15-451 Algorithms, Fall 2009

Homework # 4 **Due: Tue-Wed, October 20-21, 2009**

Ground rules:

- This is an oral presentation assignment. You should work in groups of three. At some point before **Saturday, October 17 at 11:59pm** your group should sign up for a 1-hour time slot on the signup sheet on the course web page.
- Each person in the group must be able to present every problem. The TA/Professor will select who presents which problem. The other group members may assist the presenter.
- You are not required to hand anything in at your presentation, but you may if you choose.

Problems:

1. **Boruvka's MST Algorithm.** Boruvka's MST algorithm (from 1926) is a bit like a distributed version of Kruskal. We begin by having each vertex mark the shortest edge incident to it. (For instance, if the graph were a 4-cycle with edges of lengths 1, 3, 2, and 4 around the cycle, then two vertices would mark the "1" edge and the other two vertices will mark the "2" edge.) For the sake of simplicity, assume that all edge lengths are distinct so we don't have to worry about how to resolve ties. This creates a forest F of marked edges. *(Convince yourself why there won't be any cycles!)* In the next step, each tree in F marks the shortest edge incident to it (the shortest edge having one endpoint in the tree and one endpoint not in the tree), creating a new forest F'. This process repeats until we have only one tree.

   (a) Show correctness of this algorithm by arguing that the set of edges in the current forest is always contained in the MST.

   (b) Show how you can run each iteration of the algorithm in $O(m)$ time with just a couple runs of Depth-First-Search and no fancy data structures (heaps, union-find). Remember, this algorithm was from 1926!

   (c) Prove an upper bound of $O(m \log n)$ on the running time of this algorithm.

2. **Road trip!** Velma and the gang are going on a road trip to San Francisco immediately after their midterms. To plan the trip, they have laid out a map of the U.S., and marked all the places they think might be interesting to visit along the way. However, the requirements are:

   (a) Each stop on the trip must be closer to SF than the previous stop.

   (b) The total length of the trip can be no longer than D.

   Velma wants to visit the most places subject to these conditions. As a first step, she creates a DAG with n nodes (one for each location of interest) and an edge from i to j if there is a road from i to j and j is closer to SF than i. Let d_{ij} be the length of edge (i, j) in this graph.

Help out Velma by giving an $O(mn)$-time algorithm to solve her problem. Specifically, given a directed acyclic graph (DAG) G with lengths on the edges, a start node s, a destination node t, and a distance bound D, your algorithm should find the path in G from s to t that visits the most intermediate nodes, subject to having total length $\leq D$.

(Note that in general graphs, this problem is NP-complete: in particular, a solution to this problem would allow one to solve the *traveling salesman problem*. However, the case that G is a DAG is much easier.)

3. **Knapsack revisited.** Recall from class that in the knapsack problem we have n items, where each item i has a value v_i and a size s_i, and we also have a knapsack of size S. The goal is to find the maximum total value of items that can be placed into the knapsack. In particular, out of all sets of items whose total size is at most S, we want the set of highest total value. All sizes and values are assumed to be integers. In class, we gave a dynamic programming algorithm to solve this problem whose running time was $O(nS)$.

   One issue that comes up is that if the sizes are very large numbers, then $O(nS)$ may not be so good. In this problem, we want you to come up with an alternative algorithm whose running time is $O(nV)$, where V is the value of the optimal solution. So, this would be a better algorithm if the sizes are large but the values are not too big.

   Note: your algorithm should work even if the value V of the optimal solution is not known in advance. Actually, this is not such a big deal — you may want to first design your algorithm as if V were known and then find a way to get rid of that assumption.

   Hint: it might help to look at the algorithm from class and think about what the subproblems were. Then think about what subproblems you want to use instead for the new goal.