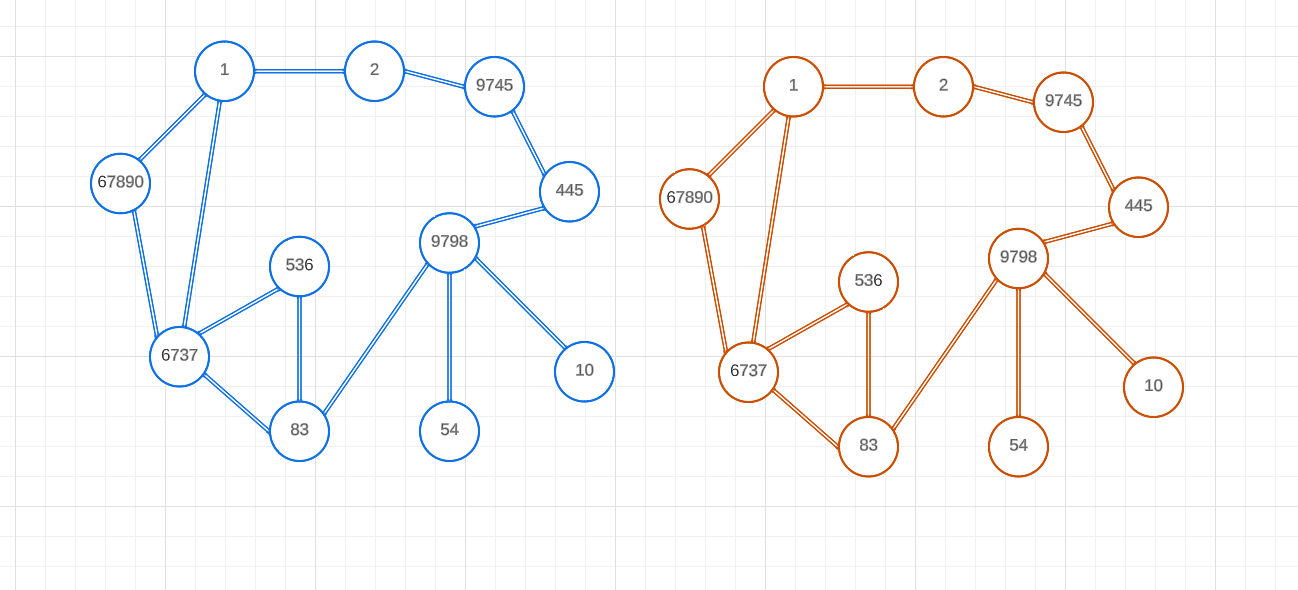
Análisis Reto 4

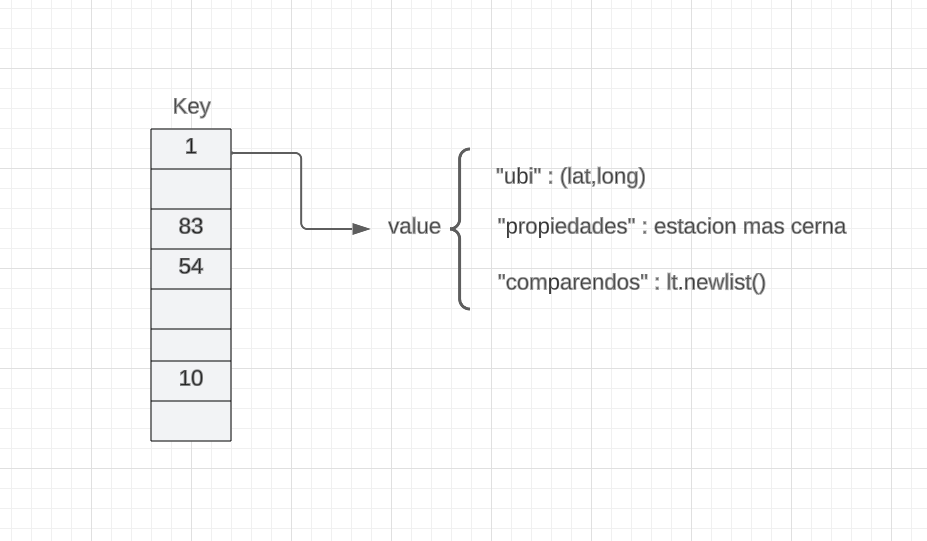
Carga de datos:



|  |  |  |
| --- | --- | --- |
| Representación | | |
| Variable | Tipo | Función |
| vertices | lista | Almacena cada vértice como un diccionario, tal marca el nombre del vértice y su latitud y longitud |
| connections | Grafo-dirigido | Crea el grafo principal con los dos pesos correspondientes, la distancia entre los puntos y la suma de los comparendos entre ellos |
| connections\_o\_comp | Grafo-no dirigido | Crea un grafo no dirigido con los que los pesos del arco sean solo la suma del número de comparendos de ambos vértices |
| mapDatos | Tabla de hash | Lo usamos para guardar cada vértice dentro del hash, lo usamos agregando (usando el vértice como llave) y el valor es un diccionario que tiene como valores, la “ubi”, que es una tupla que almacena la long y lat del vértice, las “propiedades”, que básicamente se usa para almacenar la estación más cercana al vértice, “comparendos” esta es una variable, tipo |
| mapEstaciones | Tabla de hash | Lo usamos para guardar cada estación dentro del hash, lo usamos para ir agregando, el “objeto” como llave y luego sus propiedades como el valor |
| lista\_presentacion\_estaccion  lista\_presentacion\_comparendos | listas | Listas tipo “ARRAY\_LIST” que solo se usaran para mostrar los valores, (aunque esto hará que aumente la temporalidad de la carga) |
|  |  |  |

Análisis grafico de la carga de datos:

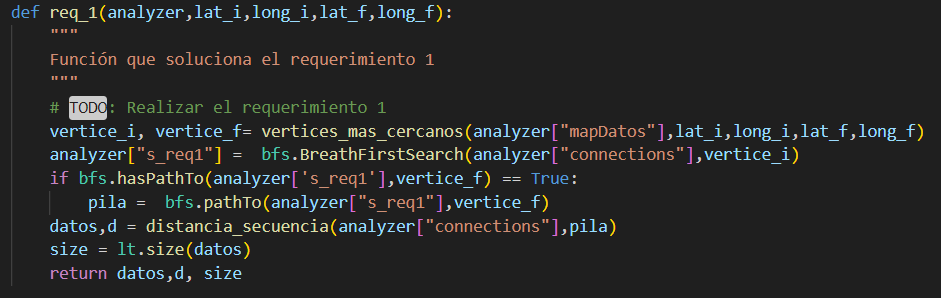
El dos son los grafos de la carga, los arcos azules representan la distancia entre los puntos y los naranjas representan el número de comparendos entre ellos, como no está dirigido el cardo debido a que no hay un orden de prioridad al momento de carga los datos, el arco es de ida y vuelta.



Así se almacenan los datos en las llaves usando como valor de la key, el vértice o la malla vial y cómo valor, la tupla de que su ubicación, la propiedad que es la estación más cercana, y los comparendos ces una lista donde se van agregando cada comparendo perteneciente a ese vértice a esa lista.

