Grupo 5 Sección 4

- Akshaya Arunachalam - a.arunachalam@uniandes.edu.co – 202020637

- Santiago Forero - s.forerog2@uniandes.edu.co - 202111446

Análisis de complejidad temporal

V = vértices del grafo

E = arcos del grafo

R = rutas discriminadas por aerolínea

Requerimiento 1: Se describe las complejidades de la función maxinterconexion()

Función	**Complejidad**	**Descripción**
m.valueSet()	O (1)	Lista
For(1)	O(V*R)	Se recorre la lista de aeropuertos (V pasos) y en cada paso se consulta las rutas asociadas (R rutas).
For (2)	O (5V)	Se recorre primero los 5 máximos grados (número de rutas) y luego se recorre la lista de aeropuertos (V pasos) en busca de aeropuertos con el mismo grado.
Total =	O (V*R) + (5V) = O(V(R+5)	

Requerimiento 2:

Inicialmente se aplica el algoritmo de Kosaraju. En este inicialmente se calcula el grafo inverso que es proporcional en tiempo a O(V+E), posteriormente se realiza un recorrido DFO con una complejidad proporcional también a O(V+E), finalmente con el resultado del DFO se realiza otro recorrido DFS también con complejidad proporcional a O(V+E). Por lo que se puede ver como la complejidad temporal del algoritmo de Kosaraju es proporcional en tiempo a O(3*(V+E)) o simplemente a O(V+E). Posteriormente se llaman las funciones scc.connectedComponents y scc.stronglyConnected ambas con complejidad O(1). Por lo que la complejidad temporal de este requerimiento es proporcional a O(V+E).

Requerimiento 3:

Función	**Complejidad**	**Descripción**
Consulta ciudad de salida	O (1)	Verificar si existe ciudad digitada, corregirla de ser necesario y consultar en el mapeo de ciudades una vez.
Consultar ciudad de destino	O (1)	Verificar si existe ciudad digitada, corregirla de ser necesario y consultar en el mapeo de ciudades una vez.
cityToairport() * 2 (se ejecuta dos veces)	O (V^2*P)	Se recorre el mapeo de aeropuertos (V pasos por latitud y V pasos por longitud) para determinar si están o no en determinada región. La región cambia P veces, siendo P usualmente pequeño (alcanzando su máximo para ciudades alejadas de aeropuertos). Si P = 100, el are de búsqueda tendrá 1000 km de radio por

		lo que se espera que casi nunca haya valores tan grandes
Dijkstra	O (E log V)	Se determinan las rutas mas cortas desde el aeropuerto de salida.
pathTo ()	O(N)	Se recorren las ciudades en el camino de costo mínimo.
Total =	O ((V^2*P) + (E log V) + (N))	

Requerimiento 4:

N = Aeropuertos que hacen parte de la ruta de expansión mínima

Inicialmente se aplica el algoritmo Prim en su versión Eager con complejidad temporal proporcional a O(E logV). Posteriormente se llama a la función cityToairport en donde inicialmente se usa la función values para obtener los valores en el radio de 10 km. Después, se realiza un recorrido de este resultado para dentro de cada longitud del rango se extrae del mapa ordenado de latitudes donde posteriormente se recorren todas las latitudes en el rango de 10 km y se extraen los aeropuertos asociados para añadirlos a la lista de donde se buscará el más cercano; todo esto ocurre con una complejidad muy pequeña debido a que en el rango de 10 km de una ciudad suelen haber pocos aeropuertos, por lo que esta complejidad no se tiene en cuenta. Posteriormente se hace un recorrido por todos los vértices para ver cuales aeropuertos hacen parte del árbol de expansión mínima, esto con complejidad O(V). Después se llama a la función weightMST con complejidad de O(E). Y finalmente se explora la rama más larga del árbol con raíz en la ciudad ingresada por el usuario en donde se frena cuando el valor del próximo aeropuerto sea igual a None, esto con una complejidad de O(N). Por lo que finalmente la complejidad temporal sería O(E logV + V + E + N).

Requerimiento 5:

Función	Complejidad	Descripción
adyacencia ()	O (A), A < V	Consulta las rutas asociadas a un vértice y para hacerlo recorre una lista con A elementos; A es el número de aeropuertos afectados por el cierre.
listadata(creado,llenado)	O (A), A < V	A es el numero de aeropuertos afectados por el cierre, listadata consulta cada aeropuerto afectado y almacena los datos de dicho aeropuerto.
Total =	O (2A)	

Requerimiento 6:

N = Cantidad de ciudades en el camino de costo mínimo entre las ciudades ingresadas por el usuario

Inicialmente se realizan 3 request al API “Airport Nearest Relevant” cuya respuesta depende de factores externos al código. Posteriormente se aplica el algoritmo de Dijkstra con complejidad temporal proporcional a O(E logV). Después se llaman a las funciones pathTo en donde se recorren odas las ciudades en el camino de costo mínimo con complejidad O(N). Finalmente se llama a la función distTo con complejidad O(1). Por lo que finalmente la complejidad temporal de este requerimiento es O(E logV + N)

Requerimiento 7:

1- Imprime los 5 aeropuertos mas interconectados, su complejidad es constante igual a O (5).

2- Se recorre cada vértice para comprobar si hacen parte de un componente fuertemente conectado, para posteriormente graficarlos con funciones de folium. O(V)

3- Imprime los aeropuertos en la ruta mas corta y luego traza la ruta, si N es el numero de aeropuertos en dicha ruta la complejidad será O(2N). (una vez traza los puntos y el segundo para la ruta).

N = Aeropuertos que hacen parte del camino de costo mínimo

4- Se recorren todos los aeropuertos que hacen parte del camino de costo mínimo para usar las funciones del folium para graficar el camino a seguir. O(N)

5- Sea A el numero de aeropuertos afectados por el cierre, la complejidad del grafico será O(A+1).