independent. Give an efficient algorithm to find the most reliable path between two given vertices.

25.2-5

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \{0, 1, \ldots, W - 1\}$ for some nonnegative integer W. Modify Dijkstra's algorithm to compute the shortest paths from a given source vertex s in $O(WV + E)$ time.

25.2-6

Modify your algorithm from Exercise 25.2-5 to run in $O((V + E) \lg W)$ time. (*Hint:* How many distinct shortest-path estimates can there be in $V - S$ at any point in time?)

25.3 The Bellman-Ford algorithm

The ***Bellman-Ford algorithm*** solves the single-source shortest-paths problem in the more general case in which edge weights can be negative. Given a weighted, directed graph $G = (V, E)$ with source s and weight function $w : E \to \mathbf{R}$, the Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source. If there is such a cycle, the algorithm indicates that no solution exists. If there is no such cycle, the algorithm produces the shortest paths and their weights.

Like Dijkstra's algorithm, the Bellman-Ford algorithm uses the technique of relaxation, progressively decreasing an estimate $d[v]$ on the weight of a shortest path from the source s to each vertex $v \in V$ until it achieves the actual shortest-path weight $\delta(s, v)$. The algorithm returns TRUE if and only if the graph contains no negative-weight cycles that are reachable from the source.

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i ← 1 to |V[G]| − 1
3       do for each edge (u, v) ∈ E[G]
4            do RELAX(u, v, w)
5  for each edge (u, v) ∈ E[G]
6       do if d[v] > d[u] + w(u, v)
7            then return FALSE
8  return TRUE
```

Figure 25.7 shows how the execution of the Bellman-Ford algorithm works on a graph with 5 vertices. After performing the usual initialization, the algorithm makes $|V| - 1$ passes over the edges of the graph. Each pass is one iteration of the **for** loop of lines 2–4 and consists of relaxing each

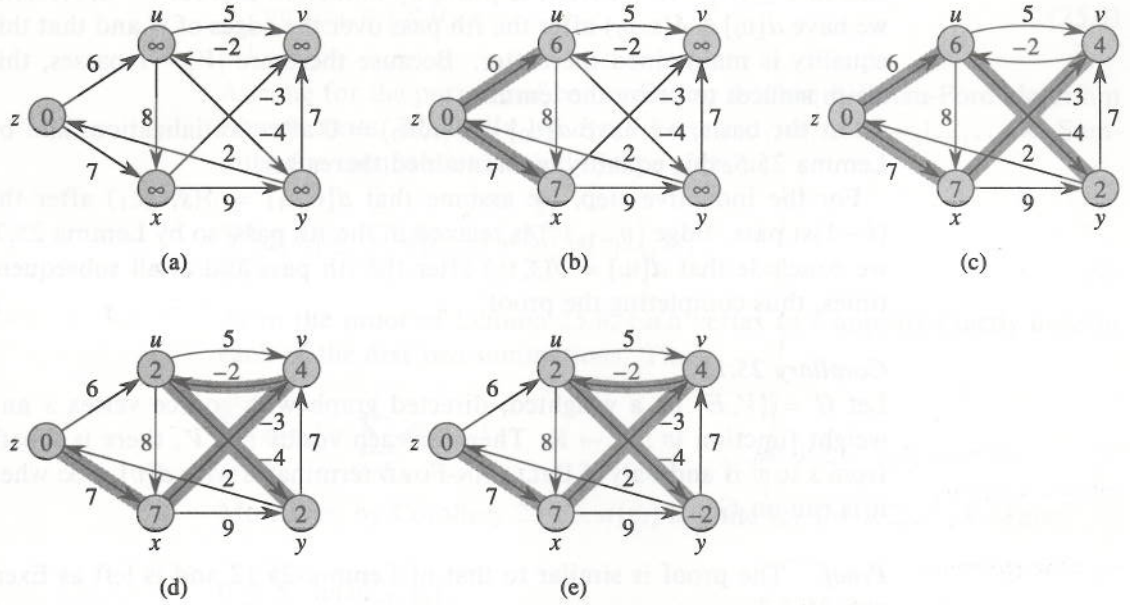

Figure 25.7 The execution of the Bellman-Ford algorithm. The source is vertex z. The d values are shown within the vertices, and shaded edges indicate the π values. In this particular example, each pass relaxes the edges in lexicographic order: $(u,v),(u,x),(u,y),(v,u),(x,v),(x,y),(y,v),(y,z),(z,u),(z,x)$. **(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.

edge of the graph once. Figures 25.7(b)–(e) show the state of the algorithm after each of the four passes over the edges. After making $|V| - 1$ passes, lines 5–8 check for a negative-weight cycle and return the appropriate boolean value. (We shall see a little later why this check works.)

The Bellman-Ford algorithm runs in time $O(VE)$, since the initialization in line 1 takes $\Theta(V)$ time, each of the $|V| - 1$ passes over the edges in lines 2–4 takes $O(E)$ time, and the **for** loop of lines 5–7 takes $O(E)$ time.

To prove the correctness of the Bellman-Ford algorithm, we start by showing that if there are no negative-weight cycles, the algorithm computes correct shortest-path weights for all vertices reachable from the source. The proof of this lemma contains the intuition behind the algorithm.

Lemma 25.12

Let $G = (V,E)$ be a weighted, directed graph with source s and weight function $w : E \to \mathbf{R}$, and assume that G contains no negative-weight cycles that are reachable from s. Then, at the termination of BELLMAN-FORD, we have $d[v] = \delta(s,v)$ for all vertices v that are reachable from s.

Proof Let v be a vertex reachable from s, and let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from s to v, where $v_0 = s$ and $v_k = v$. The path p is simple,

Lemma 25.3

Let $G = (V, E)$ be a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbf{R}$ and source vertex s. Then, for all edges $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Proof A shortest path p from source s to vertex v has no more weight than any other path from s to v. Specifically, path p has no more weight than the particular path that takes a shortest path from source s to vertex u and then takes edge (u, v). ■

Relaxation

The algorithms in this chapter use the technique of ***relaxation***. For each vertex $v \in V$, we maintain an attribute $d[v]$, which is an upper bound on the weight of a shortest path from source s to v. We call $d[v]$ a ***shortest-path estimate***. We initialize the shortest-path estimates and predecessors by the following procedure.

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ V[G]
2       do d[v] ← ∞
3          π[v] ← NIL
4  d[s] ← 0
```

