24.2-4
Give an efficient algorithm to count the total number of paths in a directed acyclic graph. Analyze your algorithm.

24.3 Dijkstra's algorithm

Dijkstra's algorithm solves the single-source shortest-paths problem on a weighted, directed graph $G = (V, E)$ for the case in which all edge weights are nonnegative. In this section, therefore, we assume that $w(u, v) \geq 0$ for each edge $(u, v) \in E$. As we shall see, with a good implementation, the running time of Dijkstra's algorithm is lower than that of the Bellman-Ford algorithm.

Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, adds u to S, and relaxes all edges leaving u. In the following implementation, we use a min-priority queue Q of vertices, keyed by their d values.

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

Dijkstra's algorithm relaxes edges as shown in Figure 24.6. Line 1 initializes the d and π values in the usual way, and line 2 initializes the set S to the empty set. The algorithm maintains the invariant that $Q = V - S$ at the start of each iteration of the **while** loop of lines 4–8. Line 3 initializes the min-priority queue Q to contain all the vertices in V; since $S = \emptyset$ at that time, the invariant is true after line 3. Each time through the **while** loop of lines 4–8, line 5 extracts a vertex u from $Q = V - S$ and line 6 adds it to set S, thereby maintaining the invariant. (The first time through this loop, $u = s$.) Vertex u, therefore, has the smallest shortest-path estimate of any vertex in $V - S$. Then, lines 7–8 relax each edge (u, v) leaving u, thus updating the estimate $v.d$ and the predecessor $v.\pi$ if we can improve the shortest path to v found so far by going through u. Observe that the algorithm never inserts vertices into Q after line 3 and that each vertex is extracted from Q

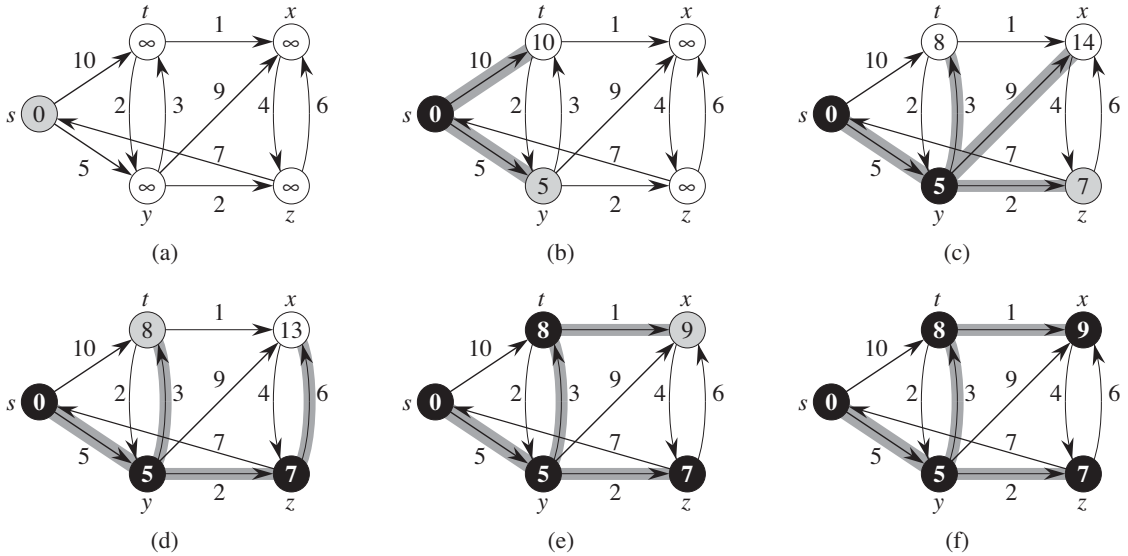

Figure 24.6 The execution of Dijkstra's algorithm. The source s is the leftmost vertex. The shortest-path estimates appear within the vertices, and shaded edges indicate predecessor values. Black vertices are in the set S, and white vertices are in the min-priority queue $Q = V - S$. **(a)** The situation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

and added to S exactly once, so that the **while** loop of lines 4–8 iterates exactly $|V|$ times.

Because Dijkstra's algorithm always chooses the "lightest" or "closest" vertex in $V - S$ to add to set S, we say that it uses a greedy strategy. Chapter 16 explains greedy strategies in detail, but you need not have read that chapter to understand Dijkstra's algorithm. Greedy strategies do not always yield optimal results in general, but as the following theorem and its corollary show, Dijkstra's algorithm does indeed compute shortest paths. The key is to show that each time it adds a vertex u to set S, we have $u.d = \delta(s, u)$.

Theorem 24.6 (Correctness of Dijkstra's algorithm)
Dijkstra's algorithm, run on a weighted, directed graph $G = (V, E)$ with nonnegative weight function w and source s, terminates with $u.d = \delta(s, u)$ for all vertices $u \in V$.

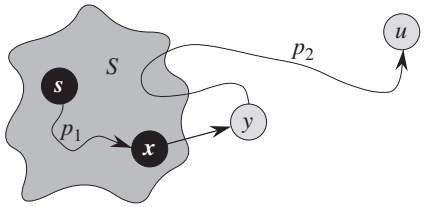

Figure 24.7 The proof of Theorem 24.6. Set S is nonempty just before vertex u is added to it. We decompose a shortest path p from source s to vertex u into $s \overset{p_1}{\rightsquigarrow} x \rightarrow y \overset{p_2}{\rightsquigarrow} u$, where y is the first vertex on the path that is not in S and $x \in S$ immediately precedes y. Vertices x and y are distinct, but we may have $s = x$ or $y = u$. Path p_2 may or may not reenter set S.

