COMS 3137 Data Structures and Algorithms. Homework 5

Submit your electronic files via `http://courseworks.columbia.edu`. Do the written assignments electronically if possible, since it is easier for all parties involved. Use the provided makefile for the programming problems.

Directory structure to be followed for submission:

```
Root folder : <Your UNI>_homework5
<all Java source files>
Readme File (describing all your files and notes to the graders)
Theory solutions (optional)
Makefile (optional if you modified the provided Makefile)
```

I.e., this directory structure should mean that typing "make" in your submission directory will compile your code with no errors. Archive your submission directory using the command (in the containing directory):

```
tar -czvf <Your UNI>_homework5.tar.gz <Your UNI>_homework5
```

1. If you are creating your classes within packages, please maintain the package directory structure during submission.
2. Include the makefile *only* if you need to change the makefile provided to you. However you are strongly advised to conform to the provided makefile.
3. The code should be commented as appropriate or the details should be explained in a readme.
4. If you are handing your theory on paper, do not print out your code.

Multiple Submissions: You can submit multiple times, but we will only consider the latest submission based on the timestamp in courseworks. Please give at least 1-2 minutes between two submissions.

Late Submissions: Any unexcused late homework submissions will be penalized 10% each day it is late. The penalty begins at the beginning of class the day the assignment is due. The 10% penalties will continue for 3 full days at which point no more late submissions will be graded. If you are submitting late, do not use courseworks and mail your submission to all the TAs.

Upload your archive to the Class Files section of courseworks in the Homework 5 subdirectory. It is also recommended that you keep a pristine copy of your submission folder in case there is any submission error.

1 Written Problems

Make sure your solutions are clear. Diagrams and math are often insufficient to convey exactly what you mean, so supplement with some text. Either pseudocode or Java are acceptable when asked to provide algorithms. Nevertheless, clear, concise English is often preferable.

1. **Weiss 9.1** (3 points) Find a topological ordering for the graph in Figure 9.79.
2. **Weiss 9.5**
   (a) (3 points) Find the shortest path from A to all other vertices for the graph in Figure 9.80.
   (b) (3 points) Find the shortest unweighted path from B to all other vertices for the graph in figure 9.80.
3. **Weiss 9.11** (3 points) Find the maximum flow in the network of figure 9.79.
4. **Weiss 9.15**
   (a) (3 points) Find a minimum spanning tree for the graph in Figure 9.82 using both Prim's and Kruskal's algorithms.
   (b) (1 point) Is this minimum spanning tree unique? Why?
5. **Weiss 9.31** (3 points) Give an algorithm to find in an undirected (connected) graph a path that goes through every edge exactly once in each direction.
6. **Weiss 9.38b** You are given a set of N sticks, which are lying on top of each other in some configuration. Each stick is specified by its two endpoints; each endpoint is an ordered triple giving its x, y, and z coordinates; no stick is vertical. A stick may be picked up only if there is no stick on top of it.
   (a) (**0 points**; I'm just including this for reference.) Explain how to write a routine that takes two sticks a and b and reports whether a is above, below, or unrelated to b. (This has nothing to do with graph theory.)
   (b) (3 points) [Assuming we have the routine from part (a), g]ive an algorithm that determines whether it is possible to pick up all the sticks, and if so, provides a sequence of stick pickups that accomplishes this.

2 Programming Problems

1. (20 points) A time traveler comes back from the 23rd century and shows you how to create a new kind of indestructible cable that supports infinite bandwidth and can send data instantly. (They figured out how to move information faster than the speed of light via their time-travel technology) Unfortunately, it costs millions of dollars to produce one mile of the cable. However, since we have infinite bandwidth and zero latency, it is only necessary to connect all major cities into one connected graph, no matter the connectivity structure.

Given the GPS coordinates of major cities around the world `cities.txt`[1], draw the connectivity structure that connects every city to every other city but uses the minimum amount of cable, under the assumption that the distance between any two cities proportional to

$$\sqrt{(\text{longitude}_1 - \text{longitude}_2)^2 + (\text{latitude}_1 - \text{latitude}_2)^2}.$$

(This is a bad assumption in real life because it essentially assumes the world is flat.)

Write a program `CityConnector.java` that reads the input file creates a fully connected graph with weights computed by the distance formula above, executes Prim's Algorithm, and displays the graph of the cities connected using the minimum distance tree (using JScrollPane and `GraphDraw.java`). Label the nodes with the city names. You will need to transform the GPS coordinates to a reasonable scale to use as your x,y coordinates. Nodes will overlap (due to very dense areas) no matter how much you scale, so just choose a reasonable scale that lets you see the connectivity.[2]

- Create a graph class `CityGraph.java` that uses an adjacency matrix.
- CityGraph should have a method `Prim()`, that changes the adjacency matrix to the minimum spanning tree using Prim's algorithm.

2. (18 points) You are a city pigeon who wants to travel and see the world. Unfortunately, since you are addicted to city life, you cannot travel more than a certain distance without having to stop at another major city. You want to have a program that outputs the shortest paths between any two cities (with pit stops along the way). Luckily, you found the `cities.txt` file and learned about the Floyd-Warshall All-Pairs-Shortest-Path algorithm from your Data Structures and Algorithms class.

   Using the distance formula from the previous problem, connect all cities within a radius of d degrees. Then compute the shortest paths (the cost and the actual path) for all pairs of cities. Your program should take as input the text file containing the city data and a number representing the number of longitude-latitude units you are able to travel without stopping. Then it should poll the user for a start city and an end city, and output the path and total distance between the two. You should add methods such as `findShortestPaths()` and `printPath()` (or however you wish to organize your code) to `CityGraph.java` to support this functionality. Example usage would be as follows.

```
java ShortestPathFinder cities.txt 30
All paths computed
Enter the start city:
New York
Enter the destination city ('quit' to exit):
Paris
Shortest path:
Paris (48.86, 2.34) <- Malaga (36.72, -4.42) <- Dakar (14.72, -17.48)
<- Fortaleza (-3.78, -38.59) <- Maturin (9.75, -63.17)
<- Santo Domingo (18.48, -69.91) <- New York (40.67, -73.94)
Total path length: 129.18870704459715
```

[1]City data obtained from `http://gael-varoquaux.info/blog/?p=84`. The data format is tab-delimited and each row is: (city name) (longitude) (latitude).

[2]I found 20 × (longitude − minLongitude) and 20 × (maxLatitude − latitude) produced a reasonable size, where minLongitude is the westernmost longitude and maxLatitude is the northernmost latitude.