EECS 2510 – Non-Linear Data Structures and Programming in C++

Lab 5 – Due Monday, May 2, 2016 at 12:00 NOON

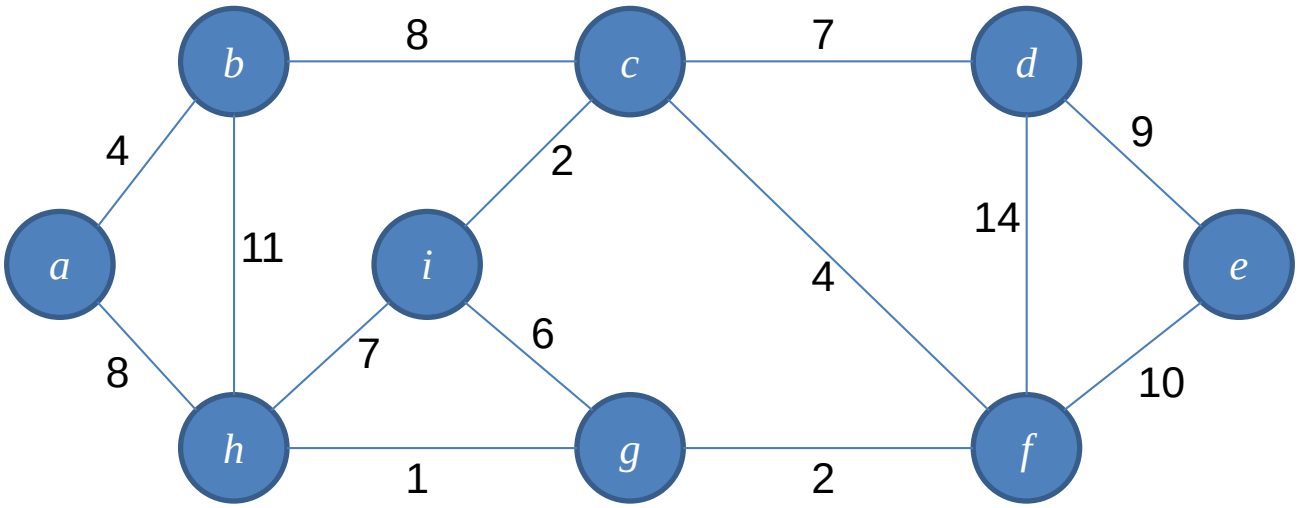
For this programming lab, you are to implement the two algorithms we covered for producing minimum spanning trees for a weighted, non-directed graph – Kruskal's algorithm, and Prim's algorithm (see Chapter 23).

Your input will be in the form of a text file. The first line in the file will be an integer *N*. Following the line will be *N* more lines, each with the name of a node. Node names will be one- or two-character strings, one per line. These will be a la Excel's column headers

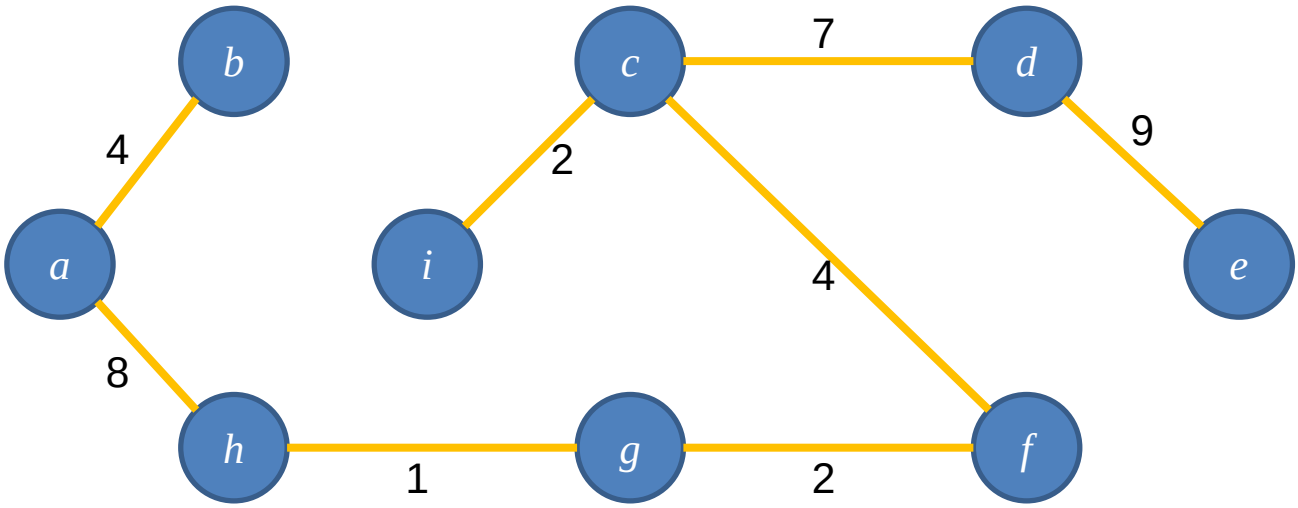
Following the node names will be an adjacency matrix consisting of *N* rows of *N* items per row. Each will be a positive edge weight (potentially, a floating point value, so count on using `double` as the data type for your adjacency matrix). The items on each row will be space-delimited, so the `<<` stream operator will work for you to do the file input.

