Universidad de los Andes
Facultad de Ingeniería

Departamento de Ingeniería de Sistemas y Computación
Estructuras de Datos y Algoritmos
ISIS-1225

ABET Engineering Accreditation Commission

ANÁLISIS DEL RETO

Mateo Sáánchez Zapata, 202321354, m.sanchezz@uniandes.edu.co

Santiago Garzon Garcia, 202512373, S.garzong2@uniandes.edu.co

Santiago Ismael Escobar Maidana, 202516956, si.escobar@uniandes.edu.co

Requerimiento <<n>>

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Parámetros necesarios para resolver el requerimiento.
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Si se implementó y quien lo hizo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(...)
Paso 2	O(...)
Paso ….	O(...)
TOTAL	***O(...)***

Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el analisis de complejidad.

Requerimiento Ejemplo

Descripción

```python
def get_data(data_structs, id):
    """
    Retorna un dato a partir de su ID
    """
    pos_data = lt.isPresent(data_structs["data"], id)
    if pos_data > 0:
        data = lt.getElement(data_structs["data"], pos_data)
        return data
    return None
```

Este requerimiento se encarga de retornar un dato de una lista dado su ID. Lo primero que hace es verificar si el elemento existe. Dado el caso que exista, retorna su posición, lo busca en la lista y lo retorna. De lo contrario, retorna None.

Entrada	Estructuras de datos del modelo, ID.
Salidas	El elemento con el ID dado, si no existe se retorna None
Implementado (Sí/No)	Si. Implementado por Juan Andrés Ariza

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Buscar si el elemento existe (isPresent)	O(n)
Obtener el elemento (getElement)	O(1)
TOTAL	***O(n)***

Análisis

A pesar de que obtener un elemento en un *ArrayList,* dada su posición, tiene complejidad constante, la implementación de este requerimiento tiene un orden lineal O(n). Esto debido a que, lo primero que se hace es verificar si el elemento hace parte de la lista. Específicamente, a la hora de buscar un elemento en una lista, en el peor de los casos es necesario recorrer toda la lista, es decir, complejidad lineal.

Requerimiento <<1>>

Descripción

Este requerimiento se encarga de identificar vuelos que pertenecen a una aerolínea específica (carrier_code) y cuyo retraso en salida (dep_delay_min) está dentro de un rango definido (min_delay y max_delay). Los datos de los vuelos están almacenados en una estructura de árbol (catalog["flights"]["root"]). El algoritmo recorre todo el árbol (usando un recorrido *inorder* implementado con traverse_tree) para encontrar los vuelos que cumplen con los criterios. Finalmente, los vuelos encontrados se ordenan (primero por retraso ascendente y luego por fecha/hora de salida

ascendente) usando al.merge_sort y se devuelve un resumen de los resultados (tiempo de ejecución, total de vuelos, y los primeros/últimos 5 vuelos).

Entrada	Estructura de datos del catálogo (árbol de vuelos), código de la aerolínea, retraso mínimo, retraso máximo.
Salidas	Un diccionario con el tiempo de ejecución, los parámetros de entrada, el total de vuelos encontrados, y los datos de los primeros 5 y últimos 5 vuelos ordenados.
Implementado (Sí/No)	Si se implementó por Santiago Garzon.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	Recorrido del árbol (traverse_tree) para filtrar los vuelos O(N)
Paso 2	Almacenamiento de vuelos en matching_flights O(1)*
Paso 3	Ordenamiento de los vuelos (al.merge_sort)O(M log M)
Paso 4	Obtención de los primeros 5 y últimos 5 vuelos O(1)
TOTAL	***O(N + M log M)***

Análisis

El requerimiento tiene una complejidad final de **O(N + M log M)$**, donde N es el número total de vuelos y M es el número de vuelos que cumplen con el filtro. La función opera en dos fases principales: primero, realiza un recorrido completo de la estructura de datos en árbol que contiene todos los vuelos, visitando los N nodos para aplicar los criterios de filtro (aerolínea y rango de retraso), lo que requiere un tiempo de O(N). Segundo, la lista resultante de M vuelos que cumplen el criterio es ordenada utilizando el algoritmo *Merge Sort*, lo que aporta una complejidad de O(M log M). Dado que el peor caso ocurre cuando la mayoría de los vuelos cumplen el filtro (M approx N), la complejidad total asintótica se considera **O(N log N)**, dominada por la fase de ordenamiento de los datos después de su extracción del árbol.

