**Diseño de proyecto**

**Carga de datos**

* Implementación: Utilizar lectura de archivos GSON para cargar los vértices y los arcos de la maya vial en un grafo dirigido y se utilizan las propiedades del grafo para conocer el número de vértices y arcos. A medida que se carga un vértice este se asigna a una zona en una tabla de símbolos que indica el conjunto de IDs de vértices que hay en un cuadrante del mapa. Posteriormente se utiliza lectura de archivos JSON para cargar los comparendos y las estaciones policía y guardarlas, cada una, en una lista encadenada, y se va llevando el registro del comparendo y la estación de mayor OBJECTID. Para cada estación y comparendo cargado se realiza la parte inicial asignando un vértice a cada uno.
* Parámetros de entrada: ruta del archivo Json a leer
* Retorno: Comparendo y estación con mayor OBJECT\_ID
* Orden del algoritmo: E+V+(L+K)\*V/1000000

**Parte Inicial**

**1.**

* Implementación: Se ubica la zona en la tabla de símbolos a la cual pertenece la coordenada recibida por parámetro, y se busca en la lista de IDs de vértices de la zona aquel que sea más cercano a dicha coordenada y se retorna el ID del vértice.
* Parámetros: Double de latitud y double de longitud
* Retorno: Vértice más cercano
* Orden: V/1000000

**2.**

* Implementación: Para cada comparendo se busca el vértice más cercano según sus coordenadas utilizando el método del punto anterior y se le asigna al comparendo un nuevo atributo que es el ID de su vértice encontrado, también se le asigna al vértice un nuevo comparendo en un atributo que es la lista de comparendos impuestos en dicho vértice
* Parámetros: Comaprendo y vertice
* Retorno: Ninguno
* Orden: V/1000000

**3.**

* Implementación:Se recorren todos los arcos y se les asigna la suma de la cantidad de comparendos en sus vértices de origen y de llagada, este valor se obtinee sumando el atmaño de las listas encadenadas de comparendos del vertice de origen y del de destino
* Parámetros: el grafo
* Retorno: ninguno
* Orden: V

**4.**

* Implementación: Para cada estación se busca el vértice más cercano según sus coordenadas utilizando el método del punto anterior y se le asigna a la estación un nuevo atributo que es el ID de su vértice encontrado, también se le asigna al vértice un nuevo atributo que es la estación que contiene, este atributo queda en null para la mayoría de vértices
* Parámetros: Estación y vértice
* Retorno: Ninguno
* Orden: V/1000000

**Parte A**

1A.

* Implementación: Se utiliza el algoritmo creado previamente para ubicar los vértices de inicio y de fin según las coordenadas recibidas por parámetro. Posteriormente se utiliza el algoritmo de Dijkstra para encontrar el camino más corto desde el vértice de origen hasta el vértice final. Se le pone una condición de parada al Dijkstra para cuando haya encontrado un camino al vértice final. Luego se utiliza el vector edgeTo[] del algoritmo para encontrar los vértices que hacen parte del camino y se grafican sobre un mapa y se imprime su información.
* Parámetros de entrada: doubles de las coordenadas de cada ubicación
* Retorno: Lista con la distancia del camino y con una pila de los vértices que hacen parte del camino.
* Orden del algoritmo: V\*E\*logV

2A.

* Implementación: Se copian todos los comparendos a una cola de prioridad donde la prioridad está dada por la gravedad. Posteriormente se hacen M eliminaciones de la cola y se usa el algoritmo de dijkstra para encontrar el árbol de caminos más cortos entre los comparendos más graves y esta se grafica como la red de cámaras. Se retorna un alista encadenada con la distancia de la red y con las pilas de vértices que hacen parte de la red.
* Parámetros de entrada: Entero del número de vértices a buscar
* Retorno: Lista encadenada con la distancia de la red y las pilas de vértices
* Orden del algoritmo: V\*E\*logV

**Parte C**

1C.

* Implementación: Se copian los comparendos a una cola de prioridad donde la prioridad es la gravedad y se hacen M eliminaciones. Y para cada comparendo se usa el algoritmo de dijkstra para ubicar el vértice con estación más cercano, para ello se pone una condición de parada al algoritmo que es al encontrar un camino a un vértice con una estación. Posteriormente se utiliza el arreglo edgeTo[] para conocer los vértices que hacen parte de cada camino y se grafican en el mapa y se imprimen en la consola.
* Parámetros: Entero del número de comparendos a atender
* Retorno: Lista encadenada de pilas con la distancia de cada camino y los vértices que lo conforman
* Orden del algoritmo: V\*E\*logV

2C.

* Implementación: Primero se crea un nuevo grafo no dirigido. Luego para cada comparendo se utiliza obtiene su camino más corto a una estación con un algoritmo similar al del punto anterior. Posteriormente se agrega al grafo creado únicamente los vértices y arcos que hacen parte de estos caminos. Posteriormente se usa el algoritmo de deapth first search para encontrar los componentes conectados del grafo. Por último se crea una paleta de colores (arreglo con varios colores) y se grafica todo el grafo con los colores correspondientes a cada componente conectado y se imprime la información de los componentes. Para esto se le asigna a cada estación un nuevo atributo de la cantidad de comparendos que atiende para así conocer el tamaño del radio de cada estación a graficar. También se crea una lista encadenada con la cantidad de vértices en cada componente que se usa para imprimir
* Parámetros de entrada: Ninguno
* Retorno: grafo no dirigido
* Orden del algoritmo: E+V+(V\*E\*logV)