Informe
Matemática Discreta II
Minimum spanning tree for each edge
Problema:609E

Abel Molina Sánchez
Grupo 2-11
Ciencias de la Computación
Universidad de La Habana

18 de noviembre de 2020

1. Problema

E. Minimum spanning tree for each edge

Connected undirected weighted graph without self-loops and multiple edges is given. Graph contains n vertices and m edges.
For each edge (u, v) find the minimal possible weight of the spanning tree that contains the edge (u, v).
The weight of the spanning tree is the sum of weights of all edges included in spanning tree.

Input: First line contains two integers n and m ($1 \leq n \leq 2\Delta 10^5$, $n - 1 \leq m \leq 2\Delta 10^5$) — the number of vertices and edges in graph. Each of the next m lines contains three integers $u_i, v_i, w_i (1 \leq u_i, v_i \leq n, u_i \neq v_i, 1 \leq w_i \leq 10^9)$ — the endpoints of the i-th edge and its weight.

Output: Print m lines. i-th line should contain the minimal possible weight of the spanning tree that contains i-th edge.
The edges are numbered from 1 to m in order of their appearing in input.

Interpretación del problema:

Se recibe como entrada del problema un grafo $G = (V, E)$ simple, no dirigido, conexo y ponderado y se quiere buscar el costo del árbol abarcador de costo mínimo(minimum spanning tree(mst)) que contiene entre sus aristas a la arista i $\forall i, i = 1, 2, ...m = |E|$.

Primera Idea: Conociendo el algoritmo de Kruskal que dado un grafo devuelve su árbol abarcador de costo mínimo, basándose en la ordenación de las aristas por su peso, no ordenar las aristas de antemano y en cada ejecución del algoritmo a la arista i asignarle un peso mínimo inferior a todas, en este caso 0, de tal forma que una vez ordenadas las aristas durante la ejecución del algoritmo de Kruskal ésta sea la de peso mínimo, por lo cual formará parte del mst, luego una vez obtenido el costo del mst le sumaría el costo de la arista para tener su solución. Este proceso se repetiría por cada arista, con lo cual se estarían ordenando las aristas $|E|$, lo cual es bastante ineficiente ya que la complejidad temporal de la solución sería $O(|E|^2 log|V|)$, pero se realizarían $|E|$ operaciones de $O(|E|log|E|)$.

Idea de solución: Utilizando el algoritmo de Kruskal, determinar T, un mst del grafo original. Como no existirá otro mst con menor costo porque sería una contradicción con el hecho de que es mínimo, para todas las aristas que formen parte de T, su solución sería el costo de T. Luego por cada arista $e \in E[G]/E[T]$, la idea es agregarla al mst; al agregarla se formaría un ciclo, ya que T es un árbol, por lo cual es necesario retirar una arista para mantener la condición de árbol. Entonces, se busca y se elimina la mayor arista distinta de e que forma parte de este ciclo, o lo que es lo mismo, al buscar la solución para la arista $e = (u, v)$, busco la mayor arista que forma parte del camino p que une a u y a v en T y la retiro, luego agrego a (u, v).
Sobre esta idea, al ser T un árbol se puede notar que el lca(ancentro común más cercano), entre u y v,digamosle l, formará parte de p. Luego, la arista de mayor peso en el camino p, será también la mayor arista entre los caminos p_{ul} y p_{vl} que unen respectivamente a u y v con l. Luego una vez

hallado T, la solución pasa por cada arista saber el costo de la arista de mayor costo entre los caminos desde sus extremos hasta el lca de sus extremos, restarlo al peso de T y sumar el peso de la arista a la que se le busca solución.
Ahora veamos como buscar el recorrido desde u y v(los extremos de una arista) hasta su lca.
Esto se puede realizar de forma lineal por la altura del árbol, la altura del árbol seá a lo sumo $|V| - 1$, por lo que podemos decir que hallar el lca de dos vértices en el árbol lo podemos realizar en $O(|V|)$. Si realizamos esta operación por cada aristas que no forma parte de T, tendremos que encontrar la solución será $O(|E|log|V|) + O(|E||V|)$ que es $O(|E||V|)$ que ya mejora la idea anterior.
Ahora, existen varios métodos y técnicas para hallar el lca que tienen una cota no lineal. Para la solución del problema voy a utiizar la técnica de Binary Lifting, con la cual en $O(|V|log|V|)$ precomputo los ancestros binarios de todos los vértices del árbol y el peso máximo en el camino hasta cada uno. Luego la búsqueda de la solución para cada arista se podrá realizar en $O(log|V|)$, utilizando los valores precalculados. Hallar la solución para todas las aristas quedará en $O(|E|log|V|)$ que es también el tiempo de ejecución para hallar el mst, por lo cual el algoritmo de solución quedaría en $O(|E|log|V|)$, mejorando las ideas anteriores.
Luego de explicar la idea de la solución paso a demostrar todo lo planteado.

