1 Ejercicio 1

1.1 Descripción del problema

En una provincia de Optilandia las ciudades estan conectadas por rutas bidireccionales. Las rutas no se encuentran en las mejores condiciones, por lo que se decidió mejorar algunas rutas que tendrán categoría premium. Estas rutas premium muchas veces se encuentran en mejor estado y utilizan caminos más directos que suelen ser más cortos que sus alternativas. Todas las rutas tienen asociado la distancia entre las ciudades que une.

Una empresa de logística, Transportex, tiene la tarea de transportar mercadería desde una ciudad de origen hacia una ciudad de destino. Nos pidió que desarrollemos un software que encuentre el mejor camino entre el par de ciudades, es decir, el que recorra la mínima distancia. El problema que tienen es que, debido a regulaciones provinciales, las empresas de transporte solo pueden pasar por k rutas premium. Esto lo hacen porque que sino habría que mantener el estado de las rutas con demasiada frecuencia.

Escribir un algortimo que tome los datos de las rutas, la máxima cantidad de rutas premium que puede tomar un camión, el origen y destino, y calcule cual es el mínimo costo de una ruta que salga del origen y llegue al destino cumpliendo con las normativas vigentes. El algoritmo implementado debería tener una complejidad no peor que $O(n^2.k^2)$, donde n es la candidad de ciudades y k la máxima cantidad de rutas premium que puede utilizar la empresa. En caso de no existir solución devolver -1.

1.2 Solución Propuesta

En primer lugar, para almacenar toda la información de entrada, elegimos representar a la provincia como un Grafo donde las ciuadades son los nodos, y las rutas las aristas, cuyo peso asociado es las distancia que tienen asignadas las mismas, y marcando a aquellas rutas que se les haya nombrado como preimum.

Luego para resolver el problema, elegimos volcar la información almacenada en un nuevo grafo, con aristas dirigidas, y donde cada nodo va a representar una instancia más específica, almacenando ademas de un objeto, la ciudad, un estado de la misma. En consecuencia, obtenemos una mayor cantidad de nodos.

Clarificando esto útlimo, nuestro nuevo grafo va a tener almacenados $n * k$ nodos, donde cada nodo se va a formar de dos números, el primero diciendo a que ciudad refiere, nuevamente, y el segundo marcando cuantas rutas premium a lo sumo se utilizaron para llegar a dicha ciudad. Ejemplificando, ahora el nodo igual a $(1, 5)$ representa que se utilizaron 5 rutas o menos para llegar a la ciudad 1.

Al igual que cambiaron nuestro nodos, ahora también debe de cambiar el significado de las aristas, antes representaban directamente el si habia una ruta que conectaba dos ciudades, c1 y c2, y cual era la distancia de la misma. Y ahora en cambio como los nodos tambien representan estados, las rutas se les suma el valor semántico de si es posible moverse de un estado al otro.

Por lo que sean n_i y n_j dos nodos cualquiera del nuevo Digrafo, van a estar conectados por una arista dirigida desde $n1$, si y solo si pasa que:

- Las ciudades de ambos nodos están conectadas por una ruta NO PREMIUM en Optilandia y la cantidad de rutas premium utilizadas para llegar a la ciudad c_i es menor o igual a la cantidad de rutas utilizadas para llegar a la ciudad c_j.
- Las ciudades de ambos nodos están conectadas por una ruta PREMIUM en Optilandia y la cantidad de rutas premium utilizadas para llegar a la ciudad c_i es menor estricta a la cantidad de rutas utilizadas para llegar a la ciudad c_j.

Esto porque plantear que se puede llegar desde la ciudad c_i con x rutas premium utilizadas, a la ciudad c_j con y $(y < x)$ rutas premium utilizadas, atravez de alguna ruta, es absurdo ya que no hay ruta que contrareste haber utilizado alguna ruta premium en el pasado.