Proof We use the following loop invariant:

> At the start of each iteration of the **while** loop of lines 4–8, $v.d = \delta(s, v)$ for each vertex $v \in S$.

It suffices to show for each vertex $u \in V$, we have $u.d = \delta(s, u)$ at the time when u is added to set S. Once we show that $u.d = \delta(s, u)$, we rely on the upper-bound property to show that the equality holds at all times thereafter.

Initialization: Initially, $S = \emptyset$, and so the invariant is trivially true.

Maintenance: We wish to show that in each iteration, $u.d = \delta(s, u)$ for the vertex added to set S. For the purpose of contradiction, let u be the first vertex for which $u.d \neq \delta(s, u)$ when it is added to set S. We shall focus our attention on the situation at the beginning of the iteration of the **while** loop in which u is added to S and derive the contradiction that $u.d = \delta(s, u)$ at that time by examining a shortest path from s to u. We must have $u \neq s$ because s is the first vertex added to set S and $s.d = \delta(s, s) = 0$ at that time. Because $u \neq s$, we also have that $S \neq \emptyset$ just before u is added to S. There must be some path from s to u, for otherwise $u.d = \delta(s, u) = \infty$ by the no-path property, which would violate our assumption that $u.d \neq \delta(s, u)$. Because there is at least one path, there is a shortest path p from s to u. Prior to adding u to S, path p connects a vertex in S, namely s, to a vertex in $V - S$, namely u. Let us consider the first vertex y along p such that $y \in V - S$, and let $x \in S$ be y's predecessor along p. Thus, as Figure 24.7 illustrates, we can decompose path p into $s \overset{p_1}{\rightsquigarrow} x \rightarrow y \overset{p_2}{\rightsquigarrow} u$. (Either of paths p_1 or p_2 may have no edges.)

We claim that $y.d = \delta(s, y)$ when u is added to S. To prove this claim, observe that $x \in S$. Then, because we chose u as the first vertex for which $u.d \neq \delta(s, u)$ when it is added to S, we had $x.d = \delta(s, x)$ when x was added

to S. Edge (x, y) was relaxed at that time, and the claim follows from the convergence property.

We can now obtain a contradiction to prove that $u.d = \delta(s, u)$. Because y appears before u on a shortest path from s to u and all edge weights are nonnegative (notably those on path p_2), we have $\delta(s, y) \leq \delta(s, u)$, and thus

$$\begin{aligned} y.d &= \delta(s, y) \\ &\leq \delta(s, u) \\ &\leq u.d \qquad \text{(by the upper-bound property)} . \end{aligned} \tag{24.2}$$

But because both vertices u and y were in $V - S$ when u was chosen in line 5, we have $u.d \leq y.d$. Thus, the two inequalities in (24.2) are in fact equalities, giving

$$y.d = \delta(s, y) = \delta(s, u) = u.d .$$

Consequently, $u.d = \delta(s, u)$, which contradicts our choice of u. We conclude that $u.d = \delta(s, u)$ when u is added to S, and that this equality is maintained at all times thereafter.

Termination: At termination, $Q = \emptyset$ which, along with our earlier invariant that $Q = V - S$, implies that $S = V$. Thus, $u.d = \delta(s, u)$ for all vertices $u \in V$. ■

Corollary 24.7

If we run Dijkstra's algorithm on a weighted, directed graph $G = (V, E)$ with nonnegative weight function w and source s, then at termination, the predecessor subgraph G_π is a shortest-paths tree rooted at s.

Proof Immediate from Theorem 24.6 and the predecessor-subgraph property. ■

Analysis

How fast is Dijkstra's algorithm? It maintains the min-priority queue Q by calling three priority-queue operations: INSERT (implicit in line 3), EXTRACT-MIN (line 5), and DECREASE-KEY (implicit in RELAX, which is called in line 8). The algorithm calls both INSERT and EXTRACT-MIN once per vertex. Because each vertex $u \in V$ is added to set S exactly once, each edge in the adjacency list $Adj[u]$ is examined in the **for** loop of lines 7–8 exactly once during the course of the algorithm. Since the total number of edges in all the adjacency lists is $|E|$, this **for** loop iterates a total of $|E|$ times, and thus the algorithm calls DECREASE-KEY at most $|E|$ times overall. (Observe once again that we are using aggregate analysis.)

The running time of Dijkstra's algorithm depends on how we implement the min-priority queue. Consider first the case in which we maintain the min-priority

queue by taking advantage of the vertices being numbered 1 to $|V|$. We simply store $v.d$ in the vth entry of an array. Each INSERT and DECREASE-KEY operation takes $O(1)$ time, and each EXTRACT-MIN operation takes $O(V)$ time (since we have to search through the entire array), for a total time of $O(V^2 + E) = O(V^2)$.

If the graph is sufficiently sparse—in particular, $E = o(V^2/\lg V)$—we can improve the algorithm by implementing the min-priority queue with a binary min-heap. (As discussed in Section 6.5, the implementation should make sure that vertices and corresponding heap elements maintain handles to each other.) Each EXTRACT-MIN operation then takes time $O(\lg V)$. As before, there are $|V|$ such operations. The time to build the binary min-heap is $O(V)$. Each DECREASE-KEY operation takes time $O(\lg V)$, and there are still at most $|E|$ such operations. The total running time is therefore $O((V + E) \lg V)$, which is $O(E \lg V)$ if all vertices are reachable from the source. This running time improves upon the straightforward $O(V^2)$-time implementation if $E = o(V^2/\lg V)$.

