**ANÁLISIS DEL RETO**

Juliana Rodríguez Morales – 202421552 – js.rodriguezm1234

Maria Clara Quijano - 202420069 - m.quijanoa

Juan Andrés Lozada - 202510410-j.lozadab

# **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 análisis de complejidad.

# **Requerimiento Ejemplo**

## **Descripción**



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.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Ya que, gracias a que los datos no se encuentran tan dispersos con respecto a la línea de tendencia, la curva coincide con el comportamiento lineal esperado.

## Texto El contenido generado por IA puede ser incorrecto.**Requerimiento 1**

## Texto El contenido generado por IA puede ser incorrecto.Imagen que contiene Interfaz de usuario gráfica El contenido generado por IA puede ser incorrecto.

## **Descripción**

Para poder dar la información que solicitaba el requerimiento 1 primero se empezó con la iteración de los viajes cargados previamente, de ahí se hizo un filtro donde solo se consideraran los viajes con la cantidad de pasajeros dado por el parámetro. Lo siguiente fue ir sumando los datos de cada viaje en las variables que pedía (como el tiempo, propina, costo, etc), al mismo tiempo se iba guardando por aparte la cantidad de veces que aparecía cada tipo de pago y cada fecha. Después de iterar se sacaban los promedios de las variables, y se hacían comparaciones para encontrar la fecha y tipo de pago más frecuente.

|  |  |
| --- | --- |
| **Entrada** | Catalog (donde están todos los trayectos); pasajeros (input dado por el usuario) |
| **Salidas** | Tiempo de ejecución. De los viajes con n pasajeros se muestra:  Total de trayectos, Tiempo promedio, Costo total promedio, Distancia promedio, Promedio de lo pagado en peajes, Nombre y cantidad del tipo de pago más usado, Promedio de propina, Fecha de inicio más recurrente. |
| **Implementado (Sí/No)** | Sí. Por Juliana Sofía Rodríguez Morales |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Paso 1. Asignación de variables | O(1) -> 9 veces |
| Paso 2. Iteración de todos los viajes en catalog | O(n) |
| Paso 3. Llamado funciones tamaño | O(1) |
| Paso 4. Comparación de # pasajeros | O(n) |
| Paso 5. Aumento de variables | O(n) |
| Paso 6. Llamado funciones is\_present, add\_last, change\_info | O(n^2) -> peor caso recorre la lista, add\_last puede redimensionar la lista (arr) |
| Paso 7. Comparación fechas y pago más usado | O(n) -> recorre |
| ***TOTAL*** | ***O(n^2)*** |

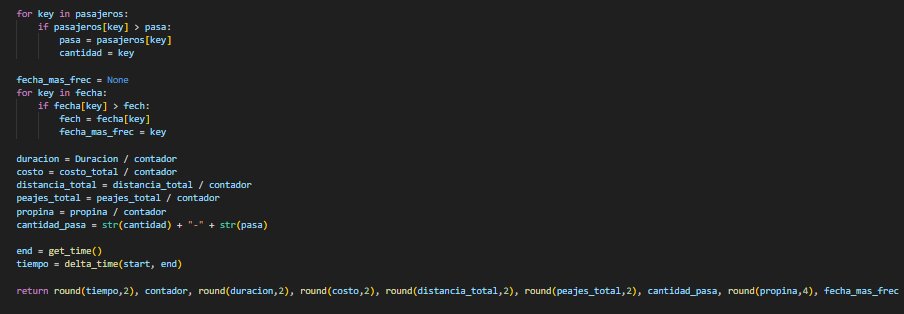
## **Análisis**

Aunque el código es en su mayoría O(n) debido a la iteración en el for de todos los viajes y por el uso de operadores simples como adición, comparación, y asignación; dentro del for se implementan funciones que cambian la complejidad del código. Las funciones hacen sus propios recorridos de listas con tamaño m, además de estar dentro del for; por ejemplo, el add\_last puede, en su peor caso, ser O(m) si la lista es un arraylist y está completa. La razón del porqué se toma como n^2 es porque m nunca es mayor a n sino que m <= n, pues son listas basadas en ella.

En sí, el comportamiento del código es cuadrático debido a los peores casos de las funciones y operaciones hechas.

