CSE 5311 Homework 5 Solution

Problem 25.1-3

What does the matrix

$$L^{(0)} = \begin{pmatrix} 0 & \infty & \infty & \dots & \infty \\ \infty & 0 & \infty & \dots & \infty \\ \infty & \infty & 0 & \dots & \infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \infty & \infty & \infty & \dots & 0 \end{pmatrix}$$

used in the shortest-paths algorithms correspond to in regular matrix multiplication?

Answer

The matrix $L^{(0)}$ corresponds to the identity matrix

$$I = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

of regular matrix multiplication. Substitute 0 (the identity for $+$) for 1 (the identity for min), and 1 (the identity for $\cdot$) for 0 (the identity for $+$).

Problem 25.1-5

Show how to express the single-source shortest-paths problem as a product of ma- trices and a vector. Describe how evaluating this product corresponds to a Bellman- Ford-like algorithm (see Section 24.1).

Answer

The all-pairs shortest-paths algorithm in Section 25.1 computes

$$L^{(n-1)=W^{n-1}=L^{(0)}} = L^{(0)} \cdot W^{n-1}$$

where $l_{ij}^{n-1} = \delta(i, j)$ and $L^{(0)}$ is the identity matrix. That is, the entry in the ith row and jth column of the matrix "product" is the shortest-path distance from

vertex i to vertex j, and row i of the product is the solution to the single-source shortest-paths problem for vertex i.

Notice that in a matrix "product" $C = A \cdot B$,the ith row of C is the ith row of A "multiplied" by B. Since all we want is the ith row of C, we never need more than the ith row of A.

Thus the solution to the single-source shortest-paths from vertex i is $L_i^{(0)} \cdot W^{n-1}$, where $L^{(0)}$ is the ith row of $L^{(0)}|a$ vector whose ith entry is 0 and whose other i entries are ∞.

Doing the above "multiplications" starting from the left is essentially the same as the BELLMAN-FORD algorithm. The vector corresponds to the d values in BELLMAN-FORD—the shortest-path estimates from the source to each vertex.

- The vector is initially 0 for the source and 1 for all other vertices, the same as the values set up for d by INITIALIZE-SINGLE-SOURCE.
- Each "multiplication" of the current vector by W relaxes all edges just as BELLMAN-FORD does. That is, a distance estimate in the row, say the distance to v, is updated to a smaller estimate, if any, formed by adding some $w(u, v)$ to the current estimate of the distance to u.
- The relaxation/multiplication is done $n - 1$ times.

Problem 25.2-6

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

Answer

Here are two ways to detect negative-weight cycles:

1. Check the main-diagonal entries of the result matrix for a negative value. There is a negative weight cycle if and only if $d_{ii}^{(n)} < 0$ for some vertex i:
   - $d_{ii}^{(n)}$ is a path weight from i to itself; so if it is negative, there is a path from i to itself (i.e., a cycle), with negative weight.
   - If there is a negative-weight cycle, consider the one with the fewest vertices.
     - If it has just one vertex, then some $w_{ii} < 0$, so d_{ii} starts out negative, and since d values are never increased, it is also negative when the algorithm terminates.
     - If it has at least two vertices, let k be the highest-numbered vertex in the cycle, and let i be some other vertex in the cycle. $d_{ik}^{(k-1)}$ and $d_{ki}^{(k-1)}$ have correct shortest-path weights, because they are not based on negative-weight cycles. (Neither $d_{ik}^{(k-1)}$ nor $d_{ki}^{(k-1)}$ can include k as an intermediate vertex, and i and k are on the negative-weight cycle with the fewest vertices.) Since $i \rightsquigarrow k \rightsquigarrow i$ is a negative-weight cycle, the sum of those two weights

is negative, so $d_{ii}^{(k)}$ will be set to a negative value. Since $d_{ii}^{(k)}$ values are never increased, it is also negative when the algorithm terminates.

In fact, it suffices to check whether $d_{ii}^{(n-1)} < 0$ for some vertex i. Here's why. A negative-weight cycle containing vertex i either contains vertex n or it does not. If it does not, then clearly $d_{ii}^{(n-1)} < 0$. If the negative-weight cycle contains vertex n, then consider $d_{nn}^{(n-1)}$. This value must be negative, since the cycle, nn starting and ending at vertex n, does not include vertex n as an intermediate vertex.

