Introduction To Algorithm
Third Edition
Answer

Xia Ding

December 20, 2015

24.1

24.1–1

It's easy to do it by yourself.

24.1–2

Necessity According to ***Lemma 24.2***, if there is a path from s to v, then BELLMAN-FORD will terminates with $v.d = \delta(s, v) < \infty$.

Sufficiency Because $v.d = \infty$ for all $v \in V$ except s at the begin, if algorithm terminates with $v.d < \infty$, there must be a node v_1 with $v_1.d < \infty$ incident with v. And so on, there must be a node v_n incident with s because s it's the only node whose $d < \infty$ at the begin. So there is a path between s and v.

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. Therefor, no d values will change in the $(m + 1)$st iteration. Because we don't 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.
Algorithm 1 and 2.
The test for a negative-weight cycle has been removed above, because this version of the algorithm will never get out of the **while** loop unless all d values stop changing.

Algorithm 1 FORD-(M+1)(G, w, s)

```
changes = TRUE
while changes == TRUE do
    changes = FALSE
    for all edge (u, v) ∈ G.E do
        RELAX-M(u, v, w)
    end for
end while
```

Algorithm 2 RELAX-M(u, v, w)

```
if v.d > u.d + w(u, v) then
    v.d = u.d + w(u, v)
    v.π = u
    changes = TRUE
end if
```

24.1–4

TODO

24.1–5

TODO

24.1–6

Algorithm ..
In the check stage of Bellman-Ford algorithm, if $v.d > u.d + w(u, v)$, there must be a negative weight cycle $< v, v_1, v_2 \ldots, v_n, u >$, we can use $u.\pi$ to get the node before u and so on until we reach the v, then we get the nodes in that cycle.

24.2

24.2–1

It's easy to do by yourself.

24.2–2

Because the last node if is reachable, it must be relaxed by some nodes incident with it in $|V| - 1$ iterations. So all nodes have been relaxed. It's correct.

24.2–3

TODO

Algorithm 3 NEGATIVE-CYCLE(G)

```
s = RANDOM-SELECT(G.V)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1 do
    for all edge(u, v) ∈ G.E do
        RELAX(u, v, w)
    end for
end for
for all edge(u, v) ∈ G.E do
    if v.d > u.d + w(u, v) then
        t = u
        while t ≠ v do
            print(t)
            t = t.π
        end while
        print(v)
    end if
end for
```

24.2–4

TODO

24.3

24.3–1

It's easy to do by yourself.

24.3–2

Let $V = a, b, c, d,\ E = (a, b, 1), (b, c, -2), (c, b, 1), (b, d, 1)$. When we calculate the shortest path between from a to d, after the procedure, $b.d = 0$, producing incorrect answer.

24.3–3

Yes, the algorithm still works. Let u be the leftover vertex that doesn't get extracted from the priority queue Q. If u isn't reachable from s, then $u.d = \delta(s, u) = \infty$. If u is reachable from s, then there is a shortest path $p = s \leadsto x \to u$. When the node x was extracted, $x.d = \delta(s, x)$ and then the edge (x, u) was relaxed; thus, $u.d = \delta(s, u)$.

24.3–4

Algorithm ..

Algorithm 4 CHECK(G)

```
for i = 1 to |G.V| do
    for all j in G.Adj[i] do
        if j.d > i.d + w(i, j) then
            return FALSE
        end if
    end for
end for
return TRUE
```

24.3–5

$V = a, b, c, d,\ E = (a, d, 9), (a, c, 1), (c, d, 1), (d, b, 1).$

24.3–6

24.3–7

24.3–8

24.3–9

24.3–10

24.4

0.1 24.4–1

0.2 24.4–2

0.3 24.4–3

0.4 24.4–4

0.5 24.4–5

0.6 24.4–6

0.7 24.4–7

0.8 24.4–8

0.9 24.4–9

0.10 24.4–10

0.11 24.4–11

0.12 24.4–12

24.5

0.13 24.5–1

0.14 24.5–2

0.15 24.5–3

0.16 24.5–4

0.17 24.5–5

0.18 24.5–6

0.19 24.5–7

0.20 24.5–8

Problems

0.21 24–1

0.22 24–2

0.23 24–3

0.24 24–4

0.25 24–5

0.26 24–6