Reto No. 4: Retomando los Aires

Documento de Análisis:

Participantes del Grupo:

Nicolas Yesid Rivera Lesmes ny.rivera@uniandes.edu.co 2021166756

Santiago Gustavo Ayala Ciendua s.ayalac@unaindes.edu.co 202110734

Evaluar complejidad:

Requerimiento 1: O (n)

El requerimiento solo tiene un ciclo grande que es por la lista de vertices como se ve en la foto de abajo, por eso su complejidad es de O(n), esto ya que las funciones de degree son solo de complejidad O(1). Asi que al final queda con complejidad O(n)

```
def CuantasConexionesTiene(graf_dir):

    lista_vert = gr.vertices(graf_dir)
    lista_d = lt.newList("ARRAY_LIST")

    for elemento in lt.iterator(lista_vert):

        num1 = gr.outdegree(graf_dir, elemento)
        num2 = gr.indegree(graf_dir, elemento)
        num = num1 + num2

        dicit = {"aero": elemento, "conexiones": num}

        lt.addLast(lista_d, dicit)

    r = ms.sort(lista_d, CmpNum)

    return lista_d
```

Requerimiento 2: O (n):

El requerimiento dos utiliza funciones brindadas por la librería.

```
def req2 (analyzer,codigo1,codigo2):

    analyzer["com_graph_directed"] = scc.KosarajuSCC(analyzer["conexiones_dir"])
    conected = scc.stronglyConnected(analyzer["com_graph_directed"],codigo1,codigo2)
    num_compo = scc.connectedComponents(analyzer["com_graph_directed"])

    return num_compo,conected
```

En cuanto a la función que utiliza el algoritmo de Kosaraju es asintóticamente óptima y lineal teniendo una complejidad de Θ (V + E).

En cuánto a las otras funciones no se registró en la documentación una complejidad exacta. Sin embargo, si estas funciones se basan en el algoritmo de componentes fuertemente conectados de Tarjan su complejidad sería lineal.

Requerimiento 3: O (n)

El requerimiento 3 utiliza dos elementos que tienen complejidad de O(n), el primero es el algoritmo de Dijkstra que su complejidad real es (E + V) y como son dos constantes seria como O(n)

```
def CaminoCortoCiudades(origen, destino, graf_dir):

    search = dj.Dijkstra(graf_dir, origen)
    path = dj.pathTo(search, destino)

    return path
```

Y el otro elemento que tendira complejidad O(n) seria un ciclo como este que se utilizaba para vincular la ciudad y el aeropuerto.

```
def EncontrarAeropuertoIda(origen_final, hash_aero):

    lista_for = mp.keySet(hash_aero)

    lim_min_lat = float(origen_final["lat"]) - 0.1
    lim_max_lat = float(origen_final["lat"]) + 0.1
    lim_max_long = float(origen_final["long"]) + 0.1
    lim_min_long = float(origen_final["long"]) - 0.1

    aeropuerto = ""

    while aeropuerto == "":

        for element in lt.iterator(lista_for):

            o = mp.get(hash_aero, element)["value"]
            for e in lt.iterator(o):

                if float(e["latitud"]) <= lim_max_lat and float(e["latitud"]) >= lim_min_lat and float(e["longitud"]) <= lim_max_long and flo
                    print(1)
                    aeropuerto = e["IATA"]

        lim_min_lat = lim_min_lat - 0.1
        lim_max_lat = lim_max_lat + 0.1
        lim_max_long = lim_max_long + 0.1
        lim_min_long = lim_min_long - 0.1

    return aeropuerto
```

A pesar de ser un ciclo dentro de otro como se ve en la foto, la complejidad no sube de O(n) por dos razones, 1 el while es finito y cortaria todo y 2 porque el segundo while es muy limitado ya que solo revisa las ciudades homonimas que son escasas.

