Project #3 (STREET MAPPING)

CSC 172 (Data Structures and Algorithms), Fall 2022,
University of Rochester
Due Date: Wednesday December 14th 2022 (end of day - 11:59 pm)

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 and provide shortest path directions between any two arbitrary intersections using Dijkstra's algorithm.

Input Data

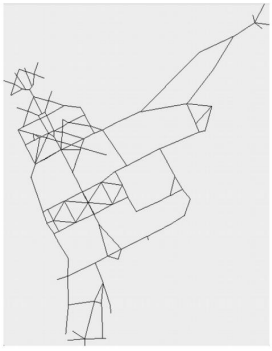
The geographical data necessary to run your application is provided in the format of a tab-delimited text files. Each line will consist of 4 pieces of data, as defined below:
Intersections start with "i", followed by a unique string ID, and decimal representations of latitude and longitude.

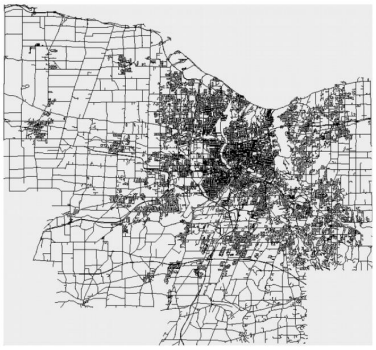
i IntersectionID Latitude Longitude

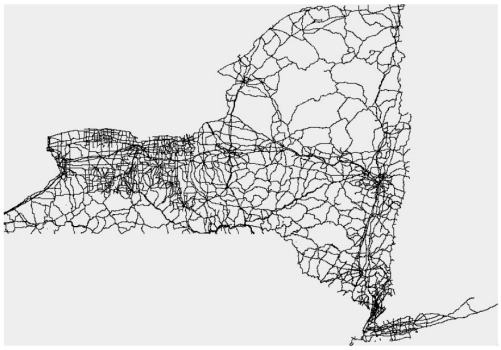
Roads start with "r", followed by a unique string ID, and the IDs of the two intersections it connects.

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. You can download these three files as a zip file from https://www.cs.rochester.edu/u/pawlicki/csc/172/CSC172FL22_project3_data.zip


ur.txt


monroe.txt


nys.txt

Deliverable

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

Basic Mapping

- Implement your own `Graph`, `Node` and `Edge` classes. For this you may use code available online or other sources, but you must cite the source.
- 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. Talk to your lab TAs for further clarification if required). 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.

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

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

Your program should only display a map if `--show` is present. Below, you can find how a few of the sample runs may look like:

```
java StreetMap ur.txt --show --directions HOYT MOREY // Showing both map
    and the directions
java StreetMap ur.txt --show  // Just showing the map
java StreetMap ur.txt --directions HOYT MOREY // Showing the map is
   optional.
```

Getting Started

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.

Hand In

Each student must submit individually, no team projects.

Hand in the source code from this project at the appropriate location on the Blackboard system at my.rochester.edu. You should hand in a single compressed/archived (i.e. "zipped") file named `proj3.zip` which contains the following

1. A plain text file named `README` that includes your (and your collaborators) 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 and finding the shortest path between two intersections.

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.

Grading Rubric

30% Basic mapping
- 15% Implementation
- 15% Correctness

50% Directions between intersections
- 25% Implementation
- 25% Correctness

20% README with colaboration information, detailed description of how you structured your project, approached the challenges the larger maps presented, and the runtime analysis of your code.

Detailed description of your project should include: A brief summary of how your program works, Classes used, their private and public members and methods. You should state the input and output parameters of each method. See 'Hand In' section for further details.

Extra Credit is available for projects that have interactive and/or exceptionally beautiful maps. The Lab TAs will decide if you deserve any extra-credit. You must discuss extra credit with your lab TA in advance of the due date.

- For calculating the distance between two intersections, you must use This uses the 'haversine' formula. Check https://en.wikipedia.org/wiki/Haversine_formula. It's ok to use implementation of this method found online as it is as long as you cite the source.
- There might be two intersections which are not connected.
- 'show' and 'direction' may appear in any order.
- For a large map, we advice you not to add all intersections to the priority queue at the beginning.
- If you use a java.util.PriorityQueue, for changing priority of any entry, you must remove it and add it again. You are welcome to write your own heap implementation (if you want) to avoid removing the entry.
- We pushed the due date to the extreme leaving just enough time for us to finish grading before it's due. Submit your code as many times as you wish. It serves also as your backup!