# Análisis del reto 4

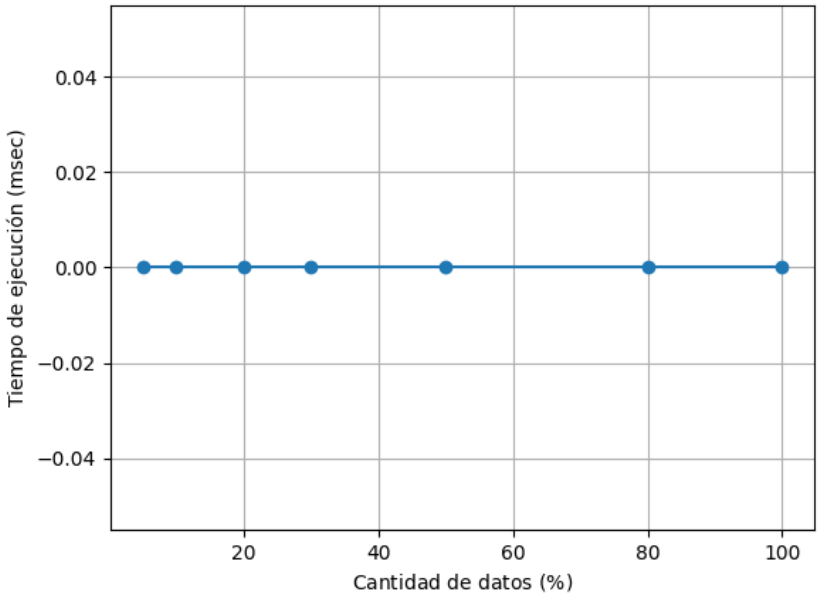
Estudiante 1: Sergio Iván Rincón, 201914107, si.rincon@uniandes.edu.co
Estudiante 2: Luis Ernesto Tejón, 202113150, l.tejon@uniandes.edu.co

Diciembre de 2021

## 1. Análisis de complejidad por requerimiento

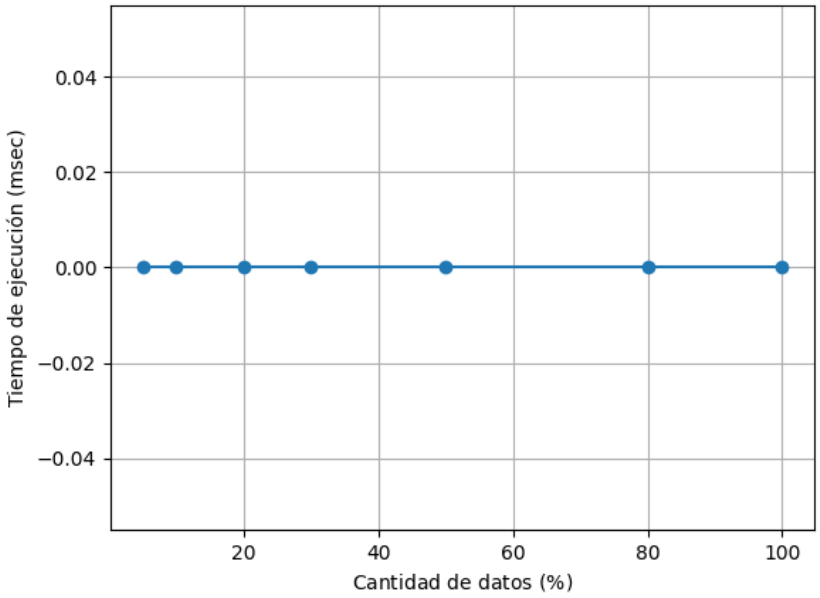
### 1.1. Requerimiento 1

- **Tipo de TAD usado:** Grafo, con ED lista de adyacencias, Map tipo linear probing
- **Carga de datos:** En la carga de datos se crea el grafo dirigido agregando cada uno de los aeropuertos y de las rutas como vértices y arcos del mismo. Además, se crea un mapa de aeropuertos con la información de los mismos. Al momento de agregar las rutas al grafo de aeropuertos se programo para que cada vez se cargue una ruta aunque esta ya esté una vez en el grafo y por tanto no se pueda agregar otra vez, esta igualmente sume 1 al outdegree del vértice de partida (vertexA) y también sume 1 al indegree del vértice de llegada (vertexB) de la ruta.
- **Descripción requerimiento:** En este caso, y para determinar los aeropuertos más interconectados, se empleó la función degree, pero gracias a la modificación que se le hizo a los indegree y outdegree, es función nos retornaría los valores de tal manera que nos indicaría el número de rutas asociadas al vértice y sin importar que algunas rutas se repitieran en el .csv por lo que solo estaban una vez en el grafo. Luego con esto, se buscaban los vértices que tenían más conexiones en el grafo. Para esto se recorren los vértices del grafo y se determina el resultado. Ahora bien, este recorrido se hace en la carga de datos, por lo que la consulta mantiene complejidad $O(1)$, pues en esta solo se consulta la información guardada en memoria durante la carga de datos.
- Complejidad de la consulta: $O(1)$
- Gráfica de pruebas temporales (tiempos reportados en milisegundos y cantidad de datos en porcentaje):