Requerimiento 4: O (n^2):

A pesar de que puede ser una complejidad O(n^2) muy sutil, esta es su complejidad debido a que se usa el algoritmo dfs, y este su complejidad es O(b^2), b siendo el numero de ramificaciones por nodo promedio, entonces tomariamos es b como n y la complejidad queda O(n^2).

```
def MstPrim(graf_nodir, ciudad_org, hash_ae):

    Ae = mp.get(hash_ae, ciudad_org)["value"]
    ly = ""

    for element in lt.iterator(Ae):

        ly = element["IATA"]


    #mst1 = pr.PrimMST(graf_nodir)
    #mst2 = pr.edgesMST(graf_nodir, mst1)

    defs = df.DepthFirstSearch(graf_nodir, ly)
    #difs = df.dfsVertex(defs, graf_nodir, ly)


    return defs
```

Requerimiento 5: O (1):

Esta funcion no realiza ciclos de ningun tipo y solo utiliza funciones de la estructura como gr.degree o gr.adjacents las cuales son O(1) y se ven las funciones en la foto a continuacion:

```
def SaberConectados(graf_dir, inicio):

    lista_vertex = gr.adjacents(graf_dir, inicio)

    return lista_vertex

def CuantosAfectados(graf_dir, inicio):

    num1 = gr.outdegree(graf_dir, inicio)
    num2 = gr.indegree(graf_dir, inicio)
    num = num1 + num2

    return num
```

Ambientes de pruebas

	Máquina 1	Máquina 2
Procesadores	Intel® Core™ i5-9300H CPU @ 2.4GHz	Intel® Core™ i5-8250U CPU @ 3.7GHz
Memoria RAM (GB)	8 GB	8 GB
Sistema Operativo	Windows 10 Pro-64 bits	Windows 11 Pro-64 bits

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

Maquina 1

Resultados (3 toma de datos)

Tamaño de la muestra	Req 1 Tiempo (mseg)	Req 2 Tiempo (mseg)	Req 3 Tiempo (mseg)	Req 4 Tiempo (mseg)	Req 5 Tiempo (mseg)	Req 6 Tiempo (mseg)
DATOS SMALL	15.625	31.25	93.75	0.0	0.0	
DATOS 5pct	31.25	46.875	93.75	15.625	0.0	
DATOS 10 pct	46.875	78.125	140.625	46.875	0.0	
DATOS 20 pct	93.75	187.5	250.0	171.875	0.0	
DATOS 30 pct	93.75	375.0	296.875	468.75	0.0	
DATOS 50 pct	140.625	718.75	531.25	968.75	0.0	
DATOS 80 pct	140.625	781.25	578.125	984.375	0.0	
DATOS LARGE	203.125	1984.375	1406.25	3234.375	0.0	

Tamaño de la muestra	Req 1 Memoria	Req 2 Memoria	Req 3 Memoria	Req 4 Memoria	Req 5 Memoria	Req 6 Memoria	
DATOS SMALL							
DATOS 5pct							

DATOS 10 pct	Muy poca variabilidad para medir – No hay tiempo suficiente para que se registre un uso extensivo de la memoria Siempre se mantuvo entre los valores de 6.00G y 6.20G incluso al cambiar el tamaño de los archivos
DATOS 20 pct	
DATOS 30 pct	
DATOS 50 pct	
DATOS 80 pct	
DATOS LARGE	

Maquina 2

Resultados (3 tomas de datos).

