Tarea 4
Algoritmos

Emmanuel Peto Gutiérrez
José Luis Vázquez Lázaro

26 de octubre de 2022

Problema 4

a)

Se recibe una gráfica $G = (V, E)$ y se construye una gráfica $G' = (V, E')$, donde el conjunto de aristas E' es el mismo que E pero sus aristas tienen como peso el inverso aditivo de su correspondiente en E. Es decir, si uv es una arista de E con peso p, entonces uv está en E' pero su peso es $-p$.

Se ejecuta el algoritmo de Kruskal sobre G' para obtener un árbol generador de peso mínimo T'. Luego, se construye un árbol T con las mismas aristas de T' pero con el inverso aditivo de los pesos. Así, el árbol T es el árbol generador de peso máximo de G.

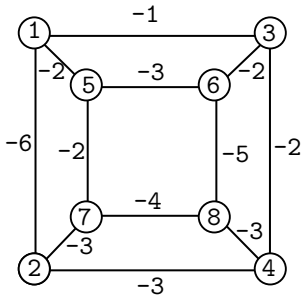
Algoritmo 1: ArbolMaximo($G = (V, E)$)

Entrada: Una gráfica G en su representación de lista de aristas.
Salida: Lista de aristas T que forman un árbol generador de peso máximo de G.

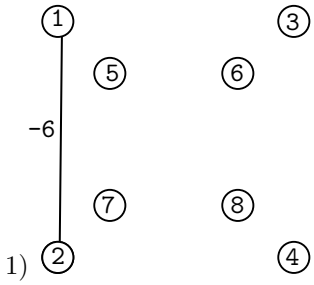
```
1 E' = ∅;
2 for (uv, p) ∈ E do
3     E'.Add((uv, −p));
4 G' = (V, E');
5 T' = Kruskal(G');
6 T = ∅;
7 for (uv, p) ∈ T' do
8     T.Add((uv, −p));
```

b)

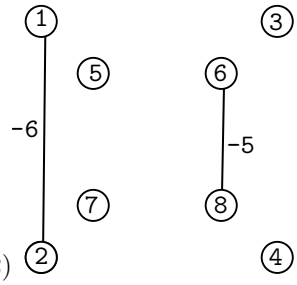
Gráfica G':



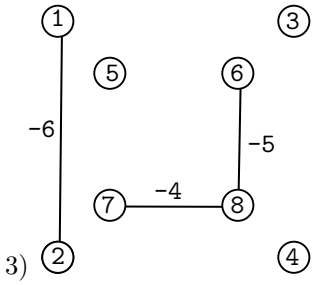
Ejecución:



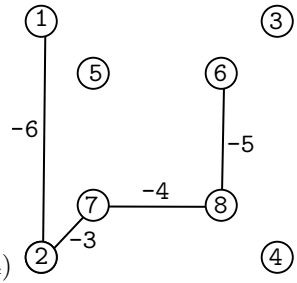
1)



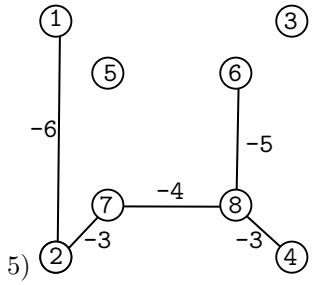
2)



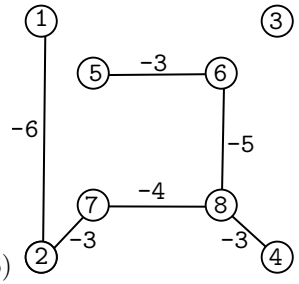
3)



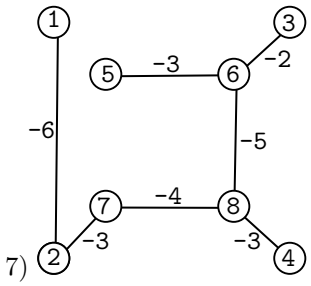
4)



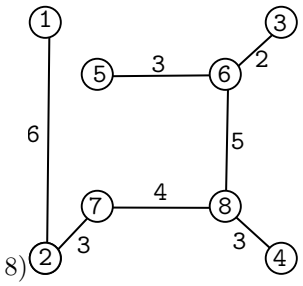
5)



6)



7)



8)

De los pasos 1) al 7) es la ejecución del algoritmo de Kruskal. Del paso 7) al 8) es la construcción del árbol de peso máximo T a partir de T'.

c)

Se tiene como precondición que G tiene solamente pesos no negativos, así que la gráfica G' solo tendrá aristas con pesos menores o iguales a 0. Al ejecutar el algoritmo de Kruskal se agregan las aristas en orden creciente (siempre que no se forme un ciclo), y dado que todas serán negativas se toman primero las aristas cuyo valor absoluto es el mayor.

