Lecture: Second Best Minimum Spanning Tree -- Minimum Spanning Tree II.

Unit: 6.

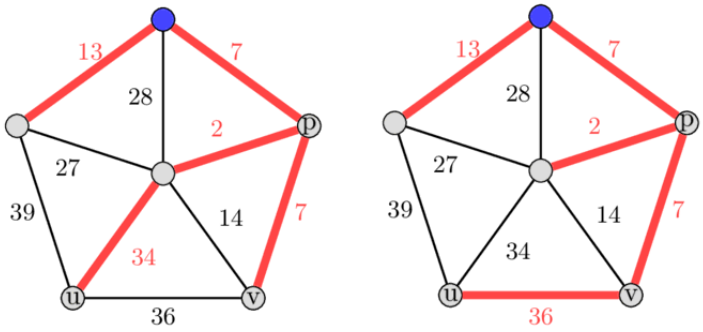
Instructor: Fredin.

SECOND MINIMUM SPANNING TREE (SECOND BEST)

Este se trata de literalmente encontrar la segunda (mejor) opción que siga cumpliendo con todas las reglas que un mínimum spanning tree cumple, con las propiedades que un árbol son capaces de cumplir además de su sentido o significado en el problema.

Veamos un ejemplo, es posible pensar sobre un ejemplo en donde el primer spanning tree no sea útil, esto bien podría ser en un caso en donde tenemos **'n'** pueblos o ciudades y se nos otorga una serie de caminos los cuales estará conformando un grafo, de modo que las aristas son los potenciales caminos a construir para que la comunicación entre las 'n' pueblos o ciudades sea con el menor costo posible. Por lo cual podríamos imaginar que el primer camino que se construye son para usuarios con cierto prestigio a nivel económico, aquellas que pagan por circular ahí, mientras que la segunda ruta sea para el resto de los usuarios. Así que la pregunta sería para un usuario, ¿qué le conviene más?, ¿pagar por la ruta privada o escoger la ruta común? *Esto en consideración del costo que genera al circular cada camino.*

Entonces, ya se ve la necesidad de considerar otro camino el cual en términos de grafos el segundo mejor camino es posible encontrarlo tan solo al intercambiar una arista del *MSP* no contemplada anteriormente:



MSP-original Second Best MSP

Por lo cual una propiedad a definir será que esta segunda solución debe tener por lo menos una arista que no corresponda al *MSP* original. De modo que es posible tan solo reemplazar **1 o más de 2 aristas** para encontrar el segundo *MSP*, entonces va diferir de 1 o más aristas que el Original.

Por lo cual, todo se resume en encontrar:

Minimo(MST_segundo – MST_original)

Condición o meta a llegar

Nota: es de esta manera debido a que el segundo siempre tendrá un peso mayor al original, debido a que este es el mejor, el primero encontrado.

LOWEST COMMON ANCESTOR

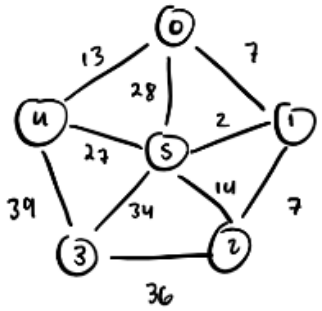
Un acercamiento fue por medio del algoritmo ya conocido, *Kruskal* apoyándose de *UNION_FIND* para que sea más eficiente.

Esta solución podría recordar un poco a las soluciones dadas por una búsqueda completa, debido a que en síntesis ese algoritmo se dedica en ir removiendo aristas del *MSP* original para reemplazarla con una arista no usada con el fin de ir comprobando su tamaño para cumplir con:

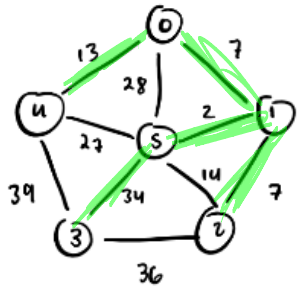
Minimo(MST_segundo – MST_original)

Pero en este acercamiento la estrategia a seguir será que en lugar de eliminar aristas del *MSP* original se tendrá que añadir las aristas, una por una, que no fueron contempladas para la solución original.

