Capítulo 2
Grafos y algoritmos

1. Conceptos básicos de la teoría de grafos.

En este capítulo veremos algoritmos para resolver algunos problemas de optimización en la teoría de grafos. Comencemos por dar algunas definiciones que necesitaremos luego.

Definición 1.1. Un *grafo* es un par (V, E) donde V es un conjunto finito y los elementos de E son pares de elementos distintos de V. Llamaremos *vértices* o *nodos* a los elementos de V y *ramas*, *arcos* o también *flechas* a los elementos de E.
Cuando nos importe el orden en las ramas, los elementos de E serán pares ordenados y diremos que el grafo es *dirigido*. Cuando no nos importe el orden, los elementos de E serán pares no ordenados y diremos que el grafo es *no dirigido*. En ambos casos usaremos la notación (v, w) para indicar una rama con la convención de que si el grafo es dirigido nos estamos refiriendo al par ordenado (en cuyo caso (v, w) y (w, v) denotan ramas distintas) y si es no dirigido nos estamos refiriendo al par no ordenado (en cuyo caso (v, w) y (w, v) denotan la misma rama).

Definición 1.2. Sea $G = (V, E)$ un grafo. Dados $v, w \in V$ diremos que una sucesión $\mathcal{C} = (e_1, \ldots, e_n)$ de elementos de E es un *camino* en G de v a w si $e_1 = (v, w)$ o $e_1 = (w, v)$ cuando $n = 1$, o si $e_i \neq e_{i+1}$ para todo $1 \leq i \leq n-1$ y existen $v_1, \ldots v_{n-1} \in V$ tales que $e_1 = (v, v_1)$ o $e_1 = (v_1, v)$, $e_n = (v_{n-1}, w)$ o $e_n = (w, v_{n-1})$ y $e_i = (v_{i-1}, v_i)$ o $e_i = (v_i, v_{i-1})$ para todo i tal que $2 \leq i \leq n-1$ cuando $n > 1$. En tal caso diremos que $v, v_1, \ldots, v_{n-1}$ y w son los vértices del camino $\mathcal{C}$.

Si $v, v_1, \ldots, v_{n-1}$ y w son todos distintos diremos que el camino es *simple*. Si $v = w$ y $v, v_1, \ldots, v_{n-1}$ son todos distintos diremos que el camino es un *ciclo* o también que es un *circuito*.

Cuando el grafo es dirigido y $e_1 = (v, v_1)$, $e_n = (v_{n-1}, w)$ y $e_i = (v_{i-1}, v_i)$ para todo i tal que $2 \leq i \leq n-1$ si $n > 1$, o cuando $e_1 = (v, w)$ si $n = 1$ diremos que $\mathcal{C}$ es un *camino dirigido* de v a w. Es decir, en un grafo dirigido tenemos los conceptos de camino y camino dirigido. Análogamente se definen los conceptos de *camino dirigido simple* y *ciclo dirigido*.

Diremos que un grafo G es *acíclico* si no existe ningún ciclo (dirigido o no) en G.

Definición 1.3. Diremos que un grafo $G = (V, E)$ es *conexo* si para todo par de vértices $u, v \in V$, $u \neq v$, existe un camino en G de u a v.
Diremos que el grafo es *fuertemente conexo* si para todo par de vértices $u, v \in V$, $u \neq v$, existe un camino dirigido en G de u a v. Observemos que este concepto sólo tiene sentido en el caso de un grafo dirigido.

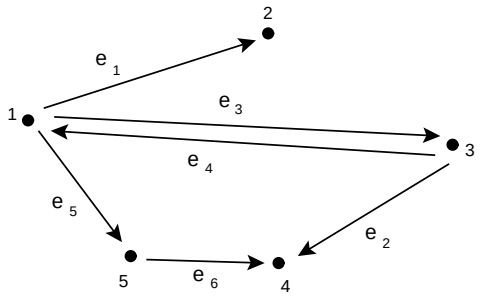
Definición 1.4. Un grafo $G = (V, E)$ se dice *completo* si para todo par de vértices $u, v \in V$, $u \neq v$, vale que $(u, v) \in E$.

