# 24 Single-Source Shortest Paths

Professor Patrick wishes to find the shortest possible route from Phoenix to Indianapolis. Given a road map of the United States on which the distance between each pair of adjacent intersections is marked, how can she determine this shortest route?

One possible way would be to enumerate all the routes from Phoenix to Indianapolis, add up the distances on each route, and select the shortest. It is easy to see, however, that even disallowing routes that contain cycles, Professor Patrick would have to examine an enormous number of possibilities, most of which are simply not worth considering. For example, a route from Phoenix to Indianapolis that passes through Seattle is obviously a poor choice, because Seattle is several hundred miles out of the way.

In this chapter and in Chapter 25, we show how to solve such problems efficiently. In a ***shortest-paths problem***, we are given a weighted, directed graph $G = (V, E)$, with weight function $w : E \to \mathbb{R}$ mapping edges to real-valued weights. The ***weight*** $w(p)$ of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges:

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

We define the ***shortest-path weight*** $\delta(u, v)$ from $u$ to $v$ by

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there is a path from } u \text{ to } v , \\ \infty & \text{otherwise} . \end{cases}$$

A ***shortest path*** from vertex $u$ to vertex $v$ is then defined as any path $p$ with weight $w(p) = \delta(u, v)$.

In the Phoenix-to-Indianapolis example, we can model the road map as a graph: vertices represent intersections, edges represent road segments between intersections, and edge weights represent road distances. Our goal is to find a shortest path from a given intersection in Phoenix to a given intersection in Indianapolis.

Edge weights can represent metrics other than distances, such as time, cost, penalties, loss, or any other quantity that accumulates linearly along a path and that we would want to minimize.

The breadth-first-search algorithm from Section 22.2 is a shortest-paths algorithm that works on unweighted graphs, that is, graphs in which each edge has unit weight. Because many of the concepts from breadth-first search arise in the study of shortest paths in weighted graphs, you might want to review Section 22.2 before proceeding.

### Variants

In this chapter, we shall focus on the ***single-source shortest-paths problem***: given a graph $G = (V, E)$, we want to find a shortest path from a given ***source*** vertex $s \in V$ to each vertex $v \in V$. The algorithm for the single-source problem can solve many other problems, including the following variants.

**Single-destination shortest-paths problem:** Find a shortest path to a given ***destination*** vertex $t$ from each vertex $v$. By reversing the direction of each edge in the graph, we can reduce this problem to a single-source problem.

**Single-pair shortest-path problem:** Find a shortest path from $u$ to $v$ for given vertices $u$ and $v$. If we solve the single-source problem with source vertex $u$, we solve this problem also. Moreover, all known algorithms for this problem have the same worst-case asymptotic running time as the best single-source algorithms.

**All-pairs shortest-paths problem:** Find a shortest path from $u$ to $v$ for every pair of vertices $u$ and $v$. Although we can solve this problem by running a single-source algorithm once from each vertex, we usually can solve it faster. Additionally, its structure is interesting in its own right. Chapter 25 addresses the all-pairs problem in detail.

### Optimal substructure of a shortest path

Shortest-paths algorithms typically rely on the property that a shortest path between two vertices contains other shortest paths within it. (The Edmonds-Karp maximum-flow algorithm in Chapter 26 also relies on this property.) Recall that optimal substructure is one of the key indicators that dynamic programming (Chapter 15) and the greedy method (Chapter 16) might apply. Dijkstra's algorithm, which we shall see in Section 24.3, is a greedy algorithm, and the Floyd-Warshall algorithm, which finds shortest paths between all pairs of vertices (see Section 25.2), is a dynamic-programming algorithm. The following lemma states the optimal-substructure property of shortest paths more precisely.

***Lemma 24.1 (Subpaths of shortest paths are shortest paths)***
Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be a shortest path from vertex $v_0$ to vertex $v_k$ and, for any $i$ and $j$ such that $0 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be the subpath of $p$ from vertex $v_i$ to vertex $v_j$. Then, $p_{ij}$ is a shortest path from $v_i$ to $v_j$.

