Universidad de los Andes

Ingeniería de Sistemas y Computación
ISIS1206 – Estructura de Datos
Taller 7

Taller 7: Grafos

Objetivos

- Estudiar, implementar, probar y documentar la estructura de datos grafo no dirigido
- Utilizar adecuadamente herramientas para el desarrollo de software en equipos

Lectura Previa

Estudiar la teoría de los grafos dirigidos con peso. Consultar las secciones 4.1 a 4.4 del libro guía *Algorithms* de Sedgewick y Wayne.

Las Fuentes de Datos

La malla vial de Bogotá, representada como un grafo donde los vértices son las intersecciones de las calles y los arcos las calles, puede ser utilizada en una gran cantidad de aplicaciones para navegación terrestre. A continuación, se mostrarán unos ejemplos con el contenido de los archivos de *bogota_vertices.txt* y *bogota_arcos.txt*:

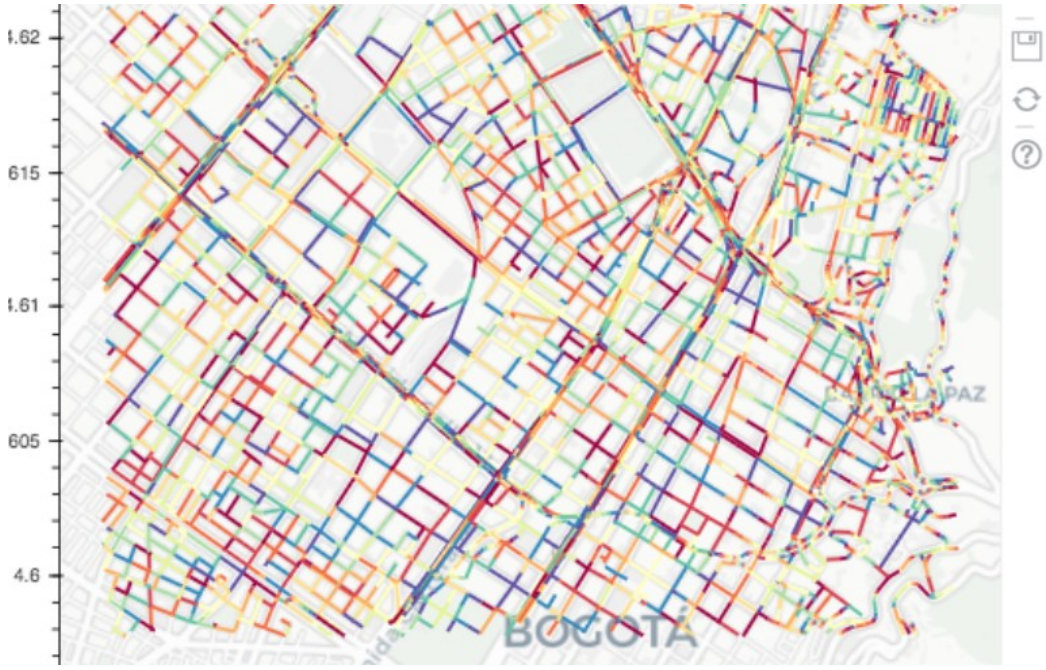
Se representarán las **intersecciones de la malla vial** como los **vértices del grafo**. Estas intersecciones están representadas en el archivo *bogota_vertices.txt* con id, longitud, latitud y MOVEMENT_ID (zona de uber):

```
id;long;lat;MOVEMENT_ID
0;-74.08921298299998;4.582989396000016;275
1;-74.08952746199999;4.582560966000016;275
2;-74.09389220199999;4.576679366000008;684
…
```

De cada intersección se debe almacenar solamente cuatro datos: el identificador único de la intersección (id), longitud (lon), latitud (lat), y zona Uber (MOVEMENT_ID). Cada vértice es de tipo intersección.

Las vías de la malla vial que conectan las intersecciones están almacenadas en el archivo *bogota_arcos.txt*. Estas vías se representan como un conjunto de adyacencias donde el primer campo especifica el id del vértice origen, y los demás campos son los ids de sus vértices adyacentes:

0 1 733
1 49
…



Lo que su grupo debe hacer (parejas)

Parte 1 – Configuración del Repositorio

1. Cree en bitbucket un repositorio llamado T7_201920. Al momento de crearlo recuerde la URL que se muestra en la parte superior derecha de la página de bitbucket: Por ejemplo
   Repository_url = https://login-usuario@bitbucket.org/login-usuario/T7_201920.git
   donde login-usuario corresponde a su login en Bitbucket.
2. Cree el README del repositorio donde aparezcan los nombres y códigos de los miembros del grupo de trabajo.
3. Realice el procedimiento para crear el directorio en su computador de trabajo para que relacione este directorio con el repositorio remoto T7_201920. El trabajo debe repartirse entre ambos estudiantes del grupo.

Parte 2 – Desarrollo

4. Implemente la estructura de datos GrafoNoDirigido que soporte vértices con llave genérica (*K*). El grafo debe ser almacenado utilizando listas de adyacencias. No se tendrá en cuenta la dirección de la vía.