Requerimiento <<2>>

Descripción

Para el requerimiento 2 recorrimos en orden el RBT que almacena todos los vuelos y filtramos únicamente aquellos que llegaban al aeropuerto destino y presentaban anticipo (delay negativo). El

rango ingresado por el usuario se transformó a su equivalente en minutos negativos para facilitar la comparación. Luego ordenamos los resultados con merge_sort usando un comparador propio que prioriza el mayor anticipo y desempata por fecha-hora real de llegada. Finalmente, construimos las listas de los primeros y últimos cinco vuelos.

Entrada	catalog, dest_code, min_early y max_early
Salidas	exec_time_ms, dest_code, min_early, max_early, total_flights, first_5y last_5.
Implementado (Sí/No)	Mateo Sánchez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrido in-order del árbol Red-Black	O(n) (Visitamos cada nodo una sola vez.)
Filtrado de vuelos que cumplen destino y rango de anticipo	O(1) por vuelo → total O(n) (Es solo ver campos y comparar valores.)
Inserción de vuelos válidos en un array_list	O(1) amortizado por inserción → total O(k) (k = cantidad de vuelos válidos, k ≤ n)
Ordenamiento con merge_sort	O(k log k) (Ordenamos únicamente los vuelos filtrados.)
Construcción de listas first_5 y last_5	O(1)
TOTAL	*O(n + k log k)*

Análisis

Las pruebas con distintos destinos y rangos de anticipo mostraron que el requerimiento responde de forma consistente y rápida, con tiempos de ejecución muy bajos incluso recorriendo todos los vuelos del año. Esto coincide con su complejidad O(n + k log k), donde k es el número de vuelos filtrados, que en la práctica siempre es pequeño. Los resultados se ordenan correctamente por anticipo y luego por fecha-hora de llegada, y cuando no existen vuelos que cumplan las condiciones la salida es clara

Requerimiento <<3>>

Descripción

Este requerimiento identifica los vuelos que pertenecen a una aerolínea específica, tienen un destino concreto y cuya distancia recorrida se encuentra dentro de un rango definido. El algoritmo obtiene todos los vuelos almacenados en el árbol maestro de tipo RBT mediante la función rbt.values, lo que genera una lista enlazada con los registros en orden. Posteriormente, recorre dicha lista de forma secuencial para aplicar los filtros correspondientes. Los vuelos que cumplen los criterios se almacenan en una estructura array_list. Una vez recolectados, se ordenan utilizando quick_sort con un criterio que prioriza la distancia ascendente y desempata por fecha/hora de llegada. Finalmente, se genera un resumen del requerimiento que incluye el tiempo de ejecución, el total de vuelos que cumplen el filtro, y los primeros y últimos 5 vuelos ya ordenados.

Entrada	Estructura del catálogo (árbol RBT con vuelos), código de aerolínea, destino, distancia mínima y máxima.
Salidas	Un diccionario con el tiempo de ejecución, los parámetros de entrada, el total de vuelos encontrados, y los primeros/últimos 5 vuelos ordenados.
Implementado (Sí/No)	Si se implementó por Santiago Escobar

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	Normalización del rango de distancias O(1)
Paso 2	Obtención de todos los valores del RBT con rbt.values O(N)
Paso 3	Recorrido secuencial de la lista enlazada y filtrado de vuelos O(N)
Paso 4	Inserciones en matching_flights mediante add_last O(1) amortizado

Paso 5	Ordenamiento de la lista filtrada con al.quick_sort O(M log M)
Paso 6	Extracción de primeros 5 y últimos 5 vuelos O(1)
TOTAL	***O(N + M log M)***

