Diseño y Análisis de Algoritmos

Problema 1: El viaje

Integrantes:

- Ana Karla Caballero González
- Carlos Arturo Pérez Cabrera
- Diana Laura Pérez Trujillo

1. Problema

Ana Karla y Diana quieren hacer un viaje por carretera de La Habana a Guantánamo. **Objetivo**: Fiesta. **Obstáculo**: Precio de la gasolina. Incluyendo el punto de salida (La Habana) y de destino (Guantánamo), hay un total de n puntos a los que es posible visitar, unidos por m carreteras cuyos costos de gasolina se conocen. Los compañeros comienzan entonces a planificar su viaje.

Luego de pensar por unas horas, Ana Karla va a hacia Diana y le entrega una hoja. En esta hoja se encontraban q tuplas de la forma (u, v, l) y le explica que a partir de ahora considerarían como útiles sólo a los caminos entre los puntos u y v cuyo costo de gasolina fuera menor o igual a l, para u, v, l de alguna de las q tuplas.

Diana la miró por un momento y le dijo: *Gracias*. La verdad esta información no era del todo útil para su viaje. Pero para no desperdiciar las horas de trabajo de Ana Karla, se dispuso a buscar lo que definió como carreteras útiles. Una carretera útil es aquella que pertenece a algún camino útil. Ayude a Ana Karla y Diana encontrando el número total de carreteras útiles.

PD: Cuando le contaron del plan a Carlos, este se preguntó extrañado por qué Ana y Diana no habían simplemente buscado el camino de costo mínimo entre La Habana y Guantánamo. Hay que estudiar más discreta.

2. Interpretación Matemática

Se modela el problema dado a un grafo no dirigido y ponderado G de n vértices y m aristas, un L una lista de q tuplas (u, v, l) donde $u, v \in V(G)$ y $l \geq 0$.

Se define que $e \in E(G)$ es una "arista útil" de G en L (G_L), si e está en algún camino de u a v con costo menor o igual que l para algún $(u, v, l) \in L$. Dicho esto se desea encontrar todas las aristas útiles del grafo.

3. Consideraciones para las soluciones

Se implementó una estructura Grafo con un conjunto de aristas bidireccionales (si existe la arista $< u, v >$ tambien existe $< v, u >$), con sus respectivos costos y un conjunto de vértices.

4. Soluciones

4.1. Primera Solución

Como se desea obtener el número de aristas útiles del grafo según las tuplas (u, v, l) donde u es el origen, v el destino y l el coste máximo de los caminos

útiles de un nodo a otro, se lleva a cabo un algoritmo que buscará para cada q-tupla dada todos los caminos útiles. Finalmente, obteniendo estos caminos se tendría las aristas útiles que serán aquellas que conforman dichos caminos.

El algoritmo comienza desde el nodo u del grafo y analizará lo que ocurre si avanza a cada nodo adyacente de él. Este procedimiento se continua haciéndolo para cada nodo asegurando que todos los posibles caminos válidos son encontrados, mediante combinaciones de aristas partiendo de u.
En cada paso comprueba que al tomar una nueva arista, el costo de esta, sumada al camino actual no se sobrepase de l, por lo que es evidente que este algoritmo no es infinito, pues su condición de parada es que la suma de los costos sea menor que l. Dicha condición permite que al no sobrepasar l el camino obtenido sea útil en caso de que el nodo final sea v.
Como se retornan todos los caminos útiles, solo quedaría agrupar las aristas que se encuentran en alguno de estos caminos, por lo que se itera por cada camino útil añadiendo al conjunto de las aristas útiles todas las aristas e que pertenecen a estos caminos.

4.1.1. Algoritmo y Correctitud

Se crea el Grafo G a partir del n y m y las aristas. Se dispone de un conjunto vacío $useful\_edges$ donde se irán almacenando las aristas útiles encontradas.
Se itera por cada tupla q que se encuentran en la lista $roads$ buscando todos los caminos útiles de una tupla, apoyándose en el método get_path.

get_path es el encargado de recorrer el grafo de manera recursiva para analizar todos los posibles caminos, y de estos guardará en $total\_path$ aquellos que son útiles.

