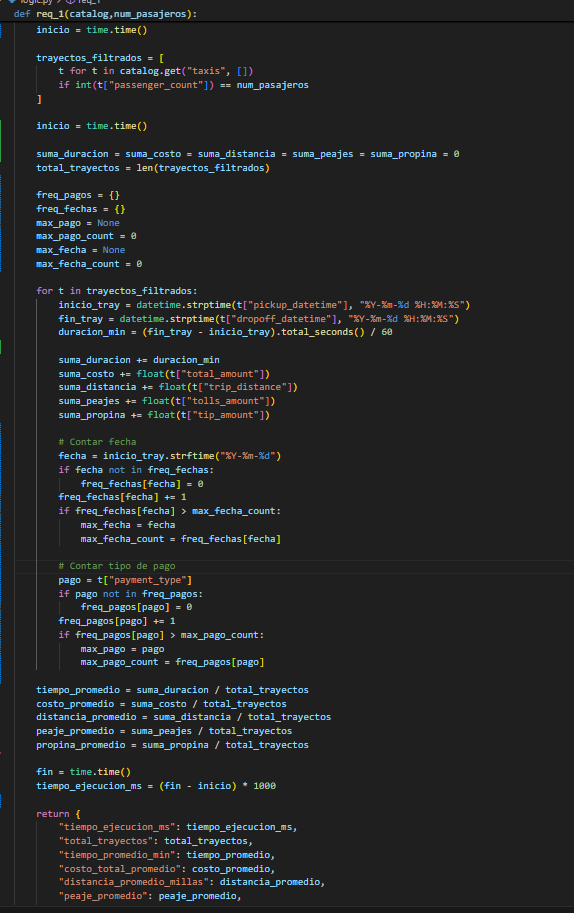
**ANÁLISIS DEL RETO**

Bastien Quentin Clement Thirion, 202525085, b.thirion@uniandes.edu.co

Jonathan David Galeano Sosa, 202226332, j.galeanos@uniandes.edu.co

# **Requerimiento 1**



## **Descripción**

Este requerimiento se encarga de filtrar los trayectos del catálogo según el número de pasajeros. Una vez filtrados, calcula estadísticas agregadas: duración promedio, costo promedio, distancia promedio, peajes y propinas promedio. Además, identifica el tipo de pago más usado y la fecha de inicio más frecuente de los trayectos.

|  |  |
| --- | --- |
| **Entrada** | Catalog : Catalogo con los trayetos cargados  num\_pasajeros : numero de pasajeros a filtrar |
| **Salidas** | Tiempo de ejecucion  Total de trayectos con ese número de pasajeros.  Promedio de duración, costo, distancia, peajes y propinas.  Nombre y cantidad del tipo de pago más usado.  Fecha más frecuente de inicio de trayectos. |
| **Implementado (Sí/No)** | Si, Bastien Thirion |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Filtrar trayectos por número de pasajeros | O(n) |
| Recorrer todos los trayectos filtrados | O(m) con m<n |
| Para cada trayecto: convertir fechas, calcular duración y acumular costos/distancias/peajes/propinas, frecuencias de tipo de pago y de fechas | O(1) |
| Calcular promedias y retornar resultados | O(1) |
| ***TOTAL*** | ***O(n)*** |

## **Análisis**

La implementación itera una sola vez sobre los trayectos (n elementos) y una sola vez sobre los trayectos filtrados, y en cada iteración todas las operaciones realizadas (suma de valores, conversión de fechas, actualización de diccionarios) tienen un costo constante O(1).

Por lo tanto, el algoritmo tiene un comportamiento lineal O(n), que se confirma experimentalmente: con catálogos grandes (500 000 trayectos), los tiempos de ejecución están en el orden de segundos, lo cual corresponde a un crecimiento lineal respecto al tamaño de entrada.