5. El API que debe implementar está compuesto, al menos, por las siguientes operaciones:

Operación	Descripción
`Graph (int V)`	Crea un grafo No dirigido de tamaño V vértices y sin arcos
`int V()`	Número de vértices
`int E()`	Número de arcos. Cada arco No dirigido debe contarse una única vez.
`void addEdge(K idVertexIni, K idVertexFin, double cost)`	Adiciona el arco No dirigido entre el vértice IdVertexIni y el vértice IdVertexFin. El arco tiene el costo *cost*.
`V getInfoVertex(K idVertex)`	Obtener la información de un vértice. Si el vértice no existe retorna null.
`void setInfoVertex(K idVertex, V infoVertex)`	Modificar la información del vértice idVertex
`double getCostArc(K idVertexIni, K idVertexFin)`	Obtener el costo de un arco, si el arco no existe, retorna -1
`void setCostArc(K idVertexIni, K idVertexFin, double cost)`	Modificar el costo del arco entre los vértices idVertexIni e idVertexFin
`void addVertex(K idVertex, V infoVertex)`	Adiciona un vértice con un Id único. El vértice tiene la información InfoVertex.
`Iterable <K> adj (K idVertex)`	Retorna los identificadores de los vértices adyacentes a idVertex
`void uncheck()`	Desmarca todos los vértices del grafo
`void dfs(K s)`	Ejecuta la búsqueda de profundidad sobre el grafo con el vértice s como origen. Los vértices resultado de la búsqueda quedan marcados y deben tener información que pertenecen a una misma componente conectada.
`int cc()`	Obtiene la cantidad de componentes conectados del grafo. Cada vértice debe quedar marcado y debe reconocer a cuál componente conectada pertenece. En caso de que el grafo esté vacío, retorna 0.
`Iterable<K> getCC(K idVertex)`	Obtiene los vértices alcanzados a partir del vértice idVertex después de la ejecución de los metodos dfs(K) y cc().

6. Diseñe escenarios de prueba para probar el correcto funcionamiento de la estructura.

7. Construya sets de pruebas (es decir casos de prueba en JUnit) para verificar y validar la implementación del API.

8. Desarrolle un método para cargar el grafo a partir de las fuentes de datos. El objetivo es construir un **Grafo No Dirigido** que represente la malla vial de Bogotá tomando como base la información de los archivos *bogota_vertices.txt y bogota_arcos.txt.* En caso de que un id de vértice encontrado dentro del archivo de arcos, no esté en el archivo de vértices, se debe omitir la creación de ese arco. Para cada arco se debe almacenar la distancia *Haversine* entre sus dos vértices. La fórmula Haversine calcula la distancia en kilómetros entre dos localizaciones geográficas sobre la superficie de la tierra. Esta distancia Haversine se debe almacenar como costo del arco. Verifique que el grafo quedó bien creado en memoria y reporte la cantidad de vértices y arcos del grafo por consola.

Sugerencia 1: Utilizar un API o código ya disponible para calcular la distancia *Haversine* entre dos puntos. Por ejemplo: https://github.com/jasonwinn/haversine/blob/master/Haversine.java

Sugerencia 2: En caso de obtener un error por desbordamiento de la pila (Stack overflow error), puede incrementar el tamaño de la pila a 4M u 8M usando el argumento –Xss, p. ej: *-Xss4M*

9. Defina un esquema JSON para persistir el grafo; este grafo extendido con las distancias Haversine se va a utilizar en el proyecto 3.

10. Desarrolle un método para cargar el grafo persistido en el paso 9, leyendo el archivo JSON definido por usted, verifique que el grafo quedó bien creado en memoria y reporte la cantidad de vértices y arcos del grafo por consola.

11. Desarrolle un método que retorne la cantidad de componentes conectados en el grafo construido utilizando los métodos de búsqueda por profundidad (dfs) y componentes conectados (cc) provistos por el API.

12. Grafique con ayuda de Google Maps la parte del grafo resultante de la carga de la zona delimitada por las siguientes coordenadas: Longitud min= -74.094723, Longitud max= -74.062707, Latitud min= 4.597714 y Latitud max= 4.621360. Pintar los nodos intersección (con círculos), y los arcos (que conectan los nodos).

Sugerencia 3: Utilizar el API JxMaps (https://www.teamdev.com/jxmaps) ó mapbox (https://docs.mapbox.com/android/java/overview/#installation) para Java
Restricción de uso: El uso de JxMaps tiene una restricción de tiempo en la licencia. En la página web (https://www.teamdev.com/jxmaps#evaluate) un estudiante puede solicitar una licencia de proyecto académico, gratuita, pero esta licencia tiene una restricción de uso de **30 días**.

Importante: Como el taller 7 y el proyecto 3 van a tener el requerimiento de despliegue de mapas de Google Maps, hay que administrar el tiempo de su licencia para que alcancen a hacer los 2 desarrollos. Para los grupos se les recomienda primero solicitar la licencia de uno de los estudiantes y si se requiere luego solicitar la licencia del segundo estudiante. Para los grupos de 1 solo estudiante se recomienda hacer el mejor uso posible de su licencia en los **30 días que tienen de licencia**.

Entrega

1. Para hacer la entrega del taller usted debe agregar a su repositorio los usuarios de los monitores y su profesor, siguiendo las instrucciones del documento “Guía Creación de Repositorios para Talleres y Proyectos.docx”.
2. Entregue su taller por medio de BitBucket. Recuerde, si su repositorio No tiene acceso al profesor ni a los monitores, su taller no será calificado por más de que lo haya desarrollado.