We can in fact achieve a running time of $O(V \lg V + E)$ by implementing the min-priority queue with a Fibonacci heap (see Chapter 19). The amortized cost of each of the $|V|$ EXTRACT-MIN operations is $O(\lg V)$, and each DECREASE-KEY call, of which there are at most $|E|$, takes only $O(1)$ amortized time. Historically, the development of Fibonacci heaps was motivated by the observation that Dijkstra's algorithm typically makes many more DECREASE-KEY calls than EXTRACT-MIN calls, so that any method of reducing the amortized time of each DECREASE-KEY operation to $o(\lg V)$ without increasing the amortized time of EXTRACT-MIN would yield an asymptotically faster implementation than with binary heaps.

Dijkstra's algorithm resembles both breadth-first search (see Section 22.2) and Prim's algorithm for computing minimum spanning trees (see Section 23.2). It is like breadth-first search in that set S corresponds to the set of black vertices in a breadth-first search; just as vertices in S have their final shortest-path weights, so do black vertices in a breadth-first search have their correct breadth-first distances. Dijkstra's algorithm is like Prim's algorithm in that both algorithms use a min-priority queue to find the "lightest" vertex outside a given set (the set S in Dijkstra's algorithm and the tree being grown in Prim's algorithm), add this vertex into the set, and adjust the weights of the remaining vertices outside the set accordingly.

Exercises

24.3-1

Run Dijkstra's algorithm on the directed graph of Figure 24.2, first using vertex s as the source and then using vertex z as the source. In the style of Figure 24.6, show the d and π values and the vertices in set S after each iteration of the **while** loop.

24.3-2
Give a simple example of a directed graph with negative-weight edges for which Dijkstra's algorithm produces incorrect answers. Why doesn't the proof of Theorem 24.6 go through when negative-weight edges are allowed?

24.3-3
Suppose we change line 4 of Dijkstra's algorithm to the following.

4 **while** $|Q| > 1$

This change causes the **while** loop to execute $|V| - 1$ times instead of $|V|$ times. Is this proposed algorithm correct?

24.3-4
Professor Gaedel has written a program that he claims implements Dijkstra's algorithm. The program produces $v.d$ and $v.\pi$ for each vertex $v \in V$. Give an $O(V + E)$-time algorithm to check the output of the professor's program. It should determine whether the d and π attributes match those of some shortest-paths tree. You may assume that all edge weights are nonnegative.

24.3-5
Professor Newman thinks that he has worked out a simpler proof of correctness for Dijkstra's algorithm. He claims that Dijkstra's algorithm relaxes the edges of every shortest path in the graph in the order in which they appear on the path, and therefore the path-relaxation property applies to every vertex reachable from the source. Show that the professor is mistaken by constructing a directed graph for which Dijkstra's algorithm could relax the edges of a shortest path out of order.

24.3-6
We are given a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has an associated value $r(u, v)$, which is a real number in the range $0 \le r(u, v) \le 1$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u, v)$ as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices.

24.3-7
Let $G = (V, E)$ be a weighted, directed graph with positive weight function $w : E \to \{1, 2, \ldots, W\}$ for some positive integer W, and assume that no two vertices have the same shortest-path weights from source vertex s. Now suppose that we define an unweighted, directed graph $G' = (V \cup V', E')$ by replacing each edge $(u, v) \in E$ with $w(u, v)$ unit-weight edges in series. How many vertices does G' have? Now suppose that we run a breadth-first search on G'. Show that

the order in which the breadth-first search of G' colors vertices in V black is the same as the order in which Dijkstra's algorithm extracts the vertices of V from the priority queue when it runs on G.

24.3-8
Let $G = (V, E)$ be a weighted, directed graph with nonnegative weight function $w : E \to \{0, 1, \ldots, W\}$ 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.

24.3-9
Modify your algorithm from Exercise 24.3-8 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?)

24.3-10
Suppose that we are given a weighted, directed graph $G = (V, E)$ in which edges that leave the source vertex s may have negative weights, all other edge weights are nonnegative, and there are no negative-weight cycles. Argue that Dijkstra's algorithm correctly finds shortest paths from s in this graph.

24.4 Difference constraints and shortest paths

Chapter 29 studies the general linear-programming problem, in which we wish to optimize a linear function subject to a set of linear inequalities. In this section, we investigate a special case of linear programming that we reduce to finding shortest paths from a single source. We can then solve the single-source shortest-paths problem that results by running the Bellman-Ford algorithm, thereby also solving the linear-programming problem.

Linear programming

In the general ***linear-programming problem***, we are given an $m \times n$ matrix A, an m-vector b, and an n-vector c. We wish to find a vector x of n elements that maximizes the ***objective function*** $\sum_{i=1}^{n} c_i x_i$ subject to the m constraints given by $Ax \leq b$.

Although the simplex algorithm, which is the focus of Chapter 29, does not always run in time polynomial in the size of its input, there are other linear-programming algorithms that do run in polynomial time. We offer here two reasons to understand the setup of linear-programming problems. First, if we know that we

can cast a given problem as a polynomial-sized linear-programming problem, then we immediately have a polynomial-time algorithm to solve the problem. Second, faster algorithms exist for many special cases of linear programming. For example, the single-pair shortest-path problem (Exercise 24.4-4) and the maximum-flow problem (Exercise 26.1-5) are special cases of linear programming.

