25.2-6

How can we use the output of the Floyd-Warshall algorithm to detect the presence of a negative-weight cycle?

25.2-7

Another way to reconstruct shortest paths in the Floyd-Warshall algorithm uses values $\phi_{ij}^{(k)}$ for $i, j, k = 1, 2, \ldots, n$, where $\phi_{ij}^{(k)}$ is the highest-numbered intermediate vertex of a shortest path from i to j in which all intermediate vertices are in the set $\{1, 2, \ldots, k\}$. Give a recursive formulation for $\phi_{ij}^{(k)}$, modify the FLOYD-WARSHALL procedure to compute the $\phi_{ij}^{(k)}$ values, and rewrite the PRINT-ALL-PAIRS-SHORTEST-PATH procedure to take the matrix $\Phi = \left(\phi_{ij}^{(n)}\right)$ as an input. How is the matrix Φ like the s table in the matrix-chain multiplication problem of Section 15.2?

25.2-8

Give an $O(VE)$-time algorithm for computing the transitive closure of a directed graph $G = (V, E)$.

25.2-9

Suppose that we can compute the transitive closure of a directed acyclic graph in $f(|V|, |E|)$ time, where f is a monotonically increasing function of $|V|$ and $|E|$. Show that the time to compute the transitive closure $G^* = (V, E^*)$ of a general directed graph $G = (V, E)$ is then $f(|V|, |E|) + O(V + E^*)$.

25.3 Johnson's algorithm for sparse graphs

Johnson's algorithm finds shortest paths between all pairs in $O(V^2 \lg V + VE)$ time. For sparse graphs, it is asymptotically faster than either repeated squaring of matrices or the Floyd-Warshall algorithm. The algorithm either returns a matrix of shortest-path weights for all pairs of vertices or reports that the input graph contains a negative-weight cycle. Johnson's algorithm uses as subroutines both Dijkstra's algorithm and the Bellman-Ford algorithm, which Chapter 24 describes.

Johnson's algorithm uses the technique of ***reweighting***, which works as follows. If all edge weights w in a graph $G = (V, E)$ are nonnegative, we can find shortest paths between all pairs of vertices by running Dijkstra's algorithm once from each vertex; with the Fibonacci-heap min-priority queue, the running time of this all-pairs algorithm is $O(V^2 \lg V + VE)$. If G has negative-weight edges but no negative-weight cycles, we simply compute a new set of nonnegative edge weights that allows us to use the same method. The new set of edge weights $\widehat{w}$ must satisfy two important properties:

1. For all pairs of vertices $u, v \in V$, a path p is a shortest path from u to v using weight function w if and only if p is also a shortest path from u to v using weight function $\widehat{w}$.
2. For all edges (u, v), the new weight $\widehat{w}(u, v)$ is nonnegative.

As we shall see in a moment, we can preprocess G to determine the new weight function $\widehat{w}$ in $O(VE)$ time.

Preserving shortest paths by reweighting

The following lemma shows how easily we can reweight the edges to satisfy the first property above. We use δ to denote shortest-path weights derived from weight function w and $\widehat{\delta}$ to denote shortest-path weights derived from weight function $\widehat{w}$.

Lemma 25.1 (Reweighting does not change shortest paths)
Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $h : V \to \mathbb{R}$ be any function mapping vertices to real numbers. For each edge $(u, v) \in E$, define

$$\widehat{w}(u, v) = w(u, v) + h(u) - h(v) . \tag{25.9}$$

Let $p = \langle v_0, v_1, \ldots, v_k \rangle$ be any path from vertex v_0 to vertex v_k. Then p is a shortest path from v_0 to v_k with weight function w if and only if it is a shortest path with weight function $\widehat{w}$. That is, $w(p) = \delta(v_0, v_k)$ if and only if $\widehat{w}(p) = \widehat{\delta}(v_0, v_k)$. Furthermore, G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function $\widehat{w}$.