***Proof*** If we decompose path $p$ into $v_0 \overset{p_{0i}}{\leadsto} v_i \overset{p_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$, then we have that $w(p) = w(p_{0i}) + w(p_{ij}) + w(p_{jk})$. Now, assume that there is a path $p'_{ij}$ from $v_i$ to $v_j$ with weight $w(p'_{ij}) < w(p_{ij})$. Then, $v_0 \overset{p_{0i}}{\leadsto} v_i \overset{p'_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$ is a path from $v_0$ to $v_k$ whose weight $w(p_{0i}) + w(p'_{ij}) + w(p_{jk})$ is less than $w(p)$, which contradicts the assumption that $p$ is a shortest path from $v_0$ to $v_k$. ■

### Negative-weight edges

Some instances of the single-source shortest-paths problem may include edges whose weights are negative. If the graph $G = (V, E)$ contains no negative-weight cycles reachable from the source $s$, then for all $v \in V$, the shortest-path weight $\delta(s, v)$ remains well defined, even if it has a negative value. If the graph contains a negative-weight cycle reachable from $s$, however, shortest-path weights are not well defined. No path from $s$ to a vertex on the cycle can be a shortest path—we can always find a path with lower weight by following the proposed "shortest" path and then traversing the negative-weight cycle. If there is a negative-weight cycle on some path from $s$ to $v$, we define $\delta(s, v) = -\infty$.

Figure 24.1 illustrates the effect of negative weights and negative-weight cycles on shortest-path weights. Because there is only one path from $s$ to $a$ (the path $\langle s, a \rangle$), we have $\delta(s, a) = w(s, a) = 3$. Similarly, there is only one path from $s$ to $b$, and so $\delta(s, b) = w(s, a) + w(a, b) = 3 + (-4) = -1$. There are infinitely many paths from $s$ to $c$: $\langle s, c \rangle$, $\langle s, c, d, c \rangle$, $\langle s, c, d, c, d, c \rangle$, and so on. Because the cycle $\langle c, d, c \rangle$ has weight $6 + (-3) = 3 > 0$, the shortest path from $s$ to $c$ is $\langle s, c \rangle$, with weight $\delta(s, c) = w(s, c) = 5$. Similarly, the shortest path from $s$ to $d$ is $\langle s, c, d \rangle$, with weight $\delta(s, d) = w(s, c) + w(c, d) = 11$. Analogously, there are infinitely many paths from $s$ to $e$: $\langle s, e \rangle$, $\langle s, e, f, e \rangle$, $\langle s, e, f, e, f, e \rangle$, and so on. Because the cycle $\langle e, f, e \rangle$ has weight $3 + (-6) = -3 < 0$, however, there is no shortest path from $s$ to $e$. By traversing the negative-weight cycle $\langle e, f, e \rangle$ arbitrarily many times, we can find paths from $s$ to $e$ with arbitrarily large negative weights, and so $\delta(s, e) = -\infty$. Similarly, $\delta(s, f) = -\infty$. Because $g$ is reachable from $f$, we can also find paths with arbitrarily large negative weights from $s$ to $g$, and so $\delta(s, g) = -\infty$. Vertices $h$, $i$, and $j$ also form a negative-weight cycle. They are not reachable from $s$, however, and so $\delta(s, h) = \delta(s, i) = \delta(s, j) = \infty$.

