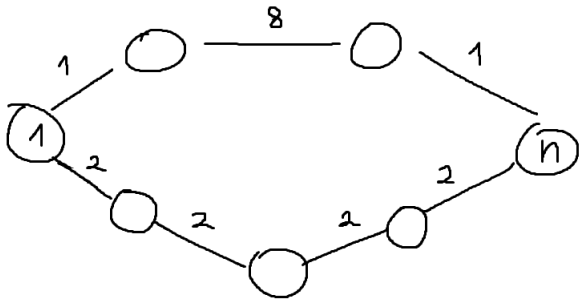
Editorial Tarea 3

Profesor: Andrés Abeliuk
Auxiliares: Daniel Báez
Julieta Coloma
Máximo Flores Valenzuela
Blaz Korecic
Diego Salas

A. El viaje

Este problema es un poco engañoso porque hay soluciones que parecieran funcionar pero no. Un ejemplo sería la solución de encontrar el camino más corto normal y luego intentar aplicar el descuento en cada una de sus aristas y quedarnos con la mejor.

Esta solución anterior falla, pues es posible que el camino más corto sin considerar el descuento y el camino más corto considerando el descuento sean completamente distintos. Por ejemplo en el siguiente grafo, el camino de abajo sin descuento cuesta 8, y con descuento 7, mientras que el de arriba sin descuento 10 y con descuento 6.



Para solucionar este problema ocuparemos una técnica común en grafos, que consiste en modificar el grafo.

La idea intuitiva es la siguiente: tendremos dos veces el mismo grafo, uno donde aún no usamos el descuento y otro donde ya lo usamos. En el grafo que aún no usamos descuento podemos movernos normalmente por una arista con peso w o elegir usar el descuento y movernos hacia el otro grafo pero con peso $\lfloor w/2 \rfloor$. En el segundo grafo solo podemos movernos normalmente (pues ya usamos el descuento). Así, la respuesta será el camino más corto entre el nodo 1 del grafo sin usar el descuento y el nodo n del grafo habiendo usado el descuento.

La idea de forma más precisa es la siguiente: duplicaremos el grafo completo. Si teníamos un grafo $G = (V, E)$, ahora tendremos $G' = (V \cup V', E \cup E' \cup D)$, donde:

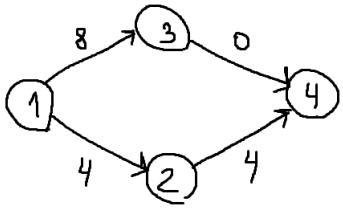
- V y E son los nodos y aristas originales.
- V' tiene "copias" de los nodos de V. Usaremos la notación $u' \in V'$ para referirnos a la copia del nodo $u \in V$.

- E' son las copias de las aristas originales pero entre los nodos de V'.
- D son aristas especiales que simbolizan usar un descuento.

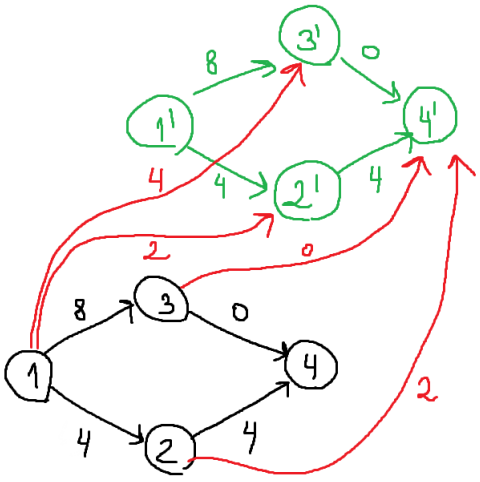
Entonces, si originalmente teníamos una arista (u, v, w) (nodos u y v, peso w) en el grafo original, tendremos las aristas:

- (u, v, w) en E,
- (u', v', w) en E',
- $(u, v', \lfloor w/2 \rfloor)$ en D.