Luego entonces a partir de la condición que se cumple cuando tenemos dos nodos, $n1$ y $n2$, conectados por una arista, queremos ver que se cumple también que dos nodos, $n1$ y $n2$, van a estar conectados por un camino simple desde $n1$, si y solo si se puede llegar desde la ciudad $c1$ a la ciudad $c2$ utilizando a lo sumo rp rutas premium, donde $rp = CantidadRutasPremiumUtilizadas(n2) - CantidadRutasPremiumUtilizadas(n1)$

Vamos a llamar al digrafo G, n_i a un nodo generico de G, donde c_i es su respectiva ciudad, y k_i su cantidad de rutas premium utilizadas hasta el momento.

⇒)- Entonces partimos desde que existe un camino simple en nuestro digrafo que conecte a los nodos n_i y n_j, desde n_i. Y queremos ver que esto implica entonces que existe una serie de rutas en Optilandia tal que se puede llegar desde c_i a c_j tal que se utilizan k_j - k_i rutas premium.

Esto lo vamos a probar entonces de manera inductiva:

$P(i) \equiv$ Si existe un camino simple dentro de G, de longitud i, que conecte a n_i y n_j, entonces hay una serie de rutas en Optilandia tal que se puede llegar desde c_i a c_j utilizando k_j - k_i rutas premium.

C.B: $P(1)$. Si existe un camino de longitud 1 que conecte a los nodos n_i y n_j entonces esto significa que estan conectados por una arista directamente. Y por como están definidas las conexiones de las aristas en primer lugar es trivial decir que la condición se cumple.

P.I: $P(i)$. Si existe un camino C de longitud i que conecte a los nodos n_1 y n_{i+1}, tal que C= $\{(n_1, n_2), ..., (n_i, n_{i+1})\}$, entonces si llamamos a $C_2 = C - (n_i, n_{i+1})$, entonces tenemos que la longitud de C_2 es ahora de $i - 1$, por lo que por *H.I* tenemos que existe una serie de rutas en Optilandia talque se puede llegar desde c_1 a c_i utilizando $k_i - k_1$ rutas premium. Y también por caso base aplicado a $C_3 = \{(n_i, n_{i+1})\}$ tenemos que sucede lo mismo para ir desde c_i a c_{i+1}, utilizando $k_{i+1} - k_i$ rutas premium. Por lo que si concatenamos estos caminos, obtenemos un recorrido desde c_1 a c_{i+1}, dentro de Optilandia que utiliza $k_i - k_1 + k_{i+1} - k_i = k_{i+1} - k_1$ rutas premium.

$\Leftarrow$)- Aca partimos entonces de que si existe un camino dentro de Optilandia tal que se puede llegar desde la ciudad c_i hasta la ciudad c_j utilizando a los usmo k rutas, entonces, existe un camino dentro de G que une a n_i y n_j, y $k_j - k_j \leq k$

A esto constructivamente se puede llegar, ya que nos basta simplemente tomar para todas las ciudades que participan en el camino que tenemos desde c_i hasta c_j, sus nodos representativos, que se corresponden con la cantidad de rutas utilizadas. Y por definicion estas conexiónes van a estar en G.

Ya entonces probado este si y solo si, podemos decir que si existen caminos que vayan desde la ciudad origen, hasta la ciudad destino, utilizando k rutas premium a lo sumo, estos van a estar conectando los nodos $(o, 0)$ (d, k), y van a ser los únicos en hacerlo. Por lo que simplemente al correr el algoritmo de Dijkstra dentro del G, vamos a obtener el mínimo de todos estos

Como agregado queda aclarar que el alogirtmo de Dijkstra se puede correr en G ya que todos los pesos en las aristas son mayores o igual a cero.

1.3 Implementación

Ya explicamos la versión teórica de como vamos a resolver el problema, ahora nos queda mostrar nuestra implementación para justificar su complegidad teórica.

El algoritmo se va a dividir en tres partes:

1. Crear primer Grafo con los datos de entrada
2. Generar el Digrafo con información mas especifica en cada nodo
3. Ejecutar Dijkstra

Para representar el primer Grafo con los datos de entrada, elegimos utilizar una Matriz de adyacencias, donde dentro de cada casilla que marca una arista, guardamos una tupla $< int, bool > \rightarrow < PesoDeLaArista,\ EsPremium >$.

