Chapter 25

Michelle Bodnar, Andrew Lohr

April 20, 2016

Exercise 25.1-1

First, the slow way:

$$L^{(1)} = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

$$L^{(2)} = \begin{pmatrix} 0 & 6 & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & 0 & \infty \\ 3 & -3 & 0 & 4 & \infty & -8 \\ -4 & 10 & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & \infty & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -2 & -3 & 0 & -1 & 2 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$$

$$L^{(4)} = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -3 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

$$L^{(5)} = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

Then, since we have reached $L^{(n-1)}$ we can stop since we know that since there are no negative weight cycles, taking higher powers will not cause the matrix to change at all. By the fast method, we get that

$$L^{(1)} = \begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

$$L^{(2)} = \begin{pmatrix} 0 & 6 & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & 0 & \infty \\ 3 & -3 & 0 & 4 & \infty & -8 \\ -4 & 10 & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & \infty & 0 \end{pmatrix}$$

$$L^{(4)} = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -3 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

$$L^{(8)} = \begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

We stop since $8 \geq 5 = n - 1$.

Exercise 25.1-2

This is consistent with the fact that the shortest path from a vertex to itself is the empty path of weight 0. If there were another path of weight less than 0 then it must be a negative-weight cycle, since it starts and ends at v_i.

Exercise 25.1-3

This matrix corresponds to the identity matrix in normal matrix multiplication. This is because anytime that we take a min of any number with infinity, we get that same number back. Another way of seeing this is that we can interpret this strange version of matrix multipllication as allowing any of our shortest paths to be a path described by one matrix followed by a path described by the other. However, the given matrix corresponds to there being no paths between

any of the vertices. This means that we won't change any of the shortest paths that are described by the matrix we are multiplying it by since we are allowing nothing more for those paths.

Exercise 25.1-4

To verify associativity, we need to check that $(W^iW^j)W^p = W^i(W^jW^p)$ for all i, j, p, where we use the matrix multiplication defined by the EXTEND-SHORTEST-PATHS procedure. Consider entry (a, b) of the left hand side. This is:

$$\begin{aligned}\min_{1\leq k\leq n}[W^iW^j]_{a,k} + W^p_{k,b} &= \min_{1\leq k\leq n}\min_{1\leq q\leq n} W^i_{a,q} + W^j_{q,k} + W^p_{k,b} \\ &= \min_{1\leq q\leq n} W^i_{a,q} + \min_{1\leq k\leq n} W^j_{q,k} + W^p_{k,b} \\ &= \min_{1\leq q\leq n} W^i_{a,q} + [W^jW^p]_{q,b}\end{aligned}$$

which is precisely entry (a, b) of the right hand side.

Exercise 25.1-5

We can express finding the shortest path from a single vertex with the modified version of matrix multiplication described in the section. We initially let v_1 be a vector indexed by the vertices of the graph. It is infinity when the corresponding vertex has no edge going to it from the source vertex, s. It is the weight of the edge going to it from s if there is one. Lastly, it is zero in the entry corresponding to s itself. Essentially, we are only taking the row of the W matrix that corresponds to s. Then, we define $v_{i+1} = v_iW$. Then, we stop computing v_i once we compute v_{n-1}. This vector then contains the correct shortest distances from the source to each vertex. Since each time we multiply the vector by the matrix, we only have to consider the entries which are non-infinite in W. There are only $|E| + |V|$ of these non-finfinite entries. So, we have that the time required for each time we mutiply the vector by the matrix, we take time at most $O(E)$. So, the total runtime would be $O(EV)$ just as in Bellman-Ford. The similarities don't stop there however. This is because v_i represents the shortest distance to each vector from s using at most i edges, and each time we multiply by W corresponds to relaxing every edge.

Exercise 25.1-6

For each source vertex v_i we need to compute the shortest-paths tree for v_i. To do this, we need to compute the predecessor for each $j \neq i$. For fixed i and j, this is the value of k such that $L_{i,k} + w(k, j) = L_{i,j}$. Since there are n vertices whose trees need computing, n vertices for each such tree whose predecessors need computing, and it takes $O(n)$ to compute this for each one (checking each possible k), the total time is $O(n^3)$.