Análisis

El requerimiento presenta una complejidad final de O(N + M log M), donde *N* es el número total de vuelos almacenados en el árbol y *M* el número de vuelos que cumplen los criterios de aerolínea, destino y rango de distancia. En primer lugar, se obtienen todos los valores del árbol maestro con un recorrido completo en orden, lo que implica un costo de O(N). Luego, se recorre la lista resultante para aplicar los filtros, también en O(N). Los vuelos que cumplen las condiciones se almacenan y posteriormente se ordenan utilizando Quick Sort, cuyo costo es O(M log M). En el peor caso, cuando *M* se aproxima a *N*, la operación de ordenamiento domina la complejidad total, resultando en O(N log N). En casos donde M es pequeño respecto a N, el recorrido inicial es el costo dominante y la complejidad queda muy cercana a O(N).

Requerimiento <<4>>

Descripción

Este requerimiento identifica las N aerolíneas con mayor número de vuelos que caen dentro de un rango de fechas y una franja horaria específicos. El algoritmo realiza un recorrido completo del árbol de vuelos (traverse_tree, N nodos), filtrando los vuelos que cumplen con los criterios de fecha y hora. Durante este recorrido, se utiliza un diccionario (airlines_data) para contar los vuelos por aerolínea y, simultáneamente, registrar el vuelo con la menor duración para cada una. Una vez finalizado el recorrido, los datos del diccionario se convierten en una lista y se ordenan de forma descendente por el número total de vuelos. Finalmente, se toman las primeras N aerolíneas de la lista ordenada y se les calcula la duración y distancia promedio.

Entrada	Estructura de datos del catálogo (árbol de vuelos), rango de fechas, franja horaria, y el número máximo de aerolíneas a retornar (top_n).
Salidas	Un diccionario con el tiempo de ejecución, los parámetros de entrada, el total de aerolíneas que operaron en el rango, y los datos detallados de las top_n aerolíneas (código, total de vuelos, promedios, y su vuelo de menor duración).
Implementado (Sí/No)	Si se implementó , por todos los integrantes

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	Recorrido del árbol (traverse_tree) y llenado del diccionario (airlines_data)O(N)
Paso 2	Conversión de diccionario (A aerolíneas) a lista (airlines_list)O(A)
Paso 3	Ordenamiento de las aerolíneas (al.merge_sort) O(A log A)
Paso 4	Extracción del top_n y cálculo de promedios O(A) o O(N) en el peor caso
TOTAL	***O(N + A log A)***

Análisis

El requerimiento tiene una complejidad final de **O(N + A log A)**, donde N es el número total de vuelos en el catálogo y A es el número de aerolíneas únicas encontradas en el rango de fechas y horas especificado. La operación más costosa es el recorrido completo de los N vuelos en la estructura de árbol para aplicar los filtros de tiempo y fecha y, simultáneamente, acumular los datos (conteo de vuelos, vuelo de menor duración) por aerolínea en un diccionario. Este paso requiere un tiempo de O(N) debido a que se deben inspeccionar todos los nodos. Posteriormente, la lista de A aerolíneas se ordena mediante *Merge Sort* para identificar el *Top N*, lo cual introduce un factor de $O(A \log A)$. Dado que el número de vuelos (N) es típicamente mucho mayor que el número de aerolíneas (A), la complejidad se aproxima a **O(N)** si A es pequeño, o a **O(N log N)** en el peor caso teórico donde A es comparable a N y domina el paso de ordenamiento.

Requerimiento <<5>>

Descripción

Para el requerimiento 5 realizamos un recorrido in-order sobre el árbol Red-Black que contiene todos los vuelos, filtrando únicamente aquellos que pertenecen al aeropuerto de destino solicitado y se encuentran dentro del rango de fechas indicado. Cada vuelo válido se procesa dentro de un map_separate_chaining, donde se acumulan estadísticas por aerolínea, como número total de vuelos, distancia promedio, duración promedio y puntualidad promedio. Luego convertimos dicho mapa en un array_list y lo ordenamos mediante merge_sort usando un comparador que prioriza la mejor puntualidad. Finalmente seleccionamos las N aerolíneas más puntuales, identificamos su vuelo de mayor distancia

