Universidad de los Andes *Mayo , 2019*
Estructuras de Datos *ISIS 1206*
Daniel Yamin *Proyecto 3*
Sebastian Puerto

---

1. Requerimientos Funcionales:

Nombre	0. Carga del grafo no dirigido
Resumen	Se carga un grafo no dirigido de toda la malla vial de la ciudad de Washington D.C a partir de un archivo JSON. Tras cargar y crear el grafo correctamente, se informa el número de vértices y el número de arcos totales.
Entradas	
El archivo JSON con la información del grafo	
Resultados	
Un grafo no dirigido y el número de vértices y arcos del mismo.	
Complejidad	O(E)

Nombre	1. Carga de las infracciones de todos los meses del año
Resumen	Al grafo generado en el requerimiento 0, se le añaden las infracciones en movimiento de todos los meses del año 2018. Dichca infracción, se agrega en el vértice más cercano a la ubicación geográfica de la infracción, utilizando las distancia harvesiana.
Entradas	
Archivos CSV con la información de las infracciones.	
Resultados	
El número total de infracciones cargadas y el número de las infracciones de cada mes.	
Complejidad	O(V*I)

Nombre	2. Encontrar el camino del costo mínimo
Resumen	Se encuentra el camino de menor costo (menor cantidad de infracciones en la ruta) para un viaje entre dos ubicaciones geográficas aleatorias del grafo. En la consola se muestra el camino a seguir, el número de vértices, el costo y la distancia estimada. Luego, se muestra el camino en Google Maps.
Entradas	

Ubicación geográfica 1 : (latitud, longitud)	
Ubicación geográfica 2 : (latitud, longitud)	
Resultados	
Se muestra el camino en la consola, donde a la vez se informa para cada uno de los vértices, el ID y la ubicación geográfica. Además, se informa el costo mínimo (menor cantidad de infracciones) y la distancia estimada en kilómetros. Por último, se genera un mapa en google maps con la ruta encontrada.	
Complejidad	O(E + V log V)

Nombre	3. Determinar los N vértices con mayor número de infracciones
Resumen	Se encuentran los n vértices de la ciudad de Washington con un mayor número de infracciones, donde n es un parámetro elegido por el usuario. Posteriormente, se identifica los componentes conectados o subgrafos que se definan entre los n vértices encontrados. Por último, se muestra en consola: ● Los vértices resultantes en consola con su información y ordenados de mayor a menor ● El número de componentes conectadas Y se marcan dichos vértices en Google Maps.
Entradas	
N: número de vértices con mayor cantidad de infracciones a buscar	
Resultados	
Los principales resultados son: ● En Consola: Los vértices con su ID, ,ubicación y total de infracciones, ordenados de mayor a menor por número de infracciones. Además, se informa el número de componentes conectados. ● En Google Maps: Se muestra la localización de los vértices resultantes usando un color 1. Además, se identifica la componente conectada más grande usando un color 2.	
Complejidad	O(E + V log N)

Nombre	4. Encontrar el camino más corto entre dos ubicaciones geográficas
Resumen	Se encuentra el camino más corto: aquel con menor número de vértices para un viaje entre dos ubicaciones geográficas. Se informa al usuario el camino a seguir y la información pertinente sobre el mismo. Por último, se visualiza en el mapa dicho camino.
Entradas	

Ubicación geográfica 1 : (latitud, longitud)	
Ubicación geográfica 2 : (latitud, longitud)	
Resultados	
En consola, se muestra el camino a seguir, el total de vértices, los vértices con su ID y ubicación geográfica y la distancia estimada en kilómetros. Además, se realiza la visualización de dicho camino en un mapa de Google Maps. .	
Complejidad	O(E + E log V)

