Universidad de los Andes

Estructura de datos

Proyecto 3: Primera entrega.

Integrante Uno: Santiago Cala Código: 201729218

Integrante Dos: Sofía Gutiérrez Código: 201612121

Documentación y Complejidad Requerimientos (Notación O)

Listado de requerimientos funcionales:

|  |  |
| --- | --- |
| Nombre | **R1 (1A) – Camino de costo mínimo por menor cantidad de infracciones en la ruta** |
| Resumen | Dar el camino de costo mínimo (menor cantidad de infracciones en la ruta) para un viaje entre dos ubicaciones geográficas (latitud, longitud), escogidas aleatoriamente |
| Descripción | Se escogen al azar con ayuda de la función math.random 2 nodos y utilizando el algoritmo edge-weighted DAGs propuesto en el libro como más rápido que los tradicionales Dijkstra y Bellman-Ford, siendo los pesos de los arcos la cantidad de infracciones en la ruta. |
| Entradas | Ninguna |
| Complejidad | Para este caso se tiene: O(E+V) |
| Resultados | Se muestra en consola el camino a seguir, informando sus vértices (Id, Ubicación Geográfica), el costo mínimo (menor cantidad de infracciones), y la distancia estimada (en Km). Además se visualiza el camino resultante en Google Maps. |

|  |  |
| --- | --- |
| **Nombre** | **R2 (2A) - n vértices con mayor número de infracciones en la ciudad de Washington D.C y sus componentes conectadas** |
| Resumen | Se determina los n vértices con mayor número de infracciones en la ciudad de Washington D.C y sus componente conectadas |
| Descripción | Se guardan en un Heap orientado a mayor los vértices de acuerdo al número de infracciones y posteriormente se sacan los n primeros y por medio de BFS se halla la componente conectada de cada vértice |
| Entradas | n |
| Complejidad | La complejidad del ordenamiento por Heap es O(VlogV). Mientras que la de BFS es V+E. Al escogerse n vértices, el peor aso es que sea V. Entonces se tiene una complejidad total de O(V\*(V+E)) |
| Resultados | Se muestran los n vértices resultantes en la consola, ordenados de  mayor a menor por el número de infracciones, y el número de componentes conectadas (subgrafos) definidos entre dichos vértices en el grafo original. Por cada componente se informan los identificadores de los  vértices que la componen.  En Google Maps se marca la localización de los vértices resultantes usando colores |

-----------------------------------------------------------------------------------------------------------------------------------------

|  |  |
| --- | --- |
| Nombre | **R3 (1B)** |
| Resumen | Encontrar el camino más corto (menor número de vértices) para un viaje entre dos ubicaciones geográficas (latitud, longitud), escogidas aleatoriamente al interior del grafo |
| Descripción | Para poder cumplir con este requerimiento toca usar un algoritmo de menor recorrido con base en la cantidad de nodos que se recorre. También es necesario tener en cuenta que hay que generar dos nodos aleatorios para poder encontrar el menor camino posible. Si se quiere hacer esto se generan dos números aleatorios y se busca en la tabla de Hash dos referencias de nodos. Ya con estas dos cosas es posible realizar el requerimiento, ya que los nodos cuentan con atributos que describen su latitud y su longitud, y el algoritmo de menor recorrido permite encontrar la ruta más corta. |
| Entradas | En realidad para este método no hay entradas, pues los dos nodos se escogen aleatoriamente. |
| Complejidad | Después de investigar las complejidades de los algoritmos de camino más cortos es posible ver que el más eficiente (considerando la cantidad de nodos y arcos que hay después de cargar los datos) es el de Floyd-Warshall que tiene una complejidad de O(V^3). (V siendo el número de nodos que hay). Si se le suma esta complejidad a la de generar los nodos aleatorios (que sería una simple búsqueda de O(1)) se podría concluir que la complejidad sería de O(V^3) + O(1). |
| Resultados | Al final, el resultado debería imprimir en la consola un texto con el camino a recorrer informando el total de vértices, sus vértices (Id, Ubicación Geográfica) y la distancia estimada (en Km). Además en el mapa debería mostrar el camino que se debería recorrer. |

