Elevator algorithm

Nick, Max, Edward, Thomas

March 2025

1 Objectives, Requirements and scope

1.1 Objectives

1.1.1 Create an efficient algorithm that transports people between floors

It is imperative in this course work to design and create multiple algorithms to find the most efficient algorithm, using graphs and multiple metrics to indicate which algorithm is the best. This will help us understand how lifts work and how they decide which route to take to efficiently move people between floors.

1.2 Scope of coursework

- An algorithm that to interpret input files
- An algorithm to transport people between floors
- A GUI that visualises the movement of the lift, clearly indicating what is on the lift and when it stops

1.3 Requirements

- An algorithm that converts an input file into valid JSON
- An algorithm that checks that the input file is valid and if it is not describe how it should be corrected
- At least two algorithms that take all passengers from their floors to their requested floors in a valid manner
- Each algorithm must display its stops and show who was picked up and dropped off at each floor
- Each algorithm takes into account capacity of the lift making sure that the lift only takes on the maximum amount of passengers at a time
- A chart showing the efficiency of at least two of the algorithms

- A GUI that visualises the movement of the lift, clearly indicating what is on the lift and when it stops

2 What did each of us do

2.1 Max

- algorithm.py The core elevator logic handling movement and passenger management through the Algorithm class, with run_algorithm processing floor requests and returning simulation results
- run_algorithm.py The main execution script that loads configurations and runs the elevator simulation with result formatting
- generator.py Creates random test scripts for the elevator simulation with configurable parameters like floor count and capacity.

2.2 Thomas

- I created the front end of the application, using Flask as a web server and a HTML templating engine. This involved writing the processing functions and the HTML, CSS and JavaScript for each individual web page.
- I created the communication between the various back end elements and the front end. This includes handling how data (e.g. configuration files/dictionaries) is passed between the user input and the algorithms, choosing which algorithm to use, implementing those files and passing the output to the animation.
- I created the animation to visually represent what each different algorithm is doing. This involved using JavaScript to dynamically style HTML components based on the movements the elevator took.
- I wrote multiple functions to handle inputting and storing configuration files for using pre-made configs. Functionality includes loading from files, saving new configs to files, translating between .txt and .json based configuration, and implementing the validity and type checking functions written by Nick.

2.3 Nick

- Specification file: wrote and edited the specification file for the coursework
- Pseudo code: wrote and created the pseudo code for each algorithm
- Data analysis: analysed the data to work our how well each algorithm performed

2.4 Edward

- Algorithm.py: a python file that handles the elevator movements and logic, as well as including a text simulation through print statements when in debug mode
- AlgorithmAlternate.py: an alternate version of Algorithm.py, which completes requests in less movements with an additional assumption that people only get on the elevator if it's going in the same direction they want to go. This also includes a text simulation in debug mode

3 Design

3.1 System Requirements

3.1.1 Language

Python 3.10+

3.1.2 Python packages

- flask

3.2 Nick's design of the scan and look algorithm

3.2.1 Scan algorithm

The scan algorithm works by moving in a set direction until it reaches the end of the lift, at which point it will turn around and go in the other direction, picking up and dropping of people along the way.

The algorithm first starts by checking if there are any people who need to get off at the current floor, if there are then it drops people off at that floor.

Then the algorithm checks if there are any people who want to get on the lift at the current floor, while there is still space the algorithm will take people from that floor onto the lift.

Next the algorithm checks if the lift is at the top or the bottom of the elevator, if it is then it switches to the opposite direction

This then repeats until there is no longer any people left in the lift or waiting for the lift

3.2.2 Look algorithm

The look algorithm works by moving in a set direction until it reaches the end of the lift or if there is no more calls in the current direction, at which point it will turn around and go in the other direction, picking up and dropping of people along the way.

The algorithm first starts by checking if there are any people who need to get off at the current floor, if there are then it drops people off at that floor.

