Rishad Rahman (Deepak's Recitation) April 3, 2015

6.046 Problem 6-1

Collaborators: *None*

(a) The case where $r = w_{i,j}$ is trivial, the graph is the same and so the shortest paths don't change. Otherwise...

I. $r < w_{i,j}$

Algorithm: $\forall m, n$ $D_{mn} \leftarrow \min(D_{mi} + r + D_{jn}, D_{mn})$ If D_{mn} changes then update $\Pi_{mn} \leftarrow \Pi_{jn}$.

Correctness: Note D_{mi}, D_{jn} do not get updated since $D_{mi} < D_{mi} + r + D_{ji}$ and vice versa. Suppose $D_{mi} + r + D_{jn} < D_{mn}$ but D_{mn} itself is less than any path that does not contain (i, j) because it is the previous shortest path. So the shortest path contains (i, j) and the distance is $D_{mi} + r + D_{jn}$ by the shortest path property so D updates correctly. Since now our path now goes from $j \to n$, Π updates correctly. If $D_{mn} < D_{mi} + r + D_{jn}$ then D_{mn} is less than the distance of the shortest path which contains (i, j) and is also less than the distance of the any path which does not contain (i, j) since it is the previous shortest path hence D and Π do not get updated as expected.

Complexity: We do constant work for all m, n pairs so the time complexity is $\Theta(V^2)$.

II. $r > w_{i,j}$

Algorithm: Perform all pairs Dijsktra. Not fruitful but there aren't many options since if $r > w_{i,j}$, there may now exist multiple path candidates from $m \to n$ which have a lower path weight than the one containing (i, j) i.e. if r is very large so that (i, j) has the highest weight, so it seems like we would have to consider the shortest paths from all pairs anyways.

Correctness: Follows from correctness of Dijsktra and preconditions of the problem.

Complexity: $\Theta(V^2 \log V + EV) = O(V^3)$ worst case.

(b) Consider

- $V = \{1, 2, ..., n, n+1, n+2\}$

- $\forall\ i \mathrel{!}= j \in 1, 2, ..., n$ we have $(i, n+1), (n+2, j), (i, j) \in E$ the former two with weights 1 and the last with weight 4
- $(n+1, n+2) \in E$ with weight 3

$$D = \begin{pmatrix} 0 & 4 & \cdots & 4 & 1 & 0 \\ 4 & 0 & \cdots & 4 & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 4 & 4 & \cdots & 0 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 0 & 3 \\ 1 & 1 & \cdots & 1 & 0 & 0 \end{pmatrix}$$

which follows since (i, j) is weight 4 while $(i, n+1, n+2, j)$ is weight 5. However if we do a dynamic update on parameters $(n+1, n+2, 1)$ then the latter weight changes to 3 so all the 4's in D would update to 3 which takes $(n-1)n = n^2 - n$ time hence any dynamic APSP algorithm takes $\Omega(V^2)$ time on this graph.

(c) We now define D_{ij}^{kh} as the shortest path distance from i to j using only vertices from $1, 2, ..., k$ and with at most h hops. We have the following recursive formula:

$$D_{ij}^{kh} = \min(D_{ij}^{k-1,h}, \min_{0 \le m \le h}(D_{i,k}^{k-1,m} + D_{k,j}^{k-1,h-m}))$$

The recurrence arises from the fact that going to k with at most m hops and leaving from k with at most $h-m$ hops guarantees the total number of hops is at most h. Any path of length m going through k must satisfy this. Now there are V^3h subproblems which take h time to complete hence the complexity is $\Theta(V^3h^2)$.

(d) The algorithm is almost identical to that in section 25.1 except that we exponentiate to $L^{(h)} = W^h$ instead of W^{n-1} since $L^{(m)}$ is defined as the distance matrix for paths that contain at most m edges. As a result we get $\Theta(V^3 \lg h)$ runtime instead of $\Theta(V^3 \lg V)$.

(e) The case where $r = w_{i,j}$ is trivial, the graph is the same and so the shortest paths don't change. For the other cases we assume as input or available as a resource, $D^{(p)}$ and $D^{(p)}$ $\forall\ m, n$ and for $0 \le p \le h-1$ where $D_{ij}^{(p)}$ is the minimum path distance from i to j using $\le p$ hops.

