ANÁLISIS DEL RETO

Ximena López, ax.lopez@uniandes.edu.co, 202312848

Juan David Torres, jd.torresa1@uniandes.edu.co, 202317608

Sofia Losada, s.losadam@uniandes.edu.co, 202221008

Requerimiento <<1>>

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	• Punto de origen • Punto de destino
Salidas	• La distancia total que tomará el camino entre el punto de origen y el de destino. • El total de vértices que contiene el camino encontrado. • La secuencia de vértices (sus identificadores) que componen el camino encontrado
Implementado (Sí/No)	Si - Sofia Losada

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Transformar en tupla la coordenada inicial y de destino que entra por parámetro	O(1)
Paso 2 Por medio de un condicional se pregunta si en la tabla de hash cuyas llaves son las coordenadas y valores son id están presentes las coordenadas. En caso de que no encuentre se implementa la función auxiliar calcular_distancia_minima para aproximar la coordenada a la más cercana dentro de la malla vial.	O(1)
Paso 3 En caso de que no encuentre se implementa la función auxiliar calcular_distancia_minima para aproximar la coordenada a la más cercana dentro de la malla vial.	O(1)

De esta manera, se obtiene una lista de las llaves (coordenadas) y se itera sobre esta, para obtener la respectiva distancia implementando la función haversiana. Las distancias se añaden a arbol_distancias y su valor asociado es la coordenada Se obtiene la llave menor del arbol, para determinar la menor distancia y su valor asociado la coordenada La coordenada se busca en la tabla de hash, para obtener su id asociado	O(N) O(1) O(1) O(1)
Paso 4 En control['camino'] se almacena el camino dfs desde la coodernada inicial por todo el grado malla_vial	O(V + E)
Paso 5 En path se almacena el camino completo desde la coordenada inicial hasta el destino	O(V + Y)
TOTAL	***O(V + Y)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	**Tiempo (ms)**	**Memoria (kB)**
4 (longitud y latitud inicial, destino)	13.981	23.659

Tablas de datos

Procesadores	**1,8 GHz Intel Core i5 de dos núcleos**
Memoria RAM	8 GB 1600 MHz DDR3
Sistema Operativo	MacOS Big Sur 11.7.10

Análisis

La complejidad de este algoritmo esta dada por la implementación de una tabla de has en donde se tenía como llave las coordenadas y como valor asociado el id respectivo. Se implementó una función auxiliar para calcular la distancia entre nodos y además la distancia mínima al nodo más cercano, dado el caso que la coordenada no se encontrar en un vértice dentro de la malla vial. Se implemento un dfs, pues de esta forma se pudo establecer el recorrido con menores saltos y más corto en el grafo desde el inicio hasta el destino.

Requerimiento <<2>>

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	• Punto de origen • Punto de destino
Salidas	• La distancia total que tomará el camino entre el punto de origen y el de destino. • El total de vértices que contiene el camino encontrado. • La secuencia de vértices (sus identificadores) que componen el camino encontrado
Implementado (Sí/No)	Si - Sofia Losada

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Transformar en tupla la coordenada inicial y de destino que entra por parámetro	O(1)
Paso 2 Por medio de un condicional se pregunta si en la tabla de hash cuyas llaves son las coordenadas y valores son id están presentes las coordenadas. En caso de que no encuentre se implementa la función auxiliar calcular_distancia_minima para aproximar la coordenada a la más cercana dentro de la malla vial.	O(1)
Paso 3	O(1)

En caso de que no encuentre se implementa la función auxiliar calcular_distancia_minima para aproximar la coordenada a la más cercana dentro de la malla vial. De esta manera, se obtiene una lista de las llaves (coordenadas) y se itera sobre esta, para obtener la respectiva distancia implementando la función haversiana. Las distancias se añaden a arbol_distancias y su valor asociado es la coordenada Se obtiene la llave menor del arbol, para determinar la menor distancia y su valor asociado la coordenada La coordenada se busca en la tabla de hash, para obtener su id asociado	O(N) O(1) O(1) O(1)
Paso 4 En control['camino'] se almacena el camino bfs desde la coodernada inicial por todo el grado malla_vial	O(V + E)
Paso 5 En path se almacena el camino completo desde la coordenada inicial hasta el destino	O(V + Y)
TOTAL	***O(V + Y)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	**Tiempo (ms)**	**Memoria (kB)**
4 (longitud y latitud inicial, destino)	16.983	20.653

