Gabriela Caballero - 201812917- g.caballero

Alvaro Plata - 201820098 - ad.plata

**Listado de requerimientos funcionales**

**Proyecto 3**

**Parte inicial:**

**Datos de entrada:** Archivos con la información de los vértices y arcos que conforman la malla vial

**Descripción:** Cargar el grafo de la malla vial a partir de los archivos: Se lee primero el archivo de los vértices, creando uno nuevo con la información de cada línea. Luego se lee el archivo de los arcos, creando uno nuevo con la información de cada línea y uniendo los dos vértices indicados. Se añade el mismo arco a la lista de arcos de ambos vértices

**Datos de salida:** Se informa la cantidad de vértices y arcos cargados, la cantidad de componentes conectadas, la cantidad de vértices que tienen las 5 componentes conectadas más grandes y se grafica el mapa con la información del grafo.

**Complejidad:** La complejidad de cargar el grafo es O(N + K), siendo N la cantidad de vértices y K la cantidad de arcos, pues se debe cargar cada uno una sola vez.

**Datos de entrada:** El grafo ya cargado

**Descripción:** Añadir la información del costo de tiempo y velocidad a cada arco: Primero se crea una tabla de hash, en la que se guardarán los tiempos promedio de los viajes uber, siendo la llave un String con la zona origen y la zona destino. Se lee el archivo de los viajes uber semanales del primer trimestre y se guarda la información de cada tiempo promedio en la tabla de hash. Luego, se recorre cada vértice del grafo y por cada vértice se recorre su lista de arcos. Por cada arco se tiene la información de la zona origen y destino de los dos vértices a los que conecta, entonces se busca la entrada correspondiente en la tabla de hash, se aumenta en 1 un contador para saber cuántos viajes hay entre esas dos zonas y se suma su tiempo promedio a otro contador. Al final, se revisa si la cantidad de viajes realizados es mayor a 0, en ese caso, el costo de tiempo del arco será el contador de tiempos promedio sobre el contador de cantidad de viajes. En caso contrario, se revisa si ambos vértices corresponden a la misma zona, y si es así se le asigna un costo de 10, si no, un costo de 100. Para ese mismo arco el costo de velocidad será el costo distancia sobre el costo tiempo.

**Datos de salida:** Este requerimiento se encuentra contenido en el requerimiento 1.

**Complejidad:** La complejidad es O(K + N), siendo K el número de arcos que hay en el grafo (porque se debe revisar cada arco una vez) y N el número de viajes Uber semanales del primer trimestre, (porque se debe añadir la información del tiempo promedio de cada viaje a la tabla de hash)

**3)**

**Datos de entrada:** El grafo cargado con la información de los archivos.

**Descripción:** Crear y cargar el archivo JSON con la información del grafo. Para crear el archivo JSON se crean dos JSONArrays, uno con la información de las intersecciones (latitud, longitud, id de la zona) y uno con la información de los arcos.

Para cargarlo, se debe leer primero la información de los vértices, crearlos y luego crear los arcos para conectarlos.

**Datos de salida:** Se informa la cantidad de vértices y arcos cargados, la cantidad de componentes conectadas, la cantidad de vértices que tienen las 5 componentes conectadas más grandes y se grafica el mapa con la información del grafo.

**Complejidad:** La complejidad de cargar el grafo es O(N + K), siendo N la cantidad de vértices y K la cantidad de arcos, pues se debe cargar cada uno una sola vez.

**4)**

**Datos de entrada:** La latitud y longitud a buscar.

**Descripción:** Encontrar el ID del vértice más cercano por distancia Haversine. Se recorren todos los vértices del grafo y para cada uno se calcula su distancia hasta la ubicación dada por el usuario. El ID del vértice será el que tenga la distancia más corta.

**Datos de salida:** Se informa el ID del vértice que haya tenido la menor distancia hasta la ubicación dada.

**Complejidad:** O(N), pues se debe recorrer cada vértice una sola vez.

**Parte A:**

**Datos de entrada:** Latitud y longitud origen, latitud y longitud destino.

**Descripción:** Encontrar el camino de menor costo (tiempo promedio) para un viaje entre dos ubicaciones. Primero, se debe encontrar el vértice más cercano a la ubicación origen y la ubicación destino con el algoritmo utilizado para el requerimiento inicial, estos serán los vértices origen y destino. Se usará el algoritmo de Dijkstra implementado con un MinHeap, ya que esta estructura permite tener control de cuál es el camino más corto hasta el momento y tiene una menor complejidad a la implementación con arreglos de tamaño fijo.

