Observaciones Reto 3

José Vicente Vargas Panesso – 201815601

Daniel Reales – 201822265

1. Análisis de complejidades:

Requerimiento 1:

La primera subrutina invocada por el requerimiento es realizar el algoritmo de Kosaraju sobre el grafo que describe las conexiones. Debido a que este algoritmo realiza dos recorridos sobre el grafo, este tiene una complejidad temporal lineal de $O(V + E)$, donde E representa el número de arcos y V el número de vértices.

Posteriormente, debido a que la función que implementa este algoritmo retorna una estructura que contiene los componentes fuertemente conectados, se procede a consultar cuantos de estos hay en el grafo de interés. Este operación tiene una complejidad de $O(1)$. Posteriormente se realizan operaciones para determinar el *id* de cada uno de los *landing points*. Debido a que estas son operaciones de consulta sobre mapas, tienen complejidad $O(1)$.

Una vez se conoce cuales son los vertices del gráfo que el usuario determinó por parámetro mediante las consultas antermente descritas, se procede a realizar un llamado a la función *sronglyConnected()* que retorna un booleano indicativo de si los vértices estan en el mismo componente fuertemente conectado. Para determinarlo realiza una consulta sobre un mapa que contiene el identificador del componente de cada uno de los vertices. Debido a que las consultas son sobre mapas, la complejidad temporal de la subrutina es de $O(1)$.

En conclusión, si se realiza la suma de cada una de las complejidades de las subrutinas invocadas al realizar el llamado de la función que da solución al requerimiento 1, vemos que de puede representar mediante:

$$O(V + E) + O(1)$$

Para finalmente obtener que la complejidad temporal del requerimiento es de

$$O(V + E)$$

Requerimiento 2:

El requerimiento 2 inicia por recuperar los nombres de los paises cuyo valor se almacena en un mapa. Debido a que esta es la estructura de datos utilizada entonces estas consultas tienen una complejidad temporal de $O(1)$.

Posteriormente, se procede a aplicar el algoritmo de *Dijkstra* utilizando como vertice base la capital del país 1 ingresado por parámetro. Teóricamente sabemos que este algoritmo posee una complejidad temporal de $E \log(V)$.

Una vez finaliza el algoritmo, se utiliza la lista resultante para calcular tanto: 1) la ruta de costo mínimo necesaria y 2) el costo total de dicha ruta. Para determinar esta información acorde a lo indicado se invoca a la función *pathTo()* cuya implementación se realiza mediante un recorrido lineal adjuntando los caminos que se deben tomar a una pila hasta que se llegue al vértice de origen. Por lo tanto, si la ruta de costo mínimo entre el país A y el país B tiene m elementos esta función tendrá complejidad temporal $O(m)$

Finalmente, se realiza un llamado a la función *distTo()*. Esta función recibe por parámetro los dos caminos y retorna el costo mínimo asociado a la ruta encontrada por el algoritmo. Para ello utiliza un mapa y su operación asociada *get* y una indexación. Por tanto, esta subrutina tiene complejidad temporal $O(1)$.

En conclusión, si se realiza una suma de todas las subrutinas invocadas se tiene que la complejidad temporal del requerimiento es de:

$$O(E \log(V)) + O(m) + O(1)$$

Esto puede ser aproximado por:

$$O(E \log(V))$$

Requerimiento 3:

El requerimiento 3 inicia por realizar el algoritmo de PRIM para encontrar el MST. Teóricamente sabemos que este algoritmo tiene una complejidad temporal de $O((V + E) \log(V))$. Posteriormente, utilizando el mapa que contiene a todos los vértices marcados, se consulta el tamaño. Esta operación tiene complejidad $O(1)$.

Posteriormente, se invoca la función *weightMST()*. Esta función retorna el costo total asociado al MST encontrado. Para determinar este valor itera sobre cada uno de los vertices siguiento la ruta adecuada. Por tanto, si existen M arcos en el MST entonces esta función tendrá una complejidad temporal de $O(m)$.

