engineering method

Members:

- Leidy Daniela Londoño - A00392917
- Isabella Huila Cerón - A00394751

Phase 1: Identification of the problem

Within an airline there has been disorder at the time of the entry and exit of passengers from the plane. For this reason, it has been necessary to develop and design an efficient system that improves this process, taking into account the various factors and challenges associated with this problem.

customer	airline
user	Crew Members
Functional requirements	**R1**: upload passenger information **R2:**Search passenger information **R3:**Register the arrival of the passenger to the lounge **R4:**Order the passengers according to the prioritizations defined for boarding the plane **R5:**Show the order that passengers should enter. **A6:**Order the passengers according to the prioritizations defined for the disembarkation of the plane. **A7:**Show the order that passengers should exit.
Problem context	● Within an airline there has been disorder at the time of the entry and exit of passengers from the plane. For this reason, it has been necessary to develop and design an efficient system that improves this process, taking into account the various factors and challenges associated with this problem.

non-functional requirements	R1: The system needs to be secure with the data of the passenger R2: The system needs to be efficient using the minimum resources necessary R3: The system needs to be functional in such a way that it fulfills the intended functions R4: The system needs to be reliable in such a way that the crew members trust the application.

Name or identifier	**R1**: upload passenger information		
Summarizes	The system must allow the initial loading of passenger information and this information will be stored in a database.		
Inputs	Input name	datatype	Selection or repetition condition
	Yam	String	
	go	String	

	numberofchair	String	that the chair is not available
	flightNumber	String	
	boardingGroup	String	
General activities necessary to obtain the results	1. get passenger information from the ticket you have purchased. 2. Upload passenger information to a database.		
Result or postcondition	1. have the information of passengers loaded in the database		
Outputs	Output name	datatype	Selection or repetition condition
	message	String	that an error has occurred and the information cannot be loaded into the database

Name or identifier	**R2:** Search passenger information		
Summarizes	The system must allow the search for a passenger, using the ID to obtain their complete information		
Inputs	Input name	datatype	Selection or repetition condition

	go	String	That the corresponding id has not been found
General activities necessary to obtain the results	1. enter the passenger ID that is required to search 2. The system should search for the passenger in the most efficient way possible 3. The system must display the complete information of the passenger		
Result or postcondition	1. The wanted passenger		
Outputs	Output name	datatype	Selection or repetition condition
	passenger info	String	That the passenger has not been found

Name or identifier	**R3:**Register the arrival of the passenger to the lounge		
Summarizes	The system must allow registering the arrival of a passenger when they enter the waiting room. Arriving passengers are entered into a hash-table		
Inputs	Input name	datatype	Selection or repetition condition
	go	String	

General activities necessary to obtain the results	1. When the passenger is searched, the person in charge of calling the boarding station will mark the passenger who is in the room. 2. The passenger is entered into a hash-table		
Result or postcondition	1. The change of arrival status of the passenger to the lounge		
Outputs	Output name	datatype	Selection or repetition condition
	message	String	An error has occurred and the status does not change

Name or identifier	**R4:**Sort the passengers according to the prioritizations defined for boarding the plane		
Summarizes	The system must allow ordering the passengers taking into account the prioritization and the order of arrival.		
Inputs	Input name	datatype	Selection or repetition condition
General activities necessary to obtain the results	1. That the system searches the hash-table for passenger information. 2. The system compares with respect to the priorities of each passenger. 3. The system orders according to that comparison.		

Result or postcondition	1. The list of passengers ordered according to their group and their priorities		
Outputs	Output name	datatype	Selection or repetition condition

Name or identifier	**R5:** Show the order that passengers should enter.		
Summarizes	Show the final list of the order in which passengers will enter according to their priorities and arrival time.		
Inputs	Input name	datatype	Selection or repetition condition
General activities necessary to obtain the results	1. get the ordered list of passengers		
Result or postcondition	1. the passenger list		
Outputs	Output name	datatype	Selection or repetition condition
	list	String	

Name or identifier	**R5:** Sort the passengers according to the prioritizations defined for the disembarkation of the plane.		
Summarizes	The system must order the passengers according to the proximity of the aisle and the order of arrival at the room will be given priority. This section should go into a priority queue and that queue is pushed to the output stack.		
Inputs	Input name	datatype	Selection or repetition condition
General activities necessary to obtain the results	1. The system looks for the information in the hash-table of passengers. For each row, an intermediate priority queue will be created, where they will enter taking into account the proximity of the aisle and the order of arrival. The queue information would be stored in a stack.		
Result or postcondition	1. the ordering of passengers at the time of departure		
Outputs	Output name	datatype	Selection or repetition condition

Name or identifier	**A6:** Show the order that passengers should exit.

