Departamento de Ingeniería de Sistemas y Computación
Estructuras de Datos Y algoritmos
ISIS1225 - 202119

Documento de Análisis Reto 4

Ana Sofía Castellanos Mosquera
202114167
a.castellanosm@uniandes.edu.co

Martín Santiago Galván Castro
201911013
ms.galvan@uniandes.edu.co

Análisis General – Guardar datos:

Para el cumplimiento de los requerimientos, primero fue necesario planear la manera en la que los datos se leerían, en que orden y en que estructura de datos usar para guardarlos. En este reto, se leyeron 3 archivos .csv. Estos archivos fueron: "connections.csv", el cual contiene información entre las conexiones entre los puntos de conexiones de la red; "countries.csv", el cual contiene información acerca de los países de la red, en donde se incluye información de la capital y cantidad de usuarios del país; y, por último, un archivo llamado "landing_points.csv", que contiene información acerca de todos los puntos de conexión en la red.

Para guardar la información de estos tres archivos, se generó un catálogo que contiene 2 tablas de hash, un grafo dirigido y dos listas. El propósito de las tablas de hash, guardadas como "hashCountryCap" y "hashidInfo", es de guardar la información de los archivos countries.csv y landing_points.csv. La primera tabla de hash tiene como llave el nombre de un país. Y dentro de su valor, se guarda la información de la capital del país como la posición y usuarios. Mientras que para "hashidInfo", se guardan como llaves los id de los puntos de conexión, y como valor se tiene la información de ese punto de conexión.

El propósito del grafo es de guardar todos los puntos de conexión como vértices y poner como conexiones los cables entre estos. Se utilizo como costo la distancia geográfica entre los dos puntos de conexión, la cual se calcula a partir del uso de la ecuación de haversine conociendo las coordenadas de los puntos de conexiones.

Por último, en las dos listas se guardan registros ordenados de lo que se lee en el archivo de países y de puntos de conexión.

Primero, se leyó el archivo de países. Por cada entrada en el Excel, primero se incluyó como vértice en el mapa la capital del país. Posteriormente, se guarda en la tabla de hash como se describió anteriormente.

Después, se leyó el archivo que contiene los puntos de conexión. En este caso, se agrega cada punto de conexión al grafo, y se guarda también en la tabla de hash como se describió anteriormente. Posteriormente, se toma el país de origen del landing point revisando su

nombre. Usando el nombre del país, se ingresa a la tabla de hash por países en búsqueda del nombre del vértice de la capital del país y sus coordenadas. Haciendo uso de esta información se calcula la distancia entre el landing point y la capital. Este será el costo de la conexión entre la capital y el punto de conexión. Se generan conexiones de la capital al punto de conexión y del punto de conexión a la capital.

Por último, se carga el archivo de connections. En este, se identifican los vértices de origen y destino a partir de sus id. Con sus id se busca en la tabla de hash de puntos de conexión la información de sus coordenadas. Usando haversine, se calcula la distancia entre ambos puntos y se usa esto como el peso.

Análisis de complejidades:

Requerimiento 1:

El primer requerimiento consistía en encontrar la cantidad de clústeres dentro de ola red de cables submarinos y si dos puntos de conexión pertenecen o no al mismo cluster. Se tiene como entrada del requerimiento los nombres de dos landing points.

Para realizar este requerimiento, fue necesario hacer uso del algoritmo de kosaraju. El cual, su procedimiento es primero revertir las conexiones del grafo, para luego realizar el algoritmo DFO en el grafo reversado. Al hacer esto, obtiene el orden topológico de dicho grafo. Procede a realizar recorridos BFO en el grafo original, pero en el orden topológico obtenido anteriormente. Cada vez que empieza un recorrido empezando desde un nodo, marca los nodos con un índice para indicar a que componente pertenecen

Posteriormente, solo se revisan los índices de dos nodos en el grafo para revisar si pertenecen al mismo componente.

La complejidad del algoritmo escrito depende principalmente del algoritmo para encontrar los índices de cada nodo. Usando una lista de adyacencia, los recorridos hechos deberían de completarse de manera lineal recorriendo al menos dos veces cada nodo y cada arco. Por último, la revisión de los índices al final para verificar si pertenecen al mismo componente tiene una complejidad constante, dado a que se busca el vértice en la estructura de búsqueda devuelta por el API del algoritmo de kosaraju.