Al terminar el algoritmo, se tendrá un árbol T' y $w(T') = p$. Por la correctitud del algoritmo de Kruskal se tiene que cualquier otro árbol generador de G' tendrá un peso mayor o igual a p. Como todas las aristas de G' son negativas, cualquier árbol generador U de G' tendrá peso negativo y como $w(T')$ es el menor de esos pesos, entonces $|w(T')| \geq |w(U)|$.

Por la construcción de T, se tiene que $w(T) = |w(T')|$. T es un árbol generador de G y para cualquier otro árbol generador S de G se tendrá que $w(S) \leq w(T)$, y por lo tanto T es un árbol generador de peso máximo de G. ■

d)

Se va a suponer que la gráfica está en su representación por lista de aristas y que el algoritmo de Kruskal usa la estructura de datos Union-Find optimizada. El orden de la gráfica es n y su tamaño es m.

- Construir la gráfica G' a partir de G toma tiempo $\Theta(m)$.
- Ejecutar el algoritmo de Kruskal toma tiempo $O(m \log n)$.
- Construir T a partir de T' toma tiempo $\Theta(n)$, pues un árbol generador tiene $n - 1$ aristas.

Entonces la complejidad es $O(m \log n + m + n)$. Como el término que crece más rápido es $m \log n$ se puede decir que la complejidad es $O(m \log n)$.

Problema 5

a)

Se va a suponer que la entrada es una gráfica G con pesos en su representación con listas de adyacencias. Cada elemento en una lista de adyacencias de v es un par (u, p), donde u es un vecino de v y p es el peso de la arista vu.

También se supone que existe una función EsConexa que recibe una gráfica en su representación por listas de adyacencias y devuelve 1 si es conexa y 0 si no lo es.

El algoritmo se ejecuta de la siguiente forma.

1) Recorrer las listas de adyacencias de todos los vértices. Si se encuentra un par (u, p) tal que $p > b$, entonces se elimina al elemento (u, p) de la lista.

2) Al terminar el paso 1) se tendrá una subgráfica generadora $H \subseteq G$. Ejecutar la función EsConexa sobre H y devolver el mismo valor de retorno de la función EsConexa.

Nota: La función EsConexa se puede crear de la siguiente manera: inicialmente se marcan todos los vértices como *no visitado* (puede ser un atributo booleano del vértice). Ejecutar DFS sobre la gráfica, a partir de un vértice arbitrario, y marcar como *visitado* los que se visiten por el algoritmo DFS. Si todos los vértices están marcados como visitados entonces se devuelve 1, en otro caso se devuelve 0.

En el Algoritmo CuelloDeBotella se muestra un pseudocódigo que resuelve el problema. Cada vértice v tiene una lista de sus vecinos llamada *ady*. Se utiliza un iterador para recorrer las listas de adyacencias, la función Next() devuelve al siguiente elemento en la lista y la función Delete() elimina al último elemento devuelto.[1]

Algoritmo 2: CuelloDeBotella$(G = (V, E), b)$

Entrada: Una gráfica G en su representación por listas de adyacencias y un número b.

Salida: 1 si existe una subgráfica generadora conexa donde cada arista tiene peso a lo más b; 0 en otro caso.

```
1 for v ∈ V do
2     iterador = v.ady.ListIterator();
3     while iterador.HasNext() do
4         par = iterador.Next();
5         if par.peso > b then
6             iterador.Delete();
7 return EsConexa(G);
```

b)

