# Documento Proyecto 3

0. Cargar el Grafo No Dirigido (grafo más grande) de la malla vial de la ciudad completa de Washington D.C. (formato JSON), creado en el taller 8. Informar el total de vértices y el total de arcos que definen el grafo cargado. Solo es permitido leer una vez la información de los archivos.

|  |  |
| --- | --- |
| **Nombre del Requerimiento** | cargarGrafo() |
| **Funcionalidad** | Carga el grafo de la malla vial de Washington. |
| **Especificación** | |
| **Entrada** | No hay entradas |
| **Proceso** | Se toma el archivo xml, se lee con SAX Pasrser y se representa el grafo persistido en T8. |
| **Salida** | Grafo que representa la malla vial de Washington DC. |
| **Complejidad** | O(V + E) Para el almacenamiento de la estructura de datos por listas de adyacencia. |

1. Al grafo creado, se debe agregar la información de cada una de las infracciones de todos los meses del año (nuevos archivos \*.csv). Para este fin, ubique el vértice más cercano a la ubicación geográfica de la infracción y sobre este vértice almacene la información de la infracción que considere relevante. Utilice la distancia harvesiana (en kilómetros) entre dos ubicaciones geográficas para calcular la distancia entre la ubicación de un vértice del grafo y la ubicación de una infracción. Informar el total de infracciones cargadas y el número de las infracciones de cada mes.

|  |  |
| --- | --- |
| **Nombre del Requerimiento** | distanciaVertInfr() |
| **Funcionalidad** | Calcula la distancia Hervesiana en Kilómetros entre un vertice del grafo y una infracción. |
| **Especificación** | |
| **Entrada** | No hay entradas. |
| **Proceso** | Se agrega la información a los vértices para tener coordenadas del lugar de la infracción y se calcula la distancia entre el vértice y la infracción. |
| **Salida** | Distancia en kilómetros entre un vértice y una infracción, además se mostrará por consola la cantidad de infracciones cargadas y la cantidad de infracciones para cada mes del año. |
| **Complejidad** | O(V) Para el cálculo de la distancia, V es la cantidad de vértices. |

# Parte A:

2. Encontrar el camino de costo mínimo (menor cantidad de infracciones en la ruta) para un viaje entre dos ubicaciones geográficas (latitud, longitud), escogidas aleatoriamente al interior del grafo.

|  |  |
| --- | --- |
| **Nombre del Requerimiento** | costoMinimo(coordP1, coordP2) |
| **Funcionalidad** | Calcula la ruta con menos infracciones entre dos puntos del mapa. |
| **Especificación** | |
| **Entrada** | Coordenadas de los puntos. |
| **Proceso** | Se utiliza el grafo de malla vial de la ciudad y se ubican los puntos, luego se estudian todas las rutas posibles para llegar de uno a otro y se escoge la que tenga menos infracciones. |
| **Salida** | Ruta entre los puntos con menor cantidad de infracciones. |
| **Complejidad** | O(V + E) |

3. Determinar los n vértices con mayor número de infracciones en la ciudad de Washington D.C. n es un dato de entrada dado por el usuario. Adicionalmente identificar las componentes conectadas (subgrafos) que se definan únicamente entre estos n vértices. Las componentes conectadas (subgrafos) solo pueden usar arcos del grafo original que conecten estos n vértices.

|  |  |
| --- | --- |
| **Nombre del Requerimiento** | vertMasInfracciones(int n) |
| **Funcionalidad** | Obtener la cantidad de vértices (ingresada por el usuario) con mayor número de infracciones dentro del mapa. |
| **Especificación** | |
| **Entrada** | Cantidad de vértices con más infracciones que el usuario desea ver. |
| **Proceso** | Se toma el grafo de la malla vial de la ciudad, se obtiene la información de las infracciones de cada uno. Mediante un número alto se empieza a comparar el número de infracciones y se toman los n vértices que más infracciones tengan. |
| **Salida** | Vértices con más infracciones en el mapa. Componentes conexos definidos por los vértices obtenidos. |
| **Complejidad** | O(V + E) |