Tablas de datos

Procesadores	**1,8 GHz Intel Core i5 de dos núcleos**
Memoria RAM	8 GB 1600 MHz DDR3
Sistema Operativo	MacOS Big Sur 11.7.10

Análisis

La complejidad de este algoritmo esta dada por la implementación de una tabla de has en donde se tenía como llave las coordenadas y como valor asociado el id respectivo. Se implementó una función auxiliar para calcular la distancia entre nodos y además la distancia mínima al nodo más cercano, dado el caso que la coordenada no se encontrar en un vértice dentro de la malla vial. Se implemento un bfs, pues de esta forma se pudo establecer el recorrido con menores saltos y más corto en el grafo desde el inicio hasta el destino.

Requerimiento <<3>>

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	• La cantidad de cámaras de video que se desean instalar (M) • La localidad donde se desean instalar
Salidas	El tiempo que se demora algoritmo en encontrar la solución (en milisegundos). La siguiente información de la red de comunicaciones propuesta: * El total de vértices de la red. * Los vértices incluidos (identificadores). * Los arcos incluidos (Id vértice inicial e Id vértice final). * La cantidad de kilómetros de fibra óptica extendida. * El costo (monetario) total.
Implementado (Sí/No)	Si, por Ximena Lopez

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Sacar la lista de los vértices de la localidad	O(1)
Paso 2	O(E)

Iterar por cada una para poder obtener los arcos con los que conecta	
Paso 3 Calcular la distancia haversiana de cada arco al momento de iterar	O(1)
Paso 4 Se obtiene la lista de los vértices de la localidad cono su respectivo número de comparendos. Se realiza un sort de esa lista Se realiza una sublista de las primeras n en la lista de vertices.	O(1) O(logn) O(n)
Paso 5 Se hace un for iterativo dentro de esa sublista para agregar como vértice a el subgrafo Se añade a una lista de vértices solo el id de los vértices que se van agregando	O(V) O(1)
Paso 6 Se realiza un for por esa lista de vértices para calcular el camino más corto de acuerdo al subgrafo de la malla vial y ese se va agregando como peso a el subgrafo con los n vertices necesarios para las cámaras. Por dentro lleva un for por los faltantes sacando un dist to. O se realiza un for por medio de la lista de vértices para calcular la distancia haversiana que hay entre cada uno de los vértices añadiendose como el nuevo peso del subgrafo.	O(n*ElogV) O(V)
Paso 7 Se ejecuta el algoritmo de prim para calcular el MST sobre el grafo.	O(V*V)
Obtener peso del MST.	O(1)
Obtener arcos del MST.	O(E)
TOTAL	***O(V*V) o O(V*V+n*ElogV)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	**Tiempo (ms)**	**Memoria (kB)**

20 cámaras, localidad de CHAPINERO	500.9	

Tablas de datos

Procesadores	**1,8 GHz Intel Core i5 de dos núcleos**
Memoria RAM	8 GB 1600 MHz DDR3
Sistema Operativo	INTEL PENTIUM GOLD

Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el análisis de complejidad.

Este requerimiento tiene dos opciones y una es con base a la malla vial, en este por el recorrido que se hace creando un subgrafo de la localidad e implementar djk para el camino más corto experimentalmente y teóricamente va a tener una mayor complejidad en el peor caso, que es de O(V*V+ n*ElogV). Por otra parte, es el algoritmo implementado para la distancia entre sí de los n vértices, el cual tiene una complejidad de O (V*V), en donde su peor caso es el recorrido del algoritmo Prim.