Observación 1.5. Si $G = (V, E)$ es un grafo completo y $m = \#V$ entonces

$$\#E = \begin{cases} \binom{m}{2} = \frac{m(m-1)}{2} & \text{si } G \text{ es no dirigido} \\ \binom{m}{2}.2! = m(m-1) & \text{si } G \text{ es dirigido} \end{cases}$$

Antes de continuar veamos algunos ejemplos.

Ejemplo 1.6. Consideremos el siguiente grafo dirigido $G = (V, E)$, con vértices 1, 2, 3, 4 y 5 y ramas $e_1 = (1, 2)$, $e_2 = (3, 4)$, $e_3 = (1, 3)$, $e_4 = (3, 1)$, $e_5 = (1, 5)$ y $e_6 = (5, 4)$, es decir $V = \{1, 2, 3, 4, 5\}$ y $E = \{(1, 2), (3, 4), (1, 3), (3, 1), (1, 5), (5, 4)\}$, al que representaremos gráficamente en la forma

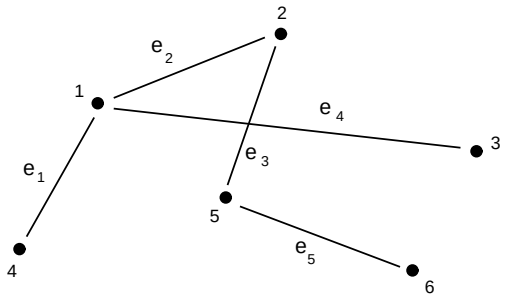


Este es un grafo dirigido, conexo, pero no fuertemente conexo (no hay un camino dirigido de 2 a 3).

La sucesión (e_1, e_3) es un camino simple de 2 a 3. Este camino no es dirigido. El grafo no es acíclico: la sucesión (e_5, e_6, e_2, e_3) es un ciclo. Este ciclo no es un ciclo dirigido.
La sucesión (e_4, e_5, e_6) es un camino dirigido simple de 3 a 4. La sucesión (e_3, e_4) es un ciclo dirigido. La sucesión $(e_1, e_4, e_2, e_6, e_5)$ es un camino de 2 a 1. Este camino no es dirigido ni simple.

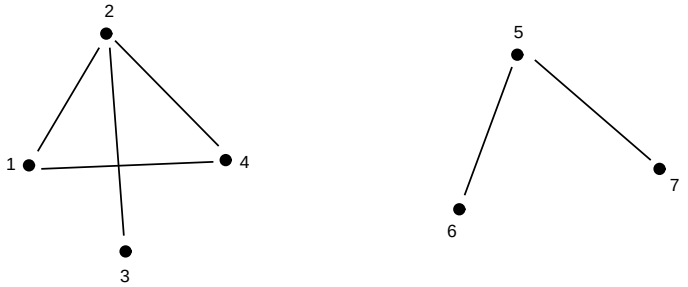
El grafo no es completo: $(2, 1) \notin E$.

Ejemplo 1.7. Consideremos el grafo no dirigido $G = (V, E)$, con vértices 1, 2, 3, 4, 5 y 6 y ramas $e_1 = (1, 4)$, $e_2 = (1, 2)$, $e_3 = (2, 5)$, $e_4 = (1, 3)$ y $e_5 = (5, 6)$ $V = \{1, 2, 3, 4, 5, 6\}$ y $E = \{(1, 4), (1, 2), (2, 5), (1, 3), (5, 6)\}$, al que representaremos gráficamente en la forma



Este es un grafo no dirigido, conexo y acíclico. No es completo: $(4, 5) \notin E$. La sucesión (e_1, e_2, e_3, e_5) es un camino simple de 4 a 6.

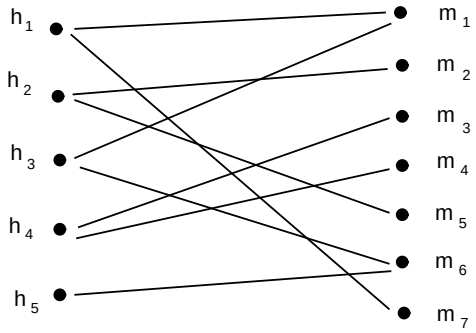
Ejemplo 1.8. Consideremos el grafo



Este es un grafo no dirigido. No es conexo (no existe ningún camino que una 1 y 7) sino que tiene dos componentes conexas, una conteniendo un ciclo (el ciclo $(1, 2), (2, 4), (4, 1)$) y otra acíclica.