Exercise 25.1-7

To have the procedure compute the predecessor along the shortest path, see the modified procedures, EXTEND-SHORTEST-PATH-MOD and SLOW-ALL-PAIRS-SHORTEST-PATHS-MOD

Algorithm 1 EXTEND-SHORTEST-PATH-MOD(Π, L,W)

```
n= L.rows
Let L' = (l'_ij) be a new n × n matrix.
Π' = (π'_ij) is a new n × n matrix
for i=1 to n do
    for j = 1 to n do
        l'_ij = ∞
        π_ij = NIL
        for  k=1 to n do
            if l_ik + l_jk < l_ij then
                l_ij = l_ik + l_jk
                π'_ij = π_kj
            end if
        end for
    end for
end for
return Π', L'
```

Algorithm 2 SLOW-ALL-PAIRS-SHORTEST-PATHS-MOD(W)

```
n= W.rows
L^(1) = W
Π^(1) = (π^(1)_ij) where π^(1)_ij = i if there is an edge from i to j, and NIL otherwise.
for m=2 to n-1 do
    Π^(m), L^(m) = EXTEND-SHORTEST-PATH-MOD(Π^(m−1), L^(m−1),W)
end for
return Π^(n−1), L^(n−1)
```

Exercise 25.1-8

We can overwrite matrices as we go. Let $A \star B$ denote multiplication defined by the EXTEND-SHORTEST-PATHS procedure. Then we modify FASTER-ALL-EXTEND-SHORTEST-PATHS(W). We initially create an n by n matrix L. Delete line 5 of the algorithm, and change line 6 to $L = W \star W$, followed by $W = L$.

Exercise 25.1-9

For the modification, keep computing for one step more than the original, that is, we compute all the way up to $L^{(2^{k+1})}$ where $2^k > n - 1$. Then, if

there aren't any negative weight cycles, then, we will have that the two matrices should be equal since having no negative weight cycles means that between any two vertices, there is a path that is tied for shortest and contains at most $n-1$ edges. However, if there is a cycle of negative total weight, we know that it's length is at most n, so, since we are allowing paths to be larger by $2^k \geq n$ between these two matrices, we have that we would need to have all of the vertices on the cycle have their distance reduce by at least the negative weight of the cycle. Since we can detect exactly when there is a negative cycle, based on when these two matrices are different. This algorithm works. It also only takes time equal to a single matrix multiplication which is littlee oh of the unmodified algorithm.

Exercise 25.1-10

A negative-weight cycle appears when $W_{i,i}^m < 0$ for some m and i. Each time a new *power* of W is computed, we simply check whether or not this happens, at which point the cycle has length m. The runtime is $O(n^4)$.

Exercise 25.2-1

k	D^k
0	$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$
1	$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ \infty & 2 & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$
2	$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$
3	$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ 1 & 0 & \infty & 2 & 0 & \infty \\ 3 & 2 & 0 & 4 & 2 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 8 & 7 & \infty & 9 & 0 & \infty \\ 6 & 5 & 10 & 7 & 5 & 0 \end{pmatrix}$
4	$\begin{pmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & \infty & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$
5	$\begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ 0 & 2 & 0 & 4 & -1 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$
6	$\begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$

Exercise 25.2-2

We set $w_{ij} = 1$ if (i, j) is an edge, and $w_{ij} = 0$ otherwise. Then we replace line 7 of EXTEND-SHORTEST-PATHS(L,W) by $l'_{ij} = l'_{ij} \vee (l_{ik} \wedge w_{kj})$. Then run the SLOW-ALL-PAIRS-SHORTEST-PATHS algorithm.

Exercise 25.2-3

See the modified version of the Floyd-Warshall algorithm:

Algorithm 3 MOD-FLOYD-WARSHALL(W)

```
n= W.rows
D^0 = W
π^0 is a matrix with nil in every entry
for i=1 to n do
    for j = 1 to n do
        if i ≠ j and D^0_{i,j} < ∞ then
            π^0_{i,j} = i
        end if
    end for
end for
for  k=1 to n do
    let D^k be a new n × n matrix.
    let π^k be a new n × n matrix
    for i=1 to n do
        for j = 1 to n do
            if d^{k-1}_{ij} ≤ d^{k-1}_{i,k} + d^{k-1}_{k,j} then
                d^k_{i,j} = d^{k-1}_{i,j}
                π^k_{i,j} = π^{k-1}_{i,j}
            else
                d^k_{i,j} = d^{k-1}_{i,k} + d^{k-1}_{k,j}
                π^k_{i,j} = π^{k-1}_{k,j}
            end if
        end for
    end for
end for
```