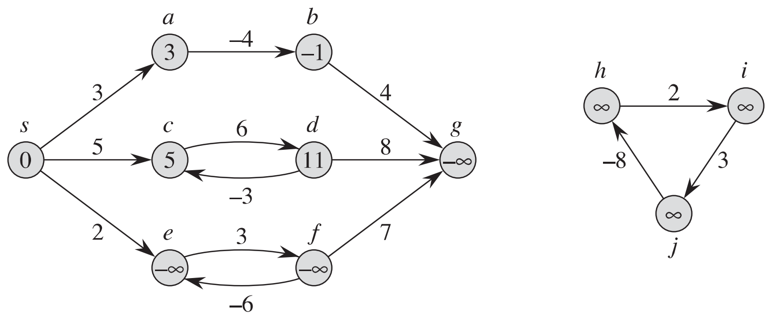


**Figure 24.1** Negative edge weights in a directed graph. The shortest-path weight from source $s$ appears within each vertex. Because vertices $e$ and $f$ form a negative-weight cycle reachable from $s$, they have shortest-path weights of $-\infty$. Because vertex $g$ is reachable from a vertex whose 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 $\infty$, even though they lie on a negative-weight cycle.

Some shortest-paths algorithms, such as Dijkstra's algorithm, assume that all edge weights in the input graph are nonnegative, as in the road-map example. Others, such as the Bellman-Ford algorithm, allow negative-weight edges in the input graph and produce a correct answer as long as no negative-weight cycles are reachable from the source. Typically, if there is such a negative-weight cycle, the algorithm can detect and report its existence.

### Cycles

Can a shortest path contain a cycle? As we have just seen, it cannot contain a negative-weight cycle. Nor can it contain a positive-weight cycle, since removing the cycle from the path produces a path with the same source and destination vertices and a lower path weight. That is, if $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a path and $c = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ is a positive-weight cycle on this path (so that $v_i = v_j$ and $w(c) > 0$), then the path $p' = \langle v_0, v_1, \ldots, v_i, v_{j+1}, v_{j+2}, \ldots, v_k \rangle$ has weight $w(p') = w(p) - w(c) < w(p)$, and so $p$ cannot be a shortest path from $v_0$ to $v_k$.

That leaves only 0-weight cycles. We can remove a 0-weight cycle from any path to produce another path whose weight is the same. Thus, if there is a shortest path from a source vertex $s$ to a destination vertex $v$ that contains a 0-weight cycle, then there is another shortest path from $s$ to $v$ without this cycle. As long as a shortest path has 0-weight cycles, we can repeatedly remove these cycles from the path until we have a shortest path that is cycle-free. Therefore, without loss of generality we can assume that when we are finding shortest paths, they have no cycles, i.e., they are simple paths. Since any acyclic path in a graph $G = (V, E)$

contains at most $|V|$ distinct vertices, it also contains at most $|V| - 1$ edges. Thus, we can restrict our attention to shortest paths of at most $|V| - 1$ edges.

### Representing shortest paths

We often wish to compute not only shortest-path weights, but the vertices on shortest paths as well. We represent shortest paths similarly to how we represented breadth-first trees in Section 22.2. Given a graph $G = (V, E)$, we maintain for each vertex $v \in V$ a ***predecessor*** $v.\pi$ that is either another vertex or NIL. The shortest-paths algorithms in this chapter set the $\pi$ attributes so that the chain of predecessors originating at a vertex $v$ runs backwards along a shortest path from $s$ to $v$. Thus, given a vertex $v$ for which $v.\pi \neq \text{NIL}$, the procedure PRINT-PATH$(G, s, v)$ from Section 22.2 will print a shortest path from $s$ to $v$.