Sometimes we don't really care about the objective function; we just wish to find any ***feasible solution***, that is, any vector x that satisfies $Ax \leq b$, or to determine that no feasible solution exists. We shall focus on one such ***feasibility problem***.

Systems of difference constraints

In a ***system of difference constraints***, each row of the linear-programming matrix A contains one 1 and one -1, and all other entries of A are 0. Thus, the constraints given by $Ax \leq b$ are a set of m ***difference constraints*** involving n unknowns, in which each constraint is a simple linear inequality of the form

$$x_j - x_i \leq b_k \ ,$$

where $1 \leq i, j \leq n, i \neq j$, and $1 \leq k \leq m$.

For example, consider the problem of finding a 5-vector $x = (x_i)$ that satisfies

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix} .$$

This problem is equivalent to finding values for the unknowns x_1, x_2, x_3, x_4, x_5, satisfying the following 8 difference constraints:

$$\begin{aligned} x_1 - x_2 &\leq 0 \ , && (24.3) \\ x_1 - x_5 &\leq -1 \ , && (24.4) \\ x_2 - x_5 &\leq 1 \ , && (24.5) \\ x_3 - x_1 &\leq 5 \ , && (24.6) \\ x_4 - x_1 &\leq 4 \ , && (24.7) \\ x_4 - x_3 &\leq -1 \ , && (24.8) \\ x_5 - x_3 &\leq -3 \ , && (24.9) \\ x_5 - x_4 &\leq -3 \ . && (24.10) \end{aligned}$$

One solution to this problem is $x = (-5, -3, 0, -1, -4)$, which you can verify directly by checking each inequality. In fact, this problem has more than one solution. Another is $x' = (0, 2, 5, 4, 1)$. These two solutions are related: each component of x' is 5 larger than the corresponding component of x. This fact is not mere coincidence.

Lemma 24.8
Let $x = (x_1, x_2, \ldots, x_n)$ be a solution to a system $Ax \leq b$ of difference constraints, and let d be any constant. Then $x + d = (x_1 + d, x_2 + d, \ldots, x_n + d)$ is a solution to $Ax \leq b$ as well.

Proof For each x_i and x_j, we have $(x_j + d) - (x_i + d) = x_j - x_i$. Thus, if x satisfies $Ax \leq b$, so does $x + d$. ■

Systems of difference constraints occur in many different applications. For example, the unknowns x_i may be times at which events are to occur. Each constraint states that at least a certain amount of time, or at most a certain amount of time, must elapse between two events. Perhaps the events are jobs to be performed during the assembly of a product. If we apply an adhesive that takes 2 hours to set at time x_1 and we have to wait until it sets to install a part at time x_2, then we have the constraint that $x_2 \geq x_1 + 2$ or, equivalently, that $x_1 - x_2 \leq -2$. Alternatively, we might require that the part be installed after the adhesive has been applied but no later than the time that the adhesive has set halfway. In this case, we get the pair of constraints $x_2 \geq x_1$ and $x_2 \leq x_1 + 1$ or, equivalently, $x_1 - x_2 \leq 0$ and $x_2 - x_1 \leq 1$.

Constraint graphs

We can interpret systems of difference constraints from a graph-theoretic point of view. In a system $Ax \leq b$ of difference constraints, we view the $m \times n$ linear-programming matrix A as the transpose of an incidence matrix (see Exercise 22.1-7) for a graph with n vertices and m edges. Each vertex v_i in the graph, for $i = 1, 2, \ldots, n$, corresponds to one of the n unknown variables x_i. Each directed edge in the graph corresponds to one of the m inequalities involving two unknowns.

More formally, given a system $Ax \leq b$ of difference constraints, the corresponding ***constraint graph*** is a weighted, directed graph $G = (V, E)$, where

$$V = \{v_0, v_1, \ldots, v_n\}$$

and

$$E = \{(v_i, v_j) : x_j - x_i \leq b_k \text{ is a constraint}\} \cup \{(v_0, v_1), (v_0, v_2), (v_0, v_3), \ldots, (v_0, v_n)\} .$$

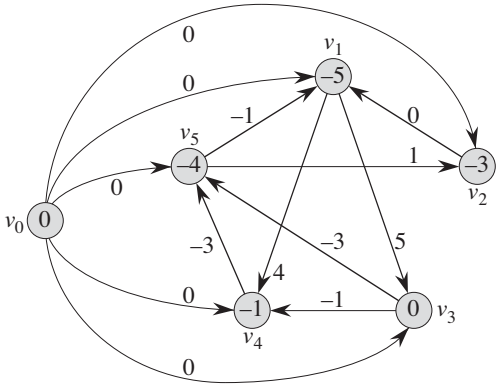

Figure 24.8 The constraint graph corresponding to the system (24.3)–(24.10) of difference constraints. The value of $\delta(v_0, v_i)$ appears in each vertex v_i. One feasible solution to the system is $x = (-5, -3, 0, -1, -4)$.

The constraint graph contains the additional vertex v_0, as we shall see shortly, to guarantee that the graph has some vertex which can reach all other vertices. Thus, the vertex set V consists of a vertex v_i for each unknown x_i, plus an additional vertex v_0. The edge set E contains an edge for each difference constraint, plus an edge (v_0, v_i) for each unknown x_i. If $x_j - x_i \leq b_k$ is a difference constraint, then the weight of edge (v_i, v_j) is $w(v_i, v_j) = b_k$. The weight of each edge leaving v_0 is 0. Figure 24.8 shows the constraint graph for the system (24.3)–(24.10) of difference constraints.

