CS W3137 Assignment 5. Due 4/21 at class time

Non-programming problems (30 points):
1 - Problem 7.12 (5 points)
2 - Problem 7.35 (5 points)
3 - Problem 7.37 (5 points)
4 - Problem 9.2 (5 points)
5 - Problem 9.19 (5 points)
6 - Problem 9.20 (5 points)

Programming Problems (70 points):

Programming 1 (35 Points) - Fibroconnect.com has asked for your help as a consultant. Fibroconnect wants to connect major US cities via fiber optic cable to create their own internet. They have asked you to find the minimum total cable length needed to create a path to connect every city. The input to your program will be the file http://www.cs.columbia.edu/~jweisz/W3137/HMWK/cityxy.txt
which contains the name of each city and its X and Y coordinates. Below are a few lines from the file:

Boston 2542 1230
Chicago 1756 1048
Omaha 1350 990
SaltLake 565 1025
Peoria 1676 962
NewYork 2435 1081
…

Here is what you must do:

(a) Read in each city and its X Y coordinates.

(b) (7 points) Assume the cities in the graph are fully connected - an edge exists between every city pair. Using a 2-D Euclidean distance metric for the edge costs, print out all the edge pairs and their distances. Print them out as: city1 city2 distance. Also, to reduce the print out length, make sure you only print out each city pair and its distance once (i.e. DO NOT print “city1 city2 distance” and also print out “city 2 city 1 distance”).

(c) (18 points)Using these edge costs (path lengths), implement Kruskal’s algorithm for finding the minimum spanning tree of the city graph. You are required to use a Priority Queue in your algorithm (Java Collections classes allowed). Output the edge pairs that make up the minimum spanning tree.

(d) (10 points) Using a JAVA GUI window, draw a map of the cities, along with the edges that represent the Minimum Spanning tree as calculated by Kruskal’s algorithm. (see figure 1 for the fully connected city graph BEFORE the MST computation). Note: the X Y coordinates of each city is in a coordinate system in which the first coordinate (X) increases from left to right (same as the JAVA GUI X coordinate system), and the second coordinate (Y) decreases from the top of the page to the bottom (the reverse of the JAVA GUI Y coordinate system). Therefore, you need to transform the Y coordinate of each city to reflect this. Also note that all city names are a single string (i.e. no white space as “Las Vegas” becomes “LasVegas”) to make it easier to input and identify city name strings.

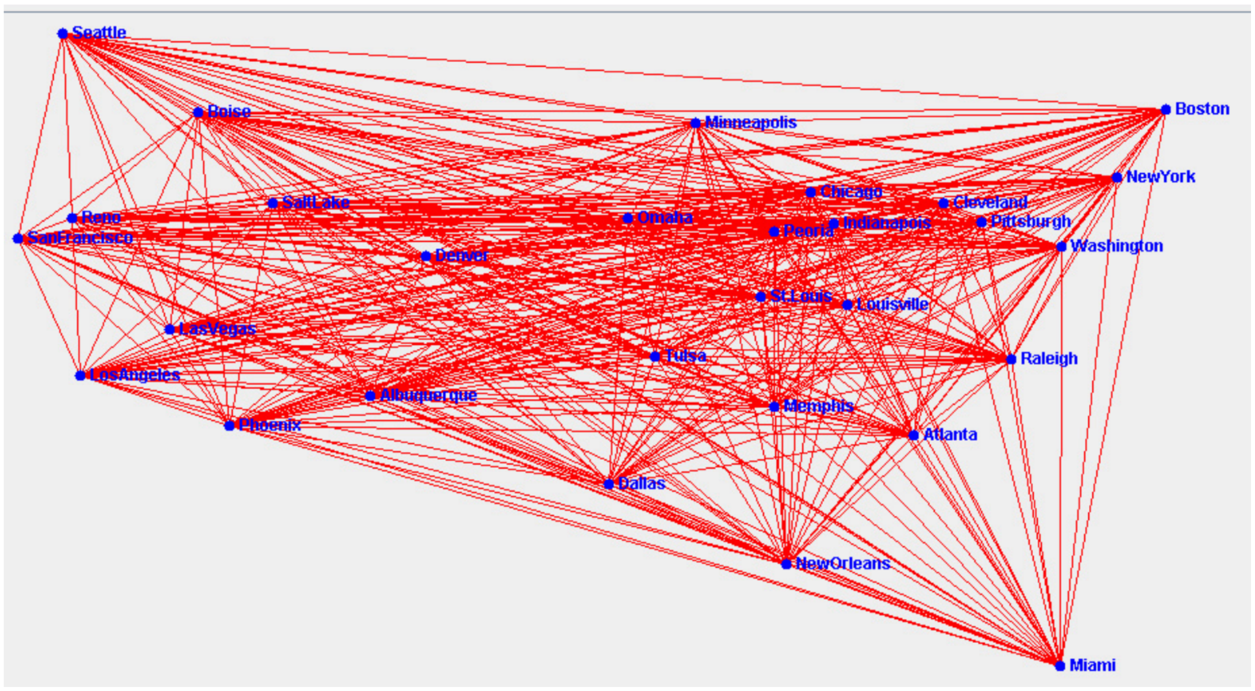

Figure 1: Sample output of fully connected US City map before MST computation. Programing Problem 1 (Minimum spanning tree) will produce a similar map with only the edges of the MST draw in.

