CSC263 Project Part 1

Arun Jolly Marlier, Eric Hasegawa, Faustine Leung

July 18 2020

Real Life Situation

For our project, we will be addressing the problem of assigning a delivery personnel within a food delivery service, similar to Uber Eats or Door Dash. We want to store the data in a manner that is accessible and concise to make matching a restaurant and a deliverer easier.

The main data that we will be accounting for is the locations of every car and restaurant in a neighbourhood.

Assumptions:
- Only cars will be delivering food.
- When not delivering food, all cars are parked and stationary.
- There is only one deliverer per order.

Stored Data

The ADT that we have chosen to model this problem is a weighted directed graph. Each node in the graph will represent one of:

1. A Restaurant
2. A Car

Every Car node will contain a weighted path directed at every Restaurant node with each weight being the estimated time to travel for each journey in minutes. These travel times will be stored in an adjacency matrix, as is standard with weighted directed graphs.

Operations

In terms of operations our ADT will include:

- addCar(c): This will add a node representing Car c's location (in coordinates) to our graph, and will create weighted directed edges from the new node to all existing restaurant nodes, with each estimated travel time being a weight. This operation will be called when a deliverer is available, whether it be when (s)he starts his/her shift or when (s)he is finished a delivery.

- removeCar(c): This will remove the Car c node from the graph, as well as all associated edges. This operation will be called when a deliverer ends his/her shift or when a deliverer has already accepted another delivery.

- addRestaurant(r): This will add a node representing Restaurant r's location (in coordinates) to our graph, and will create new weighted directed edges from every Car location to the new node, with each estimated travel time being a weight. This operation will be called when a restaurant opens for the day.

- removeRestaurant(r): This will remove the Restaurant r node from the graph, as well as all associated edges. This operation will be called when a restaurant closes for the day.

- getCars(): This will return a list of all the coordinates of the car nodes in the graph.

- getRestaurants(): This will return a list of all the coordinates of the restaurant nodes in the graph.

- findDeliverer(): This is the main operation for solving our problem. At the point of a customer ordering food, this operation will iterate through all Car nodes and will find the smallest value x, where,

$$x = \text{abs(Weight of edge from chosen Car node to Restaurant node) -} \\ \text{(Estimated cooking time for food)).}$$

This is to minimize the waiting time and to get the food to the customer as quickly as possible.