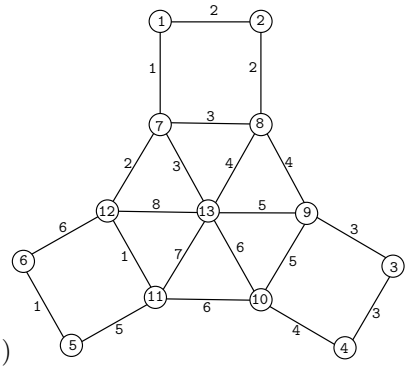
En una gráfica no dirigida, en la representación de listas de adyacencias, para cada arista $uv \in E$, u aparece en la lista de adyacencias de v y v aparece en la lista de adyacencias de u. Sin embargo, para reducir el número de figuras en esta tarea se hará una simplificación: cada vez que se borre al vértice v de la lista de u, también se va a borrar u de la lista de v, entendiendo que en el algoritmo

[1] Este iterador funciona como el *ListIterator* de Java.

real la segunda eliminación puede ocurrir varias iteraciones después. Tampoco se mostrará la ejecución del algoritmo DFS (de la función EsConexa).

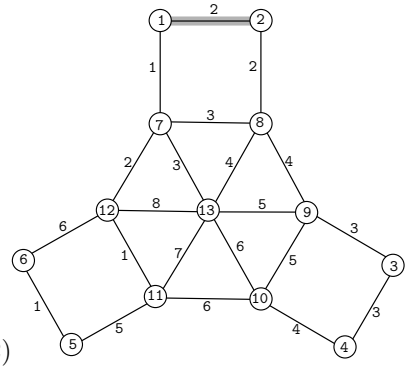
En el algoritmo se revisan todas las aristas, y para seguir la pista del algoritmo se va a sombrear de gris la arista que se está revisando en la iteración actual.

1)



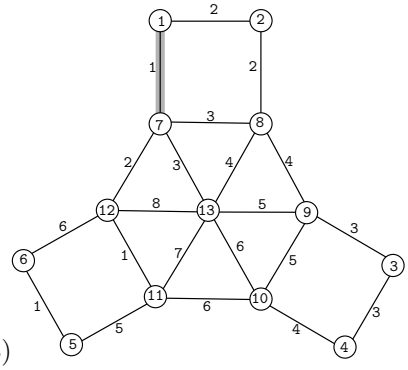
1 →	2,2	7,1				
2 →	1,2	8,2				
3 →	4,3	9,3				
4 →	3,3	10,4				
5 →	6,1	11,5				
6 →	5,1	12,6				
7 →	1,1	8,3	12,2	13,3		
8 →	2,2	7,3	9,4	13,4		
9 →	8,4	10,5	13,5	3,3		
10 →	4,4	9,5	11,6	13,6		
11 →	5,5	10,6	12,1	13,7		
12 →	6,6	7,2	11,1	13,8		
13 →	7,3	8,4	9,5	10,6	11,7	12,8

2)



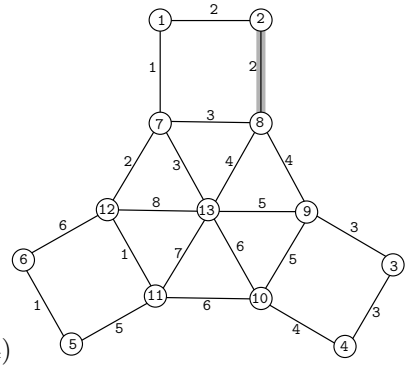
1 →	2,2	7,1				
2 →	1,2	8,2				
3 →	4,3	9,3				
4 →	3,3	10,4				
5 →	6,1	11,5				
6 →	5,1	12,6				
7 →	1,1	8,3	12,2	13,3		
8 →	2,2	7,3	9,4	13,4		
9 →	8,4	10,5	13,5	3,3		
10 →	4,4	9,5	11,6	13,6		
11 →	5,5	10,6	12,1	13,7		
12 →	6,6	7,2	11,1	13,8		
13 →	7,3	8,4	9,5	10,6	11,7	12,8

3)



