15-451 Algorithms, Spring 2009

Homework # 4 **due: Tue-Thu 24-26, March , 2009**

Ground rules:

- This is an oral presentation assignment. You should work in groups of three. Please sign up for a 1-hour time slot for your group *in recitation* on Wednesday, March 18. From Thursday morning until Sunday, March 22 at 11:59pm, there will be a signup sheet posted on the door of Doherty Hall, 4301a. You can use this to sign up, although the later you do this, the less likely you are to get a good time slot.
- 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.

1. **Modifying Boruvka's MST Algorithm (Maxim Makatchev)**

   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. Assume also that graph is connected, so $m = \Omega(n)$, where m is number of edges and n is number of vertices. 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.

   We will consider a slight modification of Boruvka's algorithm, where after we find the shortest edges incident to each of the vertices of graph G, we add them to the current forest F and then we *contract* the graph G by these edges. This procedure is repeated until there is no edges left.

   Definition: Graph G' is a contraction of G by an edge $e = (u, v)$ if the set of vertices of G' excludes both u and v, merging them into a new vertex x, and the set of edges of G' is formed from the set of edges of G by renaming all u's and v's in the original edges into x. Note that this operation may produce duplicate edges $(s, t), (s, t)$ and self-loops (u, u). So contraction is concluded with a cleaning phase, which removes self-loops and replaces duplicate edges with the one of them that has shortest length.

   (a) Prove the correctness of the contraction-based Boruvka's algorithm. (Hint: Show that the set of edges in the current forest is always contained in the MST).

   (b) Write a pseudocode for the single iteration of the contraction-based Boruvka's algorithm. This single iteration should run in $O(m)$ time and have no fancy data structures. Remember, that each iteration includes finding the shortest edges, and contraction, which itself includes a "cleaning" phase that removes duplicate edges and self-loops.

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

Now that you know the contraction-based Boruvka algorithm, you will show that it gives particular good (linear) running time on sparse graphs that remain sparse after contractions. One of such classes is a class of planar graphs—the graphs that can be laid out on a plane such that no two edges intersect. In particular, it is known that (a) for any planar graph with at least 2 edges $m \leq 3n - 6$ and (b) a contraction of a planar graph is itself a planar graph.

(d) Show that for planar graphs contraction-based Boruvka's algorithm runs in $O(n)$ time.

2. **Problem 2: All-Pairs Shortest Paths (David Abraham)**

(a) Suppose a government agency wants to monitor all traffic on the internet. One way to do this would be to set up a computer and require that all traffic be routed through this computer. In this problem, we are going to write an algorithm to find the shortest path between all pairs of computers, subject to the requirement that all paths must go through a given computer.

Formally, suppose we are given an undirected graph $G = (V, E)$ in which each edge e has a length $l(e) >= 0$, and a special vertex $x \in V$. Write the fastest algorithm you can to find the length of the shortest path between all pairs of vertices, where the shortest path must go through x. It is OK if the shortest path from vertex u to v first goes to v, then goes to x and finally goes back to v.

(b) Now suppose that *three* government agencies want to monitor the internet traffic, all independently of one another. This means that our shortest paths now have to go through three different computers, x, y and z, though the order does not matter.

Write the fastest algorithm you can to find the length of the shortest path between all pairs of vertices, where the shortest path must go through x, y and z in no particular order.

(c) What is the running time of your algorithm if all paths must go through $\lg n$ different vertices? Is this polynomial time?

(d) Suppose that the three government agencies decide to cooperate and share data. Now our shortest paths only have to go through *one* of x, y and z.

Write the fastest algorithm you can to find the length of the shortest path between all pairs of vertices, where the shortest path must go through x, y or z.

For all of the questions above, please have your algorithms ready to present in pseudocode

3. **Problem 3: Min-Bisection (Pranjal Awasthi)**

Let $G = (V, E)$ be an undirected graph with the vertex set V and edge set E. A bisection of G is a partition of the vertex set into two subsets V_1 and V_2 of equal size, i.e.,

$$|V_1 - V_2| = \begin{cases} 0 & \text{if } |V| \text{ is even} \\ 1 & \text{if } |V| \text{ is odd} \end{cases}$$

Note that every vertex must belong to one and only one of the subsets. The min-bisection problem is to find a bisection of G which minimizes the number of edges connecting the two subsets. An edge e connects V_1 and V_2, if one of its end points lies in V_1 and the other lies in V_2. In general there is no known polynomial time algorithm for this problem. However, if the graph is a tree, the problem becomes easier.

Design a polynomial time algorithm to solve min-bisection when the input graph G is a tree[1]. You should be able to prove the correctness of your algorithm and analyze its running time.

Hint: Think dynamic programming.

[1]The tree need not be a binary tree