Proof We start by showing that

$$\widehat{w}(p) = w(p) + h(v_0) - h(v_k) . \tag{25.10}$$

We have

$$\begin{aligned}
\widehat{w}(p) &= \sum_{i=1}^{k} \widehat{w}(v_{i-1}, v_i) \\
&= \sum_{i=1}^{k} (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\
&= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_0) - h(v_k) \qquad \text{(because the sum telescopes)} \\
&= w(p) + h(v_0) - h(v_k) .
\end{aligned}$$

Therefore, any path p from v_0 to v_k has $\widehat{w}(p) = w(p) + h(v_0) - h(v_k)$. Because $h(v_0)$ and $h(v_k)$ do not depend on the path, if one path from v_0 to v_k is shorter than another using weight function w, then it is also shorter using $\widehat{w}$. Thus, $w(p) = \delta(v_0, v_k)$ if and only if $\widehat{w}(p) = \widehat{\delta}(v_0, v_k)$.

Finally, we show that G has a negative-weight cycle using weight function w if and only if G has a negative-weight cycle using weight function $\widehat{w}$. Consider any cycle $c = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = v_k$. By equation (25.10),

$$\begin{aligned}\widehat{w}(c) &= w(c) + h(v_0) - h(v_k) \\ &= w(c),\end{aligned}$$

and thus c has negative weight using w if and only if it has negative weight using $\widehat{w}$. ■

Producing nonnegative weights by reweighting

Our next goal is to ensure that the second property holds: we want $\widehat{w}(u, v)$ to be nonnegative for all edges $(u, v) \in E$. Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, we make a new graph $G' = (V', E')$, where $V' = V \cup \{s\}$ for some new vertex $s \notin V$ and $E' = E \cup \{(s, v) : v \in V\}$. We extend the weight function w so that $w(s, v) = 0$ for all $v \in V$. Note that because s has no edges that enter it, no shortest paths in G', other than those with source s, contain s. Moreover, G' has no negative-weight cycles if and only if G has no negative-weight cycles. Figure 25.6(a) shows the graph G' corresponding to the graph G of Figure 25.1.

Now suppose that G and G' have no negative-weight cycles. Let us define $h(v) = \delta(s, v)$ for all $v \in V'$. By the triangle inequality (Lemma 24.10), we have $h(v) \le h(u) + w(u, v)$ for all edges $(u, v) \in E'$. Thus, if we define the new weights $\widehat{w}$ by reweighting according to equation (25.9), we have $\widehat{w}(u, v) = w(u, v) + h(u) - h(v) \ge 0$, and we have satisfied the second property. Figure 25.6(b) shows the graph G' from Figure 25.6(a) with reweighted edges.

Computing all-pairs shortest paths

Johnson's algorithm to compute all-pairs shortest paths uses the Bellman-Ford algorithm (Section 24.1) and Dijkstra's algorithm (Section 24.3) as subroutines. It assumes implicitly that the edges are stored in adjacency lists. The algorithm returns the usual $|V| \times |V|$ matrix $D = d_{ij}$, where $d_{ij} = \delta(i, j)$, or it reports that the input graph contains a negative-weight cycle. As is typical for an all-pairs shortest-paths algorithm, we assume that the vertices are numbered from 1 to $|V|$.