Requerimiento <<4>>

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructuras de datos, número de comparendos más graves
Salidas	Número de nodos que hacen parte de la red, IDs de los nodos, arcos que hacen parte, distancia total de la red, costo de la red
Implementado (Sí/No)	Implementado por Juan David Torres.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

Obtener los n comparendos más graves.	O(nlogn) N viene del ciclo, logn es el tiempo promedio para extraer cada uno de los valores de un árbol RBT.
Añadir cada uno de los vértices por comparendo al nuevo grafo	O(n*1) N viene de iterar, O(1) es el tiempo que toma añadir un nodo en una lista de adyacencias.
Iterar a través de los vértices.	O(v)
Iterar a través de los vértices.	O(v)
Verificar que un vértice no se encuentre en la lista de adyacencia de otro vértice.	O(n)
Obtener uno los vértices de la tabla de hash con todos los vértices	O(1)
Añadir un arco al grafo.	O(1), lista de adyacencias.
Usar algoritmo de prim para calcular el MST sobre el grafo.	O(V*V)
Obtener peso del MST.	O(1)
Obtener arcos del MST.	O(E)
Iterar a través de los arcos del MST para obtener los vértices, número de nodos, y otros elementos que se piden dentro del requerimiento.	O(E)
TOTAL	***O(V*V)***

Pruebas Realizadas

Las pruebas realizadas se llevaron a cabo en un computador con las siguientes especificaciones:

	Máquina
Procesadores	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
Memoria RAM (GB)	12,0 GB
Sistema Operativo	Windows 11 Home Single Language

Para las pruebas realizadas, se ingresó las 5 infracciones con mayor prioridad. A continuación, se encuentra el promedio obtenido en tiempo de ejecución y memoria usada a lo largo de 3 intentos.

Entrada	**Tiempo (ms)**	**Memoria (kB)**
5	29.769	14.773

Análisis

La complejidad temporal de este requerimiento recae, principalmente, en el algoritmo de Prim que se encarga de calcular el MST para el grafo dado. Su complejidad en el peor de los casos es O(V*V), por lo que es posible afirmar que la mayor parte del tiempo de ejecución del algoritmo se encuentra en este. De la misma forma, la creación del subgrafo tiene una complejidad temporal de O(V*V), por lo que es posible que este también sea responsable de parte de la complejidad temporal del requerimiento. Los otros pasos tienen complejidades temporales menores. Por un lado, el obtener las llaves con mayor prioridad dentro de un árbol rbt tiene una complejidad de O(logn), que al estar dentro de un ciclo, tienen una complejidad temporal compuesta de O(nlogn). Sin embargo, esta es menor a la de O(V*V). Otras partes del código tienen complejidades lineales o constantes, por lo que su contribución a la complejidad temporal es mínima comparativamente.

Requerimiento <<5>>

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	• Cantidad de cámaras de video que se desean instalar • Clase de vehículo.
Salidas	El tiempo que se demora algoritmo en encontrar la solución o El total de vértices de la red. o Los vértices incluidos (identificadores). o Los arcos incluidos (Id vértice inicial e Id vértice final). o La cantidad de kilómetros de fibra óptica extendida. o El costo (monetario) total
Implementado (Sí/No)	Implementado por Sofia Losada

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 Se crea una tabla de hash cuyas llaves son las coordenadas y su valor una lista con los elementos tipo de vehículo y apariciones.	O(N)
Paso 2 Se accede a la lista y se itera sobre el elemento vehículos para comparar con el parámetro. Si el elemento es igual al parámetro se añade al un arbol el valor (key: las apariciones; value: vehículo)	O(n)
Paso 3	O(M)

Por medio de un ciclo while se busca en el árbol la cantidad M de valores máximos. Se obtiene la llave máxima y su pareja llave valor Se añade la pareja llave valor a una lista (lst_cant_camaras) Se elimina la llave maxima del arbol	
Paso 4 Se itera sobre la lista lst_cant_camaras Se pregunta si el elemento se encuentra en la tabla de hash coordenadas, de ser así se obtiene la coordenada y su id respectivo para añadirlo a la lista lst_id	O(M)
Paso 5 Se itera sobre la lista lst_id Por cada iteración se pregunta si el elemento se encuentra en la malla vial , de no ser así se aproxima aplicando la función haversiana	O(m)
Paso 6 Teniendo los id de todos los vertices en el grafo se implementa primMST para realizar el recorrido más corto y se retorna.	O(V)
TOTAL	***O(N*M)***