# **Requerimiento 2**

## **Descripción**

Analiza los trayectos filtrados por un método de pago específico, calculando promedios de duración, costo, distancia, peajes y propinas, además de identificar el número de pasajeros más frecuente y la fecha de finalización más común.

|  |  |
| --- | --- |
| **Entrada** | Catalog : Catalogo con los trayetos cargados  Metodo\_pago : Nombre del método de pago a filtrar |
| **Salidas** | tiempo\_ms: Tiempo de ejecución en milisegundos  total\_viajes: Cantidad de viajes con ese método de pago  duracion\_promedio\_min: Tiempo promedio en minutos  costo\_promedio: Promedio del costo total  distancia\_promedio: Promedio de distancia en millas  peajes\_promedio: Promedio de peajes pagados  propinas\_promedio: Promedio de propinas  pasajero\_mas\_frecuente: Número de pasajeros más frecuente con su frecuencia  fecha\_finalizacion\_mas\_frecuente: Fecha de finalización más común |
| **Implementado (Sí/No)** | Si, Jonathan |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Recorre toda la lista | O(n) |
| Recorre viajes en tamaño de x, siendo v <= n | O(x) |
| Calcular la moda de los datos | O(x) |
| Calculo de promedio | O(1) |
| ***TOTAL*** | ***O(n)*** |

## **Análisis**

La función trabaja efectivamente con datos de menor y mediano tamaño, esto gracias a que no son demasiado extensos, el cálculo de los promedios se hace más cómodamente al no tener que tener en cuenta muchos datos. Su complejidad de O(n) hace que cuando se manejan una gran cantidad de datos, la función pueda arrojar un tiempo de ejecución más uniforme y pronosticado en perspectiva de su entrada.

# **Requerimiento 3 no implementado**

# **Requerimiento 4**

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

## **Descripción**

Este requerimiento recibe un rango de fechas y busca identificar la combinación de barrios origen–destino con mayor o menor costo promedio de viaje.  
Para ello:

Se cargan los centroides de los barrios desde un archivo externo.

Se filtran los trayectos del catálogo que estén dentro del rango de fechas.

Para cada trayecto válido se determina el barrio de origen y destino más cercano (usando la fórmula de Haversine).

Se acumulan las métricas de distancia, duración y costo para cada par de barrios.

Finalmente, se calculan los promedios y se selecciona la combinación con mayor o menor costo promedio según el filtro dado.

|  |  |
| --- | --- |
| **Entrada** | catalog : catalogo con los trayectos de taxi  filtro : criterio de seleccion (“MAYOR” o “MENOR”)  fecha\_inicio\_str : fecha inicial en formato “YYYY-MM-DD”  fecha\_fin\_str : fecha final en formato “YYYY-MM-DD” |
| **Salidas** | Origen y destino seleccionados.  Distancia promedio (millas).  Tiempo promedio (minutos).  Costo total promedio (USD).  Número total de trayectos procesados.  Tiempo de ejecución en milisegundos. |
| **Implementado (Sí/No)** | Si, grupal |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Carga de centroides | O(m) con m el nombre de centroids |
| Recorrido de todos los trayectos del catalogo (n trayectos), operaciones clasicas y barrio\_mas\_cercano | O(n\*m) |
| Cálculo de promedios por combinación origen–destino | O(k) con k el nombre de promedios |
| Selección de la combinación con mayor/menor costo y retorno de resultados | O(k) |
| ***TOTAL*** | ***O(n\*m)*** |

## **Análisis**

El costo dominante está en la búsqueda del barrio más cercano para cada trayecto, que requiere recorrer todos los centroides (O(m)) dos veces (origen y destino).  
Por lo tanto, el tiempo crece de forma proporcional a n·m.  
Con catálogos grandes (ej. 500 000 trayectos) y ~30 barrios, el tiempo puede ser de varios segundos, lo cual se alinea con las pruebas experimentales (~35s en datasets grandes).

