1. El Problema

Alejandro y Semastián 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, Alejandro va entusiasmado hacia Semastián 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.

Semastián lo 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 Alejandro, se dispuso a buscar lo que definió como carreteras útiles. Una carretera útil es aquella que pertenece a algún camino útil. Ayude a Alejandro y Semastián encontrando el número total de carreteras útiles.

PD: Cuando le contaron del plan a Yisell, esta se preguntó extrañada por qué Alejandro y Semastián no habían simplemente buscado el camino de costo mínimo entre La Habana y Guantánamo. Hay que estudiar más discreta.

2. Propuesta de análisis del ejercicio

Para la solución que se quiere ofrecer, es necesario demostrar que dada $e \in E(G)$, $e_u =< x, y > \Leftrightarrow \exists t = (u, v, l), t \in U$ tal que:

$$c(p_m(u, x)) + c(< x, y >) + c(p_m(y, v)) \leq l$$

$$\vee$$

$$c(p_m(u, y)) + c(< x, y >) + c(p_m(x, v)) \leq l$$

donde $p_m(i, j)$ es cualquier camino de costo mínimo desde i a j, $\forall i, j \in G$

2.1. Demostrando $\Rightarrow$

Sea $e \in E(G), e_u$, se tiene entonces que $e =< x, y > \in p(u, v)_u$, donde $c(p(u, v)_u) < l$, $(u, v, l) \in t$, $t \in U$. Sin pérdida de generalidad, dividamos $p(u, v)_u$ tal que $p(u, v)_u = p(u, x) \cup < x, y > \cup p(y, v)$. Se sabe por tanto, que $c(p(u, x)) + c(< x, y >) + c(p(y, v)) \leq l$ por definición. Luego cómo existe $p(u, x)$ y $p(y, v)$, existe entonces $p_m(u, x)$ y $p_m(y, v)$ respectivamente. Por tanto, $c(p_m(u, x)) + c(< x, y >) + c(p_m(y, v) \leq c(p(u, x)) + c(< x, y >) + c(p(y, v)) \leq l$, por lo que $\exists p(u,v)_u = p_m(u,x) \cup <x,y> \cup p_m(y,v)$.

La demostración para $c(p_m(u,y)) + c(x,y) + c(p_m(x,v)) \leq l$ es análoga.

2.2. Demostrando $\Leftarrow$

Se tiene que $\exists t = (u,v,l), t \in U$ tal que:

$$c(p_m(u,x)) + c(<x,y>) + c(p_m(y,v)) \leq l$$

$$\vee$$

$$c(p_m(u,y)) + c(<x,y>) + c(p_m(x,v)) \leq l$$

Sin pérdida de generalidad asumamos el caso primero. Luego, se tiene que $p(u,v) = p_m(u,x) \cup <x,y> \cup p_m(y,v) = p(u,v)_u$. Luego

$$e = <x,y> \in p(u,v)_u \Rightarrow e_u$$

.

De esta forma queda demostrada la doble implicación, y por tanto la base teórica.

3. Propuesta de código

```
import math

def travel(G,U):
        dijkstra = None
        # Chose wish Dijkstra algorithm are going to be used
        if G.EdgesCount * math.log(G.VerticesCount) < G.
            VerticesCount**2:
                dijkstra = heap_dijkstra
        else:
                dijkstra = array_dijkstra

        # Initialize distance
        distance = {}



        # Apply dijkstra for any vertex in U

        for u,v,l in U:
                if distance.get(u) is None:
                        distance[u] = dijkstra(G,u)
                if distance.get(v) is None:
                        distance[v] = dijkstra(G,v)

```

```python


        # Find and count the util edges in the graph
        util_edge_count = 0
        for x, y, w in G.Edges:
                for u, v, l in U:
                        if (distance[u][x] + distance[v][y] + w <= l or distance[u][y] + distance[v][x] + w <= l):
                        util_edge_count += 1
                        break

        return util_edge_count
```

Listing 1: Ejemplo de código Python

4. Complejidad temporal del código

Las líneas hasta antes del primer *for*, tienen una complejidad temporal de $O(1)$. En ella se inicializan las variables *dijkstra* y *distance*, que representan la variante del método de dijkstra a utilizar y un diccionario en el que se guardarán las distancias de los nodos calculados. Dado que el algoritmo de Dijkstra con Heap binario tiene un costo de $O(|E|log|V|)$, y el algoritmo de Dijkstra con arrays tiene un costo de $O(|V|^2)$, se hace la distinción al principio del algoritmo para garantizar que el peor caso posible sea entonces $O(|V|^2)$. Particularizando en el problema actual, se tiene entonces que el costo de aplicar Dijkstra será $O(min(|E|log|V|, |V|^2) = O(min(m * log(n), n^2))$. Luego en el primer *for* se aplica Dijkstra $2q$ veces, donde $2q \leq n$. Por tanto, la complejidad temporal del primer *for* sería de $O(2q * min(m * log(n), n^2)) = O(q * min(m * log(n), n^2))$

Finalmente, el último for tendría una complejidad temporal de $O(m * q)$ ya que se revisan todas las aristas del grafo con todas las tuplas de U.

El costo total del algoritmo es de

$$T(n, m, q) = O(1) + O(q * min(m * log(n), n^2)) + O(m * q)$$

$$T(n, m, q) = O(1 + q * min(m * log(n), n^2) + q * m)$$

$$T(n, m, q) = O(q * (min(m * log(n), n^2) + m))$$

Para un grafo denso, $m \approx n^2$, $min(m * log(n), n^2) = n^2$, y $O(m + n^2) = O(n^2)$. Si el grafo no es denso entonces $min(m * log(n), n^2) =$ m*log(n), y $O(m + m * log(n)) = O(m * log(n))$, por lo que quedaría en un final:

$$T(n, m, q) = O(q * min(m * log(n), n^2)$$