2. Alternatively, one could just run the normal FLOYD-WARSHALL algorithm one extra iteration to see if any of the d values change. If there are negative cycles, then some shortest-path cost will be cheaper. If there are no such cycles, then no d values will change because the algorithm gives the correct shortest paths.

Problem 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 $\hat{w} = w(u,v) - w^*$ for all edges $(u,v) \in E$. What is wrong with the professor's method of reweighting?

Answer

It changes shortest paths. Consider the following graph. $V = \{s, x, y, z\}$ and there are 4 edges: $w(s,x) = 2, w(x,y) = 2, w(s,y) = 5$ and $w(s,z = -10)$. So we'd add 10 to every weight to make $\hat{w}$. With w, the shortest path from s to y is $s \to x \to y$, with weight 4. With $\hat{w}$, the shortest path from s to y is $s \to y$, with weight 15. (The path $s \to x \to y$ has weight 24.) The problem is that by just adding the same amount to every edge, you penalize paths with more edges, even if their weights are low.

Problem 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.

Answer

In this solution, we assume that $\infty - \infty$ is undefined; in particular, it's not 0. Let $G = (V, E)$, where $V = \{s, u\}, E = \{(u, s)\}$ and $w(u, s) = 0$. There is only one edge, and it enters s. When we run Bellman-Ford from s, we get

$h(s) = \delta(s, s) = 0$ and $h(u) = \delta(s, u) = \infty$. When we reweight, we get $\hat{w}(u, s) = 0 + \infty - 0 = \infty$. We compute $\hat{\delta} = \infty$, and so we compute $d_{us} = \infty + 0 - \infty \neq 0$, we get an incorrect answer.

If the graph G is strongly connected, then we get $h(v) = \delta(s, v) < \infty$ for all vertices $v \in V$. Thus, the triangle inequality says that $h(v) \leq h(u) + w(u, v)$ for all edges $(u, v) \in E$, and so $\hat{w}(u, v) = w(u, v) + h(u) - h(v) \geq 0$. Moreover, all edge weights $\hat{w}(u, v)$ used in Lemma 25.1 are finite, and so the lemma holds. Therefore, the conditions we need in order to use Johnson's algorithm hold: that reweighting does not change shortest paths, and that all edge weights $\hat{w}(u, v)$ are nonnegative. Again relying on G being strongly connected, we get that $\hat{\delta}(u, v) < \infty$ for all edges $(u, v) \in E$, which means that $d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$ is finite and correct.

Problem 26.1-6

Professor Adam has two children who, unfortunately, dislike each other. The prob- lem is so severe that not only do they refuse to walk to school together, but in fact each one refuses to walk on any block that the other child has stepped on that day. The children have no problem with their paths crossing at a corner. Fortunately both the professor's house and the school are on corners, but beyond that he is not sure if it is going to be possible to send both of his children to the same school. The professor has a map of his town. Show how to formulate the problem of determining whether both his children can go to the same school as a maximum-flow problem.

Answer

Create a vertex for each corner, and if there is a street between corners u and v, create directed edges (u, v) and v, u. Set the capacity of each edge to 1. Let the source be corner on which the professor's house sits, and let the sink be the corner on which the school is located. We wish to find a flow of value 2 that also has the property that $f(u, v)$ is an integer for all vertices u and v. Such a flow represents two edge-disjoint paths from the house to the school.

Problem 26.2-1

Prove that the summations in equation (26.6) equal the summations in equation (26.7).

Answer

Lemma

1. If $v \notin V_1$, then $f(s, v) = 0$.
2. If $v \notin V_2$, then $f(v, s) = 0$.
3. If $v \notin V_1 \cup V_2$, then $f'(s, v) = 0$.
4. If $v \notin V_1 \cup V_2$, then $f'(v, s) = 0$.

Proof

1. Let $v \notin V_1$ be some vertex. From the definition of V_1, there is no edge from s to v. Thus $f(s, v) = 0$.

2. Let $v \notin V_2$ be some vertex. From the definition of V_2, there is no edge from v to s. Thus $f(v, s) = 0$.

3. Let $v \notin V_1 \cup V_2$ be some vertex. From the definition of V_1 and V_2, neither (s, v) nor (v, s) exists. Therefore, the third condition of the definition of residual capacity (equation (26.2)) applies, and $c_f(s, v) = 0$. Thus, $f'(s, v) = 0$.