# **Requerimiento 2**





### **Descripción**

Este requerimiento se encarga de filtrar los viajes según el método de pago y calcular estadísticas como duración promedio, costo, distancia, peajes, propina, cantidad de pasajeros más frecuente y fecha más común.

**Entradas:**

* Catálogo de viajes.
* Tipo de pago a filtrar (pago).

**Salidas:**

* Cantidad de viajes con ese pago.
* Promedio de duración.
* Promedio de costo, distancia, peajes y propina.
* Cantidad de pasajeros más frecuente.
* Fecha más común de finalización de viajes.

**Implementado (Sí/No):** Sí, implementado por Juan Andrés Lozada.

## **Análisis de complejidad**

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Recorrer la lista de viajes y filtrar por tipo de pago | O(n) |
| Calcular estadísticas de la lista filtrada | O(n) |
| Determinar pasajero y fecha más frecuentes | O(m), con *m* ≤ *n* |
| ***TOTAL*** | ***O(n)*** |

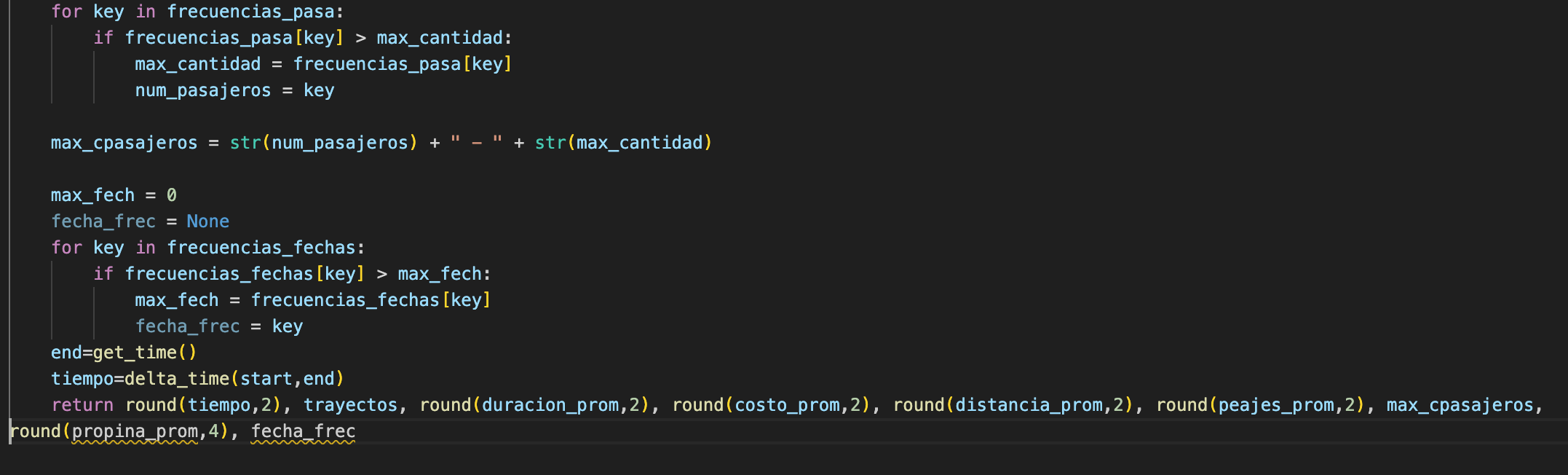
### **Análisis**

A pesar de que el requerimiento realiza varios cálculos (promedios, conteos de frecuencia y acumulaciones), todos dependen de recorrer los viajes una sola vez para filtrar y otra para acumular. El uso de diccionarios permite contar en tiempo constante promedio, por lo que no aumenta la complejidad.  
 El comportamiento es **lineal respecto al número de viajes** (O(n)), lo cual se refleja en un crecimiento proporcional al tamaño de los datos.

