Problemitas de camino mínimo

Franco Assenza y Santiago Cifuentes

Fecha de compilación: 5 de octubre de 2021

1. DAG de caminos mínimos

Si al correr Dijkstra partiendo desde u vamos guardando todas las aristas (ahora dirigidas) que son parte de algún camino mínimo (partiendo desde u), nos quedamos con lo que de ahora en adelante llamo el DAG de caminos mínimos. DAG por Directed Acyclic Graph, o grafo dirigido acíclico.

Una forma de no armar explícitamente el DAG de caminos mínimos es notar qué aristas están en el: para que una arista (u, v) esté en el DAG de caminos mínimos, el camino mínimo a v tiene que ser igual al camino mínimo a u + el peso de la arista (u, v).

2. Investigation

Problema original: `https://cses.fi/problemset/task/1202`

Enunciado 1. *Tenemos un grafo con pesos positivos en las aristas, y dados 2 vértices queremos saber*

- *Camino mínimo.*
- *Cantidad de caminos mínimos distintos.*
- *Camino mínimo con mínima cantidad de aristas.*
- *Camino mínimo con máxima cantidad de aristas.*

2.1. Camino mínimo

Camino mínimo es una aplicación directa de Dijkstra.

2.2. Cantidad de caminos mínimos

Hay al menos 2 formas de resolver de caminos mínimos distintos

- Podemos modificar Dijkstra: en vez de guardar para cada nodo solo el camino mínimo, además guardamos la cantidad de caminos mínimos.

  Cuando Dijkstra analiza una arista (u, v) es porque tiene un camino mínimo a u. En ese caso hay 2 situaciones

- Mejoramos la distancia actual a v. En ese caso, marcamos la mejor distancia, y que hay tantos caminos mínimos a v como caminos mínimos a u.
- No mejoramos la distancia actual a v. En ese caso agregamos a la cantidad de caminos mínimos a v la cantidad de caminos mínimos a u.

Para ver que es correcto, podemos ver que la cantidad de caminos mínimos a un cierto nodo v es la suma de la cantidad de caminos mínimos a sus vecinos por los cuales llegamos de manera mínima a v. Para más formalización se puede hacer via inducción.

Nos queda la complejidad de Dijkstra.

- Otra forma es usar Dijkstra como caja negra, y obtener el DAG de caminos mínimos. Ahí podemos decir que la cantidad de caminos mínimos a un nodo v es la suma de las cantidades de caminos mínimos a los nodos que tienen una arista hacia v en el DAG de caminos mínimos. Para que no quede exponencial usamos programación dinámica.

  Sigue ganando la complejidad de Dijkstra.

2.3. Camino mínimo con mínima cantidad de aristas

Una opción es hacer Dijkstra en el grafo donde reemplazamos las aristas $u \to v$ con peso w por aristas con peso $(w, 1)$, sumando tuplas de forma $(a, b) + (c, d) = (a + c, b + d)$, y donde $(a, b) > (c, d)$ si $a > c$ o $a = c$ y $b > d$.

El peso de un camino va a ser $(\text{peso en el grafo original}, \text{cantidad de aristas en el camino})$. Por ende Dijkstra nos va a dar el camino mínimo con menor cantidad de aristas, porque es más liviano que otros en el grafo modificado.

Otra opción es hacer BFS en el DAG de caminos mínimos, pensando las aristas del DAG como de peso 1, para buscar el camino mínimo en dicho grafo.

Una tercera opción es recursión + dinámica: el camino mínimo de menor cantidad de aristas a un vértice v es el camino de menos cantidad de aristas a uno de sus padres en el DAG de caminos mínimos, + 1.

2.4. Camino mínimo con máxima cantidad de aristas

Las opciones son análogas al caso anterior: usando $(w, -1)$ en las aristas (notar que si $w > 0$ y $p < 0$, $(w, p) > (0, 0)$, con lo que no hay aristas negativas), o bien haciendo recursión + dinámica, donde el camino mínimo de mayor cantidad de aristas a un vértice v es el máximo de lo que nos dan los padres en el DAG de caminos mínimos.

