Algoritmos y Programación II – curso Buchwald

Facultad de Ingeniería de la Universidad de Buenos Aires (FIUBA)

TP N°2 - ALGUEIZA

Ayudante de grupo: Martin Buchwald

Grupo: G09

Integrantes:

- GARCIA CAMPOS, LUCAS EDUARDO (110099)
- CHEN, HELEN ELIZABETH (110195)

INTRODUCCIÓN

Este informe consiste en la descripción del análisis y diseño de nuestro sistema de consulta de vuelos, requerido por el aeropuerto de Algueiza, centrándonos en su modularización, y en la explicación de los algoritmos y estructuras de datos utilizados en función de los requisitos establecidos.

ANÁLISIS Y DISEÑO

Estructuras y TDAs

Para empezar a diseñar la solución, lo primero que se nos ocurrió fue plantear en el sistema un nuevo TDA “Airport” que sea el que se encargue de ejecutar los distintos comandos, actualizando el sistema o imprimiendo por pantalla la información que se requiera. A su vez, al momento de la implementación, tuvimos que decidir qué TDAs auxiliares utilizaremos para poder alcanzar los objetivos que se nos propusieron con los debidos tiempos de ejecución. Entonces, llegamos a la conclusión que lo mejor sería utilizar un ABB y un Hash, y en ellos guardar toda la información de los vuelos.

La información a guardar en cada estructura de datos era la misma, que era una estructura de vuelo llamada “flight” que contenía toda la información del mismo que se encontraba en los archivos ingresados, pero la forma de guardarlos era diferente para cada estructura. En el ABB decidimos crear una estructura aparte que sea “dateAndId”, ya que al tratar de guardar con solo la fecha originalmente, nos encontramos con el problema de que cuando un vuelo tiene la misma fecha que otro, este sobreescribirá la información del otro que ya estaba guardado y se perderá. Entonces para asegurarnos que esto no suceda creamos esta estructura de forma que los vuelos guardados se guarden en orden de la fecha de salida, y en caso que dos vuelos coincidan en la fecha, ordenarlos por los números de vuelo. Y por otro lado, en el Hash los guardamos utilizando como clave el número de código de vuelo, y de

esta manera poder conseguir la información de un vuelo específico lo más rápido posible.

Algoritmos

1. agregar_archivo

En el primer requerimiento se nos pidió que el sistema pueda guardar y/o actualizar la información de los vuelos provenientes de diversos archivos de tipo .csv en un tiempo de ejecución de **O(V log n)**, siendo V la cantidad de vuelos en el nuevo archivo y n la cantidad total de los vuelos en el sistema.

Para lograr esto, nuestro algoritmo se basa en leer cada línea del archivo que contiene la información de un solo vuelo, y en cada lectura procesa la información en un Hash (costando O(1)), siendo el código del vuelo la clave, y la estructura flight su valor. A su vez, en un Diccionario Ordenado(ABB), se guarda/actualiza la clave, que es una estructura dateAndId (que contiene la información de la fecha y el código del vuelo) con el mismo valor almacenado en el Hash, donde cada vez que ocurre esto cuesta O(log n).

Por lo tanto, debido a que es necesario realizar este proceso V cantidad de veces, la complejidad total es O(V log n).

2. ver_tablero

En cuanto al segundo, se nos pidió que se imprima por pantalla una K cantidad de vuelos dentro de las fechas indicadas, ordenados tanto ascendente como descendentemente según se nos pida, en tiempo O(log(v)), siendo v la cantidad de vuelos.

Para esto, se decidió iterar el ABB y aprovechar su propiedad de estar ordenado, e ir guardando la información de todos los vuelos contenidos en esas fechas, y cuando ya se hayan iterado todos o se haya alcanzado la cantidad de vuelos solicitados a mostrar, se cortara la iteración y se imprimirá todos los vuelos obtenidos. Esta operación cuesta

O(log(V)) en el caso promedio, o O(V) en el caso de querer visualizar todos los vuelos.

3. info_vuelo

Respecto al tercero, se debe obtener la información de un vuelo con su código en un tiempo de ejecución O(1).

A fin de cumplir con este requisito, nuestro algoritmo accede a la información del vuelo pedido que se encuentra contenida en la estructura flight, que es un valor dentro de nuestro Hash, utilizando el código del vuelo ingresado como clave (en O(1)).

4. prioridad_vuelos

Se nos exigió que el sistema deba mostrar de mayor a menor una cantidad determinada (K) de vuelos de mayor prioridad junto a su código en O(n + K log n).

Para alcanzar este objetivo, en principio tuvimos que guardar en un arreglo toda la información de los vuelos almacenada como estructura de tipo flight en nuestro Hash, cuya complejidad temporal es de O(n), siendo n la cantidad total de vuelos en el sistema.

A continuación, con el arreglo obtenido, decidimos crear un Heap a partir del arreglo (O(n)) de tipo flight de máximas prioridades donde cada vez que desencolemos nos dará el vuelo de mayor prioridad, costando O(log n) por cada operación, e imprimirá la prioridad del vuelo junto a su código. Como el sistema solo debe mostrar K cantidad de vuelos de mayor prioridad, esta acción debe ocurrir K veces, siendo su costo total de O(K log n).

Por consiguiente, sumando la complejidad temporal de la primera operación con la de la segunda, genera O(n + K log n) como el total de tiempo de ejecución para obtener los vuelos de mayor prioridad.

5. siguiente_vuelo

En quinto lugar, fue requerida la posibilidad de visualizar la información del siguiente vuelo que haya para una conexión definida con una fecha inicial en un tiempo de ejecución de O(log Fconexión), siendo Fconexión la cantidad de fechas diferentes en las que se pueda realizar dicho viaje.

Con lo dicho en mente, teniendo nuestro ABB ordenado por fechas, decidimos iterarlo desde la fecha ingresada hasta que encuentre un vuelo con el mismo origen y destino (la conexión), generando un costo de log(Fconexión). El primer vuelo que encuentre con estas características lo imprime en pantalla.

6. borrar

Para el comando borrar, se nos solicitó que sea ejecutado en tiempo O(K log(n)), siendo K la cantidad de vuelos entre los rangos de fechas a eliminar, y n la cantidad de vuelos totales del sistema.

Para esto, decidimos iterar el ABB de forma que siempre que haya un vuelo entre esos rangos, lo eliminemos tanto del Hash como del ABB, y guardamos su información en un arreglo para después poder imprimir todos los vuelos que se hayan eliminado.

Debido a que el comando de Borrar en un Hash cuesta O(1), y en un ABB cuesta O(log(n)), en total el tiempo de ejecución será de O(K log(n)), tal como nos pedía el enunciado.

CONCLUSIÓN

Para concluir, mediante la correcta modularización, la elección adecuada de estructuras de datos y algoritmos descritos anteriormente, logramos crear una solución del problema que sea efectiva, organizada, mantenible, y escalable.