Por lo que el algoritmo de inicializacion es :

Algorithm 1 Inicialización

```
int n = input()                                  ▷ O(1): Obtengo la cantidad de ciudades
int r = input()                                  ▷ O(1): Obtengo la cantidad de rutas
int o = input()                                  ▷ O(1): Obtengo la ciudad origne
int d = input()                                  ▷ O(1): Obtengo la ciudad destino
int k = input()                                  ▷ O(1): Obtengo la cantidad de rutas Preimum permitidas usar
< int, bool > m = CrearMatriz(n, n)              ▷ O(n^2): Creo la matriz para la representación del Grafo
for i ∈ [0..r) do                                ▷ O(r): Ciclo para ingresar las rutas
    int c1 = input()                             ▷ O(1): Ciudad inicio de ruta
    int c2 = input()                             ▷ O(1): Ciudad fin de ruta
    bool p = input()                             ▷ O(1): Si la ruta es premium o no.
    int d = input()                              ▷ O(1): Distancia de la ruta

    m[c1][c2] =< d, p >                          ▷ O(1)
    m[c2][c1] =< d, p >                          ▷ O(1): Ingreso a ambas ciudades la informacion que se pueden
                                                 ▷ comunicar con la otra ciudad por la ruta i

end for
```

Complejidad: de esta seccion es $O(n^2 + r)$ ya que todas las operaciones son primitivas, excepto la creación de la matriz, que por el tamaño se deben de inicializar todas las posiciones que son n^2, y r se paga por el ciclo for dentro de la incialización que es cuando se ingresan todos los datos de las rutas

Ahora al representar nuestro segundo Grafo, nos encontramos con 2 diferencias respecto al anterior. En primer lugar, tenemos que nuestras aristas ahora son dirigidas, por lo que la arista que una a los nodos n_i y n_j, va encontrarse reproducida solamente en la casilla que tenga como primera coordenada al nodo en que nace.

En segundo lugar, no necesitamos ahora guardar dentro de cada arista si representa una ruta premium, esta información pasa al nodo, que maneja la cantidad de rutas premium que hay que asumir como utilizadas.

Ahora tenemos entonces el problema que la ventaja desde el punto de vista que a las aristas no hay que guardarlas más como tuplas, nos trae la consecuencia que hay que encontrar alguna manera de representar la nueva información que maneja un nodo como un indice de nuesta matriz que sea indisinguible.

Para ello optamos decir que el indice $i = x * (k + 1) + j$ (siendo $0 \leq x < n$ y $j = i\%(k + 1)$) representa el nodo (x, j). Donde x es la ciudad que refiere, y j la cantidad de rutas premium que el nodo asume como utilizadas hasta el momento.

Nuestro algoritmo de generación del segundo grafo entonce es la siguiente:

Algorithm 2 Creación grafo de $n * k$ nodos

```
int m2 = CrearMatriz(n * (k + 1), n * (k + 1))                          ▷ O(n^2 * k^2)
for i ∈ [0, .., n * k + 1) do                                           ▷ O(n * k)
    int nodo1 = i/(k + 1)                                               ▷ O(1): Obtengo la ciudad que refiere
    int cantP1 = i%(k + 1)                                              ▷ O(1): Obtengo sus rutas premium "utilizadas"
    for j ∈ [0, .., n * (k + 1)) do                                     ▷ O(n * k)
        int nodo2 = j/(k + 1)                                           ▷ O(1): Lo mismo que antes pero para los vecinos
        int cantP2 = j%(k + 1)
        if CumplenCondicionParaAristaDirigida(nodo1, nodo2) then        ▷ O(1)
            m2[i][j] = DistanciaDeRuta(m[nodo1][nodo2])                 ▷ O(1)
        else
            m2[i][j] = −1                                               ▷ O(1): Si no existe ruta le asigno -1, que es un valor inválido
                                                                        ▷ dentro de las distancias de las rutas
        end if
    end for
end for
```

Complejidad: $O(n^2 * k^2)$. En primer lugar se ejecuta la operación de instanciar una matriz de tamaño $n^2 * k^2$. Y luego el resto de las operaciones son de complegidad $O(1)$, que las ejecutamos dentro de una unionde ciclos donde cada uno se completa en $O(n*k)$ pasos, por lo que al haber uno dentro del otro nos da una complejidad de $O(n^2 * k^2)$

Una vez ya representado nuestro segundo Grafo, el cual ya explicamos la propiedad teórica que tenía, nos falta ejecutar el algoritmo de dijkstra buscando el mejor camino desde el nodo, (o,0), al nodo (d,k).

