**Análisis de Complejidad Reto 4**

A continuación, se presentará el análisis de complejidad temporal de cada uno de los requerimientos del reto 4:

**Requerimiento 1:**

Este requerimiento recibe como parámetros los nombres de 2 Landing Points y pide devolver la cantidad de componentes conectados del grafo que representa las conexiones de cables submarinos y un valor booleano que informe si los Landing Points están en el mismo componente conectado o “clúster”. La estrategia general que se usó fue construir un grafo dirigido y aplicar dos algoritmos sobre este grafo. El primer algoritmo por usar fue Kosaraju-Sharir sobre un grafo dirigido (cuya estructura es básicamente la de un grafo dirigido, solo que se usa un arco de ida y otro de vuelta de forma independiente). De esta forma, la cantidad de SCC encontrados por Kosaraju-Sharir se correspondería con la cantidad de CC sobre el grafo no dirigido (recordemos que el concepto CC es equivalente a SCC si se trata de un grafo no dirigido). Luego, se realizó DFS sobre uno de los vértices que entra por parámetro y en el mapa de vértices marcados se pregunta si el segundo vértice está presente. De esta forma sabemos si ambos vértices son alcanzables y por lo tanto podemos responder si están en el mismo CC. La información necesaria para ejecutar los dos algoritmos (o para sacar información de estructuras que estos devuelven) estaba guardada en mapas, por lo que la mayoría de las líneas tienen una complejidad . Por otro lado, sabemos que la complejidad de DFS es donde V es el conjunto de vértices y E el conjunto de arcos. Finalmente, la complejidad de Kosaraju – Sharir es conservando la notación anterior. Concluimos que:

**Requerimiento 2:**

Principalmente para la realización de este requerimiento, se utilizaron las estructuras ‘countries’ y ‘LandingPoints’, las cuales corresponden a dos maps que guardan cierta información acerca de cada landing point o los cables a los cuales están conectados. En primer lugar, se realiza un recorrido en la estructura ‘countries’ donde se asume que su complejidad es O (), teniendo en cuenta que este pertenece al número de países que se encuentran. Ahora bien, dentro de este recorrido se realiza otro, donde el numero de elementos que se recorren son los landing points que tiene cada país, por lo que su complejidad es O (). Luego de estos recorridos, se hace un ultimo donde se cambia de estructura y se utiliza la de ‘LandingPoints’ con el fin de obtener la información especifica para cada landing point que se agregó a lista\_LP, donde se encuentran los que sirven como punto de interconexión. Este recorrido posee una complejidad temporal O (), teniendo en cuenta que corresponde a los elementos de la lista\_LP. Finalmente, también se debe considerar el acceso a los mapas para sacar información como las parejas llave-valor, o el añadir un elemento en la lista, donde su complejidad general concierne a O(k). En ese sentido, la complejidad temporal del requerimiento es:

**Requerimiento 3:**

Para el desarrollo de este requerimiento se usaron dos funciones. El objetivo del requerimiento 3 es recibir como parámetros el nombre de 2 paises y encontrar la ruta de costo mínimo entre ambos. La primera función llamada “consulta\_ruta\_mínima” obtiene información del catálogo de datos a partir de la información que entra por parámetro. En el catálogo solo saca informaciones de mapas, por lo que la mayoría de las líneas presentan una complejidad . Luego con esta información, la función debe de implementar uno de los dos algoritmos para encontrar una ruta mínima. Como el grafo no contiene valores negativos (una distancia negativa carece de interpretación física), se ignora Bellman-Ford y se implementa el algoritmo de Dijkstra sobre el grafo dirigido, el cual tiene una complejidad de además de un espacio extra con complejidad espacial . Posteriormente, se usa la función “ruta encontrada” la cual es la encargada de construir la ruta desde el punto de partida al punto de llegada con la información guardada en la estructura de Dijkstra. Esta función en el peor de los casos recorre todo el mapa de vértices visitados guardado en la estructura de Dijkstra y si este mapa guarda elementos, tenemos que la complejidad de esta función es . Con este análisis, concluimos que:

**Requerimiento 4:**

El análisis de complejidad para este requerimiento se basa en una función ´consulta\_red\_expansion\_minima’ donde en primer lugar, utilizando el grafo no dirigido ‘connections’ se aplica el algoritmo prim Eager con el fin de obtener un MST para así cumplir con el objetivo principal del requerimiento. Su complejidad temporal corresponde a , donde pertenece al número de los arcos, y al número de vértices del grafo. En segundo lugar, se realiza un recorrido por los nodos del MST que se obtuvo gracias al algoritmo, con el fin de sacar el costo total del MST, la conexión más larga y corta; su complejidad se basa en O (), donde es el total de nodos. Particularmente, para conocer la conexión mas larga y corta, también se utiliza el algoritmo de ordenamiento mergesort el cual su complejidad corresponde a , donde especifica el número de nodos, ya que se agregó la misma cantidad a la lista que se quería ordenar (‘lista\_info’).

Luego, se realiza otro recorrido con complejidad temporal O (), donde concierne al número de llaves que se encuentran en el mapa ‘mapa vértices’ y aquí se utiliza una función auxiliar ‘consulta\_rama\_mas\_larga’ donde su característica principal es la recursividad. La complejidad temporal de esta función siempre será la del peor caso, puesto que tiene que hallar la rama más larga, por lo tanto, es O , donde es la altura del MST. No obstante, es importante considerar que se repite veces debido al llamado de la función en el recorrido de las llaves del ‘mapa\_vertices’. Finalmente, también se debe tener en cuenta el acceso a los mapas, el añadir un elemento a las diferentes listas y demás, donde su complejidad general concierne a O(k). En ese sentido, la complejidad temporal del requerimiento es:

**Requerimiento 5:**

Este requerimiento recibe como parámetro el nombre de un Landing Point y debe devolver una lista de paises afectados por una falla en el Landing Point que recibió como parámetro (en orden descendente de distancia). Vale la pena aclarar que se considera como país afectado a aquel país que cuente con Landing Points directamente conectados al Landing Point con fallas. Para esto NO se usó ningún algoritmo de grafos, sino que por el contrario se aprovecho la estructura de datos del catálogo en mapas para hacer una búsqueda eficiente en tiempo que solo tuviera en cuenta los vértices estrictamente necesarios. La función de consulta del requerimiento se llama “consulta\_paises\_afectados” y la mayoría de sus líneas de código son accesos a información guardad en mapas, por lo que presenta complejidad . Posteriormente, se itera la lista de cables conectados al Landing Point que presenta la falla con el objetivo de conocer que vértices del grafo de la forma LandingPoint-CableID representan todo el Landing Point. Luego, a cada uno de los vértices se les aplica la función “gr.adjacents” la cual recorre la lista de adyacencias que representa al grafo, por lo que presenta una complejidad de O(|V|). Esta se recorre con el objetivo de obtener los vértices adyacentes a cada vértice que hace parte del Landing Point que entra por parámetro y obtener el nombre del país afectado. Como está operación se realiza para todos los vértices que componen cierto Landing Point, se tiene que, si n vértices componen el Landing Point con fallas, la complejidad de este proceso es . Hasta este momento, la función ha logrado encontrar un conjunto de países afectados y los guarda en un mapa donde cada llave es la distancia del país al Landing Point con fallas. Luego se aplica mergesort sobre la lista de llaves con una complejidad de para finalmente recorrer el mapa en el orden de la lista ordenada y devolver esta lista de países, esto último con complejidad . Con este análisis concluimos que:

**Resultados Pruebas de Tiempo y Espacio:**