Summarizes	Show the departure order of the passengers according to their row or, as a last resort, their arrival time		
Inputs	Input name	datatype	Selection or repetition condition
General activities necessary to obtain the results	1. get the ordered landing list of passengers		
Result or postcondition	1. the passenger list		
Outputs	Output name	datatype	Selection or repetition condition
	list	String	

Phase 2:

1. **Gatekeeper:**It is an application that allows airlines and airports to manage the process of boarding and disembarking passengers. The app helps airlines create a logical boarding sequence based on the number of available seats, aircraft capacity, and passenger preference.

   GateKeeper is a boarding management platform that allows airlines and airports to automate and optimize the process of boarding passengers on the plane. The platform uses artificial intelligence algorithms to create a logical boarding sequence based on the number of available seats, aircraft capacity and passenger preference.

   The GateKeeper platform offers several functionalities, among them:

   Seat management: The platform allows airlines to automatically assign seats to passengers and optimize the distribution of passengers on the plane. The platform can also take passenger preferences into account, such as seat location or the need for extra legroom.

Boarding Pass Scanning: The platform allows airlines to scan passengers' boarding passes at the gate, helping to streamline the boarding process and reducing the possibility of human error.

Boarding time alerts: The platform can send alerts to passengers about boarding time and flight departure time. This helps passengers to plan their time more effectively and reduce the stress associated with the boarding process.

Data Analytics: The platform provides real-time data analytics on the boarding process, allowing airlines to identify issues and take corrective action to improve the efficiency of the boarding process.

In short, GateKeeper is a boarding management platform that uses artificial intelligence algorithms to optimize the process of boarding passengers on the plane. The platform offers several functionalities, including seat management, boarding pass scanning, boarding time alerts, and real-time data analysis.

Taken from:

- **https://thenextweb.com/artificial-intelligence/2020/09/10/how-ai-is-making-air-travel-safer-and-more-efficient/**

- **https://gatekeeperhq.com/**

- **https://www.forbes.com/sites/sap/2018/06/25/ai-takes-flight-the-latest-in-airport-automation/?sh=6a45a3747ba7**

2. **Amadeus**is a global technology company that provides software and technology solutions for the travel and tourism industry. The company specializes in travel reservation systems and technology solutions for airlines, hotels, travel agencies and other travel service providers. Some of the services and solutions offered by Amadeus include:

   Airline Reservation System: Amadeus offers an online travel reservation system for airlines, which allows them to manage reservations, issue tickets and check-in passengers.

   Hotel Reservation Systems: Amadeus provides software solutions for hotels that enable them to manage reservations, rates and availability.

   Corporate Travel Solutions: Amadeus offers corporate travel management solutions that help companies plan, book and manage business travel for their employees.

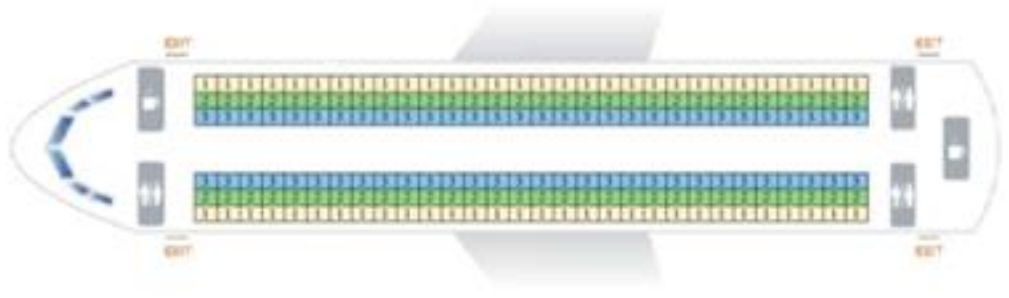
Mobility Solutions: Amadeus provides mobile solutions for travelers and travel service providers, including applications for flight reservations, hotel reservations and travel management.