|  |  |
| --- | --- |
| Nombre | **R4 (2B)** |
| Resumen | Este requerimiento consiste en obtener una imágen de una parte del grafo completo en una cuadrícula de un tamaño dado por parámetro. |
| Descripción | Para hacer este método, lo primero que hay que tener en cuenta es la forma en la que se calcula la distancia harvesiana. Esta se puede dar por la siguiente fórmula:    Antes de calcular las distancias, sin embargo, toca tener en cuenta los nodos que están incluidos dentro de la región que fue seleccionada por parámetro. Con base en esta información se puede proceder a hacer la cuadrícula que va a ser de dimensiones M x N. Después se tendrá que recorrer la lista de los nodos para ver:   1. Si pertenecen a la región demarcada por los parámetros que dió el usuario 2. Si efectivamente pertenecen, a cuál de los cuadros se deberían asignar.   Para hacer esto simplemente toca hacer una comparación entre la ubicación del nodo en el mapa inicial y compararla con los límites del área que se dió por parámetro. Posteriormente se calcula la distancia harvesiana con los cuadrados más cercanos para poder ver a cuál asignárselo. |
| Entradas | Las entradas de método están divididas en dos grupos, un par de vectores de coordenadas mínimas y máximas, y las dimensiones de la cuadrícula. El primer grupo estaría compuesto por cuatro parámetros:   * Longitud mínima * Longitud máxima * Latitud mínima * Latitud máxima   El segundo grupo sencillamente sería compuesto por el largo y el ancho de la cudarícula determinado por:   * M (largo) * N (ancho) |
| Complejidad | Para empezar, es importante tener en cuenta que los nodos no necesariamente están ordenados por ubicación, entonces no se pueden recorrer hasta un punto sino que toca recorrerlos todos. Esto resulta en una complejidad de O(V) (siendo V otra vez la cantidad de nodos del grafo). A esta complejidad, después toca sumarle el hecho de que toca calcular la distancia harvesiana de la ubicación de este nodo hasta los cuadrados de la cuadrícula más cercanos para saber en cuál incluirla. Para esto toca recorrer la matriz, lo cual tendría una complejidad de O(N\*M). Entonces el total de la complejidad de este requerimiento sería de O(V\*N\*M) como mínimo. |
| Resultados | Lo que debería resultar de este requerimiento sería un mensaje en la consola que incluya el número de vértices en el grafo resultado de la aproximación. También debe mostar el identificador y la ubicación geográfica de cada uno de estos vértices. Adicionalmente el requerimiento debería mostrar las ubicaciones de los vértices resultantes de la aproximación de la cuadrícula en Google Maps. |

-----------------------------------------------------------------------------------------------------------------------------------------

|  |  |
| --- | --- |
| Nombre | **R5 (1C)** |
| Resumen | Calcular un árbol de expansión mínima (MST) con criterio distancia, utilizando el algoritmo de Kruskal, aplicado a la componente conectada (subgrafo) más grande |
| Descripción | Para poder hacer este requerimiento es necesario tener ya listo el grafo con todos los nodos y arcos y aguardados. Después se aplica el arlgortimo de Kruskal (como lo pide el requerimiento) para la componente más grande (que el algoritmo lo saca solo, pues este se hace en componentes conectadas). |
| Entradas | Este requerimiento no tiene entradas por parámetro, pero si necesita del grafo ya bien compuesto para poder realizarse bien. |
| Complejidad | En el peor de ls casos la complejidad del algortimo de Kruskal es O(E\*Log(V)), y como toca desarrollar este algoritmo toca incluir esta complejidad. |
| Resultados | Al final, este requerimiento deberá mostrar en la consola de texto el tiempo que toma el algoritmo en encontrar la solución (en milisegundos), y la siguiente información del árbol generado: los vértices (identificadores), los arcos incluidos (Id vértice inicial e Id vértice final), y el costo total (distancia en Km) del árbol. Además, deberá mostrar en Google Maps el resultado del árbol. |

