OBSERVACIONES RETO 4

Samuel Alejandro Jiménez Ramírez – 20211665
Marilyn Stephany Joven Fonseca – 202021346

Ambiente de pruebas

	Máquina 1	Máquina 2
Procesadores	Intel® Core™ i5 - 10310U CPU @ 1.70 GHz 2.21GHz 64 bits	AMD Ryzen 5 4500U Radeon Graphics 2.38GHz
Memoria RAM (GB)	16 GB (15,6 utilizable)	16 GB
Sistema Operativo	Windows 10 Pro	Windows 10 Home

Tabla 1. Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.

Resultado de pruebas

Para tomar el tiempo promedio por ejecución de cada requerimiento acá enlistado, usamos el archivo con terminación **large**. Tomamos 3 muestras de cada requerimiento y sacamos la media aritmética que es la que está escrita en la tabla.

Para la creación del catálogo, decidimos usar dos diferentes tipos de grafos, el dirigido y el no dirigido ya que primero: el dirigido en el que guardamos todas las rutas que tenía el archivo, y el no dirigido sirve para guardar arcos donde sepamos que hay dos arcos (uno de ida y otro de regreso) entre los dos vértices.

También cabe aclarar que los vértices de ambos grafos estaban conformados por los aeropuertos que entraban del primer archivo.

Finalmente, también usamos estructuras de apoyo como mapas ordenados, listas y tablas de hash, los tipos de estas fueron respectivamente: rbt, para que no tuviéramos retrasos a la hora de correr requerimientos, ya que la complejidad de este tipo está al momento de crear el catálogo; arreglos, para hacer recorridos más rápidos y escoger elementos nada más con el index y que sea eficiente; y linear probing, para que fuera más sencillo el manejo de datos en las tablas de hash.

```
catalog['IATAS'] = mp.newMap(numelements=14000,
                             maptype='PROBING',
                             comparefunction=compareIATA)

catalog['AN-ID'] = mp.newMap(numelements=14000,
                             maptype='PROBING',
                             comparefunction=compareIATA)

catalog['routes'] = gr.newGraph(datastructure='ADJ_LIST',
                                directed=True,
                                size=14000,
                                comparefunction=compareIATA)
catalog['connected'] = gr.newGraph(datastructure='ADJ_LIST',
                                directed=False,
                                size=14000,
                                comparefunction=compareIATA)
catalog["rama"] = gr.newGraph(datastructure='ADJ_LIST',
                                directed=False,
                                size=14000,
                                comparefunction=compareIATA)

catalog["cities"] = lt.newList(datastructure="ARRAYLIST")
catalog["path"] = mp.newMap(numelements=100000,maptype="LINEAR_PROBING",loadfactor=0.95)
catalog["salida"] = lt.newList()
catalog['repeat'] = lt.newList()
catalog['cities2'] = mp.newMap(maptype="PROBING", numelements= 41002)
catalog['withroutes'] = lt.newList(datastructure="SINGLE_LINKED")
catalog['repeatmap'] = mp.newMap(maptype="PROBING", comparefunction=compareIATA)
catalog['latitudeairports'] = om.newMap(omaptype="RBT")
catalog["longitudeairports"] = om.newMap(omaptype="RBT")
```

Para las muestras no se va a medir la memoria utilizada puesto que esta es constante debido a la carga de datos que se realiza al comienzo del programa, y nunca se elevó a valores preocupantes que afectaran el buen funcionamiento de la máquina. Mismo caso que ocurrió con el procesador, nunca subió en ninguna de las dos máquinas a más del 50%.

Cabe destacar que el tiempo que se demora guardando datos de los archivos cvc con este tipo de estructuras de datos es de **53,51 s** en la **Máquina 1,** y de **47,59 s** para la **Máquina 2**.

	Tiempo promedio Máquina 1 (s)	**Tiempo promedio Máquina 2 (s)**
Requerimiento 1	0,15	0,17
Requerimiento 2	4,24	3,78
Requerimiento 3	1,85	2,20
Requerimiento 4	9,05	7,03
Requerimiento 5	0,125	0,03

Análisis complejidad temporal de cada requerimiento

Requerimiento 1:

Para el requerimiento 1 tenemos una complejidad de **O(n)** donde n es la cantidad de aeropuertos que tienen así sea una sola ruta en el grafo, ya sea de salida o de llegada. Esta complejidad se deduce debido a que se tiene que hacer un recorrido a esta lista para sacar los vértices con mayor influencia en la red aérea. Se hacen otros recorridos y otras operaciones, pero estas son despreciables para la muestra.

Requerimiento 2:

Para el requerimiento 2 tenemos una complejidad de **O(E+V)** donde E es la cantidad de arcos del grafo, y V la cantidad de vértices, esta complejidad nos la devuelve el algoritmo de Kosaraju que es el utilizado para resolver este problema. Se hacen más de una operación desde la librería de Kosaraju pero estas complejidades son menores o despreciables.

Requerimiento 3:

Para el requerimiento 3 tenemos una complejidad de **O(E log V)** donde E es la cantidad de arcos del grafo, y V la cantidad de vértices, esta complejidad nos la devuelve el algoritmo de Dijkstra que es el que utilizamos para obtener la ruta más óptima en cuanto a peso. Se hacen muchas más operaciones, pero estas son menores para la toma de complejidad o despreciables.

Requerimiento 4:

Para el requerimiento 4 tenemos una complejidad de **O(E log V)** donde E es la cantidad de arcos del grafo, y V la cantidad de vértices, esta complejidad nos la devuelve el algoritmo de **prim** para sacar el **MST,** más adelante se hace un **DFS** pero es con una muestra menor que no afecta la complejidad. Sin embargo, en la práctica vemos que este es más demorado por la cantidad de operaciones que tiene dentro, aunque estas son menores o despreciables.

Requerimiento 5:

Para el requerimiento 5 tenemos una complejidad de **O(1)** que es retornada por la operación de sacar los adyacentes a un vértice, se hacen recorridos pero son pequeños y constantes así que no cambia esta complejidad.

Requerimiento 6:

Para la complejidad del requerimiento 6 tenemos la misma situación que en el requerimiento 3: **O(E log V),** donde E es la cantidad de arcos del grafo y V es la cantidad de vértices, esta complejidad nos la retorna el algoritmo Dijkstra. Puede tener mayores complejidades en el API pero al ser este desconocido no las tomaremos en cuenta.

Requerimiento 7:

Para el requerimiento 7 se puede decir que tenemos todas las complejidades del requerimiento 1 al requerimiento 5 sumadas, ya que este las ejecuta para graficarlas, puede tener mayor complejidad dentro de la visualización, pero estas son desconocidas para la toma de muestra.

Conclusiones

- La creación del catálogo se torna más pesada por todos los tipos de estructuras que se manejan al tiempo, sin embargo, siempre se trató de hacer lo más eficientemente posible sin afectar el tiempo de ejecución de los requerimientos que era lo más importante.
- En comparación con los otros retos, esta vez el procesador y el uso de memoria si subieron un poco, pero tampoco nunca se llegó a un número preocupante o que afectara el funcionamiento de la máquina.
- Es más fácil trabajar los algoritmos en un grafo que en cualquier otra estructura exceptuando las listas, lo que se podría considerar complejo es el entendimiento de estos para saber cuándo ejecutarlos y cómo.
- Se logró analizar en qué caso utilizar los algoritmos dados para cualquier tipo de recorrido o información que se necesitara del grafo.