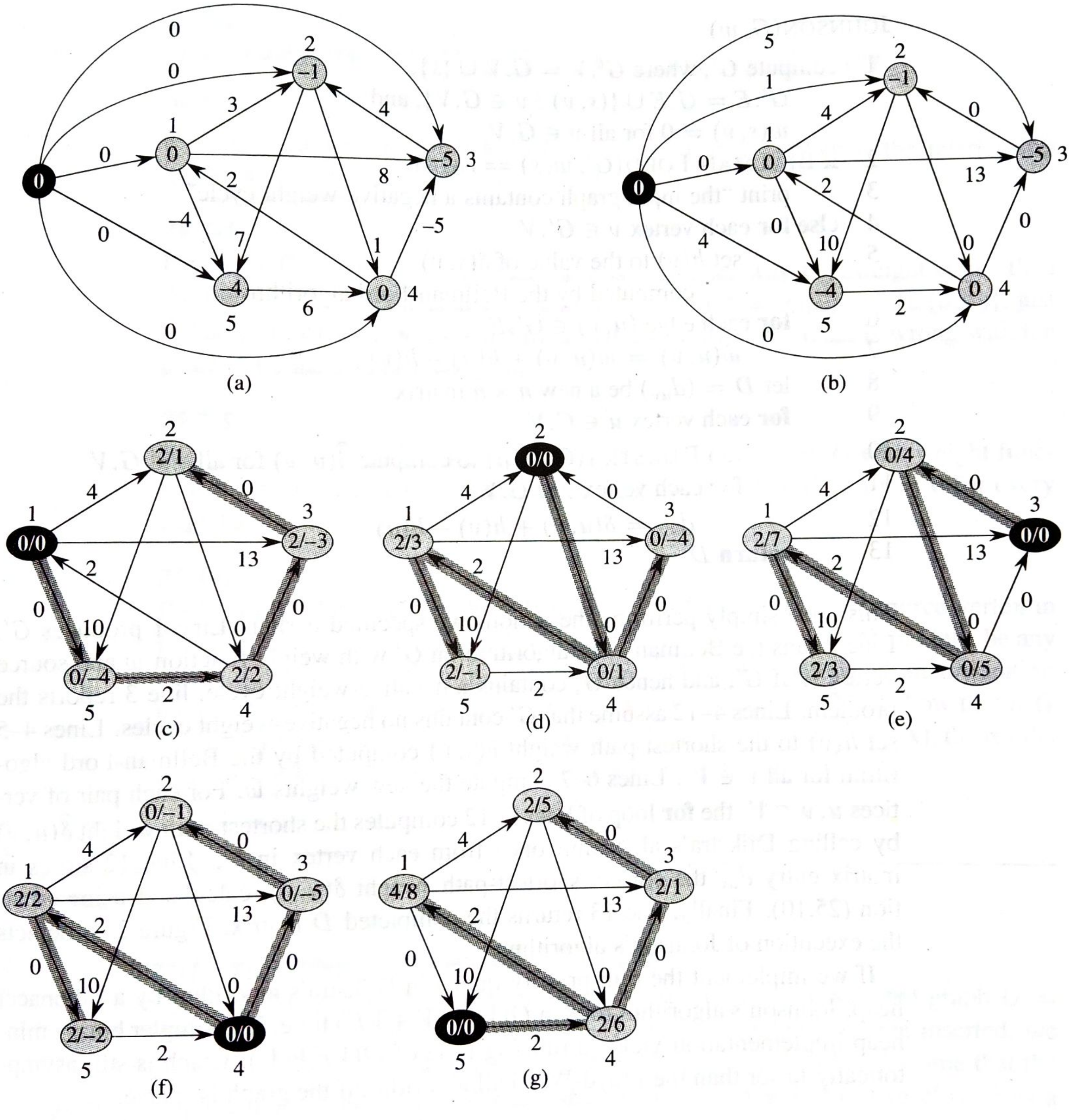

Figure 25.6 Johnson's all-pairs shortest-paths algorithm run on the graph of Figure 25.1. Vertex numbers appear outside the vertices. **(a)** The graph G' with the original weight function w. The new vertex s is black. Within each vertex v is $h(v) = \delta(s, v)$. **(b)** After reweighting each edge (u, v) with weight function $\widehat{w}(u, v) = w(u, v) + h(u) - h(v)$. **(c)–(g)** The result of running Dijkstra's algorithm on each vertex of G using weight function $\widehat{w}$. In each part, the source vertex u is black, and shaded edges are in the shortest-paths tree computed by the algorithm. Within each vertex v are the values $\widehat{\delta}(u, v)$ and $\delta(u, v)$, separated by a slash. The value $d_{uv} = \delta(u, v)$ is equal to $\widehat{\delta}(u, v) + h(v) - h(u)$.

