Tarea Integradora I

Gloria Vanesa Vicuña - A00369332
Ricardo Medina Sterling - A00369009
Alejandro Osejo Ochoa - A00372469

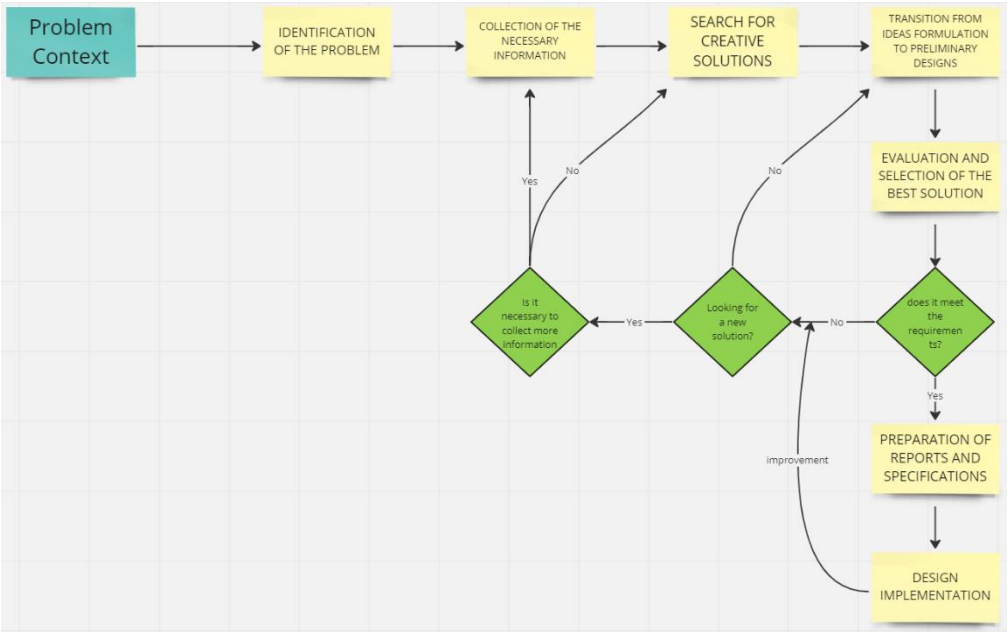
Computación y Estructuras Discretas I
Universidad Icesi
2023

ENGINEERING METHOD

Problem context

A recognized airline wants to create a first version of a system whose main objective is to improve the order in the airplane boarding and exiting process, display passenger information, and register their arrival.

Solution Development



PART 1: IDENTIFICATION OF THE PROBLEM

Functional requirements:

R1. Passenger upload to the system:

> **R1.1** The system must allow the upload of passenger information corresponding to a flight through a user-generated plain text file.
>
> **R1.2** The database must be simulated in this first version of the system.

R2. Passenger check-in at the boarding lounge:

R2.1 The system should allow an efficient search of passenger information once they arrive at the boarding lounge.

R2.2 The system must allow the registration of passengers' arrival at the boarding lounge to keep track of the order of arrival.

R3. Order of entry to the aircraft:

R3.1 The system must show the crew member in charge in which order passengers should enter the aircraft.

R3.2 The system must consider the order of arrival of the passengers, taking into account the sections of the aircraft and starting from the furthest from the entrance door to the one closest to it.

R3.3 For the first class, the system must consider other data such as accumulated miles, special attention required, and third age, among others, to establish the order of entry.

R4. Order of departure from the aircraft:

R4.1 The system must show, the crew member in charge, in which order the passengers must leave the aircraft.

R4.2 The exit order must consider the aircraft configuration, where those who exit first are those in the first rows, and for each row, the order is established by proximity to the aisle or order of arrival as the last instance.

Identification of needs and symptoms:

- The crew in charge needs to know the order in which passengers must enter and exit the aircraft.
- The order in which passengers enter and exit the aircraft depends on different factors.
- The solution to the problem must be efficient because the amount of data will be significantly too large in future versions.

Problem Definition:
The airline requires a system that shows the order in which passengers must enter and exit the aircraft and display passenger information.

PART2: COLLECTION OF THE NECESSARY INFORMATION

In order to have total clarity in the concepts involved, this search is made for the definitions of the terms that will be useful in the project:

Source:

https://www.geeksforgeeks.org

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.

A formal definition of the Big O: Big O notation is used to describe the efficiency or complexity of an algorithm in terms of its worst case. Furthermore, it is useful because it allows you to compare the efficiency of different algorithms without having to worry about irrelevant details such as processing speed or the exact number of operations performed.

Temporal complexity: Temporal complexity is a concept in computer science that refers to the amount of time an algorithm takes to execute relative to the size of its input. Also, Temporal complexity is commonly measured using the Big O notation. Therefore, understanding and analyzing the temporal complexity is essential for designing and selecting efficient and effective algorithms to solve problems in computer science.