Then the algorithm checks if there are any people who want to get on the lift at the current floor, while there is still space the algorithm will take people from that floor onto the lift.

Next the algorithm checks if the lift is at the top or the bottom of the elevator, if it is then it switches to the opposite direction

Then the algorithm checks if there are still calls in the current direction, if not then it will change direction as long as there is people in the lift

This then repeats until there is no longer any people left in the lift or waiting for the lift

3.3 Edward's proposed algorithm

This first section will try to explain how the elevator in theory should function in terms of how it determines which direction to prioritise, and what to give priority to.

The first part explains the order of priority The second part explains when the elevator opens or doesn't open a floor Third part explains how to use Fourth part explains what I believe my algorithm currently achieves Fifth part explains what I plan to implement

The order of priority:

1. People on the elevator wanting to travel in the same direction the elevator's already going

2. People on the elevator wanting to travel against the direction the elevator's currently going.

3. People off the elevator and calling the elevator

In the case of two conflicting directions that both have the same priority, then the shortest distance is favoured. If both distances are equal then the distance travelling downwards would be favoured, however this can be changed.

This is based off the idea that once you're on the elevator, you don't expect to be dragged around due to other people that called the elevator, if you were to call an elevator with the down button, you wouldn't expect to go up.

The reason the elevator prioritises the direction it is already travelling is to stop a risk of the elevator ever getting stuck anywhere, imagine the elevator switching between floor 2 and 3 indefinitely - if the elevator only prioritised the closest floor then anybody wanting to get to a further floor would end up stuck.

When the elevator stops at a floor:

If someone on the elevator has chosen to leave at a certain floor, the elevator will stop at that floor.

If someone waiting at the floor has called the elevator in a certain direction that matches the current direction of the elevator.

The reason the elevator doesn't open its doors if someone wants to go in the opposite direction is because they would end up stuck on the elevator until it reaches its final destination and changes directions, at which point the elevator could've picked the person up on the way back instead.

4 Pseudocode

4.1 Nick's pseudocode of the scan and look algorithm

4.1.1 Scan algorithm

```
set current floor to 1
set direction to -1
set the last stop to the current floor

set the lift to an empty list
set the stops to an empty list
set movements to an empty list
set amount leaving to an empty lift

while there is any floor requests left or people in the lift:
    set stopping to false
    if the current floor is a request in the lift
        set stopping to true
        add the amount leaving at this floor to the amount leaving list
        remove every request to stop at current floor from the lift
    if there is any people waiting to get on the lift at the current floor and there is spac
        set stopping to true
    while there is any people waiting to get on the lift at the current floor and there is s
        add the first floor request at the current floor
        remove the first floor request from the config
    if the current floor is 1 or the current floor is the floor count
        times the direction by negative 1
    if stopping is true:
        add the current floors to the stops
        add a tuple of (last stop, current floor) to the movement list
        set the last stop to the current floor
        print the stop
    add the direction to the current floor
```

4.1.2 Look algorithm

```
set current floor to 1
set direction to -1
set the last stop to the current floor

set the lift to an empty list
set the stops to an empty list
set movements to an empty list
```

```
set amount leaving to an empty lift

while there is any floor requests left or people in the lift:
    set stopping to false
    if the current floor is a request in the lift
        set stopping to true
        add the amount leaving at this floor to the amount leaving list
        remove every request to stop at current floor from the lift
    if there is any people waiting to get on the lift at the current floor and there is spac
        set stopping to true
    while there is any people waiting to get on the lift at the current floor and there is s
        add the first floor request at the current floor
        remove the first floor request from the config
    if the current floor is 1 or the current floor is the floor count
        times the direciton by negative 1

    if all calls in the lift are less than the current floor and there are people in the lif
        set direction to -1
    else if all calls in the lift are greater than the current floor and there are people in
        set direction to 1

    if stopping is true:
        add the current floors to the stops
        add a tuple of (last stop, current floor) to the movement list
        set the last stop to the current floor
        print the stop
    add the direction to the current floor
```