2. Demostraciones

Lema 1: *Al quitar una arista del camino p de u a v en el árbol, éste se divide en dos componentes conexas de tal forma que v esté en una y u en la otra.*

Demostración: Supongo que u y v siguen en la misma componente conexa, entonces existirá un camino $q \neq p$ que une a u y a en T, pero ésta es una contradicción con el hecho de que todo par de vértices esté unido por un único camino es un árbol.

Lema 3: *Si una arista (u, v) no es parte de un árbol abarcador (spanning tree), se puede cambiar la arista (u, v) por una de las aristas que forman parte del camino de u a v en el árbol y el resultado seguirá siendo un árbol abarcador.*

Demostración: Si removemos una arista intermedia (x, y) en el camino (u, v) en el árbol, se crearán dos subárboles, donde u y v formarán parte de un subárbol distinto (Lema 1). Luego, por lo mismo existe un único camino desde cualquier vértice del subárbol de v hasta v (y viceversa) y desde el subárbol de u hasta u (y viceversa). Luego conecto la arista (u, v) y entonces existe un único camino desde cualquier vértice del subárbol de u hasta cualquier vértice del subárbol de v, con lo cual es conexo y se mantiene la cantidad de aristas en $n - 1$

Ahora, lo primero es saber si la idea de solución es correcta para el problema dado, lo cual se demostrará con el siguiente teorema:

Teorema(Correctitud) *Sea $G = (V, E)$ un grafo simple,conexo, no dirigido y ponderado, sea T un árbol abarcador de costo mínimo(mst) de G, sea la arista $(u, v) \in E$ tal que $(u, v) \notin T$, luego de insertar la arista (u, v) en T y extraer la arista de mayor peso del ciclo que se forma en en T, el árbol resultante es un mst que contiene a (u, v).*

Demostración:
Se tiene T mst de G.
Sea xy la arista de peso máximo en el camino $p(u - v)$ en T.
Sea $T' = T - xy + uv$ (por Lema 3, T' sigue siendo árbol abarcador de G)
Se tiene que :
$w(T') = w(T) - w(xy) + w(uv) \geq w(x, y)$, si es igual T' mst que contiene a (u, v).
Luego trabajemos con $w(uv) > w(xy)$

Supongamos que T' no es un mst que contiene a u, v.
Entonces tiene que existir al menos una arista e tal que:
$w(ij) > w(e)$
$w(T') + w(e) - w(ij) < w(T')$

Siendo (i, j) una arista que forma parte del ciclo que se forma al añadir e a T' y $(u, v) \neq (i, j)$, porque tiene que estar (u, v) en el árbol para que sea un mst que lo contiene.

Sea T" $= T' - ij + e$
$= T - xy + uv - ij + e$

$w(T") < w(T')$
$w(T - xy + uv - ij + e) < w(T - xy + u, v)$ luego quito (u, v) y coloco a (x, y)
$w(T - xy + xy - u, v + u, v - ij + e) < w(T - xy + xy + u, v - u, v)$
$w(T - ij + e) < w(T)$, lo cual es una contradicción con el hecho de que T es mst de G.
Con lo cual queda demostrado el teorema y la correctitud del algoritmo de solución.