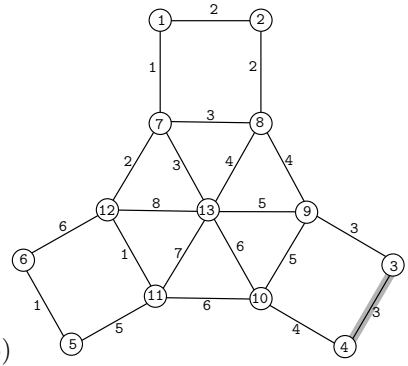
1 →	2,2	7,1				
2 →	1,2	8,2				
3 →	4,3	9,3				
4 →	3,3	10,4				
5 →	6,1	11,5				
6 →	5,1	12,6				
7 →	1,1	8,3	12,2	13,3		
8 →	2,2	7,3	9,4	13,4		
9 →	8,4	10,5	13,5	3,3		
10 →	4,4	9,5	11,6	13,6		
11 →	5,5	10,6	12,1	13,7		
12 →	6,6	7,2	11,1	13,8		
13 →	7,3	8,4	9,5	10,6	11,7	12,8

4)



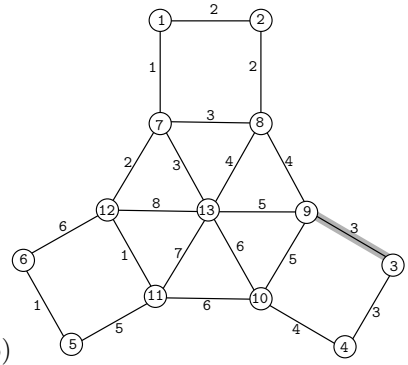
1 →	2,2	7,1				
2 →	1,2	8,2				
3 →	4,3	9,3				
4 →	3,3	10,4				
5 →	6,1	11,5				
6 →	5,1	12,6				
7 →	1,1	8,3	12,2	13,3		
8 →	2,2	7,3	9,4	13,4		
9 →	8,4	10,5	13,5	3,3		
10 →	4,4	9,5	11,6	13,6		
11 →	5,5	10,6	12,1	13,7		
12 →	6,6	7,2	11,1	13,8		
13 →	7,3	8,4	9,5	10,6	11,7	12,8

5)



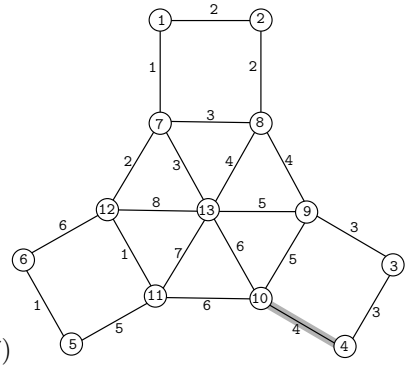
1 →	2,2	7,1				
2 →	1,2	8,2				
3 →	4,3	9,3				
4 →	3,3	10,4				
5 →	6,1	11,5				
6 →	5,1	12,6				
7 →	1,1	8,3	12,2	13,3		
8 →	2,2	7,3	9,4	13,4		
9 →	8,4	10,5	13,5	3,3		
10 →	4,4	9,5	11,6	13,6		
11 →	5,5	10,6	12,1	13,7		
12 →	6,6	7,2	11,1	13,8		
13 →	7,3	8,4	9,5	10,6	11,7	12,8

6)



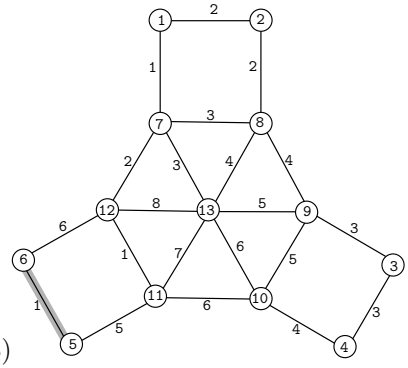
1 →	2,2	7,1				
2 →	1,2	8,2				
3 →	4,3	9,3				
4 →	3,3	10,4				
5 →	6,1	11,5				
6 →	5,1	12,6				
7 →	1,1	8,3	12,2	13,3		
8 →	2,2	7,3	9,4	13,4		
9 →	8,4	10,5	13,5	3,3		
10 →	4,4	9,5	11,6	13,6		
11 →	5,5	10,6	12,1	13,7		
12 →	6,6	7,2	11,1	13,8		
13 →	7,3	8,4	9,5	10,6	11,7	12,8

7)