The following theorem shows that we can find a solution to a system of difference constraints by finding shortest-path weights in the corresponding constraint graph.

Theorem 24.9

Given a system $Ax \leq b$ of difference constraints, let $G = (V, E)$ be the corresponding constraint graph. If G contains no negative-weight cycles, then

$$x = (\delta(v_0, v_1), \delta(v_0, v_2), \delta(v_0, v_3), \ldots, \delta(v_0, v_n)) \tag{24.11}$$

is a feasible solution for the system. If G contains a negative-weight cycle, then there is no feasible solution for the system.

Proof We first show that if the constraint graph contains no negative-weight cycles, then equation (24.11) gives a feasible solution. Consider any edge $(v_i, v_j) \in E$. By the triangle inequality, $\delta(v_0, v_j) \leq \delta(v_0, v_i) + w(v_i, v_j)$ or, equivalently, $\delta(v_0, v_j) - \delta(v_0, v_i) \leq w(v_i, v_j)$. Thus, letting $x_i = \delta(v_0, v_i)$ and

$x_j = \delta(v_0, v_j)$ satisfies the difference constraint $x_j - x_i \le w(v_i, v_j)$ that corresponds to edge (v_i, v_j).

Now we show that if the constraint graph contains a negative-weight cycle, then the system of difference constraints has no feasible solution. Without loss of generality, let the negative-weight cycle be $c = \langle v_1, v_2, \ldots, v_k \rangle$, where $v_1 = v_k$. (The vertex v_0 cannot be on cycle c, because it has no entering edges.) Cycle c corresponds to the following difference constraints:

$$\begin{aligned}
x_2 - x_1 &\le w(v_1, v_2) , \\
x_3 - x_2 &\le w(v_2, v_3) , \\
&\vdots \\
x_{k-1} - x_{k-2} &\le w(v_{k-2}, v_{k-1}) , \\
x_k - x_{k-1} &\le w(v_{k-1}, v_k) .
\end{aligned}$$

We will assume that x has a solution satisfying each of these k inequalities and then derive a contradiction. The solution must also satisfy the inequality that results when we sum the k inequalities together. If we sum the left-hand sides, each unknown x_i is added in once and subtracted out once (remember that $v_1 = v_k$ implies $x_1 = x_k$), so that the left-hand side of the sum is 0. The right-hand side sums to $w(c)$, and thus we obtain $0 \le w(c)$. But since c is a negative-weight cycle, $w(c) < 0$, and we obtain the contradiction that $0 \le w(c) < 0$. ■

Solving systems of difference constraints

Theorem 24.9 tells us that we can use the Bellman-Ford algorithm to solve a system of difference constraints. Because the constraint graph contains edges from the source vertex v_0 to all other vertices, any negative-weight cycle in the constraint graph is reachable from v_0. If the Bellman-Ford algorithm returns TRUE, then the shortest-path weights give a feasible solution to the system. In Figure 24.8, for example, the shortest-path weights provide the feasible solution $x = (-5, -3, 0, -1, -4)$, and by Lemma 24.8, $x = (d-5, d-3, d, d-1, d-4)$ is also a feasible solution for any constant d. If the Bellman-Ford algorithm returns FALSE, there is no feasible solution to the system of difference constraints.

A system of difference constraints with m constraints on n unknowns produces a graph with $n + 1$ vertices and $n + m$ edges. Thus, using the Bellman-Ford algorithm, we can solve the system in $O((n+1)(n+m)) = O(n^2 + nm)$ time. Exercise 24.4-5 asks you to modify the algorithm to run in $O(nm)$ time, even if m is much less than n.

Exercises

24.4-1

Find a feasible solution or determine that no feasible solution exists for the following system of difference constraints:

$$
\begin{aligned}
x_1 - x_2 &\le 1, \\
x_1 - x_4 &\le -4, \\
x_2 - x_3 &\le 2, \\
x_2 - x_5 &\le 7, \\
x_2 - x_6 &\le 5, \\
x_3 - x_6 &\le 10, \\
x_4 - x_2 &\le 2, \\
x_5 - x_1 &\le -1, \\
x_5 - x_4 &\le 3, \\
x_6 - x_3 &\le -8.
\end{aligned}
$$

24.4-2

Find a feasible solution or determine that no feasible solution exists for the following system of difference constraints:

$$
\begin{aligned}
x_1 - x_2 &\le 4, \\
x_1 - x_5 &\le 5, \\
x_2 - x_4 &\le -6, \\
x_3 - x_2 &\le 1, \\
x_4 - x_1 &\le 3, \\
x_4 - x_3 &\le 5, \\
x_4 - x_5 &\le 10, \\
x_5 - x_3 &\le -4, \\
x_5 - x_4 &\le -8.
\end{aligned}
$$

24.4-3

Can any shortest-path weight from the new vertex v_0 in a constraint graph be positive? Explain.

24.4-4

Express the single-pair shortest-path problem as a linear program.

24.4-5

Show how to modify the Bellman-Ford algorithm slightly so that when we use it to solve a system of difference constraints with m inequalities on n unknowns, the running time is $O(nm)$.

24.4-6

Suppose that in addition to a system of difference constraints, we want to handle ***equality constraints*** of the form $x_i = x_j + b_k$. Show how to adapt the Bellman-Ford algorithm to solve this variety of constraint system.

24.4-7

Show how to solve a system of difference constraints by a Bellman-Ford-like algorithm that runs on a constraint graph without the extra vertex v_0.