4.2 Edward's proposed algorithm

4.2.1 Floor check function

```
function floorCheck(current floor, queued floors):
    if current floor is in queued floors:
        return true
    return false
```

4.2.2 Follow up function

```
function followup(current floor, backup queue, prior):
    if prior is up:
        if max of backup queue is greater than current floor:
            return true
    else:
        if min of backup queue is less than current floor:
```

```
            return true
    return false
```

4.2.3 Pathing function

```
function pathing(current floor, queued floors):
    set checkup and checkdown to false
    get the first floor in queued floors and store it in current floor
    while there another floor in queued floors not yet gone through:
        if next floor in queued floors is greater than current floor:
            set checkup to true
        else if the next floor in queued floors is less than current floor:
            set checkdown to true
        if checkup and checkdown are false:
            return stop
        else if checkdown is false:
            return up
        else if checkup is false:
            return down
        store the absolute difference between the current floor and max of queued floors in
        store the absolute difference between the current floor and min of queued floors in
        if the distance up is less than the distance down:
            return up
        return down
```

4.2.4 Take request function

```
function takeRequest(current floor, calls up, calls down):
    set checkup and checkdown to true
    if there are no calls up:
        set checkdown to false
    if there are no calls down:
        set checkup to false

    if checkup and checkdown is false:
        return stop
    else if checkup is false:
        return down
    else if checkdown is false:
        return up

    set finalup and finaldown to the current floor
    if the minimum of the calls up are less than the finalup:
        set final up to the minimum of the calls up
    set longest up call to the absolute difference between the current floor and the final u
    if the maximum of the calls down are less than the finaldown:
```

```
        set final down to the maximum of the calls down
    set longest down call to the absolute difference between the current floor and the final
    if the longest call up is less than the longest call down:
        return down
    return up
```

4.2.5 Queue class

```
class queue:
    function new():
        create an empty list to store items in the queue
    function size():
        return the length of the current queue
    function addItem(item):
        add the item to the end of the list
    function checkNext():
        get the first item in the list
    function removeNext():
        remove the first item in the list
```

4.2.6 Main loop

```
load the config from the file using the load_config_from_file function
set hard capacity to 20
set soft capacity to half of hard capacity
set weight count to 0
set prior to none
set direction to none
set weight decrease to 0
set weight increase to 0
set time count to 0
create an empty queue using the queue class
store all calls going up from their floor in calls up
store all calls going down from their floor in calls down
add all calls into waiting queue


if the there are less more floors than the amount specified in the config file:
    print that there is more floors that the specified amount
    exit the program

if there are any missing floors:
    add the remaining missing floors to the config
```

```
do:
    set reset to false
    set stop here to the result of floorcheck using current floor and queued floors
    set stop here up to the result of floorcheck using current floor and calls up
    set stop here down to the result of floorcheck using current floor and calls down
    if (stop here up is true and direction is up or stop here down is true and direction is
        if stop here up is true:
            remove current floor from the calls up
            set stop here up to false
        else:
            remove current floor from the calls down
            set stop here down to false
        loop for the amount of people at the current floor:
            add the floor requested of the first person to the queued floors
            add the floor requested of the first person to the queue object
            remove the floor request of the first person from queued floors
            increment weight count
            increment weight increase
        output the current floor saying that the algorithm has stopped at this floor
        output the new queued floors

    if stop here is true:
        for any floors in queued floors that is equal to current floor:
            increment weight count
        remove all occurances of current floor queued floors
        if stop here up is true:
            remove the current floor from calls up
        if stop here down is true:
            remove the current floor from calls down
        if stop here down is true or stop here up is true:
            loop for the amount of people at the current floor:
                add the floor requested of the first person to the queued floors
                add the floor requested of the first person to the queue object
                remove the floor request of the first person from queued floors
                increment weight count
                increment weight increase
        output the current floor saying that the algorithm has stopped at this floor
        output the new queued floors

    if weight count is greater than the hard capacity:
        get the weight difference between weight count and the hard capacity
        loop for the weight difference:
            add the first queued floor to the queue object
            if the first value in queued floors is greater than the current floor:
                add the current floor to calls up
            else:
```

```
            add the current floor to calls down
        drop the first value in queued floors off to the current floor
        decrement weight count
        increment weight decrease
        remove the first value from queued floors

timecount = timecount + 1
loop for the size of the queue object:
    if the next item in the queue object is in queued floors or calls up or calls down:
        break from the loop
    else:
        set time count to 0
        remove the next item from the queue object


if there is any calls up or down:
    set call check to true
else:
    set call check to false

if there are no queued floors and call check is true:
    set the direction to the result of takerequest(current floor, calls up, calls down)
    set reset to true
    if direction is up:
        set destination to the highest call down
        if the destination is lesser than the current floor:
            set x to -1
        else:
            set x to 1
        while true:
            if the current floor is the destination:
                set direction to down
                set reset to true
                break from loop
            else:
                add x to current floor
                increment movements
    else if direction is down:
        set destination to the lowest call up
            if the destination is greater than the current floor:
                set x to 1
            else:
                set x to -1
            while true:
                if the current floor is the destination:
                    set direction to up
```

```
                    set reset to true
                    break from loop
                else:
                    add x to current floor
                    increment movements

    if reset is true:
        set reset to false
    else:
        if there are no queued floors and there are no calls up or down:
            break from the loop
        if there are no queued floors and the prior direction was none
            set the current direction using the pathing function with current floors and que
        else:
            set direction to the pathing result

        if direction is up:
            increment current floor
            increment movements
        else if direction is down:
            decrement current floor
            increment movements
        else:
            break from the loop
        set prior to the current direction
        print the current floor
```

