**Homework #6B: Power Grid (due Wednesday, December 3, 2014, beginning of class)**

**Learning Objectives**

• Reading, understanding, and using a graph ADT

• Kruskal's algorithm and its implementation (heap, uptree)

• More practice with debugging and constructing good test cases

The power grid (the infrastructure and wires needed to provide electricity) in Tacoma has been destroyed by a huge fire. An emergency plan has been put into place, but it is expensive and so therefore cannot be a long-term solution. You are part of the reconstruction team. You are one of the software engineers who need to figure out a good way to connect the power grid.

You have decided that the best way to do this is to first list all the places that need power. Each one of these places will be represented by a vertex in a graph. Making a direct power connection between some pairs of places will be impossible (because connecting them would require digging up parts of the city that can't be disrupted or because it would require construction that is far too expensive). Of the remaining possible connections, you would have to estimate how much it would cost to connect the two places.

Fortunately, someone has already provided all of this information to you. Your task is to come up with a set of connections such that there is an electrical path from every vertex to every other vertex. The head of your software design team has decided that Kruskal's minimum spanning tree algorithm is the best way to solve this problem. You should implement the algorithm in a way that is asymptotically optimal for Kruskal's algorithm (otherwise, what's the point?). Since the graph is sparse, you should use a heap implementation of the algorithm.

You are provided with code that implements a simple graph ADT. You will need to write code that implements Kruskal's algorithm. You are not allowed to change the provided graph code. (This is typically how code works in a large software project. You are given base code and you need to write against the methods provided by that code.) You should allow the user to specify a graph file that has place and cost data (terminal input is fine – you do not need to write a GUI). As output, you should list the set of edges in the minimum spanning tree your algorithm generates. Also, you should output the total cost of the minimum spanning tree.

**Tips**

Familiarize yourself with the graph ADT code before doing anything. Notice that when the graph data is read in, it returns a hash table that maps labels to vertices. (Why is that necessary?) Then read and understand how Kruskal's algorithm works (that is, do homework 6A). Only then will you be ready to write code.

**Starter Code**

Here is the java code to start with: Vertex.java Edge.java SimpleGraph.java GraphInput.java InputLib.java KeyboardReader.java You should generate your own test cases and test them on your code.

**Software Engineering**

Understand the graph ADT code and Kruskal's algorithm before writing code. You will waste valuable time if you don't. (Thought questions: Is the graph ADT code written in a way you would have implemented it? Why or why not? Are there object-oriented principles you think are not being adhered to in the code?) As usual, comment your code and use good programming practices (good variable names and good indentation), so that it is easy to figure out what it is doing when/if you detect a bug.

**What to Turn In**

• Electronically turn in the code you created. You should not turn in the starter code, since you should not have changed it.

• Electronically turn in two test files, containing graph data with a non-trivial number of vertices (at least 10) and a non-trivial number of edges (at least 25). Think carefully about your test cases, because they should illustrate that your algorithm works and will therefore be useful in debugging.

• Please Zip your code (just the .java files and the test cases – no .class files, please) with the test cases into one file and e-mail it.

• Also, turn in a printout of your code.

• Finally, turn in a report (at most one or two pages) that described how you approached the problem, what troubles you had, and what you learned.