In the midst of executing a shortest-paths algorithm, however, the $\pi$ values might not indicate shortest paths. As in breadth-first search, we shall be interested in the ***predecessor subgraph*** $G_\pi = (V_\pi, E_\pi)$ induced by the $\pi$ values. Here again, we define the vertex set $V_\pi$ to be the set of vertices of $G$ with non-NIL predecessors, plus the source $s$:

$$V_\pi = \{v \in V : v.\pi \neq \text{NIL}\} \cup \{s\} .$$

The directed edge set $E_\pi$ is the set of edges induced by the $\pi$ values for vertices in $V_\pi$:

$$E_\pi = \{(v.\pi, v) \in E : v \in V_\pi - \{s\}\} .$$

We shall prove that the $\pi$ values produced by the algorithms in this chapter have the property that at termination $G_\pi$ is a "shortest-paths tree"—informally, a rooted tree containing a shortest path from the source $s$ to every vertex that is reachable from $s$. A shortest-paths tree is like the breadth-first tree from Section 22.2, but it contains shortest paths from the source defined in terms of edge weights instead of numbers of edges. To be precise, let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \to \mathbb{R}$, and assume that $G$ contains no negative-weight cycles reachable from the source vertex $s \in V$, so that shortest paths are well defined. A ***shortest-paths tree*** rooted at $s$ is a directed subgraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, such that

1. $V'$ is the set of vertices reachable from $s$ in $G$,
2. $G'$ forms a rooted tree with root $s$, and
3. for all $v \in V'$, the unique simple path from $s$ to $v$ in $G'$ is a shortest path from $s$ to $v$ in $G$.

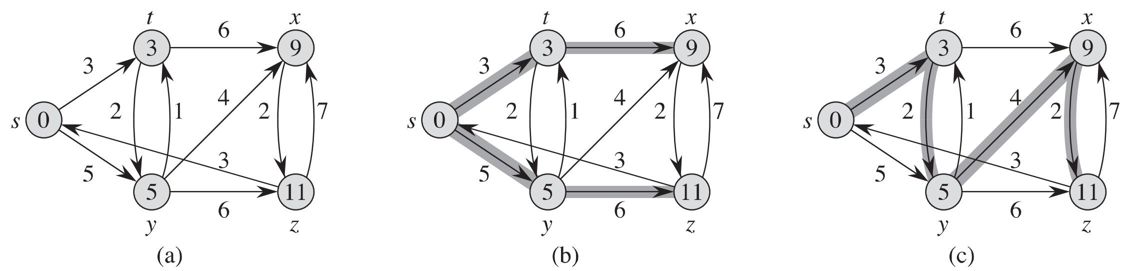


**Figure 24.2** **(a)** A weighted, directed graph with shortest-path weights from source $s$. **(b)** The shaded edges form a shortest-paths tree rooted at the source $s$. **(c)** Another shortest-paths tree with the same root.

Shortest paths are not necessarily unique, and neither are shortest-paths trees. For example, Figure 24.2 shows a weighted, directed graph and two shortest-paths trees with the same root.