# **Requerimiento 3**

## **Descripción**





La función evalúa los trayectos que cumplen con el filtro de precios y luego se van sumando los valores que se tienen que sacar el promedio y retornar. Adicionalmente, se encuentra el número de pasajeros que es más frecuente en estos trayectos.

|  |  |
| --- | --- |
| **Entrada** | Catálogo de viajes, máxima cantidad que se paga por un viaje, y mínima cantidad que se paga por un viaje. |
| **Salidas** | * Tiempo promedio de la duración de los trayectos * precio total promedio de los trayectos * distancia promedio de los trayectos * precio promedio pagado en peajes de los trayectos * número y cantidad de pasajeros más frecuente en los trayectos * cantidad de propina promedio pagada en los trayectos * fecha de finalización de trayecto con mayor frecuencia. |
| **Implementado (Sí/No)** | Si. Implementado por Maria Clara Quijano |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

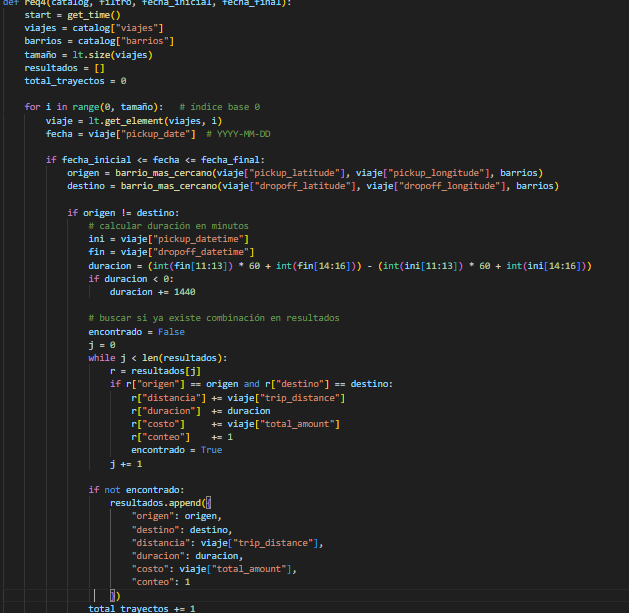
|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Recorrer lista y buscar si los elementos cumplen con el filtro | O(n) |
| Calculo Promedios | O(1) |
| Calcular las frecuencias de la cantidad de pasajeros | O(m) |
| Buscar fecha mas frecuente | O(p) |
| ***TOTAL*** | ***O(n) -> m y p en el peor caso son n*** |

## **Análisis**

La complejidad de la función depende del tamaño de la lista debido a que se deben recorrer todos los viajes para poder implementar el filtro. Adicionalmente, para encontrar las frecuencias, el peor caso que puede llegar a ser es n ya que esa es la cantidad de elementos en la lista. Los elementos que pasen el filtro son n o una cantidad menor.

# **Requerimiento 4**

### **Descripción**





Este requerimiento busca, en un rango de fechas dado, las combinaciones de barrios de origen y destino, acumulando distancia, duración y costo de los viajes. Finalmente, selecciona la combinación con **mayor o menor costo promedio** según el filtro indicado.

**Entradas:**

* Catálogo de viajes y barrios.
* Filtro (MAYOR o MENOR).
* Fecha inicial y fecha final.

**Salidas:**

* Trayecto con mayor/menor costo promedio.
* Distancia y duración promedio de ese trayecto.
* Cantidad total de trayectos considerados.

**Implementado (Sí/No):** Sí, implementado por Juan Andres Lozada Baragan.

## **Análisis de complejidad**

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Recorrer todos los viajes y verificar rango de fechas | O(n) |
| Determinar barrio más cercano (origen y destino) | O(k) k=num de barrios |
| Actualizar/insertar combinación en la lista de resultados | O(r), siendo r ≤ n |
| Seleccionar combinación con mejor costo promedio | O(r) |
| ***TOTAL*** | O(n·k + r) ≈ O(n·k) |

