ANÁLISIS DEL RETO

Estudiante 1, Ileanne Valderrama Ocampo - 202514355 – i.valderramao@uniandes.edu.co

Estudiante 2, Juan Pablo Peñaranda Cely – 202422285 – jp.penaranda@uniandes.edu.co

Requerimiento <<1>>

Este requerimiento obtiene los trayectos dentro de una franja específica de fecha y hora de recogida, retorna después todos los datos de esos trayectos que si

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Catalogo (catalog), fecha y hora inicial de la franja (fecha_y_hora_inical), tamaño de la muestra (tamanio_muestra) y fecha y hora final de la franja (fecha_y_hora_final).
Salidas	Tiempo de ejecución(tiempo), número de trayectos que cumplen con el filtro(elementos_filtrados) y los datos de ese recorrido (rta).
Implementado (Sí/No)	Si, Ileanne Valderrama

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Conversión de fechas de entrada	O(1)
Recorrido y filtrado de la lista de taxis	O(n)
Ordenamiento de los viajes filtrados (quick_sort)	O(n log n)
Obtención de sublistas (sub_list)	O(k)
Unión de las sublistas (add_last)	O(k)
TOTAL	***O(n log n)***

Análisis

El requerimiento tiene esta complejidad gracias al quick sort, ya que este divide la lista hasta que solo quede uno, lo que distribuye los datos, a pesar de eso en los otros casos al ser O(n) porque tiene complejidad lineal al recorrer “taxis”, al igual que las de O(k) para las sub listas, como payment_type y más.

Requerimiento <<2>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Catálogo (`catalog`) que contiene la lista de viajes (), el valor mínimo de latitud (`lat_min`), el valor máximo de latitud (`lat_max`) y el número de trayectos a mostrar (N).
Salidas	Un diccionario con: `time_ms`: tiempo de ejecución en milisegundos. `total`: número total de trayectos dentro del rango de latitud. `first`: los primeros N trayectos (más cercanos al límite superior de latitud) `last`: los últimos N trayectos (más cercanos al límite inferior).
Implementado (Sí/No)	Si, Juan pablo Peñaranda

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrer todos los viajes en el catálogo para filtrar por latitud	O(n)
Ordenar los viajes filtrados por latitud y longitud	O (n log n)
Calcular los primeros y últimos N elementos	O(n)
TOTAL	**O (n log n)**

Análisis

El algoritmo recorre toda la lista de viajes en el catálogo y selecciona aquellos cuya latitud de recogida está dentro del rango. Posteriormente, ordena los resultados en orden descendente según la latitud y longitud, lo cual permite obtener los trayectos geográficamente más cercanos al límite superior del rango.

La complejidad total está dominada por el proceso de ordenamiento, resultando en O(n log n).
En pruebas con el conjunto de datos `taxis-small.csv`, el requerimiento retornó correctamente los trayectos filtrados, mostrando los N primeros y últimos viajes junto con el tiempo de ejecución.

Requerimiento <<3>>

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)	No

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
N/A	N/A
N/A	N/A
N/A	N/A
TOTAL	N/A

Análisis

N/A

Requerimiento <<4>>

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

Descripción

BObtiene los trayectos que terminan en una fecha especifica y después o antes del filtro de “antes o despues”. Muestra los n primeros y los n ultimos

Entrada	Catalogo(catalog), fecha de terminación(fecha_terminación), momento de interés (momento), tiempo de referencia del trayecto (tiempo_referencia), tamaño de la muestra (tamanio_muestra)
Salidas	Tiempo de ejecución (tiempo), numero total de trayectos (elementos_filtrados) y la información de los taxis.
Implementado (Sí/No)	Si, Ileanne

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Creación de tabla de hash y asignación de trayectos por fecha	O(n)

Filtrado de trayectos según momento y tiempo_referencia	O(k)
Ordenamiento de los trayectos filtrados	O(k log k)
Obtención de sublistas de tamaño N y unión en muestra	O(n)
TOTAL	O(n + k log k)

Análisis

El ordenamiento permite que los trayectos salgan en orden de entrada, mientras se recorre todo lineal según lo que se desee buscar, si “antes” de la hora o “despues” de la hora.

Requerimiento <<5>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Catálogo (`catalog`) que contiene la lista de taxis, la hora de terminación (`hora_terminacion`, en formato `"%Y-%m-%d %H"`) y el tamaño de la muestra a mostrar (`tamanio_a_mostrar`).
Salidas	`total`: número total de trayectos que finalizaron en la hora especificada. `muestra`: lista de los primeros y últimos N trayectos terminados en esa hora. `tiempo`: tiempo total de ejecución en milisegundos.
Implementado (Sí/No)	Si, Juan pablo peñaranda

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrer todos los viajes en el catálogo y construir la tabla hash (agrupación por hora de terminación)	O(n)
Verificar si existe la llave (`contains`) en la tabla hash	O(1)
Obtener la lista asociada a la llave (`get`)	O(1)
Ordenar los trayectos filtrados (`merge_sort`)	O(n log n)
Extraer los primeros y últimos N trayectos (`sub_list`)	O(N)
TOTAL	**O(n log n)**

Análisis

El algoritmo utiliza una tabla hash (`hash_table`) para agrupar los viajes según su hora de terminación (`dropoff_datetime` truncada a la hora).
Esto permite una búsqueda eficiente de todos los viajes terminados en una hora específica. Luego, la lista de trayectos correspondiente a esa hora se ordena por fecha y hora, y se extraen los primeros y últimos N elementos, garantizando así que los resultados se muestren desde los más antiguos hasta los más recientes. El costo computacional más alto proviene del ordenamiento, que domina el tiempo total del algoritmo con una complejidad O(n log n).

Durante las pruebas con los archivos de muestra (`taxis-small.csv` y `taxis-large.csv`), el requerimiento mostró tiempos de ejecución razonables y retornó correctamente la cantidad total de trayectos, junto con los N primeros y últimos dentro de la hora indicada.

Requerimiento <<6>>

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

Descripción

Obtiene los trayectos que inician en un barrio especifico dentro de un rango de tiempo.

Entrada	Catalogo (catalog), Barrio de recogida(neighborhood_name), hora incial del rango de recogida(hora_inicial), hora final del rango de recogida(hora_final), tamaño de la muestra (tamanio_a_mostrar)
Salidas	Total, muestra y tiempo
Implementado (Sí/No)	Si, Ileanne Valderrama

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Verificar existencia del barrio y obtener coordenadas	O(1)
Recorrido y filtrado de todos los trayectos por hora y distancia	O(n)
Ordenamiento de los trayectos filtrados (merge_sort)	O(k log k)
Obtención de sublistas n primeros y n últimos	O(k)
TOTAL	O(n + k log k)

Análisis

La tabla hash permite acceder rápidamente a las coordenadas del barrio, se usa haversine para sacar la distancia y se garantiza por el ordenamiento de que la estructura salga del más antiguo al más reciente.

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