Definición 1.9 Diremos que $G = (V, E)$ es un *grafo bipartito* si existen dos conjuntos disjuntos P y Q tales que $V = P \cup Q$ y toda rama $e \in E$ tiene un extremo en P y el otro extremo en Q.

Ejemplo 1.10. El grafo



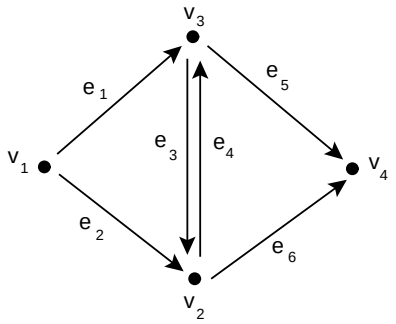
es bipartito. En este caso $P = \{h_1, h_2, h_3, h_4, h_5\}$ y $Q = \{m_1, m_2, m_3, m_4, m_5, m_6, m_7\}$.

Definición 1.11. Sea $G = (V, E)$ un grafo dirigido. Si $(u, v) \in E$ diremos que u es la *cola* y que v es la *punta* de la flecha (u, v).

A cada grafo dirigido $G = (V, E)$ le podemos asociar una matriz que contiene toda la información sobre el grafo. Esta matriz se llama la *matriz de incidencia vértice-rama* de G.
Si $V = \{v_1, \ldots, v_m\}$ y $E = \{e_1, \ldots, e_n\}$, la matriz de incidencia vértice-rama de G es la matriz $||a_{ij}|| \in \mathbb{R}^{m \times n}$, definida por

$$a_{ij} = \begin{cases} 1 & \text{si } v_i \text{ es la cola de } e_j \\ -1 & \text{si } v_i \text{ es la punta de } e_j \\ 0 & \text{en otro caso} \end{cases}$$

Ejemplo 1.12. Dado el grafo



su matriz de incidencia vértice-rama es la matriz

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 1 & 0 & 1 \\ -1 & 0 & 1 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 \end{pmatrix}$$

Observación 1.13. La matriz de incidencia vértice-rama de un grafo G tiene, en cada columna, un 1, un -1 y el resto de los coeficientes nulos. Esto se debe a que cada rama tiene una sola cola y una sola punta. Luego, en una matriz de incidencia vértice-rama la suma de todas las filas es cero.

2. Forestas, árboles y hojas.

Definición 2.1. Una *foresta* es un grafo acíclico. Un *árbol* es un grafo acíclico conexo, es decir, una componente conexa de una foresta.

Definición 2.2. Sea $G = (V, E)$ un grafo y sea $u \in V$. Definimos

$$i(u) = \#\{v \in V \,/\, (v, u) \in E\} \qquad \textit{indegree} \text{ de } u$$

$$o(u) = \#\{v \in V \,/\, (u, v) \in E\} \qquad \textit{outdegree} \text{ de } u$$

y definimos el grado de u como

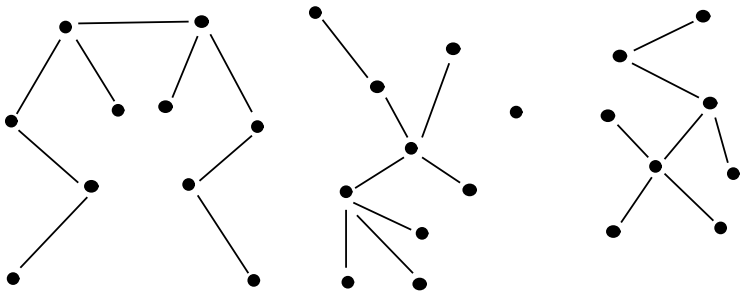
$$deg\ (u) = \begin{cases} i(u) + o(u) & \text{si } G \text{ es dirigido} \\ i(u) & \text{si } G \text{ es no dirigido} \end{cases}$$

Tanto en el caso dirigido como no dirigido resulta que $deg\ (u)$ es el número de ramas que inciden en el vértice u. (Observemos que si G es no dirigido entonces los conjuntos que definen $i(u)$ y $o(u)$ son iguales y por eso no los contamos dos veces).