Un análisis de la complejidad:

$$2(O(V+E)) + 2(O(1))$$

$$O(n)$$

La complejidad de recorrer todos los vértices y los nodos se vuelve O(n) por simplificar, dado a que igualmente, la complejidad del algoritmo crece de manera lineal.

Requerimiento 2:

El requerimiento N2 se resolvió considerando que en el catálogo se encuentra un grafo que contiene las conexiones entre landing points y además existe para cada país una conexión

entre la capital y los landing points que pertenecen a dicho país. A partir de allí se sigue la siguiente estrategia:

1. Obtener a partir de los países que ingresan por parámetro las capitales de estos, ya que son las capitales las que se encuentran en el grafo y permitirán encontrar el camino mínimo
2. Obtener los caminos mínimos para la capital del primer país a través del algoritmo de Dijkstra
3. Verificar y obtener a través de la función pathTo el camino existente entre la capital del primer país y del otro país. Esta funcion devuelve en caso de que exista un camino una lista que contiene el camino vértice a vértice para llegar al destino, en caso de que no exista la función devolverá None.
4. Reconstruir el camino entre las capitales de los países y calcular el costo mínimo total que conecta a las capitales.

Para poder obtener las capitales a partir de los nombres de los países el algoritmo del requerimiento busca a partir del país ingresado la capital mediante la función get(…) para la tabla de Hash presente en el catálogo que se llama 'hashCountryCap'. Esta operación cuenta con una complejidad de O(1).

A partir de estas capitales se buscan los caminos mínimos para una de estas capitales a través del algoritmo de Dijkstra. Este algoritmo crea un minPQ y una estructura para ir guardando el camino más corto de los vértices al que se ingresa por parámetro. Para ello va por cada vértice y revisa para sus adyacentes si se puede relajar el peso del arco o si lo deja como estaba previo a la iteración, luego continua con otro vértice que se elige a partir de la prioridad del minPQ hasta revisar en el peor caso todos los vértices.

En cada revisión de los adyacentes de cada vértice tras relajar el peso de los arcos se modifica su valor en el minPQ en caso de que ya exista o se inserta en el minPQ, esta operación tiene una complejidad de O(logV), puesto que al ser una cola de prioridad se verifica primero hacía que mitad se debe dirigir el peso del arco antes de insertarlo o actualizarlo y dependerá de la cantidad de vértices, ya que en la cola están todos los pesos de los arcos para llegar al vértice que se indique en el Dijkstra, desde cualquier otro vértice.

Esta operación de actualizar o insertar al minPQ que cuenta con complejidad O(logV) se debe repetir en cada iteración la cantidad de adyacentes que tenga un vértice, lo cual en el peor caso es de V-1 adyacentes. Es decir que se tendría una complejidad de:

$$O(V-1) * O(logV)$$

Ahora bien esto es para una iteración, sin embargo en total se tendrán que realizar V iteraciones por lo que la complejidad total del algoritmo es de:

$$O(V) * O(V-1) * O(\log V)$$

Cabe mencionar que V (V-1) es también la cantidad de arcos total que pueden existir en el grafo (E) en el peor caso, en el cual todos los vértices están conectados con todos, por lo cual la complejidad se puede reescribir como:

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

$$O(E \log V)$$

El algoritmo del requerimiento tras encontrar los caminos más cortos para una capital, busca el camino más corto con la otra capital mediante la función pathto(…), la cual revisa en primer lugar si existe un camino entre las capitales si no existe retorna None, de lo contrario se añade el camino vértice a vértice encontrado por el Dijkstra a una pila la cual es el retorno, por lo cual en el peor caso se deben añadir V-1 elementos a la pila.