# **Requerimiento <<n1>>**

## **Descripción**

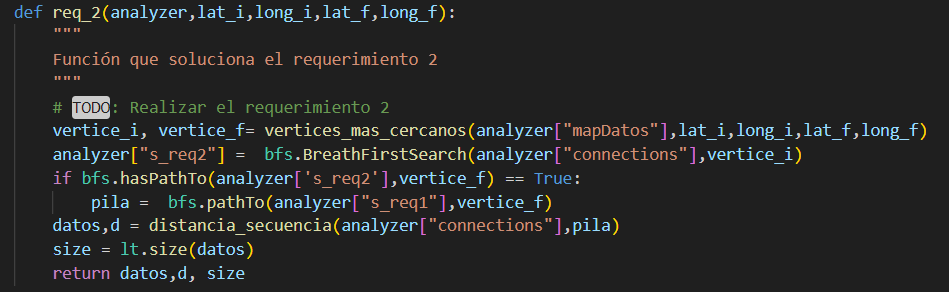


En el primer requerimiento lo que hicimos fue tomar los datos ingresados por parámetro y utilizamos la fórmula de haversine que nos sirve para encontrar los vértices más cercanos a una tupla de coordenadas, luego hicimos un BFS desde el vértice inicial para determinar si existe un camino al vértice final con haspathto y si es así que regrese la secuencia que se genera con pathto. Además, utilizamos distancia\_secuencia que nos sirve para determinar la distancia total que recorre en kilómetros. En este caso utilizamos BFS y no DFS porque, aunque no se necesita el camino más corto, con el DFS lanza que se excede el recorrido.

|  |  |
| --- | --- |
| **Entrada** | Punto de origen y destino (en latitud y longitud) |
| **Salidas** | La secuencia de vértices del camino, la distancia y el total de vértices. |
| **Implementado (Sí/No)** | Por Juan Lago |

# **Requerimiento <<n2>>**

**Descripción**

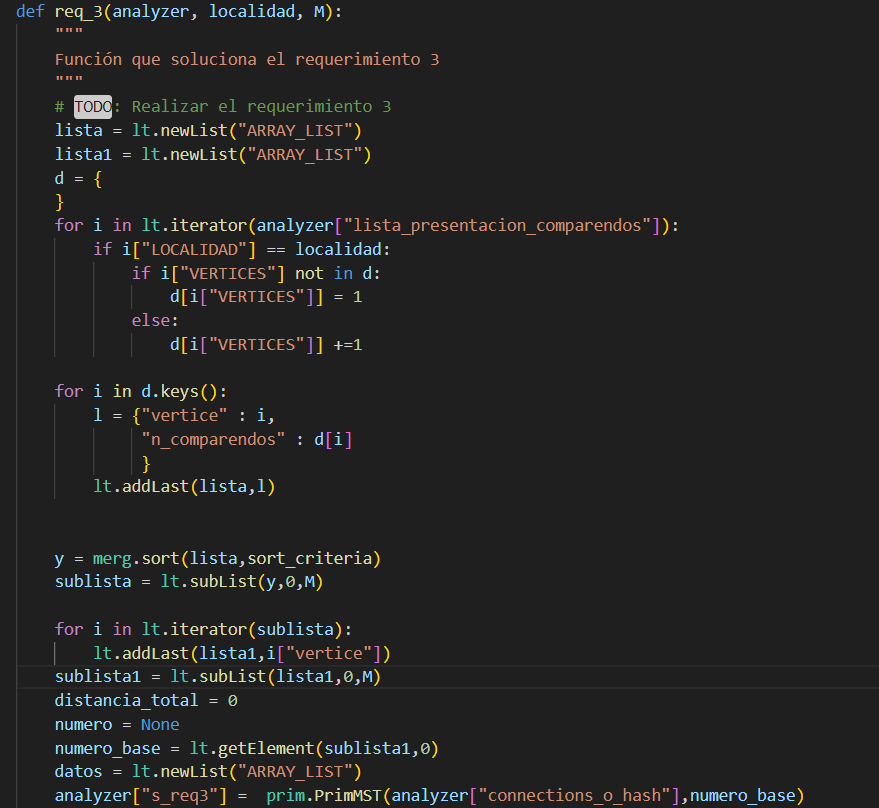


Este requerimiento es muy similar al anterior en la forma en la que los resolvimos pues se toman las coordenadas, se asocian a los vértices y se comprueba por medio de un BFS si haspathto y su respectivo pathto, sin embargo, en este caso si se usa el BFS porque lo que queremos es calcular el camino más corto.

|  |  |
| --- | --- |
| **Entrada** | Punto de origen y destino (en latitud y longitud) |
| **Salidas** | Secuencia de vértices, distancia entre vértices y total de vértices |
| **Implementado (Sí/No)** | Por Juan Lago |

