ANÁLISIS DEL RETO

Camilo Andrés Molina Plata, 202221119, c.molinap@uniandes.edu.co

Samuel David Rozen Mogollón, 202123592, s.rozen@uniandes.edu.co

Santiago Macías Cuel, 202221028, s.maciasc@uniandes.edu.co

Requerimiento <<1>>

Descripción

El requerimiento 1 consiste en encontrar el camino más corto entre dos vértices en un mapa.

Entrada	data_structs: Un diccionario que contiene las estructuras de datos necesarias para el requerimiento. latitud_vertice_origen: La latitud del vértice de origen. longitud_vertice_origen: La longitud del vértice de origen. latitud_vertice_destino: La latitud del vértice de destino. longitud_vertice_destino: La longitud del vértice de destino.
Salidas	total_vertices: El número de vértices en el camino encontrado. total_distancia: La distancia total del camino encontrado. pathTo: La lista de vértices que componen el camino encontrado.
Implementado (Sí/No)	Si.

Análisis de complejidad

- Extraer los vértices más cercanos al origen y al destino. O(N)
- Realiza una búsqueda en amplitud desde el vértice más cercano al origen hasta el vértice más cercano al destino. O(N)
- Obtiene la longitud del camino encontrado. O(1)
- Obtiene la distancia total del camino encontrado. O(N-1)

TOTAL O(N)

Análisis

El requerimiento 1 consiste en encontrar el camino más corto entre dos vértices en un mapa. El algoritmo utilizado es una búsqueda en amplitud, que tiene una complejidad de O(N), donde N es el número de vértices en el mapa. El requerimiento está implementado de forma correcta y cumple con los requisitos especificados. Sin embargo, la complejidad del algoritmo se puede mejorar utilizando una estructura de datos más eficiente para representar el mapa.

Requerimiento <<2>>

Descripción

El requerimiento 2 consiste en encontrar el camino más corto entre dos vértices en un mapa, donde la distancia entre dos vértices se calcula utilizando la fórmula de Haversine.

Entrada	data_structs: Un diccionario que contiene las siguientes estructuras de datos: mapa_vertices: Un diccionario que mapea un identificador de vértice a un objeto vértice. distancias: Una matriz que almacena la distancia entre dos vértices. latitud_vertice_origen: La latitud del vértice de origen. longitud_vertice_origen: La longitud del vértice de origen. latitud_vertice_destino: La latitud del vértice de destino. longitud_vertice_destino: La longitud del vértice de destino.
Salidas	total_vertices: El número de vértices en el camino encontrado. total_distancia: La distancia total del camino encontrado. pathTo: La lista de vértices que componen el camino encontrado.
Implementado (Sí/No)	Si.

Análisis de complejidad

- Extraer los vértices más cercanos al origen y al destino. O(N)
- Realiza una búsqueda en amplitud desde el vértice más cercano al origen hasta el vértice más cercano al destino. O(N)
- Obtiene la longitud del camino encontrado. O(1)
- Obtiene la distancia total del camino encontrado. O(N-1)

TOTAL O(N)

Análisis

El requerimiento 2 consiste en encontrar el camino más corto entre dos vértices en un mapa, utilizando la fórmula de Haversine para calcular la distancia entre dos vértices. El algoritmo utilizado es el mismo que el utilizado en el requerimiento 1, una búsqueda en amplitud. La complejidad del algoritmo es O(N), donde N es el número de vértices en el mapa. El requerimiento está implementado de forma correcta y cumple con los requisitos especificados. Sin embargo, la complejidad del algoritmo se puede mejorar utilizando una estructura de datos más eficiente para representar el mapa.

Requerimiento <<3>>

Descripción

El requerimiento 3 consiste en encontrar un conjunto de cámaras de seguridad que cubran una localidad, de modo que el costo total de la instalación sea mínimo.

Entrada	data_structs: Un diccionario que contiene las siguientes estructuras de datos: mapa_localidades_maxpq_comparendos: Un diccionario que mapea una localidad a una cola de prioridad de vértices ordenados por el número de comparendos. mapa_vertices: Un diccionario que mapea un identificador de vértice a un objeto vértice. n_camaras: El número de cámaras de seguridad que se deben instalar. localidad: La localidad a la que se deben instalar las cámaras de seguridad.
Salidas	total_camaras: El número total de cámaras de seguridad instaladas. id_vertices: La lista de identificadores de las cámaras de seguridad instaladas. arcos: La lista de arcos que forman el árbol de cobertura mínima. extension: La extensión del árbol de cobertura mínima. costo: El costo total de la instalación de las cámaras de seguridad.
Implementado (Sí/No)	Si, por Camilo Molina

Análisis de complejidad

- Extraer los N vértices con más comparendos de la localidad. O(N log N)
- Construir el subgrafo con los N vértices extraídos. O(N)
- Agregar las aristas entre los vértices del subgrafo. O(N^2)
- Encontrar el árbol de cobertura mínima del subgrafo. O(N log N)
- Obtener las salidas. O(N)

La complejidad total del algoritmo es O(N^2).

