Intro to Algorithms, COMP-160, Homework #10

Benjamin Tanen, 04/15/2016

1. **Let G be a DAG. You must travel from one vertex s to another vertex t in G. To use some edges, you have to pay. For other edges, you get a reward. The amounts vary by edge. Show how to get from s to t, maximizing your profit (or minimizing your loss). What is the time complexity?**

   We are attempting to find the optimal way to traverse from s to t in our DAG G. For this, we can define $p(v_i, v_j)$ to return the maximized profit for traveling from v_i to v_j. Alternatively, we can say $p(v_i, v_j) = \max(\sum_{e \in E} w(e))$.

   We can start this problem by topologically sorting our V vertices such that v_i will only have edges to $v_{i+1}, v_{i+2} \ldots, v_{|V|}$. We also want to make sure that the vertex s is our v_1, but this is trivial. This sorting takes $O(V + E)$ time (using DFS).

   Next, we will also want to make a $2 \times |V|$ array. In each column i, we can store the optimized profit to traverse from v_1 to v_i (in the first row of column i) and the previous vertex in our optimal path before v_i (in the second row of column i). For example, if there is an edge from v_1 to v_2 (thus the optimal path to v_2), we would store $p(v_1, v_2) = w(v_1, v_2)$ and the index of $v_1 (= 1)$ in column 2. Before we store anything in this array though, we initialize the values in the first row to $-\infty$.

   From here we traverse through our sorted nodes in order. Say we are considering the node v_i in our sorted list. Since we are considering v_i, we know that we have already looked at every edge that points to v_i since our list is topologically sorted. Thus, the first value stored in column i is the maximized profit for traversing to v_i, $p(v_1, v_i)$.

   Now, for our current node v_i, we consider every adjacent node v_j we can traverse to. If $p(v_1, v_i) + w(v_i, v_j)$ is greater than the first value currently stored in column j, we replace this first value with $p(v_1, v_i) + w(v_i, v_j)$ and store the index of v_I as the second value in column j. If it is not, we just continue checking our other edges.

   If we continue this process for every node in our topological order, we will eventually fully fill out our table to hold the optimal profit for traveling from s to any other vertex. We did this by enumerating over every edge so the total work to build the table is $O(E)$.

   Thus, from here, we can simply query the column associated with the vertex t to get $p(s, t)$. We can also get the index of the previous node in the path from s to t. If we then query the previous index at this node, we can "backwards traverse" from t to s, telling us the optimal path from s to t. Backwards traversing using these indices takes $O(V)$ in the worst case.

   Thus, given all of this work, it takes $O(V + E) + O(E) + O(V) = O(V + E)$ to find the optimal path from s to t.