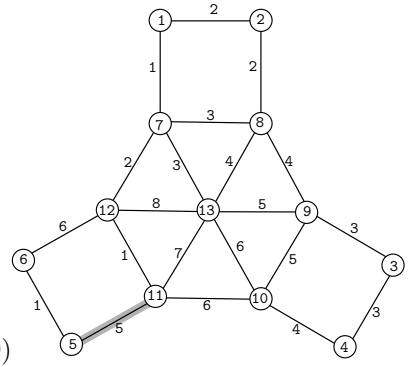
1	2,2	7,1				
2	1,2	8,2				
3	4,3	9,3				
4	3,3	10,4				
5	6,1	11,5				
6	5,1	12,6				
7	1,1	8,3	12,2	13,3		
8	2,2	7,3	9,4	13,4		
9	8,4	10,5	13,5	3,3		
10	4,4	9,5	11,6	13,6		
11	5,5	10,6	12,1	13,7		
12	6,6	7,2	11,1	13,8		
13	7,3	8,4	9,5	10,6	11,7	12,8

8)



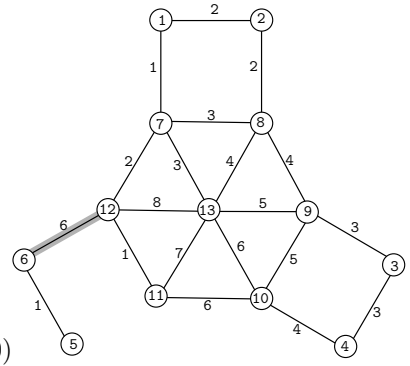
1	2,2	7,1				
2	1,2	8,2				
3	4,3	9,3				
4	3,3	10,4				
5	6,1	11,5				
6	5,1	12,6				
7	1,1	8,3	12,2	13,3		
8	2,2	7,3	9,4	13,4		
9	8,4	10,5	13,5	3,3		
10	4,4	9,5	11,6	13,6		
11	5,5	10,6	12,1	13,7		
12	6,6	7,2	11,1	13,8		
13	7,3	8,4	9,5	10,6	11,7	12,8

9)



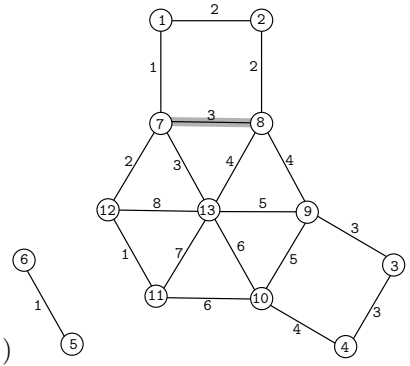
1	2,2	7,1				
2	1,2	8,2				
3	4,3	9,3				
4	3,3	10,4				
5	6,1	11,5				
6	5,1	12,6				
7	1,1	8,3	12,2	13,3		
8	2,2	7,3	9,4	13,4		
9	8,4	10,5	13,5	3,3		
10	4,4	9,5	11,6	13,6		
11	5,5	10,6	12,1	13,7		
12	6,6	7,2	11,1	13,8		
13	7,3	8,4	9,5	10,6	11,7	12,8

10)



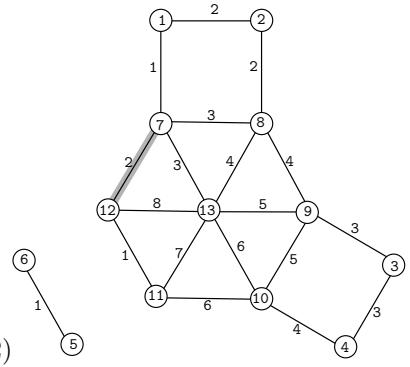
1	2,2	7,1				
2	1,2	8,2				
3	4,3	9,3				
4	3,3	10,4				
5	6,1					
6	5,1	12,6				
7	1,1	8,3	12,2	13,3		
8	2,2	7,3	9,4	13,4		
9	8,4	10,5	13,5	3,3		
10	4,4	9,5	11,6	13,6		
11	10,6	12,1	13,7			
12	6,6	7,2	11,1	13,8		
13	7,3	8,4	9,5	10,6	11,7	12,8

11)