I. $r < w_{i,j}$

Algorithm: $\forall m, n\ D_{mn} \leftarrow \min(\min_{0 \le p \le h-1}(D_{mi}^{(p)} + r + D_{jn}^{(h-1-p)}), D_{mn})$. But now we have to update $D^{(p)}$ for all p. But these are just smaller instances of our problem so we repeat the algorithm for $p = 0$ to $h-1$ to compute $D^{(p)}$

Correctness: Similar to (a) we havat the D_{mi}, D_{jn} are unaffected. The correctness argument is essentially the same but we need the additional resources of $D_{mi}^{(p)}$ and $D_{jn}^{(p)}$ because the shortest path candidate from m to n containing (i, j) using at most h hops must contain the shortest path from m to i and shortest path from j to n where the total number of hops is $h - 1$. Correcting $D^{(p)}$ is the same argument.s 0

Complexity: We do h work for all m, n pairs but there are h matrices to update so the time complexity is $\Theta(V^2h^2)$.

II. $r > w_{i,j}$

Algorithm: Perform the matrix multiplication to compute $D^{(p)}$ for all $0 \leq p \leq h$ using the updated weight.

Correctness: Follows from (c).

Complexity: $\Theta(V^3h)$ since we are computing up to W^h where W is the weight matrix.

III.

NOTE: We can use one matrix multiplication instead to get $\Theta(V^3 \log h)$ runtime but this doesn't update resources so we'd have to stick to this for both cases if we decide to do this. The algorithm above basically sacrifices a little bit of runtime for $r > w_{i,j}$ to get a little bit of faster runtime for $r < w_{i,j}$.

Rishad Rahman (Deepak's Recitation) April 3, 2015

6.046 Problem 6-2

Collaborators: *None*

(a) Suppose T_1 and T_2 are distinct MSTs. Consider the minimum edge weight that is contained in one of either T_1 or T_2 but not both, WLOG let it be contained in T_1 and denote it by e_1. Consider $T_2 \cup \{e_1\}$ which contains a cycle. One edge of the cycle e_2, must not be contained in T_1. We have $w(e_1) < w(e_2)$ and $T_2 \cup \{e_1\} \backslash \{e_2\}$ is spanning tree with weight $w(T_2) + w(e_1) - w(e_2) < w(T_2) = w(T_1)$ which contradicts the fact that T_1 and T_2 are distinct MSTs.

(b) **Algorithm**: Let S be the initial set of components determined by $A = \emptyset$, which is basically the set of vertices. Each vertex also has a pointer to which component it belongs to. Continue the following until $|S| = 1$:

$E = \emptyset$
For $C \in S$.

- Iterate through $E[v]$ for $v \in V[C]$ to find the lightest weighted edge, e_C coming out of C which is possible since we know the component tree of each vertex
- Add e_C to E

For e_C in E add e_C to A and update S along with the component tree pointers of each vertex.

Correctness: Note that at the end $|A| = V - 1$ since every time an edge is added to A, the number of component trees decreases by 1 as we are combining two, and since we started with V components this will happen $V - 1$ times in order for $|S| = 1$. We claim that at each step the edges of each component tree, C, is an MST of the vertices of C and that when we combine two component trees C and C' with edge e_C then the new component tree is an MST of the vertices of the combined trees. If we show this then A forming an MST of the original graph follows by induction. We have initially every component is an MST since each consists of a single vertex. Let's look at C and suppose e_C is the minimum edge weight going out of C into C'. Take a cut which isolates C from the rest of the graph. Then e_C is contained in an MST of the vertices of $C \cup C'$ but $C \cup C' \cup e_C$ is a spanning tree of those vertices but since we are dealing with unique edge weights, this must be an MST of those vertices as well. Induction complete.

Complexity: We iterate through all the edges for each vertex and we do this V times hence we have a runtime of $\Theta(V^2 + EV)$.

(c) Take $V = \{1, 2, 3, 4\}$, $E = \{(1, 2), (2, 3), (3, 4)\}$ and $V_1 = \{1, 3\}$, $V_2 = \{2, 4\}$. The weights could be anything, it will be impossible for the algorithm to find the MST since all three edges pass the cut.

(d) **Algorithm**: Run DFS to find cycle, remove lowest weighted edge. Continue until no cycles remaining.

Correctness: Suppose $1, 2, ...n$ form a cycle and $(n, 1)$ is the highest weighted edge of the cycle. Suppose an MST contains $(n, 1)$. Removing this edge breaks the MST into two trees, T_1 and T_2 whose union must contain all the vertices of the cycle. Note there must exist another edge of the cycle which connects T_1 and T_2 by the definition of a cycle and since this edge has a lower weight than the heaviest weighted cycle edge we get a lower weighted spanning tree, contradiction hence we can remove the heaviest weighted cycle edge safely.

Complexity: We run DFS $E - V + 1$ times so the runtime is $\Theta(E^2 - V^2 + E + V)$.