ECE745 – Assignment 1

Issued on May 11th and due before May 21st at 11:59 PM

1) Implement Prim's algorithm for identifying the minimum spanning tree in a weighted undirected graph. The graph is read from an input text using the following simple format (note, what comes after "//" are just comments and do not need to be included):

```
k           // the number of vertices labeled 0 ... k-1
2 3 6       // edge between vertices 2 and 3 and its weight is 6
...
0 k-1 10  // edge between vertices 0 and k-1 and its weight is 10
```

It is your responsibility to ensure that the input graph is connected. If not, then an error message should be printed. Note, the order of edges in the "problem1.input" is arbitrary.

The input arguments should be:

```
./spanning_tree <filename> <mode>
```

<filename> - name of the file with the input graph
<mode> - 0 linked list / 1 binary heap

The output should print the size of the minimum spanning tree and all its edges.

2) Implement Dijkstra's algorithm for single-source shortest path to a single-destination for a weighted undirected graph with non-negative edges. The format of the input graph should be the same as for problem 1.

The input arguments should be:

```
./shortest_path <filename> <source node> <destination node> <mode>
```

<filename> - name of the file with the input graph (edge weights must be positive)
<source node> - source node for Dijkstra’s algorithm
<destination node> - destination node for Dijkstra’s algorithm
<mode> - 0 linked list / 1 binary heap

The output should print the size of the shortest path and all its edges in the correct order from the source to the destination nodes.

Notes:

For both problems the data structures should be DYNAMIC, i.e., you MUST use pointers and dynamic memory management. Both problems must use the linked list data structure as the reference implementation. The implementations for improved runtime should use binary heaps. Nevertheless, although not mandatory, if you have “spare” time you are encouraged to use more advanced data structures, like binomial or Fibonacci heaps.

Commit and push your sources to your GitHub repo before the deadline. In the doc subfolder you should have a file called Report (in simple text format) that should explain the basic intuition of your approach for each problem. It should also analyze the time and memory complexity of your algorithms (in big O notation). When printed, the Report file should not exceed one page. Note, your program will be tested with large graphs and hence the runtime/memory will be compared against your analytical results.

The Makefile should guarantee that when running "make" under Linux with gcc/g++, the sources will compile. When the executable is run without any arguments, print a “help message” that explains each command line argument.

This assignment is worth 10% of your final grade.