**Alejandro Londoño Bermúdez – A00395978**

**Julián Andrés Mendoza Castro – A00395237**

**Santiago Valencia García – A00395902**

**Método de la Ingeniería**

**Contexto del problema:**

Una reconocida aerolínea requiere de un sistema cuyo objetivo principal es mejorar el orden en el proceso de ingreso y salida de los aviones, ya que no existe un sistema de software que permita mantener un orden correcto de los pasajeros al momento de ingresar y salir del avión.

**Identificación del problema:**

* La aerolínea requiere de un sistema que permita mejorar el orden en el proceso de entrada y salida del avión.
* No existe un sistema de software que permita a la aerolínea mantener un buen orden de los pasajeros al momento de ingresar y salir del avión.
* El sistema debe priorizar la llegada puntual de los pasajeros al avión.
* La solución del problema debe ser eficiente ya que en el futuro se usará gran cantidad de datos.

**Especificación de requerimientos:**

* El sistema debe permitir cargar a los pasajeros correspondientes a un vuelo.
* Para el proceso de lista de pasajeros en la entrada, el sistema debe:
  + Priorizar a los pasajeros de primera clase, a partir de diferentes ítems de prioridad como lo son: vulnerabilidad y millas acumuladas.
  + Reconocer la puntualidad de todos sus pasajeros.
  + Hacer la lista de entrada en orden descendente a partir de las filas del avión (Contemplando lejanía a la puerta).
* Para el proceso de lista de pasajeros en la salida, el sistema debe:
  + Hacer la lista de entrada en orden ascendente a partir de las filas del avión (Contemplando cercanía a la puerta).
  + Priorizar a los pasajeros con cercanía al pasillo.
  + Priorizar a los pasajeros a partir de su momento de llegada a sala de abordaje.

**Métodos de almacenamiento de la información:**

* **Hash table:** Una hash table es una estructura de datos que asocia llaves o claves con valores. Toma un valor de entrada y lo convierte en un índice en la tabla. La tabla está compuesta de una matriz de filas donde se puede contener uno o más elementos. Cuando se inserta un nuevo elemento, se aplica la función hash para determinar en qué fila debe almacenarse. Cuando se busca un elemento, se usa nuevamente la función hash para encontrarlo en la tabla.
* **ArrayList:** Un arraylist es una estructura de datos que permite almacenar elementos en memoria. Tiene la ventaja de ser más flexible y dinámico, es decir, puede expandirse o contraerse si se añade o se elimina un elemento.
* **Cola:** Una cola es una estructura de datos que se utiliza para almacenar elementos en orden secuencial. Sigue el principio de “Primero en entrar, primero en salir”.
* **Pila:** Una pila es una estructura de datos que se utiliza para almacenar elementos en orden secuencial. Sigue el principio de “Último en entrar, primero en salir”.
* **Montículo:** Un montículo es una estructura de datos que se utiliza para mantener un conjunto de elementos con una relación de orden parcial. Cada elemento del montículo tiene un valor asociado que determina su prioridad en el conjunto. Los elementos se ordenan de tal manera que el elemento con mayor prioridad se encuentra en la raíz del árbol, mientras que los elementos de menor prioridad se encuentran en los nodos inferiores. Esta estructura se denomina montículo máximo.
* **Cola de prioridad:** Una cola de prioridad es una estructura de datos que permite acceder y eliminar el elemento con la mayor prioridad en cualquier momento. Cada elemento de la cola prioridad tiene un valor asociado que determina su prioridad en el conjunto. Su implementación puede basarse en montículos.

Referencias

Association for Computing Machinery (ACM). (Sitio web). Recuperado de https://www.acm.org/

Institute of Electrical and Electronics Engineers (IEEE). (Sitio web). Recuperado de https://www.ieee.org/

Association for Information Systems (AIS). (Sitio web). Recuperado de https://aisnet.org/

**Búsqueda de soluciones creativas**

A partir del desarrollo de diferentes estructuras de datos vistas, se emplean algunas de éstas para tratar de darle solución al problema; la elección de éstas estructuras es dada por las diferentes funcionalidades y usabilidad que cumplen y se adaptan al contexto del problema.