## 1.2. Requerimiento 2

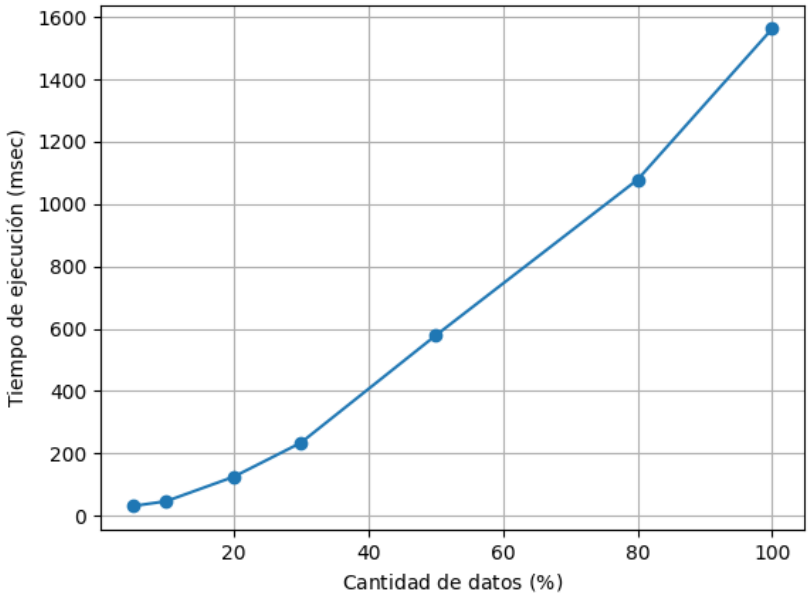
- **Tipo de TAD usado:** TAD graph con listas de adyacencia.
- **Carga de datos:** Para este requerimiento en la carga de datos se aplica el algoritmo de Kosaraju en el grafo dirigido.
- **Descripción requerimiento:** Para este requerimiento simplemente se consulta el número de componentes fuertemente conectados y si los dos aeropuertos dados por parámetro están en el mismo componente, esto con las funciones agregadas en el api para trabajar con el algoritmo de Kosaraju.
- Complejidad de la consulta: $O(1)$
- Gráfica de pruebas temporales (tiempos reportados en milisegundos y cantidad de datos en porcentaje):




### 1.3. Requerimiento 3

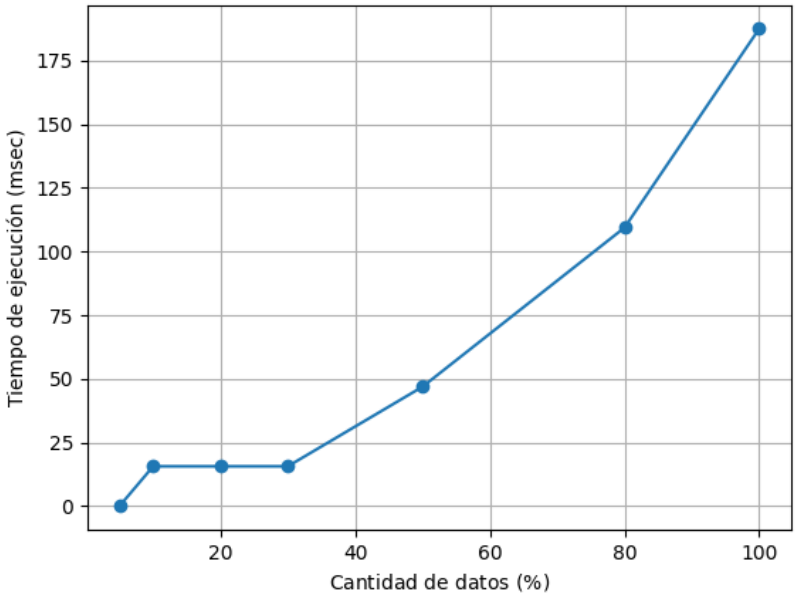
- **Tipo de TAD usado:** Graph, con ED lista de adyacencias, Ordered map tipo RBT, map tipo linear probing, lista tipo array list
- **Carga de datos:** En este requerimiento se crea un grafo dirigido con los aereopuertos y las rutas de los archivos CSV. Además, se genera un RBT organizando los aereopuertos por coordenadas y un Map con listas de ciudades.
- **Descripción requerimiento:** Este requerimiento tiene varios pasos para resolverse. En el primero, las ciudades ingresadas se buscan en un map de ciudades clasificadas por nombre. Cuando hay ciudades homónimas se le pregunta al usuario para que especifique la ciudad. Luego de que se encontró la ciudad, se empieza a buscar aereopuertos cercanos a la misma mediante cuadros de n*10 km de longitud. Cuando se encuentran aereopuertos, se determina la distancia entre las ciudades y los mismos mediante la fórmula brindada en las referencias del reto. Luego de determinar los aereopuertos más cercanos, se hace dijkstra en el grafo mediante el aereopuerto de origen y se determina la ruta de distancia mínima hacia el aereopuerto de destino.