De modo que se tendría lo siguiente:



En donde el grafo mínimum spanning tree es el siguiente:

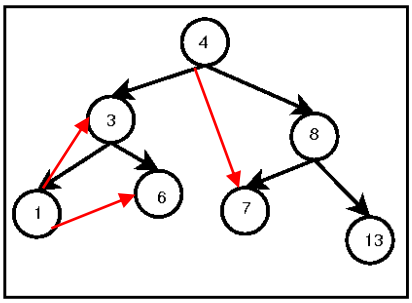


Peso total: 63

Ahora, desarrollando la idea de este algoritmo lo que se debe de hacer es que al contemplar estas aristas que se tiene actualmente usadas se tendrá que añadir una arista no contemplada anteriormente.

Se debe de tener en cuenta es el hecho de **cómo se comportan** un árbol al agregar una arista, y es que sea cual sea (excepto para la raíz) la arista a añadir siempre se va a generar un ciclo.

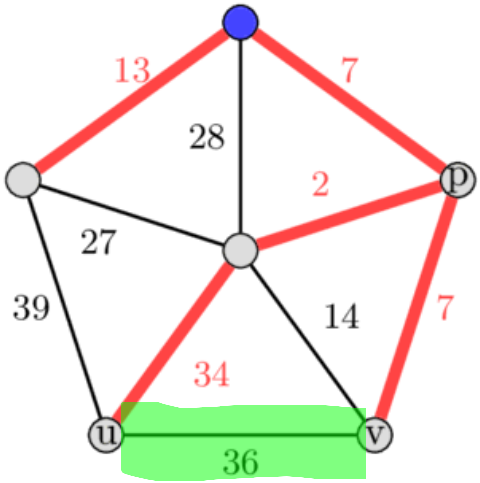
Un árbol es un grafo sin ciclos, además de otras propiedades, tal que la cantidad de aristas que tiene son las suficientes para poder cumplir con la solicitud de llegar por un camino de un vértice a otro, de forma que al ponerle una arista extra se generará un ciclo:



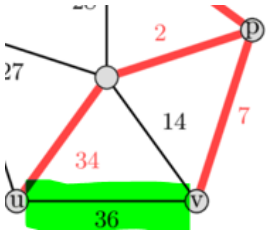
De modo que al añadir una arista se obtendrá un ciclo, una vez contemplado eso nos debemos de enfocar en dicho ciclo esto se debe a que no es posible contemplar dicho ciclo así que **debemos de hacer la eliminación de una arista para quitar ese ciclo.**

CRITERIO DE ELIMINACIÓN

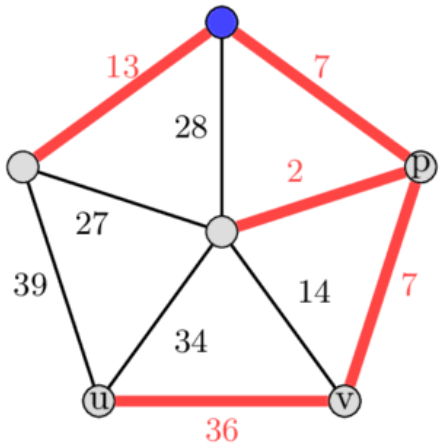
El criterio a seguir para eliminar la arista es contemplar cuál de esas aristas es mayor, sin tomar en cuenta la arista agregada solo las aristas del árbol original. Una vez que fue contemplada esa arista (la de mayor peso) se deberá de considerar su eliminación.



Se deberá de suponer que la arista agregada en este caso será aquella que corresponde a un peso de 36, por ende ya tenemos el ciclo formado. Así que ahora se debe de computar, encontrar, una arista que sea diferente a la agregada tal que cumpla que sea la mayor dentro del ciclo:



Entonces, visualmente es fácil ver que la arista a eliminar será aquella con un peso de 34, puesto que se trata de la mayor. Entonces se elimina del MSP actual y obtenemos el siguiente MSP:



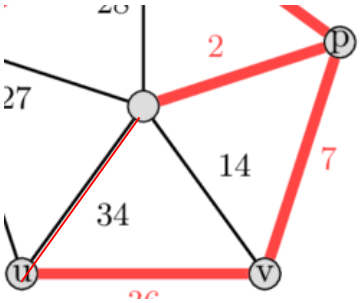
Esto mismo se tendrá que realizar con todas aquellas aristas que no pertenecían al árbol principal, aunque en este caso este ya es el Second best MSP.