Ahora demostremos la correctitud de cada paso de la implementación

Lema 2: *Para todo (u, v) que no pertenece a T (mst de G), la arista h que une la componente conexa de u con la de v durante la ejecución de Kruskal es la de peso máximo del camino p de u - v.*

Demostración: Supongo que la que une la componente de u y v no sea la más grande del camino, entonces $\Rightarrow \exists$ una arista e cuyo peso $w(e) > w(h)$ y que forma parte del camino p. Si $w(e) > w(h)$ entonces e fue ingresada luego de h, ya que por orden de $h < e$ sería seleccionada primero, por lo cual al agregar e, se formaría un ciclo ya que h conecta a $u...v$.

2.1. Binary Lifting

Definición: Llamamos árbol con raíz a un árbol del cual se selecciona un vértice que llamaremos raíz.

Definición: Llamaremos ancestro de un vértice u de un árbol con raíz, a un vértice v que se encuentra en el camino desde la raíz hasta u,

Definición: Llamaremos padre de un vértice u en un árbol con raíz a un vértice v que es el ancestro de u más cercano a u. O sea, en el camino de la raíz a u, sea (v,u) la última arista, entonces v será el padre de u

Definición: Se dice que el vértice u es hijo del vértice v si v es padre de u.

Definición: En un árbol con raíz se dice que un vértice u es descendiente de un vértice v, si v es ancestro de u.

Definición: Dado un vértice de un árbol con raíz se llama nivel a la distancia del camino desde la raíz hasta ese vértice. Nivel de la raíz = 0.

Definición: Se llama altura (h) al nivel máximo de un vértice de un árbol.

Lema 4: *Dado un árbol con n vértices su altura* $h \leq n - 1$.

Demostración: Directo por el hecho de que un árbol tiene $n - 1$ aristas. Por lo cual no puede existir un camino dentro del árbol cuya distancia sea $> n - 1$ porque eso implicaría la existencia de un ciclo, lo cual a su vez vuelve a contradecir el hecho de que sea un árbol.

Definición: Se llama $lca(u, v)$ al ancestro común más cercano para los vértices u, v. Esto es el vértice l de mayor nivel que forma parte tanto del camino de la raíz a v, como del camino de la raíz a u.

Lema 5*. Si v forma parte del camino desde la raíz a u en un árbol T, v es el* $lca(v, u)$.

Demostración: Se cumple, ya que de existir, tendría que existir un $lca(v, u) \neq v$. $nivel[l] > nivel[v]$, lo cual es una contradicción con el hecho de ser v ancestro de u.

Definición: Un ancestro binario de un vértice v es un ancestro de v que está a una distancia potencia de 2 de v.

Definición: v es el j-ésimo ancestro binario de u, si v es ancestro de u y está a distancia 2^j de u.

Lema 6: *Un vértice v de un árbol con n vértices puede tener a lo sumo* $logn - 1$ *ancestros binarios.*

Supongo que existe un vértice del árbol que tiene $\geq logn$ ancestros binarios, digamos que tiene $logn$, luego el ancestro más lejano estará a distancia $2^{logn} = n$ de v, lo cual indica que existen n aristas en el camino de v a su ancestro pero esto es una contradicción con el hecho de que es un árbol con n vértices (porque tiene n - 1 aristas).

La idea de la técnica de Benary Lifting es precalcular en una matriz $dp[1, n][1, logn]$ donde $dp[v][i]$ contiene la referencia al 2^i ancestro del vértice v, basándose en la siguiente recursión:

$dp[v][i] = padre[v]$ si $j = 0$, $(padre[raiz] = -1)$
$dp[v][i] = dp[dp[v][i - 1]][i - 1]$ si $i > 0$

Esto quiere decir que el k-ésimo ancestro binario de un vértice, es el (k-1)-ésimo ancestro binario de su $(k - 1)$-ésimo ancestro binario. Esto es,en otras palabras, que el ancestro de un nodo

v que está a distancia 2^k de v, es el mismo que el que se encuentra a distancia 2^{k-1} del ancestro que está a distancia 2^{k-1} de v. Y esto es que que $2^k = 2^{k-1} + 2^{k-1}$

Demostración de la recursión:

Inducción sobre las potencias de 2:

Sea $n = 1$
$2^1 = 2 = 2^0 + 2^0 = 1 + 1$. Luego se cumple para $n = 1$

Hipótesis: se cumple para $n = k$
$2^k = 2^{k-1} + 2^k - 1$
Sea $n = k + 1$
$2^{k+1} = 2^k$,2 por hipótesis $2^k = 2^{k-1} + 2^{k-1}$
$2^{k+1} = (2^{k-1} + 2^{k-1})$,2
$2^{k+1} = 2$,$2^{k-1} + 2$,2^{k-1}
$2^{k+1} = 2^k + 2^k$, luego se cumple para $n = k$, por tanto, por inducción matemática se cumple para todo $n \in N$.