Entrada	catalog, date_initial, date_final, dest_code y top_n
Salidas	exec_time_ms, date_range, dest_code, top_n, total_airlines y la lista airlines
Implementado (Sí/No)	Si, los 3 integrantes

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrido in-order del RBT	O(n) (Visitamos todos los vuelos una vez.)
Inserción y actualización de estadísticas en el mapa	O(1) promedio por vuelo → O(n) total
Obtención de los valores del mapa	O(m)
Cálculo de promedios por aerolínea	O(m)
Ordenamiento de aerolíneas con merge_sort	O(m log m)
Selección del Top N	O(top_n)

Análisis

Para varios rangos de fechas y aeropuertos destino, el requerimiento identificó correctamente las aerolíneas válidas, calculó sus promedios y seleccionó el vuelo de mayor distancia, todo con tiempos muy bajos acordes a la complejidad O(n + m log m). El ordenamiento por puntualidad promedio se reflejó correctamente en las pruebas y los resultados fueron consistentes con los datos observados. Asimismo, cuando no hubo aerolíneas dentro del rango, la función respondió adecuadamente

Requerimiento <<6>>

Descripción

Este requerimiento identifica las aerolíneas más estables en cuanto a retrasos de salida dentro de un rango determinado de fechas y distancias. Para ello, el algoritmo obtiene todos los vuelos almacenados en el árbol maestro mediante rbt.values y los recorre aplicando una serie de filtros (fechas válidas, distancias en el rango permitido y retrasos conocidos). Los vuelos válidos se agrupan por aerolínea dentro de un mapa hash (airlines_data), donde se registran tanto los retrasos como los vuelos asociados a cada aerolínea. Posteriormente, se calcula para cada aerolínea el número total de vuelos, el promedio de retraso, la desviación estándar y el vuelo cuyo retraso se encuentra más cerca del promedio. Luego se construye una lista con todas las aerolíneas válidas, la cual se ordena mediante al.quick_sort usando como criterios la desviación estándar ascendente y, en caso de empates, el retraso promedio ascendente. Finalmente, se extraen las M aerolíneas más estables y se construye el reporte de salida.

Entrada	Estructura del catálogo (árbol de vuelos), rango de fechas, rango de distancias, cantidad máxima de aerolíneas a retornar (top_m).
Salidas	Un diccionario con el tiempo de ejecución, los parámetros de entrada, la cantidad total de aerolíneas encontradas y los datos detallados de las top_m aerolíneas más estables.
Implementado (Sí/No)	Sí se implementó, por todos los integrantes.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	Normalización de fechas y distancias O(1)
Paso 2	Obtención de todos los vuelos del RBT mediante rbt.values O(N)
Paso 3	Recorrido de todos los vuelos, verificación de filtros y agrupamiento por aerolínea en mapa hash O(N)
Paso 4	Conversión del mapa a lista O(A)
Paso 5	Cálculo por aerolínea: promedio, varianza, desviación estándar y vuelo representativo O(N)
Paso 6	Construcción de lista airlines_list O(A)
Paso 7	Ordenamiento de la lista de aerolíneas usando quick_sort O(A log A)
Paso 8	Selección del top_m y formateo de resultados O(A)
TOTAL	***O(N + A log A)***

Análisis

La complejidad del requerimiento es O(N + A log A), donde N representa la cantidad total de vuelos en el catálogo y A el número de aerolíneas que tienen al menos un vuelo válido en el rango establecido. El proceso comienza con un recorrido completo del árbol maestro para obtener todos los vuelos, seguido de un filtrado y agrupamiento por aerolínea, operaciones que requieren un tiempo O(N). Posteriormente, para cada aerolínea se calculan estadísticas de retraso y el vuelo más cercano al promedio, lo cual también toma tiempo proporcional al número total de vuelos evaluados, es decir O(N). Finalmente, se ordenan las aerolíneas por estabilidad utilizando Quick Sort, lo que introduce un costo de O(A log A). En condiciones típicas, donde el número de aerolíneas es mucho menor al número total de vuelos, domina el término lineal O(N). Sin embargo, en el peor caso teórico (A ≈ N), la operación de ordenamiento podría incrementar la complejidad total hasta aproximarse a O(N log N)