# **Requerimiento 5**

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

## **Descripción**

El requerimiento busca identificar la franja horaria del día (0–23 horas) en la que los trayectos de taxi presentan el costo promedio más alto o más bajo dentro de un rango de fechas dado.  
Para resolverlo, se recorren todos los viajes, se filtran por fecha de inicio y se agrupan por hora de recogida. Luego, se calculan estadísticas (promedio de costo, duración, pasajeros) y se selecciona la franja según el filtro indicado.

|  |  |
| --- | --- |
| **Entrada** | catalog : catalogo con los trayectos de taxi  filtro : criterio de seleccion (“MAYOR” o “MENOR”)  fecha\_inicio\_str : fecha inicial en formato “YYYY-MM-DD”  fecha\_fin\_str : fecha final en formato “YYYY-MM-DD” |
| **Salidas** | Tiempo de ejecución del requerimiento (ms).  Filtro aplicado ("MAYOR" o "MENOR").  Número total de trayectos considerados en el rango de fechas.  Para la franja horaria seleccionada:  Intervalo horario ([h – h+1)),  Costo promedio,  Número de trayectos,  Duración promedio,  Pasajeros promedio,  Costo máximo de un trayecto,  Costo mínimo de un trayecto. |
| **Implementado (Sí/No)** | Si grupal |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Recorrer todos los viajes del catálogo, Filtrar por rango de fechas , Determinar franja horaria y acumular datos en diccionario | O(n) |
| Calcular estadísticas por cada franja horaria (24) | O(1) |
| Seleccionar franja con mayor/menor costo promedio | O(1) |
| ***TOTAL*** | ***O(n)*** |

## **Análisis**

La implementación recorre los viajes una sola vez, filtrando por fechas y acumulando estadísticas en un diccionario de franjas (0–23). Como el número de franjas es fijo (24), los cálculos de promedios y la selección de la franja son operaciones en tiempo constante.

Esto garantiza que el algoritmo escala linealmente con el número de trayectos (O(n)), lo cual lo hace eficiente incluso con catálogos grandes (ej. cientos de miles o millones de trayectos).

En las pruebas realizadas, el tiempo de ejecución fue razonable (del orden de segundos para catálogos medianos). La salida cumple con todos los requisitos: tiempo, filtro aplicado, total de trayectos considerados y estadísticas completas de la franja horaria más/menos costosa.

# **Requerimiento 6**

## **Descripción**

Analiza trayectos que inician en un barrio dado dentro de un rango de fechas, calculando promedios de distancia y duración, el barrio destino más frecuente y las estadísticas de cada método de pago, destacando el más usado y el que más recaudó.

|  |  |
| --- | --- |
| **Entrada** | catalog : Lista de trayectos  neighborhoods: Lista de barrios con sus centroides  barrio\_inicio: Nombre del barrio de inicio  fecha\_inicial: Fecha inicial ("%Y-%m-%d")  fecha\_final: Fecha final ("%Y-%m-%d") |
| **Salidas** | total\_viajes: Cantidad de viajes filtrados.  distancia\_promedio: Distancia promedio de los viajes.  duracion\_promedio: Duración promedio en minutos.  barrio\_destino\_mas\_visitado: Barrio más frecuente como destino.  metodos\_pago: Lista con estadísticas por método de pago:  metodo: Nombre del método de pago.  cantidad\_trayectos: Número de trayectos con ese método.  promedio\_pago: Promedio de dinero pagado.  es\_mas\_usado: True si fue el método con más viajes.  es\_mas\_recaudo: True si fue el método que más dinero recaudó.  duracion\_promedio: Promedio de duración de trayectos con ese método. |
| **Implementado (Sí/No)** | Si, Jonathan |

## **Análisis de complejidad**

Análisis de complejidad de cada uno de los pasos del algoritmo

|  |  |
| --- | --- |
| **Pasos** | **Complejidad** |
| Pasar str a datetime | O(1) |
| Se recorre los viajes (n) a la vez que se revisan los barrios (m) y se recolecta los datos | O(n\*m) |
| Se recorren los destinos | O(n) |
| Análisis de métodos de pago | O(n) |
| ***TOTAL*** | ***O(n\*m)*** |

## **Análisis**

La función se comporta alrededor de O(n) aunque podríamos anotarlo como O(n\*m) donde m es la cantidad de barrios, los cuales en su mayoría no son una estadística muy grande que comprometa mucho la complejidad de la función, aunque si hay que tenerlo en cuenta. Con todos los tamaños de cantidad de datos se comporta bien siendo más tardío cuando los registros son cada vez mas numerosos teniendo que recorrer más aumentando su ejecución.

# **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.