CS 223
Laboratory Assignment #5
Due: at the end of the lab in Week 16 (May 4, 07)

The Problems

1. Minimum Spanning Tree (Prim's Algorithm)
Write a program to compute a Minimum Spanning Tree using Prim's algorithm. Prim's algorithm is simple but uses a priority queue data structure. You can either use JAVA's built-in Priority Queue class, or you can use your own for Lab2.

Input: Your program should read an undirected weighted graph from a file while looks like:

```
<number of vertices>
<number of edges>
<endpoint 1> <endpoint 2> <weight>
<endpoint 1> <endpoint 2> <weight>
...
```

Example:

5
6
0 3 8
1 2 20
2 3 5
1 3 9
2 4 11
0 4 16

represents the following graph:

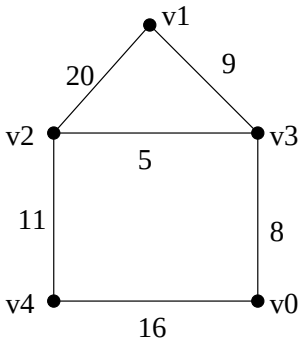

Figure 1. A simple graph.

Output: Print out which edges belong to MST of this graph. Example:

```
MST for input graph:
(v2,v3), (v1,v3), (v0,v3), (v2,v4)
Total weight:  23
```

You sample output should include this example and another graph (generated by yourself) which contains at least 15 edges.

2. Montana Travel Planner (Dijkstra's Algorithm)

Montana Department of Transportation has a raw data file (http://www.cs.montana.edu/courses/223/city.txt) for the direct distances between cities/towns around the whole Montana (feel free to do some pre-processing to make it more readable). In this problem, you need to design a Java program (including a GUI) to compute the optimal driving routes between cities and towns within Montana.

To make the problem technically more interesting, we assume that the vehicle has a limited range when it is driven between two cities/towns (say your vehicle can only travel for $\leq$130 miles and you have to stop over at some city/town to pump gas) — this will help trimming some long edges in the graph. You need also to assume that your vehicle has certain speed limit as well, say $\leq$60 miles/hour (think of you have an old car from 1970's).

Your program should have four input text fields: **Vehicle Range**, **Vehicle Speed**, **Source City** and **Destination City**. It should have a button **Find Route**, when pressed it should return the shortest route from **Source City** to **Destination City**. Each leg of the route should be no longer than the **Vehicle Range** parameter. Print this route in a separate window, which should also include the total distance and the estimated time for travel (following the given **Vehicle Speed**).

Run the program for at least 3 different source/destination pairs and hand in the corresponding output.

3. All-Pair Shortest Path (Floyd's Algorithm)

Write a program which implements Floyd's algorithm. Use the same data file from the Montana Department of Transportation (http://www.cs.montana.edu/courses/223/city.txt). As in question 2, trim all long edges, say trim all those links >130 miles.

Output 1: As there are about 70 cities/towns in the file, for the output, you can print and hand in at least three disjoint sub-matrices from the output you have obtained. For instance, each of these sub-matrices are 8×8 (8 rows and 8 columns); i.e., first 8 rows and first 8 columns, last 8 rows and last 8 columns, and row 15-22 and column 15-22, etc.

Output 2: Show the actual shortest route from Bozeman to Great Falls and from Billings to Missoula.