2.1.1. lca

Demostración de la correctitud del método $lca(u, v)$

Lo primero que se busca hacer es igualar los niveles de u y v dentro del árbol. Sin pérdida de generalidad seleccionamos v como el de mayor nivel de u y v. La idea del método es ir 'subiendo' por los ancestros de v, utilizando los ancestros binarios de éste hasta alcanzar un ancestro de v al mismo nivel que u y guardando en cada paso de la arista de mayor peso encontradda en el camino que están computadas en dw.

Luego, si u = v, entonces v sería el lca (u,v) y se retorna el costo de la arista de mayor peso encontrada en el camino. Si u no es igual a v, se busca subir a u y v hasta el nivel inmediatamente superior al del lca (u,v), guardando en cada paso el peso de la arista de mayor costo encontrada entre los caminos de u y v hacia el lca. Luego se retorna la mayor entre: las aristas que unen a u y lca(u,v), v y lca,v y la mayor encontrada anteriormente durante los caminos de u,v hasta nivel $[lca(u, v)] + 1$.

Lema 7: *Sea w el i-ésimo ancestro binario de un vértice v está en el nivel* $[w] = nivel \forall = 2^i$

Demostración: Directo, por la definición de ancestro binario y de nivel. Si nivel es la distancia desde la raíz hasta v, y el ancestro binario i-ésimo de v es el ancestro de v que está a distancia 2^i de v. Luego, sea w el i-ésimo ancestro binario de v, la distancia de w a la raíz va a ser igual:
(sea r la raíz)

$d(r,w) = d(r,v) - d(v,w)$, sustituyendo
$nivel(w) = nivel(v) - 2^i$

Demostrar la correctitud de lca:

Lema 8: *La diferencia f entre el nivel de v y u tiene que ser a lo sumo* $n-1$

Demostración: Por Lema 1 (altura) se sabe que el mayor nivel de un árb ol de n vértices es $n-1$ y se sabe que el menor nivel por definición es el de la raíz, cuyo nivel es 0. Luego, supongamos que la altura del árbol es máxima, sea v un vértice del árbol en el nivel $n-1$, su diferencia con el nivel de la raíz es $n-1$; como no hay otro nodo en el árbol cuyo nivel sea menor que la raíz, entonces la diferencia f entre el nivel de u y el nivel de v está acotada por $n-1$.
$nivel[v] - nivel[u] \leq n-1$.

Lema 9: *Todo número entero* $n > 0$ *se puede descomponer como suma de potencias de 2.*

Demostración:
Inducción:
Caso base $n = 1$
$1 = 2^0$

Hipótesis: se cumple para $n = k-1$ ($k-1$ puede descomponerse como suma de potencias de 2).

Sea $n = k$
$k = (k-1) + 1$
$k = (k-1) + 2^0$ (por hipótesis $k-1$ se descompone como suma de potencias de 2, luego sea q una descomposicón de $k-1$ en suma de potencias de 2)
$k = q + 2^0$

por lo cual se cumple para $n = k$. Por inducción matemática se cumple para todo $n \in N$, $n > 0$

Lema 10: $\forall n \in N_+, \exists i \in N$, *tal que* $N - 2^i \geq 0$.

Caso base $n = 1$
$1 - 2^0 = 1 - 1 = 0$

Hipótesis: se cumple para $n = k-1$ $\exists i > 0$ tal que $(k-1) - i \geq 0$
Sea $n = k$
$k = (k-1) + 1$
$k = (k-1) + 2^0$ por hipótesis de inducción para k-1 existe un i que cumple $(k-1) - 2^i \geq 0$
Sea ese i tal que $(k-1) - 2^i \geq 0$
$k - 2^i = (k-1) - 2^i + 1$
$k - 2^i = (k-1) - 2^i \geq 0$
$k - 2^i \leq 1 \geq 0$

Luego se cumple para $n = k$.