Análisis

El requerimiento 3 consiste en encontrar un conjunto de cámaras de seguridad que cubran una localidad, de modo que el costo total de la instalación sea mínimo.

El algoritmo utilizado es el siguiente:

- Se extraen los N vértices con más comparendos de la localidad.
- Se construye un subgrafo con los N vértices extraídos.
- Se agregan las aristas entre los vértices del subgrafo, utilizando la fórmula de Haversine para calcular la distancia entre dos vértices.
- Se encuentra el árbol de cobertura mínima del subgrafo utilizando el algoritmo de Prim.
- Se obtienen las salidas.

El requerimiento está implementado de forma correcta y cumple con los requisitos especificados.

Requerimiento <<4>>

Descripción

Encontrar un conjunto de cámaras de seguridad que cubran una localidad, de modo que el costo total de la instalación sea mínimo, teniendo en cuenta la gravedad de los delitos en cada vértice.

Entrada	data_structs: Un diccionario que contiene las siguientes estructuras de datos: mapa_localidades_maxpq_comparendos: Un diccionario que mapea una localidad a una cola de prioridad de vértices ordenados por el número de comparendos. mapa_vertices: Un diccionario que mapea un identificador de vértice a un objeto vértice. n_camaras: El número de cámaras de seguridad que se deben instalar. localidad: La localidad a la que se deben instalar las cámaras de seguridad.
Salidas	total_camaras: El número total de cámaras de seguridad instaladas. id_vertices: La lista de identificadores de las cámaras de seguridad instaladas. arcos: La lista de arcos que forman el árbol de cobertura mínima. extension: La extensión del árbol de cobertura mínima. costo: El costo total de la instalación de las cámaras de seguridad.
Implementado (Sí/No)	Si. Por Samuel Rozen.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

- Extraer los N vértices con más gravedad de la localidad. O(N log N)
- Construir el subgrafo con los N vértices extraídos. O(N)
- Agregar las aristas entre los vértices del subgrafo. O(N^2)
- Encontrar el árbol de cobertura mínima del subgrafo. O(N log N)
- Obtener las salidas. O(N)

Total

La complejidad total del algoritmo es O(N^2)

Análisis

El requerimiento 4 consiste en encontrar un conjunto de cámaras de seguridad que cubran una localidad, de modo que el costo total de la instalación sea mínimo, teniendo en cuenta la gravedad de los delitos en cada vértice.

El algoritmo utilizado es el siguiente:

Se extraen los N vértices con más gravedad de la localidad, utilizando la cola de prioridad mapa_gravedad_comparendos_maxpq.

Se construye un subgrafo con los N vértices extraídos.

Se agregan las aristas entre los vértices del subgrafo, utilizando la fórmula de Haversine para calcular la distancia entre dos vértices.

Se encuentra el árbol de cobertura mínima del subgrafo utilizando el algoritmo de Prim.

Se obtienen las salidas.

El requerimiento está implementado de forma correcta y cumple con los requisitos especificados. La complejidad temporal del algoritmo es la misma que el algoritmo del requerimiento 3.

Requerimiento <<5>>

Descripción

Encontrar un conjunto de cámaras de seguridad que cubran una localidad, de modo que el costo total de la instalación sea mínimo, teniendo en cuenta la gravedad de los delitos en cada vértice y la clase del vehículo que los comete.

