Final Project: Waste Collection Routing Optimization for Smart Cities

In this project you will develop a waste collection routing optimisation system for a simulated smart city environment using graphs.

Waste management is a growing problem in urban areas globally, especially in developing countries where the manual, disorganized waste collection systems are unable to scale with population growth and increasing landfill. The burden on municipal waste management authorities has rapidly increased with the spread of urban areas and overall increase in waste. Waste collection systems often fail to keep up and are unable to collect waste timely, resulting in overflowing garbage bins. This in turn leads to improperly disposed garbage, which results in health hazards like air pollution and spread of diseases via parasites and contaminated water. This cycle of disease and pollution can be stopped if waste is collected well in time with a cost-efficient smart solution.

Current collection systems are manual and rely on manual inspection of bins by staff, which has a high cost in terms of human labour and salaries, or else rigid schedules that do not account for daily variation in waste levels. Thus, the authorities do not actually have a regularly updated data source for which bins are full daily, and neither is there any historic data maintained that can point to continuously spilling bins that can indicate a need for installing more bins in some locations or changing the collection schedule to a higher frequency.

In this project you will simulate a smart city that is equipped with smart, IoT-connected garbage bins. These bins have sensors that record hourly readings of their fill-level and transmit the data to a waste management control center. Based on this data, the control center schedules dispatch of waste trucks to collect garbage from bins that are close to full. Importantly, the control center decides the best route for the trucks to take, given the locations of the full bins.

Prerequisite: Data creation

1. You are required to create a map using actual data from Google Maps (you can use locations of Islamabad) and place **50 bins** on various locations on the map, as well as **5 garbage dumping locations** (where the waste is to be dumped by trucks after collection) and **5 different truck-departure locations**.
2. You should get **real data** from Google Maps about the travel time between all the bin and dumping locations (you can get this data once and then assume that it is constant over the time of the simulation). This data will be used to build your graph (essentially you have a 56-vertex graph that is fully connected, i.e. every vertex has a path to every other vertex, and the weight of the edges is the travel time you get from Google Maps).
3. Also create a dataset that represents **hourly** (random) bin fill levels for all the bins, for a period of **3 days**.

Save all of the above datasets as a file or multiple files that will be retrieved when you run the simulation. Data should be in human readable format and will be checked during the demo.

Phase 1:

In this phase you will first build a graph from the dataset files. Next you will run a simulation over the bin fill dataset. The workflow is as follows. A user logs in as either the controller or a truck driver. If the user is a controller, he can view the bin fill data for the past hour (randomly pick a time from the 3 days of data you generated). Based on the data, he can set a collection threshold (say 75%, so all bins that are above 75% fill level will be considered collectible). The controller then selects a truck departure location. The trucks in one departure location are responsible for a certain section of the map (you will decide which – essentially each truck is responsible for 10 bins on the map, as there are 5 departure locations). Now you will generate a new graph comprising only of the collectible bins from the last hour from the given section of the map (say, only the bins in F sector that are collectible). Now you run a shortest-path algorithm (specifically you will implement Djikstra's algorithm) over the graph to decide the best possible route for collections. That is, use Djikstra's algorithm to find the most optimised (shortest) route from the source to destination while collecting all the bins.

You will output this route to the controller who will then allocate a truck to collect the bins in this hour. The route will look like, Bin A -> Bin C …. -> dumping point, where A and C are real locations (you will output real location names, e.g. Street 45 F-10/3 Bin), that come from your map data. The controller should allocate trucks from all 5 locations so that bins can be collected from the whole city every day.

If a truck driver logs in, he can see whether his truck is the one that is nominated by the controller to collect bins in that hour, as well as his route. That is all the truck driver can do in this phase. We will add more functionality for the truck driver in Phase 2.

You need to build a good user interface that allows controllers and truck drivers to register and log in, and then to schedule a collection or view their allocated route respectively. You should display the time of collection (and randomly retrieve an hour's data against that time) – then mark that data as used and use a different hour's data next time the simulation is run. The controller should be able to schedule multiple collections in a loop (e.g. one by one allocate trucks to all sections of the map).

The details that are not specified are left open to you to specify as you see fit.

Phase 2:

In phase 2 you will add another layer of complexity to your basic optimisation by adding live tracking and dynamic path changes. Now during your simulation, once the controller has allocated a truck to a route, the controller can monitor which bins the truck has already collected, and which are remaining. You can use a sleep function to simulate the truck slowly picking up bins. Now allow bins to communicate fill levels at run time – so suppose the controller has set the collection threshold to 75%, and 10 bins were collectible according to this threshold when he scheduled the collection. Now in Phase 2 of the project, a bin is also able to send an alert to the controller when it reaches the collectible threshold – so instead

of fetching bin fill levels from a static data file, you should keep checking whether new bins have become collectible any time that a controller is logged in. If a new bin becomes full, the controller should be alerted. Now he can check on the live tracking screen which truck is responsible for collecting this bin and dynamically re-calculate the shortest path for its remaining collection. This means generating a new graph based on the remaining (original) collectible bins plus the newly collectible bin and then calculating optimised path from the truck's current location to the departure point while collecting all the bins. In this phase, the truck driver can also

The output and UI required for this phase is the same as phase 1 **plus** the following:

1. Controller can now see dispatched trucks and view the live tracking of each currently dispatched truck – for example as follows:
   [Truck 1 -> Bin A collected at 12:00 pm -> Bin B collected at 12.13 p.m ->Bin C currently collecting. Remaining: Bin D, Bin F, Bin H.]
   [Truck 2 -> Collected bins E, G and I. Dumped at 12.14 p.m.]
2. Controller receives an alert if another bin becomes collectible while other trucks are out. Controller can add this bin to a dispatched truck's route. The truck's route should be changed and a message sent to the truck driver.
3. A truck driver who is logged in can see (1) his allocated route (2) his own live tracking (3) receive additional collectible bin messages asynchronously and (4) view a dynamically updated route.

Requirements:

You must implement a heap for the priority queue used internally by Djikstra's algorithm. You cannot use **any** libraries for any data structures you need for your code – code your own implementation of the heap, graph and Djikstra's algorithm. You may use Vectors in place of arrays however, if you find them more efficient. Any additional libraries you wish to use you must get approved by your instructor first.

Some details in the project are not specified – you can specify these for your own project as you see fit. Grading may be relative to some degree, i.e. better interfaces and more intricate and efficient logic will get more marks even if basic functionality is the same. Bonus marks are also possible for outstanding projects.

Submission:

Your project will be marked through a demo. The demo will be of Phase 2, as that is built on top of Phase 1. But you must submit Phase 1 separately on its own deadline (deadlines are on the Google Classroom submission) to be able to receive a demo slot for Phase 2. If you do not submit Phase 2, you will not receive a demo slot – you will just receive some marks for submitting Phase 1.

Submit all your code and data files as a single zip with your group ID as groupID_finalproject.zip. You will get a group ID from the group registration sheet uploaded separately.

This project should be done in groups of minimum two and maximum 4 members.