3. Flight Discount

Problema original: `https://cses.fi/problemset/task/1195`

Enunciado 2. *Tenemos un grafo con pesos positivos en las aristas, podemos dividir el peso de una arista por la mitad, y queremos saber el camino mínimo entre 2 vértices u y v.*

Armamos un grafo con el doble de vértices del grafo original, donde para cada vértice u tenemos uno u', y para cada arista (u, v) de peso p, tenemos la arista (u, v) de peso p, la arista (u', v') de peso p' y la arista dirigida (u, v') de peso $\frac{p}{2}$.

Intuitivamente, cuando tomamos la arista (u, v') estamos diviendo el peso de la arista (u, v) sobre 2, y seguimos en la copia del grafo, donde no tenemos más permitida esa operación.

Luego, notando que como los pesos son positivos siempre vamos a querer usar la posibilidad de dividir una arista por 2, el camino mínimo de u a v' es la respuesta a nuestro problema.

Otra forma de resolver este problema es, para cada posible arista que querramos dividir por 2, ver el valor si dividimos por 2 esa arista y pasamos por esa arista: la respuesta para una arista (a, b) va a ser el camino mínimo de u a a, el peso de la arista dividido 2, y el camino mínimo de b a v.

Los caminos mínimos de u a todos los vértices nos los da Dijkstra. Para tener los caminos mínimos desde cada vértice hacia v, podemos correr un Dijkstra desde v con las aristas invertidas de dirección, si la tuvieran (si es no dirigido no hace falta).

4. Consistency

Problema original: `https://www.facebook.com/codingcompetitions/hacker-cup/2021/qualification-round/problems/A2`

Enunciado 3. *Tenemos un string s.*

Hay k distintos tipos de reemplazos: podemos reemplazar algunas ciertas letras por ciertas otras. Por ejemplo, puede ser que podamos reemplazar una a por una c, y no necesariamente una c por una a.

Queremos hacer que el string tenga todas las letras iguales, aplicando la menor cantidad necesaria de reemplazos. O reportar que no es posible, si no lo es.

Por un lado, vamos a querer saber cuánto nos cuesta reemplazar cierta letra por cierta otra letra. Esto nos lo da el grafo de reemplazos que viene en el enunciado, el que aplicamos Floyd-Warshall para tener el costo de reemplazar cierta letra por cierta otra letra.

Luego, probamos todas las letras posibles, y el costo de llevar s a cierta letra l va a ser la suma para cada letra de s de la distancia a dicha letra. Tomamos el mejor.

5. Capitalism

Problema original: `https://codeforces.com/problemset/problem/1450/E`

Enunciado 4. *Tenemos un grafo con pesos en los nodos, que desconocemos. Sabemos que 2 vértices que tienen una arista entre sí tienen diferencia 1 entre sus valores. No necesariamente tienen diferencia distinta de 1 si no tienen arista. De algunos pares de nodos con arista nos dicen cuál tiene un valor mayor. De otros, solo que tienen diferencia 1.*

Queremos saber si existe un grafo con estas características y, si lo hay, dar una asignación maximizando la diferencia entre el valor máximo y el mínimo en los nodos.

Las aristas unen vértices con valores de distintas paridades. Es decir que si el grafo no es bipartito ya podemos descartar que haya solución.

En el grafo anotamos las siguientes aristas: si sabemos que $a_u + 1 = a_v$, ponemos $u \to v$ de peso 1 y $v \to u$ de peso -1, representando $a_u - a_v \leq -1$ y $a_v - a_u \leq 1$. Si sabemos que $|a_u - a_v| = 1$, ponemos $u \to v$ y $v \to u$ de peso 1, representando que ambas diferencias son a lo sumo 1.

Si tenemos un ciclo de peso negativo, llegamos a una contradicción con las desigualdades: $0 > (a_1 - a_2) + (a_2 - a_3) + \ldots + (a_n - a_1) = 0$. En este caso tampoco podemos dar solución.

Las desigualdades nos marcan que $a_v - a_u \leq dist(u \to v)$. Entonces la respuesta no puede ser mayor al diámetro del grafo $maxdist(u \to v)$.