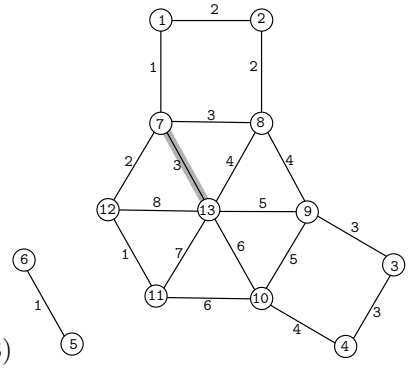
1	2,2	7,1				
2	1,2	8,2				
3	4,3	9,3				
4	3,3	10,4				
5	6,1					
6	5,1					
7	1,1	8,3	12,2	13,3		
8	2,2	7,3	9,4	13,4		
9	8,4	10,5	13,5	3,3		
10	4,4	9,5	11,6	13,6		
11	10,6	12,1	13,7			
12	7,2	11,1	13,8			
13	7,3	8,4	9,5	10,6	11,7	12,8

12)



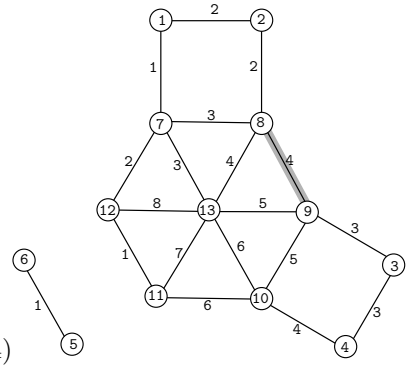
1	2,2	7,1				
2	1,2	8,2				
3	4,3	9,3				
4	3,3	10,4				
5	6,1					
6	5,1					
7	1,1	8,3	12,2	13,3		
8	2,2	7,3	9,4	13,4		
9	8,4	10,5	13,5	3,3		
10	4,4	9,5	11,6	13,6		
11	10,6	12,1	13,7			
12	7,2	11,1	13,8			
13	7,3	8,4	9,5	10,6	11,7	12,8

13)



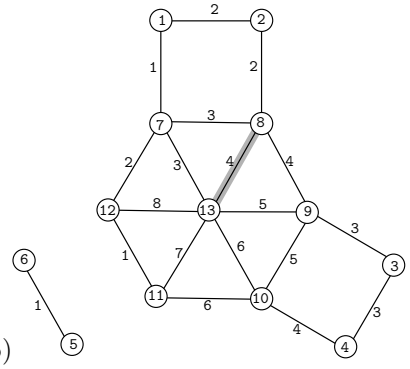
1 →	2,2	7,1				
2 →	1,2	8,2				
3 →	4,3	9,3				
4 →	3,3	10,4				
5 →	6,1					
6 →	5,1					
7 →	1,1	8,3	12,2	13,3		
8 →	2,2	7,3	9,4	13,4		
9 →	8,4	10,5	13,5	3,3		
10 →	4,4	9,5	11,6	13,6		
11 →	10,6	12,1	13,7			
12 →	7,2	11,1	13,8			
13 →	7,3	8,4	9,5	10,6	11,7	12,8

14)



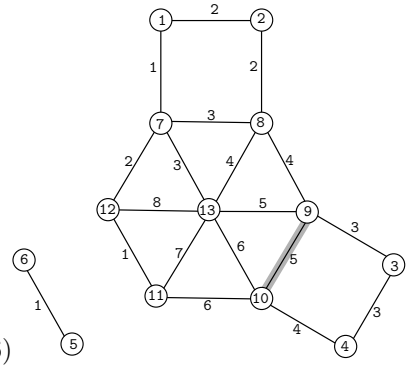
1 →	2,2	7,1				
2 →	1,2	8,2				
3 →	4,3	9,3				
4 →	3,3	10,4				
5 →	6,1					
6 →	5,1					
7 →	1,1	8,3	12,2	13,3		
8 →	2,2	7,3	9,4	13,4		
9 →	8,4	10,5	13,5	3,3		
10 →	4,4	9,5	11,6	13,6		
11 →	10,6	12,1	13,7			
12 →	7,2	11,1	13,8			
13 →	7,3	8,4	9,5	10,6	11,7	12,8

15)