1. **Almacenar a los pasajeros en una cola para la llegada y una pila para la salida**: Para el ingreso, se puede utilizar una cola FIFO (primero en entrar, primero en salir), donde cada pasajero se agrega al final de la cola cuando llega a la sala de abordaje. Cuando llegue el momento de abordar el avión, los pasajeros se sacan de la cola en orden, permitiendo su ingreso al avión en el mismo orden en el que llegaron a la sala de abordaje. Para la salida, se puede utilizar una cola diferente, como una cola LIFO (último en entrar, primero en salir) o una cola de prioridad. Si se utiliza una cola LIFO, los pasajeros se agregan a la cola en orden inverso a su ubicación en el avión. De esta manera, cuando llegue el momento de salir del avión, los pasajeros se eliminan de la cola en orden inverso al que se agregaron, asegurando que los que estén más cerca de la puerta de salida salgan primero.
2. **Utilizar una lista doblemente enlazada para el ingreso y la salida**: Se puede utilizar una lista doblemente enlazada para manejar tanto el ingreso como la salida de los pasajeros. Cada nodo de la lista representará un pasajero y contendrá su información, como su nombre, número de asiento, prioridad, etc. Durante el proceso de ingreso, los pasajeros se agregarán a la lista en el orden en que lleguen a la sala de abordaje y se deberá ordenar a partir de las diferentes prioridades. Cuando llegue el momento de abordar el avión, se puede iterar sobre la lista y permitir el ingreso de los pasajeros en el orden en que aparecen. Para la salida, se puede utilizar la misma lista, pero ordenada de forma inversa. Los pasajeros que se encuentren en las filas más cercanas a la puerta de salida se eliminarán primero de la lista, seguidos por los de filas más alejadas.
3. **Utilizar una matriz para el ingreso y un árbol para la salida**: Para manejar el ingreso, se puede utilizar una matriz en la que cada fila representa una sección del avión y cada columna representa un asiento. Cuando los pasajeros lleguen a la sala de abordaje, se les asignará un asiento en la matriz según la disponibilidad. Para la salida, se puede utilizar un árbol donde cada nodo representa una fila del avión. Los pasajeros se agregarán a los nodos correspondientes según la fila en la que se encuentren. A medida que los pasajeros vayan saliendo del avión, se irán eliminando de la estructura de árbol, lo que permitirá a la tripulación ver en qué orden deben salir los pasajeros restantes. Además, se puede utilizar la información de la matriz para establecer el orden de llamado de los pasajeros durante el proceso de ingreso, es decir, empezar por las filas más alejadas a la puerta de ingreso y avanzar hacia las más cercanas.
4. **Almacenar la información de los pasajeros en una hashtable y utilizar dos colas de prioridad, una para la llegada y otra para la salida de los pasajeros**: Se utiliza una hashtable para almacenar la información de los pasajeros. La hashtable permite un acceso eficiente a la información de cada pasajero, ya que se utiliza una función hash para asignar un índice único a cada uno de ellos, lo que permite acceder a su información de manera constante en tiempo O(1). Además, la hashtable también permite actualizar la información de los pasajeros y eliminarlos del sistema de manera eficiente. Luego, se utilizan dos colas de prioridad, una para la llegada y otra para la salida de los pasajeros. La cola de prioridad para la llegada se utiliza para ordenar a los pasajeros en el orden en que llegan a la sala de abordaje, teniendo en cuenta factores como la hora de llegada y la prioridad de la clase a la que pertenecen. La cola de prioridad para la salida se utiliza para ordenar a los pasajeros en el orden en que deben salir del avión, teniendo en cuenta la configuración de las filas del avión.

**Descarte de ideas no viables**

1. Almacenar a los pasajeros en una cola para la llegada y una pila para la salida.

Trabajar solamente con éstas estructuras no nos permite satisfacer todas los requerimientos descritos en el problema como lo son: dar prioridad por diferentes conceptos a los pasajeros de clase alta; la salida debe empezar por las filas más cercanas a la puerta, primero salen los asientos más cercanos al pasillo (Al haber dos asientos por fila con la misma característica se debe priorizar por orden de llegada) y se sigue sacando por orden de llegada a los demás asientos de la fila.

