Ex1

Names of the submitters: Yan Naigebaver, Ron Shvindelman.

IDs: 323906842, 208232652

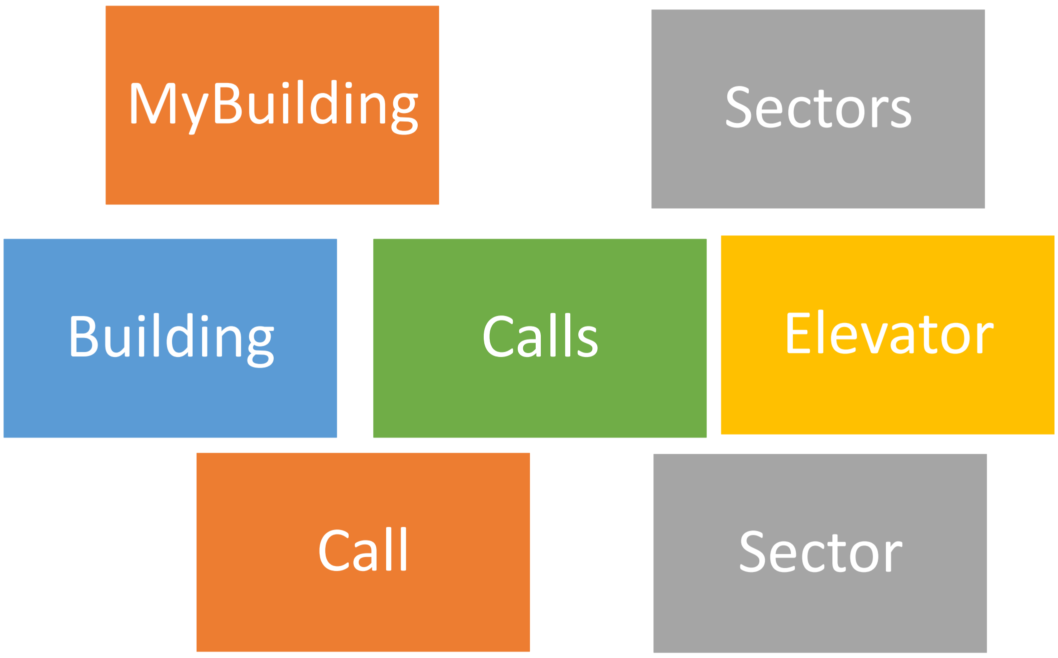
Understanding the data

The given files are a json file of a building containing number of elevators. Each elevator has several attributes. Such as speed, time to open, time to stop, time to close, time to start. which should be concerned when implementing the algorithm because those have direct effect on the average waiting time. Another given file is the Calls_x.csv which represents a list of calls. Each call in the list has attributes such as source of the call, destination of the call, time called. These attributes are important to analyze to know exactly which elevators to assign in order to minimalize the average waiting time.

Intuition for the Algorithm

The algorithm supposed to organize and divide the list of calls into groups of calls to certain sectors. Choosing the number of sectors depends on how many elevators there are in the building. Each sector would be under the control of a specific elevator. The elevator is selected after analyzing the list of calls and showing the clear distribution of calls into the sections. The properties of the elevator should match the need of the sector. For example, a sector with a lot of calls should be matched with a fast-working elevator rather than a slow one. Although having a single elevator to a sector might create problems when the sector is spammed by calls. To solve this problem, we have created a special case at each call of allocation. We evaluate whether the elevator is being spammed and if so then send the first elevator which is not spammed which handles the closest sector.

The classes used to implement this idea



Description of the classes

Call- Basic representation of the call contains 3 attributes, time called, source floor, destination floor.

Sector- Basic representation of the range of a sector as described, contains 2 attributes, the lower floor of the sector and the upper floor of the sector.

Elevator- Basic representation of the elevator, contains all the attributes as described.

Building- The building constructor receives the .json file of the building and converts it to an object with certain attributes of a building such as Elevator list, minimum floor of the building and the maximum floor of the building.

Calls- The Calls constructor receives the .csv file of the list of calls and converts it to a list of Call.

Sectors- The Sectors constructor receives Calls, and the number of elevators in the building. The number of floors for a Call list is determined as taking the difference between the maximal call and minimal call. The Sectors class sets the sectors by taking the number of floors and dividing by the number of elevators. The Sectors are being sorted by the amount of calls each sector receives using quicksort.

MyBuilding- MyBuilding constructor receives Calls, Building, Sectors. The purpose of this class is to link between all the classes created and give a proper algorithm which solves the off-line allocation problem. Finally, this class creates an Ex1_out.csv file which has ordered answered calls to specific building and calls list.

Testers used

In some cases where it is harder to officially create cases, we've used more of a visual kind of testing. printing the To-Strings of Sectors for example.

Test_Sectors – contains functions to test how the sectors were set, and how the sectors were sorted.

Test_Building – contains function to test for the elevators were sorted.

Bibliography

Round trip time - https://www.researchgate.net/publication/330447836_Calculation_of_the_Elevator_Round-Trip_Time_under_Destination_Group_Control_using_Offline_Batch_Allocations_and_Real-Time_Allocations

אלגוריתם otis - https://elevation.fandom.com/wiki/Otis_Elevoice_(floor_announcement)

אלגו מעליות - https://elevation.fandom.com/wiki/Elevator_algorithm

Destination Dispatch - https://elevation.fandom.com/wiki/Destination_dispatch

online vs offline algo - https://www1.icsi.berkeley.edu/pubs/techreports/TR-92-044.pdf