

Camilo Ortiz Cruz 201821615 c.ortizc@uniandes.edu.co

Kevin Fernando Gómez Camargo 202015120 k.gomezc@uniandes.edu.co

## Análisis de complejidad temporal en notación Big O

### Requerimiento 1

1. Recorrer los vértices del grafo dirigido  $V_d$
2. Por cada vertice sumar el indegree con el outdegree  $1$
3. Si el valor de la suma es mayor crear una lista y agregar el vertice con su valor de la suma  $1$
4. Si es igual agregar el vertice y el valor de la suma a la lista creada  $1$

1. Recorrer los vértices del grafo no dirigido  $V_{nd}$
2. Por cada vertice sumar el degree  $1$
3. Si el valor de la suma es mayor crear una lista y agregar el vertice con su valor de la suma  $1$
4. Si es igual agregar el vertice y el valor de la suma a la lista creada  $1$

$$\begin{aligned} T_{total} &= V_d + V_{nd} & V_d > V_{nd} \\ &= O(V_d) \end{aligned}$$

Como se puede ver la complejidad final es el número de vértices del grafo dirigido, esto se debe a que el grafo dirigido siempre va contener más o los mismos vértices que el grafo no dirigido y porque en notación O el termino mayor es el que queda, dado que este requerimiento era buscar un máximo consideramos que esta complejidad es muy buena ya que para encontrar el valor mas grande se debe recorrer todos los elementos por tanto consideramos que estamos cerca de los más óptimo.

### Requerimiento 2

Preprocesamiento:

0. Calcular los componentes conectados  $O(E+V)$

Algoritmo:

1. Dar numero de componentes  $1$
2. Encontrar si dos vertices están fuertemente conectados  $1$

$$T_{total} = O(1)$$

Como se puede observar la complejidad dio constante, esto se debe a que el cálculo de los componentes fuertemente conectados (KosarajuSCC) se realizó en la carga de datos porque este no necesita de ningún parámetro del usuario. Como ya está en el catálogo entonces las funciones necesarias para responder el requerimiento solo son connectedComponents y stronglyConnected, que son constantes. Así, consideramos que esta es la solución óptima.

### Requerimiento 3

1. Encontrar aeropuerto mas cercano  
Complejidad función findNearestAirport:

1. Sacar los valores que estén en el rango de latitud en el árbol de las latitudes de los aeropuertos  $\log(\text{lat}) + \#A_{\text{long}}$
2. Iterar sobre los valores del árbol de latitud, los cuales son arboles de longitud y los valores de este  $\#A_{\text{long}}$  son aeropuertos.
3. Por cada arbol de longitud sacar los valores que estén en el rango de longitud  $\log(\text{long}) + \#air$
4. Iterar sobre esos valores y calcular la distancia a la ciudad y escoger el que este a menos distancia  $\#air$
5. De no encontrar ninguno incrementar el radio de búsqueda hasta encontrar o hasta que el radio sea de mas de 1000 km 1000

$Total = \#A_{\text{long}} * \#Air$

2. Calcular arbol de rutas mas cortas (Dijkstra) desde el aeropuerto de origen  $E \log(v)$
3. Encontrar el path hacia el aeropuerto de destino  $\#arcs_{\text{Origen Destino}}$
4. Calcular la distancia hacia el aeropuerto de destino 1

$Total = O(E \log(v))$

En el caso del requerimiento 3, la complejidad temporal la determina el algoritmo Dijkstra esto se debe a que el numero de arcos por el logaritmo de vértices es mayor a la complejidad de encontrar los aeropuertos, ya que en esencia la complejidad de encontrar el aeropuerto cercano es la de recorrer todos los aeropuertos que estén en el rango de búsqueda lo cual siempre será menor que todos los arcos del grafo multiplicado por el logaritmo del número de vértices.