In order to have that $\pi^{(k)}_{ij} = l$, we need that $d^{(k)}_{ij} \geq d^{(k)}_{il} + w_{lj}$. To see this fact, we will note that having $\pi^{(k)}_{ij} = l$ means that a shortest path from i to j last goes through l. A path that last goes through l corresponds to taking a chepest path from i to l and then following the single edge from l to j. However, This means that $d_{il} \leq d_{ij} - w_{ij}$, which we can rearrange to get the desired inequality. We can just continue following this inequality around, and if we ever get some cycle, $i_1, i_2, \ldots i_c$, then we would have that $d_{ii_1} \leq d_{ii_1} + w_{i_1 i_2} + w_{i_2 i_3} + \cdots + w_{i_c i_1}$. So, if we subtract the common term from both sides, we get that $0 \leq w_{i_c i_1} + \sum_{q=1}^{c-1} w_{i_q i_{q+1}}$. So, we have that we

would only have a cycle in the precedessor graph if we ahvt that there is a zero weight cycle in the original graph. However, we would never have to go around the weight zero cycle since the constructed path of shortest weight favors ones with a fewer number of edges because of the way that we handle the equality case in equation (25.7).

Exercise 25.2-4

Suppose we are updating $d_{ij}^{(}k)$. To be sure we get the same results from this algorithm, we need to check what happens to $d_{ij}^{(k-1)}$, $d_{ik}^{(k-1)}$ and $d_{kj}^{(k-1)}$. The $d_{ij}^{(k-1)}$ term will be unchanged. On the other hand, the other terms won't change because any shorest path from i to k which includes k necessarily includes it only once (since there are no negative-weight cycles) so it is the length of a shortest path using only vertices 1 through $k-1$. Thus, updating in place is okay.

Exercise 25.2-5

If we change the way that we handle the equality case, we willl still be generating a the correct values for the π matrix. This is because updating the π values to make paths that are longer but still tied for the lowest weight. Making $\pi_{ij} = \pi_{kj}$ means that we are making the shortest path from i to j passes through k at some point. This has the same cost as just going from i to j, since $d_{ij} = d_{ik} + d_{kj}$.

Exercise 25.2-6

If there was a negative-weight cycle, there would be a negative number occurring on the diagonal upon termination of the Floyd-Warshall algorithm.

Exercise 25.2-7
We can recursively compute the values of $\phi_{ij}^{(k)}$ by, letting it be $\phi_{ij}^{(k-1)}$ if $d_{ik}^{(k)} + d_{kj}^{(k)} \geq d_{ij}^{(k-1)}$, and otherwise, let it be k. This works correctly because it perfectly captures whether we decided to use vertex k when we were repeatedly allowing ourselves use of each vertex one at a time. To modify Floyd-Warshall to compute this, we would just need to stick within the innermost for loop, something that computes $\phi_{ij}^{(k)}$ by this recursive rule, this would only be a constant amount of work in this innermost for loop, and so would not cause the asymptotic runtime to increase. It is similar to the s table in matrix-chain multiplication because it is computed by a similar recurrence.

If we already have the n^3 values in $\phi_{ij}^{(k)}$ provided, then we can reconstruct the shortest path from i to j because we know that the largest vertex in the path from i to j is $\phi_{ij}^{(n)}$, call it a_1. Then, we know that the largest vertex in the path before a_1 will be $\phi_{ia_1}^{(a_1-1)}$ and the largest after a_1 will be $\phi_{a_1j}^{(a_1-1}$. By

continuing to recurse until we get that the largest element showing up at some point is NIL, we will be able to continue subdividing the path until it is entirely constructed.

Exercise 25.2-8

Create an n by n matrix A filled with 0's. We are done if we can determine the vertices reachable from a particular vertex in $O(E)$ time, since we can just compute this for each $v \in V$. To do this, assign each edge weight 1. Then we have $\delta(v, u) \leq |E|$ for all $u \in V$. By Problem 24-4 (a) we can compute $\delta(v, u)$ in $O(E)$ for all $u \in V$. If $\delta(v, u) < \infty$, set $A_{ij} = 1$. Otherwise, leave it as 0.