La idea principal de como ejecutar esto es declarar un arreglo de tuplas $< int, bool >$, llamado *distancias*, de tamaño $(n * (k + 1))$ donde cada indice representa al mismo nodo que ya explicamos para la matriz. Y vamos a ir guardando en cada posicion, la distancia parcial del nodo que refiere su indice, hacia el nodo $(o, 0)$, y si ya ocurrió la iteración del algoritmo de Dijkstra donde fue visitado dicho nodo. Que por invariante es que ya se obtuvo la distancia óptima.

Esto se va a ejecutar hasta que se haya logrado visitar a todos los nodos de los cuales se puede llegar desde $(o, 0)$, o hasta lograr iniciar la iteración del algoritmo de Dijkstra en el cual nos paramos en el nodo (d, k).

Para eso ejecutamos la siguiente implementación:

Algorithm 3 Algoritmo de Dijkstra

```
int min = −1
< int, bool > distancias = Arreglo < int, bool > [n ∗ (k + 1)]                    ▷ O(n ∗ k)
distancias[o ∗ (k + 1) + 0].first = 0      ▷ O(1) : Al nodo (o, 0) le asignamos distancia 0, ya que esta es trivial
                                                          ▷ Y es de donde vamos a iniciar el algoritmo
for int z = 0 ; z < n ∗ (k + 1); z++ do
    min = MinimoDeLosNoVisitados(distancias, n ∗ (k + 1))      ▷ O(n ∗ (k + 1)): Le asigno a min el indice
                                                          ▷ que representa alnodo que cumple que tiene la distancia
                                                  ▷ minima dentro del arreglo de distancias, y no fue visitado aún.
    if min == −1 then
        break                                          ▷ Si no se obtuvo un indice válido es que ya recorri
                                                  ▷ toda la componente conexa de la cual forma parte el nodo (o, 0)
    else if min == d ∗ (k + 1) + k then
        break                                                    ▷ Si llegue al nodo que buscaba
    end if

    for int j = 0; j < (n ∗ (k + 1)); j++ do                  ▷ Si tengo que seguir el algoritmo
                                                          ▷ actualizo los vecinos del nodo min
        if SonVecinos(min, j, m2) then                                              ▷ O(1)
            if SiMejoroElCaminoHaciaJ(min, j, distancais) then
                distancias[j].first = distancias[min] + m2[min][j]      ▷ O(1): Actualizo la distancia de J
                                                                  ▷ en el arreglo distanciaas
            end if
        end if
    end for

    if min == −1 then                                          ▷ Si min es un indice inválido
        return − 1                                                      ▷ No hay solución al problema
    else
        returnmin
    end if
end for
```

Complejidad: $O(n^2 * k^2)$. En primer lugar ejecutamos una incialización de un arreglo de tamaño $n*(k+1)$, y luego ejecutamos un ciclo For que en cada iteración va a tratar a un nodo de este arreglo, que no haya visitado anterior mente, con lo que a lo sumo se repite $n * (k + 1)$ veces. Y dentro de cada iteración ejecutamos dos operación de complejidad $O(n*(k+1))$. Por lo que en total el algoritmo muestra una complejidad de $O(n*(k+1) * 2*n*(k+1))$ $= O(n * (k + 1))$

Vamos a llamar al digrafo
- N=Cantidad de ciudades en Optilandia
- M=Cantidad de rutas en Optilandia
- CRP=Cantidad de rutas premium en Optilandia
- K=Cantidad de rutas premium disponibles a usar en Optilandia