CAR ROUTING CHALLENGE FOR AUTONOMOUS VEHICLE

-Amey S Kulkarni

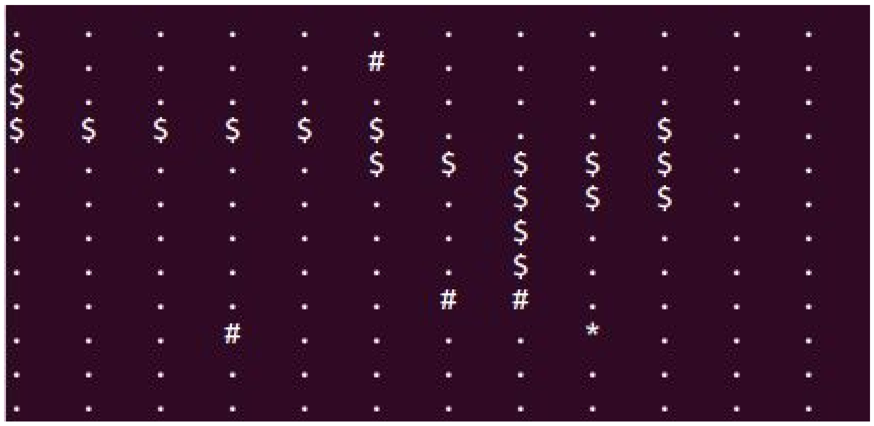
I have worked on solving the car routing challenge. The code is written in C++ and is attached in the submitted .zip file.

The Problem Statement

An autonomous car is to be used to perform services of the likes of Uber pool and Lyft Shared Rides. The car is deployed to collect a person's friends from different places in the city. The car routing console will get requests from all these people. These requests are stored in the JSON file. The JSON file stores the name of the traveler, the start and the end point. The task is to plan a path for the car to drop everyone in the least time possible. This is a typical case of the high-level planning problem. The algorithm which I have devised is described below. All the justification for taking a particular decision are also stated.

Algorithm

1. Take user input on the number of rows and columns in the city.
2. Read the JSON file and store the details of all the travelers in a vector of class objects.
3. In every loop, ie. after every unit movement of the car, it is checked if any new request needs to be recorded(Dynamic requests handling).
4. The car starts at (0,0)
5. All the start points in the city are populated in the vector "allstartp"
6. If a start point is reached, its corresponding endpoint gets unlocked and it appears on the visualization map(also gets populated in the vector "endp").
7. The program at each step(in each loop) calculates the distance from all the destinations(either start points or unlocked endpoints) and decides to go to the closest point.
8. If a start and an end point are at an equal distance from the current position of the car, the car decides to go to the endpoint to reduce the idle time the existing traveler spends in the car. It is assumed that the car will be notifying each traveler about how long will be the waiting time so that the waiting traveler can put his/her time to use.
9. Provision or rather a code of multiple people boarding and dropping at the same point has been given.
10. The time spent by each traveler in the car is calculated and the total time taken to complete the whole process is also calculated.
11. The visualization map is a small demonstration of the whole process. The city junctions are represented as dots and the movement of the car by the '$' sign. The start points for all the travelers are denoted as '*' in the map and the unlocked endpoints are symbolized as '#'. Symbolically speaking, $ should move within the map to collect all the * and # is the fastest manner possible.
12. Information on the number of people in the car and board/drop status of the traveler is published in each time step.



```
.  .  .  .  .  .  .  .  .  .  .  .
$  .  .  .  .  #  .  .  .  .  .  .
$  .  .  .  .  .  .  .  .  .  .  .
$  $  $  $  $  $  .  .  .  $  .  .
.  .  .  .  .  $  $  $  $  $  .  .
.  .  .  .  .  .  .  $  $  $  .  .
.  .  .  .  .  .  .  $  .  .  .  .
.  .  .  .  .  .  .  $  .  .  .  .
.  .  .  .  .  .  #  #  .  .  .  .
.  .  .  #  .  .  .  .  *  .  .  .
.  .  .  .  .  .  .  .  .  .  .  .
.  .  .  .  .  .  .  .  .  .  .  .
```

Figure 1. Visualization map

How to run the code:

1. The CmakeLists.txt and other related files are configured and attached.
2. Two programs are in the folder namely: “autocar.cpp” and “carroutingreq.cpp”.
3. Both programs essentially do the same thing. I have simplified the carroutingreq.cpp so that its easier you the judge to check the output.

    a. Program carroutingreq.cpp: In this program, the dynamic request handling concept is simulated just for demonstration. It is assumed that new dynamic requests are obtained only at time 20 and 29. The user can input as many requests as required at these points of time. This is just done to prove the capability of the code.

    b. Program autocar.cpp: This program checks the dynamic requests at each time step. It can take any amount of requests at each point of time and reroute the car as required. This is a full proof solution but as a user, the judge, at each point of time should click ‘y’ or ‘n’ whenever asked about the new requests. I assumed it will be very inconvenient for the judge to do this at every time step, hence I have provided the carroutingreq.cpp.

4. Requestsfr.json’ is the file which stores all the requests. The c++ programs pull all the data from this file.
5. Line number 58 in both programs will have to be changed based on the folder location in the computer.
6. The program should not be fed with out of bound data i.e if the city has 10 x 10 streets, then no start or end position should be out of these limits.
7. #include "json/value.h", #include "json/json.h", #include <json/writer.h> are the major headers for reading and writing in the JSON file.
8. For the current set of initial requests in the JSON file, at least 12 x 12 city limits are expected.