# Parte B:

4. Encontrar el camino más corto (menor número de vértices) para un viaje entre dos ubicaciones geográficas (latitud, longitud), escogidas aleatoriamente al interior del grafo.

|  |  |
| --- | --- |
| **Nombre del Requerimiento** | camMasCorto(coordP1, coordP2) |
| **Funcionalidad** | Obtiene el camino más corto (en cuanto a cantidad de vértices) entre dos vértices del grafo. |
| **Especificación** | |
| **Entrada** | Coordenadas de los puntos. |
| **Proceso** | Se utiliza el grafo de malla vial de la ciudad y se ubican los puntos, luego se estudian todas las rutas posibles para llegar de uno a otro y se escoge la que tenga menos vértices entre los puntos elegidos por el usuario. |
| **Salida** | Ruta de menor longitud entre los vértices dados. |
| **Complejidad** | O(V + E) |

5. A partir de las coordenadas de un área de interés de la ciudad (LonMin, LatMin) y (LonMax, LatMax) y dos valores enteros (N >= 2 y M >= 2), definir una cuadricula regular de N columnas (incluyendo LonMin y LonMax) por M filas (incluyendo LatMin y LatMax). Las intersecciones de la cuadrícula así definida contienen N x M ubicaciones geográficas separadas de forma uniforme en el área de interés (incluyendo sus límites). Aproximar estas N x M ubicaciones a los vértices más cercanos en el grafo (usar la distancia harvesiana). El resultado de la aproximación va dar un conjunto de máximo N x M vértices en el grafo.

|  |  |
| --- | --- |
| **Nombre del Requerimiento** | verticesZonaInteres(lonMin, latMin, lonMax, latMax, int n, int m) |
| **Funcionalidad** | Encuentra los nxm vértices del grafo de la malla vial de la ciudad que representan las ubicaciones geográficas separadas uniformemente en la zona de interés delimitada por las longitudes y latitudes dadas por el usuario. |
| **Especificación** | |
| **Entrada** | Longitud y latitud mínima y máxima de la zona de interés. Enteros n y m (mayores o iguales a 2) para poder ubicar los vértices esparcidos en la zona y retornarlos. |
| **Proceso** | Se ubican las coordenadas de la zona de interés en el mapa, luego se obtiene la cuadrícula de nxm ubicaciones. Estas ubicaciones se aproximan a los vértices más cercanos, y éstos se retornan. |
| **Salida** | Nxm vértices que representan las ubicaciones dentro de la zona de interés. |
| **Complejidad** | O(V + E) Para la búsqueda de los vértices. |

# Parte C:

6. Calcular un árbol de expansión mínima (MST) con criterio distancia, utilizando el algoritmo de Kruskal, aplicado a la componente conectada (subgrafo) más grande encontrada en el punto 3.

|  |  |
| --- | --- |
| **Nombre del Requerimiento** | mstKruskal(Grafo g) |
| **Funcionalidad** | Calcula un MST del grafo que entra por parámetro utilizando el algoritmo de Kruskal. |
| **Especificación** | |
| **Entrada** | Subgrafo g más grande obtenido en el punto 3. |
| **Proceso** | Se recorre el grafo obteniendo los arcos con menor distancia (sin que se creen ciclos) y se van seleccionando estos arcos. Los vértices y arcos obtenidos al hacer esto forman el árbol de expansión mínima. |
| **Salida** | Árbol de expansión mínima obtenido al realizar el algoritmo de Kruskal al grafo que entra como parámetro. |
| **Complejidad** | O(E\*log(E)) (Si se usa Merge Sort - E es la cantidad de arcos) |

7. Calcular un árbol de expansión mínima (MST) con criterio distancia, utilizando el algoritmo de Prim, aplicado a la componente conectada (subgrafo) más grande encontrada en el punto 3.

