Assignment 3

External Documentation

Submitted by

Sakib Sadman

B00934956

Submission Date: March 11, 2023

Overview

The implementation of this program follows the method descriptions in the assignment system. Refer to that document for those method requirements.

The main goal of this program is to create a map by adding streets and finding efficient paths for a delivery service from source to destination. In context of the assignment, efficient paths prioritizes routes without left turns unless absolutely necessary.

It contains additional features like finding furthest street from source, detecting loops, getting simplified routes etc.

Files and external data

In addition to the classes provided with this assignment, this implementation has three (four including main class) additional classes.

- Main- contains main method, it is where the program is executed from.
- Street- A class to capture streets in the map. Aside from keeping street id, length etc, it also maintains two lists, for keeping track of neighbouring streets.
- Edge- Class to capture relations between connected street. This class contain information such as distance, turn direction, entry point (the intersection used to enter the street).
- Leg- Captures leg number, name of the street and the turn direction of a route.

Assumptions

- Only time allowed to take left turn is when the only street available from the current leg is on the left. Otherwise it won't take left turns even if the destination is not reachable otherwise. For example, in figure 1, although the only way to reach destination is by taking a left turn from source, it will not take that turn.

  It would however, take left turn into street **A** if street **B** did not exist.

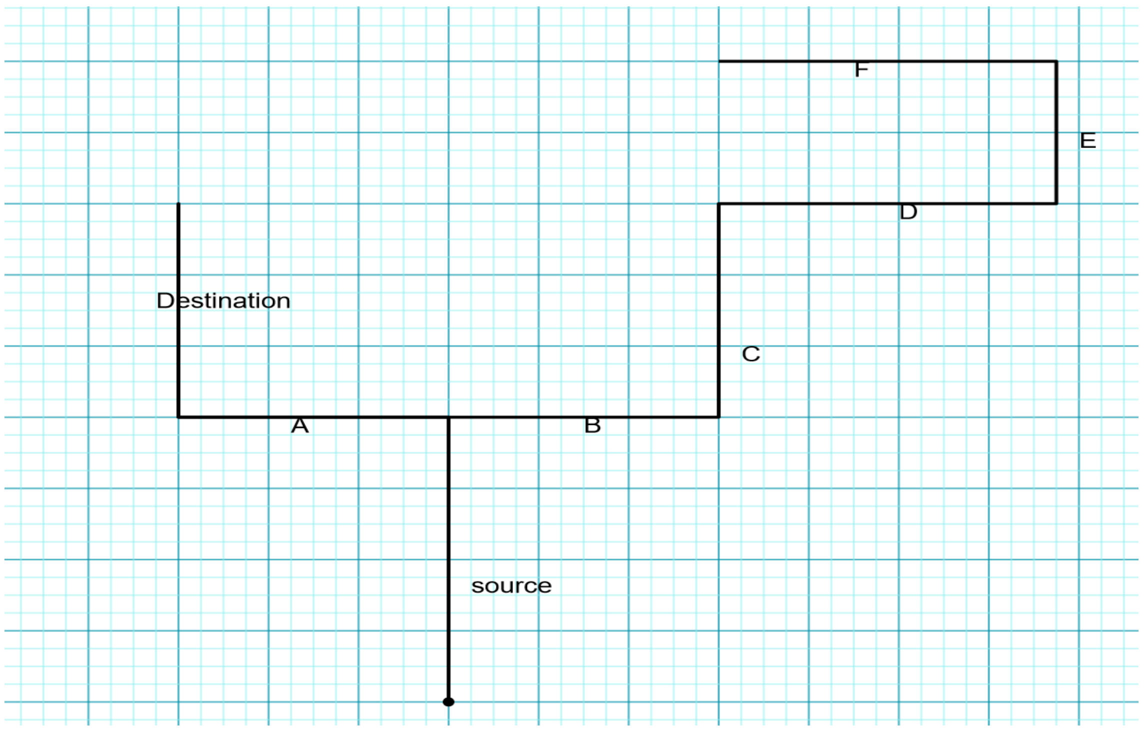

Figure 1: streets on a map

Choices

- For cases when we appear on the other side of the destination street (destination location is on the other side of the street), as per requirement on this assignment, we can take u-turn only if that street is a dead end.
  If that street isn't a dead end, I have chosen to return the route leading upto that street. The driver is expected to park his vehicle and cross the street on foot.

Key algorithm and design elements

MapPlanner

addStreet

First the method checks how many common intersections exists with the rest of the map. If there are 0 connections, it is added normally. If two common intersections exists, we return false. If one maximum common intersection exists with another street in the map:

We store the connection between them in two steps:

Once in the street object's arraylists. Street objects maintains two arraylists to keep track of neighboring streets, one for neighboring streets connected with it's start point and one for neighbors connected to its endpoint. This helps determine if left turn should be taken and whether it is possible to take u-turns.

Also, the connections are stored with an adjacency list. This list helps keeping track of the distance, street id, turn direction, point used to enter that street etc.

furhestStreet

For furthestStreet function, shortest distance from depot location to all streets of the map are stored in an array by implementing Djiktra's algorithms. The street with the maximum distance is returned afterwards.

routeNoLeftTurn

First, we append the depot street with the turn needed to go to the next leg. Then, Djikstra algorithm is implemented to reach destination street without taking left turn unless only a ledt turn street exists in the current street's endpoint.

After reaching destination street, if we enter from that street's start point, we check if the depot side matches the destination street side. If it matches, we return true.

If we enter the destination street from the end point, we return true if destination side does not match depot side, since entering from the end point of the street of flip it's street side.

If we can't make u-turns on the last street, return the route leading upto the destination street.

Route

appendTurn

Add the parameters to a leg object with would record it's street id, leg number and turn direction and store that object in an arraylist.

turnOnto + turnDirection

Since leg number stands from one and they are stored at an arraylist starting from index 0, return item from arraylist of index legnumber-1.

Length

For first leg and last leg, add half of street length. For other streets, add full street length.

Loops

Iterate over the arraylist of type leg, if a keep track of the starting leg. If a common intersection is found, instantiate a new subroute, passing the start leg, the last leg until the common intersection was found and the route.

After that, add it to an arratlist of type SubRoute and re-initialize the route.

Simplify

Initialize a new route. Keep iterating through legs arraylist, appending turns in that route except for straight turns until the end.

Limitations

This program does not take left turns even if reaching the destination is not possible otherwise (like the one depicted in figure 1.). A better implementations would be to add backtracking to it takes left turn after exploring all paths other than left turns