Airport Management Systems: Amadeus also offers airport management solutions such as flight information systems, baggage handling solutions and gate management systems.

In short, Amadeus is a technology company that provides solutions and services to help the travel and tourism industry improve operations and better serve travelers.

Taken from:
https://amadeus.com/en/industries/airlines

3. **Wilma system:**
LThe passengers embark divided into three groups according to the seat that has been assigned to them. The boarding order is as follows: first the passengers with seats near the window, then the passengers with seats in the middle and finally the passengers with seats in the aisle. At the time boarding begins, all passengers in the first group are called to board in a random sequence within the group. Once the last passenger belonging to the first group has proceeded to board, the second group is called to board. As in the previous group, the passengers belonging to this segment embark in a random sequence within the group. Finally, the third group boards following the same pattern as the two previous groups.



Wilma. Source: Delcea et al, 2018 [23].

Also according to what we have seen in class, we have knowledge of some structures that we can implement to solve this problem, an investigation on these are the following:

- Hash Tables:
A hash table is an effective data structure for implementing dictionaries. Although looking up an item in a hash table can take as long as looking up an item in a linked list -O(n) in the worst case-, in practice, the hash table works extremely well. Under reasonable assumptions, the average time to lookup an item in a hash table is O(1).

A hash table generalizes the simplest notion of an ordinary array. Direct addressing in an ordinary array makes effective use of our ability to examine an arbitrary position in an array in O(1) time. We can take advantage of direct addressing when we can afford to assign an array that has a position for each possible key.
When the number of keys stored is small relative to the total number of possible keys, hash tables become an effective alternative to directly addressing an array, since a hash table typically uses an array of size proportional to the number of keys. stored. Instead of using the key as the array index directly, the array index is computed from the key.
In some applications, the direct address table itself may contain the elements in the dynamic set. That is, instead of storing an element's key and satellite data in an object external to the direct address table, with a pointer from a table position to the object, we can store the object at the table position itself. , saving space. We use a special key inside an object to indicate an empty position. Also, there is often no need to store the object's key, since if we have an object's index on the table, we have its key. If the keys to are integers and can be used as indices into the table, we can implement the dictionary operations with an O(1) runtime.
If two different elements have the same hash value, a collision occurs. To resolve collisions, the hash table uses different strategies, such as chaining and linear scanning.
The worst-case execution time for the insert is O(1). The insert procedure is fast in part because it assumes that the element x being inserted is not present in the table; if necessary, we can check this assumption (at additional cost) by looking for an element whose key is x:key before inserting. If the hashtable supports deletion, then your linked lists must be doubly linked in order for us to quickly delete an item.
How fast a hash table lookup depends on several factors, such as the size of the table, the hash function used, the table load, and the average length of the lists that are formed in each slot. In general, looking up a hash table can be very fast in the average case, but can be very slow in the worst case if all the elements are lumped into a single slot, resulting in a list having to be traversed long to find the desired item.
In practice, the lookup in a hash table is O(1 + n/m), where n is the number of elements in the table, m is the number of slots, and O(1) is the average access time to a hash table. slot. However, in the worst case, the search can take O(n) if all the elements are grouped into a single slot.

- Stacks:
  In a stack, the element removed from the array is the most recently inserted: the stack implements a LIFO (last-in, first-out) policy. Similarly, in a queue, the element removed is always the one that has been in the array the longest: the queue implements a first-in, first-out (FIFO) policy. There are several efficient ways to implement stacks and queues in a computer.
  The INSERT operation on a stack is often called a PUSH, and the DELETE operation, which does not take an element argument, is often called a POP. These names are allusions to physical batteries, such as the spring-loaded plate stacks used in cafeterias. The order in which the plates are pushed from the stack is the reverse of the order in which they were pushed onto the stack, since only the top plate is accessible.

