Algorithms
Computer Science 140 & Mathematics 168
Spring 2007
Homework 8b
Due Tuesday, March 27

1. **[10 Points] Proof of Dijkstra's Algorithm.** Write the proof of correctness of Dijkstra's Algorithm in your own words - except now generalize it slightly to permit edge weights to be non-negative but possibly 0. Please use the proof technique that we developed in class. Note that the proof that we did in class is not the same as the proof in the book. (The book provides intuitive arguments but not a rigorous proof.) Your proof should be clear, precise, and complete.

   In your proof, please use the following notation from class: Let s denote the start vertex (we are trying to find the shortest path lengths from s to every other vertex), let $d(u)$ denote the length of the shortest path from s to u, and let $\ell(u)$ denote the label that the algorithm associates with vertex u at any given time. Note that $d(u)$ is a graph theoretic notion whereas $\ell(u)$ is an artifact of the algorithm.

2. **[20 Points] Hurts' On-board Navigation System Re-revisited.** Hurts Car Rental has designed a new generation of alternative fuel vehicles. The new vehicles use a special fuel comprising a finely minced mixture of the "Algorithms" textbook, chocolate fudge Pop Tarts, Spam, Spam, Spam, and Mountain Dew. Due to this rather unusual fuel requirement, there are only certain cities in the country where the vehicles can be refueled. Thus, to get from the start city to the destination city, the driver must plan a route that ensures that the car can be refueled along the way. (Note that this problem differs in several important ways from the first Hurts problem on Assignment 3b. First, we no longer use fuel packs - we can now fill up with fuel like a normal car. Second, the graph is no longer a straight line - it's a general graph. Third, we only care about minimizing the distance travelled, not about the cost of fuel.)

   The on-board computer on such a vehicle contains a weighted directed graph in which the vertices represent cities, the directed edges represent one-way roads, and the weights on the edges represent the (strictly positive) lengths of the roads. The graph, of course, can have cycles!

The computer knows which select cities have filling stations. To use the navigation system, the user will enter the starting city, the destination city, and the range of the vehicle on a full tank. (You may assume that the starting city and the destination cities, being Hurts rental cities, both have filling stations - and that the car starts with a full tank of fuel.) The computer will respond with the *shortest* route from the starting city to the destination city that ensures that the vehicle doesn't run out of fuel or determines that no such route exists.

You should assume that there is some constant C such that every vertex has in-degree and out-degree (number of entering edges and number of exiting edges, respectively) at most C.

(a) **Describe** and **analyze the running time** of an efficient algorithm for determining such a path. There are several ways to do this problem, but if you break this problem up into a couple intermediate steps you will probably find that the important problem of proving correctness (in the next part of the problem) is relatively easy.

(b) *Prove* that your algorithm finds an optimal solution. Be careful and precise. A very short but rigorous proof is possible if your algorithm was designed carefully.

(Note: Your algorithm may be built up of several algorithms which we have seen in class. You may apply any algorithms that we have seen in class and need not re-prove their correctness. However, if you use an algorithm that is at all different from one that we've examined in class, you must indeed prove its correctness.)

3. **[10 Points] Faster Implementations of Kruskal and Prim.** Consider the minimum spanning tree problem again but now assume that all edge weights are integers between 1 and W where W is some integer constant.

(a) How fast can you make Kruskal's Algorithm run? Explain your implementation and its running time.

(b) How fast can you make Prim's Algorithm run? Explain your implementation and its running time.

4. **[10 Points] Forrester's MST Algorithm!** Professor I. M. A. Forrester of the Pasadena Institute of Technology has developed a new algorithm for the minimum spanning tree (MST) problem. It is based on the divide-and-conquer paradigm and it goes like this: Given a graph $G = (V, E)$, partition the set of vertices V into two sets V_1 and V_2 whose sizes are equal (or differ by at most 1). Let E_1 be the edges incident only on vertices in V_1 and let E_2 be the edges incident only on vertices in V_2. Recursively solve the MST problem on the two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge that crosses the cut (V_1, V_2) and use that edge to unite the two spanning trees found by the recursive calls.

   Professor Forrester did not prove the correctness of his algorithm. Your job is to either carefully prove the correctness or give a counterexample that shows that the algorithm is not correct.

5. **[15 Points] Barůvka's Algorithm!** In class we mentioned that the first algorithm for computing minimum spanning trees was published by the Czech mathematician Otakar Barůvka in 1926 and was used for laying out electrical networks in Czechoslovakia (actually, in a part of Czechoslovakia known as Bohemia - but that detail will probably not make a large impact in your solution to this problem). The algorithm works like this:

```
A = {};  \\ Comment:  A is a subset of a minimum spanning tree
Consider the n  vertices in the graph as n connected components;
while A contains fewer than n-1 edges
{
   for each connected component C
   {
      Find the least weight edge (u,v) with one vertex in C and
        one vertex not in C;
      Add edge (u,v) to A;
   }
   Compute the new connected components;
}
return A \\ Comment:  This is intended to be a MST!
```

   Notice that if C_1 and C_2 are two different connected components before

we begin the for loop, then inside the for loop the algorithm will choose the least weight edge coming out of component C_1 and also the least weight edge coming out of C_2. The edge chosen by C_1 might join C_1 and C_2 into a new connected component, but this new connected component will not be discovered until the for loop has ended! In other words, both C_1 and C_2 will each get an opportunity to choose the least weight edges coming out of their components!

(a) Give a counter-example that shows that Borůvka's Algorithm doesn't work!! (You might find it useful to use the fact that some edges in the graph may have the same weight.) Show your counter-example graph and explain carefully why Borůvka's Algorithm would not compute a minimum spanning tree in this case.

(b) Now assume that no two edges in the graph have the same weight. By the OPTIONAL BONUS PROBLEM below, such a graph has exactly one minimum spanning tree (you may just use this fact here, although you are encouraged to prove it in the bonus problem!). Under this assumption, prove that Borůvka's Algorithm is correct.

(c) Why doesn't your proof from part (b) work if some edges in the graph have the same weights?

(d) How could Borůvka's Algorithm be modified slightly to work in the most general case that edge weights are not necessarily distinct?

6. **[Optional 15 Point Bonus Problem]** Let G be a connected undirected graph in which each edge has a distinct edge weight (that is, no two edges have the same weight). Prove that there is a *unique* minimum spanning tree in the graph. (This problem does not require that you know anymore graph theory than we have already used in class.)