**Datos de salida:**

* **En consola:** El método retorna el conjunto de vértices que conforman el camino contenidos en un objeto Iterable<Edge<Integer,Interseccion>>. Se imprime el camino a seguir, informando el total de vértices, sus Ids, latitud y longitud, el costo mínimo y la distancia estimada.
* **En mapa:** Muestra el camino resultante y los vértices origen y destino.

**Complejidad:** O(ELogV), siendo V la cantidad de vértices y E la cantidad de Arcos.

**Datos de entrada:** Número de vértices a buscar

**Descripción:** Determinar los n vértices con menor tiempo promedio. Se utiliza un MaxHeap cuya capacidad máxima sea la cantidad de vértices introducido por el usuario. Se recorre cada vértice del arreglo y se calcula su velocidad promedio, se revisa si el MaxHeap aún tiene capacidad, si sí, se agrega, si no, se revisa si la velocidad promedio del vértice que acabamos de calcular es menor a la del mayor elemento del MaxHeap. En ese caso se hace pop() al MaxHeap y se agrega el vértice en el que estamos, en caso contrario continuamos con el siguiente vértice. Al final tendremos un MaxHeap con los n vértices con menor velocidad promedio.

Para saber cuáles vértices pertenecen a las componentes conectadas de los n vértices hallados, se usa el método getCC() del grafo.

**Datos de salida:**

* **En consola:** Se muestran los n vértices resultantes en la consola de texto (su identificador, su ubicación (latitud, longitud), ordenados de menor a mayor por la velocidad promedio del vértice. Se informa el número de componentes conectadas que hay entre estos vértices y por cada componente se informan los ids de los vértices que la componen.
* **En mapa:** Marcar los n vértices resultantes con un color y la componente conectada más grande de éstos vértices con otro (con sus vértices y arcos).

**Complejidad:** O(K\*V), siendo V la cantidad de vértices en el arco y K un número que varía según la cantidad de arcos adyacentes que tenga cada vértice.

**Datos de entrada:** No hay.

**Descripción:** Se encuentra la componente conectada más grande como se encontró en los requerimientos iniciales. De esta componente se selecciona un vértice al azar y desde éste se calcula el MST utilizando Prim, implementado con una cola de prioridad para guardar los arcos y poder acceder al arco de menor costo.

**Datos de salida:**

* **En consola:** Imprime el tiempo que se demora el algoritmo, e información del árbol (el total de vértices en el componente, los vértices, los arcos incluidos y el costo total).
* **En mapa:** Muestra el árbol generado en el mapa, sus arcos y vértices.

**Complejidad:** O(E\*LogV), siendo E la cantidad de Arcos y V la cantidad de vértices.

**Parte B:**

**Datos de entrada:** Latitud y longitud origen ,latitud y longitud destino.

**Descripción:** Encontrar el camino de menor costo (distancia Haversine) para un viaje entre dos localizaciones geográficas. Primero, se debe encontrar el vértice más cercano a la ubicación origen y la ubicación destino con el algoritmo utilizado para el requerimiento inicial, estos serán los vértices origen y destino. Se usará el algoritmo de Dijkstra implementado con un MinHeap, ya que esta estructura permite tener control de cuál es el camino más corto hasta el momento y tiene una menor complejidad a la implementación con arreglos de tamaño fijo.

**Datos de salida:**

* **En consola:** El método retornael camino a seguir, informando el total de vértices, sus vértices , el costo mínimo y la distancia estimada.
* **En mapa:** Muestra el camino resultante.

**Complejidad:** O(ELogV), siendo V la cantidad de vértices y E la cantidad de Arcos.

**Datos de entrada:** Latitud y longitud origen, y tiempo.

**Descripción:** Buscar cuales vértices son alcanzables desde el origen por un tiempo dado. Para este requerimiento se encuentra el MST según el costo del tiempo implementado con una cola de prioridad desde el vértice más cercano a la ubicación dada por el usuario. Después, se debe revisar cuál es el costo para para llegar a cada vértice según el MST, sabiendo que es el menor costo posible, y si ese costo es menor o igual al tiempo dado por el usuario se agrega a una lista encadenada, que al final contendrá todos los vértices que son alcanzables en ese tiempo.