Nota: una cosa a observar es que la arista a eliminar es la mayor para poder hacer que la diferencia de pesos entre la agregada con la eliminada sea la más pequeña, en este caso la diferencia es de 2.

Otra punto a tomar en cuenta será la arista que se debe de añadir, esto es debido a que las demás aristas en ese grafo nos daría un peso a un mayor. Entonces, podría decirse que la arista a añadir deberá tener un peso cercano a la arista que estará formado a los vértices hoja.

De modo a que la complejidad, de razonamiento, se reduce a modelar la forma de calcular esa arista mayor, y como tal es el nombre del algoritmo, se usará *Lowest common ancestor* para realizar esta tarea (LCA).

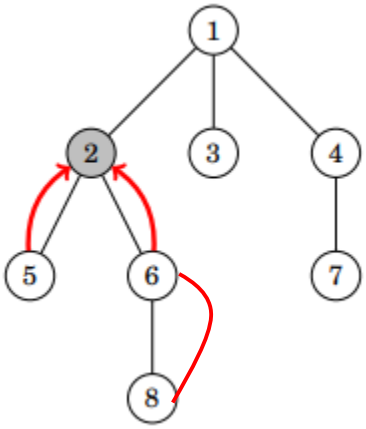
El uso de LCA será para poder computar aquella arista cuyo peso es menor del camino formado de 'u' al ancestro y de 'v' al ancestro.



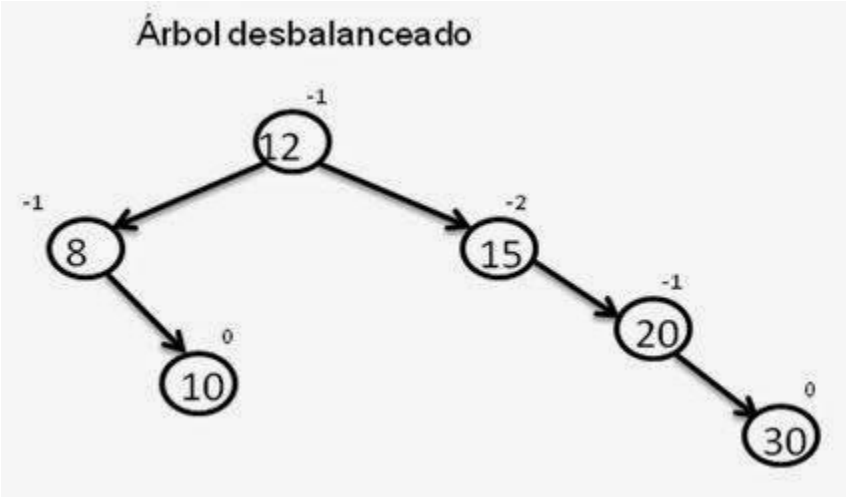
MÉTODO POR BÚSQUEDA COMPLETA

Este sería el método más rápido a pensar y a la vez la más ineficiente cuando se presentan varias queries que se debe de procesar.

El problema que resuelve este algoritmo es encontrar el ancestro en común más cercano para dos vértices dados, por lo cual simplemente se podría tomar el vértice ‘u’ e ir subiendo en su camino hasta llegar con la raíz, recorrer el árbol, mientras que memorizamos los vértices recorridos para después realizar lo mismo con el vértice ‘v’ y comparar si coinciden en algún vértice, en caso de que esto sea cierto se tendrá que ese vértice en donde coinciden ambos recorridos hacia la raíz corresponde a un vértice ancestro en común.



Esto sería en tiempo logarítmico, no obstante se debe de tener en cuenta aquel caso en donde el árbol esté muy desbalanceado provocando que esto llegue a ser lineal.



This requires $O(n)$ per (u, v) query and can be very slow if there are many queries.

USANDO UN RECORRIDO

Esto sería algo similar a lo realizado por un range mínimum query.