# **Requerimiento <<n3>>**

**Descripción**





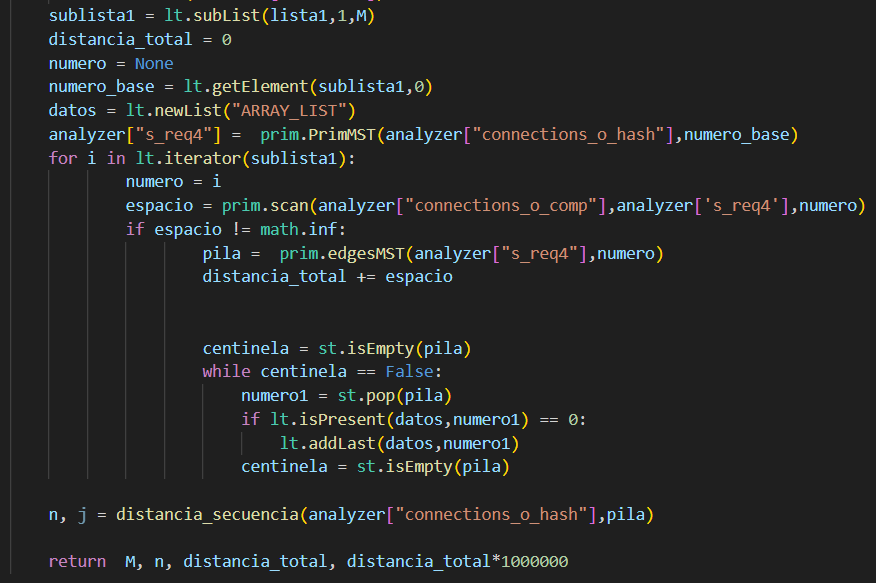
Para el tercer requerimiento abrimos dos listas que utilizaremos después y un diccionario donde a continuación al revisar cada valor de la lista de comparendos si coincide con la localidad y no está en el diccionario creado entonces se va a agregar con valor igual a uno, en caso de que ya este se le suma uno al vértice en el diccionario, esto sirve para determinar la cantidad de comparendos que hay por vértice en una localidad. También hacemos un sort para ordenarlo y se hace una sublista con la cantidad M que me pida. Luego, tomamos el numero base de la primera posición de la sublista porque vamos a hacer un Prim que nos ayudara a determinar el árbol mínimo, entonces toma la distancia desde el punto base hasta todos los puntos del vértice de la localidad y va sumando la distancia. Abajo se hace un bucle para determinar cuáles valores ya se revisaron.

|  |  |
| --- | --- |
| **Entrada** | Cantidad cámaras, localidad donde se quieren instalar |
| **Salidas** | Total, de vértices, los vértices incluidos, la cantidad de kilometro y el costo monetario |
| **Implementado (Sí/No)** | Juan David Ortiz |