Nótese que se itera por cada nodo adyacente al nodo en que se encuentra el final del camino actual, no se excluye alguno que haya sido visitado ni tampoco de donde viene el llamado recursivo anterior.

4.1.2. Complejidad Temporal

Teniendo en cuenta que el peor caso en este problema es que G sea denso, con sus aristas costo 1, y las q tuplas con valores de l grandes, en relación con la cantidad de aristas en G, se puede decir que la complejidad temporal será $O(n^n)$.
Inicialmente se iteran por las tuplas, lo que resulta en $O(q)$. Dentro de dicho ciclo se ejecuta el método recursivo get_path que se puede llamar a sí mismo, hasta $n-1$ veces. En el ciclo, además se recorrerán todos los caminos posibles, en este caso m^l y se recorren todas las aristas de los caminos que a los sumo serán l. Se obtiene así una complejidad temporal de $O(q * n^n + l * n^n)$.

4.2. Segunda Solución

En una segunda solución se propone el uso del algoritmo Dijkstra. Antes de entrar de lleno en el algoritmo implementado, se debe tener en cuenta que para determinar si cierta arista $h < u, v >$ es útil, esta debe pertenecer a algún camino útil de alguna q tupla, por lo tanto, el problema se reduce a garantizar la pertenencia de dicha arista al camino útil. Además un camino útil se puede particionar de la siguiente manera:

$d(x,u) + c(u,v) + d(v,y) \leq l \vee d(x,v) + c(v,u) + d(u,y) \leq l$

donde:

x, u, v, y : son vértices, de forma que existe una tupla (x, y, l) y donde (u, v) es una arista que pertenece al camino.

$d(x, u)$: es el costo del algun camino de costo mínimo entre x y u.

$c(u, v)$: es el costo de la arista (u, v).

Demostremos que:
$e = (u, v)$ es una arista útil en G_L sí y solo sí existe $(u, v, l) \in L$ tal que $(x, u) + c(u, v) + d(v, y) \leq l \vee d(x, v) + c(v, u) + d(u, y) \leq l$

Si existe $(u, v, l) \in L$ tal que $d(x, u) + c(u, v) + d(v, y) \leq l \vee d(x, v) + c(v, u) + d(u, y) \leq l$ entonces $e = (u, v)$ es una arista útil en G_L entonces, sin pérdida de generalidad, existe un camino $(u, ..., x)$ y un camino $(y, ..., v)$ (caminos de costo mínimo) tal que el camino $(u, ..., x) \cup (x, y) \cup (y, ..., v) = (u, ..., v)$ donde el costo de ese camino será menor o igual que l. Luego, la arista (x, y) es una arista útil.

Sea $e = (x, y)$ una arista útil en G_L entonces por definición existe una tupla (u, v, l) que pertenece a la lista tal que existe algún camino $(u, ..., v)$ tal que la arista e pertenece a ese camino y el costo del camino $(u, ..., v) \leq l$.
Luego, existen algun camino $(u, ..., x)$ y $(y, ..., v)$ (o análogo $(x, ..., v)$ y $(u, ..., y)$) tal que $(u, ..., x) \cup (x, y) \cup (y, ..., v) = (u, ..., v)$ y se cumple que el costo del camino $(u, ..., x)$ más el costo de la arista (x, y) más el costo del camino $(y, ..., v)$ es menor o igual que l.
Luego, como existe algún par de caminos $(u, ..., x)$ y $(y, ..., v)$ entonces existirán dos caminos $(u, ..., x)'$ y $(y, ..., v)'$ de costo mínimo.

Se puede decir que $d(u, ..., x) = c(u, ..., x)'$ es menor igual que cualquier camino de $(u, ..., x)$.
Análogo con el camino $(y, ..., v)$.
Por tanto, $c(u, ..., x') + c(x, ..., y) + c(y, ..., v)' \leq c(u, ..., x) + c(x, ..., y) + c(y, ..., v) \leq l$
$d(u, ..., x) + c(x, y) + d(y, ..., v) \leq l$.