|  |  |
| --- | --- |
| Nombre | **R6 (2C) - MST con algoritmo de prim en Componente conectada más grande** |
| Resumen | Se calcula un árbol de expansión mínima (MST) con criterio distancia, utilizando el algoritmo de Prim, aplicado a la componente conectada (subgrafo) más grande |
| Descripción | Para este caso |
| Entradas | Ninguna |
| Complejidad | O(ElogE) ya que el peor caso es que el grafo esté todo conectado siendo la componente conectada más grande |
| Resultados | Se muestra en consola el tiempo que toma el algoritmo en encontrar la solución (en milisegundos), los vértices (identificadores), los arcos incluidos (Id vértice inicial e Id vértice final), y el costo total (distancia en Km) del árbol. En Google Maps se muestra árbol generado resultante en: sus vértices y sus arcos |

|  |  |
| --- | --- |
| Nombre | **R7 (3C)** |
| Resumen | 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 |
| Descripción | Para realizar este algoritmo toca empezar desde un nodo y después ir recorriendo todos los arcos e ir marcándolos con el costo que se toma ir hasta este nodo. A medida que se van recorriendo los nodos y se van volviendo a encontrar nodos viejos es posible ir reemplazando los valores del costo si se encontró un camino más corto. Al final cuando se quiera encontrar el camino más barato (o más corto) se hace el recorrido de regreso por los nodos ya marcados como el más barato. Esto es en resumidas cuentas lo que hace el algoritmo de Dijkstra. |
| Entradas | Las entradas para este requerimiento son en esencia los resultados de lo que salió en el punto 5. Igual es importante tener en cuenta que para el punto 5, el usuario tenía que ingresar algunos parámetros. Los parámetros son los siguientes:   El primer grupo estaría compuesto por cuatro parámetros:   * Longitud mínima * Longitud máxima * Latitud mínima * Latitud máxima   El segundo grupo sencillamente sería compuesto por el largo y el ancho de la cudarícula determinado por:   * M (largo) * N (ancho)   Al final lo que le tiene que entrar a este requerimiento es una cuadrícula MxN y una longitud mínima y máxima. |
| Complejidad | La complejidad del algoritmo es la del algoritmo de Dijkstra que es equivalente a O(V^2). |
| Resultados | Al final, el requerimiento deberá en la consola de texto el tiempo que toma el algoritmo en encontrar la solución (en milisegundos) y la siguiente información de cada camino resultante: su secuencia de vértices (identificadores) y su costo (distancia en Km). Adicionalmente deberá mostrar en Google Maps los caminos de costo mínimo junto con sus vértices y sus arcos. Adicionalmente deberá resaltar el camino más largo (en distancia) usando un color diferente. |

|  |  |
| --- | --- |
| Nombre | **R8 (4C)- Camino más corto** |
| Resumen | 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. |
| Descripción | Para este caso en primer lugar se generan las 2 ubicaciones aleatoriamente y para el caso del camino más corto, también se puede usar el algoritmo de Dijkstra pero teniendo como costo para los costos el número de infracciones y el número de vértices |
| Entradas | ninguna |
| Complejidad | La complejidad del algoritmo es la del algoritmo de Dijkstra que es equivalente a O(V^2). Que corresponde a recorrer los vertices y sus adyacentes (todos) en el peor caso |
| Resultados | Se muestra en consola el tiempo que toma el algoritmo en encontrar la solución (en milisegundos), la secuencia de vértices (identificadores), el total de infracciones y la distancia calculada (en Km) |