# **Requerimiento <<n4>>**

**Descripción**



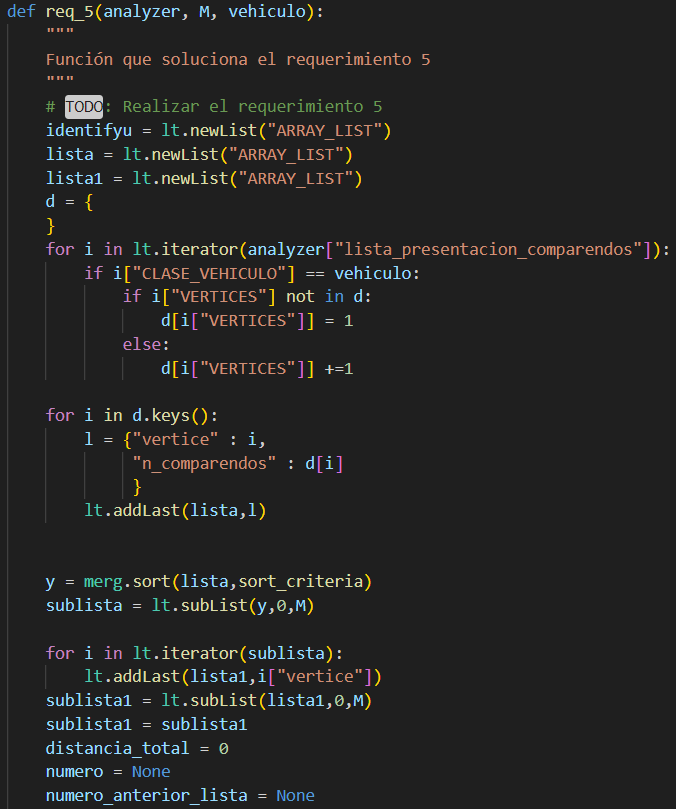


En este igualmente se crean dos listas y un diccionario en el cual, tras revisar la lista de comparendos, si los vértices no están entonces se agregan al diccionario junto con el tipo de servicio y la infracción, y si sí esta entonces se utiliza organizar mayor comp que es un sort criterio para ordenar los más graves. Después de eso se meten en una lista con los vértices y se realiza un mergesort para luego sacar una sublista de los M con más gravedad. Tras eso se trabaja como en los anteriores, se toma cuenta de la distancia y se utiliza un Prim para hallar la distancia total y el recorrido de los vértices.

|  |  |
| --- | --- |
| **Entrada** | Cantidad de cámaras que se quieren instalar |
| **Salidas** | Total de vértices, vértices incluidos, distancia en km, costo monetario |
| **Implementado (Sí/No)** | Por Mauricio Martínez |