Encontramos entonces que, pese a ser la cola y pila son estructuras de datos sencillas y con buena complejidad en la mayoría de sus métodos, al hacer uso de éstas solamente, no se pueden satisfacer los requerimientos esenciales.

1. **Utilizar una lista doblemente enlazada para el ingreso y la salida**:

Utilizar una lista doblemente enlazada para el ingreso y salida de pasajeros no es eficiente en términos de tiempo de ejecución y memoria, ya que se deben ordenar los datos dos veces, lo que requiere una cantidad significativa de procesamiento. Además, la inserción y eliminación de elementos en una lista doblemente enlazada también puede ser costosa en términos de tiempo de ejecución.

En lugar de una lista doblemente enlazada, es más eficiente utilizar estructuras de datos como colas de prioridad, tablas hash o árboles binarios de búsqueda balanceados, que pueden ordenar los datos de forma más eficiente y permitir un acceso rápido y fácil a los elementos en la estructura. Estas estructuras de datos también pueden ser más escalables y manejar grandes cantidades de datos de manera más eficiente que una lista doblemente enlazada.

**Criterios para evaluar las ideas**

­ *Criterio A.* Precisión de la solución. La alternativa entrega una solución:

­ [2] Exacta (se prefiere una solución exacta)

­ [1] Aproximada

­ *Criterio B.* Eficiencia. Se prefiere una solución con mejor eficiencia que las otras consideradas. La eficiencia puede ser:

­ [4] Constante

­ [3] Mayor a constante

­ [2] Logarítmica

­ [1] Lineal

­ *Criterio C.* Completitud. Se prefiere una solución que encuentre todas las soluciones. Cuántas soluciones entrega:

­ [3] Todas

­ [2] Más de una si las hay, aunque no todas

­ [1] Sólo una o ninguna

­ *Criterio D.* Facilidad en implementación algorítmica:

­ [2] Compatible con las operaciones aritméticas básicas de un equipo de cómputo moderno ­ [1] No compatible completamente con las operaciones aritméticas básicas de un equipo de cómputo moderno

**Idea: Matriz para el ingreso y un árbol para la salid**a:

Criterio A. Precisión de la solución: [2] Exacta. La solución debe permitir una asignación precisa de los asientos a los pasajeros y un orden preciso de salida.

Criterio B. Eficiencia: [3] Mayor a constante. La solución debe ser más eficiente que una solución constante, ya que el proceso de abordaje y salida de un avión es un proceso que requiere rapidez y eficiencia.

Criterio C. Completitud: [2] Más de una si las hay, aunque no todas. La solución debe permitir la asignación de asientos a la mayoría de los pasajeros, aunque puede haber algunos casos en los que no sea posible.

Criterio D. Facilidad en implementación algorítmica: [2] Compatible con las operaciones aritméticas básicas de un equipo de cómputo moderno. La solución debe ser compatible con las operaciones aritméticas básicas de un equipo de cómputo moderno para que pueda ser implementada de manera sencilla y eficiente.

Total: 9

**Idea: Hashtable y dos colas de prioridad**:

Criterio A. Precisión de la solución: [2] Exacta. La solución debe permitir una asignación precisa de los asientos a los pasajeros y un orden preciso de salida.

Criterio B. Eficiencia: [3] Mayor a constante. La solución debe ser más eficiente que una solución constante, ya que el proceso de abordaje y salida de un avión es un proceso que requiere rapidez y eficiencia.

Criterio C. Completitud: [3] Todas. La solución debe permitir la asignación de asientos a todos los pasajeros de manera precisa.

Criterio D. Facilidad en implementación algorítmica: [2] Compatible con las operaciones aritméticas básicas de un equipo de cómputo moderno. La solución debe ser compatible con las operaciones aritméticas básicas de un equipo de cómputo moderno para que pueda ser implementada de manera sencilla y eficiente.

Total: 10

Idea a desarrollar: Desarrollo de una Hashtable y dos colas de prioridad para entrada y salida de pasajeros