CSC 520 Programming Project 3
Minimal Spanning Trees

Due: Tuesday November 30, 2010

Minimum spanning trees are important in networking applications, since they provide a way of connecting all the nodes in a graph together in a minimal structure. In modern wireless and mobile ad-hoc networking systems, devices are constantly being added and removed from the network, and transmission delays along links are variable. Thus it is important to be able to maintain an MST as the underlying network characteristics change.

The goal of this project is to input an undirected graph with weighted edges, compute an initial MST for the graph. Then as changes occur in the graph structure, the MST is to be updated without rebuilding from scratch. These changes include the insertion and deletion of vertices, insertion and deletion of edges, and changing the weights of edges.

Input :
to program is an integer *n*, followed by an integer e and a list of *e* edges (*u, v, c*). *n* is the number of nodes in the graph, *e* is the number of edges and each of the *e* edges is specified by its endpoints *u* and *v* (integer between 1 and *n*) and its cost *c*. You may assume that the nodes are labeled 1 through n, and all edges have positive cost. You may also assume that the graph is connected.

Requirements:

1. Use the adjacency list representation to implement the graph.
2. Build the graph's MST using Kruskal's algorithm.
3. Use a heap to store the edges based on their costs.
4. You may also find the union and find operations useful.
5. The minimal spanning tree (MST) found must be stored appropriately.
6. You are allowed to use simple, linear data structures from the Java or C++ libraries (e.g. strings, linked-lists, vectors, stacks, queues). You are not allowed to use anything more sophisticated (no trees, dictionaries, priority queues, etc.).

Implement the following functions:

1. **PrintTree *v***
   Print the contents of the current MST(s), given a particular vertex *v* as the root. Print the MST according to a preorder traversal of the tree.

2. **Path *u v***
   Given two vertices ***u v***, compute and print the path between them (from ***u*** to ***v***) using the edges in the MST.

3. **AddEdge *u, v, c***
   Suppose an edge (u, v) with cost c is added to the graph, find the minimal spanning tree of this new graph efficiently by updating the existing minimal spanning tree. The

update operations must be performed incrementally, by making the fewest possible changes to the MST. In particular, it is illegal for AddEdge operations to completely rebuild the MST from scratch.

- Test your program thoroughly using different graphs.
- Submit a copy of your program including the source code and the class file (or executable file) to Blackboard.   I'll run your program with my test data.
- Hand in a hard copy of your **well documented** program, the graphs you used to test the program, the corresponding minimal spanning trees and their costs, the edges added and the new updated minimal spanning trees.

Optional: Implement other functions to update a minimal spanning tree as changes occur in the graph structure without rebuilding from scratch. Other changes you should consider are decreasing the cost of an existing edge, deleting an edge, adding a new node and edges from the node, and deleting a node.

insert-vertex *x y*	Insert vertex u in the graph
decrease-cost *u v w*	Decrease the cost of edge (u,v) by w units
delete-vertex *u*	Delete vertex u from the graph
delete-edge *u v*	Delete edge (u,v) from the graph
increase-cost *u v w*	Increase the cost of edge (u,v) by w units