  Ahora bien, la complejidad de la consulta en este caso está dictada por Dikstra, es decir que es $O(E \log V)$
- Complejidad de la consulta: $O(E \log V)$
- Gráfica de pruebas temporales (tiempos reportados en milisegundos y cantidad de datos en porcentaje):



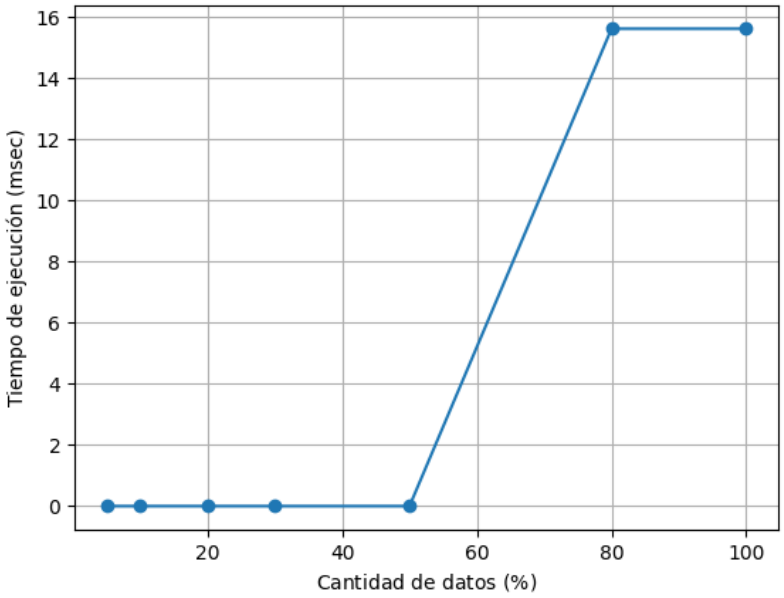

### 1.4. Requerimiento 4

- **Tipo de TAD usado:** Tad Graph
- **Carga de datos:** Para este requerimiento en la carga de datos se aplicó el algoritmo "Prim" de la librería con lo que se halló el árbol de recubrimiento mínimo, luego con este se creó un grafo no dirigido y sin ciclos con cada uno de los vértices y arcos del árbol de recubrimiento.
- **Descripción requerimiento:** Luego con este árbol se calculó haciendo uso de un dfs el mayor camino posible, para luego con esta información calcular el peso total del mayor camino posible, sabiendo esto se sabía entonces que para que el pasajero pudiera completar el camino tendría que pasar dos veces por cada arco, es decir tendría que tener el doble de millas que el peso del camino más largo en millas, por tanto para calcular cuántas millas le faltarían al pasajero para poder hacer todo el camino simplemente se le restó las millas del pasajero al costo total del camino más largo multiplicado por 2.
- Complejidad de la consulta: Tomando $V$ y $E$ como los vértices y arcos del camino más largo y **no** del grafo, la complejidad del dfs es $O(V + E)$ mientras que la de la iteración que se hizo para encontrar el costo del camino más largo es $O(V)$ dando una complejidad final de $O(2V + E)$
- Gráfica de pruebas temporales (tiempos reportados en milisegundos y cantidad de datos en porcentaje):




### 1.5. Requerimiento 5

- **Tipo de TAD usado:** Grafo tipo lista de adyacencias y map
- **Carga de datos:** En este caso se emplean las mismas estructuras que se usaron en el req 1, por lo que la carga de datos es la misma
- **Descripción requerimiento:** Para resolver este requerimiento se empleó un grafo dirigido. Mediante este grafo, se determinaron los vértices adyacentes al aereopuerto que se iba a cerrar. De esta manera, estos vértices adyacentes corresponderían a los aereopuertos afectados. Adicionalmente, se determinaron los vértices que tenían rutas que terminaban en el grafo, de tal manera que estos aereopuertos también se pudiesen identificar. Luego de esto, y gracias a la identificación, se buscaba la info de los aereopuertos en un map y se daba respuesta al requerimiento. En este caso, si hubiese un vértice con conexiones a todos los demás en el grafo, se tendrían que identificar todos los vértices, por lo que la complejidad es O(n). Sin embargo, en promedio la complejidad es mucho menor.
- Complejidad de la consulta: $O(n)$
- Gráfica de pruebas temporales (tiempos reportados en milisegundos y cantidad de datos en porcentaje):




# 2. Datos adicionales

## 2.1. Ambientes de prueba

Los requerimientos se probaron en la máquina 1 y en la máquina 2. En el documento se reportan los tiempos encontrados en la máquina 1.

| | **Máquina 1** | **Máquina 2** |
|---|---|---|
| **Procesadores** | AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx 2.10 GHz | Ryzen 5 5600g CPU @ 3.9GHz, 4.4GHz |
| **Memoria RAM (GB)** | 16GB | 12GB |
| **Sistema Operativo** | Windows 10 home 64-bits | Windows 11 home 64-bits |

## 2.2. Tiempos de la carga de datos (en milisegundos):