Programming 2 (35 Points) -

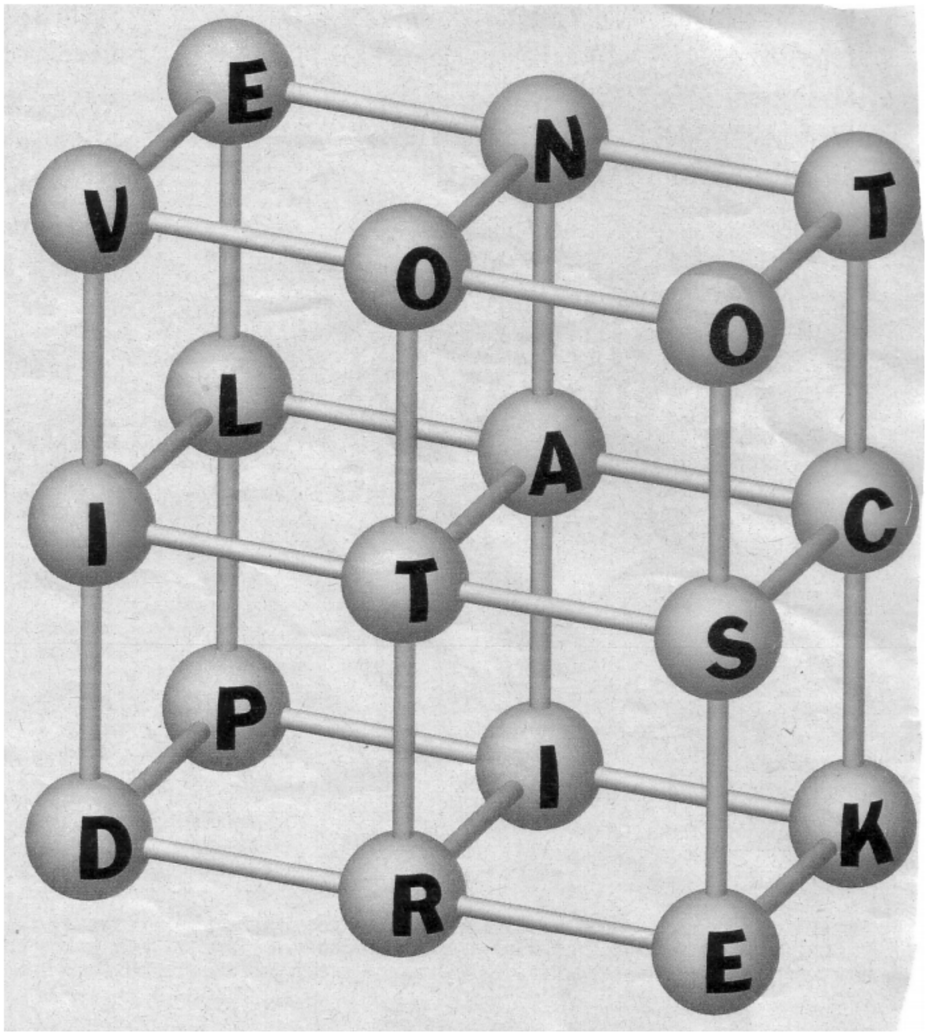

Figure 2: *3-D Word Hunt.* Find as many legal words of length N as you can. You may return to a letter and use it more than once (e.g. word “TITAN”), but you cannot “stand” on a letter and use it more than once (e.g. word “DITTO”).

The New York Times Magazine has a puzzle called 3-D Word Hunt (see fig 2). In this puzzle, you need to search a 3D matrix of letters for legal words. Each node of the matrix contains a single letter and a list of connections to other adjacent nodes. The puzzle is solved by finding all legal words of length N. A word is formed by concatenating each of N adjacent letters as you traverse the matrix. Letters can be repeated, but a single node’s letter cannot be repeated by “standing” on the same node you are on. In graph terms, this means no self-loops.

(a) Read in the word hunt matrix. It is contained in the file:

http://www.cs.columbia.edu/~jweisz/W3137/HMWK/graph.txt
The file is organized as follows:

Letter
X Y Z coordinates of letter in the matrix
X Y Z coordinates of adjacent node
X Y Z coordinates of adjacent node
Letter
X Y Z coordinates of letter in the matrix
X Y Z coordinates of adjacent node
X Y Z coordinates of adjacent node
X Y Z coordinates of adjacent node
Letter
(and so on - note that not all nodes have the same number of adjacent nodes)

Search for legal words of length N, where N is a user specified parameter. A word is legal if it can be found in a dictionary. You will use /usr/dict/words, the standard Unix dicitonary as our dictionary. You will do a lookup for a legal word using a Hash Table of all the words in /usr/dict/words. Your hash function will be a standard polynomial multiplier for strings. For Collision Detection and resolution you will use Open Addressing with Linear Probing. Since there are 25,143 words in the dictionary, use a table size of the smallest prime number greater than 2 * 25,143, which would be 50,287. Your program should output all legal words of length N that can be found in the dictionary by traversing the matrix. N is a user input.

Here are the first few lines of the file:

```
v
0 0 0
1 0 0
0 1 0
0 0 1
e
0 0 1
0 0 0
0 1 1
1 0 1
o
0 1 0
```

```
0 0 0
0 2 0
0 1 1
1 1 0
....
```

Extra Credit (5 points): Program the Word Hunt in a GUI with a display of the matrix and an animation of each letter as it is visited.

Note: The Unix dictionary file is not very good, since it leaves out some words and also most suffixes and prefixes (e.g doesn't do plurals etc.) For testing and fun, you may want to try a larger dictionary which contains 152,476 words (don't forget to increase your hash table size if you use it):
http://www.cs.columbia.edu/~jweisz/W3137/HMWK/largedictionary.txt