OOP 21 - Assignment Ex1

Python and offline algorithm

Authors

Raz Gavrieli 209380922 | Amit Melamed 316329069 | Eran David 207640806

We begin by defining the biggest difference between the Online and Offline algorithm.
At the time of solving the Online algorithm we have taken for granted the function that calculates the Elevator's position.

In this assignment we have explored the idea of writing our own algorithm that can foresee the position of a given elevator and a list of calls that it had to satisfy.
The idea of this kind of solution would involve Dynamic Programming, using matrices to hold the value of time that the elevator would get to a specific floor. And that way calculating when the elevator would get to every floor at any given time.

On the other hand, we had another more elegant solution that would divide the calls to all the elevators, trying to scatter the calls in an even manner. When we choose this route, we noticed that it is important to divide the calls to groups of time zones, meaning that a call that has taken place at the beginning of the day won't affect the decision for a call that is taking place at the middle of the day, much later.
This solution is **also** using the position of the elevator, trying to update it as the code runs. But (and this is taken into account) the position is estimated and not exact.

Taking inspiration from the first assignment, we've implemented the idea of a callsQueue for every elevator. That way we can have group-control and have a more efficient solution.

Explaining Offline Algorithm

First we've created two new objects, Call and Elevator.

Class Elevator -
This object holds all the data given from the building through the json file.
It also holds a list named callsQueue, so each elevator has its own queue of calls.

clearCompleteCalls -
If the 'current' time + INTERVAL is bigger than the call's time,
delete this call from callsQueue.

Class Call -
This object holds all the data given from the csv file.
It also holds information about the distance (in floors) of the call, and the allocated elevators for this call.
The allocatedElevator field is used to write into the output which elevator has been allocated to each call.

calcTime -
This function calculates the estimated time for the elevator to arrive to the source floor and then to the destination floor.

Eventually the time is multiplied by the amount of calls waiting in the elevator's callsQueue. To simulate the amount of stops the elevator has to take in the path.

The returned value is not exact, but it does not matter because we use this value only to **compare** between elevators.

Main -
We will divide the explanation into two parts:

1. **Receiving the input and saving the output -**

In the main code, the first thing we do is to get the input and read from the given files.
Then we create two lists that will be used later in the allocating algorithm:
elevators[] - is the list that holds all the elevators.
callsArr[] - is the list of calls that we will need to satisfy.

For each call in callsArr
Initiate the function allocateElevator for this call. #this initiates the main algorithm#

After running the main algorithm on all the calls, we write the new data into an array and we save this array in output.csv

2. **Main allocating algorithm -**

allocateElevator -

#this is the main allocating algorithm

First, decide which algorithm should work on this call.
If the building has only 2 elevators
    Use allocateElevatorFor2Elevators
If the building has less than 6 elevators
    Use allocateElevatorMedium
#else use the following algorithm:

For each elevator
    Calculate it's calcTime (As explained above)

Take the minimum clacTime and:
Change call.allocatedElevator to the chosen elevator.
Append the call to the elevator's calls queue.
For each elevator update it's callsQueue (clearCompleteCalls)

allocateElevatorMedium -

Same as the main algorithm only uses different values for the function calcTime.

allocateElevatorFor2Elevators -

Check which elevator is faster.
If the call is in the upper two thirds of the building, use the faster elevator.
Else (the call is in the lower third of the building), use the slower elevator.
This method does not use calcTime nor callsQueue.

Documentation

Performance:

The data is wrriten as |average waiting time/uncompleted calls|				
Building\Calls	A	B	C	D
1	112.92/0	-	-	-
2	44.67/0	-	-	-
3	44.25/0	485.032/140	523.140/65	491.445/84
4	24.92/0	205.967/8	194.849/8	204.922/15
5	12.84/0	44.305/0	43.389/0	46.476/0

UML Diagram: