ANÁLISIS DEL RETO

Luis Alejandro Rodríguez Arenas, Cód. 202321287, la.rodrigueza12@uniandes.edu.co

Santiago Rojas, Cod. 202420928, s.rojasg23@uniandes.edu.co

Requerimiento <<1>>

Este requerimiento se encarga de procesar la información de todos los viajes del catálogo para obtener estadísticas sobre aquellos viajes que cumplen con un filtro específico: el número de pasajeros.
El algoritmo recorre todos los viajes almacenados, filtra los que cumplen con la condición de pasajeros, acumula diferentes métricas (duración, costo, distancia, peajes, propinas), y determina tanto el medio de pago más frecuente como la fecha más recurrente en los viajes seleccionados.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Catálogo de viajes, número de pasajeros
Salidas	Diccionario con estadísticas: total de viajes filtrados, promedios de duración, costo, distancia, peajes y propinas, además del medio de pago más utilizado y la fecha más frecuente.
Implementado (Sí/No)	Si se implementó y lo hizo Santiago Rojas

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Obtener tamaño de la lista de viajes (`lt.size`)	O(1)
Recorrer todos los viajes en la lista (`for i in range(size)`)	O(n)
Acceder a cada viaje (`lt.get_element`)	O(1) por acceso, total O(n)
Filtrar por número de pasajeros	O(1) por viaje, total O(n)
Acumular métricas (duración, costos, distancias, etc.)	O(1) por viaje, total O(n)
Contar frecuencia de métodos de pago y fechas (uso de diccionario)	O(1) amortizado por operación, total O(n)
O(m), donde m es el número de métodos de pago distintos (m << n, se aproxima a O(1))	O(n), se aproxima a O(1)

Determinar la fecha más usada (recorrido sobre `date_counter`)	O(n), se aproxima a O(1)
TOTAL	***O(n)***

Análisis

El algoritmo presenta un comportamiento lineal, dado que necesita recorrer todos los viajes del catálogo para aplicar el filtro y realizar las operaciones de acumulación. Cada operación realizada dentro del recorrido es constante, y el uso de diccionarios permite manejar eficientemente los conteos de frecuencias.

En la práctica, el tiempo de ejecución crece de forma proporcional al número de viajes. Esto concuerda con lo esperado, ya que la tarea principal consiste en iterar sobre la totalidad de los registros.

Gracias a esta implementación, se pueden obtener promedios y estadísticas de manera eficiente, incluso en catálogos con un número considerable de viajes, manteniendo un desempeño consistente con la complejidad lineal.

Requerimiento <<2>>

Descripción

Este requerimiento se encarga de procesar la información de los viajes del catálogo filtrando únicamente aquellos que fueron pagados con un método de pago específico.
El algoritmo recorre todos los viajes almacenados, selecciona los que cumplen con el filtro de método de pago, acumula métricas (duración, costo, distancia, peajes, propinas), y determina tanto el número de pasajeros más frecuente como la fecha de finalización más común.

Entrada	Catálogo de viajes, número de pasajeros

Salidas	Diccionario con estadísticas: total de viajes filtrados, promedios de duración, costo, distancia, peajes y propinas, además del número de pasajeros más frecuente y la fecha de finalización más recurrente.
Implementado (Sí/No)	Si se implementó y lo hizo Santiago Rojas

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Obtener tamaño de la lista de viajes (`lt.size`)	O(1)
Recorrer todos los viajes (`for i in range(size)`)	O(n)
Acceder a cada viaje (`lt.get_element`)	O(1) por acceso, total O(n)
Filtrar por método de pago	O(1) por viaje, total O(n)
Acumular métricas (duración, costos, distancias, etc.)	O(1) por viaje, total O(n)
Contar frecuencia de pasajeros y fechas (uso de diccionario)	O(1) amortizado por operación, total O(n)
Determinar el número de pasajeros más frecuente	O(n), se aproxima a O(1)
Determinar la fecha más frecuente	O(n), se aproxima a O(1)
TOTAL	O(n)

Análisis

El algoritmo presenta un comportamiento lineal, dado que necesita recorrer todos los viajes del catálogo para aplicar el filtro y realizar las operaciones de acumulación. Cada operación realizada dentro del recorrido es constante, y el uso de diccionarios permite manejar eficientemente los conteos de frecuencias.

En la práctica, el tiempo de ejecución crece de forma proporcional al número de viajes. Esto concuerda con lo esperado, ya que la tarea principal consiste en iterar sobre la totalidad de los registros.

Gracias a esta implementación, se pueden obtener promedios y estadísticas de manera eficiente, incluso en catálogos con un número considerable de viajes, manteniendo un desempeño consistente con la complejidad lineal.