24.4-8 ⋆

Let $Ax \leq b$ be a system of m difference constraints in n unknowns. Show that the Bellman-Ford algorithm, when run on the corresponding constraint graph, maximizes $\sum_{i=1}^{n} x_i$ subject to $Ax \leq b$ and $x_i \leq 0$ for all x_i.

24.4-9 ⋆

Show that the Bellman-Ford algorithm, when run on the constraint graph for a system $Ax \leq b$ of difference constraints, minimizes the quantity $(\max\{x_i\} - \min\{x_i\})$ subject to $Ax \leq b$. Explain how this fact might come in handy if the algorithm is used to schedule construction jobs.

24.4-10

Suppose that every row in the matrix A of a linear program $Ax \leq b$ corresponds to a difference constraint, a single-variable constraint of the form $x_i \leq b_k$, or a single-variable constraint of the form $-x_i \leq b_k$. Show how to adapt the Bellman-Ford algorithm to solve this variety of constraint system.

24.4-11

Give an efficient algorithm to solve a system $Ax \leq b$ of difference constraints when all of the elements of b are real-valued and all of the unknowns x_i must be integers.

24.4-12 ⋆

Give an efficient algorithm to solve a system $Ax \leq b$ of difference constraints when all of the elements of b are real-valued and a specified subset of some, but not necessarily all, of the unknowns x_i must be integers.

24.5 Proofs of shortest-paths properties

Throughout this chapter, our correctness arguments have relied on the triangle inequality, upper-bound property, no-path property, convergence property, path-relaxation property, and predecessor-subgraph property. We stated these properties without proof at the beginning of this chapter. In this section, we prove them.

The triangle inequality

In studying breadth-first search (Section 22.2), we proved as Lemma 22.1 a simple property of shortest distances in unweighted graphs. The triangle inequality generalizes the property to weighted graphs.

Lemma 24.10 (Triangle inequality)
Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$ 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 Suppose that p is a shortest path from source s to vertex v. Then p 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).

Exercise 24.5-3 asks you to handle the case in which there is no shortest path from s to v. ■

Effects of relaxation on shortest-path estimates

The next group of lemmas describes how shortest-path estimates are affected when we execute a sequence of relaxation steps on the edges of a weighted, directed graph that has been initialized by INITIALIZE-SINGLE-SOURCE.

Lemma 24.11 (Upper-bound property)
Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$. Let $s \in V$ be the source vertex, and let the graph be initialized by INITIALIZE-SINGLE-SOURCE(G, s). Then, $v.d \geq \delta(s, v)$ for all $v \in V$, and this invariant is maintained over any sequence of relaxation steps on the edges of G. Moreover, once $v.d$ achieves its lower bound $\delta(s, v)$, it never changes.

Proof We prove the invariant $v.d \geq \delta(s, v)$ for all vertices $v \in V$ by induction over the number of relaxation steps.

For the basis, $v.d \geq \delta(s, v)$ is certainly true after initialization, since $v.d = \infty$ implies $v.d \geq \delta(s, v)$ for all $v \in V - \{s\}$, and since $s.d = 0 \geq \delta(s, s)$ (note that $\delta(s, s) = -\infty$ if s is on a negative-weight cycle and 0 otherwise).

For the inductive step, consider the relaxation of an edge (u, v). By the inductive hypothesis, $x.d \geq \delta(s, x)$ for all $x \in V$ prior to the relaxation. The only d value that may change is $v.d$. If it changes, we have

$$\begin{aligned} v.d &= u.d + w(u, v) \\ &\geq \delta(s, u) + w(u, v) && \text{(by the inductive hypothesis)} \\ &\geq \delta(s, v) && \text{(by the triangle inequality)} , \end{aligned}$$

and so the invariant is maintained.

To see that the value of $v.d$ never changes once $v.d = \delta(s, v)$, note that having achieved its lower bound, $v.d$ cannot decrease because we have just shown that $v.d \geq \delta(s, v)$, and it cannot increase because relaxation steps do not increase d values. ■

Corollary 24.12 (No-path property)
Suppose that in a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, no path connects a source vertex $s \in V$ to a given vertex $v \in V$. Then, after the graph is initialized by INITIALIZE-SINGLE-SOURCE(G, s), we have $v.d = \delta(s, v) = \infty$, and this equality is maintained as an invariant over any sequence of relaxation steps on the edges of G.

Proof By the upper-bound property, we always have $\infty = \delta(s, v) \leq v.d$, and thus $v.d = \infty = \delta(s, v)$. ■

Lemma 24.13
Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$, and let $(u, v) \in E$. Then, immediately after relaxing edge (u, v) by executing RELAX(u, v, w), we have $v.d \leq u.d + w(u, v)$.

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

Lemma 24.14 (Convergence property)
Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$, let $s \in V$ be a source vertex, and let $s \rightsquigarrow u \to v$ be a shortest path in G for

some vertices $u, v \in V$. Suppose that G is initialized by INITIALIZE-SINGLE-SOURCE(G, s) and then a sequence of relaxation steps that includes the call RELAX(u, v, w) is executed on the edges of G. If $u.d = \delta(s, u)$ at any time prior to the call, then $v.d = \delta(s, v)$ at all times after the call.