Considerando lo anterior, la complejidad final para el algoritmo del Requerimiento 2 es de:

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

Debido a que E depende en el peor caso de V y que se toma la peor complejidad para Big O se tiene que la complejidad del algoritmo es de:

$$O(E \log V)$$

Requerimiento 3:

Para solucionar el requerimiento 3 debido a que se debía buscar la menor infraestructura se optó por implementar el algoritmo de árbol de recubrimiento mínima el cual se implementa en la librería mediante el algoritmo de Prim. Tras ello se debía buscar la rama más larga del árbol para lo que se realizó un nuevo algoritmo. Este requerimiento sigue la siguiente estrategia:

1. Hallar el árbol de recubrimiento mínimo del grafo
2. Encontrar el peso del árbol MST y la cantidad de vértices presentes en este.
3. Buscar la rama más larga del árbol MST

En primer lugar para hallar el árbol MST se hizo uso del algoritmo Prim el cual cuenta con una complejidad O(E logV). Esta complejidad resulta de observar que el algoritmo va añadiendo a una cola de prioridad MinPQ los adyacentes de un vértice que se selecciona a partir del menor de los adyacentes en la cola de prioridad, este procedimiento cuenta con una complejidad de O(logV), para un elemento pero debido a que es sobre toda la lista de adyacentes de un vértice en el peor caso se hará este O(logV) un total de V-1 veces.

Ahora este proceso de añadir a la cola de prioridad y verificar si es menor que los que se encuentran en EdgeTo puede ser en el peor caso de V con lo cual se obtiene una complejidad de:

$$\boldsymbol{O(logV) * O(V) * O(V-1)}$$

V(V-1) es la cantidad de vértices máximo que se tendrá en el peor caso por lo cual la complejidad es de

$$O(E \log V)$$

Tras este algoritmo de Prim se busca el camino de la raíz a la hoja que cuenta con más arcos para esto se obtienen todos los vértices de la tabla de Hash EdgeTo y se itera sobre ella lo cual tiene una complejidad O(N), dentro de la iteración se obtienen los valores que conectan al vértice con su antecesor, por lo cual se realiza un ciclo para llegar hasta el vértice sin antecesor, la raíz.

Aunque pareciera que este doble ciclo tuviera complejidad $O(V^2)$ en realidad el segundo ciclo no se ejecuta si el vértice no tiene un antecesor lo cual hace que en el peor de los casos, cuando todos los vértices se desprenden de una misma rama el algoritmo solo revise el segundo ciclo-de los antecesores-, lo cual hace en O(V) veces pasando por los demás en O(1).

Con esto en mente la complejidad del algoritmo es de:

$$O(Elog\ V)\ +\ O(N) * O(1)$$

$$O(ElogV)$$

Análisis de rendimiento:

Para el análisis de rendimiento, se comprobó el uso de memoria y el tiempo de ejecución de los requerimientos en las máquinas de los dos estudiantes. Las características técnicas de dichas maquinas se presentan en la siguiente tabla:

	Máquina 1	**Máquina 2**
Procesadores	Intel(R) Core (TM) i7-10750H CPU @ 2.60GHz 2.59 GHz	AMD Ryzen 3 3200G with Radeon Vega Graphics 3.60 GHz
Memoria RAM (GB)	16.0 GB	16.0 GB
Sistema Operativo	Windows 10 Home Single Language	Windows 10 Pro

Para realizar los requerimientos se utilizó para cada requerimiento las siguientes entradas:

Requerimiento 1:

- Redondo Beach
- Vung Tau

Requerimiento 2:

- Australia
- Russia

Requerimiento 3: No requiere de parámetros a ingresar

La máquina 1 es de la estudiante Ana Sofía Castellanos Mosquera y la maquina 2 es del estudiante Martín Santiago

Resultados Maquina 1:

Tabla resultados		
Req1	1030.645	92.08367
Req2	310.4427	25.199
Req3	1062.14	54.985

A partir de los resultados es posible inferir que a comparación de las respuestas para anteriores retos, el uso de grafos reduce sustancialmente el tiempo de respuesta de los requerimientos y el uso de la memoria. De esta manera el requerimiento 2 es el que responde de forma más rápida y el que consume menor memoria, lo cual se debe al algoritmo que se implementa para solucionar el requerimiento.

Resultados Maquina 2:

Los resultados de la maquina 2 se expresan mediante la siguiente tabla:

	Tiempo [ms]	**Memoria [kB]**
Req 1	1243,80767	141,2456667
Req 2	723,128667	42,85966667
Req 3	1566,30833	61,586

La forma en la que se obtuvieron estos resultados fue en que, se repitió el requerimiento 3 veces seguidas usando los mismos parámetros de búsqueda. Estos parámetros de búsqueda fueron los siguientes para los requerimientos:

Mas información acerca de los resultados individuales y graficas se encuentran en el archivo de Excel en Data-Maquina 2