Pruebas Realizadas

Las pruebas realizadas se llevaron a cabo en un computador con las siguientes especificaciones:

Procesadores	**1,8 GHz Intel Core i5 de dos núcleos**
Memoria RAM	8 GB 1600 MHz DDR3
Sistema Operativo	MacOS Big Sur 11.7.10

Para las pruebas realizadas, se ingresó las 5 infracciones con mayor prioridad. A continuación, se encuentra el promedio obtenido en tiempo de ejecución y memoria usada a lo largo de 3 intentos.

Entrada	**Tiempo (ms)**	**Memoria (kB)**
2	50.987	28.9872

Análisis

En el algoritmo se implementaron diferentes tipos de estructuras que influyeron en la complejidad temporal. De esta manera se escogió implementar una tabla de Hash pues esta permite acceder fácilmente a los datos. Su implementación dificultó los procesos pues almacenaba los datos como diccionarios, lo que no permitía la iteración fácil sobre ellos. A su vez la implementación del árbol RBT fue una herramienta útil para poder almacenar valores en formato llave-valor de manera que no requirió recorrer todo el árbol para determinar la mayor.

Así mismo, la implementación del algoritmo prim MST para determinar el camino más corto aumentó la complejidad considerablemente, pues en un principio se planteó para que realizara el camino sobre el grafo original, mientras que su implementación más optima es sobre un subgrafo.

La complejidad total está dada por el tamaño de las estructuras que manipuló a lo largo del algoritmo, pues el tamaño de datos en sus recorridos se redujo considerablemente, conforme el algoritmo avanza.

Requerimiento <<6>>

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructuras de datos, número de comparendos más graves, estación de policía desde la cual se quiere calcular las distancias más cortas.
Salidas	Para cada vértice con una de las infracciones de mayor gravedad: Número de nodos que hacen parte de las rutas, IDs de los nodos, arcos que hacen parte, distancia total de la ruta.
Implementado (Sí/No)	Implementado por Juan David Torres.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Obtener los n comparendos más graves.	O(nlogn) N viene del ciclo, logn es el tiempo promedio para extraer cada uno de los valores de un árbol RBT.
Obtener el nodo al que hace parte la estación de policía especificada por el usuario.	O(n)
Utilizar el algoritmo de Djkstra para obtener un subgrafo con las menores rutas posibles desde la estación de policía especificada.	O(ElogV). Peor caso para algoritmo de Dijkstra en una cola de prioridad.
Iterar a través de los n comparendos más graves.	O(n)
Obtener el camino más corto desde la estación a cada uno de los vértices con los n comparendos más graves.	O(E') E' corresponde a un número reducido de arcos que conectan la estación con el nodo correspondiente.

Iterar a través de cada una de las rutas para cada uno de los nodos con un comparendo urgente.	O(n)
Iterar a través de los arcos de la ruta para obtener los vértices, número de nodos, distancia y otros elementos que se piden dentro del requerimiento.	O(E')
TOTAL	***O(n*E')***

Pruebas Realizadas

Las pruebas realizadas se llevaron a cabo en un computador con las siguientes especificaciones:

	Máquina
Procesadores	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
Memoria RAM (GB)	12,0 GB
Sistema Operativo	Windows 11 Home Single Language

Para las pruebas realizadas, se ingresó las 5 infracciones con mayor prioridad en la Estación de Policía de Ciudad Bolívar. Igualmente, distintos prints acompañaron la ejecución del requerimiento para verificar qué partes del código se tardaban un mayor tiempo. A continuación, se encuentra el promedio obtenido en tiempo de ejecución y memoria usada a lo largo de 3 intentos.

