CSC172 PROJECT 4

STREET MAPPING

1 Introduction

This project will require you to create a rudimentary mapping program in Java. Given a data set representing the roads and intersections in a specific geographic region, your program should be able to plot a map of the data, provide shortest path directions between any two arbitrary intersections using Dijkstra's algorithm, and be able to generate the minimum weight spanning tree for the entire map.

2 Input Data

The geographical data necessary to run your application will be provided in the format of a tab-delimited text file. Each line will consist of 4 pieces of data, as defined below:

Intersections start with “i”, followed by a unique string ID, and decimal representations of latitude and longitude.

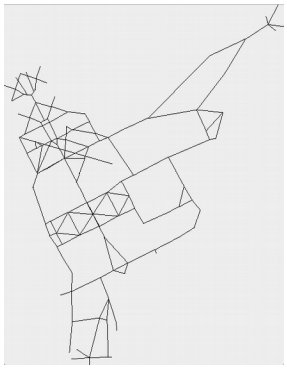
```
i    IntersectionID Latitude  Longitude
```

Roads start with “r”, followed by a unique string ID, and the IDs of the two intersections it connects.

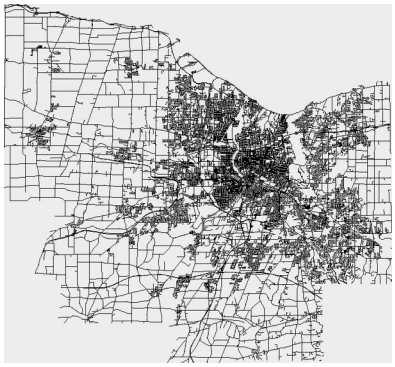
```
r    RoadID    Intersection1ID    Intersection2ID
```

You may safely assume that all input files will declare intersections before their IDs are used in roads.

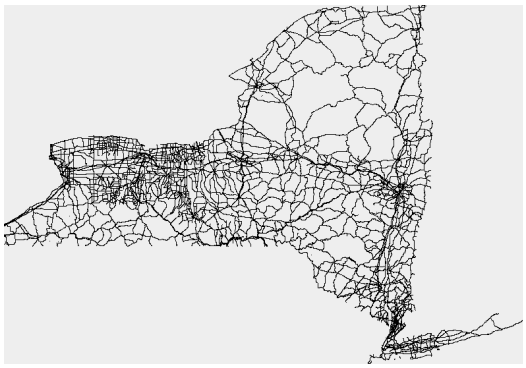
Three different data sets are provided for your testing purposes with this project. The first data set, “ur.txt” represents a subset of the pedestrian sidewalks on our campus. Building entrances have meaningful intersection IDs such as “CSB” or “SUEB” for your convenience. The second and third data set test your program's ability to scale well, with the latest census data on roads in Monroe County and NYS.



ur.txt



monroe.txt



nys.txt

3 Deliverable

Your program will be evaluated on how well it accomplishes the following three tasks and command line specification:

Basic Mapping

- ☐ Implement your own Graph, Node and Edge classes. You may use your previous implementations from lab or the textbook, but be sure to cite any sources you use.
- ☐ Construct a Graph object using the information read in from the specified input file
- ☐ Draw the map using Java Graphics (no third party graphing libraries allowed). The map should scale with the size of the window.

Directions Between Intersections

- ☐ Implement Dijkstra's algorithm to find the shortest path between any two arbitrary intersections, as provided by the command line arguments.
- ☐ When the shortest path has been discovered, the intersections followed to reach the destination should be printed out to the console in order. Additionally, your program should print out the total distance traveled in miles.
- ☐ Finally, if the program is displaying the map, it should highlight (in a different color, stroke width, etc.) the solution path found.

Minimum Weight Spanning Tree

- ☐ Imagine you're a Meridian and need to show a prospective student around the entire campus. Compute minimum weight spanning tree of the map to accomplish this task.
- ☐ Once the minimum weight spanning tree has been computed, a list of all the roads traveled should be printed to the console.
- ☐ If the program is displaying the map, the minimum weight spanning tree should be displayed and highlighted in a different color, stroke, etc.

Command Line Arguments

Your program should accept the following set of command line arguments:

```
java ProgramName map.txt [-show] [-directions startIntersection
     endIntersection] [-meridianmap]
```

You can safely assume that your program will never be run with both the `-directions` and `-meridianmap` flags. Your program should only display a map if `-show` is present.

4 Getting Started

To help you better understand the map data, visualize where certain roads or intersections are, and verify that your shortest path algorithm is producing the correct answer, a website has been set up where you can play around with the UR campus map data at https://www.ryanpuffer.com/172.

It is highly recommended that you get your program to work with the UR campus map before moving onto Monroe County or NYS map data. The size and complexity of those maps introduce new issues that are best handled after you've mastered the basic project requirements.

5 Hand In

Hand in the source code from this lab at the appropriate location on the Blackboard system at my.rochester.edu. You should hand in a single compressed/archived (i.e. “zipped” file that contains the following.)

1. A plain text file named README that includes your contact information, a detailed synopsis of how your code works and any notable obstacles you overcame, and a list of all files included in the submission. If you went above and beyond in your implementation and feel that you deserve extra credit for a feature in your program, be sure to explicitly state what that feature is and why it deserves extra credit.

   The README for this project should clearly explain any design or implementation choices you made, the expected runtime of plotting the map, finding the shortest path between two intersections, and generating the minimum weight spanning tree. Stating the Big-Oh value is not enough – explain the intermediary calculations you took to determine the runtime of your program and comment on how these values will affect how your program scales with larger data sets.
2. Source code files representing the work accomplished in this project. All source code files should contain author identification in the comments at the top of the file.
3. A plain text file named OUTPUT that includes author information at the beginning and shows the compile and run steps of your code. The best way to generate this file is to cut and paste from the command line.

6 Grading

30% Basic mapping

- 20% Implementation
- 10% Correctness

30% Directions between intersections

- 20% Implementation
- 10% Correctness

30% Minimum weight spanning tree

- 20% Implementation
- 10% Correctness

10% README with detailed description of how you structured your project, approached the challenges the larger maps presented, and the runtime analysis of your code.

Extra Credit is available for projects that have interactive and/or exceptionally beautiful maps.