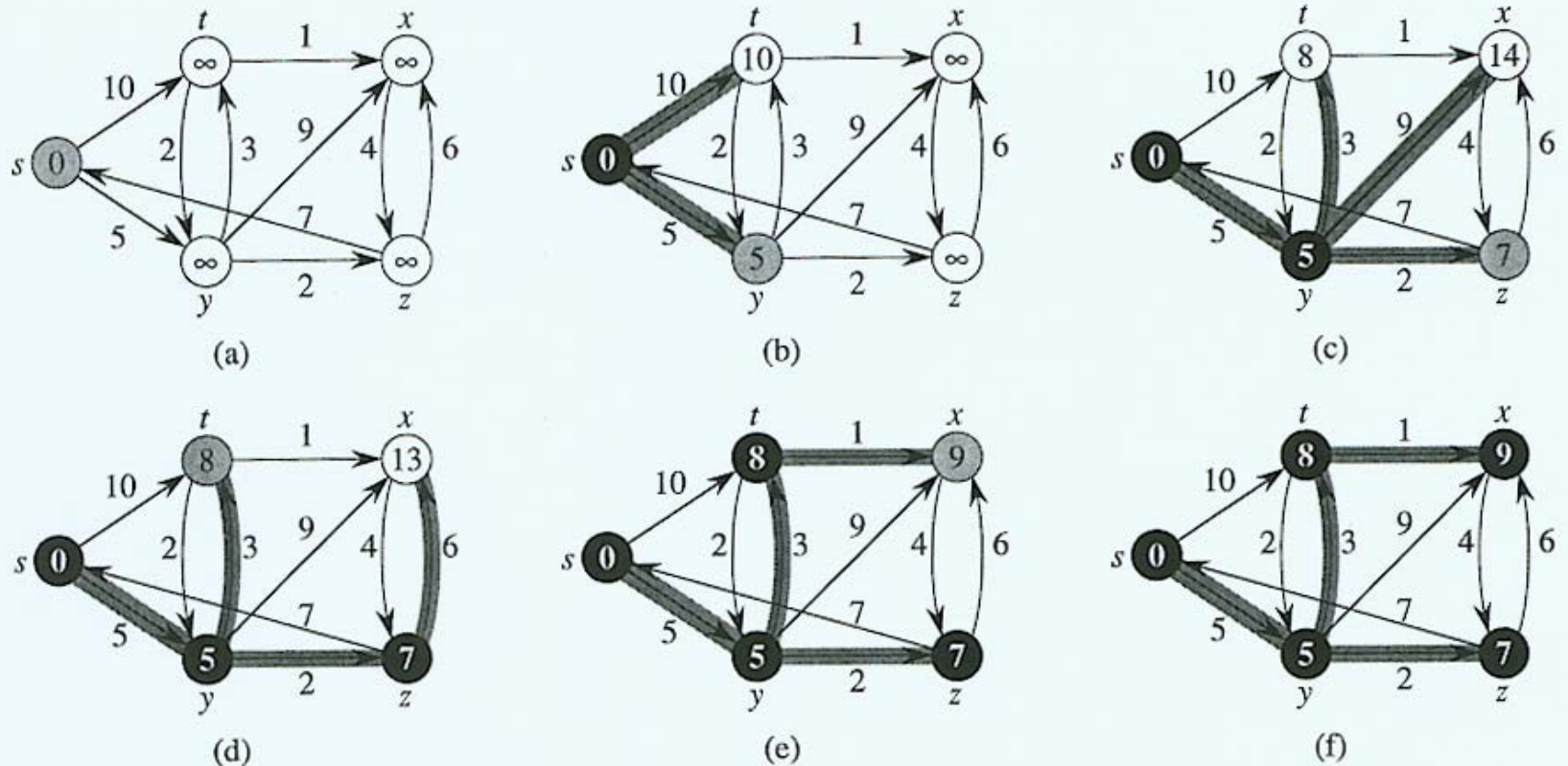
Figure 24.6 The execution of Dijkstra's algorithm. The source s is the leftmost vertex. The shortest-path estimates are shown within the vertices, and shaded edges indicate predecessor values. Black vertices are in the set S, and white vertices are in the min-priority queue $Q = V - S$. **(a)** The situation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