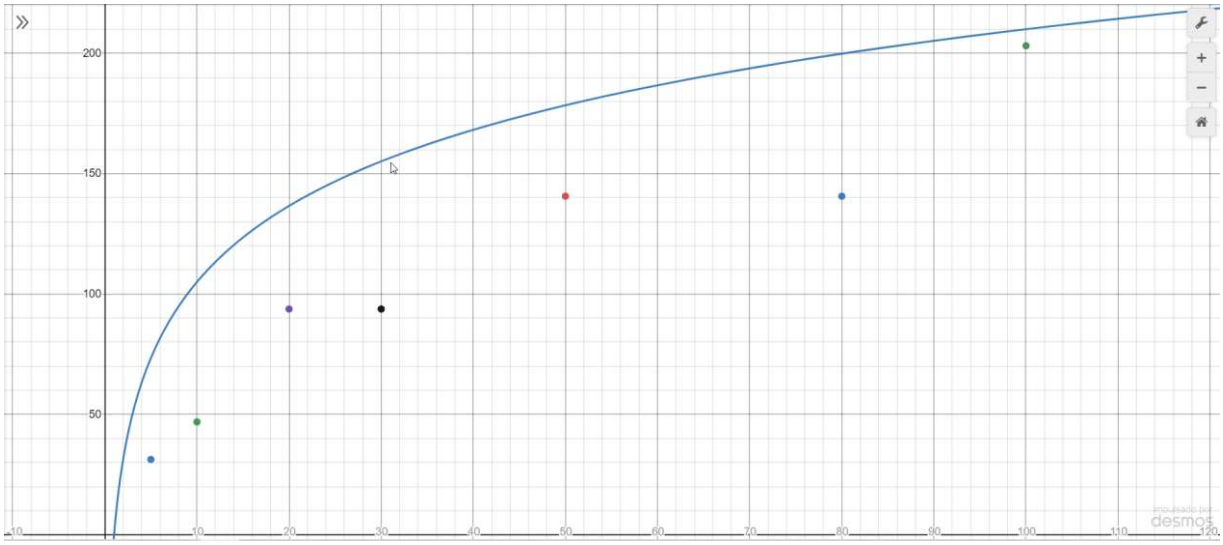
Tamaño de la muestra	Req 1 Tiempo (mseg)	Req 2 Tiempo (mseg)	Req 3 Tiempo (mseg)	Req 4 Tiempo (mseg)	Req 5 Tiempo (mseg)	Req 6 Tiempo (mseg)
DATOS SMALL	46.875 mseg	31.25 mseg	109.375 mseg	0.0 mseg	0.0 mseg	
DATOS 5pct	46.875 mseg	46.875 mseg	93.75 mseg	15.625 mseg	15.625 mseg	
DATOS 10pct	46.875 mseg	125.0 mseg	156.25 mseg	46.875 mseg	15.625 mseg	
DATOS 20 pct	62.5 mseg	187.5 mseg	328.125 mseg	218.75 mseg	0.0 mseg	
DATOS 30pct	93.75 mseg	437.5 mseg	421.875 mseg	625.0 mseg	0.0 mseg	
DATOS 50pct	140.625 mseg	843.75 mseg	687.5 mseg	1187.5 mseg	0.0 mseg	
DATOS 80pct	218.75 mseg	1578.125 mseg	1343.75 mseg	2531.25 mseg	0.0 mseg	
LARGE	281.25 mseg	2343.75 mseg	2015.625 mseg	4031.25 mseg	0.0 mseg	

Tamaño de la muestra	Req 1 Memoria	Req 2 Memoria	Req 3 Memoria	Req 4 Memoria	Req 5 Memoria	Req 6 Memoria	
DATOS SMALL							

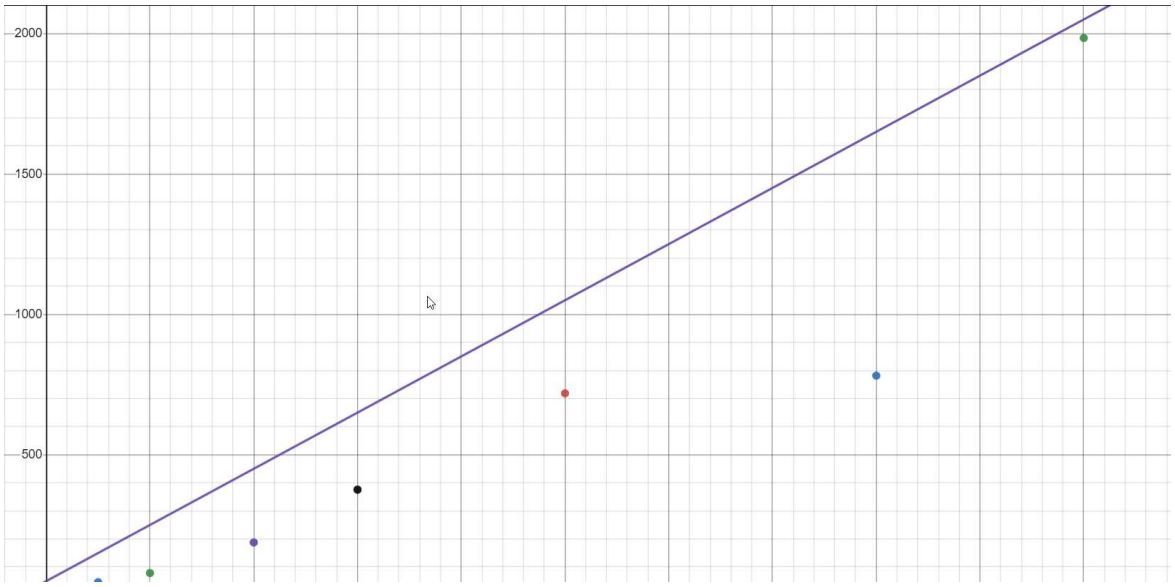
DATOS 5pct	Muy poca variabilidad para medir – No hay tiempo suficiente para que se registre un uso extensivo de la memoria
DATOS 10 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 5.85 - 5.86 G
DATOS 20 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 5.95 - 6.03 G
DATOS 30 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 6.19 - 6.26 G
DATOS 50 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 6.33 - 6.37 G
DATOS 80 pct	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 6.46 - 6.71 G
DATOS LARGE	Durante lo que corrió el programa después de un reposo, la memoria oscilo entre el siguiente rango: 6.81 - 6.94 G