Proof By the upper-bound property, if $u.d = \delta(s, u)$ at some point prior to relaxing edge (u, v), then this equality holds thereafter. In particular, after relaxing edge (u, v), we have

$$\begin{aligned} v.d &\le u.d + w(u, v) && \text{(by Lemma 24.13)} \\ &= \delta(s, u) + w(u, v) \\ &= \delta(s, v) && \text{(by Lemma 24.1)} \ . \end{aligned}$$

By the upper-bound property, $v.d \ge \delta(s, v)$, from which we conclude that $v.d = \delta(s, v)$, and this equality is maintained thereafter. ■

Lemma 24.15 (Path-relaxation property)
Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$, and let $s \in V$ be a source vertex. Consider any shortest path $p = \langle v_0, v_1, \ldots, v_k \rangle$ from $s = v_0$ to v_k. If G is initialized by INITIALIZE-SINGLE-SOURCE(G, s) and then a sequence of relaxation steps occurs that includes, in order, relaxing the edges $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$ after these relaxations and at all times afterward. This property holds no matter what other edge relaxations occur, including relaxations that are intermixed with relaxations of the edges of p.

Proof We show by induction that after the ith edge of path p is relaxed, we have $v_i.d = \delta(s, v_i)$. For the basis, $i = 0$, and before any edges of p have been relaxed, we have from the initialization that $v_0.d = s.d = 0 = \delta(s, s)$. By the upper-bound property, the value of $s.d$ never changes after initialization.

For the inductive step, we assume that $v_{i-1}.d = \delta(s, v_{i-1})$, and we examine what happens when we relax edge (v_{i-1}, v_i). By the convergence property, after relaxing this edge, we have $v_i.d = \delta(s, v_i)$, and this equality is maintained at all times thereafter. ■

Relaxation and shortest-paths trees

We now show that once a sequence of relaxations has caused the shortest-path estimates to converge to shortest-path weights, the predecessor subgraph G_π induced by the resulting π values is a shortest-paths tree for G. We start with the following lemma, which shows that the predecessor subgraph always forms a rooted tree whose root is the source.

Lemma 24.16

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$, let $s \in V$ be a source vertex, and assume that G contains no negative-weight cycles that are reachable from s. Then, after the graph is initialized by INITIALIZE-SINGLE-SOURCE(G, s), the predecessor subgraph G_π forms a rooted tree with root s, and any sequence of relaxation steps on edges of G maintains this property as an invariant.

Proof Initially, the only vertex in G_π is the source vertex, and the lemma is trivially true. Consider a predecessor subgraph G_π that arises after a sequence of relaxation steps. We shall first prove that G_π is acyclic. Suppose for the sake of contradiction that some relaxation step creates a cycle in the graph G_π. Let the cycle be $c = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_k = v_0$. Then, $v_i.\pi = v_{i-1}$ for $i = 1, 2, \ldots, k$ and, without loss of generality, we can assume that relaxing edge (v_{k-1}, v_k) created the cycle in G_π.

We claim that all vertices on cycle c are reachable from the source s. Why? Each vertex on c has a non-NIL predecessor, and so each vertex on c was assigned a finite shortest-path estimate when it was assigned its non-NIL π value. By the upper-bound property, each vertex on cycle c has a finite shortest-path weight, which implies that it is reachable from s.

We shall examine the shortest-path estimates on c just prior to the call RELAX(v_{k-1}, v_k, w) and show that c is a negative-weight cycle, thereby contradicting the assumption that G contains no negative-weight cycles that are reachable from the source. Just before the call, we have $v_i.\pi = v_{i-1}$ for $i = 1, 2, \ldots, k-1$. Thus, for $i = 1, 2, \ldots, k-1$, the last update to $v_i.d$ was by the assignment $v_i.d = v_{i-1}.d + w(v_{i-1}, v_i)$. If $v_{i-1}.d$ changed since then, it decreased. Therefore, just before the call RELAX(v_{k-1}, v_k, w), we have

$$v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i) \qquad \text{for all } i = 1, 2, \ldots, k-1 . \tag{24.12}$$

Because $v_k.\pi$ is changed by the call, immediately beforehand we also have the strict inequality

$$v_k.d > v_{k-1}.d + w(v_{k-1}, v_k) .$$

Summing this strict inequality with the $k-1$ inequalities (24.12), we obtain the sum of the shortest-path estimates around cycle c:

$$\begin{aligned} \sum_{i=1}^{k} v_i.d &> \sum_{i=1}^{k} (v_{i-1}.d + w(v_{i-1}, v_i)) \\ &= \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i) . \end{aligned}$$

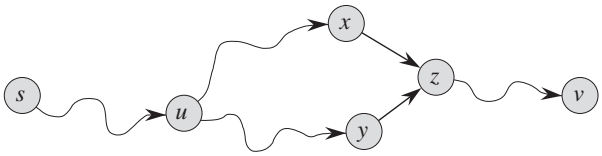

Figure 24.9 Showing that a simple path in G_π from source s to vertex v is unique. If there are two paths p_1 ($s \rightsquigarrow u \rightsquigarrow x \rightarrow z \rightsquigarrow v$) and p_2 ($s \rightsquigarrow u \rightsquigarrow y \rightarrow z \rightsquigarrow v$), where $x \neq y$, then $z.\pi = x$ and $z.\pi = y$, a contradiction.

But

$$\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d \ ,$$

since each vertex in the cycle c appears exactly once in each summation. This equality implies

$$0 > \sum_{i=1}^{k} w(v_{i-1}, v_i) \ .$$

Thus, the sum of weights around the cycle c is negative, which provides the desired contradiction.

We have now proven that G_π is a directed, acyclic graph. To show that it forms a rooted tree with root s, it suffices (see Exercise B.5-2) to prove that for each vertex $v \in V_\pi$, there is a unique simple path from s to v in G_π.

We first must show that a path from s exists for each vertex in V_π. The vertices in V_π are those with non-NIL π values, plus s. The idea here is to prove by induction that a path exists from s to all vertices in V_π. We leave the details as Exercise 24.5-6.