$\text{DIJKSTRA}(G, w, s)$

1. // $\text{INITIALIZE-SINGLE-SOURCE}(G, s)$
    for each vertex $v \in V[G]$
        do $d[v] \leftarrow \infty$
            $\pi[v] \leftarrow \text{NIL}$
    $d[s] \leftarrow 0$
2. $S \leftarrow \emptyset$
3. $Q \leftarrow V[G]$
4. **while** $Q \neq \emptyset$
5.   **do** $u \leftarrow \text{EXTRACT-MIN}(Q)$
6. $S \leftarrow S \cup \{u\}$
7.   **for** each $v \in Adj[u]$
8.   **do** // $\text{RELAX}(u, v, w)$
                    if $d[v] > d[u] + w(u, v)$
                        then $d[v] \leftarrow d[u] + w(u, v)$
                            $\pi[v] \leftarrow u$

DIJKSTRA(G, w, s)
1.   // INITIALIZE-SINGLE-SOURCE(G, s)
         for each vertex $v \in V[G]$
              do $d[v] \leftarrow \infty$
                   $\pi[v] \leftarrow$ NIL
              $d[s] \leftarrow 0$
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 $v \in Adj[u]$
8.   **do** // RELAX(u, v, w)
                        if $d[v] > d[u] + w(u, v)$
                             then $d[v] \leftarrow d[u] + w(u, v)$
                                  $\pi[v] \leftarrow u$

MST-PRIM(G, w, r)
1. $Q \leftarrow V[G]$
2.   **for** each $u \in Q$
3.   **do** $key[u] \leftarrow \infty$
4. $key[r] \leftarrow 0$
5. $\pi[r] \leftarrow$ NIL
6.   **while** $Q \neq \emptyset$
7.   **do** $u \leftarrow$ EXTRACT-MIN(Q)
8.   **for** each $v \in Adj[u]$
9.   **do if** $v \in Q$ and $w(u, v) < key[v]$
10.   **then** $\pi[v] \leftarrow u$
11. $key[v] \leftarrow w(u, v)$

All Pairs Shortest Path Algorithm

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

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

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

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

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

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

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

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.
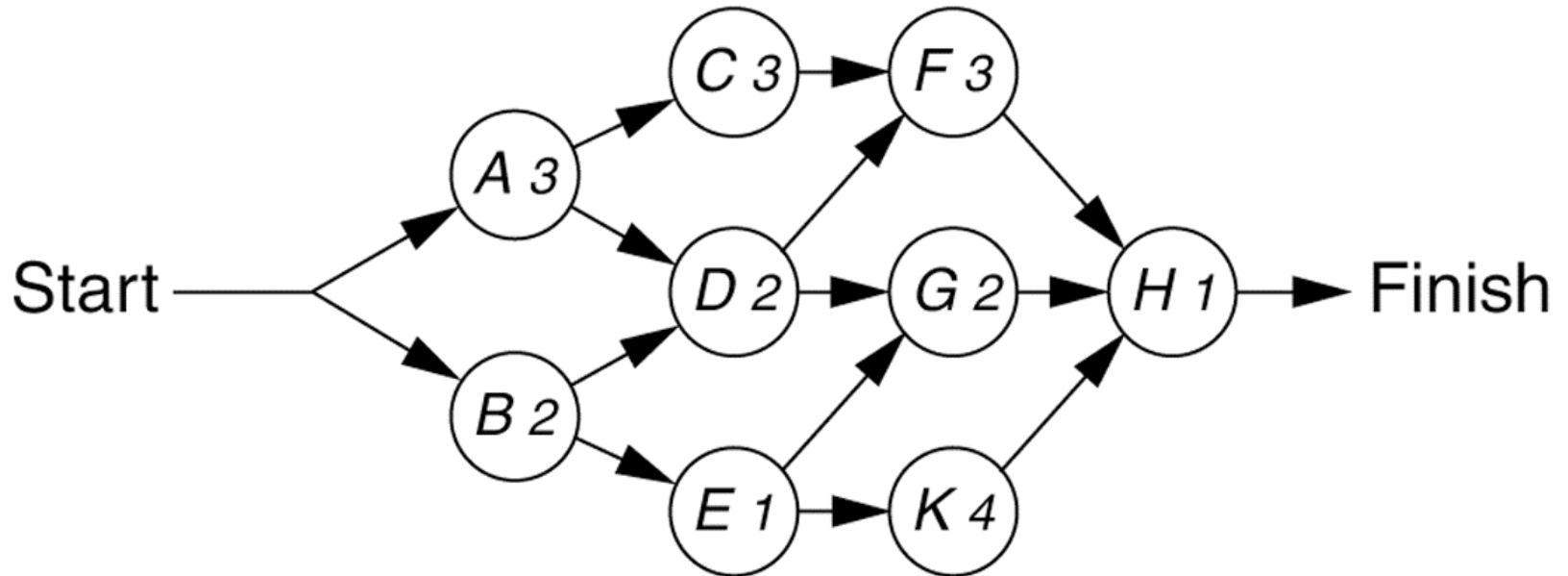
Figure 14.33