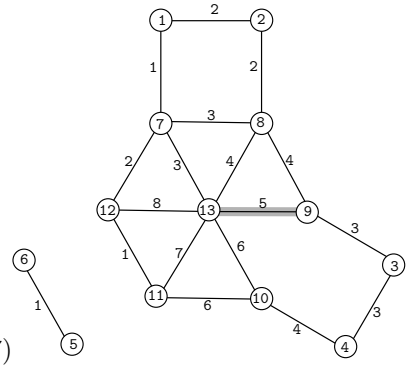
1 →	2,2	7,1				
2 →	1,2	8,2				
3 →	4,3	9,3				
4 →	3,3	10,4				
5 →	6,1					
6 →	5,1					
7 →	1,1	8,3	12,2	13,3		
8 →	2,2	7,3	9,4	13,4		
9 →	8,4	10,5	13,5	3,3		
10 →	4,4	9,5	11,6	13,6		
11 →	10,6	12,1	13,7			
12 →	7,2	11,1	13,8			
13 →	7,3	8,4	9,5	10,6	11,7	12,8

16)



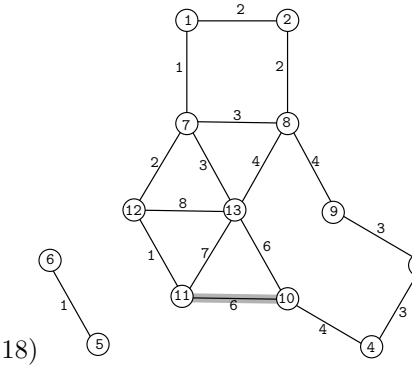
1	2,2	7,1				
2	1,2	8,2				
3	4,3	9,3				
4	3,3	10,4				
5	6,1					
6	5,1					
7	1,1	8,3	12,2	13,3		
8	2,2	7,3	9,4	13,4		
9	8,4	10,5	13,5	3,3		
10	4,4	9,5	11,6	13,6		
11	10,6	12,1	13,7			
12	7,2	11,1	13,8			
13	7,3	8,4	9,5	10,6	11,7	12,8

17)



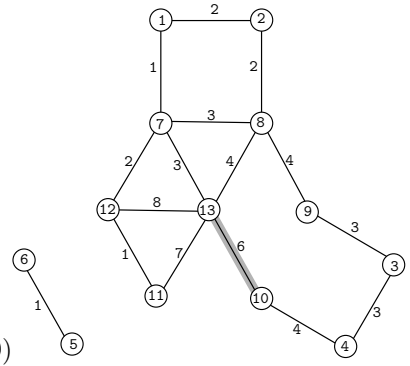
1	2,2	7,1				
2	1,2	8,2				
3	4,3	9,3				
4	3,3	10,4				
5	6,1					
6	5,1					
7	1,1	8,3	12,2	13,3		
8	2,2	7,3	9,4	13,4		
9	8,4	13,5	3,3			
10	4,4	11,6	13,6			
11	10,6	12,1	13,7			
12	7,2	11,1	13,8			
13	7,3	8,4	9,5	10,6	11,7	12,8

18)



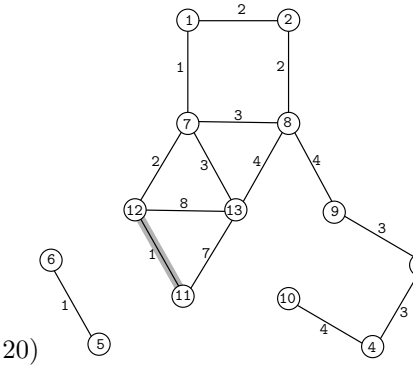
1	2,2	7,1				
2	1,2	8,2				
3	4,3	9,3				
4	3,3	10,4				
5	6,1					
6	5,1					
7	1,1	8,3	12,2	13,3		
8	2,2	7,3	9,4	13,4		
9	8,4	3,3				
10	4,4	11,6	13,6			
11	10,6	12,1	13,7			
12	7,2	11,1	13,8			
13	7,3	8,4	10,6	11,7	12,8	

19)



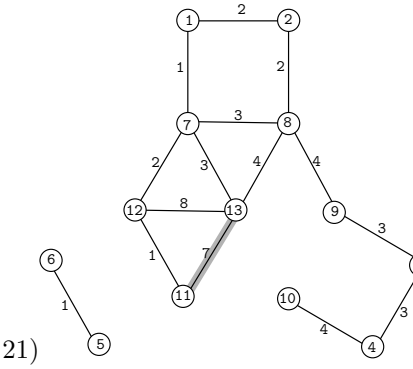
1	2,2	7,1				
2	1,2	8,2				
3	4,3	9,3				
4	3,3	10,4				
5	6,1					
6	5,1					
7	1,1	8,3	12,2	13,3		
8	2,2	7,3	9,4	13,4		
9	8,4	3,3				
10	4,4	13,6				
11	12,1	13,7				
12	7,2	11,1	13,8			
13	7,3	8,4	10,6	11,7	12,8	