|  |  |
| --- | --- |
| **Nombre del Requerimiento** | mstPrim( Grafo g) |
| **Funcionalidad** | Obtiene el MST del subgrafo más grande obtenido en el punto 3. |
| **Especificación** | |
| **Entrada** | Subgrafo g más grande obtenido en el punto 3. |
| **Proceso** | Se crea una lista de vértices visitados y se escoge un vértice inicial V1 aleatorio. Se obtienen los arcos que conectan al vértice escogido y se toma el de menor distancia. El vértice V2 con el que conecta V1 se agrega a la lista de visitados. Ahora se miran los arcos que salen de V1 y V2 y se vuelve a realizar el procedimiento. Se hace una recursión para que se recorra todo el subgrafo y al final, los vértices y arcos resultantes forman el MST. |
| **Salida** | Árbol de expansión mínima obtenido al realizar el algoritmo de Prim al grafo que entra como parámetro. |
| **Complejidad** | O(V^2) – Si se usa búsqueda por matriz de adyacencia. (V es la cantidad de vértices)  O(V\*log(V) + E\*log(V)) – Si se usa un Binary Heap y una Lista de adyacencia. (V es la cantidad de vértices y E es la cantidad de arcos). |

8. Calcular los caminos de costo mínimo (algoritmo de Dijkstra) con criterio distancia que conecten los vértices resultado de la aproximación de las ubicaciones de la cuadricula N x M encontrados en el punto 5.

|  |  |
| --- | --- |
| **Nombre del Requerimiento** | minCostDijkstra( nxm vértices punto 5) |
| **Funcionalidad** | Obtiene el camino más corto para llegar desde un vértice a los otros usando el algoritmo de Dijkstra |
| **Especificación** | |
| **Entrada** | Vértices obtenidos en el punto 5. |
| **Proceso** | Se hace una lista de los vértices visitados y una tabla para llegar desde el vértice inicial escogido al resto de los vértices. Se marca el vértice inicial como marcado y se miran los arcos que salen de este vértice. Se toma el arco con menor distancia y se marca el vértice al que llega el arco como visitado. Se vuelven a evaluar los arcos para llegar a los demás vértices y se toma el arco con menor distancia, se actualiza la tabla de distancias y la lista de marcados. Se continúa con este proceso recursivamente hasta obtener la ruta de menor longitud que conecta todos los vértices. |
| **Salida** | Ruta de menor costo (en cuanto a distancia) que conecta todos los vértices obtenidos en el punto 5. |
| **Complejidad** | O((E + V)\*log(V)) – Si se usa un Fibonacci Heap o un MinHeap. |

9. Encontrar el camino más corto (con criterio menor número de infracciones en la vía y menor cantidad de vértices) para un viaje entre dos ubicaciones geográficas (latitud, longitud), escogidas aleatoriamente al interior del grafo.

|  |  |
| --- | --- |
| **Nombre del Requerimiento** | masCortoInfYVert(coordP1, coordP2) |
| **Funcionalidad** | Encuentra el camino con menor costo (en cuanto a cantidad de vértices e infracciones) entre dos puntos del mapa escogidos por el usuario. |
| **Especificación** | |
| **Entrada** | Coordenadas (latitud y longitud) de los puntos entre los que se quiere realizar el viaje. |
| **Proceso** | Se utiliza el algoritmo de Dijkstra para el grafo de toda la malla vial de la ciudad y se obtiene la ruta con menor longitud entre los vértices. Luego se aplica el algoritmo utilizado para el punto 2 y se obtiene la ruta con menor cantidad de infracciones entre los dos puntos. Se finaliza uniendo las dos rutas y aplicando Dijkstra nuevamente para que no se vea afectada la longitud del camino. |
| **Salida** | Ruta de menor longitud y cantidad de vértices entre dos puntos del mapa. |
| **Complejidad** | O(E + V\*log(V)) (Para realizar Dijkstra usando un Fibonacci Heap).  O(E + V) Para la búsqueda (ya sea DFS o BFS)  E es la cantidad de arcos y V la cantidad de vértices. |