# **Requerimiento <<n5>>**

**Descripción**



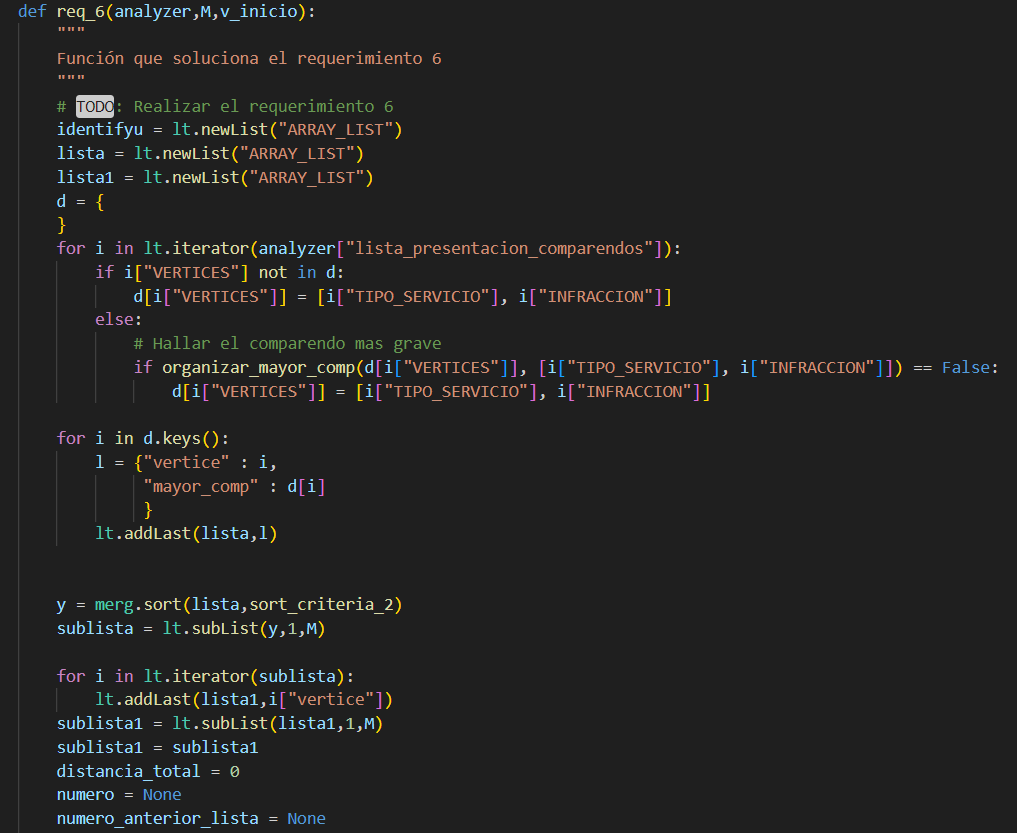


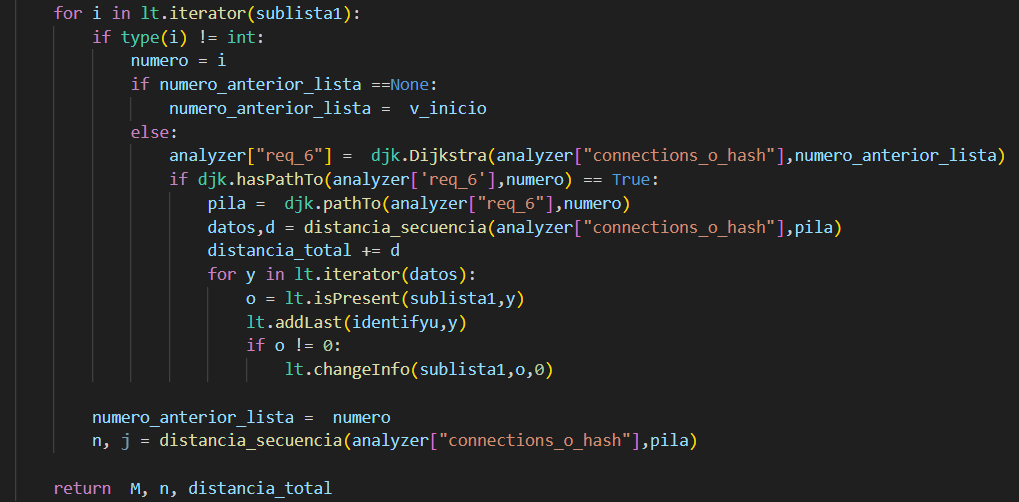
El requerimiento cinco sigue el procedimiento del 4. Se crean listas y un diccionario donde se guarda la información de la cantidad de comparendos según la clase de vehículo, luego, se guardan en una lista a la que se le hace mergesort para ordenar los datos y que para la sublista con M datos estén los vértices con mayor cantidad de comparendos. Por último, se vuelve a utilizar el Prim para encontrar la distancia total, el costo total y el orden de los vértices. Para el costo se multiplica la distancia por el costo de instalación de la fibra óptica.

|  |  |
| --- | --- |
| **Entrada** | Cantidad cámaras a instalar, clase de vehículo |
| **Salidas** | Total de vértices, vértices incluidos, distancia en km, costo monetario |
| **Implementado (Sí/No)** | Mauricio Martínez |