JOHNSON(G, w)

1 compute G', where $G'.V = G.V \cup \{s\}$,
    $G'.E = G.E \cup \{(s, v) : v \in G.V\}$, and
    $w(s, v) = 0$ for all $v \in G.V$
2 **if** BELLMAN-FORD(G', w, s) == FALSE
3   print "the input graph contains a negative-weight cycle"
4 **else for** each vertex $v \in G'.V$
5   set $h(v)$ to the value of $\delta(s, v)$
            computed by the Bellman-Ford algorithm
6   **for** each edge $(u, v) \in G'.E$
7         $\widehat{w}(u, v) = w(u, v) + h(u) - h(v)$
8   let $D = (d_{uv})$ be a new $n \times n$ matrix
9   **for** each vertex $u \in G.V$
10   run DIJKSTRA$(G, \widehat{w}, u)$ to compute $\widehat{\delta}(u, v)$ for all $v \in G.V$
11   **for** each vertex $v \in G.V$
12             $d_{uv} = \widehat{\delta}(u, v) + h(v) - h(u)$
13   **return** D

This code simply performs the actions we specified earlier. Line 1 produces G'. Line 2 runs the Bellman-Ford algorithm on G' with weight function w and source vertex s. If G', and hence G, contains a negative-weight cycle, line 3 reports the problem. Lines 4–12 assume that G' contains no negative-weight cycles. Lines 4–5 set $h(v)$ to the shortest-path weight $\delta(s, v)$ computed by the Bellman-Ford algorithm for all $v \in V'$. Lines 6–7 compute the new weights $\widehat{w}$. For each pair of vertices $u, v \in V$, the **for** loop of lines 9–12 computes the shortest-path weight $\widehat{\delta}(u, v)$ by calling Dijkstra's algorithm once from each vertex in V. Line 12 stores in matrix entry d_{uv} the correct shortest-path weight $\delta(u, v)$, calculated using equation (25.10). Finally, line 13 returns the completed D matrix. Figure 25.6 depicts the execution of Johnson's algorithm.

If we implement the min-priority queue in Dijkstra's algorithm by a Fibonacci heap, Johnson's algorithm runs in $O(V^2 \lg V + VE)$ time. The simpler binary min-heap implementation yields a running time of $O(VE \lg V)$, which is still asymptotically faster than the Floyd-Warshall algorithm if the graph is sparse.

Exercises

25.3-1

Use Johnson's algorithm to find the shortest paths between all pairs of vertices in the graph of Figure 25.2. Show the values of h and $\widehat{w}$ computed by the algorithm.

25.3-2
What is the purpose of adding the new vertex s to V, yielding V'?

25.3-3
Suppose that $w(u, v) \geq 0$ for all edges $(u, v) \in E$. What is the relationship between the weight functions w and $\widehat{w}$?

25.3-4
Professor Greenstreet claims that there is a simpler way to reweight edges than the method used in Johnson's algorithm. Letting $w^* = \min_{(u,v)\in E} \{w(u, v)\}$, just define $\widehat{w}(u, v) = w(u, v) - w^*$ for all edges $(u, v) \in E$. What is wrong with the professor's method of reweighting?

25.3-5
Suppose that we run Johnson's algorithm on a directed graph G with weight function w. Show that if G contains a 0-weight cycle c, then $\widehat{w}(u, v) = 0$ for every edge (u, v) in c.

25.3-6
Professor Michener claims that there is no need to create a new source vertex in line 1 of JOHNSON. He claims that instead we can just use $G' = G$ and let s be any vertex. Give an example of a weighted, directed graph G for which incorporating the professor's idea into JOHNSON causes incorrect answers. Then show that if G is strongly connected (every vertex is reachable from every other vertex), the results returned by JOHNSON with the professor's modification are correct.

Problems

25-1 ***Transitive closure of a dynamic graph***
Suppose that we wish to maintain the transitive closure of a directed graph $G = (V, E)$ as we insert edges into E. That is, after each edge has been inserted, we want to update the transitive closure of the edges inserted so far. Assume that the graph G has no edges initially and that we represent the transitive closure as a boolean matrix.

a. Show how to update the transitive closure $G^* = (V, E^*)$ of a graph $G = (V, E)$ in $O(V^2)$ time when a new edge is added to G.

b. Give an example of a graph G and an edge e such that $\Omega(V^2)$ time is required to update the transitive closure after the insertion of e into G, no matter what algorithm is used.