**Datos de salida:**

* **En consola:** Imprimir los identificadores y la ubicación (lat, lon) de los vértices alcanzables.
* **En mapa:** Marcar la localización de origen de un color y las localizaciones de los vértices alcanzables en otro.

**Complejidad:** O(E\*LogV + V), ya que es la complejidad de crear el MST, más la complejidad de revisar cada vértice después de creado.

**Datos de entrada:** No hay.

**Descripción:** Se encuentra la componente conectada más grande como se encontró en los requerimientos iniciales. De esta componente se selecciona un vértice al azar y desde éste se calcula el MST utilizando Kruskal, implementado con una cola de prioridad para guardar los arcos y poder acceder al arco de menor costo.

**Datos de salida:**

* **En consola:** Imprime el tiempo que se demora el algoritmo, e información del árbol (el total de vértices en el componente, los vértices, los arcos incluidos y el costo total).
* **En mapa:** Muestra el árbol generado en el mapa, sus arcos y vértices.

**Complejidad:** O(ElogE), siendo E el número de arcos del grafo.

**Parte C:**

**Datos de entrada:** No hay.

**Descripción:** Construir un nuevo grafo simplificado no dirigido de las zonas Uber, donde cada zona es representada con un único vértice. Para este requerimiento se deben recorrer todos los vértices. Creamos una HashTable para guardar en ella la información de cada zona y las zonas a las cuales están conectadas, es decir, las llaves serán los identificadores de cada zona, y como nuestra implementación permite almacenar varios Values en una sola Key, los values serán las zonas a las cuales está conectada. Esta información se obtendrá al analizar cada vértice: Se revisa su zona, si no existe ningún elemento en el HashTable con esa zona se crea, y se agrega un Value a esa zona con la zona de destino. Así al final tendremos un HashTable con la información para crear los nuevos vértices y los nuevos arcos.

La posición de cada nuevo vértice será la ubicación del primer vértice que se encuentre de esa zona.

**Datos de salida:**

* **En consola:** Reportar la cantidad de vértices y arcos.
* **En mapa:** Muestra el grafo resultante en Google Maps: sus vértices y sus arcos.

**Complejidad:** O(V), porque se debe recorrer cada vértice y obtener su información una vez

**Datos de entrada:** id zona origen, id zona destino.

**Descripción:** Calcular el camino de costo mínimo basado en el tiempo promedio entre dos zonas. Se realiza el mismo procedimiento que con el requerimiento 1 de la parte A, pero esta vez sobre el nuevo grafo creado en el requerimiento anterior. Para mostrar el tiempo promedio entre estas zonas se realiza igual que con el requerimiento inicial, con una HashTable.

**Datos de salida:**

* **En consola:** Tiempo que toma el algoritmo en encontrar la solución y del camino resultante: su secuencia de vértices y zonas , y su costo total. Además mostrar el tiempo promedio entre las dos zonas reportado por el archivo de resolución semanal de Uber.
* **En mapa:** Muestra el grafo resultante en Google Maps: sus vértices y sus arcos.

**Complejidad:** O(ELogV), siendo V la cantidad de vértices y E la cantidad de Arcos.

**Datos de entrada:** ID de la zona origen

**Descripción:** Calcular los caminos de menor longitud a todas las zonas alcanzables, después seleccionar el camino más largo, este camino será el camino a la zona más distante desde el origen. Para este requerimiento se usará el algoritmo BFS, ya que este garantiza el camino más corto según la cantidad de arcos entre cualquier par de vértices, y se retornará un Iterable con los arcos que hagan parte de la solución, similar a cómo se encuentra un MST.

**Datos de salida:**

* **En consola:** tiempo que toma el algoritmo en encontrar la solución (en milisegundos) y del camino resultante más largo: su secuencia de vértices y zonas, y su número total arcos.
* **En mapa:** Muestra el camino más distante desde la zona de origen, sus vértices y sus arcos.

**Complejidad:** O(V), donde V es la cantidad de vértices en el grafo de zonas.