**Problem Statement:**

Implement an algorithm for an elevator scheduling system that can be configured for a building that has ‘N’ Floors and ‘M’ elevators. The objective of this algorithm is to minimize the wait + travel time for any person requesting for a lift.

Please highlight the data structures utilized in the algorithm and how they optimize the wait times when the person requests for an elevator from say Floor ‘A’ to Floor ‘B’.

You can assume that the time for an elevator to stop at a floor will be ‘5’ second and to travel each floor (without stopping) will be ‘1’ second. Please write logger statements that will display the total amount of wait/travel time for a person once the transaction is complete.

Write a test case which simulates the people interacting with this elevator system. This test case can simulate the people requesting an elevator at different floors at different times (not all will request at the start of the elevator system).

**Assumptions:**

* Each person is requesting for himself
* Each person is requesting only once
* Time taken to move from one floor to its neighbour is 1 unit.
* Time taken for opening and closing of door is 5 unit.
* Number of floors is greater than numbers of lifts.
* Lifts has no capacity limits.

**Approach:**

For this lift Elevator problem, there is no fixed solution. Its all depends on the input to the system. The approach used here is a variation of sector or Zone approach. First, the solution will first divide the lifts on the basis of the floor they can stop. 70% of the list will be divided into different zones all having the common floor as 0. Other 30% of lift will stop on all the floor.

While(hashmap having request is not null || time<=86400)

* For every second keep checking the floors for requests.
* If any request is there.
* Check which lift can process it.
* Now check for the state of the lift and current floor
* If the state is 0 i.e idle

1. If source and current floor of list is same, then process it.
2. If Source >current, then set state as 1 and set movingtoServe as 1.
3. Source<current, , then set state as 2 and set movingtoServe as 1.

* If the state is 1 i.e moving up

1. If source and current are equal and source < destination then add it to hashmap.
2. If source and current are equal and source > destination then leave it.
3. If source and current are equal and lift is empty then process it and set the direction as per the process.

* If the state is 2 i.e moving down

1. If source and current are equal and source > destination then add it to hashmap.
2. If source and current are equal and source < destination then leave it.
3. If source and current are equal and lift is empty then process it add it to hashmap.

* Now for every lift check if any person in the lift has its destination on the current floor. If so then onboard them.
* Now for every lift, If it is stopped then decrease the stoppage time.
* If it is Empty and movingtoServe is not 1 then change the state to idle.
* If state is 1 and stoppage time is not zero then floor+1 and if stoppage time is not 0 then decrease the stoppage time by 1.
* If state is 2 and stoppage time is not zero then floor-1 and if stoppage time is not 0 then decrease the stoppage time by 1.

**Used Data Structure:**

Class : Person

class Variables :

* waittime (waiting time for a person)
* traveltime (travelling time for a person)
* arrivaltime (arrival time for person)
* personID (specific id for a person)
* Source (Floor from the person has made a request)
* Destination (Floor at which the person has to go)

Class : Lift

class Variables :

* Floor (Lift position at a specific time)
* direction (Direction in which lift is moving----0 for idle, 1 for Up, 2 for down)
* stoppagetime (Time at which the lift will take to open and close)
* movingtoServe (value will be 1 if a empty list has been making a move to process the person on some other floor)
* ArrayList<Persons> persons (contain the persons currently on the lift)
* ArrayList<Integers> stop (conatains where the lift can stop)

Class Algo:

Methods:

* void createPartition(int nol,int nof, HashMap<Integer,Lift> lift)

To partition the lifts on the basis of floors they will stop

* HashMap<Integer,ArrayList<Person>> storePersonCallFloorWise(int arr[][],int nop)

Will create a hashmap storing floor wise persons calls

Input:

First line will contain 3 space separated Integer which defines number of floor n, no. of lifts m, no. of persons p respectively. Next p lines will contains 3 space separated integers defining source floor, destination floor, time of arrival(in seconds).

Note: Lift count will start from 0

Sample Input:

5 3 3

1 2 3

2 3 4

3 4 1

Sample Output:

Lift 1 will stop on [0, 1, 2]

Lift 2 will stop on [3, 4, 0]

Lift 3 will stop on [0, 1, 2, 3, 4]

------------------------------------Person in the order they picked Up----------------------------

------------------------------------------------------------------------------------------

Person ID :1

Waiting time =1

Travel time =6

------------------------------------------------------------------------------------------

Person ID :2

Waiting time =0

Travel time =6

------------------------------------------------------------------------------------------

Person ID :3

Waiting time =3

Travel time =6

**Benefits :**

* As the floors of the lifts is partitioned, so lift need not to stop on the all the floors.
* Ground floor is common for all the lifts. So, groun floor can be reached by any lift.
* As 30% of lifts can move on all the floors. So, no floor are left unreached.
* Partiion further can be done on requirement basis, like as if on some floor a cafertia or restaurant is available.
* Scenarios in which a particular source and destination is used again and again, there the waiting and processing will be minimized.

**Area of Improvement:**

* Capacity of the lift can also be used as a parameter.
* On the basis of type of building and the current time.

For Offices : In morning time most of the employees will go up. So all the idle can be made to rest on the ground floor.

In Evening time most of the employee will go home, So all the idle can be made to rest on the top floor.