20)



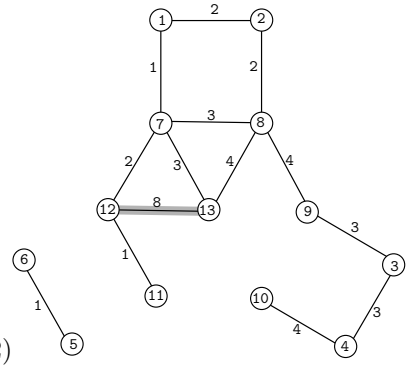
1	2,2	7,1				
2	1,2	8,2				
3	4,3	9,3				
4	3,3	10,4				
5	6,1					
6	5,1					
7	1,1	8,3	12,2	13,3		
8	2,2	7,3	9,4	13,4		
9	8,4	3,3				
10	4,4					
11	12,1	13,7				
12	7,2	11,1	13,8			
13	7,3	8,4	11,7	12,8		

21)

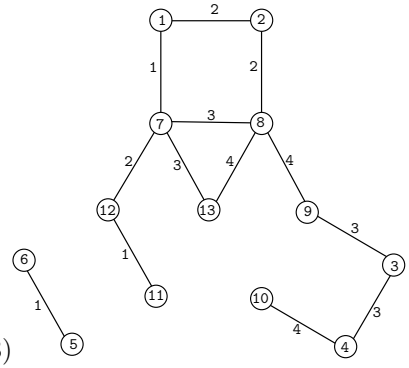


1	2,2	7,1				
2	1,2	8,2				
3	4,3	9,3				
4	3,3	10,4				
5	6,1					
6	5,1					
7	1,1	8,3	12,2	13,3		
8	2,2	7,3	9,4	13,4		
9	8,4	3,3				
10	4,4					
11	12,1	13,7				
12	7,2	11,1	13,8			
13	7,3	8,4	11,7	12,8		

22)



23)



En este caso, la subgráfica resultante no es conexa, así que se devuelve 0.

c)

El problema consiste en decidir si existe una subgráfica generadora H de G, conexa, tal que todas las aristas de H tienen peso menor o igual a b. Dicho de otra forma, una subgráfica generadora H de G que cumpla esas propiedades no puede tener ninguna arista cuyo peso sea mayor que b.

Lo que hace el algoritmo es borrar de G todas las aristas de peso mayor a b y después comprobar si la gráfica resultante es conexa. Digamos que G' es el resultado de elmininar todas esas aristas. Si G' es conexa entonces se devuelve 1, pues G' cumple con las propiedades de: generar a G, ser conexa y tener solo aristas de peso menor o igual a b. De hecho, cualquier subgráfica generadora y conexa de G' cumple con esa propiedad.

Si G' no es conexa el algoritmo devuelve 0. Nótese que G' es una subgráfica generadora maximal con la propiedad de tener aristas de peso a lo más b. Así que cualquier subgráfica generadora de G que sólo tenga aristas de peso a lo más b será subgráfica de G', y como G' no es conexa, entonces cualquier subgráfica generadora de G' tampoco lo será. Por lo tanto, no puede existir una subgráfica generadora de G, conexa, que tenga solo aristas de peso menor o igual a b, y por lo tanto el algoritmo devuelve el resultado correcto. ■

d)

- Recorrer todas las listas de adyacencias tiene complejidad $O(n + m)$.

- Si se utilizan listas ligadas, la operación de eliminar (DELETE()) del iterador toma tiempo $O(1)$. En el peor caso se realizarán $O(n+m)$ eliminaciones.
- La función EsConexa toma tiempo $O(n+m)$, pues ejecutar DFS en una gráfica en su representación por listas de adyacencias toma tiempo $O(n+m)$ y ver si todos los vértices fueron visitados toma tiempo $O(n)$.

Así, el algoritmo toma tiempo total $O(n+m)$.