Luego, queda demostrado que $e = (x, y)$ es una arista útil en G_L sí y solo sí existe $(u, v, l) \in L$ tal que $(u, ..., x) + c(x, y) + d(y, ..., v) \leq l \vee d(u, ..., y) + c(y, x) + d(x, ..., v) \leq l$.

4.2.1. Algoritmo

Como se tiene una estructura de grafo G, en el cual se almacenan todas las aristas, donde G no es dirigido, entonces hay que tener en cuenta que $d(u, ..., v) = d(v, ...u)$ para todo $u, v \in V(G)$.

La solución presentada se apoya del algoritmo de Dijkstra para poder calcular dado un vértice el camino de costo mínimo hacia cualquier otro del grafo.

Primeramente, se itera por todos los caminos de la lista L calculando el camino de costo mínimo de u a w y de v a w donde w es cualquier nodo del grafo.

Se utilizan dos versiones del algoritmo: una utilizando cola de prioridad y otro un arreglo, que se usarán en dependencia de las características del grafo.

Se recorre cada arista del grafo donde se busca por cada camino de la lista, comprobando que dicha arista cumpla una de las dos condiciones necesarias. Cuando una arista lo cumple, se incrementa el contador de aristas útiles y se sale del ciclo.

4.2.2. Complejidad Temporal

Como se mencionó anteriormente, se implementarion dos versiones diferentes del algoritmo de Dijkstra. En el caso de la versión con cola de prioridad, su complejidad es $O(|E|log(|V|))$. La otra versión es $O(|V|^2)$.

Como en el peor caso se calculan dos q vértices, que siempre sería menor o igual que n vértices, el algoritmo Dijkstra con complejidad $O(min(|E|log(|V|), |V|^2))$, se tiene en esta primera parte una complejidad temporal $O(q * D)$ donde D es la complejidad del algoritmo de Dijkstra.

Verificar por cada arista del grafo que esta se encuentre en alguna tupla de la lista tiene una complejidad de $O(q * |E|)$.

Luego, se tiene que la complejidad temporal del algoritmo presentado es $O(q * (D + |E|))$.

Notar que cuando el grafo es denso, $|E|$ es casi $|V|^2$, por lo que $D = |E|log(|V|)$ y $O(|E| + |V|^2) = O(|V|^2)$.

4.3. Tercera Solución

En esta solución se propone una mejora al algoritmo anteriormente presentado.

De lo demostrado previamente se puede deducir que una arista $e = (x, y)$ es útil para alguna tupla de la forma (u, v_i, l_i) con $u, v_i \in V(G), 1 \leq i \leq n$ si y solo si existe dicho i tal que se cumpla una de las siguientes condiciones:

$(u, ..., x) + c(x, y) + d(y, ..., v_i) \leq l_i \vee d(u, ..., y) + c(y, x) + d(x, ..., v_i) \leq l_i$

Se deduce:

$-l_i + d(y, ..., v_i) \leq d(u, ..., x) + c(x, y)$ (1)

$-l_i + d(x, ..., v_i) \leq d(u, ..., y) + c(y, x)$ (2)

Sin pérdida de generalidad, trabajando con (1), si existe algún i tal que se cumpla esta condición, entonces:

$\text{mín}_{1 \leq i \leq n}(-l_i + d(y, ..., v_i)) \leq -l_i + d(y, ..., v_i)$

El objetivo que se quiere alcanzar es calcular el valor de $\text{mín}_{1 \leq i \leq n}(-l_i + d(y, ..., v_i))$ para todo $y \in V(G)$.

Para ello se construye un nuevo grafo G' basado en el grafo G pero que contiene un nuevo nodo u_0 conectado de manera unidireccional a todos los v_i con peso $-l_i$ de las tuplas (u, v_i, l_i).