Luego, como para todo n natural $\exists i \in N$ tal que $n - 2^i \geq 0$, también existe un máximo i que lo cumple.

Lema 11: *Sea n un número natural, sea i el mayor natural tal que* $n - 2^i \geq 0$*, entonces* $n - 2^i < 2^i$
Supongo que $n - 2^i \geq 2^i$, entonces
$n - 2^i - 2^i \geq 0$
$n - (2{,}2^i) \geq 0$
$n - 2^{i+1} \geq 0$

luego existe $q = i + 1$ tal que $n - 2^q \geq 0$ lo cual es una contradicción con el hecho de que i es el máximo.

Lema 12: *Sea n número natural ¿0, sea i la mayor potencia de 2 tal que* $n - 2^i = q > 0$*. La mayor potencia k de 2 tal que* $q - 2^k \geq 0$ *cumple que* $k < i$*.*
Supongo que $k \geq i$, por Lema 1 $n - 2^i < 2^i$
$q < 2^i$
$q - 2^k < 2^i - 2^k$
$q = 2^k \geq 0$
$0 \leq q - 2^k < 2^i - 2^k$ pero como $i \leq k$, $2^i \leq 2^k \rightarrow 2^i - 2^k \leq 0$
$0 \leq q - 2^k < 2^i - 2^k \leq 0$
$0 \leq q - 2^k < 0$
lo cual es una contradicción. Por tanto $k < i$.

Teorema 1: Luego de terminado el primer ciclo for, en el método lca, el $nivel[v] = nivel[v]$.

Demostración:
Si al inicio, el $nivel[v] = nivel[u]$, entonces se cumple.
Luego se garantiza que de ser distintos $nivel[v] > nivel[u]$.
Luego por Lema 8 se garantiza que la diferencia f entre nivel [v] y nivel [u] es a lo sumo $n - 1$. Como $n - 1 < 2^{log_2 n} = n$ se garantiza por el Lema 9 que existe en el conjunto $0, 1, ..., logn$ en i tal que se cumple que $f - 2^i \geq 0$. Luego se garantiza que hay máximo que lo cumple. Como se itera desde $i = logn, ..., 0$ se garantiza que el primer i que lo cumple será el máximo. Luego por Lema 11 se garantiza que luego de encontrar un máximo tal que $f - 2^i \geq 0$ sigue existiendo un máximo en el conjunto $i, i - 1, ..., 0$ que cumple que la nueva diferencia $f' - 2^i \geq 0$. Luego, si es un momento intermedio la diferencia f entre $nivel[v]$ y $nivel[u]$ se hace 0, ya se cumple. Si no por la unión del Lema 9 y el Lema 11 se garantiza que exisirá un i que sigue cumpliendo la condición y que será máximo. Luego como no hay descenso infinito en los naturales, la diferencia f se hará 0 luego de finalizado el algoritmo, por lo cual $nivel[v] = nivel[u]$.

Luego:

Caso 1: si u es igual a v, significa que u es ancestro de v y por Lema 4, $v = lca(u, v)$.

Como en cada paso se va manteniendo el peso de la mayor arista encontrada, se garantiza que ese será el valor retornado.

Caso 2: $u' \neq v$ significa que u no es el $lca(u, v)$ y existe entonces un vértice l, cuyo $nivel[l] < nivel[u]ynivel[v]$. Luego si u y v siguen el recorrido de sus ancestros al mismo nivel, el primer nivel tal que $u = v$ será el nivel del lca, luego u y v seguirán el camino de sus ancestros hasta el nivel $[l] + 1$, de tal forma que sea $padre[u] = padre[v] = l$.

Teorema 2: Luego de la ejecución del segundo ciclo for de lca(u,v), se cumple que $nivel[u] = nivel[v] = nivel[l] + 1$, siendo $l = lca(u, v)$.

Demostración: Sin pérdida de generalidad utilicemos $nivel[v]$ como $nivel[u]$ y $nivel[v]$.

Sea $f = nivel[v] - nivel[l]$, $f > 0$ porque si $f = 0$ como $lca(u, v) \neq v$ porque sino no llegaría a esta instancia del algoritmo, sino una contradicción con el hecho de ser l el lca(u,v), por lo tanto $f > 0$.