Spatial complexity: Spatial complexity, also known as memory usage or memory consumption, is a term used in computer science to measure the amount of memory space needed to solve a problem or run an algorithm. Besides, spatial complexity refers to how much memory is needed to run an algorithm relative to the size of the data input. Therefore, it is important in the optimization of programs and algorithms, since a reduction in the amount of memory used can significantly improve the performance and execution speed of the program.

Hash tables: Hash tables, also known as hash maps, are a data structure used in programming to store and retrieve data efficiently. Also, Hash tables are useful when you need to quickly look up or retrieve values. Instead of searching through a list or array.

FIFO: FIFO is an abbreviation for first in, first out. It is a method for handling data structures where the first element is processed first, and the newest element is processed last.

LIFO: LIFO is an abbreviation for last in, first out. It is a method for handling data structures where the first element is processed last, and the last element is processed first.

Binary heaps: A binary heap is a hierarchical data structure used to implement a priority queue. It is represented as a complete binary tree in which each parent node has a value greater than or equal to any of its children. Thus, the binary heaps are powerful and efficient tools for solving a variety of optimization and algorithmic problems that require the maintenance of a priority queue.

PART 3: SEARCH FOR CREATIVE SOLUTIONS

Alternative 1: Safe Passengers in Hash Tables

In this alternative we save all passengers in Hash Tables, considering that allows us to get the information faster by a key.

Alternative 2: Safe passengers in Arraylist

The second alternative consists of saving the passengers in an ArrayList and employing it to save the priority information about the VIP passengers and the normal passengers.

Alternative 3: Safe Passenger in Binary Tree

Finally, the third alternative consists of saving the passengers in a Binary Tree, and we save the information considering the position and the priority.

PART 4: TRANSITION FROM IDEAS FORMULATION TO PRELIMINARY DESIGNS

The first thing that we develop in this stage is to discard the non-feasible alternatives for the development of the code. So, we discard the third alternative because it represents a lot of problems.

A careful review of the other alternatives leads us to the following:

Alternative 1: Safe passengers in Hash Tables.

- Hash tables allow access to elements very quickly.
- Hash tables can handle large volumes of data effectively.
- Hash tables use memory very efficiently, as they only store the elements that are needed and do not waste space on null or empty elements.
- The implementation of a hash table is relatively simple.
- Hash tables allow you to search for items by key, making them ideal for storing and searching for related data by key.

Alternative 2: Safe passengers in Arraylist.

It is not appropriate to use this kind of structure for this type of exercise because it can have complications at the time of consultation.

PART 5: EVALUATION AND SELECTION OF THE BEST SOLUTION

Criteria

The criteria that will allow the evaluation of alternative solutions must be defined and based on this result choose the solution that best meets the needs of the problem posed. The criteria we chose, in this case, are listed below. Also, we already chose the first alternative because

it is the one that can be adapted the most to solve the problem, however, it was analyzed and explained why it is a good criterion.

- Criterion A. Precision of the solution. The alternative delivers a solution:

  The precision is fundamental in every programming code, in this case, the precision is very good because it uses a hash function to assign a unique key to each data item that is stored in the table.

- Criterion B. Efficiency. A solution with better efficiency is preferred over the others considered. The efficiency can be:

  The efficiency in the alternative one, Hash Tables uses memory very efficiently, as they only store the elements that are needed and do not waste space on null or empty elements.

- Criteria C. Completeness. A solution that finds all solutions is preferred. How many solutions does it deliver:

  Alternative 1 is completeness because it allows us to find all the question marks that the problem of the plane appears.

- Criteria D. Ease of algorithmic implementation:

  The use of hash tables is very efficient in terms of access time and element insertion, which makes them very popular in implementing algorithms and data structures in programming.

PART 6: PREPARATION OF REPORTS AND SPECIFICATIONS

Problem Specification (in terms of input/output):
Problem 1: Allow the upload of passenger information corresponding to a flight through a user-generated plain text file.
inputs: the plain text file.

output: null

Problem 2: Check in the passenger arrival and display passenger information.
inputs: passenger id and records the order of arrival to the waiting room.
output: the name of the passenger, his age, his identification, his accumulated miles, if they need special care, and if they're first class.

Problem 3: Order the entry to the aircraft prioritizing VIP clients if they need priority attention and their accumulated miles
input: a plain text file in the order of arrival
output: Order of entry to the aircraft showing the passenger's id.

Problem 4: order the exit of the aircraft prioritizing VIP clients if they need priority attention and their accumulated miles.
input: a plain text file in the order of arrival.
output: Order of exit from the aircraft showing the passenger's id.