Luego podemos construir una respuesta óptima: tomemos u y v que realicen la distancia máxima (por ejemplo con Floyd-Warshall). Asignemos $a_i = dist(u \to i)$. Las desigualdades se cumplen para a_i. Si tenemos una arista dirigida (u, v) va a cumplir $a_v - a_u = 1$, como queríamos. En el caso de las no dirigidas tenemos que $|a_u - a_v| \leq 1$. Como es bipartito no van a ser iguales, y se va a cumplir $|a_u - a_v| = 1$.

6. Crocodiles

Problema original: `https://ioinformatics.org/files/ioi2011problem4.pdf`

Enunciado 5. *Tenemos un grafo con pesos en las aristas. Estamos en un nodo, y existen ciertos nodos que son salidas del grafo, y queremos llegar a uno de ellos.*

Cada vez que estamos en cierto nodo del grafo, hay exactamente una arista del grafo que está bloqueada.

Queremos un plan simple que nos permita llegar a una salida, minimizando el tiempo en el que estamos garantizados de haber llegado a una salida.

Plan simple significa que tenemos instrucciones fijas para cada nodo: dado un nodo u, vamos a ir hacia el nodo v y, si esa arista está bloqueada, hacia el nodo w.

La solución es una aplicación casi directa de Dijkstra. La distancia inicial a las salidas es 0.

Para cada nodo vecino de la frontera actual de Dijkstra, en vez de guardar solo el camino mínimo, guardamos los 2 caminos más chicos.

Para expandir la frontera, elegimos el nodo que tenga su segundo camino más chico entre todos. Y esta va a ser la distancia obtenida a dicho nodo, dado que asumimos que nos bloquean el mejor camino.

7. Are we lost yet?

Problema original: `https://codingcompetitions.withgoogle.com/codejam/round/000000000043319e/0000000000432b87`

Enunciado 6. *Tenemos un grafo con pesos en las aristas. Pero no sabemos el peso de las aristas, sino que para cada arista* (u, v) *tenemos un intervalo* $[a, b]$ *tal que sabemos que el peso cae en dicho intervalo.*

Nos dan un camino $c_1, c_2, \ldots, c_n$*. Queremos saber si es un camino mínimo y, en caso que no lo sea, la primera arista* (c_i, c_j) *que no es parte de ningún camino mínimo de* c_1 *a* c_n*.*

Fijemos un prefijo del camino con $p \leq n$, $c_1, \ldots, c_p$. Podemos intentar ver para cada prefijo si es o no prefijo de un camino mínimo. Para encontrar la primera arista que falla podemos probar todos o hacer una búsqueda binaria.

Al prefijo elegido le vamos a asignar el menor costo posible, es decir, para cada arista, el extremo inferior de su intervalo. Poner valores más altos puede mantener la diferencia entre los caminos mínimos que empiezan con nuestro prefijo o empeorarla.

Luego queremos llegar lo más rápido posible desde c_p, y lo más lento posible desde c_1 (que va a ser como mucho tan lento como $c_p + dist(c_1, c_p)$).

Y esto es una aplicación casi directa de Dijkstra: corremos Dijkstra empezando con c_1 y c_p de frontera, con distancias 0 y la suma de los bordes inferiores de los intervalos de (c_1, c_2), (c_2, c_3), ..., como sugerido previamente.

Si en Dijkstra tomamos una arista de las del camino, tomamos el borde inferior.

Si llegamos por una expansión de frontera que parte desde c_p (implementativamente podemos ir marcando los nodos, priorizando llegar desde c_p, que quiere empatarle al otro camino), tomamos el borde inferior del intervalo.

Si llegamos por una expansión de frontera que partió desde c_1, significa que hay un camino más rápido a ese nodo que usando el camino propuesto, así que tomamos el borde superior del intervalo.

Con este Dijkstra, si llegamos a c_n expandiendo c_p entonces nuestro prefijo es prefijo de un camino mínimo a c_n. De lo contrario, sin importar la elección de las aristas, no tenemos un prefijo de un camino mínimo.