Definición 2.3. Sea $G = (V, E)$ un árbol. Diremos que $u \in V$ es una *hoja* sii $deg\ (u) \leq 1$, es decir, una hoja es un vértice en el que incide a lo sumo una rama.

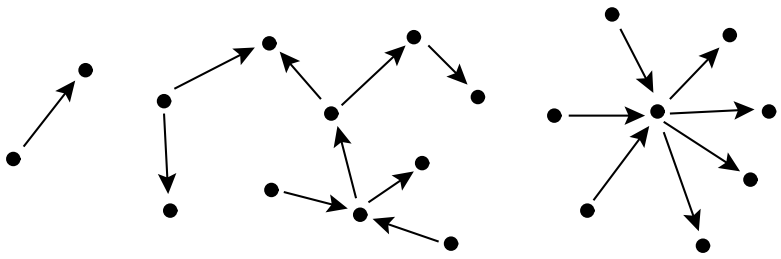
Observación 2.4. Si G es un árbol con un único vértice (y ninguna rama) entonces ese vértice es una hoja. En cualquier otro caso, un árbol siempre tiene por lo menos dos hojas.

Ejemplo 2.5. El grafo no dirigido



es una foresta compuesta por cuatro árboles. El primero tiene cuatro hojas, el segundo seis, el tercero una y el cuarto cinco.

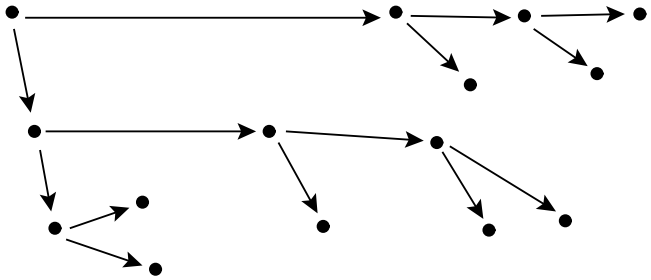
Ejemplo 2.6. El grafo dirigido



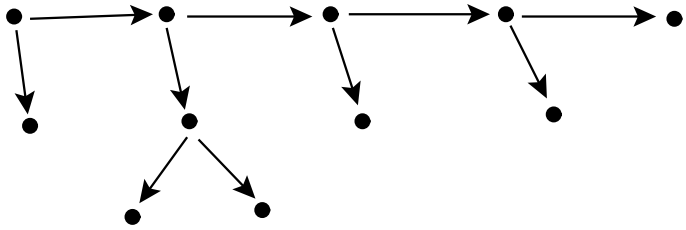
es una foresta compuesta por tres árboles con dos, cinco y siete hojas cada uno respectivamente.

Definición 2.7. Un *árbol binario* es un árbol dirigido $G = (V, E)$ tal que $o(u) = 2$ para todo $u \in V$ que no sea una hoja.

Ejemplo 2.8. El grafo dirigido



es un árbol binario. También lo es el grafo

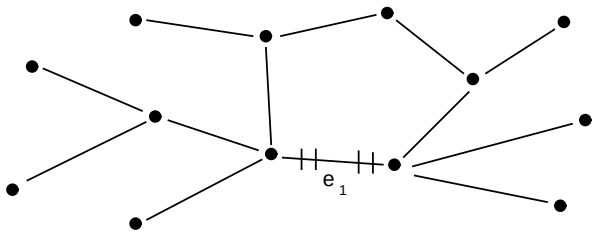


Proposición 2.9. Si un árbol tiene n ramas entonces tiene $n+1$ vértices.

Demostración: Por inducción en n.
El caso $n = 1$ es trivial. Supongamos que la proposición vale para todo árbol con n ramas y sea G un árbol con $n+1$ ramas. Sea u una hoja de G y sea e la única rama de G que incide en u (es decir, $e = (w,u) \in E$ si $i(u) = 1$ y $\{v \in V \,/\, (v,u) \in E\} = \{(w,u)\}$ o bien $e = (u,w) \in E$ si $o(u) = 1$ y $\{v \in V \,/\, (u,v) \in E\} = \{(u,w)\}$. Sea $G' = (V', E')$, donde $V' = V - \{u\}$ y $E' = E - \{e\}$. Entonces G' es un árbol con n ramas. Luego, por hipótesis inductiva, $\#V' = n+1$, de donde $\#V = n+2$. □

Proposición 2.10. Si G es un grafo conexo con m vértices y $m-1$ ramas entonces G es un árbol.

