Project (Option 2): Emory MapQuest[1]

Due: Friday, May 4, 11:59pm

(Optional) demo due: Sunday, April 30, 11:59pm

---

1 Requirement

Implement a program for finding the shortest route from an origin city (source) to a destination city (target) based on a shortest path algorithm, the A* algorithm.

Output The program takes the source and target as commandline arguments, and outputs a path in the following form.

```
Route: <total distance>
<source>  <city 1>  <road 1>
<city 1>  <city 2>  <road 2>
  ...
<city K>  <target>  <road K>
```

Each line consists of two cities and the highway name that connects them. The second city of a line must be the same as the first city of the next line. Together, the sequence of unique cities from top to bottom for the connecting cities represents the computed route from the source to the target. As an example, the route from Atlanta to Memphis would appear as follows.

```
Route: 433 miles
Atlanta     Chattanooga I-75
Chattanooga Nashville   I-24
Nashville   Memphs      I-40
```

Input file The program should build a map from a given input file. The input file will have a list of cities, their latitudes and longitudes, followed by, for each city, its immediately adjacent cities, names of roads that connect them, and the distance. An input file MapData.txt is provided.

```
<N>  // this is the number of cities on the map
<city 1> <lat 1> <long 1>
<city 2> <lat 2> <long 2>
 ...
<city N> <lat N> <long N>

<city 1> <N1>  // N1 is the number of connected cities
<city 1 1> <road 1 1> <distance 1 1>
<city 1 2> <road 1 2> <distance 1 2>
 ...
<city 1 N1> <road 1 N1> <distance 1 N1>

<city 2> <N2>  // N1 is the number of connected cities
<city 2 1> <road 2 1> <distance 2 1>
<city 2 2> <road 2 2> <distance 2 2>
 ...
<city 2 N2> <road 2 N2> <distance 2 N2>
```

[1] All project related files including this handout, data files, and programs can be downloaded from the project directory `http://www.mathcs.emory.edu/~cs171000/share/proj/MapQuest`.

2 Suggested Class Design and Data Structures

You may consider implementing your program using several classes with the suggested data structures. Please feel free to modify it or design your own (in which case please justify it in your README file).

1. A class that represents a single city, which consists of a name (a String object), a longitude and latitude. Provide the usual `get` methods.

2. A class that represents a connection from a city, which consists of the other city C, a road name (a String object), and a distance. This class will be used both for representing the static map information, and for representing stops along a path/route that is being calculated. In the first case, a connection represents how C can be reached (and its cost) relative to a given city. In the second case, a connection represents one of the stops of a path between two cities. Provide `get` methods for each of its variables.

3. Data structures to store the cities and connections (the map). You can use an array to store the list of cities. You can use another array to store the list of connections for each city (adjacency list representation of the map).

4. A Path class that consists an array of `Connection` (representing the stops along the path), an actual cost and an estimate (to the target).

   A method of the class is `extend`, which allows a city to be added to the end of current connections (and update the actual costs). It is important to make a copy of the connections to avoid aliasing, which may throw off the distance calculations.

5. A priority queue of `Path`, ordered according to the sum of the actual and the estimate cost. You need to implement your own priority queue either using an array or a binary heap tree.

Program Details : The sketch of the A* algorithm is as follows

```
given input: source; target;

initialize a priority queue PQ to empty
guess = path of just the source
while (guess.last not the target) {
  for each c connected to guess.last
    add guess.extend(c) to PQ
  guess = remove PQ // remove the path with the highest priority
}
```

In order to compute the estimate distance, there are several formulas for calculating the straightline distance between any two cities (given their longitudes and latitudes). The following pages gives the haversine formula:

`http://www.movable-type.co.uk/scripts/latlong.html`

3 Displaying the Route

A program DrawRoute is provided for graphically displaying the route that your program produces on a US map. You can run the program directly using the output from your program. For extra credit, you can build a GUI interface for the user to input source and target cities and then display the shortest route on the map.

In order to use the program, you need to download the files: DrawRoute.jar, usa.jpg (with state borders), usa1.jpg (without state borders)

```
- To run the program without a background map, type

  java -jar DrawRoute.jar < inputRoute.txt

  where inputRoute.txt is the output from your program.

- To run the program with a background map, type

  java -jar DrawRoute.jar -b usa1.jpg < inputRoute.txt

  or you can use usa.jpg which gives a busier picture.

- To "pipe" the output of your program directly into the
  program, you can type

  java YourProgram < MapData.txt | java -jar DrawRoute.jar -b usa1.jpg
```

4 Grading

Base credits:

- Functionally correct program. (85 credits)
- Well organized code (good design of classes/methods). (10 credits)
- Error condition handling. (5 credits)

Extra credits:

- GUI or additional functionality and exceptional efforts and creativity. (up to 10 credits)

5 Submission

Please make sure that you have added the following statement at the top of each source code file you wrote and include the name of each team member.

```
/*
  THIS CODE IS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING
  A TUTOR OR CODE WRITTEN BY OTHER STUDENTS. Team Member Names.
*/
```

Please submit your project to Amy as a tar or zip file containing: your source files, the README file and any other files you are using in your project.

The project is due on May 4. However, you are encouraged to submit a demo version of your program by April 29. Based on the submitted demo versions, selected teams will be invited to present/demonstrate their project in class on May 1 (for 5 extra points). This demo version will not be used for grading.