TTIC 31010 / CMSC 37000 Algorithms, Winter Quarter 2019

Homework # 2 **Presentations: Jan 30 – Feb 1, 2019**

This is an oral-presentation assignment. You should work in groups of three. At some point before midnight on Sunday January 27, your group should sign up for a 1-hour time slot on the signup sheet by following the instructions on the course home page.

The Professor/TA/Grader reserves the right to select which group member presents which problem. You are allowed to bring in notes to help you (e.g., a writeup of your solutions).

You are not required to hand anything in at your presentation, but you may if you choose.

Problems:

1. **Widest paths**

   Big-Trucks-R-Us wants to find the *widest* path from its warehouse to every other location in the city. Specifically, assume you have a graph G, a source node s in G, and that every edge e in G has a non-negative "width" $w(e)$. Define the width of a path to be the *minimum* width of any edge on that path. For example if a path has three edges of lengths 3, 2, and 4, respectively, then this path has width 2. The widest path from s to some other node t in G is the path from s to t whose width is largest. For example, if the graph were a 4-cycle with edges of width 1,3,2,4 around the cycle, then the widest path between any two opposite corners would have width 2.

   Give an efficient greedy algorithm for finding the *widest* path from s to every other node in the graph, and argue why your algorithm is correct. Hint: modify Dijkstra's algorithm.

2. **Boruvka's MST Algorithm.** Boruvka's MST algorithm (from 1926) is a bit like a distributed version of Kruskal's algorithm. 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, and so that the MST is unique. 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.

3. **Spring Break.**

   You and your friends decide to go on a road trip to New Orleans for Spring Break. To plan the trip, you have laid out a map of the U.S., and marked all the places you think might be interesting to visit along the way. However, the requirements are:

   (a) Each stop on the trip must be strictly closer to New Orleans than the previous stop.

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

   You and your friends want to visit the most interesting places subject to these conditions. As a first step, you create a directed graph 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 New Orleans than i. Let d_{ij} be the length of edge (i, j) in this graph.

   Give an $O(mn)$-time algorithm to solve your optimization problem. Specifically, given a directed acyclic graph (DAG) G with n nodes, m edges and with lengths on the edges, and given 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.)