A partir de ahora consideramos f como la diferencia que se obtendría en cada iteración en caso de cumplirse la condición if. Esto es para saber el momento en el que f se podría hacer 0 en el algoritmo, ya que esto no puede ocurrir como se intenta demostrar.

Luego por Lema 10 se garantiza la existencia de un i tal que $f - 2^i \geq 0$, como existe i, existe i que será el máximo que lo cumpla, luego como se itera desde $i = logn, ..., 0$ se garantiza que en cada momento el primer i que lo cumple será el máximo para f.

Ahora, como para todo ancestro w de u y v, tal que $nivel[w] \leq nivel[l]$ (posible w = l)se cumple que es un ancestro común por ser l el lca, si en la iteración son alcanzables como un i-ésimo ancestro, se cumplirá que $dp[u][i] = dp[v][i]$, lo que en la condición if se garantiza que no ocurra ese salto, ya que ese recorrido no forma parte del camino de u, v. Luego puede ocurrir que $w = l$, por lo cual de no actualizar se podría perder el recorrido hasta alcanzar $nivel[l] + 1$ que es lo que se busca, esto no ocurre por el siguiente lema:

Lema 12: *Sea* $k = 2^i, k - 1 = \sum_{j=0}^{i-1} 2^i$

Demostración: como $\sum_{j=0}^{i-1} 2^i$ es una serie geométrica, se tiene que:
$S = \frac{-1+2^{i-1+1}}{-1+2} = -1 + 2^i$ como $2^i = k$, será

$S = k - 1$

De esta forma se garantiza que aunque no haya ocurrido el paso hasta el ancestro 2^i, se alcanzará el ancestro $2^i - 1$ en las siguientes iteraciones, con lo cual se cumpliría el teorema.

Ahora, digamos que $l \neq w$, en toda iteración, por Lema 4, para f seguirá existiendo un i en las iteraciones $i, i-1, ..., 0$, tal que $f - 2^i \geq 0$, y como no hay descenso infinito en los números N, f va a ser igual a 0 en una iteración.
Digamos que fue en la iteración $K > 0$, luego como $f = 0$ $dp[v][k] = dp[v][k] = l$, por la condición if no se realizará ese salto y por el Lema 12 se demuestra que el $nivel[l] + 1$ se alcanza en las siguientes iteraciones, con lo cual se cumpliría el teorema. Ahora, digamos que $f = 0$ se alcanza en la última iteración, esto es $f - 2^0 = 0$, esto es $f = 1$ y como por la condición if u y v se mantienen en su nivel, entonces se cumple que están en el $nivel[l]+1$, con lo cual queda demostrado.

Luego se retorna el valor de la arista de peso máximo, que a su vez será el mayor entre el peso de la arista de mayor peso en todo el recorrido desde u y v, hasta los ancestros a $nivel[l] + 1$ y las aristas que unen a u y v con l, respectivamente.

Análisis temporal de lca:

Cada una de las operaciones que se realizan durante la ejecución de los ciclos for dentro del método lca (u,v), son $O(1)$.

Luego, en cada ciclo for se realizan *logniteraciones* siendo $n = |V|$, luego el tiempo de ejecución del método está acotado por $O(log|V| + log|V|)$ que es $O(log|V|)$.

2.1.2. dft:precalculo

Definición: Se llama descendientes de un vértice v en un árbol a los vértices para los cuales v es un ancestro.

Definición: Se llama hijo de un vértice v a un vértice u tal que u es descendiente de v y adyacente a él.

Definición: Se le llama hoja a un vértice h en un árbol tal que no tiene descendientes.

Definición: Llamemos visitor a un vértice v al momento que se llama a $df + (v, pv$.

Definición: Diremos que un vértice v es visitado desde u, si se llama a $dft(v, u)$, esto es con u como padre de v.

Teorema (DFT): *El algoritmo de recorrido en profundidad de un árbol representado por lista de adyacentes visita cada vértice una sola vez durante su ejecución.*

```
  dft(v,pv)
1     parent[v] = pv
```

```
2     por cada vértice u en ady[v]
3         si u != pv, entonces
4              dft(u,v)

v: vértice pv: vértice padre de v
```

Demostración: Cada vértice será visitado una sola vez por el hecho de que cada vértice solo tiene aristas con su padre y en caso de no ser una hoja, con sus hijos.