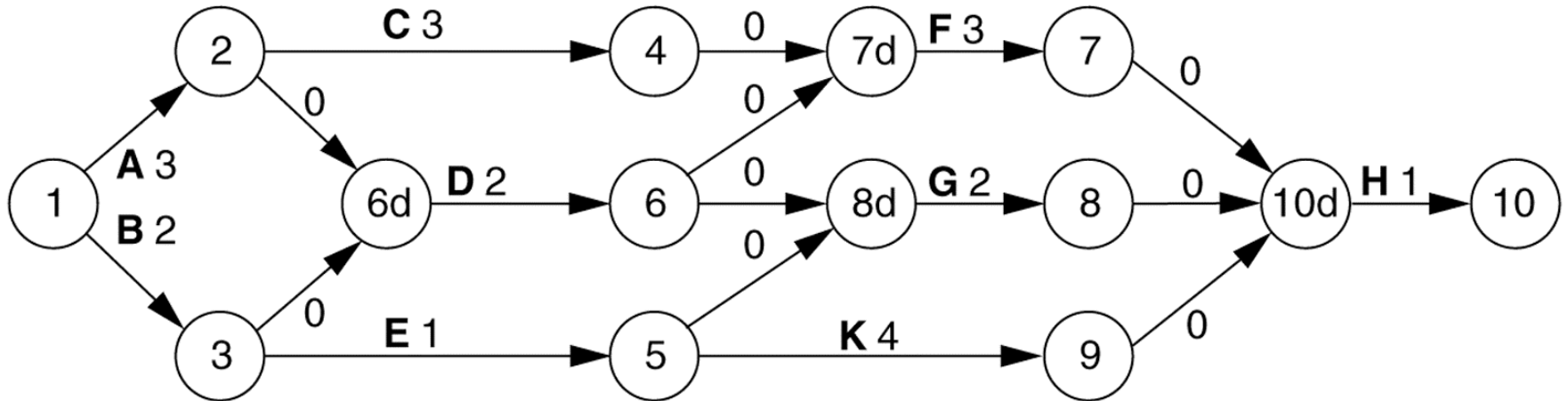An activity-node graph

Figure 14.34
An event-node graph

Figure 14.35
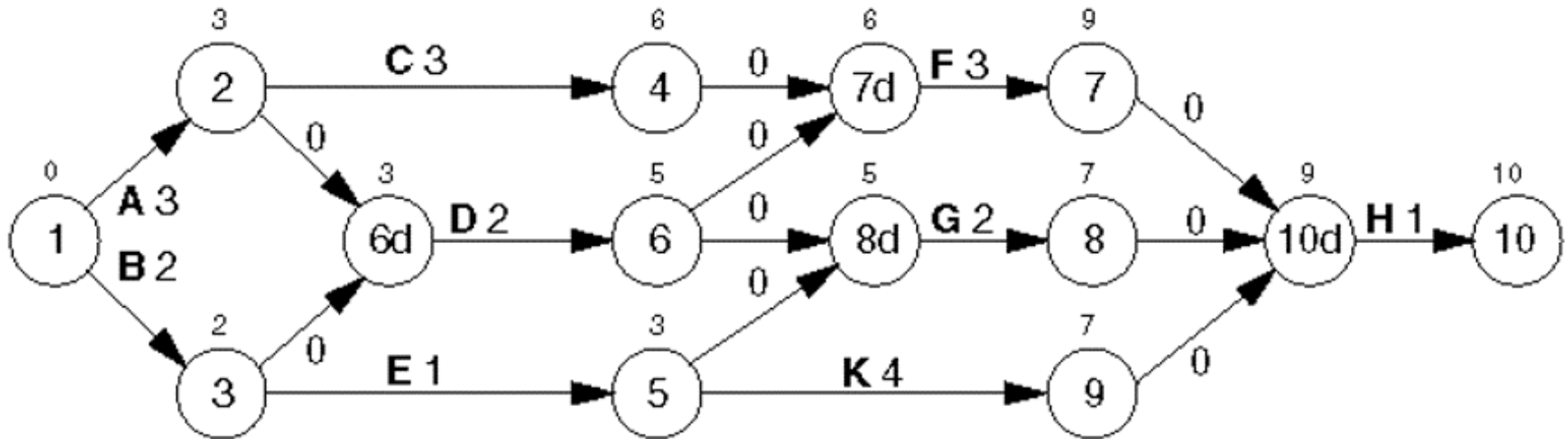Earliest completion times
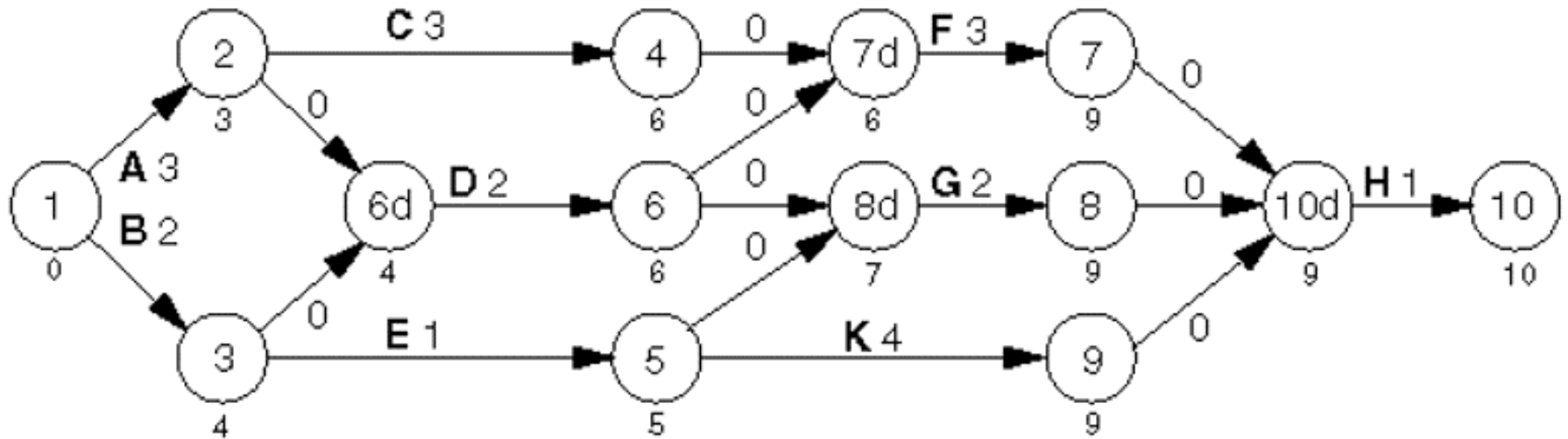
Figure 14.36
Latest completion times