4.2.7 Alternate Main loop

```
load the config from the file using the load_config_from_file function
set hard capacity to 20
set soft capacity to half of hard capacity
set weight count to 0
set prior to none
set direction to none
set weight decrease to 0
set weight increase to 0
set time count to 0
create an empty queue using the queue class
store all calls going up from their floor in calls up
store all calls going down from their floor in calls down
add all calls into waiting queue
```

```
if the there are less more floors than the amount specified in the config file:
    print that there is more floors that the specified amount
    exit the program

if there are any missing floors:
    add the remaining missing floors to the config

do:
    set reset to false
    set stop here to the result of floorcheck using current floor and queued floors
    set stop here up to the result of floorcheck using current floor and calls up
    set stop here down to the result of floorcheck using current floor and calls down
    if (stop here up is true and direction is up or stop here down is true and direction is
        if stop here up is true and direction is up:
            remove current floor from the calls up
            set stop here up to false
        else if stop here down and direction is down:
            remove current floor from the calls down
            set stop here down to false
        loop for the amount of people at the current floor:
            add the floor requested of the first person to the queued floors
            add the floor requested of the first person to the queue object
            remove the floor request of the first person from queued floors
            increment weight count
            increment weight increase
        output the current floor saying that the algorithm has stopped at this floor
        output the new queued floors

    if stop here is true:
        for any floors in queued floors that is equal to current floor:
            increment weight count
        remove all occurances of current floor queued floors
        if stop here up is true:
            remove the current floor from calls up
        if stop here down is true:
            remove the current floor from calls down
        if stop here down is true or stop here up is true:
            loop for the amount of people at the current floor:
                add the floor requested of the first person to the queued floors
                add the floor requested of the first person to the queue object
                remove the floor request of the first person from queued floors
                increment weight count
                increment weight increase
        output the current floor saying that the algorithm has stopped at this floor
        output the new queued floors
```

```
if weight count is greater than the hard capacity:
    get the weight difference between weight count and the hard capacity
    loop for the weight difference:
        add the first queued floor to the queue object
        if the first value in queued floors is greater than the current floor:
            add the current floor to calls up
        else:
            add the current floor to calls down
        drop the first value in queued floors off to the current floor
        decrement weight count
        increment weight decrease
        remove the first value from queued floors

timecount = timecount + 1
loop for the size of the queue object:
    if the next item in the queue object is in queued floors or calls up or calls down:
        break from the loop
    else:
        set time count to 0
        remove the next item from the queue object


if there is any calls up or down:
    set call check to true
else:
    set call check to false

if there are no queued floors and call check is true:
    set the direction to the result of takerequest(current floor, calls up, calls down)
    set reset to true
    if direction is up:
        set destination to the highest call down
        if the destination is lesser than the current floor:
            set x to -1
        else:
            set x to 1
        while true:
            if the current floor is the destination:
                set direction to down
                set reset to true
                break from loop
            else:
                add x to current floor
                increment movements
    else if direction is down:
        set destination to the lowest call up
```

```
                if the destination is greater than the current floor:
                    set x to 1
                else:
                    set x to -1
                while true:
                    if the current floor is the destination:
                        set direction to up
                        set reset to true
                        break from loop
                    else:
                        add x to current floor
                        increment movements

    if reset is true:
        set reset to false
    else:
        if there are no queued floors and there are no calls up or down:
            break from the loop
        if there are no queued floors and the prior direction was none
            set the current direction using the pathing function with current floors and que
        else:
            set direction to the pathing result

        if direction is up:
            increment current floor
            increment movements
        else if direction is down:
            decrement current floor
            increment movements
        else:
            break from the loop
        set prior to the current direction
        print the current floor
```