Finalmente, se procede a realizar un llamado a la función *encontrarRama()*. En ella se realizan dos recorridos sobre todos los vertices del MST. En primera instancia, se realiza un recorrido para copiar el MST a un nuevo grafo inicializado al momento del llamado de la función. Esto tiene una complejidad temporal de $O(m)$, donde m es el número de vertices en el MST. Posterior a este recorrido, se aplica el algoritmo de *Dijkstra* para determinar la ruta mínima desde el nodo raíz del MST hasta cada uno de los nodos. Para encontrar la distancia de la ruta

más larga se itera sobre cada uno de los nodos distintos al nodo raíz recuperando la longitud de la ruta mínima. Esto tiene tambien una complejidad temporal $O(m)$.

En conclusión, teniendo en cuenta el llamado al algoritmo PRIM, la aplicación del algoritmo *Dijkstra* y los recorridos sobre los vertices, tenemos que la complejidad temporal corresponde a:

$$O((V + E)\log(V)) + 2O(m) + O(E\log(V))$$

Esto puede ser resumido a

$$O((V + E)\log(V))$$

Nota: Debe tenerse en cuenta que el algoritmo de Dikstra se aplica ahora sobre el MST. Por esta razón el número de arcos al que se refiere en esta complejidad no es el número de arcos del grafo original sino del MST.

---

2. Tiempo de ejecución.

	Máquina 1	Máquina 2
Procesadores	Intel core i7-7700HQ	Intel i5-8250U
Memoria RAM (GB)	16 GB	8 GB
Sistema Operativo	Windows 10	Arch Linux

Especificaciones de las máquinas para ejecutar las pruebas de rendimiento.

Maquina 1

Resultados

Requerimiento 1	**Resultados**
Tiempo (ms)	5522.044
Espacio (kB)	134.8183

Tabla 1. Comparación de Desempeño en términos de tiempo y espacio para cada una de las implementaciones del Requerimiento 1

Requerimiento 2	**Resultados**
Tiempo (ms)	1212.827
Espacio (kB)	32.9326

Tabla 2. Comparación de Desempeño en términos de tiempo y espacio para cada una de las implementaciones del Requerimiento 2

Requerimiento 3	**Resultados**
Tiempo (ms)	5740.770
Espacio (kB)	99.343

Tabla 3. Comparación de Desempeño en términos de tiempo y espacio para cada una de las implementaciones del Requerimiento 3

Maquina 2

Resultados

Requerimiento 1	**Resultados**
Tiempo (ms)	5618.749
Espacio (kB)	117.4375

Tabla 4. Comparación de Desempeño en términos de tiempo y espacio para cada una de las implementaciones del Requerimiento 1

Requerimiento 2	**Resultados**
Tiempo (ms)	879.1225
Espacio (kB)	22.91406

Tabla 5. Comparación de Desempeño en términos de tiempo y espacio para cada una de las implementaciones del Requerimiento 2

Requerimiento 3	**Resultados**
Tiempo (ms)	3633.1804
Espacio (kB)	99.38671

Tabla 6. Comparación de Desempeño en términos de tiempo y espacio para cada una de las implementaciones del Requerimiento 3

Análisis de Complejidades de Tiempo y Espacio

Como es posible observar en los resultados antes presentados, el requerimiento que tuvo un mayor nivel de uso de memoria fue el requerimiento 1. Esto se debe al uso de estructuras *helper* tales como el grafo *G reverso* necesario para realizar el algoritmo de Kosaraju.

En cuanto al requerimiento con una mayor demanda de tiempo de ejecución, encontramos que es el requerimiento 3. Esto se ajusta al análisis teórico realizado en la primera sección. En este se resalta que la complejidad temporal de este corresponde a $O((V+E)\log(V))$. Esto es más costoso en tiempo en comparación con complejidades de $O(V+E)$ o $O(E\log(V))$.