Entrada	data_structs: Un diccionario que contiene las siguientes estructuras de datos: mapa_vehiculos_maxpq_comparendos: Un diccionario que mapea una clase de vehículo a una cola de prioridad de vértices ordenados por el número de comparendos. mapa_vertices: Un diccionario que mapea un identificador de vértice a un objeto vértice. n_camaras: El número de cámaras de seguridad que se deben instalar. clase_vehiculo: La clase del vehículo que se desea cubrir.
Salidas	total_camaras: El número total de cámaras de seguridad instaladas.

	id_vertices: La lista de identificadores de las cámaras de seguridad instaladas. arcos: La lista de arcos que forman el árbol de cobertura mínima. extension: La extensión del árbol de cobertura mínima. costo: El costo total de la instalación de las cámaras de seguridad.
Implementado (Sí/No)	Si. Santiago Macías Cuel

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

- Extraer los N vértices con más gravedad de la localidad. O(N log N)
- Construir el subgrafo con los N vértices extraídos. O(N)
- Agregar las aristas entre los vértices del subgrafo. O(N^2)
- Encontrar el árbol de cobertura mínima del subgrafo. O(N log N)
- Obtener las salidas. O(N)

TOTAL: O(N^2)

Análisis

El requerimiento 5 consiste en encontrar un conjunto de cámaras de seguridad que cubran una localidad, de modo que el costo total de la instalación sea mínimo, teniendo en cuenta la gravedad de los delitos en cada vértice y la clase del vehículo que los comete.

El algoritmo utilizado es el siguiente:

Se extraen los N vértices con más gravedad de la localidad, utilizando la cola de prioridad mapa_vehiculos_maxpq_comparendos.

Se construye un subgrafo con los N vértices extraídos.

Se agregan las aristas entre los vértices del subgrafo, utilizando la fórmula de Haversine para calcular la distancia entre dos vértices.

Se encuentra el árbol de cobertura mínima del subgrafo utilizando el algoritmo de Prim.

Se obtienen las salidas.

Requerimiento <<6>>

Descripción

Encontrar los N recorridos más cortos desde las estaciones de policía más cercanas a los comparendos graves, hasta los comparendos graves

Entrada	data_structs: Un diccionario que contiene las siguientes estructuras de datos: mapa_gravedad_comparendos_maxpq: Un diccionario que mapea una gravedad de comparendo a una cola de prioridad de vértices ordenados por la gravedad. mapa_estaciones: Un diccionario que mapea un nombre de estación de policía a un objeto estación. mapa_estaciones_subgrafo: Un diccionario que mapea un nombre de estación de policía a un subgrafo de la estación. num_comparendos_graves: El número de comparendos graves que se deben atender.
Salidas	recorridos: Una lista de objetos info_comparendo_estacion, cada uno de los cuales contiene la siguiente información: id_vertice_cercano_comparendo: El identificador del vértice más cercano al comparendo grave. gravedad: La gravedad del comparendo grave. estacion_mas_cercana: El nombre de la estación de policía más cercana al comparendo grave. distancia: La distancia entre la estación de policía más cercana y el comparendo grave. path: La ruta desde la estación de policía más cercana hasta el comparendo grave.
Implementado (Sí/No)	Si. Santiago Macías Cuel, Samuel Rozen y Camilo Molina

Análisis de complejidad

Paso 1: Extraer los N comparendos graves

Complejidad: O(N)

Explicación: Se debe recorrer la cola de prioridad mapa_gravedad_comparendos_maxpq, que tiene una complejidad temporal de O(N).

Paso 2: Buscar la estación de policía más cercana

Complejidad: O(1)

Explicación: Se utiliza una búsqueda en diccionario, que tiene una complejidad temporal de O(1).

Paso 3: Calcular el camino más corto

Complejidad: O(E log V)

Explicación: Se utiliza el algoritmo de Dijkstra, que tiene una complejidad temporal de O(E log V), donde E es el número de aristas en el subgrafo de la estación de policía más cercana.

Complejidad total: O(E log V), donde E es el número de aristas en el subgrafo de la estación de policía más cercana.

Análisis

El requerimiento 6 consiste en encontrar los N recorridos más cortos desde las estaciones de policía más cercanas a los comparendos graves, hasta los comparendos graves. Este requerimiento es importante porque permite a las autoridades policiales atender los comparendos graves de manera eficiente y efectiva.

Análisis de la solución

La solución propuesta para el requerimiento 6 utiliza el algoritmo de Dijkstra. El algoritmo de Dijkstra es un algoritmo de búsqueda en grafos que encuentra el camino más corto entre dos vértices.

La solución propuesta es eficiente porque utiliza el algoritmo de Dijkstra, que es un algoritmo de complejidad O(E log V).

Requerimiento 7

Descripción

El Requerimiento 7 se enfoca en generar un histograma anual de eventos sísmicos según la región y propiedades de los eventos, como magnitud, profundidad o significancia. Además, se espera una lista tabulada con detalles de los eventos.

Entrada	
Salidas	
Implementado (Sí/No)	No