## Relaxation

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

```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

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

The process of ***relaxing*** 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 $v.d$ and $v.\pi$. A relaxation step[1] may decrease the value of the shortest-path

[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 $v.d \le u.d + w(u, v)$, which, by the triangle inequality (Lemma 24.10), must be satisfied if $u.d = \delta(s, u)$ and $v.d = \delta(s, v)$. That is, if $v.d \le u.d + w(u, v)$, there is no "pressure" to satisfy this constraint, so the constraint is "relaxed."

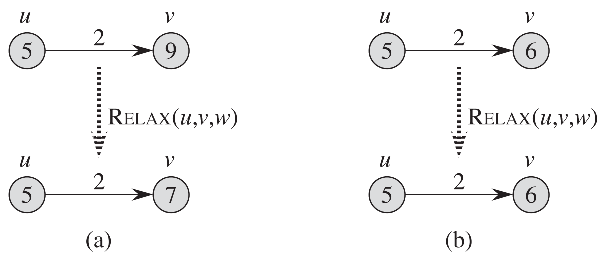


**Figure 24.3** Relaxing an edge $(u, v)$ with weight $w(u, v) = 2$. The shortest-path estimate of each vertex appears within the vertex. **(a)** Because $v.d > u.d + w(u, v)$ prior to relaxation, the value of $v.d$ decreases. **(b)** Here, $v.d \leq u.d + w(u, v)$ before relaxing the edge, and so the relaxation step leaves $v.d$ unchanged.

estimate $v.d$ and update $v$'s predecessor attribute $v.\pi$. The following code performs a relaxation step on edge $(u, v)$ in $O(1)$ time:

RELAX$(u, v, w)$
**if** $v.d > u.d + w(u, v)$
    $v.d = u.d + w(u, v)$
    $v.\pi = u$

Figure 24.3 shows two examples of relaxing an edge, one in which a shortest-path estimate decreases and one in which no estimate changes.

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. Dijkstra's algorithm and the shortest-paths algorithm for directed acyclic graphs relax each edge exactly once. The Bellman-Ford algorithm relaxes each edge $|V| - 1$ times.

### Properties of shortest paths and relaxation

To prove the algorithms in this chapter correct, we shall appeal to several properties of shortest paths and relaxation. We state these properties here, and Section 24.5 proves them formally. For your reference, each property stated here includes the appropriate lemma or corollary number from Section 24.5. The latter five of these properties, which refer to shortest-path estimates or the predecessor subgraph, implicitly assume that the graph is initialized with a call to INITIALIZE-SINGLE-SOURCE$(G, s)$ and that the only way that shortest-path estimates and the predecessor subgraph change are by some sequence of relaxation steps.

**Triangle inequality** (Lemma 24.10)
For any edge $(u, v) \in E$, we have $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

**Upper-bound property** (Lemma 24.11)
We always have $v.d \geq \delta(s, v)$ for all vertices $v \in V$, and once $v.d$ achieves the value $\delta(s, v)$, it never changes.

**No-path property** (Corollary 24.12)
If there is no path from $s$ to $v$, then we always have $v.d = \delta(s, v) = \infty$.

**Convergence property** (Lemma 24.14)
If $s \leadsto u \rightarrow v$ is a shortest path in $G$ for some $u, v \in V$, and if $u.d = \delta(s, u)$ at any time prior to relaxing edge $(u, v)$, then $v.d = \delta(s, v)$ at all times afterward.

**Path-relaxation property** (Lemma 24.15)
If $p = \langle v_0, v_1, \ldots, v_k \rangle$ is a shortest path from $s = v_0$ to $v_k$, and we relax the edges of $p$ in the order $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, then $v_k.d = \delta(s, v_k)$. This property holds regardless of any other relaxation steps that occur, even if they are intermixed with relaxations of the edges of $p$.

**Predecessor-subgraph property** (Lemma 24.17)
Once $v.d = \delta(s, v)$ for all $v \in V$, the predecessor subgraph is a shortest-paths tree rooted at $s$.

### Chapter outline

Section 24.1 presents the Bellman-Ford algorithm, which solves the single-source shortest-paths problem in the general case in which edges can have negative weight. The Bellman-Ford algorithm is remarkably simple, and it has the further benefit of detecting whether a negative-weight cycle is reachable from the source. Section 24.2 gives a linear-time algorithm for computing shortest paths from a single source in a directed acyclic graph. Section 24.3 covers Dijkstra's algorithm, which has a lower running time than the Bellman-Ford algorithm but requires the edge weights to be nonnegative. Section 24.4 shows how we can use the Bellman-Ford algorithm to solve a special case of linear programming. Finally, Section 24.5 proves the properties of shortest paths and relaxation stated above.

We require some conventions for doing arithmetic with infinities. We shall assume that for any real number $a \neq -\infty$, we have $a + \infty = \infty + a = \infty$. Also, to make our proofs hold in the presence of negative-weight cycles, we shall assume that for any real number $a \neq \infty$, we have $a + (-\infty) = (-\infty) + a = -\infty$.

All algorithms in this chapter assume that the directed graph $G$ is stored in the adjacency-list representation. Additionally, stored with each edge is its weight, so that as we traverse each adjacency list, we can determine the edge weights in $O(1)$ time per edge.