#### Requerimiento 4

Preprocesamiento:

0. Calcular MST  $E \log(v)$

Algoritmo:

1. Encontrar aeropuerto mas cercano  $\#A_{\text{long}} * \#Air$
2. Hacer BFS modificado  $E + V$
3. Encontrar el valor del mayor numero de saltos 1
4. Encontrar el vertice que esta a esa distancia 1
5. Encontrar el path hacia el vertice mas lejano  $\#arcs_{\text{Destino Origen}}$
6. Encontrar cual es el vertice mas lejano donde le alcancen las millas para llegar  $\#Vertices desde Origen$
7. Si se encuentra un vertice posible calcular el path  $\#arcs_{\text{Destino Origen}}$

$Total = O(E + V)$

Como se puede observar la complejidad del algoritmo está relacionada a realizar BFS. Consideramos que este tiempo es razonable porque para encontrar cuál es el vértice que está más lejos de un origen es necesario recorrer todos los vértices y arcos del árbol puesto que de otra forma no podríamos estar seguros, además, el resto de pasos del algoritmo solo tienen como complejidad un subconjunto de elementos de los arcos o los vértices, por lo cual  $E+V$  siempre será mayor.

#### Requerimiento 5

1. Recorrer todos los vertices  $V$
2. Por cada vertice si el vertice es el pasado por parámetro recorrer todos los adyacentes a este  $\#adj$
3. Si no es el pasado por parámetro revisar si los adyacentes contienen el vertice pasado por parámetro.  $\#adj$

$Total = O(V * \#adj)$

El algoritmo tiene una complejidad que consideramos adecuada puesto que para poder encontrar que aeropuertos son afectados se debe revisar todos los arcos entre él y todos los demás aeropuertos. Sin embargo, como necesitamos saber los aeropuertos afectados entonces se debe revisar las listas de adyacencia de cada vértice del grafo y revisar si está el de interés, lo cual es exactamente lo que hace el algoritmo. Algo importante a resaltar es que probablemente no se revisen las listas de adyacencias totalmente sino que sean recorridos parciales, pero se considera el peor caso posible para que la complejidad temporal sea  $O(V \cdot \#adj)$ .

#### **Requerimiento 6 (bono)**

En base a lo realizado en el requerimiento 3 es claro que la complejidad de este es similar ya que funciona de la misma forma solo que en vez de buscar el aeropuerto más cercano en la base de datos se calcula utilizando el API de Amadeus, por ende, la única diferencia es la complejidad de tiempo de la búsqueda de Amadeus, ya que no se conoce la implementación entonces la complejidad total será la misma del requerimiento 3  $O(E \cdot \log(v) + c)$ , pero con una constante que hace referencia a la complejidad desconocida del API. Sin embargo, para simplificar se dirá que es  $O(E \cdot \log(v))$ .

#### **Requerimiento 7 (bono)**

##### *Req. 1*

Para el requerimiento 1 la complejidad de crear el mapa es de  $O(E)$ , donde E es el numero de arcos en cada grafo, esto se debe a que en el mapa se ponen los arcos como líneas y marcadores con los aeropuertos.

##### *Req. 2*

La complejidad de este requerimiento es la de recorrer todas las llaves del mapa de componentes conectados, lo cual en esencia es recorrer todos los vértices por ende la complejidad  $O(V)$ .

##### *Req. 3*

Para este requerimiento solo es necesario recorrer el path retornado por la función del requerimiento 3, por tanto, la complejidad es  $O(P)$  siendo P el numero de elementos en el path.

##### *Req. 4*

La complejidad total de hacer este mapa es igual  $O(MB)$  siendo MB el numero de elementos en la rama más larga, hay que tener en cuenta que también se grafica el mapa de las ciudades a visitar entonces hay una complejidad adicional de C (C es el numero de ciudades que puede visitar la persona con las millas), pero dado que C como máximo es igual a la rama mas larga entonces la complejidad final es  $O(MB)$ .

##### *Req. 5*

La complejidad total de crear este mapa es de  $O(A)$  siendo A el número de aeropuertos afectados por cerrar un aeropuerto.