Exercise 25.2-9

First, compute the strongly connected components of the directed graph, and look at it's component graph. This component graph is going to be acyclic and have at most as many vertices and at most as many edges as the original graph. Since it is acyclic, we can run our transitive closure algorithm on it. Then, for every edge (S_1, S_2) that shows up in the transitive closure of the component graph, we add an edge from each vertex in S_1 to a vertex in S_2. This takes time equal to $O(V+E^*)$. So, the total time required is $\leq f(|V|, |E|)+O(V+E)$.

Exercise 25.3-1

v	$h(v)$
1	-5
2	-3
3	0
4	-1
5	-6
6	-8

u	v	$\hat{w}(u,v)$
1	2	NIL
1	3	NIL
1	4	NIL
1	5	0
1	6	NIL
2	1	3
2	3	NIL
2	4	0
2	5	NIL
2	6	NIL
3	1	NIL
3	2	5
3	4	NIL
3	5	NIL
3	6	0
4	1	0
4	2	NIL
4	3	NIL
4	5	8
4	6	NIL
5	1	NIL
5	2	4
5	3	NIL
5	4	NIL
5	6	NIL
6	1	NIL
6	2	0
6	3	18
6	4	NIL
6	5	NIL

So, the $d_{i,j}$ values that we get are

$$\begin{pmatrix} 0 & 6 & \infty & 8 & -1 & \infty \\ -2 & 0 & \infty & 2 & -3 & \infty \\ -5 & -3 & 0 & -1 & -6 & -8 \\ -4 & 2 & \infty & 0 & -5 & \infty \\ 5 & 7 & \infty & 9 & 0 & \infty \\ 3 & 5 & 10 & 7 & 2 & 0 \end{pmatrix}$$

Exercise 25.3-2

This is only important when there are negative-weight cycles in the graph. Using a dummy vertex gets us around the problem of trying to compute $-\infty+\infty$ to find $\hat{w}$. Moreover, if we had instead used a vertex v in the graph instead of

the new vertex s, then we run into trouble if a vertex fails to be reachable from v.

Exercise 25.3-3

If all the edge weights are nonnegative, then the values computed as the shortest distances when running Bellman-Ford will be all zero. This is because when constructing G' on the first line of Johnson's algorithm, we place an edge of weight zero from s to every other vertex. Since any path within the graph has no negative edges, its cost cannot be negative, and so, cannot beat the trivial path that goes straight from s to any given vertex. Since we have that $h(u) = h(v)$ for every u and v, the reweighting that occurs only adds and subtracts 0, and so we have that $w(u, v) = \hat{w}(u, v)$

Exercise 25.3-4

This doesn't preserve shortest paths. Suppose we have a graph with vertices a, b, c, d, e, and f and edges $(a, b), (b, c), (c, d), (d, e), (a, e), (a, f)$ with weights -1, -1, -1, -1, -1, 1, and -2. Professor Greenstreet would add 2 to every edge weight in the graph. Originally, the shortest path from a to e went through vertices b, c, and d. With the added weight, the new shortest path goes directly from a to e. Thus, property 1 of $\hat{w}$ is violated.

Exercise 25.3-5

By lemma 25.1, we have that the total weight of any cycles is unchanged as a result of the reweighting procedure. This can be seen in a way similar to how the last claim of lemma 25.1 was proven. Namely, we consider the cycle c as a path that has the same starting and ending vertices, so, by the first half od lemma 25.1, we have that

$$\hat{w}(c) = w(c) + h(v_0) - h(v_k) = w(c) = 0$$

This means that in the reweighted graph, we still have that the same cycle as before had a total weight of zero. Since there are no longer any negative weight edges after we reweight, this is precicely the second property of the reweighting procedure shown in the section. Since we have that the sum of all the edge weights in c is still equal to zero, but each of them individually has a non-negative weight, it must be the case that each of them individually is equal to 0.

