Selected Solutions for Chapter 24: Single-Source Shortest Paths

Solution to Exercise 24.1-3

If the greatest number of edges on any shortest path from the source is m, then the path-relaxation property tells us that after m iterations of BELLMAN-FORD, every vertex v has achieved its shortest-path weight in $v.d$. By the upper-bound property, after m iterations, no d values will ever change. Therefore, no d values will change in the $(m + 1)$st iteration. Because we do not know m in advance, we cannot make the algorithm iterate exactly m times and then terminate. But if we just make the algorithm stop when nothing changes any more, it will stop after $m + 1$ iterations.

```
BELLMAN-FORD-(M+1)$(G, w, s)$
INITIALIZE-SINGLE-SOURCE$(G, s)$
changes = TRUE
while changes == TRUE
    changes = FALSE
    for each edge $(u, v) \in G.E$
        RELAX-M$(u, v, w)$
```

```
RELAX-M$(u, v, w)$
if $v.d > u.d + w(u, v)$
    $v.d = u.d + w(u, v)$
    $v.\pi = u$
    changes = TRUE
```

The test for a negative-weight cycle (based on there being a d value that would change if another relaxation step was done) has been removed above, because this version of the algorithm will never get out of the **while** loop unless all d values stop changing.

Solution to Exercise 24.3-3

Yes, the algorithm still works. Let u be the leftover vertex that does not get extracted from the priority queue Q. If u is not reachable from s, then

$u.d = \delta(s,u) = \infty$. If u is reachable from s, then there is a shortest path $p = s \rightsquigarrow x \to u$. When the node x was extracted, $x.d = \delta(s,x)$ and then the edge (x,u) was relaxed; thus, $u.d = \delta(s,u)$.

Solution to Exercise 24.3-6

To find the most reliable path between s and t, run Dijkstra's algorithm with edge weights $w(u,v) = -\lg r(u,v)$ to find shortest paths from s in $O(E+V \lg V)$ time. The most reliable path is the shortest path from s to t, and that path's reliability is the product of the reliabilities of its edges.

Here's why this method works. Because the probabilities are independent, the probability that a path will not fail is the product of the probabilities that its edges will not fail. We want to find a path $s \overset{p}{\rightsquigarrow} t$ such that $\prod_{(u,v)\in p} r(u,v)$ is maximized. This is equivalent to maximizing $\lg(\prod_{(u,v)\in p} r(u,v)) = \sum_{(u,v)\in p} \lg r(u,v)$, which is in turn equivalent to minimizing $\sum_{(u,v)\in p} -\lg r(u,v)$. (Note: $r(u,v)$ can be 0, and $\lg 0$ is undefined. So in this algorithm, define $\lg 0 = -\infty$.) Thus if we assign weights $w(u,v) = -\lg r(u,v)$, we have a shortest-path problem.

Since lg 1 = 0, $\lg x < 0$ for $0 < x < 1$, and we have defined $\lg 0 = -\infty$, all the weights w are nonnegative, and we can use Dijkstra's algorithm to find the shortest paths from s in $O(E + V \lg V)$ time.

Alternate answer

You can also work with the original probabilities by running a modified version of Dijkstra's algorithm that maximizes the product of reliabilities along a path instead of minimizing the sum of weights along a path.

In Dijkstra's algorithm, use the reliabilities as edge weights and substitute

- max (and EXTRACT-MAX) for min (and EXTRACT-MIN) in relaxation and the queue,
- $\cdot$ for $+$ in relaxation,
- 1 (identity for $\cdot$) for 0 (identity for $+$) and $-\infty$ (identity for min) for ∞ (identity for max).

For example, we would use the following instead of the usual RELAX procedure:

```
RELAX-RELIABILITY(u, v, r)
if $v.d < u.d \cdot r(u,v)$
    $v.d = u.d \cdot r(u,v)$
    $v.\pi = u$
```

This algorithm is isomorphic to the one above: it performs the same operations except that it is working with the original probabilities instead of the transformed ones.

Solution to Exercise 24.4-7

Observe that after the first pass, all d values are at most 0, and that relaxing edges (v_0, v_i) will never again change a d value. Therefore, we can eliminate v_0 by running the Bellman-Ford algorithm on the constraint graph without the v_0 node but initializing all shortest path estimates to 0 instead of ∞.

Solution to Exercise 24.5-4

Whenever RELAX sets π for some vertex, it also reduces the vertex's d value. Thus if $s.\pi$ gets set to a non-NIL value, $s.d$ is reduced from its initial value of 0 to a negative number. But $s.d$ is the weight of some path from s to s, which is a cycle including s. Thus, there is a negative-weight cycle.

Solution to Problem 24-3

a. We can use the Bellman-Ford algorithm on a suitable weighted, directed graph $G = (V, E)$, which we form as follows. There is one vertex in V for each currency, and for each pair of currencies c_i and c_j, there are directed edges (v_i, v_j) and (v_j, v_i). (Thus, $|V| = n$ and $|E| = n(n-1)$.)

To determine edge weights, we start by observing that

$$R[i_1, i_2] \cdot R[i_2, i_3] \cdots R[i_{k-1}, i_k] \cdot R[i_k, i_1] > 1$$

if and only if

$$\frac{1}{R[i_1, i_2]} \cdot \frac{1}{R[i_2, i_3]} \cdots \frac{1}{R[i_{k-1}, i_k]} \cdot \frac{1}{R[i_k, i_1]} < 1 .$$

Taking logs of both sides of the inequality above, we express this condition as

$$\lg \frac{1}{R[i_1, i_2]} + \lg \frac{1}{R[i_2, i_3]} + \cdots + \lg \frac{1}{R[i_{k-1}, i_k]} + \lg \frac{1}{R[i_k, i_1]} < 0 .$$

Therefore, if we define the weight of edge (v_i, v_j) as

$$\begin{aligned} w(v_i, v_j) &= \lg \frac{1}{R[i, j]} \\ &= -\lg R[i, j] , \end{aligned}$$

then we want to find whether there exists a negative-weight cycle in G with these edge weights.

We can determine whether there exists a negative-weight cycle in G by adding an extra vertex v_0 with 0-weight edges (v_0, v_i) for all $v_i \in V$, running BELLMAN-FORD from v_0, and using the boolean result of BELLMAN-FORD (which is TRUE if there are no negative-weight cycles and FALSE if there is a negative-weight cycle) to guide our answer. That is, we invert the boolean result of BELLMAN-FORD.

This method works because adding the new vertex ν_0 with 0-weight edges from ν_0 to all other vertices cannot introduce any new cycles, yet it ensures that all negative-weight cycles are reachable from ν_0.

It takes $\Theta(n^2)$ time to create G, which has $\Theta(n^2)$ edges. Then it takes $O(n^3)$ time to run BELLMAN-FORD. Thus, the total time is $O(n^3)$.

Another way to determine whether a negative-weight cycle exists is to create G and, without adding ν_0 and its incident edges, run either of the all-pairs shortest-paths algorithms. If the resulting shortest-path distance matrix has any negative values on the diagonal, then there is a negative-weight cycle.

b. Assuming that we ran BELLMAN-FORD to solve part (a), we only need to find the vertices of a negative-weight cycle. We can do so as follows. First, relax all the edges once more. Since there is a negative-weight cycle, the d value of some vertex u will change. We just need to repeatedly follow the π values until we get back to u. In other words, we can use the recursive method given by the PRINT-PATH procedure of Section 22.2, but stop it when it returns to vertex u.

The running time is $O(n^3)$ to run BELLMAN-FORD, plus $O(n)$ to print the vertices of the cycle, for a total of $O(n^3)$ time.