# **Requerimiento <<n6>>**

**Descripción**



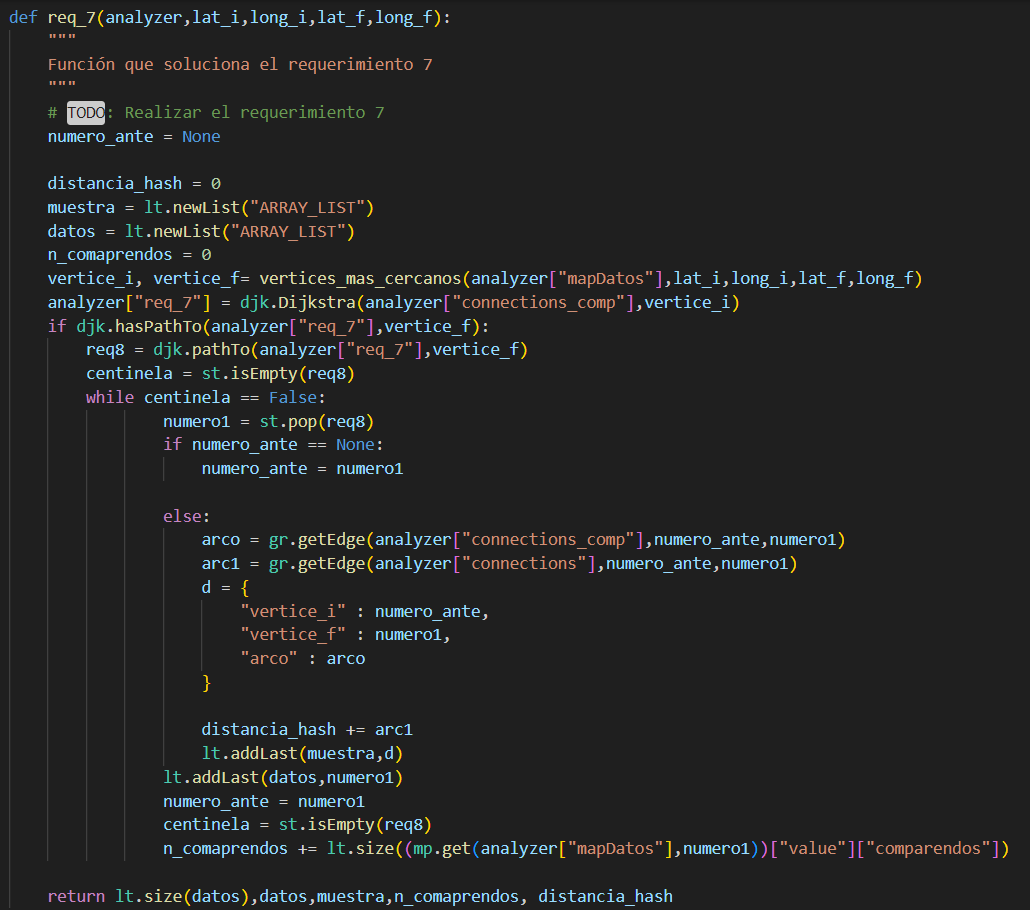


En esta ocasión se llena el diccionario con el tipo de servicio y la infracción se va revisando para sacar el comparendo más grave y se hace el mergesort y se saca la sublista del 1 al M de los más graves. Después se hace el recorrido para los vértices y se hace Dijkstra comenzando del vértice que dio el usuario que sería la estación de policía más cercana, eso es de utilidad para obtener la distancia total, distintos caminos y el recorrido de vértices.

|  |  |
| --- | --- |
| **Entrada** | Cantidad comparendos que se desea responder, estación de policía más cercana |
| **Salidas** | Total de vértices, vértices incluidos, distancia en km |
| **Implementado (Sí/No)** | Mauricio Martínez |

# **Requerimiento <<n7>>**

**Descripción**



Para el requerimiento 7 lo que hicimos fue inicializar el numero anterior, la distancia dentro del hash para saber el menor recorrido entre comparendos y también saber el hash, así mismo inicializa listas y número total de comparendos. A continuación, se determinan los vértices más cercanos con relación a las latitudes y longitudes que se ingresan como se explicó en un requerimiento anterior. Ahora se hace el Dijkstra y se inicializa dentro de la variable del analyzer y se revisa si es posible el camino de un punto a otro con el haspathto y por tanto se guarda también el pathto. Tras ello crea un bucle que saque cada parte de la pila y que vaya determinando el número anterior, luego se inicializan los arcos que sirven para determinar el peso del valor y la cantidad de comparendos y la distancia de los hashes. Y abajo se asegura que el número de comparendos va a ser igual al vértice que se saca de ese punto, y el valor de la lista del número de comparendos.

|  |  |
| --- | --- |
| **Entrada** | Punto de origen y destino (en latitud y longitud) |
| **Salidas** | Total de vértices, vértices incluidos, arcos incluidos, cantidad comparendos y distancia total en km |
| **Implementado (Sí/No)** | Juan Lago |