Course Project

CMPS323 - Design and Analysis of Algorithms

Dr. Mohammad Saleh

Eng. Alaa Hussein

Mohd Muhtasim Bashar – 202205153

Mohd Ridhwanul Hoque – 202204363

# Proposed Solution

This section may include the solution description/implementation details:

Explain the chosen approach (greedy or dynamic programming) for optimizing elevator operations.

We have chosen the ‘Greedy Approach’ to find the floors to stop with the least walking distance at any given k stops.

Describe the algorithm or methodology in detail.

Greedy algorithm consists of 3 main steps, (1) Selection Procedure, (2) Feasibility check and (3) Solution check. Our approach in the beginning is to find the costs (total walking distance) of stopping at each floor. Then select (1) the floor with the minimum distance. This selection is a greedy choice. Next, it will check (2) whether this floor has already been visited or not or is it in the proximity (+1 or -1 floor) of a current solution. Lastly, it will keep finding floors until a solution case is reached.

Discuss how the chosen approach addresses the constraints and objectives of the problem statement.

The constraint of this implementation is it will consider costs of all passengers in each floor. Also, in this case, we do not have the information of which passengers will get off at each floor, assuming they need to decide themselves.

# Solution

Provide details of the implementation of the proposed solution.

Present any data structures or algorithms specific to the implementation.

We have implemented our program using Arrays and Array Lists.

Tools and libraries used

import java.util.ArrayList;

import java.util.Collections; (To sort arrayLists)

Source code.

package v1;

import java.util.ArrayList;

import java.util.Collections;

public class GreedyOptimizer {

// returns the minimum value in an array

public static int MinArray(int[] arr) {

if (arr == null || arr.length == 0) {

throw new IllegalArgumentException("Array must not be empty or null.");

}

int min = arr[0]; // Assume the first element is the minimum

for (int i = 1; i < arr.length; i++) {

if (arr[i] < min) {

min = arr[i]; // Update minimum if a smaller value is found, except -1

}

}

return min;

}

// Method to find the index of a value in an array

public static int findIndex(int[] arr, int value) {

for (int i = 0; i < arr.length; i++) {

if (arr[i] == value) {

return i; // Return the index if the value is found

}

}

return -1; // Return -1 if the value is not found in the array

}

public static void main(String[] args) {

// **TODO** Auto-generated method stub

int k = 2; // No. of stops

int[] destination = {1, 2, 4, 7, 9, 13}; // Destination of the lift, where should it stop

int floors = 15;

int[] cost = new int[floors]; // The cost of stopping at each floor, indexes represents floor

int pax = destination.length; // No. of passengers

int minFloor; //Floor no. having the minimum walking cost

int temp = 0;

ArrayList<Integer> visit = new ArrayList<>(); // Floors to visit

// Check if k stops equals destination floors, then stop at the requested floors

if (k == destination.length) {

for (int a = 0; a < destination.length; a++) {

visit.add(destination[a]);

}

//Print the solution

System.***out***.println("\nSolution: ");

Collections.*sort*(visit);

for (Integer stop : visit) {

System.***out***.println(stop);

}

} else {

// Calculating the cost of stopping at each floor

System.***out***.println("Costs of stopping at each floor: ");

for (int i = 1; i <= floors; i++) {

temp = 0;

for (int j = 0; j < pax; j++) {

temp += Math.*abs*(destination[j] - i); //Distance of stairs between the destination floor and the current floor

}

System.***out***.println("Cost at floor " + i + " :" + temp);

cost[i - 1] = temp; //Total cost of all passenger for stopping at floor i

}

//Greedy approach

do {

// (1) Selection Procedure: Select the floor with the minimum walking distance (cost)

minFloor = *findIndex*(cost, *MinArray*(cost));

// (2)Feasibility check: Check if the selected floor is in close proximity to another floor

if (visit.indexOf(minFloor) == -1 && visit.indexOf(minFloor - 1) == -1 && visit.indexOf(minFloor + 1) == -1) {

visit.add(minFloor);

}

cost[minFloor] = 999; // Flag, saying that the floor has been visited

// (3) Solution Check

} while (visit.size() != k); //Check if k floors for solution has been found

//Print solution

System.***out***.println("\nSolution: ");

Collections.*sort*(visit);

for (Integer stop : visit) {

System.***out***.println(stop + 1); // To remove floor 0 (index 0) from consideration

}

}

}

}

# Experimental Results and Screenshots

Present the experimental setup used to evaluate the proposed solution.

int k = 3; // No. of stops

int[] destination = {1,2,4,7,9,13}; // Destination of the lift, where should it stop

int floors = 15;

Provide quantitative and qualitative results of the optimization performance.

Solution:

4

6

8

Costs of stopping at each floor:

Cost at floor 1 :30

Cost at floor 2 :26

Cost at floor 3 :24

Cost at floor 4 :22

Cost at floor 5 :22

Cost at floor 6 :22

Cost at floor 7 :22

Cost at floor 8 :24

Cost at floor 9 :26

Cost at floor 10 :30

Cost at floor 11 :34

Cost at floor 12 :38

Cost at floor 13 :42

Cost at floor 14 :48

Cost at floor 15 :54

Include analysis and interpretation of the results, such as efficiency metrics.

# Discussion

Interpret the results in the context of the problem statement and objectives.

Discuss the strengths and limitations of the proposed solution.

Identify potential areas for improvement or future research.

# Conclusion

In conclusion, we have implemented the greedy approach to solve this elevator optimization problem, the task was to minimize the walking distance for the riding passengers should they have to take the stairs in case the elevator does not stop at the level the passenger(s) wanted. The algorithm we have implemented gives us satisfying results in the context that the elevator makes a number of stops as directed by the elevator management system and stops at the best levels that favor both the passengers in terms of least walking cost and also in terms of the electricity cost that accumulates given that the elevator will always stop at the lower floor when faced with ties between solutions of equal cost. But the solution is not perfect and there is room for improvement, such is the nature of greedy algorithms as they don’t always guarantee a globally optimal solution. Nevertheless, real world data about passenger and peak hours of elevator use can be incorporated to further improve the optimization.