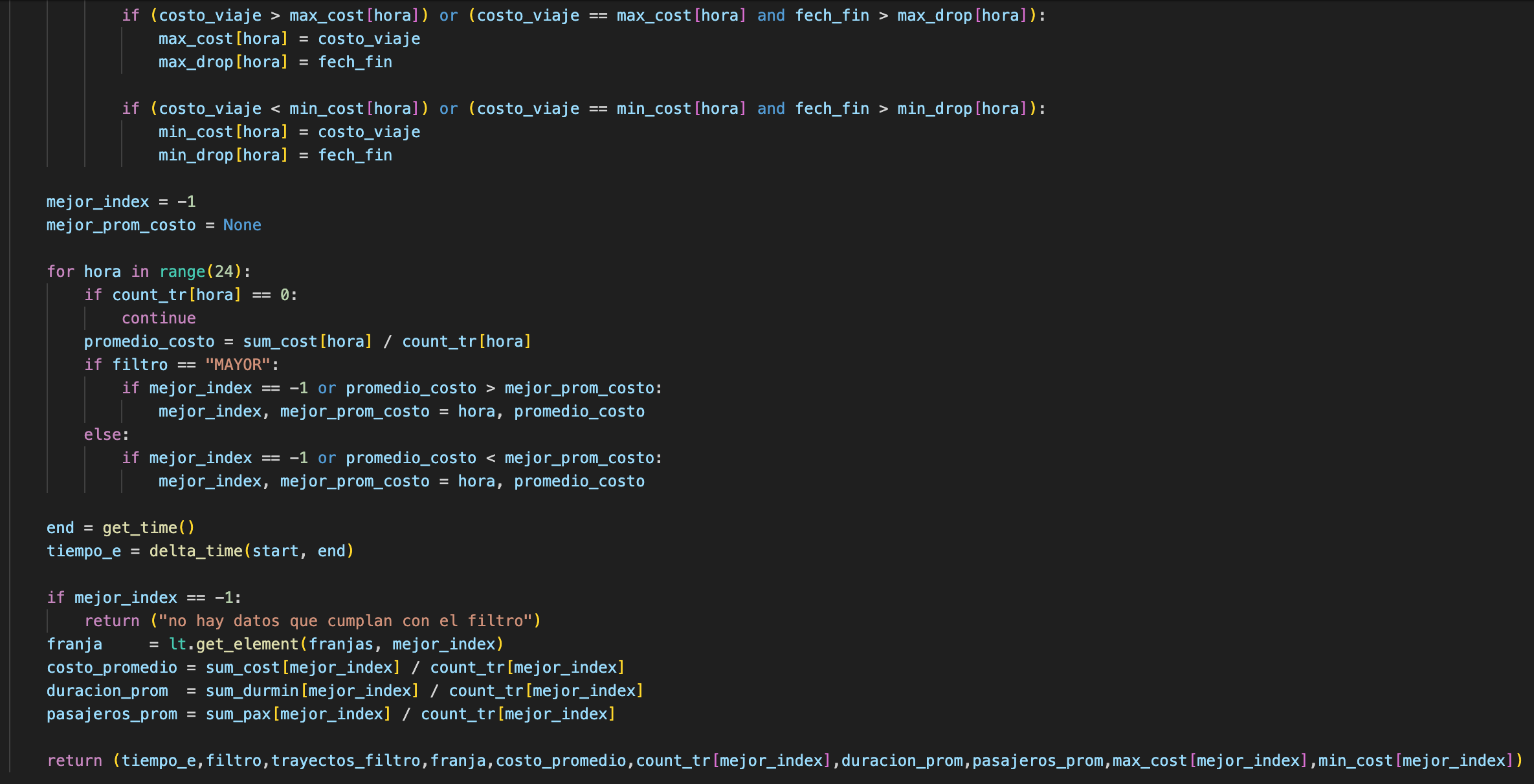
### **Análisis**

El paso más costoso es la búsqueda de barrios más cercanos, ya que para cada viaje se comparan coordenadas contra todos los barrios (*k*). Así, la complejidad total se aproxima a **O(n·k)**. La acumulación en resultados y la búsqueda de la mejor combinación aportan un costo adicional **O(r)**, pero dominado por el término principal.

En resumen, la complejidad depende linealmente del número de viajes y barrios, siendo más pesada que el requerimiento 2, pues involucra operaciones anidadas sobre las dos estructuras.

# **Requerimiento 5**

## **Descripción**



La función busca la franja de hora del día en el cual tiene un costo promedio mayor o menor, dependiendo del filtro que se busca. Retorna la información promedio de esa franja horaria.

|  |  |
| --- | --- |
| **Entrada** | Catálogo de viajes, filtro de selección del costo, fecha inicial del periodo a consultar, fecha final del periodo a consultar |
| **Salidas** | * Tiempo de la ejecución del requerimiento * Filtro de selección del costo * Número total de trayectos que cumplieron el filtro de fechas * Franja horaria * Costo promedio de los trayectos de la franja horaria * Número de trayectos incluidos en la franja horaria * Tiempo promedio de duración de los trayectos en la franja horaria * Cantidad promedio de pasajeros transportados por trayecto * Costo total del trayecto con mayor costo en la franja horaria * Costo total del trayecto con menor costo en la franja horaria |
| **Implementado (Sí/No)** | Si. Implementado por Maria Clara Quijano |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Asignación variables | O(1) |
| Recorrer viajes y verificar fechas | O(n) |
| Seleccionar la franja con mayor o menor costo promedio | O(1)-> se recorren 24 veces |
| ***TOTAL*** | ***O(n)*** |

