Algorithms
Computer Science 140 & Mathematics 168
Instructor: B. Thom
Spring 2004
Homework 9a
Due on 3/25/2004 (in class)

1. [**20 Points**] **Finding a Negative Cost Cycle.** In class we showed how the Bellman-Ford Algorithm (BF) can be used to detect if a negative cost cycle (NCC) exists in a directed graph. In this problem, you will play with this idea some more.

   (a) Develop an algorithm that will determine how *a* negative-cost cycle can be found (that is, the vertices and edges in one such cycle can be printed out) if one exists. Your algorithm need only find *one* such negative cost cycle. This algorithm should rely on the BF NCC detector we discussed in class. Describe the algorithm and briefly present its runtime. (Your algorithm need not be the fastest possible algorithm for this problem; anything polynomial in V and E is fine).

   (b) Will your algorithm always find the most negatative NCC? If so, explain why. If not, give a counter-example.

2. [**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. The algorithm works like this:

```
E_f = {};  /* Comment: E_f is a subset of a MST */
Treat the V vertices in the graph as V connected components;
while E_f contains fewer than V-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 E_f;
   }
   Compute the new connected components;
}
return E_f /* Comment:  E_f is now 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! 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) **[10 Points OPTIONAL BONUS PROBLEM #1! (due Friday by 6pm)]** 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). Show that there is a *unique* minimum spanning tree in the graph.

(e) **[10 Points OPTIONAL BONUS PROBLEM #2! (due Friday by 6pm)]** Describe an implementation (how the algorithm would be implemented using appropriate data structures) for Borůvka's Algorithm and derive its running time. Try to make your implementation as fast as you can.