Street Routing

Introduction

Routing services, also called *navigation* services help people get from one place to another. Street map applications includes information about different places, streets connecting different parts of the city, average traffic, speed limit, average accidents on every street in the city, data regarding time taken and distance covered to reach from one place to another in the city for routing by different modes including car, foot and bicycle.

The data should be reliable as well, ways that should be connected are in fact connected, one-way roads are tagged, turn restrictions are mapped, and so on. In order to better calculate the fastest route, having the record of maximum permitted speed in every part of the city is important as well especially where the speed limit differs from the assumed maximum permitted speed for the road type per country and vehicle-type.

Here is the link to my implementation of a *street routing* program.

Algorithms-for-reals/Street Routing at main · harshitagupta1512/Algorithms-for-reals

Clone this directory and cd into it. Run the command gcc main.c project.c & ./a.out. The first line will have two space separated integers representing the number of places in the city i.e number of nodes in the city graph and the number of streets in the city i.e the number of edges in the city graph/map.

https://github.com/harshitagupta1512/Algorithms-for-reals/tree/main/Street%20Routing

harshitagupta1512/
Algorithms-for-reals

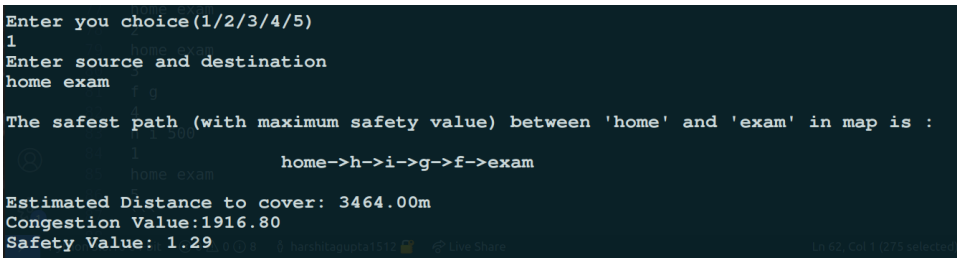
1 Contributor 0 Issues 0 Stars 0 Forks

When given the description of the places and streets of the city as input, the program provides functionalities to find the fastest or safest path between any two nodes in the graph, deleting a street between any two nodes or changing the *average number of cars on a street per hour*, as this factor can be quite dynamic based on time, festivals etc. The weight of each street is decided based on multiple factors

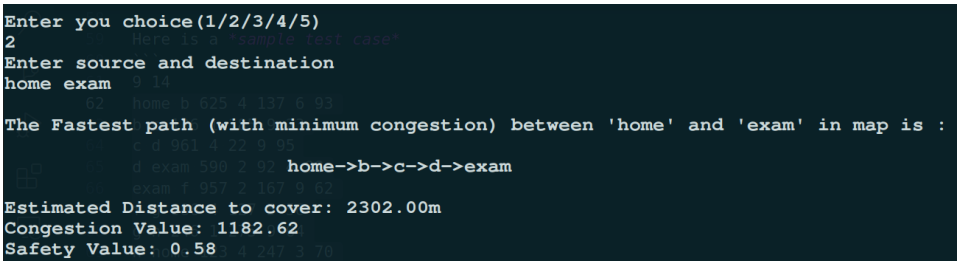
`Street_length(float)` `Number_of_lanes(int)` `Average_number_of_cars(int)` `Avergae_number_of_accidents(int)` `Speed_Limit(int)`

Furthermore, every street has a safety value

```
SD.traffic = 1.00f / (SD.numLanes * 10 * 0.6) + 1.00f / SD.speed_limit * 0.4 + SD.num_cars * 1;
SD.safety_value = 1.00f / ((SD.num_accidents * 0.8) + (0.2 * SD.speed_limit / 10)); //Safe Routing
SD.weight = SD.traffic * 0.6 + SD.length * 0.4;
```



Consider this example, where find the fastest and safest path between the home and exam nodes.



We delete the street between b and c and find the fastest path between home and exam nodes again

```
Enter you choice(1/2/3/4/5)
3
Enter the source and destination of the street to be deleted
b c
```

```
Enter you choice(1/2/3/4/5)
2
Enter source and destination
home exam

The Fastest path (with minimum congestion) between 'home' and 'exam' in map is :

                home->h->g->f->exam

Estimated Distance to cover: 2212.00m
Congestion Value: 1374.62
Safety Value: 0.69
```

```
1.FIND THE SAFEST PATH
2.FIND THE FASTEST PATH
3.DELETE STREET IN THE CITY MAP
4.CHANGE THE STREET DATA FOR ANY STREET
5.EXIT

Enter you choice(1/2/3/4/5)
4
Enter number of streets for which you want to change data for :-
1
Enter the changed streets' data (source destination (average_number_of_cars_on_the_street_per_hour))
h i 500
Done
```

Dijkstra Algorithm

Dijkstra algorithm is a very famous greedy algorithm. It is used for solving the single source shortest path problem. It computes the shortest path from one particular node to all other remaining nodes. Dijkstra algorithm works only with connected graphs, directed/undirected edges with positive weights. It provides the value or cost of the shortest paths and by making minor modifications in the actual algorithm, the shortest paths can be obtained.

In our program, we use the algorithm to find the fastest (with minimum congestion) and the safest (with maximum safety value and minimum probability of accidents) path between the source and the destination entered by the user. The corresponding function gives us the congestion value, safety value and the complete path.
Working - The dijkstra code we used in this program runs with queues. We run breadth first search and then enqueue every element into the queue and checked for their neighbours to find the least of them and continued the process until the queue was empty and stored each vertex shortest paths in the path list and finally printed the path list for the shortest path between source and destination entered by the user. The time complexity for the Dijkstra algorithm is O(NE), where N-Number of nodes/places, E-Number of Edges.