Your program is to read the node names and the adjacency matrix and then compute a minimum-weight spanning tree for the graph. You will have two sets of output, one for Kruskal, and one for Prim. The output for each algorithm will be N lines: a double, followed by the *N* – 1 edges that make up the tree, one-per-line. The edges will be specified as node1<dash>node2<colon><space>weight, with the edges sorted alphabetically by node1, and then by node2.

Below is the graph for which we walked through both algorithms:



After running through Kruskal's algorithm, we had this tree, which has a weight of 37:



The input file for this graph would look like this:

```
9
a
b
c
d
e
f
g
h
i
0  4  0  0  0  0  0  8  0
4  0  8  0  0  0  0 11  0
0  8  0  7  0  4  0  0  2
0  0  7  0  9 14  0  0  0
0  0  0  9  0 10  0  0  0
0  0  4 14 10  0  2  0  0
0  0  0  0  0  2  0  1  6
8 11  0  0  0  0  1  0  7
0  0  2  0  0  0  6  7  0
```

The output for this graph would be:

```
37
a-b: 4
a-h: 8
c-d: 7
c-f: 4
c-i: 2
d-e: 9
f-g: 2
g-h: 1
```

I suggest you try your program on several graphs. Work them out on paper, and build a text file containing their representation. Use the test case(s) to prove to yourself that your program works properly.

IF YOU RUN INTO PROBLEMS, AND HAVE MADE A GOOD ATTEMPT AT SOLVING THEM, E-MAIL ME, RATHER THAN LOSING TIME LOST IN THE WILDERNESS. DON'T WAIT TO GET STARTED ON THIS. "I WAS BUSY WITH MY OTHER CLASSES" WILL NOT BE A VALID EXCUSE. THIS IS END-OF-SEMESTER CRUNCH TIME FOR ALL OF US. THERE WILL NOT BE ANY EXTENSIONS OF ANY KIND FOR THIS ASSIGNMENT.

To turn in your program, use 7-Zip (www.7-zip.org) to create a compressed archive of your entire Visual Studio workspace (don't just send me your source and/or .exe files). Submit your 7-Zip archive to Blackboard. If, for some reason, 7-Zip is not a practical alternative for you, I will accept .zip archives, but not .rar or any other format, unless you clear it with me ahead of time.

As always, your code is to be yours and yours alone. Do not use code from any other source (including each other and the Internet), even as a reference. Failure to write this code from scratch will be considered a breach of academic integrity, and will be dealt with most severely.

Employ good programming habits. I recommend the Allman style for braces. DOCUMENT YOUR CODE. Some of you have been very lax about comments this semester. Blocks of code (and functions / methods) should have a block comment introducing what you're about to do (and how you're about to do it), and the individual steps should have enough line comments to walk someone new to your code through your thought process. You don't have to write a textbook explaining the algorithm, but for someone even vaguely with the algorithm, they should be able to read your code and understand the steps.

You will need a priority queue for this program. I suggest you implement it with a heap (and all that entails – see Chapter 6).

Your set operations (for Kruskal) may be readily implemented as a linked list of linked lists (just a suggestion). If you want to work that way, feel free. If there's some other data structure you would like to use, that's up to you, too, but you are required to stick to C++'s built-in primitives, plus whatever you can build out of them (no template objects like vectors or collections to do the work FOR you).