Requerimiento <<3>>

Descripción

Este requerimiento procesa la información del catálogo de viajes filtrando únicamente aquellos cuya tarifa total se encuentra dentro de un rango específico de costos.
El algoritmo recorre todos los viajes, selecciona los que cumplen la condición, acumula métricas (duración, costo, distancia, peajes y propinas) y determina tanto el número de pasajeros más frecuente como la fecha de finalización más común.

Entrada	Catálogo de viajes, costo mínimo, costo máximo
Salidas	Diccionario con estadísticas: total de viajes filtrados, promedios de duración, costo, distancia, peajes y propinas, además del número de pasajeros más frecuente y la fecha de finalización más recurrente.
Implementado (Sí/No)	Si se implementó y lo hizo Luis Alejandro Rodríguez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Obtener tamaño de la lista de viajes (`lt.size`)	O(1)
Recorrer todos los viajes (`for i in range(size)`)	O(n)
Acceder a cada viaje (`lt.get_element`)	O(1) por acceso, total O(n)
Filtrar por rango de costo	O(1) por viaje, total O(n)
Acumular métricas (duración, costos, distancias, etc.)	O(1) por viaje, total O(n)
Contar frecuencia de pasajeros y fechas (uso de diccionario)	O(1) amortizado por operación, total O(n)
Determinar el número de pasajeros más frecuente	O(n), se aproxima a O(1).
Determinar la fecha más frecuente	O(n), se aproxima a O(1)
TOTAL	O(n)

Análisis

El algoritmo presenta un comportamiento lineal, ya que debe recorrer todo el catálogo de viajes para aplicar el filtro por rango de costos y realizar las operaciones de acumulación. Cada operación dentro del ciclo se ejecuta en tiempo constante, y el uso de diccionarios permite manejar eficientemente los conteos de frecuencias tanto de pasajeros como de fechas.

En la práctica, el tiempo de ejecución crece de manera proporcional al número de registros en el catálogo, lo cual resulta esperable dado que la tarea principal consiste en iterar sobre todos los viajes.

De esta forma, se pueden obtener estadísticas completas y precisas para distintos rangos de precios sin afectar el desempeño, manteniendo un comportamiento consistente con la complejidad lineal.

Requerimiento <<4>>

Descripción

Este requerimiento busca identificar la combinación de barrios origen–destino con el mayor o menor costo promedio de viaje, según el filtro indicado. Para ello, se recorren todos los viajes dentro de un rango de fechas, se calculan los promedios de costo, distancia y duración por cada par origen–destino válido y, finalmente, se selecciona la combinación que cumple con el criterio del filtro.

Entrada	Catálogo de viajes, filtro, fecha inicial, fecha final
Salidas	Diccionario con tiempo de ejecución, filtro aplicado, total de viajes considerados y la mejor combinación origen–destino con sus promedios de costo, distancia y duración.
Implementado (Sí/No)	Si se implementó y se hizo en grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Conversión de fechas inicial y final (`datetime.strptime`)	O(1)
Obtener tamaño de la lista de viajes (`lt.size`)	O(1)
Recorrer todos los viajes (`for i in range(size)`)	O(n)
Acceder a cada viaje (`lt.get_element`)	O(1) por acceso, total O(n)
Conversión de la fecha del viaje (`datetime.strptime`)	O(1) por viaje, total O(n)
Filtro por rango de fechas	O(1) por viaje, total O(n)
Determinar barrio más cercano (`find_nearest_neighborhood`)	O(n), donde n es el número de barrios en `catalog["neighborhoods"]`

Actualizar estadísticas en diccionario de combinaciones	O(1) amortizado, total O(n)
Recorrer las combinaciones origen–destino	O(n), donde n es el número de combinaciones distintas
Comparación según filtro (MAYOR/MENOR)	O(1) por combinación, total O(n(combinaciones))
TOTAL	O(n)

Análisis

El algoritmo presenta un comportamiento dominado por el producto del número de viajes y el número de barrios, debido a la función de búsqueda del barrio más cercano para cada origen y destino.
Cada viaje requiere calcular su barrio de origen y de destino en función de todos los barrios registrados, lo cual incrementa la complejidad en comparación con los requerimientos anteriores.

En la práctica, el tiempo de ejecución crece de forma proporcional al número de viajes y barrios. Sin embargo, el uso de diccionarios para acumular estadísticas por combinación origen–destino mantiene eficiente la gestión de datos y permite seleccionar rápidamente la mejor opción según el filtro aplicado.

Gracias a este diseño, es posible responder consultas sobre patrones de movilidad entre barrios en diferentes intervalos de tiempo, con resultados consistentes y útiles para el análisis de diferentes tendencias de costos en la ciudad.

