# **Observaciones del reto 4**

Ernesto Perez – 202112530 – ec.perez@uniandes.edu.co

Nicolás Saavedra – 20212963 - [n.saavedrag@uniandes.edu.co](mailto:n.saavedrag@uniandes.edu.co)

# <https://github.com/EDA2021-2-SEC01-G07/Reto4--G07>

# **Análisis de complejidad**

# **Pruebas de rendimiento**

Especificaciones de la máquina de prueba

|  |  |
| --- | --- |
|  |  |
| Procesadores | AMD Ryzen 5 3600 6-Core Processor, 3593 Mhz, 6 Core(s), 12 Logical Processor(s) |
| Memoria RAM | 16GB |
| Sistema operativo | Microsoft Windows 11 pro 64-bits |

**Tiempo de ejecución promedio por requerimiento y crecimiento temporal**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Tiempo de ejecución para cada función [ms]** | | | | | |
| **Porcentaje de datos** | **Cargar datos** | **Req1** | **Req2** | **Req3** | **Req4** | **Req5** |
| **10%** | 1109.38 | 15.63 | 0.0 | 31.25 | 203.13 | 0.0 |
| **30%** | 1984.38 | 46.88 | 0.0 | 171.88 | 1218.75 | 0.0 |
| **50%** | 3515.63 | 78.13 | 0.0 | 359.38 | 3062.5 | 0.0 |
| **80%** | 7953.13 | 156.2 | 0.0 | 812.5 | 5140.625 | 0.0 |
| **100%** | 12031.25 | 187.5 | 0.0 | 1187.5 | 10953.13 | 0.0 |

*En las funciones con entradas se usaron los mismos datos de entrada de los ejemplos.*

**Graficas de tiempo de ejecución promedio por requerimiento y crecimiento temporal**

**Complejidad temporal carga de datos:** O (n)

Para cargar los datos se debe recorrer ambos archivos y por cada avistamiento meterlo a su respecto índice pero nunca se recorrer más de una vez el archivo.

**Complejidad temporal carga de datos:** O (nlog(n))

Las operaciones con mayor complejidad del código son O(V) para sacar los vertices del grafo, O(V+E) para las operaciones indegree y outdegree y el merge sort que se hace para obtener los aeropuertos con mayor interconexión lo cual tiene una complejidad de O(n(log(n)) por lo que este toma prioridad al ser el de mayor complejidad, en este caso n corresponde a los vertices que tienen al menos una salida o entrada por lo que n es un valor muy cercano a V.

**Complejidad temporal carga de datos:** O(1)

Durante la carga, hacemos el algoritmo de Kosaraju, con complejidad O(k). Esto significa que al ejecutar el requerimiento 2, tenemos toda la información sobre los componentes fuertemente conectados y solamente se tiene que sacar un valor de un diccionario, lo cual es O(1) por definición.

**Complejidad temporal carga de datos:** O (E + log(V)) (E: edges en grafo, V: vertices en grafo )

Los dos procesos que consumen más recursos en el grafo son cityToAirport y Dijsktra. Sin embargo, el promedio de intentos y de tiempo de búsqueda para encontrar una ciudad usando el arbol es mucho menor a la cantidad de edges en el grafo, por lo cual, dijisktra tiene prioridad en complejidad, y toma la complejidad total del requerimiento. Complejidad de O(E+V) que es menor por lo que el algoritmo de prim tiene prioridad.

**Complejidad temporal carga de datos:** O(E\*log(V))

Para este requerimiento se requiere hacer un arbol de expansion minima sobre el grafo no dirigido para lo cual se usa el algoritmo prim (eager) lo cual tiene una complejidad de O(E\*log(V)) siendo E los arcos y V los vertices, se hace tambien un DFS, sin embargo, esta operación tiene una complejidad de O(E+V) que es menor por lo que el algoritmo de prim tiene prioridad.

**Complejidad temporal carga de datos:** O(K) (K siendo cantidad de aeropuertos afectados)

El requerimiento solamente necesita los aeropuertos, y esos se pueden encontrar en los sets y mapa. Todas estas operaciones son O(1), sin embargo, pasando valores de un tipo de lista a otra es O(K), por lo cual el rendimiento total es O(K)