Esto es, entonces, supongamos que un momento de la ejecución sobre un vértice llamémosle u, durante el recorrido de su lista de adyacencia (línea 2), $\exists$ un vértice v que ya fue visitado y que vuelve a ser visitado desde u.

Por la condición 3, este vértice no puede ser el padre de u. Luego u visita un vértice distinto de su padre que ya fue visitado; como cada vértice de un árbol tiene solo aristas que lo unen a su padre y aristas con sus hijos, esto implica que v es un hijo de u, pero si v es hijo u, y también ya fue visitado, hay dos situaciones: v era ancestro de u, lo que directamente es una contradicción, o en otro caso hay otro camino en el árbol mediante el cual se visitó v y no forma parte del camino hasta u (porque no es ancestro); entonces tendría que existir un ciclo en el árbol, lo cual es también una contradicción.

Análisis temporal del método $dft(u, v)$

Cada vértice del grafo es visitado una vez durante la ejecución del método, lo cual es $O(|V|)$. Por cada vértice se recorren todos sus adyacentes en la lista de adyacencia, por lo cual luego de finalizada la ejecución se habrá recorrido la lista de adyacencia completa, o sea, se analizará cada visita 2 veces (por u y por v), pero como se está recorriendo un árbol $|E| = |V| - 1$, por lo cual $2|E| = 2|V| - 2$, que es $O(|V|)$, por lo tanto el recorrido de toda la lista de adyacencia es $O(V)$ para el árbol. Luego por cada vértice se hace $logn$ iteraciones (donde $n = |V|$) para precomputar las matrices dp y dw. Cada asignación dentro del ciclo for es $O(1)$, luego el ciclo se ejecuta en $O(log|V|)$. Como este proceso se repite por cada vértice, entonces quedará la ejecución del método acotada por $O(|V|log|V|)$.

2.2. Kruskal

```
Krukal(G:conexo,acíclico,no dirigido,ponderado)
1    T  = {}, S = E[G]
2    desde i = 1 hasta |V|-1
3        extraer la menor arista e en S tal que
         T+{e} es acíclico
4        T = T+{e}
5    devolver T
```

Teorema: *El algoritmo de Kruskal devuelve un árbol abarcador de costo mínimo(mst).*

Demostración:

Demostremos que T, el grafo generado por el algoritmo es un árbol abarcador. T no puede tener ciclos, ya que solo se añaden a T aristas bajo la condición de que no produzcan ciclos, luego T es acíclico. Luego, si cada arista e es añadida a T bajo la condición de no producir un ciclo en T, sea $e = (u, v)$, e debe conectar dos componentes conexos de T de forma que antes de añadir a e, $u \in C_{c_u} y v \in C_{c_v}$ con $C_{c_u} \neq C_{c_v}$ y luego de añadido e a T, $u \in C_{c_u,v} y v \in C_{c_u,v} con C_{c_u,v} = C_{c_u} \cup C_{c_v}$.

Para demostrarlo, supongamos lo contrario, esto es que $e = (u, v)$ es añadido a T con u,v que pertenecen a la misma componente conexa. Luego si u,v pertenecen a la misma componente conexa antes de agregar e, entonces existía un camino de u,v en T, luego al añadir la arista $e = (u, v)$ se formaría un ciclo, lo cual contradice que sea acíclico así como contradice la inclusión de e en T. Luego en cada inclusión se unen dos componentes conexos de T, luego se anñade a lo sumo $n - 1$ vértice para unir n componentes conexas, por lo cual T es acíclico, conexo y tiene $n - 1$ aristas, lo que significa que T es un árbol y es un árbol abarcador de G pues contiene a todos sus vértices.

Demostremos que T es mínimo:

Sea $E(T) = \{e_1, e_2, e_3, ..., e_{n-1}\}$ la lista de las aristas de T en el orden en que fueron agregadas a T, esto implica
$w(T) = \sum_{i=1}^{n-1} w(e_i)$ y,
$w(e_i) \leq w(e_j), \forall i, j : i \leq j.$

Supongamos que T no es un mínimo mst, luego sea M el mst de G con el mínimo número de aristas que no están en T.