Se demuestra que un camino de costo mínimo de u_0 a y en G' tiene un costo igual a $\text{mín}_{1 \leq i \leq n}(-l_i + d(y, ..., v_i))$.

Todo camino de u_0 a y pasa necesariamente por una arista (u_0, u_i) por construcción de G' y el menor costo de u_i a y está dado por $d(u_i, ..., y)$. Se reducen la cantidad de posibles costos de camino de costo mínimo en G', teniendo solo en cuenta todos los posibles $c'(u_0, u_i) + d(u_i, ..., y) = -l_i + d(u_i, ..., y)$.

Claramente se toma siempre el menor de estos, quedando que el costo de u_0 a y será igual a $\text{mín}_{1 \leq i \leq n}(-l_i + d(y, ..., v_i))$.

Al modelar esto al algoritmo de Dijkstra que ya ha sido implementado, se obliga a empezar con una estructura del grafo G con un nuevo nodo u_0 y todos los nodos v_i con un costo de $-l_i$.

4.3.1. Algoritmo

En principio se aplica el algoritmo Floyd-Warshall para obtener el costo del camino de costo mínimo entre cualesquiera par de vértices.
Se comienza el proceso de fijar el primer vértice de las tuplas para iterar sobre ellos, donde en cada iteración se calcula con el algoritmo de Dijkstra el $\text{mín}_{1 \leq i \leq n}(-l_i + d(y, ..., v_i))$ para todo vértice x. Una vez calculados estos valores se itera por las aristas para comprobar la condición de ser una arista útil.

4.3.2. Complejidad Temporal

La complejidad temporal de aplicar Floyd-Warshall es $O(|V|^3)$. Dentro de las iteraciones de las tuplas se recorren los v_i, lo que resulta $O(|E|)$, aplicar Dijkstra es $O(|V|^2)$ y comprobar por cada arista no marcada si es útil es $O(|E|)$ lo que da una complejidad de $O(|V| * (2 * |E| + |V|^2))$.
Finalmente, la complejidad del algoritmo es $O(|V| * (2 * |E| + |V|^2) + |V|^3)$, es decir $O(|V|^3)$.

5. Casos Prueba

Para comprobar que las propuestas fueran soluciones válidas, se implementaron un generador de casos prueba y un tester.
En el Generador de casos se generan diferentes grafos de manera aleatoria, donde dada una entrada de n vértices, se crean al menos un camino de $n - 1$ vértices (pues es interés probar dicha funcionalidad). Luego, se generan aristas entre vértices de forma aleatoria con sus respectivos costos, los cuales siguen una distribución normal entre 4 y 2. Finalmente se generan las tuplas (u, v, l), de manera que l se genera en dependencia de los costos anteriormente mencionados.
El Tester consiste de una consola interactiva donde se puede escoger qué algoritmo se quiere probar y la cantidad de casos a generar. Se escoge un número aleatorio de la cantidad de vértices que tendrá cada uno de los casos. Se establecen variables que representan la media y la desviación estándar de las aristas así como de los caminos útiles que existirán en cada grafo. (estos valores pueden ser modificados). Se muestra por la consola los datos del caso generado como la cantidad de aristas y la cantidad de caminos útiles y los datos arrojados por el algoritmo seleccionado, junto a el tiempo que demoró en realizarse la ejecución del algoritmo.

6. Organización y requerimientos

En la carpeta raíz del proyecto se encuentra una distribución para cada problema. En la carpeta *Problema*1 se encuentra el archivo *problema*1.*tex*, dentro de ella, en la carpeta *code* se encuentran los tres archivos de código fundamentales: *Problema*1.*py* que contiene el código de las tres soluciones propuestas,

Testers.py que contiene la consola interactiva para seleccionar la configuración del proceso de testing y *TestGenerator.py* que contiene la lógica de la generación de grafos para los diferentes casos.

Todo fue implementado en el lenguaje de programación *Python* y con el apoyo de las siguientes bibliotecas o módulos:

- math
- heapq
- time
- random