Design and Analysis of Algorithms **out:** February 3, 2015

Problem Set 2

Lecturer: Daniel A. Spielman **due: 2:30** PM, February 12, 2015

Instructions. Read Carefully

1. Make sure that your yale **netid** appears on **every** page of your problem set, and that it is written clearly.
2. The problem sets are due at the beginning of class. Solution sets will be provided in class. For this reason, late problem sets are not accepted.
3. You will **hand in each problem separately**, so make sure that they are not stapled together and that no problem appears on the back of another. There will be 4 boxes in the front of the class, one for each problem. Put each problem in the correct box.
4. Alternatively, you may submit the problem set electronically via the Classes server. If you do so, submit it as a **pdf** and make sure that your **netid** appears on **every** page. **Each problem must be in its own document.**
5. Cite any resources that you use, other than the textbook, TA and instructor.
6. You **must not** search the web for solutions to similar problems given out in other classes.

Collaboration Policy

You must write the solutions to the problems independently and in your own words.

However, you are allowed to discuss the problem sets in small groups of no more than 4 students. To ensure that you understand the solutions you submit, you are forbidden from taking written notes during your discussions. You must list at the top of the problem set everyone (other than course staff) with whom you have discussed the problem set. Failure to list people with whom you have discussed a problem set is considered a violation of academic honesty.

Problem 1: MSTs of Changing Graphs, part 1

We must often perform computations on inputs that change over time. For example, rather than just computing the minimum spanning tree of a graph, we may need to keep track of the minimum spanning tree as the graph changes.

We usually solve such problems by computing more than is necessary. That is, we compute additional data structures that allow us to quickly modify the minimum spanning tree as the graph changes. But, this is a topic for a graduate course.

For this problem set, we will just consider the problem of how to change a minimum spanning tree when the weight of an edge in the input graph changes. In particular, you should assume that you are given a weighted graph $G = (V, E, w)$ and a minimum spanning tree T of G. You may assume that the weights of all edges in G are distinct and that G is connected. You are then told that the weight of some edge in G has changed. Call the edge be (a, b), and the new weight be x. In this part, we will assume that the new weight is larger than the old one:

$$x > w(a, b).$$

You may also assume that x is distinct from the weight of every other edge in G.

a. Design an algorithm that computes the minimum spanning tree of the new graph in time $O(m)$, where m is the number of edges in the graph. (5 points)

   Note: Your algorithm gets as input G, T, (a, b) and x. This problem would be very difficult if your algorithm was not given the minimum spanning tree of G.

b. Prove that your algorithm is correct. If it is not completely obvious, prove that your algorithm runs in the stated time. (5 points)

Note: The algorithm that we learned in class for computing minimum spanning trees runs in time $O(m \log m)$. The algorithm I have asked you to design must be faster.

Problem 2: MSTs of Changing Graphs, part 2

This is the same as the first problem, but we now handle the case in which $x < w(a, b)$. That is, the change in the graph is a decrease in the weight of an edge.

a. Design an algorithm that computes the minimum spanning tree of the new graph in time $O(m)$, where m is the number of edges in the graph. (5 points)

b. Prove that your algorithm is correct. If it is not completely obvious, prove that your algorithm runs in the stated time. (5 points)

Problem 3: Negative Length Edges

When we learned Dijkstra's algorithm for computing shortest paths in graphs, we assumed that all edges lengths are positive. However, there are applications in which one would like to solve this problem in which some edge lengths are negative. Dijkstra's algorithm will not handle this general case.

In this problem, you will design an algorithm that handles a limited type of negative length edges: all negative length edges leave s, and no edges enter s. That is, you are given as input a directed graph $G = (V, E, l)$, where $l(a, b)$ indicates the length of edge (a, b), and a vertex $s \in V$. The graph does not have any edges from any vertex to s. All of the edge lengths are positive, except that the lengths of edges leaving s are allowed to be negative.

a. Design an algorithm that computes the distance of every vertex from s in a graph of this restricted form. Your algorithm should run in time $O((n + m) \log n)$, where $n = |V|$ and $m = |E|$. Explain why your algorithm runs in the stated amount of time. (6 points)

b. Prove that your algorithm is correct. (4 points)

Problem 4: Closest Pair in a Graph

In this problem, you are given a graph along with a subset of its vertices. Your goal is to find the closest pair of vertices in that subset. More formally, you are given an undirected graph $G = (V, E, l)$, where l assigns a positive length to every edge. You are also given $S \subset V$. For vertices $a, b \in V$, let $d(a, b)$ be the length of the shortest path between them. Your goal is to find a pair $a, b \in S$ for which $d(a, b)$ is as small as possible.

a. Design an algorithm that solves this problem in time $O((n + m) \log n)$, where $n = |V|$ and $m = |E|$. Explain why your algorithm runs in the stated amount of time. (6 points)

b. Prove that your algorithm is correct. (4 points)