Requerimiento <<5>>

Descripción

Este requerimiento permite identificar la franja horaria con el mayor o menor costo promedio, según el filtro indicado.
Para ello, se recorren todos los viajes en un rango de fechas, se agrupan por hora de inicio, y se calculan promedios de costo, duración y número de pasajeros. Además, se determina el viaje más caro y barato dentro de cada franja, con desempate por la fecha de finalización más reciente.

Entrada	Catálogo de viajes, filtro, fecha inicial, fecha final

Salidas	Diccionario con tiempo de ejecución, filtro aplicado, total de viajes considerados y la franja seleccionada con estadísticas y extremos de costo.
Implementado (Sí/No)	Si se implementó y se hizo en grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Conversión de fechas inicial y final (`datetime.strptime`)	O(1)
Obtener tamaño de la lista de viajes (`lt.size`)	O(1)
Recorrer todos los viajes (`for i in range(size)`)	O(n)
Acceder a cada viaje (`lt.get_element`)	O(1) por acceso, total O(n)
Conversión de fecha y hora del viaje (`datetime.strptime`)	O(1) por viaje, total O(n)
Filtro por rango de fechas	O(1) por viaje, total O(n)
Agrupar por franja horaria en diccionario	O(1) amortizado, total O(n)
Actualizar viaje más caro y barato (comparaciones)	O(1) por viaje, total O(n)
Recorrer las franjas horarias para elegir la mejor	O(n), maximo 24
TOTAL	O(n)

Análisis

El algoritmo presenta un comportamiento lineal, dado que necesita recorrer todos los viajes del catálogo dentro del rango de fechas para agruparlos en franjas y calcular las estadísticas correspondientes. Las operaciones internas de acumulación y comparación en cada franja son de tiempo constante gracias al uso de diccionarios.

En la práctica, el tiempo de ejecución crece de forma proporcional al número de viajes, sin verse afectado por el número de franjas, ya que este es limitado (24 horas). Esto permite obtener resultados eficientes incluso con catálogos muy grandes.

Gracias a esta estructura, se pueden analizar patrones horarios de la demanda y los costos de los viajes de forma rápida y precisa, facilitando la detección de los momentos más representativos según el criterio de filtro aplicado.

Requerimiento <<6>>

Descripción

Este requerimiento permite analizar los viajes que tienen como origen un barrio específico dentro de un rango de fechas determinado.
El algoritmo calcula promedios de distancia y duración de los viajes, identifica el barrio destino más visitado, y genera estadísticas por método de pago, indicando cuál fue el más usado y cuál generó mayores ingresos.

Entrada	Catálogo de viajes, barrio de origen, fecha inicial, fecha final
Salidas	Diccionario con tiempo de ejecución, número total de viajes desde el barrio, promedios de distancia y duración, destino más frecuente y lista de estadísticas por método de pago.
Implementado (Sí/No)	Si se hizo en grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Conversión de fechas inicial y final (`datetime.strptime`)	O(1)
Obtener tamaño de la lista de viajes (`lt.size`)	O(1)
Recorrer todos los viajes (`for i in range(size)`)	O(n)
Acceder a cada viaje (`lt.get_element`)	O(1) por acceso, total O(n)
Conversión de fecha y hora (`datetime.strptime`)	O(1) por viaje, total O(n)
Filtro por rango de fechas	O(1) por viaje, total O(n)
Identificación del barrio origen y destino (`find_nearest_neighborhood`)	O(n), se aproxima a O(1)
Acumulación de métricas (distancia, duración, totales)	O(1) por viaje, total O(n)
Actualización de contadores de destinos y métodos de pago (diccionarios)	O(1) amortizado, total O(n)
Determinar destino más visitado	O(n)
Determinar método de pago más usado y con mayor ingreso	O(n), se aproxima a O(1)
TOTAL	O(n)

Análisis

El algoritmo presenta un comportamiento lineal, ya que necesita recorrer todos los viajes del catálogo en el rango de fechas para identificar aquellos cuyo origen corresponde al barrio especificado. Durante el recorrido, las operaciones principales (acumulación de métricas, conteo de destinos y métodos de pago) se realizan en tiempo constante gracias al uso de diccionarios.

En la práctica, el tiempo de ejecución crece proporcionalmente con el número de viajes, mientras que los cálculos adicionales sobre destinos y métodos de pago no afectan de forma significativa el desempeño, ya que se realizan sobre conjuntos mucho más pequeños.

De esta manera, el algoritmo logra obtener de forma eficiente una visión detallada de los patrones de viaje desde un barrio específico, incluyendo destinos más visitados y comportamiento de pagos, manteniendo siempre una complejidad lineal.