After initialization, $\pi[v] = \text{NIL}$ for all $v \in V$, $d[v] = 0$ for $v = s$, and $d[v] = \infty$ for $v \in V - \{s\}$.

The process of ***relaxing***[1] an edge (u, v) consists of testing whether we can improve the shortest path to v found so far by going through u and, if so, updating $d[v]$ and $\pi[v]$. A relaxation step may decrease the value of the shortest-path estimate $d[v]$ and update v's predecessor field $\pi[v]$. The following code performs a relaxation step on edge (u, v).

```
RELAX(u, v, w)
1  if d[v] > d[u] + w(u, v)
2     then d[v] ← d[u] + w(u, v)
3          π[v] ← u
```

Figure 25.3 shows two examples of relaxing an edge, one in which a shortest-path estimate decreases and one in which no estimate changes.

---

[1] It may seem strange that the term "relaxation" is used for an operation that tightens an upper bound. The use of the term is historical. The outcome of a relaxation step can be viewed as a relaxation of the constraint $d[v] \leq d[u] + w(u, v)$, which, by Lemma 25.3, must be satisfied if $d[u] = \delta(s, u)$ and $d[v] = \delta(s, v)$. That is, if $d[v] \leq d[u] + w(u, v)$, there is no "pressure" to satisfy this constraint, so the constraint is "relaxed."

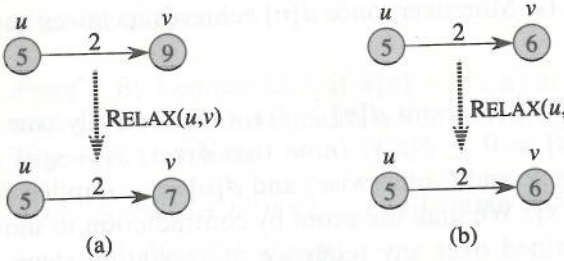

Figure 25.3 Relaxation of an edge (u, v). The shortest-path estimate of each vertex is shown within the vertex. **(a)** Because $d[v] > d[u] + w(u, v)$ prior to relaxation, the value of $d[v]$ decreases. **(b)** Here, $d[v] \leq d[u] + w(u, v)$ before the relaxation step, so $d[v]$ is unchanged by relaxation.

Each algorithm in this chapter calls INITIALIZE-SINGLE-SOURCE and then repeatedly relaxes edges. Moreover, relaxation is the only means by which shortest-path estimates and predecessors change. The algorithms in this chapter differ in how many times they relax each edge and the order in which they relax edges. In Dijkstra's algorithm and the shortest-paths algorithm for directed acyclic graphs, each edge is relaxed exactly once. In the Bellman-Ford algorithm, each edge is relaxed several times.

Properties of relaxation

The correctness of the algorithms in this chapter depends on important properties of relaxation that are summarized in the next few lemmas. Most of the lemmas describe the outcome of executing a sequence of relaxation steps on the edges of a weighted, directed graph that has been initialized by INITIALIZE-SINGLE-SOURCE. Except for Lemma 25.9, these lemmas apply to *any* sequence of relaxation steps, not just those that produce shortest-path values.

Lemma 25.4
Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbf{R}$, and let $(u, v) \in E$. Then, immediately after relaxing edge (u, v) by executing RELAX(u, v, w), we have $d[v] \leq d[u] + w(u, v)$.

Proof If, just prior to relaxing edge (u, v), we have $d[v] > d[u] + w(u, v)$, then $d[v] = d[u] + w(u, v)$ afterward. If, instead, $d[v] \leq d[u] + w(u, v)$ just before the relaxation, then neither $d[u]$ nor $d[v]$ changes, and so $d[v] \leq d[u] + w(u, v)$ afterward. ■

Lemma 25.5
Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbf{R}$. Let $s \in V$ be the source vertex, and let the graph be initialized by INITIALIZE-SINGLE-SOURCE(G, s). Then, $d[v] \geq \delta(s, v)$ for all $v \in V$, and