4. Let $v \notin V_1 \cup V_2$ be some vertex. By equation (26.2), we have that $c_f(v, s) = 0$ and thus $f'(v, s) = 0$.

We conclude that the summations in equation (26.6) equal the summations in equa- tion (26.7).

Problem 26.2-8

Suppose that we redefine the residual network to disallow edges into s. Argue that the procedure FORD-FULKERSON still correctly computes a maximum flow.

Answer

Let G_f be the residual network just before an iteration of the **while** loop of FORD-FULKERSON, and let E_s be the set of residual edges of G_f into s. We'll show that the augmenting path p chosen by FORD-FULKERSON does not include an edge in E_s. Thus, even if we redefine G_f to disallow edges in E_s, the path p still remains an augmenting path in the redefined network. Since p remains unchanged, an iteration of the **while** loop of FORD-FULKERSON updates the flow in the same way as before the redefinition. Furthermore, by disallowing some edges, we do not introduce any new augmenting paths. Thus, FORD-FULKERSON still correctly computes a maximum flow.

Now, we prove that FORD-FULKERSON never chooses an augmenting path p that includes an edge $(v, s) \in E_s$. Why? The path p always starts from s, and if p included an edge (v, s), the vertex s would be repeated twice in the path. Thus, p would no longer be a simple path. Since FORD-FULKERSON chooses only simple paths, p cannot include (v, s).

Problem 26.2-9

Suppose that both f and f' are flows in a network G and we compute flow $f \uparrow f'$. Does the augmented flow satisfy the flow conservation property? Does it satisfy the capacity constraint?

Answer

The augmented flow $f \uparrow f'$ satisfies the flow conservation property but not the capacity constraint property.

First, we prove that $f \uparrow f'$ satisfies the flow conservation property. We note that if edge $(u, v) \in E$, then $(v, u) \notin E$ and $f(v, u) = 0$. Thus, we can rewrite the definition of flow augmentation (equation (26.4)), when applied to two flows, as

$$f \uparrow f'(u, v) = \begin{cases} f(u, v) + f'(u, v), & if (u, v) \in E \\ 0, & otherwise. \end{cases}$$

The definition implies that the new flow on each edge is simply the sum of the two flows on that edge. We now prove that in $f \uparrow f'$, the net incoming flow for each vertex equals the net outgoing flow. Let $u \notin \{s, t\}$ be any vertex of G. We have

$$\begin{aligned}
&\sum_{v \in V} f \uparrow f'(v, u) \\
&= \sum_{v \in V} f(v, u) + f'(v, u) \\
&= \sum_{v \in V} f(v, u) + \sum_{v \in V} f'(v, u) \\
&= \sum_{v \in V} f(u, v) + \sum_{v \in V} f'(u, v) \qquad \text{(because } f, f' \text{ obey flow conservation)} \\
&= \sum_{v \in V} (f(u, v) + f'(u, v)) \\
&= \sum_{v \in V} f \uparrow f'(u, v)
\end{aligned}$$

We conclude that $f \uparrow f'$ satisfies flow conservation.

We now show that $f \uparrow f'$ need not satisfy the capacity constraint by giving a simple counterexample. Let the flow network G have just a source and a target vertex, with a single edge (s, t) having $c(s, t) = 1$. Define the flows f and f' to have $f(s, t) = f'(s, t) = 1$. Then, we have $(f \uparrow f')(s, t) = 2 > c(s, t)$. We conclude that $f \uparrow f'$ need not satisfy the capacity constraint.

Problem 26.3-3

Prove that the generic push-relabel algorithm spends a total of only $O(VE)$ time in performing all the $O(V^2)$ relabel operations.

Answer

By definition, an augmenting path is a simple path $s \rightsquigarrow t$ in the residual network G'_f. Since G has no edges between vertices in L and no edges between vertices in R, neither does the flow network G' and hence neither does G'_f. Also, the only edges involving s or t connect s to L and R to t. Note that although edges in G' can go only from L to R, edges in G'_f can also go from R to L.

Thus any augmenting path must go

$$s \to L \to R \to \cdots \to L \to R \to t.$$

crossing back and forth between L and R at most as many times as it can do so without using a vertex twice. It contains s, t, and equal numbers of distinct vertices from L and R—at most $2 + 2 \cdot min(|L|, |R|)$ vertices in all. The length of an augmenting path (i.e., its number of edges) is thus bounded above by $2 \cdot \min(|L, |R|) + 1$.