PUSH and POP. Each of the three stack operations takes O(1) time.

- Queues:
  We call the INSERT operation on an ENQUEUE queue, and we call the DELETE DEQUEUE operation; Like the POP operation on a stack, DEQUEUE takes no element arguments. The FIFO property of a queue makes it operate like a line of customers waiting to pay the cashier. The tail has a head and a tail. When an element is queued, it takes its place in the queue, at the end of the queue, in the same way that a newly arrived client takes a place at the end of the line. The dequeued item is always the one at the head of the queue, such as the customer that has waited the longest at the head of the line.
  ENQUEUE and DEQUEUE, each operation takes O(1) time.
- Priority Queues:
  A priority queue is a data structure for holding a set S of items, each with an associated value called a key. A top priority queue supports the following operations:
  INSERT.S; x/ inserts the element x into the set S, which is equivalent to the operation S = SU {x}.
  MAXIMO.S / returns the element of S with the largest key.
  EXTRACT-MAX.S / removes and returns the element of S with the largest key.
  INCREMENT-KEY.S; x; k/ increases the value of the key of element x to the new value k, which is assumed to be at least as large as the current value of the key of x.
  Among its other applications, we can use top priority queues to schedule jobs on a shared computer. The top priority queue keeps track of the jobs to be performed and their relative priorities. When a job is completed or interrupted, the scheduler selects the highest priority job among the pending ones by calling REMOVE-MAX. The scheduler can add a new job to the queue at any time by calling INSERT.
  Alternatively, a minimum priority queue supports the INSERT, MINIMAL, REMOVE-MIN, and DECREASE-KEY operations. A lowest priority queue can be used in an event-based simulator. The items in the queue are events to be simulated, each with an associated occurrence time that serves as its key. Events must be simulated in order of their occurrence time, because the simulation of one event may cause other events to be simulated in the future. The simulation program calls EXTRACT-MIN at each step to choose the next event to simulate. As new events occur, the simulator inserts them into the lowest priority queue by calling INSERT.

- Linked List:
  A linked list is a data structure in which objects are arranged in a linear order. Unlike an array, in which the linear order is determined by the array's indices, in a linked list the order is determined by a pointer to each object. Linked lists provide a simple and flexible representation for dynamic sets.
  a doubly linked list L is an object with a key attribute and two other pointer attributes: next and previous. The object can also contain other satellite data. Given an element x in the list, x:next points to its successor in the linked list, and x:prev points to its predecessor. If x:prev is NIL, element x has no predecessor and is therefore the first element, or head, of the list. If x:next is NIL, element x has no successor and is therefore the last element, or tail, of the list. An L:head attribute points to the first element of the list. If L:head is NIL, the list is empty.

A list can take one of several forms. It can be single or double linked, it can be ordered or not, and it can be circular or not. If a list is simply linked, we omit the previous pointer on each element. If a list is ordered, the linear order of the list corresponds to the linear order of the keys stored in the elements of the list; the minimum element is then the head of the list, and the maximum element is the tail. If the list is not ordered, the items can appear in any order. In a circular list, the previous pointer to the head of the list points to the tail, and the next pointer to the tail of the list points to the head. We can think of a circular list as a ring of items.

Searching in a linked list

The procedure LIST-SEARCH.L; k/ finds the first element with key k in list L by a simple linear search, returning a pointer to this element. If there is no object with key k in the list, then the procedure returns NIL.

To search a list of n objects, the LIST-SEARCH procedure takes a worst-case time, since it may have to search the entire list.$o(n)$

Inserting into a linked list

Given an element x whose key attribute has already been set, the LIST-INSERT procedure "splices" x into the front of the linked list.

The LIST-INSERT operation inserts a new element x into list L in O(1) constant time, since only the next and pre pointers of related objects need to be set to insert x into the list. The execution time does not depend on the number of elements in the list, so it is O(1) for any list size.

The LIST-DELETE procedure removes an element x from a linked list L. It must be given a pointer to x, and then it "cuts" it from the list by updating the pointers. If we want to remove an item with a given key, we must first call LIST-SEARCH to get a pointer to the item.

