**Trabajo Práctico Especial**

**Programación III**

**Integrante: Mateo Albert**

**E-Mail:** [**mateo.albert45@gmail.com**](mailto:mateo.albert45@gmail.com)

**Introducción al problema**

Luego de leer el enunciado identifique que el problema se basa en crear un aeropuerto utilizando la estructura de grafo donde cada nodo seria un aeropuerto y cada arista un ruta, lo primero que decidí fue realizar un esquema de la estructura del problema, de esta manera, identifique las distintas clases y atributos que voy a utilizar.

**Modelado del Problema**

En base al esquema realizado para identificar la estructura de los datos decidi crear las siguientes clases, con sus respectivos atributos, de la siguiente forma:

**Datos identificas y relaciones**

* Clase Sistema: Su función es almacenar los distintos aeropuertos y reseras, dentro de una misma clase.
* Clase Aeropuerto: Funciona como el nodo en un grafo, se relaciona con otros Aeropuertos a través de la clase Ruta.
* Clase Ruta: Permite conectar uno o mas aeropuertos entre si.
* Clase Aerolínea: Sirve para poder almacenar mas de una aerolínea en la ruta.
* Clase Reserva: Se utiliza para saber en una determinada ruta la cantidad de asientos reservados que hay para una aerolínea particular.

**Estructura de datos**

* Clase Sistema:
  + Aeropuertos: Vector <Aeropuerto>
  + Reservas: Vector <Reserva>
* Clase Sistema:
  + Nombre: String
  + País : String
  + Ciudad: String
  + Color: String
  + Estado: String
  + Rutas: Vector<Ruta>
* Clase Ruta:
  + Origen: Aeropuerto
  + Destino: Aeropuerto
  + Kilómetros: int
  + Cabotaje: boolean
  + Aerolíneas : Vector<Aerolínea>
* Clase Aerolínea:
  + Nombre: String
  + Pasajes: int

* Clase Reserva:
  + Origen: Aeropuerto
  + Destino: Aeropuerto
  + Aerolinea: String
  + Asientos\_Reservados: int

**Ventajas y desventajas Estructura**

Utilizar grafo con lista de adyacencia:

Decidí utilizar una lista de adyacencia debido a que no ocupa tanta memoria como la matriz ya que el espacio que utiliza es proporcional a la cantidad de nodos que tiene, pero la desventaja es que tiene un tiempo de ejecución de O(n).

Vector:

El crear un vector tiene la ventaja de despreocuparme del tamaño del arreglo al momento de insertar o eliminar los elementos, además que en este caso es necesario debido a que no conocemos cual va a ser la cantidad de elementos que debemos insertar en el mismo. Por el lado de las desventajas, una de ellas, es que cada vez que inserto un elemento debe copiar el arreglo actual a un arreglo mayor tamaño esto sucede en un tiempo O(n) y en algunos casos puede ser innecesario.

Crear la clase Aerolínea:

Debido a que cada ruta tiene un conjunto de aerolíneas con sus respectivos pasajes, decidí crear la clase aerolínea para así no tener que crear muchas instancias de la clase ruta con casi la misma información , de esta manera, la estructura es más fácil de comprender y de utilizar

**Implementación de servicio 1**

Para realizar el primer servicio del aeropuerto realice cuatro métodos:

Vuelo\_directo: Mediante este método se pide que ingresen los aeropuertos origen y destino y la aerolínea con la que se desea viajar, así, en este método se recorre todas las rutas del aeropuerto origen, en caso de que la ruta conecte con el aeropuerto destino se calcula la cantidad de pasajes, y si hay pasajes disponibles se calcula la cantidad de estos.

Cant\_pasajes: Debido a que cada ruta tiene un vector de aerolíneas decidí recorrer este vector y en caso de que el nombre de la aerolínea pasado por parámetro sea igual al de la aerolínea que está comparando retorna la cantidad de pasajes que tiene para esta ruta.

Cant\_Pasajes\_Disponibles: Este método mediante la ruta, aerolínea y cantidad de pasajes que son pasados como parámetro permite recorrer las reservas que tiene cargado el sistema para que en caso de que estas se apliquen a la ruta retorne la cantidad de pasajes que hay disponibles.

