Elevator Project:

Aj

Alex

Michael

Overview:

Our basic algorithm is a replication of Karp's algorithm. The key to the algorithm is not to treat the people already in the elevator as any different to the people waiting for the lift. The algorithm works with an up and a down mode of operation. You also need to keep track of the number of people wanting to move higher than a given floor and those wanting to move lower, we use separate lists for both. These multiple lists are a complicated piece of book-keeping but it is what makes the algorithm work, so we can't avoid it. On each floor k you need to count Uk the number of people on that floor and below it who want to go higher than that floor. Thus, if there is one person on each floor wanting to go to the floors as indicated, then Uk for floor 3 is 1 because on floor 3 and below there is only one person wanting to go higher than floor 3. For floor 2 Uk is 2 because there are two people wanting to go higher than floor 2 who are on floor 2 or below. We can apply this logic to create multiple elevators by getting the current floor for each elevator and calculating how many people are in the direction the elevator is traveling both in and out of the elevator. This algorithm optimizes the movement of the elevator. We use lists to store the data to optimize how our program runs because lists allow us to insert new users with a big-O of 1. We do sort the lists initially but as more passengers are spawned and more floors are available our algorithm works great!

Assumptions:

This project was fun and exciting to work on. The project was complex and required lots of work from every individual. To affectively code the solution we had to make multiple assumptions. The simulation code was massively important and required assumptions about the input type for the number of floors for the simulator, if the commands for the number of floors and rate of spawns were inputted as a string the code would not work because the elevator's passenger rate and location of passengers are generated from this input from the simulator class. The elevator's process function makes the assumption that the elevator will not have any emergencies. We also assumed that for every passenger that called the elevator they rode the elevator. This implies that one call represents one person, not a group. The elevator also assumes that no user gets in or out if the elevator is going the opposite direction, even if the door opens on the same floor. With our simulation we keep track of everyone's movement and time but we assume that once they leave the elevator that there is no need to prioritize the elevators default location based on the location of people who have been dropped off in the building. Every time we iterate the turns of the passengers and when they reach their destinations we get the turns they waited and add that to the total turns that have occurred and then divided that by the number of passengers that were spawned to get the average wait time.

UML diagram:

Big-O:

We were really smart with our algorithm and thus our big-O. By incorporating lists we always keep our destinations and pickups sorted so iterations to drop people off have the lowest big-O possible.

list.sort uses stable sort = nLogn

Passenger::addTurn() = Adds one to turns each turn O = 1

Elevator::goToFloor() = inputs the floor destinations in a sorted fashion O = 1

Elevator:: Called() = checks the direction of the object plus iterates through a list using checkUkUp O = n

Elevator::checkUkUp() & checkdown() = iterate through the directional list O = n

Elevator::queuePush() = Pushes the passenger’s input to the elevator queue O = 1

Elevator::process() = Checks multiple lists to see if they are empty and calls the correct sorting method but does not preform iterations since the lists are always sorted O = 4

Simulation::spawnPassenger() = creates a random integer that then has a chance to spawn a new passenger and make the elevator call for them O = 1

Simulation::simulate() = reads out what is happening to the passengers and increments the turn O = 1

References:

Karps algorithm = https://viblo.asia/p/sharpen-your-coding-skills-elevatorpuzzle-3KbvZq8zGmWB

Knuth, Donald, The Art Of Computer Programming. Vol 3, pp 357-360.”One tape sorting”.

Airline simulation