15-451 Algorithms, Fall 2003

Homework # 4 **due: Tue-Wed, October 21-22, 2003**

Groundrules:

- This is an oral presentation assignment. As stated in the course information handout, you should work in groups of three. At some point before **Sunday October 19 at midnight** you should sign up for a 1-hour time slot on the signup sheet on the course web page.
- You are not required to hand anything in at your presentation, but you may if you choose. If you do hand something in, it will be taken into consideration (in a non-negative way) in the grading.

Problems:

1. Dynamic Programming.

   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 have a knapsack of size S.[1] The goal is to find the maximum total value of items that can be placed into the knapsack. I.e., out of all sets of items whose total size is at most S, we want set of highest total value. In class, we gave a dynamic programming algorithm to solve this problem whose running time was $O(nS)$.

   One issue brought up in class was that if the sizes are large, 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 great.

   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 for the new goal.

2. 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.

[1] In class we used "times" instead of "sizes", but I don't want to get confused with the running time of the algorithm....

(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 $O(|E|)$ time with just a couple runs of Depth-First-Search and no fancy data structures. (Remember, this algorithm was from 1926!)

(c) Prove an upper bound of $O(|E| \log |V|)$ on the running time of this algorithm.

3. Graph Searching

Let G be a directed graph represented using an adjacency list. So, each node $G[i]$ has a list of all nodes reachable in 1 step from i (all out-neighbors of i). Suppose each node of G also has a value: e.g., node 1 might have value \$100, node 2 might have value \$50, etc.

- Give an $O(|E| + |V| \log |V|)$ time algorithm that computes, for every node, the highest value reachable from that node in 0 or more steps. For instance, if it is possible to get to any node from any other node (G is "strongly-connected"), then for every node this will be the maximum value in the entire graph.

Hint: one worthwhile preprocessing step is to sort nodes by value.