Objective: This program reads the data on points (longitude, latitude) and intersections, and then creates a map (UofR, Monroe, NY State). It also calculates the distance between any two points (places) as prompted by the user.

How the Program works: There are 3 options to choose from: Show, Show Directions, or Directions. The user prompts what map and mentions the two locations. It initially reads the files. The vertices and edges are created accordingly. The program uses the haversine formula to calculate the distance between any two locations. Then, the distance is stored as weights with respective edges. The dijkstara’s to calculate the shortest path between the two locations. The graph is then printed and the path between the two locations is highlighted by animating a dot across the calculated path. The starting point, ending point, and distance is also displayed.

Difficulties Encountered: Initially, we used Adjacency Matrix implementation for the graph, which was not very efficient both in terms of space and time. We solved that by changing to the adjacency list implementation. That solved the error of space while drawing the error. However the program was still running into Java OutOfMemoryError: heap space & GC overload errors. We realized this was because we were using the MinVertex implementation of the Dijkstra which was maintaining a very large array. We changed that implementation to one which uses a Priority queue and that solved the problem.

Lastly we were facing trouble with keeping track of the path, using an array of linked lists and traversing through it was taking too long and would sometimes run into similar out of memory errors. For this we decided to add a parent attribute to the vertices and thus totally got rid of the path array. This drastically improved the rate and functioning of our code.

Run Time Analysis:

|  |  |  |  |
| --- | --- | --- | --- |
| Maptype/command | Map | Map+Directions | Directions |
| UofR | 0.72 | 0.98 | 0.63 |
| Monroe | 1.83 | 2.74 | 2.4 |
| NYS | 8.52 | 10.08 | 7.13 |

Runtime for Dijkstra’s algorithm is O((|V|+|E|)log|E|), where V is the number of intersections in the Graph and |E| is the average number of Edges in each Linked List.

|  |  |
| --- | --- |
|  |  |

Files Included:

StreetMap.java: Main class

Info.java: Jpanel to display Origin and destination

Canvas.java: Draws the map

Edge.java: Contains method to calculate distance and creates edges with its respective distance

Vertex.java: Stores latitude and longitude and parent of each vertex

Graph.java: Implements dijkstra’s algorithm

Workload Distribution:

Most of the code was collaborated on together. Harshil helped with the graph implementation and the extra-credit where he animated a dot across the edges from the origin to the final destination. Shivali helped with implementing of Dijkstra’s and using the haversine formula.

Extra Credit:

We have added a panel on the top of the canvas which displays the origin and destination. When the path between the two points is shown, a Red dot appears at the origin which animates through the desired route and ends at the destination. A message stating that you have arrived at your destination is displayed. At the end the distance between the two places is also displayed. The speed of the animating dot, depends on the map being prompted to display.