Figure 14.37
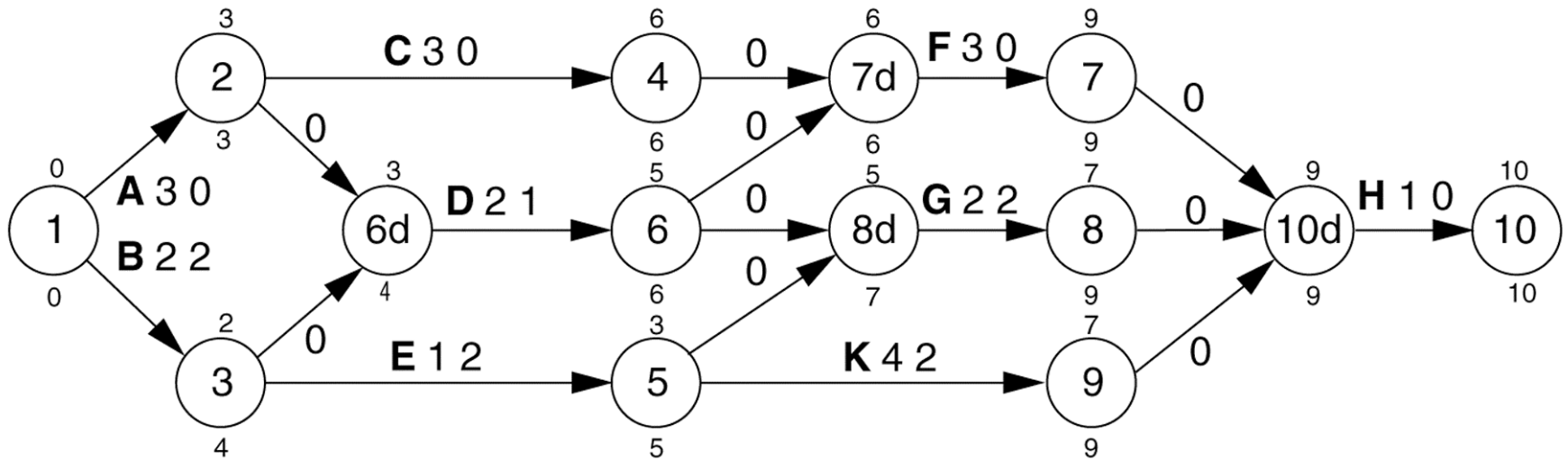Earliest completion time, latest completion time, and slack (additional edge item)

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