4.3 Max's proposed algorithm

4.3.1 Algorithm class

```
class algorithm:
    function new(config):
        set total floors to the number of floors in the config
        set capacity to the capacity in the config
        set the current floor to 1
        set the queued floors to an empty list
        set direction to none
```

```
    set weight to 0
    set time elapsed to 0
    set stop history to an empty list
    set pickups to an empty list
    set dropoffs to an empty list

function floorCheck(floor):
    return true if floor is in queued floors, else return false

function pathing():
    if there is no floors in queued floors:
        return none

    store all floors above the current floor in floors above
    store all floors below the current floor in floors below

    if there and no floors above and below:
        return "none"
    else if there are no floors above:
        return down
    else if there are no floors below:
        return up

    set the up distance to the lowest floor up from the current floor
    set the down distance to the highest floor down from the current floor
    if the up distance is less or equal than the down distance:
        return up
    return down

function move():
    if floorCheck(current floor) is true:
        remove the current floor from queued floors
        add the current floor to the stop history
        add 0 to the pickups
        add 0 to the dropoffs
        print that the algorithm is stopping at the current floor
        add 5 to time elapsed

    set the direction to the result of pathing()

    if the direction is up and the current floor is less than the total floors:
        increment current floor
        add 3 to time elapsed
        return true
    else if direction is down and current floor is greater than 1:
        decrement current floor
```

```
            add 3 to the time elapsed
            return true
        return false

    function add_passenger(weight = 1):
        add weight to weight (as in the paramters weight to the objects weight)
        add 5 to time elapsed

    function remove_passenger(weight = 1):
        remove weight from weight (as in the paramters weight to the objects weight)
        add 5 to time elapsed
```

4.3.2 Run algorithm

```
function run_algorithm(requests, config):
    create an algorithm object with the config and store it in controller

    loop through each start floor and destinations in the storted requests:
        loop through each destination in destinations:
        if the start floor is not the same as the controller's current floor:
            add the start floor to the controller's queued floors
            while controller.move() returns true:
                pass

        if the controller's weight is less than the controller's capacity:
            if the controller stop history has an item and the last controllers stop history
                increment the last controller's pickups
                call add_passenger from controller's

            add the destination to the queued floors in controller

            while controller.move() returns true:
                pass

            if the controller stop history has an item and the last controllers stop history
                increment the last controller's pickups
                call remove_passenger from controller's

        return info for debugging
```