Nombre	5. Encontrar cuadrícula de la ciudad
Resumen	A partir de las coordenadas (LonMin, LatMin) y (LonMax, LatMax) y dos valores enteros N,M (mayores a 1) se define una cuadrícula regular de N columnas y M filas. Las intersecciones de la cuadrícula tienen NxM ubicaciones geográficas. Se aproxima estas ubicaciones a los vértices más cercanos en el grafo (usando distancia harvesiana). Como resultado, se tiene un conjunto de NxM vértices del grafo.
Entradas	
LonMin, LatMin: ubicación mínima de la cuadrícula	
LonMax, LatMax:: ubicación máxima de la cuadrícula	
N: número de columnas de la cuadrícula	
M: número de filas de la cuadrícula	
Resultados	
Se muestra en consola el número de vértices en el grafo encontrado.Para cada vértice, además, se muestra el identificador (ID) y la ubicación geográfica. Además, se marcan los vértices encontrados en Google Maps.	
Complejidad	O(V log V)

Nombre	6. Calcular MST (distancia) utilizado Kruskal
Resumen	Con el algoritmo de Kruskal, se encuentra el árbol de expansión mínima (MST) utilizando como criterio la distancia, aplicado a la componente conectada más grande encontrada en el punto 3. Luego, se informa al cliente el tiempo de solución del algoritmo, e información relevante sobre el árbol.
Entradas	
Subgrafo (componente conectada) encontrada en el punto 3.	
Resultados	

Se le informa al cliente: el tiempo de solución del algoritmo (en milisegundos) y los vértices, los arcos (id) y el costo total (distancia en km) del árbol. Se muestra en Google Maps el árbol generado.	
Complejidad	O(E log E)

Nombre	7. Calcular MST (distancia) utilizado Prim
Resumen	Con el algoritmo de PRIM, se encuentra el árbol de expansión mínima (MST) utilizando como criterio la distancia, aplicado a la componente conectada más grande encontrada en el punto 3. Luego, se informa al cliente el tiempo de solución del algoritmo, e información relevante sobre el árbol.
Entradas	
Subgrafo (componente conectada) encontrada en el punto 3.	
Resultados	
Se le informa al cliente: el tiempo de solución del algoritmo (en milisegundos) y los vértices, los arcos (id) y el costo total (distancia en km) del árbol. Se muestra en Google Maps el árbol generado.	
Complejidad	O(E log V)

Nombre	8. Calcular los caminos de costo mínimo (algoritmo de Dijkstra) con criterio distancia
Resumen	Utilizando el algoritmo de Dijkstra, se encuentra los caminos de costo mínimo con criterio de distancia que conectan con los vértices resultado de la aproximación de las ubicaciones de la cuadrícula NxM del punto 5. Al cliente se le informa sobre el tiempo de solución y la información pertinente sobre el camino encontrado.
Entradas	
LonMin, LatMin: ubicación mínima de la cuadrícula	
LonMax, LatMax:: ubicación máxima de la cuadrícula	
N: número de columnas de la cuadrícula	
M: número de filas de la cuadrícula	
Complejidad	O((NM)^2 + NM log NM)

Nombre	9. Camino más corto para un viaje entre dos ubicaciones geográficas
Resumen	Se encuentra el camino más corto utilizando como criterio: menor número de infracciones en la vía y menor cantidad de vértices para un viaje entre dos

	ubicaciones. Luego, se informa al cliente sobre el tiempo de solución e información pertinente del camino. Luego, se genera una visualización en Google Maps del camino encontrado.
Entradas	
Ubicación geográfica 1 : (latitud, longitud)	
Ubicación geográfica 2 : (latitud, longitud)	
Resultados	
Se muestra en consola: ● Tiempo del algoritmo en encontrar solución (milisegundos). ● Información sobre el camino encontrado: secuencia de vértices (id), costo (distancia en kilómetros) y el total de infracciones. Además, se muestra el camino encontrado en Google Maps.	
Complejidad	O(E + V log V)

2. Estimación de Complejidad y Estructuras a Utilizar:

- **Requerimiento 0:** Como se debe leer la información de cada uno de los arcos (lo cual se hace 2 veces, una vez por cada lectura los nodos del grafo) y todo vértice del Json se encuentra asociado a al menos un arco, se tiene una complejidad O(E). SI no fuera cierto que cada vértice tuviera asociado un arco, la complejidad sería más generalmente O(E + V).

