CCPS616 Lab 3 – Minimum Spanning Trees

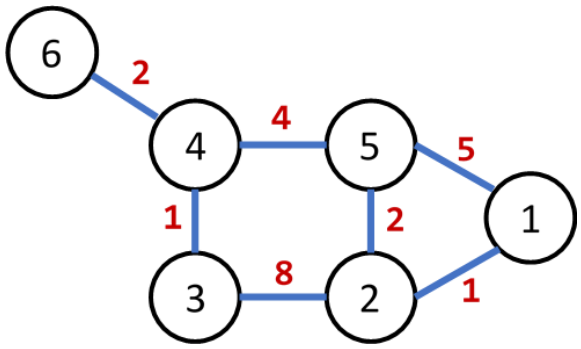
Preamble

In this lab you will write a C++ program that finds a minimum spanning tree of a graph. Your implementation must be from the ground up without linking any external libraries. Your lab should be completed in a single cpp file called **mst.cpp**, and should compile at the command line very simply as follows:

```
g++ -o lab3 mst.cpp lab3.cpp
```

Lab Description

In this lab you will write a program that accepts a graph as input and produces a *minimum spanning tree* as output. The input graph will be generated randomly in the form of an adjacency matrix. An example of the input and associated weighted graph can be seen below.




```
0 1 0 0 5 0
1 0 8 0 2 0
0 8 0 1 0 0
0 0 1 0 4 2
5 2 0 4 0 0
0 0 0 2 0 0
```

In the above example, the values in the matrix indicate the weight of the edge between two vertices. If the value is zero, there is no edge. You may assume the input graph is simple, finite, and connected.

You may choose either Prim's or Kruskal's, whichever you prefer. In either case, the input format is the same. Note that you are not required to use an adjacency matrix internally. This is just how the graph will be produced in the random generator. You may want to build an adjacency list to use instead, but it's up to you.

Your program will output an adjacency matrix in the exact same format as the input. The output will contain only those edges (and their weights) that are part of the minimum spanning tree.

You are not required to implement a priority queue or disjoint set data structure to optimize the asymptotic complexity of your implementation. That is, an $O(|V|^2)$ solution is just fine. Of course, implementing the appropriate data structures to push the complexity down to $O(|E|\ln(|V|))$ is great practice.

Testing

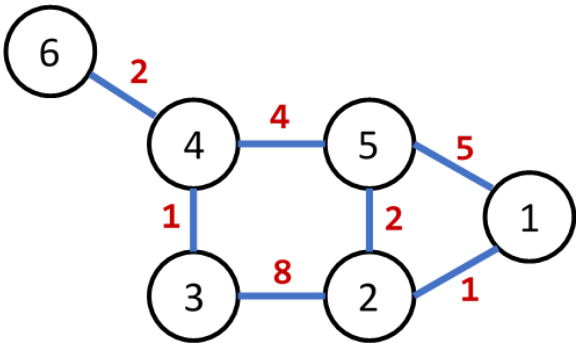
As with previous labs, a template main() function containing tests is provided for you (in lab3.cpp). The first thing you should do is look through this provided main() function and understand what it's doing. Your job is to fill in the minimumSpanningTree() function (in mst.cpp), then compile and run the program.

The main() function will test your code against some small, known examples, and then moves on to randomized testing on large graphs that are both sparse and dense.

In addition to the built-in simple examples, you are encouraged to test your program on several graphs of varying sizes. I don't expect you to produce a 1000x1000 adjacency matrix by hand, but you should test on graphs of similar order as those we looked at in class. In fact, some good test cases would be exactly those graphs we saw in class. This way, you can verify the correctness of your results before moving on to the large-scale randomized trials.

Results

As mentioned, your results should be produced in the form of an adjacency matrix. For the same graph seen in the description, the MST produced would look as follows:




```
0 1 0 0 0 0
1 0 0 0 2 0
0 0 0 1 0 0
0 0 1 0 4 2
0 2 0 4 0 0
0 0 0 2 0 0
```

Included in the provided code is a function called validateST(). It does not determine if your spanning tree is minimum. It simply ensures the number of edges is correct, that there are no cycles, and that the edges in your MST were present in the input graph.

Submission

Submit your source file (**mst.cpp**) on D2L.

Labs may be submitted individually, or in groups of up to **two**. If you submit as a group, be sure to include both names in the D2L submission as well as your source code.