## **Análisis**

Se tiene una complejidad O(n) por el filtro inicial que se le tiene que hacer a la lista, pero después se hace un recorrido 24 veces, una por cada franja de tiempo para seleccionar la franja que tiene mayor o meenor costo promedio, dependiendo de lo que le pidan. Los arreglos tienen un tamaño fijo (24), debido a que se guarda información para determinadas franjas de tiempo.

# **Requerimiento 6**

## **Descripción**

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.

## 

A partir de un barrio y un rango de fechas iniciales del inicio de trayecto se busca los trayectos que cumplen este criterio e información relevante para cada tipo de pago. La información que se presenta es el costo promedio pagado, los trayectos hechos con ese tipo de pago, mostrar el tipo de pago más recurrente y aquel con mayor recaudo.

|  |  |
| --- | --- |
| **Entrada** | Catalog con los datos de los viajes. Barrio, fecha inicial del rango y fecha final del rango, esto son los inputs dados por el usuario. |
| **Salidas** | Tiempo de la ejecución del requerimiento. Trayectos que cumplen con el barrio de inicio y su fecha de inicio está dentro del rango de fechas. Tiempo promedio de los trayectos que cumplen los criterios y el Nombre del barrio de destino más recurrente de los trayectos limitados.  Información para cada tipo de pago:   * Tipo de pago * Cantidad de trayectos * Precio promedio * Indicar si el tipo de pago fue el más recurrente * Indicar si el tipo de pago fue el que más recaudó * Tiempo promedio de trayectos |
| **Implementado (Sí/No)** | Sí. Implementado por Juliana Sofía Rodríguez Morales |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Paso 1. Asignación variables | O(1) |
| Paso 2. Iteración sobre todos los viajes | O(n) |
| Paso 3. Comparación de rango de fechas | O(n) |
| Paso 4. Llamado de función haversine y barrio cercano | O(m) -> Se recorre la lista de barrios y se compara con el punto de inicio y destino para saber barrio inicial y final. |
| Paso 5. Comparación barrio inicial con el barrio del parámetro | O(n) |
| Paso 6. Lista que muestra cuantas veces aparece cada barrio de destino | O(n^2) -> Igual que en el 1, se hace una lista para comparar donde k<= n y se toma como n^2 |
| Paso 7. Lista que muestra información para cada tipo de pago | O(n^2) -> Se hace otra lista r<=n |
| Paso 8. Recorrido de lista de barrios y tipo de pago para saber el más recurrente y el que más recaudó | O(n) -> Fuera del for, se toma como n por su peor caso |
| ***TOTAL*** | ***O(n^2\*m)*** |

## **Análisis**

Para poder conseguir los datos que se requerían en este punto se utilizaron 4 “listas” distintas, (1) la de los viajes que cumplían los criterios necesarios (sobre el barrio inicial y que estuviera dentro de los parámetros de la fecha inicial), (2) aquella que tenían los barrios y con el cuál se comparaba para poder encontrar el barrio de inicio y destino, (3 y 4) las listas que guardaban la información sobre cada tipo de pago y la información de los barrios más recurrentes. Como la mayoría estaba dentro del for, además de ser afectadas por orden de complejidad del for también se sumaba la complejidad de recorrer listas distintas. A diferencia de las listas hechas a partir de los viajes (como las 3 y 4) la lista de los barrios (2) no depende de los elementos de la lista de viajes (1). De esta forma se consigue una complejidad de n^2\*m.

La razón del porqué las listas 3 y 4 no se tomaron con sus k, r elementos es debido que, como dependen de la lista de viajes pueden ser menor o igual; en este caso, el peor caso es que tengan la misma cantidad de valores que los viajes, es decir, que todos los elementos sean únicos. La complejidad no sigue una forma común, pues es la multiplicación entre una cuadrática y una lineal.