PART 7: DESIGN IMPLEMENTATION

Implementation in a programming language(JAVA)

List of tasks to implement:

a. Load passengers:

Name	loadPassengers
Description	It is a method that loads passenger information from a text file and stores it in a data structure.
Input	• filename: String, It is a file which stores the information of the passengers
Output	void

```
public void loadPassengers(String filename) {
  try {
    BufferedReader reader = new BufferedReader(new FileReader(filename));
    String line;
```

```
            this.columns = Integer.parseInt(reader.readLine());
            int numPassengers = Integer.parseInt(reader.readLine());
            passengers = new HashTable<>(numPassengers);
            while ((line = reader.readLine()) != null) {
                String[] infoPassenger = line.split(",");
                String id = infoPassenger[0].trim();
                String name = infoPassenger[1].trim();
                int age = Integer.parseInt(infoPassenger[2].trim());
                int row = Integer.parseInt(infoPassenger[3].trim());
                char seat = infoPassenger[4].trim().charAt(0);
                boolean isFirstClass = Boolean.parseBoolean(infoPassenger[5].trim());
                int accumulatedMiles = Integer.parseInt(infoPassenger[6].trim());
                boolean isSpecialAttention =
        Boolean.parseBoolean(infoPassenger[7].trim());
                Passenger passenger = new Passenger(id, name, age, row, seat, isFirstClass,
        accumulatedMiles, isSpecialAttention);
                passengers.put(id, passenger);
            }
            arrivalQueue = new Queue<>(passengers.size());
            boardingQueue = new MaxPriorityQueue<>(passengers.size());
            exitQueue = new MinPriorityQueue<>(passengers.size());
            reader.close();
          } catch (IOException e) {
            e.printStackTrace();
          }
        }
```

b. Register Arrival Passenger

Name	addToArrivalQueue
Description	The method consists of the registration of passengers who have arrived at the check-in area, everything will be received with the id of the passengers and will be added in the queue
Input	• id: String, it is an identification of the passengers
Output	void

```
        public void addToArrivalQueue(String id) {
          arrivalQueue.enqueue(id);
        }
```

c. Set Priority Entrance

Name	setPriorityEntrance

Description	It is a method which is in charge of putting the priority of the entrance according to the type of passenger
Input	empty
Output	void

```
public void setPriorityEntrance() {
  Queue<String> tempQueue = new Queue<>(passengers.size());
  while (!arrivalQueue.isEmpty()) {
    int arrivalNum = arrivalQueue.size();
    String id = arrivalQueue.dequeue();
    tempQueue.enqueue(id);
    Passenger passenger = passengers.get(id);
    passenger.setPriorityEntrance(arrivalNum);
    boardingQueue.maxInsert(passenger.getPriorityEntrance(), id);
  }
  arrivalQueue.setQueue(tempQueue.getQueue(), tempQueue.size());
}
```

d. Determinate Proximity

Name	determineProximity
Description	The method consists of determining the proximity to the aisle of the plane
Input	• seat: char, it is used to save these characters, which will save the seat
Output	• proximity: int, it is used to give us a value of the proximity to the aisle of the plane

```
private int determineProximity(char seat) {
  int proximity = 0;
  String seats = "ABCDEFGHIJ".substring(0, columns);
  int temp = (this.columns / 2) - 1;
  int temp2 = temp + 1;
  while (temp > 0 && temp2 < this.columns) {
    if (seats.charAt(temp) == seat || seats.charAt(temp2) == seat) {
      break;
    } else {
      proximity++;
      temp--;
      temp2++;
    }
  }
  return proximity;
}
```

e. Put the priority to the departure of the plane

Name	setPriorityExit
Description	The method consists of generating the priority at the time of departure of the plane in order to know which passenger gets off first
Input	empty
Output	void

```
public void setPriorityExit() {
  Queue<String> tempQueue = new Queue<>(passengers.size());
  int arrivalNum = 0;
  while (!arrivalQueue.isEmpty() && arrivalNum < passengers.size()) {
     arrivalNum++;
     String id = arrivalQueue.dequeue();
     tempQueue.enqueue(id);
     Passenger passenger = passengers.get(id);
     int proximity = determineProximity(passenger.getSeat());
     passenger.setPriorityExit(arrivalNum, proximity);
     exitQueue.minInsert(passenger.getPriorityExit(), id);
  }
  arrivalQueue.setQueue(tempQueue.getQueue(), tempQueue.size());
}
```

f. Get boarding Queue

Name	getBoardingQueue
Description	The method consists of taking the order of arrival of the passengers in a queue, then the method will deliver an Arraylist of type Passenger
Input	empty
Output	• boardingOrder: ArrayList<Passenger>,

```
public ArrayList<Passenger> getBoardingQueue() {
  ArrayList<Passenger> boardingOrder = new ArrayList<>();
  while (!boardingQueue.isEmpty()) {
     String id = boardingQueue.extractMax();
     Passenger passenger = passengers.get(id);
     boardingOrder.add(passenger);
  }
  return boardingOrder;
}
```

g. Get exit Queue

Name	getExitQueue

Description	The method consists of taking the departure order of the passengers in a queue, then the method will return an Arraylist of type Passenger
Input	empty
Output	• exitOrder: ArrayList<Passenger>,

```
public ArrayList<Passenger> getExitQueue() {
  ArrayList<Passenger> exitOrder = new ArrayList<>();
  while (!exitQueue.isEmpty()) {
     String id = exitQueue.extractMin();
     Passenger passenger = passengers.get(id);
     exitOrder.add(passenger);
  }
  return exitOrder;
}
```