Gráficas Generales:

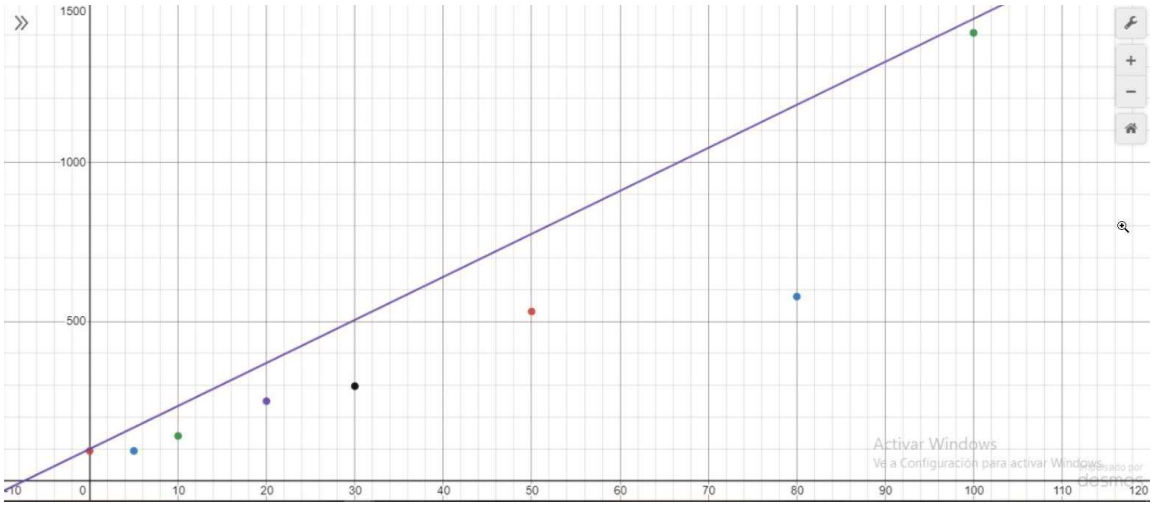
Req1:



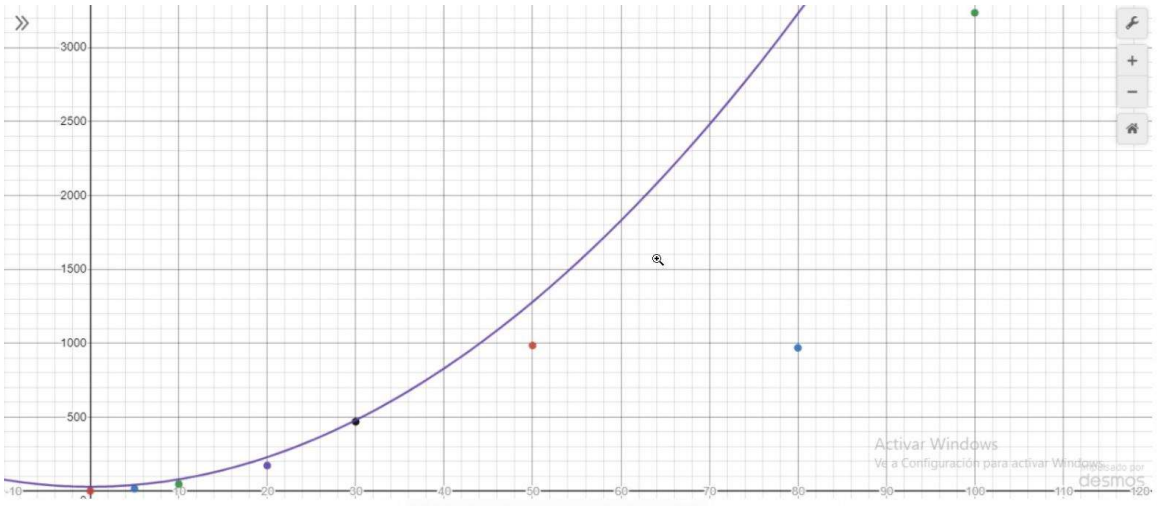
Req2:



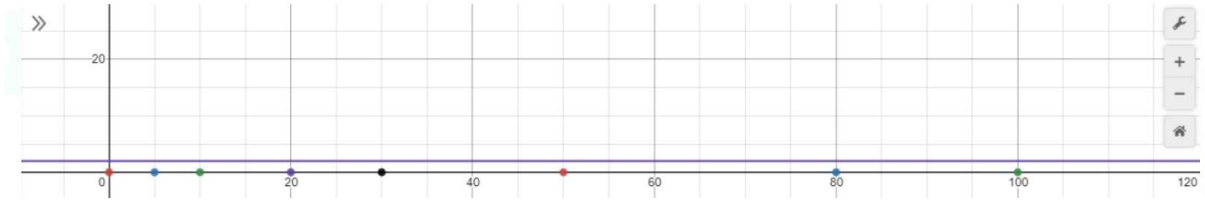
Req 3:



Req 4:



Req 5:



Preguntas de análisis

1. ¿El comportamiento de los algoritmos es acorde a lo enunciado teóricamente?

Analizando en todas hubo un rendimiento aceptable y hasta quizas bueno en general , en específico el requerimiento 5 se destaca al tener complejidad O(1) y si alguno está bajo lo esperado es el requerimiento 4 al ser O(n^2) y tener el mayor tiempo de todos.

2. ¿Existe alguna diferencia entre los resultados obtenidos al ejecutar las pruebas en diferentes máquinas?

Hubo algunas diferencias ya que dieron resultados diferentes pero en general ambos fueron muy similares.

3. De existir diferencias, ¿a qué creen que se deben?

Esto probablemente debido al gasto de memoria ya que en un computador la memoria no subió de 6.20 mientras que en la otra se subió por encima de 6.8 G esto afectando los tiempos, tambien pudo afectar el estado de la maquina y las aplicaiones de fondo.

4. ¿Cuál tipo de grafo?

Utilizamos el tipo de árbol “ADJ” list debido a que este nos parecio que organiza de mejor manera la información y a pesar de que en el digrafo pudo aver sido mejor una matriz nos gusta mas la forma de organización y funcionamiento del adj_list. En el no dirigido si era mejor unadj_list y ese usamos

5. Comparación con retos anteriores:

Sin duda alguna, respecto a los retos anteriores este fue el más completo ya que se utilizaron todaas las formas de estrcturas de datos vistas previamente para que se complementaran de la mejor manera. El tiempo de complejidad diria que fue un punto medio al tener reuqerimientos rapidos a la vez que muy complejos. Esto debido a que existian varios algoritmos y muchas estructuras de datos que podian llegar a confundir. Sin embargo fue un reto muy placentero de hacer y muy didactico.

6. Conclusiones:

A lo largo de este curso hemos visto diferentes maneras de organizar los datos y cómo esto afecta el desempeño de nuestro código. Listas , mapas y por último grafos. En ocasiones la información que poseemos no se ajusta a estructuras de datos lineales por lo que una estructura de relaciones distintas nos serviría para de una forma distinta tratar la información. Los grafos son importantes para relacionar los datos de una forma útil añadiendo maneras de utilizar datos para crear relaciones con otros. Terminamos entonces con redes inmensas de elementos relacionados con otros que, aunque su complejidad para ser entendidos aumenta, aumentan también las maneras que conseguir información de manera eficiente tratando de priorizar la eficiencia y por e nde tiempo sobre la facilidad de comprensión

Con lo anterior cabe resaltar que a pesar de tener tipos de estructuras con utilidades más interesantes, se hace necesario para preservar (utilidad/complejidad) del código las estructuras de datos más sencillas como listas y tablas de hash para organizar información adicional y útil para facilitar procesos. Los grafos ayudaron a brindar mucha variabilidad una gran pelotra de usos y de maneras de manejar la información, esto es bueno porque permite ejercicios más completos y complejos pero a la vez puede llegar a ser confuso y dificicl manejarlos. Afortunadamente usar los otros tipos de estructuras de datos volvieron este proceso más ameno y se complementaron todos muy bien.

Como se puede ver en las graficas hechas, los analisis de complejidad estuvieron realizados adecuadamente. Se podrian optimizar algunas cosas como los ciclos al recorrer los mapas pero en general todo estaba tan optimizado como era posible y los tiempos no terminaron muy grandes, un buen trabajo en su gran mayoría.