Para ejemplificar cómo quedaría el grafo modificado, imaginemos este grafo de entrada:



Al agregarle las modificaciones quedaría así. En verde se ven los nodos y aristas copiados, mientras que en rojo las aristas con descuento. Solo podemos movernos una vez entre el grafo normal y el grafo copiado, lo que encapsula la idea de poder usar el descuento solo una vez:



Así, la respuesta será el camino más corto entre 1 y n'. En el ejemplo el camino sería $1 \to 3' \to 4'$ lo que significaría usar el descuento entre la arista que está entre los nodos 1 y 3.

Hay al menos dos formas de programar esto: una es representar los nodos como pares donde $(u, 0)$ significa que estamos en un nodo de V y $(u, 1)$ en un nodo de V'.

Otra forma es definir el número de cada nodo de V' como los nodos de V más n, de forma que u' sea representado en el código como el nodo $u + n$.

C. Jerarquía

(Obs.: Para este problema, los conocimientos q_i de cada empleado no son relevantes, los podemos leer sin hacer nada con ellos). Por la forma del problema, tendremos que ocupar el algoritmo de Kruskal para encontrar el árbol cobertor mínimo. Nota que en este algoritmo, un nodo puede relacionarse con más de uno, entonces debemos hacer una ligera modificación, pues un empleado debe tener **exactamente** un supervisor (restricción del problema).

Para ello, podemos crear un vector (llamémosle `rel`) de tamaño $n :=$ cantidad de nodos donde `rel[i]` indica si el i-ésimo nodo tiene un supervisor (`true` o `false`). Consideremos ahora una arista (u, v, w). En la ejecución de `kruskal`, aparte de revisar si el representante de u es distinto al de v, **debemos** revisar que v no tenga supervisores asignados, es decir, que `rel[v]` sea `false`. Si estas dos condiciones ocurren, actualizamos el vector `rel` según corresponda y dejamos que el algoritmo haga lo suyo.

Nota que la respuesta es -1 si la cantidad de `rel[i]` que sean `false` es distinta de 1, pues sabemos que sólo el nodo raíz no estará relacionado con ningún supervisor. En cualquier otro caso, la respuesta es la que nos proporciona el algoritmo de Kruskal.

Complejidad: $\mathcal{O}(m \log m)$ por el uso del algoritmo de Kruskal.

D. Camino más corto

Para este problema queremos encontrar todas las aristas que pueden ser parte del camino más corto, eliminarlas y encontrar el menor camino sobre ese grafo con las aristas eliminadas.

Primero calculamos la menor distancia con la que se puede llegar con Dijkstra, digamos que es x. Ahora, digamos que queremos evaluar si una arista que va de u a v de tamaño w está en algún camino más corto o no. Esto puede resultar complejo, pero la estrategia es la siguiente:

- Cuando leamos los datos, guardamos el grafo original G y otro con las aristas invertidas G'. Es decir, que $(u, v, w) \in G \implies (v, u, w) \in G'$. Nota que el orden de los nodos u y v en el par ordenado son relevantes, aquí estamos diciendo que conectamos $u \to v$ en G y $v \to u$ en G', ambos con peso w.
- Consideremos ahora una arista cualquiera $(u', v', w') \in G$. Para ver si esta arista está en algún camino más corto, podemos calcular la menor distancia entre s (nodo origen) y u' con el Dijkstra ejecutado sobre el grafo G, lo mismo con v' y d (nodo destino) pero con el Dijkstra sobre G'. Si se cumple que:

$$\text{dist}(s, u') + w' + \text{dist}(v', d) = x$$

Entonces esta arista está en un camino más corto y hay que eliminarla. Nota que esto es válido, pues a pesar de que el grafo sobre G' calcule el óptimo de d a s, nunca cambiamos los pesos de las aristas, entonces la respuesta para $\text{dist}(v', d)$ será correcta.

Ahora, con el grafo nuevo G'' con las aristas de los caminos más cortos eliminadas, podemos usar nuevamente el algoritmo de Dijkstra para encontrar la respuesta.

El paso más importante de esta solución es solamente hacer los dos Dijkstras desde s y desde d una vez cada uno y solamente revisar estos valores porque si los estamos constantemente calculando para cada arista es imposible que pase por tiempo.

Complejidad: $\mathcal{O}((N + M) \log M) \sim \mathcal{O}(M \log M)$ (pues $N \ll M$) por la complejidad del algoritmo de Dijkstra.