To complete the proof of the lemma, we must now show that for any vertex $v \in V_\pi$, the graph G_π contains at most one simple path from s to v. Suppose otherwise. That is, suppose that, as Figure 24.9 illustrates, G_π contains two simple paths from s to some vertex v: p_1, which we decompose into $s \rightsquigarrow u \rightsquigarrow x \rightarrow z \rightsquigarrow v$, and p_2, which we decompose into $s \rightsquigarrow u \rightsquigarrow y \rightarrow z \rightsquigarrow v$, where $x \neq y$ (though u could be s and z could be v). But then, $z.\pi = x$ and $z.\pi = y$, which implies the contradiction that $x = y$. We conclude that G_π contains a unique simple path from s to v, and thus G_π forms a rooted tree with root s. ■

We can now show that if, after we have performed a sequence of relaxation steps, all vertices have been assigned their true shortest-path weights, then the predecessor subgraph G_π is a shortest-paths tree.

Lemma 24.17 (Predecessor-subgraph property)
Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$, let $s \in V$ be a source vertex, and assume that G contains no negative-weight cycles that are reachable from s. Let us call INITIALIZE-SINGLE-SOURCE(G, s) and then execute any sequence of relaxation steps on edges of G that produces $v.d = \delta(s, v)$ for all $v \in V$. Then, the predecessor subgraph G_π is a shortest-paths tree rooted at s.

Proof We must prove that the three properties of shortest-paths trees given on page 647 hold for G_π. To show the first property, we must show that V_π is the set of vertices reachable from s. By definition, a shortest-path weight $\delta(s, v)$ is finite if and only if v is reachable from s, and thus the vertices that are reachable from s are exactly those with finite d values. But a vertex $v \in V - \{s\}$ has been assigned a finite value for $v.d$ if and only if $v.\pi \neq$ NIL. Thus, the vertices in V_π are exactly those reachable from s.

The second property follows directly from Lemma 24.16.

It remains, therefore, to prove the last property of shortest-paths trees: for each vertex $v \in V_\pi$, the unique simple path $s \overset{p}{\rightsquigarrow} v$ in G_π is a shortest path from s to v in G. Let $p = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = s$ and $v_k = v$. For $i = 1, 2, \ldots, k$, we have both $v_i.d = \delta(s, v_i)$ and $v_i.d \geq v_{i-1}.d + w(v_{i-1}, v_i)$, from which we conclude $w(v_{i-1}, v_i) \leq \delta(s, v_i) - \delta(s, v_{i-1})$. Summing the weights along path p yields

$$\begin{aligned}
w(p) &= \sum_{i=1}^{k} w(v_{i-1}, v_i) \\
&\leq \sum_{i=1}^{k} (\delta(s, v_i) - \delta(s, v_{i-1})) \\
&= \delta(s, v_k) - \delta(s, v_0) && \text{(because the sum telescopes)} \\
&= \delta(s, v_k) && \text{(because } \delta(s, v_0) = \delta(s, s) = 0) \ .
\end{aligned}$$

Thus, $w(p) \leq \delta(s, v_k)$. Since $\delta(s, v_k)$ is a lower bound on the weight of any path from s to v_k, we conclude that $w(p) = \delta(s, v_k)$, and thus p is a shortest path from s to $v = v_k$. ■

Exercises

24.5-1

Give two shortest-paths trees for the directed graph of Figure 24.2 (on page 648) other than the two shown.

24.5-2

Give an example of a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$ and source vertex s such that G satisfies the following property: For every edge $(u, v) \in E$, there is a shortest-paths tree rooted at s that contains (u, v) and another shortest-paths tree rooted at s that does not contain (u, v).

24.5-3

Embellish the proof of Lemma 24.10 to handle cases in which shortest-path weights are ∞ or $-\infty$.

24.5-4

Let $G = (V, E)$ be a weighted, directed graph with source vertex s, and let G be initialized by INITIALIZE-SINGLE-SOURCE(G, s). Prove that if a sequence of relaxation steps sets $s.\pi$ to a non-NIL value, then G contains a negative-weight cycle.

24.5-5

Let $G = (V, E)$ be a weighted, directed graph with no negative-weight edges. Let $s \in V$ be the source vertex, and suppose that we allow $v.\pi$ to be the predecessor of v on *any* shortest path to v from source s if $v \in V - \{s\}$ is reachable from s, and NIL otherwise. Give an example of such a graph G and an assignment of π values that produces a cycle in G_π. (By Lemma 24.16, such an assignment cannot be produced by a sequence of relaxation steps.)

24.5-6

Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$ and no negative-weight cycles. Let $s \in V$ be the source vertex, and let G be initialized by INITIALIZE-SINGLE-SOURCE(G, s). Prove that for every vertex $v \in V_\pi$, there exists a path from s to v in G_π and that this property is maintained as an invariant over any sequence of relaxations.

24.5-7

Let $G = (V, E)$ be a weighted, directed graph that contains no negative-weight cycles. Let $s \in V$ be the source vertex, and let G be initialized by INITIALIZE-SINGLE-SOURCE(G, s). Prove that there exists a sequence of $|V| - 1$ relaxation steps that produces $v.d = \delta(s, v)$ for all $v \in V$.

24.5-8

Let G be an arbitrary weighted, directed graph with a negative-weight cycle reachable from the source vertex s. Show how to construct an infinite sequence of relaxations of the edges of G such that every relaxation causes a shortest-path estimate to change.