LIST-DELETE runs in O(1) time, but if we want to delete an item with a given key, it takes O(n) time in the worst case because we must first call LIST-SEARCH to find the item.

- **binary tree**

  The search tree data structure supports many dynamic set operations, including find, min, max, predecessor, successor, insert, and delete. Therefore, we can use a search tree as both a dictionary and a priority queue.

  The basic operations on a binary search tree take time proportional to the height of the tree. For a complete binary tree with n nodes, such operations are executed in time worse than . However, if the tree is a linear chain of n nodes, the same operations will take a time worse than . The expected height of a randomly constructed binary search tree is , so the basic dynamic operations on such a tree will take on average a time of .$\ominus\ (logn)\ \ominus\ (n)\ \ominus\ (logn)\ \ominus\ (logn)$

  We can represent such a tree by a linked data structure in which each node is an object. In addition to a key and additional data, each node contains left, right, and p attributes that point to the nodes corresponding to its left child, its right child, and its parent, respectively. If a child or parent is not present, the corresponding attribute contains the NIL value. The root node is the only node in the tree whose parent is NIL.

The binary search tree property allows us to print all the keys in a binary search tree in ordered order using a simple recursive algorithm called an inorder tree walk. This algorithm is so named because it prints the root key of a subtree between printing the values in its left subtree and printing the values in its right subtree. (Similarly, a preorder tree traversal prints the root before the values in any of its subtrees, and a postorder tree traversal prints the root after the values in its subtrees.) This It takes time to traverse an n-node binary search tree, since after the initial call, $\ominus\ (n)$
Tree Search:
The procedure begins its search at the root and traces a simple path down the tree. For each node x it finds, it compares the key k with x:key. If the two keys are the same, the search ends. If k is less than x:key, the search continues in the left subtree of x, since the binary search tree property implies that k could not be stored in the right subtree. Symmetrically, if k is greater than x:key, the search continues in the right subtree. The nodes found during the recursion form a simple path down from the root of the tree, so the running time of TREE-SEARCH is O(h), where h is the height of the tree.
Insert:
To insert a new value into a binary search tree T, we use the TREE-INSERT procedure. The procedure takes a node ´ for which ´:key = k, ´:left = NIL, and ´:right = NIL. Modifies T and some of the attributes of ´ in such a way that it inserts ´ at an appropriate position in the tree. Like the other primitive operations on search trees, the TREE-INSERT procedure executes in O(h) time on a tree of height h.

Phase 3 :

1. One way to fix this problem is as follows:

   To save the passengers at the time they arrive at the lounge, it would be in a hash table, where the key will make the id. In this way, when looking for the passenger, more time would be saved.

   After this, the priority of each passenger with special needs would be defined. For the entrance of the plane, priority is given to the passengers who are furthest from the gate in each row. Then it is reviewed who has special needs or who are older and, finally, who has more accumulated miles. It should be noted that these criteria apply only to each row in particular. Regarding the departure of the plane, priority is given to the passengers who are closest to the gate in each row. And when some passengers have the same priority as the other is broken by the arrival time, that is, the passenger who arrived first would have higher priority than the other. By doing this, passengers will enter a priority queue.

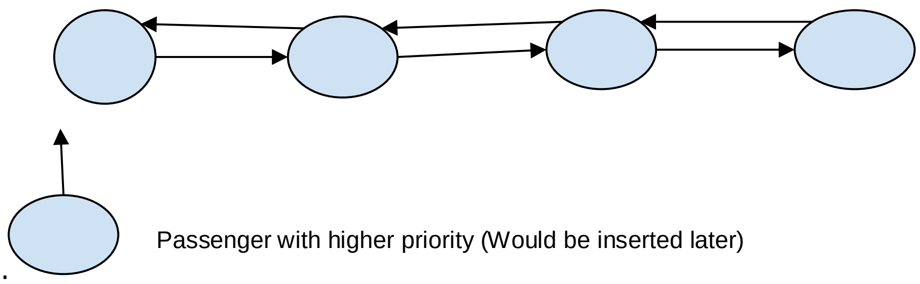
   As soon as the passengers enter, they leave the priority queue and the list of how the passengers should enter will be printed.