Como T no es un mst de G , T y M no pueden ser idénticos.

Sea ahora e_i la primera arista en $E(T)$ que no está en M. Si insertamos e_i en M, tendremos $M' = M + e_i$ un grafo que contiene un ciclo.
Luego como T es acíclico, existe una arista e en el ciclo de M' que no está en T. Luego se tiene que el grafo $M" = M' - e$ es un árbol abarcador de G (Lema 3) y:

$w(M") = w(M) - w(e) + w(e_i)$, como $w(M) \leq w(M")$, (porque M es un mst)
Entonces $w(e) \leq w(e_i)$.

Pero durante la ejecución del algoritmo al añadir la arista e_i se garantiza que es una arista de peso mńimo que al ser añadida mantiene acíclico el grafo conformado por $e_1, e_2, ..., e_i$. Al mismo tiempo, todas estas aristas, excepto e_i forman parte de M, por lo que $e_1, e_2, ..., e_{i-1}$ es también acíclico, con lo cual si durante la ejecución si se escogiøe_i antes de e para formar parte del grafo $e_1, e_2, ..., e_{i-1}$ entonces se tiene que $w(e_i) \leq w(e)$.

Como se había llegado anteriormente a que $w(e) \leq w(e_i)$, entonces se tiene que $w(e) = w(e_i)$.

Con este resultado se tiene entonces que $M"$ es también un mst de G, porque

$w(M") = w(M) - w(e) + w(e_i)$
$w(M") = w(M)$

Y a su vez se tiene que $M"$ tiene al menos un vértice más en común con T que M, lo cual es una contradicción con el hecho de que M es el mst con menos cantidad de vértices que no están en T.

Luego se tiene que T es un mst de G.

Análisis de la complejidad temporal del algoritmo de Kruskal:

La complejidad temporal del algoritmo dependerá de la forma en que se busquen las aristas así como de la forma de determinar si forman un ciclo luego de su agregación.

Primero nos detendremos en el hecho de que cada arista que se agrega a T es la mínima para la cual se cumple que T sigue siendo acíclico luego de su agregación, y por tanto el conjunto $E(T)$ de las aristas en su orden de agregación a T es $e_1, e_2, ..., e_{n-1}$ y cumple que: $w(e_i) \leq w(e_i + 1)$. Luego lo primero que se puede hacer en el algoritmo es ordenar las aristas por su peso y recorrerlas de forma lineal comprobando si pueden ser agregadas a T. Esto se garantiza porque para cualquier par de aristas e_i, e_j que cumplan la condición en la iteración k del algoritmo, si $w(e_i) < w(e_j)$, e_i sería agregado y e_j no podría formar parte ya que formaría un ciclo en T.

Luego recorrer las aristas mientras T no sea el árbol abarcador será $O(|E|)$. Para representar el árbol T y las operaciones para agregar las aristas, con el empleo del disjoint set[1], con los métodos SetOf y Merge implementado por union by rank and path compression, se puede saber si u y v pertenecen o no a la misma componente conexa a través del SetOf en $O(log|V|)$ y para agregar la arista a T en $O(1)$ con Merge(implementado sin SetOf, comprobación fuera del método), luego la ejecución del algoritmo de Kruskal que realicé para la solución del problema sería:

Por cada inicialización $O(|V|)$, luego ordenar[2] las aristas del grafo $O(|E|log|E|)$, luego en el recorrido de las aristas el peor caso se realiza $|E|$ iteraciones y en cada una se se hacen dos llamados a SetOf que es $O(log|V|)$; el resto de operaciones dentro del ciclo se realiza en $O(1)$, por lo cual crear el árbol T durante el ciclo será $O(|E||log|V|)$.

Como $|E| < |V|^2$ $log|E| < log|V|^2$ $log|E| < 2log|V|$

por lo que $log|E|$ es $O(log|V|)$ y por tanto el tiempo de ejecución de Kruskal será $O(|E|log|V|$

.

[1]Demostración de la complejidad temporal del Disjoint Set en Introduction algorithm, Third Edition Sector 214, pag. 573

[2]utilizando el sort de python, añadido a la bibliografía