## 0.1 Resolución

Para la resolución se cuenta inicialmente con una cantidad de vértices n, aristas m, particiones k, y una matriz *pesos* que indica el peso entre los pares de conjuntos. Además se creara un vector de vectores de enteros *solucion*, en el cual los conjuntos son representados por la posición del vector mas externo. Y los vectores dentro de cada posición a los elementos dentro de los conjuntos. Para determinar la k-partición de peso mínimo se empezara utilizando un conjunto. Se realizaran todas las combinaciones que existen para esa cantidad de conjuntos, guardándonos aquella cuya suma de pesos sea mínima *suma_solucion* y asignándole a *solucion* la combinación que me permitió obtener a la misma. Se incrementara en uno a *conjuntos_disponibles* y se procedera a buscar si existe una mejor combinación. En caso de haberla, a *suma_solucion* se le asigna la nueva suma mínima de pesos y se reemplaza la combinación de *solucion*. Esto se realizara hasta que exista alguna combinación donde la suma de todos los conjuntos sea cero (ya que es la mínima suma de pesos que puede haber). O cuando se terminaron de realizar las combinaciones para k conjuntos. Ya que entonces, habremos visto todas las combinaciones posibles para el maximo de conjuntos que disponemos. Obteniendo una de las posibles soluciones óptimas.
Para realizar todas la combinaciones es necesario contar con una estructura a la que denominaremos *k_particion*. La misma es una tupla, la segunda componente es similar a *solucion* ya que acá se iran realizando las combinaciones. Y la primer componente representa el peso de esa combinación. Comenzaremos metiendo todos los vértices a un solo conjunto (una posición del vector de *k_particion*). Esta combinación sera la solución inicial, y *suma_minima* el peso del conjunto. Como ya se mencionó, se incrementará en uno a *conjuntos_disponibles* y se ubicará al vértice 1 en este último conjunto. Luego, se procede a ubicar al siguiente vértice en el primero de los conjuntos de manera que no exceda a *suma_solucion* (empezando desde el primer conjunto). Una vez ubicado continuamos con el siguiente vértice, siguiendo la misma lógica. En el caso de que para algún vértice t no sea posible ubicarlo en ninguno de los conjuntos disponibles hasta el momento. Significa que la combinación que se tiene con los vértices 1 - t-1 no es útil para avanzar. Por lo que se procede a remover a t-1 al próximo conjunto posible (desde la posición que ocupa actualmente). Si se lo ubico, procedemos a tratar de ubicar nuevamente a t (empezando desde el primero de los conjuntos disponibles actualmente) y sino volvemos a retroceder. Pudiendo caer en dos casos: Se ubicaron a los n vértices, entonces encontré una combinación mejor que la que tenia representada en *solucion*. Actualizo a *suma_solucion* y a *solucion*. Removemos el ultimo vértice para úbicarlo en el siguiente conjunto posible. Y continuar realizando las combinaciones. Caso contrario retrocedí hasta llegar al vértice 1. Esto significa que termine de observar todas las combinaciones posibles para los conjuntos disponibles actualmente. Por lo que voy a agregar un nuevo conjunto, ubicar al vértice 1 en este ultimo y proceder a realizar las nuevas combinaciones. Hasta que el valor de *conjuntos_disponibles* supere k.

```
1: procedure UBICAR_VERTICE(vertices, aristas, particiones, pesos)
2:     tupla < float, vector < vector < int >>> k_particion
3:     vector < vector < int >> solucion
4:     ubicar todos los vertices en solucion[0]
5:     suma_solucion ← peso total de solucion[0]
6:     vector < int > conjunto
7:     k_particion.second.push_back(conjunto)
8:     vertice_actual ← 1
9:     for (conjuntos_disponibles = 1; conjuntos_disponibles < particiones; conjuntos_disponibles++) do
10:        if (suma_solucion == 0) then
11:            Return solucion
12:        end if
13:        vector < int > conjunto
14:        agregar atras(conjunto, vertice_actual)
15:        agregar atras(k_particion, conjunto)
16:        ubicar_siguientes_vertices (...., vertice_actual++, ....)
17:        Sacar(vertice_actual, k_particion.second[camiones_disponibles])
18:    end for
19:    Return solucion
20: end procedure
```

```
1: procedure UBICAR_SIGUIENTES_VERTICES(solucion, vertices, aristas, particiones, vertice_actual,
   conjuntos_disponibles, k_particion, pesos, suma_solucion)
2:     for (i = 1; i < conjuntos_disponibles; i++) do
3:         if (suma_solucion == 0.0) then
4:             return solucion
5:         end if
6:         if (puedo agregar vertice_actual a k_particion.second[i]) then
7:             agrego el vertice_actual a k_particion.second[i] y actualizo k_particion.first
8:             if (vertice_actual == vertices) then
9:                 suma_solucion ← k_particion.first
10:                solucion ← k_particion.second
11:            else
12:                ubicar_siguientes_vertices(..., vertice_actual + 1, ..)
13:            end if
14:            sacar vertice_actual de k_particion.second[i] y actualizar k_particion.first
15:        end if
16:    end for
17: end procedure
```

## 0.2 Complejidad

Para calcular la complejidad total vamos a analizar por partes nuestro algoritmo.