Es\_igual: Funciona como un “equals” entre una reserva y una ruta, para poder identificar a que vuelo hace referencia la reserva.

calcular\_pasajes\_disponibles: permite calcular la cantidad de pasajes que quedan disponibles.

Este algoritmo tiene una complejidad o(n)^2

**Seguimiento del algoritmo Vuelo\_directo:**

public void vuelo\_directo(Aeropuerto origen, Aeropuerto destino, String aerolinea) {

for (int i = 0; i < origen.getRutas().size(); i++) {

Ruta r = origen.getRutas().elementAt(i);

Se recorren las rutas del aeropuerto origen pasado como parámetro y se guardan en la variable r de tipo Ruta

if (r.getDestino().equals(destino)) {

Chequea que el destino de la ruta seleccionada sea igual al aeropuerto destino pasado como parámetro

int aux = r.cant\_pasajes(aerolinea);

en caso de que se encuentre una ruta con ese destino se calcula la cantidad de pasajes que tiene esa ruta con su determinada aerolínea

if (aux > 0) {

System.out.println(Cant\_Pasajes\_Disponibles(r, aerolinea, aux));

En caso de que la ruta tenga esa aerolínea la variable aux debe ser igual ala cantidad de pasajes totales que dispone para esa ruta, luego se calcula la cantidad de pasajes disponibles.

**Seguimiento del algoritmo Cant\_pasajes:**

for(int i=0;i<Aerolineas.size();i++) {

Aerolinea aux = Aerolineas.elementAt(i);

Se recorre cada una de las aerolíneas que tiene la ruta y se guardan en la variable aux tipo Aerolínea

if(aux.NombreIgual(a)) {

return aux.GetPasajes();

Si el nombre de la aerolínea pasada como parámetro es igual al que esta en la ruta se retorna la cantidad de pasajes totales que dispone

return -1;

En caso de que no la encuentra devuelve -1

**Seguimiento del algoritmo Cant\_pasajes\_Disponibles:**

public int Cant\_Pasajes\_Disponibles(Ruta r, String aerolinea, int cantidad) {

for (int i = 0; i < Reservas.size(); i++) {

if(Reservas.elementAt(i).es\_igual(r.getOrigen(), r.getDestino(), aerolinea, cantidad)){

Se recorre el vector de Reserva que tiene cargado el sistema y se verifica que reserva pertenece a la ruta

return calcular\_cantidad\_pasajes(cantidad);

} }}

return cantidad;

|  |
| --- |
|  |

En caso de ser verdadero se retorna la cantidad de pasajes disponibles y si no encuentra una ruta quiere decir que no tiene ninguna reserva, es decir, todos los pasajes están disponibles.

**Seguimiento del algoritmo cantidad\_pasajes\_disponibles:**

public int calcular\_pasajes\_disponibles(int cantidad){

return cantidad-Asientos\_Reservados

calcula la cantidad de pasajes disponibles restando los asiento totales del vuelo con los asientos reservados.

|  |
| --- |
|  |
|  |  |
|  |
|  |  |
|  |  |

**Implementación de servicio 2**

Para realizar el primer servicio del aeropuerto realice cuatro métodos:

Dfs: este algoritmo sirve para poder setear todos los nodos del grafo en el estado “Sin visitar” y utilizar el método recursivo dfs\_visit.

dfs\_visit: Mediante este algoritmo se pueden ir recorriendo cada uno de los aeropuertos hasta llegar al aeropuerto destino y se calcula la cantidad de kilómetros que tiene el camino completo.

contiene\_aerolinea\_distinta\_a:

**Conclusiones**

Luego de realizar el trabajo puede llegar a la conclusión de que el mismo fue muy útil para poder interiorizarme acerca de cómo es el funcionamiento de un grafo y que tan importantes es ir adaptando nuestros algoritmos, con los tiempos de ejecución y la utilización de la adecuado utilización del espacio de memoria requerido, para poder llegar a un óptimo funcionamiento de nuestra aplicación, esto se ve reflejado al momento de seleccionar la estructura de trabajo que se utilizó para desarrollar la este trabajo practico especial. Finalmente me hubiera gustado tener un poco mas de tiempo para lograr optimizar de una mejor manera los diversos servicios pero esto me servirá para realizar futuros trabajos prácticos con mas anticipación.