Exercise 25.3-6

Let G have vertices a, b, and c, and edge (b, c) with weight 1, and let $s = a$. When we try to compute $\hat{w}(b, c)$, it is undefined because $h(c) = h(b) = \infty$. Now suppose that G is strongly connected, and further that there are no negative-weight cycles. Since every vertex is reachable from every other, h is well-defined

for any choice of source vertex. We need only check that $\hat{w}$ satisfies properties 1 and 2. For the first property, let $p = \langle u = v_0, v_1, \ldots, v_k = v \rangle$ be a shortest path from u to v using the weight function w. Then we have

$$\begin{aligned}
\hat{w}(p) &= \sum_{i=1}^{k} \hat{w}(v_{i-1}, v_i) \\
&= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i) \\
&= \left(\sum_{i=1}^{k} w(v_{i-1}, v_i) \right) + h(u) - h(v) \\
&= w(p) + h(u) - h(v).
\end{aligned}$$

Since $h(u)$ and $h(v)$ are independent of p, we must have that w minimizes $w(p)$ if and only if $\hat{w}$ minimizes $\hat{w}(p)$. For the second property, the triangle inequality tells us that for any vertices $v, u \in V$ we have $\delta(v, u) \leq \delta(v, z) + w(z, u)$. Thus, if we define $h(u) = \delta(v, u)$ then we have $\hat{w}(z, u) = w(z, u) + h(z) - h(u) \geq 0$.

Problem 25-1

a. We can update the transitive closure in time $O(V^2)$ as follows. Suppose that we add the edge (x_1, x_2). Then, we will consider every pair of vertices (u, v). In order to of created a path between them, we would need some part of that path that goes from u to x_1 and some second part of that path that goes from x_2 to v. This means that we add the edge (u, v) to the transitive closure if and only if the transitive closure contains the edges (u, x_1) and (x_2, v). Since we only had to consider every pair of vertices once, the runtime of this update is only $O(V^2)$.

b. Suppose that we currently have two strongly connected components, each of size $|V|/2$ with no edges between them. Then their transitive closures computed so far will consist of two complete directed graphs on $|V|/2$ vertices each. So, there will be a total of $|V|^2/2$ edges adding the number of edges in each together.

   Then, we add a single edge from one component to the other. This will mean that every vertex in the component the edge is coming from will have an edge going to every vertex in the component that the edge is going to. So, the total number of edges after this operation will be $|V|/2 + |V|/4$. So, the number of edges increased by $|V|/4$. Since each time we add an edge, we need to use at least constant time, since there is no cheap way to add many edges at once, the total amount of time needed is $\Omega(|V|^2)$.

c. We will have each vertex maintain a tree of vertices that have a path to it and a tree of vertices that it has a path to. The second of which is the transitive

closure at each step. Then, upon inserting an edge, (u,v), we will look at successive ancestors of u, and add v to their successor tree, just past u. If we ever don't insert an edge when doing this, we can stop exploring that branch of the ancestor tree. Similarly, we keep doing this for all of the ancestors of v. Since we are able to short circuit if we ever notice that we have already added an edge, we know that we will only ever reconsider the same edge at most n times, and, since the number of edges is $O(n^2)$, the total runtime is $O(n^3)$.

Problem 25-2

a. As in problem 6-2, the runtimes for a d-ary min heap are the same as for a d-ary max heap. The runtimes of INSERT, EXTRACT-MIN, and DECREASE-KEY are $O(\log_d n)$, $O(d \log_d n)$, and $O(\log_d n)$ respectively. If $d = \Theta(n^\alpha)$ these become $O(1/\alpha)$, $O(n^\alpha/\alpha)$, and $O(1/\alpha)$ respectively. The amortized costs for these operations in a FIbonacci heap are $O(1)$, $O(\lg n)$, and $O(1)$ respectively.

b. Choose $d = n^\varepsilon$. Then implement Dijkstra's algorithm using a d-ary min-heap. The analysis from part (a) tells us that the runtime will be $O(n/\varepsilon + 2n^{1+\varepsilon}/\varepsilon) = O(n^{1+\varepsilon}) = O(E)$.

c. Run the algorithm from part (b) once for each vertex of the graph.

d. Using the methods from section 25.3, create the graph G' with all nonnegative edge weights using the weight function $\hat{w}$, proceed as in part (c), then convert back to the original weight function.