- **Requerimiento 1:** Para cada infracción es necesario revisar la lista entera de vértices para conocer el más cercano, lo cual tiene una complejidad de O(V*I). Si se guardan algunos datos, esta complejidad puede restringirse a una única vez de creación del grafo; si se permite la elección de un vértice lo suficientemente cercano, así no sea el más próximo, puede reducirse significativamente la complejidad.

  Nota: I corresponde al número de infracciones.

- **Requerimiento 2:** Utilizando un grafo no dirigido con pesos, donde cada nodo almacena además el número de infracciones y el peso a usar corresponde a la suma de las infracciones de los nodos que une (si un camino de V1 hasta Vn es la sucesión de vértices V1-V2-...-Vn con cada vértice agregando un costo al camino de Ci para el vértice i, el costo del camino será CT = C1 + C2 + .. + Cn; encontrar un camino que minimice este costo es equivalente a buscar un camino que minimice el costo 2C - (C1 + Cn) pues C1 y Cn son constantes, el cual es el costo que se obtiene para el camino V1-V2-...-Vn si a cada vértice entre nodos se le asigna como peso el peso de los nodos que une). Luego, sobre el grafo, se utiliza el algoritmo de Dijkstra, donde se utiliza una

IndexMinPQ, es decir un cola de prioridad indexada y orientada a menor, por lo que el algoritmo tiene una complejidad de O(E+ V log V), aunque depende de la eficiencia de la implementación de la cola de prioridad que se logre o no esta complejidad.

- **Requerimiento 3:** Para obtener los N vértices con más infracciones se crea una cola de prioridad de tamaño a los sumo N con los vértices, lo cual tiene complejidad O(V log N). Luego, se buscan los componentes conectados, utilizando dfs (depth first search) cuya implementación requiere simplemente de 2 arreglos. La complejidad de encontrar las componentes conectadas es de O(E+V). En total se obtiene una complejidad de O(E + V log N)

- **Requerimiento 4:** Se utiliza el algoritmo de Dijkstra, de nuevo soportado en una cola de prioridad indexada y orientada a menor (IndexMinPQ). En este caso, el peso de los arcos es igual a 1, y la complejidad del algoritmo es de O(E+V log V).

- **Requerimiento 5:** Se obtiene una lista ordenada de los vértices, donde el orden está dado por la distancia y el ángulo de cada vértice a un punto de referencia: O(V log V). Luego, para cada intersección en la cuadrícula se busca el vértice más cercano que no ha sido usado, siempre y cuando no se hayan ya utilizado los V vértices, lo cual tiene una complejidad de por cada intersección O(log V) y como N*M puede ser a lo sumo igual a V, una complejidad total de O(V log V). Por lo tanto, la complejidad total de este requerimiento es de O(V log V).

- **Requerimiento 6:** Para la implementación de Kruskal, es necesario utilizar una cola (Queue) para guardar el MST. Luego, la complejidad de dicho algoritmo es de O(E log E). En este caso, E corresponde al número de arcos de la componente conectada más grande encontrada en el punto 3.
- **Requerimiento 7:** Para la implementación de algoritmo de Prim, es necesario utilizar, 3 arreglos y una cola de prioridad indexada y orientedad a menor (IndexMinPQ). La obtención del árbol de expansión mínima (MST) a través de este algoritmo tiene una complejidad de O(E log V).

- **Requerimiento 8:** El cálculo de la cuadrícula se hace como parte del requerimiento 5. Se utiliza el algoritmo de Dijkstra, por lo que se requiere de una IndexMinPQ donde se guardan los arcos del grafo generado de tamaño a los sumo NM < V, es decir que el tamaño de la cola será de a lo sumo (NM)^2 < E, por lo tanto ell tiempo proporcional en calcular los caminos de costo mínimo es de O((NM)^2 + NM log NM) siempre y cuando N y M no sean muy grandes, y O(E + V log V) en cualquier caso.

- **Requerimiento 9:** De nuevo, se utiliza el algoritmo de Dijkstra para encontrar el camino más corto con criterio menor número de infracciones en la vía y menor cantidad de vértices. Este algoritmo tiene una complejidad de O(E + V logV).