**Proyecto 3 Estructuras de Datos ( ISIS-1206)**
**Documento de diseño**
**Por: Santiago Talero & Daniel S. Londoño ( 201821994 / 201821363 )**

## 1. Requerimientos funcionales

- **Req 1.1.**
  - **Datos de entrada:** N/A
  - **Descripción:** Cargar el Grafo No Dirigido (grafo más grande) de la malla vial de la ciudad completa de Bogotá. Informa el total de vértices y el total de arcos que definen el grafo cargado.
  - **Datos de salida:** N/A
  - **Complejidad Estimada: O(V+E )**
  - **Estructura de datos utilizada:** N/A

- **Req 1.2.**
  - **Datos de entrada:** N/A
  - **Descripción:** Al grafo creado, se debe agregar la información de costo. El grafo va tener 3 tipos de costo en sus arcos. El primer tipo de costo de un arco es la distancia Harversine (en kilómetros) entre las localizaciones geográficas de los vértices que conecta. El segundo tipo de costo de un arco es el tiempo de viaje entre sus vértices. El tercer tipo de costo es la velocidad del arco, calculada como su distancia dividida por su tiempo.
  - **Datos de salida:** N/A
  - **Complejidad Estimada:** N/A
  - **Estructura de datos utilizada:** N/A

- **Req 1.3.**
  - **Datos de entrada:** N/A
  - **Descripción:** Completar el grafo con los tres costos para cada arco a partir de los datos de Uber y la malla vial de Bogotá. Crear un archivo JSON para guardarlo y un método que permita cargar el nuevo grafo JSON generado.
  - **Datos de salida:** Archivo JSON.
  - **Complejidad Estimada:** N/A
  - **Estructura de datos utilizada:** N/A

- **Req 1.4**
  - **Datos de entrada:** Localización geográfica ( latitud, longitud)
  - **Descripción:** Encuentra el Id del Vértice de la malla vial más cercano por distancia Haversine.
  - **Datos de salida:** Id del vértice.
  - **Complejidad Estimada: O(V)**
  - **Estructura de datos utilizada:** *Queue. Utilizaremos una Queue ya que resulta menos engorroso de iterar para futuros usos.*

- **Req A1**
  - **Datos de entrada:** Localización geográfica origen ( latitud 1, longitud 1) y localización geográfica de destino ( latitud 2, longitud 2)
  - **Descripción:** Encuentra el camino de costo mínimo para un viaje entre dos localizaciones geográficas de la ciudad. Estas localizaciones deben aproximarse a las localizaciones más próximas en la malla vial.
  - **Datos de salida:** Se muestra el camino a seguir, resultante en Google Maps.
  - **Complejidad Estimada: O(E+V)**
  - **Estructura de datos utilizada:** *Queue. Usaremos esta estructura de datos ya que necesitamos encontrar y modelar el camino más económico. Esto lo vamos a mostrar como una Queue de tipo vértice. Es decir, una Queue con los vértices que exponen conjuntamente el camino más económico.*

- **Req A2**
  - **Datos de entrada:** Número de vértices
  - **Descripción:** Determina los *n* vértices con menor velocidad promedio en la ciudad de Bogotá.
  - **Datos de salida:** Los *n* vértices.
  - **Complejidad Estimada: O(V)**
  - **Estructura de datos utilizada:** *Queue. En una Queue de tipo vértice guardaremos los n vértices con menor velocidad promedio por motivos de simplicidad y eficiencia en cuanto a la complejidad.*

- **Req A3**
  - **Datos de entrada:** N/A
  - **Descripción:** Calcular un árbol de expansión mínima (MST) con criterio distancia, utilizando el algoritmo de Prim, aplicado al componente conectado (subgrafo) más grande de la malla vial de Bogotá.
  - **Datos de salida:** MST, tiempo que toma al algoritmo en encontrar la solución. Del árbol generado se debe mostrar el total de vértices en el componente, los vértices (identificadores), los arcos incluidos (Id vértice inicial e Id vértice final) y el costo total (distancia en Km). Vértices y arcos del árbol generado resultante en Google Maps.
  - **Complejidad Estimada: O(E log(V))**
  - **Estructura de datos utilizada:** *Priority Queue. Utilizaremos una Priority Queue ya que operacionalmente se lleva muy bien con el algoritmo Prim para calcular MST´s.*

- **Req B1**
  - **Datos de entrada:** Localización geográfica de origen ( latitud, longitud), localización geográfica de destino ( latitud, longitud).
  - **Descripción:** Encontrar el camino de menor costo (menor distancia Haversine) para un viaje entre dos localizaciones geográficas de la ciudad.

Estas localizaciones deben aproximarse a las localizaciones más próximas en la malla vial.

- **Datos de salida:** Tiempo que toma el algoritmo en encontrar la solución (en milisegundos). Del árbol generado se debe mostrar el total de vértices en el componente, los vértices (identificadores), los arcos incluidos (Id vértice inicial e Id vértice final) y el costo total (distancia en Km). Vértices y arcos del árbol generado resultante en Google Maps.
- **Complejidad Estimada: O(E+V)**
- **Estructura de datos utilizada:** *Queue. Usaremos esta estructura de datos ya que necesitamos encontrar y modelar el camino más económico. Esto lo vamos a mostrar como una Queue de tipo vértice. Es decir, una Queue con los vértices que exponen conjuntamente el camino más económico*

- **Req B2**
  - **Datos de entrada:** Latitud y longitud de origen, Tiempo
  - **Descripción:** Indica cuáles vértices son alcanzables para un tiempo T (en segundos) dado por el usuario. La localización de origen debe aproximarse a la localización más próxima en la malla vial.
  - **Datos de salida:** Los identificadores y la ubicación (latitud, longitud) de los vértices alcanzables en un tiempo T a partir de la localización de origen.
  - **Complejidad Estimada: O(V)**
  - **Estructura de datos utilizada:** *Queue. Usaremos una Queue de tipo vértice para almacenar los vértices que cumplen con esas condiciones por motivos de simplicidad, gestión de datos, y eficiencia con respecto al orden de complejidad.*

- **Req B3**
  - **Datos de entrada:** N/A
  - **Descripción:** Calcular un árbol de expansión mínima (MST) con criterio distancia, utilizando el algoritmo de ***Kruskal***, aplicado al componente conectado (subgrafo) más grande de la malla vial de Bogotá.
  - **Datos de salida:** Tiempo que toma el algoritmo en encontrar la solución (en milisegundos). Del árbol generado se debe mostrar el total de vértices en el componente, los vértices (identificadores), los arcos incluidos (Id vértice inicial e Id vértice final) y el costo total (distancia en Km). Vértices y arcos del árbol generado resultante en Google Maps.
  - **Complejidad Estimada:( E log(V))**
  - **Estructura de datos utilizada:** *Indexed Minimum PriorityQueue, Union Find. Usaremos una IndexMinPQueue de tipo vértice para almacenar los vértices por motivos de simplicidad, gestión de datos, y eficiencia con respecto al orden de complejidad y Union Find con la finalidad de que el algoritmo de Kruskal identifique ciclos y así no causar ningún problema.*