A) Complejidad de la función *ubicar_vertice*:
1) Crear el vector *solucion* y la tupla *k_particion*, en principio solo se los crea O(1).
2) Calcular el valor inicial de *suma_solucion*. Que va a ser el peso de tener a todas los vértices en un solo conjunto. Es decir tengo que calcular la suma de las m aristas. O(m)
3) Ubicar a todos los vértices en una única posición del conjunto *solucion*. Se realizan n *push_back* en *solucion*[0]. O(n).
4) Se crea un primer vector *conjunto* y se lo agrega atrás en la segunda componente de $k_particion$ O(1).
5) Se proceden a realizar las combinaciones, para esto se ejecuta un for que va desde 1 hasta k. En él, se van a realizar todas las combinaciones para los i-esimos camiones, 1 <= i <= $k$. Y se van a ir agregando los nuevos conjuntos. En cada iteración se crea uno, se agrega al vértice 1 en el mismo y se ubica al conjunto atrás, en el vector de *k_particion* O(1). Luego, se procede a llamar a la función *ubicar_siguientes_vertices* (mas adelante se analiza la complejidad de esta función). Y por último se retira al vértice (que se encuentra en la ultima posición del i-esimo conjunto) para comenzar una nueva iteración.
Por lo que se puede observar, exceptuando a la función *ubicar_siguientes_vertices*, en cada iteración el costo es O(1)

B) Complejidad de *ubicar_siguientes_vertices*:
Se trata de una función recursiva, primero vamos a comenzar por analizar que se hace en cada llamada y luego cuantas se hacen en total.
1) En cada llamada se trata ubicar a un vértice, entre los conjuntos disponibles actualmente. Para esto se realiza un for que va desde 1 hasta el valor de *conjuntos_disponibles*. En cada iteración, al comenzar se evalúa si el valor de *suma_solucion* es cero. En caso de serlo se interrumpe el ciclo y por ende finaliza la llamada. Ya que si lo es, no necesito seguir ubicando vértices encontré una solución óptima inmejorable. Costo O(1).
2) Si la *suma_solucion* no es cero, se procede a verificar, y de ser posible, a agregar el *vertice_actual* al i-esimo conjunto, 1 <= i <= *conjuntos_disponibles*. Para esto se va a proceder a sumarle a la primer componente de *k_particion* el peso que se adiciona el agregar el *vertice_actual* a dicho conjunto. Todos los vértices van a estar distribuidos entre los actuales *conjuntos_disponibles*. Si estoy tratando de ubicar al vértice t, 1 < t <= n, entonces voy a tener t-1 vértices distribuidos. Pero por cada iteración solo realizo tantas sumas como vértices haya en el i-esimo conjunto. Es decir que al realizar las *conjuntos_disponibles* iteraciones estoy realizando t-1 sumas. Complejidad total O(t-1).
3) Si no se pudo agregar el vértice, se procede a realizar una nueva iteración. En caso contrario agregó al

*vertice_actual* atrás, en el conjunto correspondiente. Y se va a proceder a verificar si el vértice que agregue es el ultimo es decir, el número n. En caso de no serlo se realiza una nueva llamada recursiva de la función pero ahora para ubicar al $vertice\_actual_{++}$. Finalizada la llamada se retira al *vertice_actual* del i-esimo conjunto para ser ubicado en otro y continuar con las combinaciones. Si el vértice es el último, y como estoy en el caso de que pude ubicarlo. Encontré una combinación mejor de la que tenia. Se actualiza el valor de *solucion_suma* a la de la primer componente de *k_particion*, O(1). Pero, además ahora es necesario guardarme la combinación de los elementos, para esto se copia el vector de la segunda componente a *solucion*. Este vector consta de tantas posiciones como *conjuntos_disponibles* haya hasta entonces. Con n vértices distribuidos en total. Costo de la copia O(n). Luego, se retira al ltimo vértice, para ser ubicado, si es posible, en alguno de los siguientes conjuntos.
Como se detalló a lo sumo en cada llamada los costos son O(t + n ), como t esta entre 1 <= t <= $n$ acotamos por n, obteniendo O(2n).
4) Por último calculamos cuantas veces realizamos esta tarea. Cada llamada recursiva va a ubicar a un único vértice. Como queremos ubicar a n vértices son n llamadas. Pero, cada vez que se agrega un nuevo conjunto. Se evalúa nuevamente ubicar a los n vértices. Y cada vez que se agrega se vuelven a ejecutar los for's de las funciones recursivas pero, ahora hasta una iteracion mas que la llamada anterior. Como se comienza ubicando al primer vértice en el último conjunto creado, y al siguiente desde el primero de los disponibles. Cada vez que el elemento es ubicado en un nuevo conjunto posibilita a lo sumo k *conjuntos_disponibles* para el siguiente vértice. Pero lo mismo ocurre con el vértice siguiente a este. Así hasta tener los n vértices. En conclusión tenemos las siguientes combinaciones:

$$\sum_{V_n=1}^{k} \sum_{V_{n-1}=1}^{k} ..... \sum_{V_1=1}^{k} 1 = k^n$$

*Donde V_s, 1<= s <= n, representa al vértice número s.*
Y cada una asociada a los costos mencionados en los ítem 1-3 de la sección B.
El costo total, de ambas secciones seria: O($m$) + O(2*n)*O($k^n$) = O($m$) + O($2 * n * k^n$)
acotando m por $n^2 - 1$, que es el número máximo de aristas que puede haber para n vértices, tenemos:
O($n^2$) + O($n * k^n$)