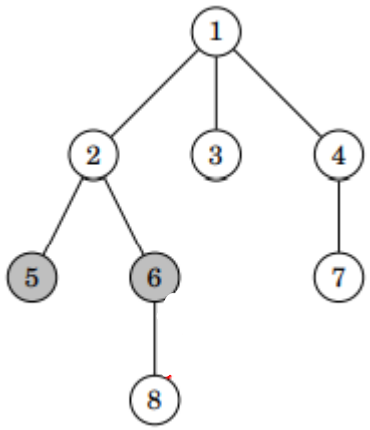
La idea será usando el recorrido de grafo DFS con un ligero retoque, el cual será que vamos a tener un arreglo en donde se estarán guardando los vértices que se van visitando y también se guardará el nivel en el que se encuentra dicho vértice. De modo que no va a importar si ese vértice fue ya visitado, lo importante es que se está pasando por él.

Nota: el dfs explicado en el grafo actual, en donde ya fue identificado el vértice ‘u’ y ‘v’.

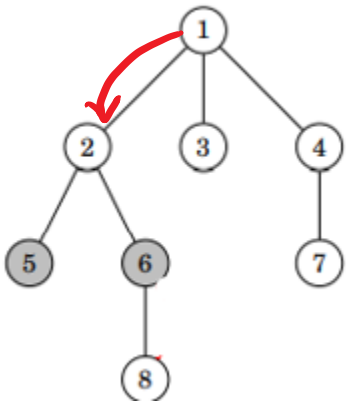
El arreglo que contendrá a los vértices y el nivel en donde se encuentra será de tamaño:

2*vértices-1

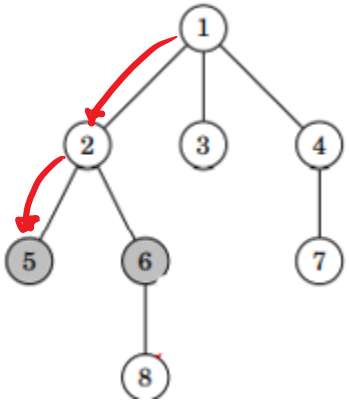
Esto se debe a que al ser no dirigido se podrá visitar el mismo vértice de ida y de vuelta.



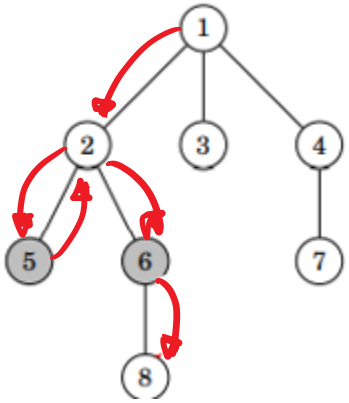
indice	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V															
L															



indice	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	1	2													
L	1	2													

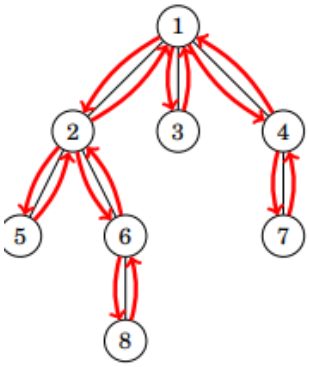


indice	.1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	1	2	5												
L	1	2	3												



indice	.1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
V	1	2	5	2	6	8									
L	1	2	3	2	3	4									

Así hasta recorrer todo el árbol obteniendo



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
node id	1	2	5	2	6	8	6	2	1	3	1	4	7	4	1
level	1	2	3	2	3	4	3	2	1	2	1	2	3	2	1

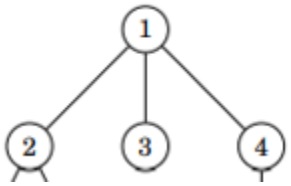
BUSCAR EL ANCESTRO EN COMÚN

Para esto se deberá de buscar en las posiciones en donde se encuentre los vértices para después buscar en ese rango (excluyendo los vértices de los cuales se busca el ancestro), qué vértice tiene el nivel más pequeño. El vértice que tenga el nivel más pequeño será el que esté más cerca de la raíz, por lo cual ese vértice será el ancestro en común más cercano para ‘u’ y ‘v’.

Un ejemplo sería en ese árbol buscar el ancestro en común de 2 y 4, debido a que estos aparecen más de una vez en el arreglo:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
node id	1	2	5	2	6	8	6	2	1	3	1	4	7	4	1
level	1	2	3	2	3	4	3	2	1	2	1	2	3	2	1

El vértice menor nivel es el vértice 1, por lo tanto ese es nuestro ancestro, haciendo la comprobación:



Entonces, una vez conocido dicho ancestro se efectúa el camino para ver quién es la arista con el peso más grande.