   Afterwards, the hash table is searched again to find out which seat corresponds to each passenger in order to enter an intermediate priority queue, with the aim that the passengers who are furthest from the aisle, but a priority is also established, where the one that has arrived later has more priority taking into account the two that are closest to the corridor. Later, the passengers who are in the intermediate queue would go into a pile and this would be done for all the rows.

Finally, it will print the output list

2. One solution is after loading the txt data, it would stuff the passengers into a binary search tree. After that, the priority of each passenger must be defined, as is done in option 1.

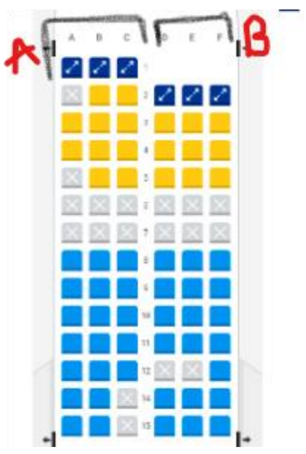
   The passengers would enter a doubly linked list, where the one with the highest priority enters as the head. This process will search all the passengers to know where to insert. When defining the boarding list, the passengers who are first on the list will be the first to enter the plane. Finally the list is printed. For the exit, the chair would be searched in the search tree, an intermediate doubly linked list is made to do it by sections, the passengers who are closest to the aisle will be inserted into that list so that each passenger who enters is inserted last ; When they are in the same column, it is compared depending on who arrived earlier and thus enter the start list. Finally, the passenger landing list is printed.



.

3. Another solution would be similar to option 1, the difference is that instead of saving the information in a Hash table, it would be saved in a doubly linked list.
4. Another solution to this problem would be similar to option 1, but the output would be done differently, minimum priority queues are used where the seat of each passenger would be searched in the hash table and the one that is closest to the The plane door would have the lowest priority, then, for each section, a minimum priority is established for the passengers who are closest to the aisle, also taking into account that the one who arrived earlier would have lower priority than the two who are in the same column, that is, as follows:

   Assuming that the columns in the plane are divided by the aisle, we would have column A and column B, so that the person who arrived earlier and is near the aisle or the door will leave the two columns first. .



Phase 4:

Alternative 2: This alternative would not be as useful for the following reasons:

1. Searching and inserting into a search tree would take much longer, since the execution time would be O(h), where h is the height of the tree. Therefore, if there is a lot of information, this time increases, making it inefficient.
2. When a passenger is going to be inserted into the doubly linked list, it would also take a long time, since it first searches for the key of each passenger to establish where the new passenger would go to be inserted and the search would be linear, therefore it would take a long time.
3. When inserting a passenger, it would have to go through all the existing nodes so that it can be located, which would take more time and we want it to be in the most efficient way.

Alternative 3:

Checking in the passenger when they arrive in the room would be useful, since its runtime is O(1) for any size of the list. However, when the search for the passengers is going to be done, it would be linear, therefore in the worst case the execution time is O(n) since it may have to search the entire list.

Step 5 :

Alternative 1:

This alternative would be useful for the following reasons:

1. By implementing the Hash table to store passenger information, the insert procedure is more efficient, since it takes O(1) in the worst case.
2. At the time of the search, it would be more efficient so that the algorithm takes O(1 + n/m) where n is the number of elements in the table and m is the number of slots and O(1) is the time of the average slot access.

3. We use priority queues because they would be more efficient for this case since they use special data structures that allow them to keep the elements in order according to their priority, these allow elements to be added and removed from the priority queue over time O(log(n)), which makes the operations very efficient, even for large data sets.
4. We also use stacks since they are efficient because their data structure allows elements to be inserted and deleted very quickly and with minimal resources. Stacks are based on a very simple principle, the last item to be pushed onto the stack is the first to be removed; It would be very useful for the departure of the plane, since when we have the intermediate queue, the one who is first in the queue would enter the stack.

Alternative 5:
This option would be equally useful as alternative 1, but at the time of departure, this alternative would be much better, because only one top priority queue is used, so there would be no need to add an intermediate structure. Therefore, this alternative would save much more memory.

Taking into account the reasons we give, to solve our problem we choose alternative 5.