Universidad de Buenos Aires
Facultad De Ingeniería
Año 2023 - 1er Cuatrimestre

Informe TP2 Algueiza
Algoritmos y programacion(95.12)

1er. Cuatrimestre
Curso: Buchwald

INTEGRANTES:

Fuentes Acuña, Brian Alex - #101785
<bfuentes@fi.uba.ar>

Carbajal Robles, Kevin Emir - #108304
<kcarbajal@fi.uba.ar>

Resumen

Se crea el TDA SistemaDeVuelos, el cual contiene los comandos que cumplen los ordenes de complejidad pedido. Este TDA usa otros tdas auxiliares, como el abb, hash y heap.

A continuación se explican los comandos implementados y los tdas que se usaron en ellos para justificar la complejidad

1. Comandos implementados

- **Comando agregar archivo**
  En este comando se guarda un vector de vuelos en el hash y abb . Dependiendo si estos vuelos existian antes o no, va a guardar/actualizar estos vuelos.
  En el caso del abb, todas sus operaciones cuestan $O(\log n)$, en el hash, en cambio, será $O(1)$. Todo esto es por cada vuelo que guardemos, que seria $O(\log n)$. Cuando guardemos todos los vuelos, el comando quedara finalmente con $O(v \log n)$, siendo v la cantidad de vuelos que se agregan y n la cantidad de vuelos totales en el sistema.

- **Comando ver tablero**
  Se iterara el abb dependiendo de las fechas que nos pidan. Como este abb esta ordenado por fecha de partida y numero de vuelo, nos dara complejidad $O(\log n)$, siendo n la cantidad de vuelos totales. En el peor caso, cuando nos pidan recorrer todas las fechas en donde partan los vuelos, la complejidad quedara $O(n)$. Para mostrar de forma *ascendente* o *descendente* se usan arreglos de tamaño predefinidos para asignarle los vuelos a mostrar, por lo que no aumenta la complejidad.

- **Comando info vuelo**
  Para este comando se usa el hash al cual se le guardaron los vuelos con el numero de vuelo como clave. Segun el numero de vuelo, devuelve el vuelo correspondiente. Como estamos operando con un hash, su complejidad queda $O(1)$.

- **Comando prioridad vuelos**
  Este comando hace uso de un heap de vuelos a partir de una funcion de comparacion y un arreglo de vuelos, este arreglo se obtiene a partir del hash. Se lo itera para obtener todos los vuelos, que cuesta $O(n)$. Como creamos el heap a partir de un arreglo, tambien costara $O(n)$. Esta funcion de comparacion compara los vuelos con respecto a su prioridad, y en caso de que sean iguales, desempata por su numero de vuelo.
  Se desencola los k vuelos pedidos del heap. Cada vez que desencolamos, cuesta $O(log(n))$, si desencolamos k veces, entonces quedara $O(klog(n))$.
  Sumandos todas las complejidades mencionadas, quedaria finalmente como $O(n + klog(n))$.

- **Comando siguiente vuelos**
  Para saber cual vuelo es el siguiente dado una fecha, se iterara el abb a partir de la misma. Este vuelo siguiente debera cumplir solamente que conecte el mismo destino pedido.
  Como iteraremos todos los vuelos siguientes hasta que alguna coincida con la conexion, obtendremos $O(log(f))$, con f la cantidad de fechas diferentes en las que se puede hacer dicho viaje.

- **Comando Borrar** Finalmente el comando borrar utiliza todas las funciones de borrar de las estructuras, usando principalmente el iterador externo del abb para borrar un rango de fechas específico.
  A medida que se borra un vuelo del abb ($O(\log n)$) también lo hace el de hash ($O(1)$), siendo entonces la complejidad $O(\log n)$ para borrar un solo vuelo. Sea K la cantidad de vuelos que hay en el rango de fechas especificado, quedara básicamente $O(K \log n)$