Entrada	**Tiempo (ms)**		**Mer**
5, Estación de Policía de Ciudad Bolívar	407182.94		1492

Análisis

Teóricamente, la peor complejidad es O(n*E'), pues esta viene de dos ciclos que se encuentran anidados el uno dentro del otro. No obstante, debido a los prints que se colocaron dentro del requerimiento, es posible ver que esta parte se ejecutaba rápidamente. Esto se debe a que los ciclos se ejecutan sobre un subgrafo previamente reducido, pues es el grafo resultante de calcular todos los caminos de menor costo desde la estación de policía dada. Por el contrario, la mayor parte del tiempo de ejecución se llevó a cabo en el algoritmo de Dijkstra. La principal razón a lo que esto se puede deber es que el algoritmo de Dijkstra se lleve a cabo sobre la totalidad del grafo de la malla vial. Esto implica que las variables E y V dentro de O(ElogV) tienen un orden de magnitud considerablemente mayor a n y E'. Por ello, es posible explicar la incongruencia con el tiempo teórico que viene de la notación O y el tiempo real de ejecución de cada uno de los componentes.

Requerimiento <<7>>

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructuras de datos, coordenadas del punto inicial, coordenadas del punto final
Salidas	Número de nodos que hacen parte de la ruta, IDs de los nodos, arcos que hacen parte, distancia total de la ruta.
Implementado (Sí/No)	Implementado por Juan David Torres.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Verificar que las coordenadas ingresadas se encuentren dentro de los límites de la ciudad.	O(1)
Recorrer las coordenadas de los nodos en la malla vial para determinar el punto más cercano a las coordenadas provistas tanto para el punto inicial como el punto final.	O(E)
Calcular la distancia haversiana entre las coordenadas provistas y el nodo en cuestión. Actualizar si se encuentra una distancia menor.	O(1)
Utilizar el algoritmo de Djkstra para obtener un subgrafo con las menores rutas posibles desde el nodo más aproximado a las coordenadas iniciales.	O(ElogV). Peor caso para algoritmo de Dijkstra en una cola de prioridad.
Obtener la pila con la ruta entre ambos puntos.	O(E'), donde E' es el número de arcos para la ruta entre los puntos especificados.
Obtener el camino más corto desde la estación a cada uno de los vértices con los n comparendos más graves.	O(E') E' corresponde a un número reducido de arcos que conectan la estación con el nodo correspondiente.
Iterar a través de los arcos del MST para obtener los vértices, número de nodos, distancia y otros elementos que se piden dentro del requerimiento.	O(E')
TOTAL	***O(ElogV)***

Pruebas Realizadas

Igualmente, distintos prints acompañaron la ejecución del requerimiento para verificar qué partes del código se tardaban un mayor tiempo. A continuación, se encuentra el promedio obtenido en tiempo de ejecución y memoria usada a lo largo de 3 intentos.

Las pruebas realizadas se llevaron a cabo en un computador con las siguientes especificaciones:

	Máquina
Procesadores	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42 GHz
Memoria RAM (GB)	12,0 GB
Sistema Operativo	Windows 11 Home Single Language

Entrada	Tiempo (ms)	Memoria (kB)
•Origen: Latitud: 4.60293518548777, Longitud: -74.06511801444837 • Destino: Latitud: 4.693518613347496, Longitud: -74.13489678235523	36265.049	235.461

Análisis

En este caso, la mayor complejidad teórica es derivada de la ejecución del algoritmo de Dijkstra O(ElogV). Debido a que en el requerimiento no se encuentran ningunos ciclos anidados dentro del otro, no se presenta una mayor complejidad teórica a la que provee el algoritmo de Dijkstra. Esto se pudo corroborar en las pruebas realizadas, pues la mayor parte del tiempo de ejecución se encontraba en la sección del código encargada de llamar a la función del algoritmo. Es posible explicar su alto tiempo de ejecución por la gran magnitud de los datos que determinan la complejidad temporal del algoritmo. En este caso, V corresponde al número de intersecciones en toda la malla vial de la ciudad de Bogotá, mientras que E corresponde a los arcos del grafo, o en otras palabras, las calles que conectan las intersecciones. Esto implica que, dado el punto inicial, el cálculo de la menor distancia se debe hacer para un número elevado de vértices.