Demostración: Sólo debemos ver que G es acíclico.
Supongamos que G tuviera un ciclo. Entonces, como se ve en el dibujo, podríamos quitar una rama e_1 del ciclo de manera que $G' = (V, E - \{e_1\})$ siguiera siendo conexo y tuviera un ciclo menos que G.

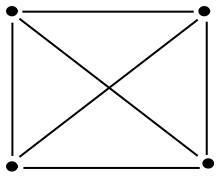


Si G' no fuera acíclico podríamos repetir el procedimiento. Así siguiendo, luego de quitar r ramas obtenemos un grafo conexo y acíclico $(V, E - \{e_1, \ldots, e_r\})$. Pero este árbol tendría m vértices y $m-1-r < m-1$ ramas. Absurdo, pues esto contradice la proposición 2.9. □

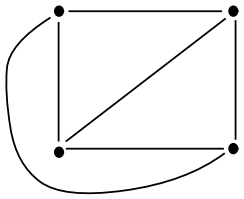
3. Grafos planares.

Definición 3.1. Un grafo no dirigido G se dice *planar* si se lo puede representar en el plano de manera tal que sus ramas sólo se corten en los vértices.

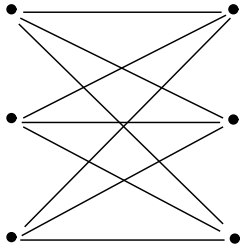
Ejemplo 3.2. El grafo



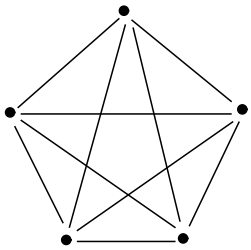
es planar ya que se puede representar en la forma



Ejemplo 3.3. No son planares
i) el grafo bipartito completo $K_{3,3}$ de 3×3

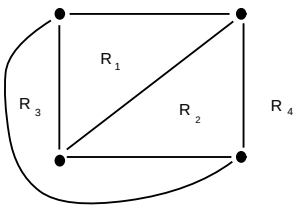


ii) el grafo completo K_5 de cinco vértices



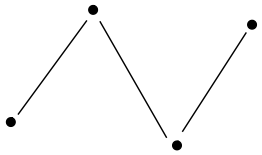
Un grafo planar divide al plano en regiones conexas a las que llamaremos *caras*. Dado un grafo planar, siempre una de sus caras es no acotada.

Ejemplo 3.4. El grafo



divide al plano en cuatro regiones $R_1, \ldots, R_4$. La cara R_4 no es acotada.

Ejemplo 3.5. El grafo



tiene una sola cara.

Un resultado clásico sobre el número de caras, vértices y ramas de un grafo es el siguiente.

Teorema 3.6. (Euler) Sean V, E y F los conjuntos de vértices, ramas y caras de un grafo planar conexo. Entonces $\#V - \#E + \#F = 2$.

Demostración: Por inducción en $\#F$. Si $\#F = 1$ tenemos un grafo conexo con una sola cara, es decir, un árbol. En este caso, $\#E = \#V - 1$. Luego $\#V - \#E + 1 = 2$ como queríamos ver.
Supongamos ahora que $\#F = n + 1$ y que el teorema vale para todo grafo con n caras. Sea e una rama que separa dos caras. Entonces el grafo que resulta de quitar esa rama es un grafo planar conexo que tiene una cara menos, la misma cantidad de vértices y una rama menos. Entonces, por hipótesis inductiva $\#V - (\#E - 1) + (\#F - 1) = 2$. Por lo tanto $\#V - \#E + \#F = 2$. □

Recordemos que si un grafo $G = (V, E)$ es completo y $m = \#V$ entonces

$$\#E = \begin{cases} \binom{m}{2} = \frac{m(m-1)}{2} & \text{si } G \text{ es no dirigido} \\ \binom{m}{2}.2! = m(m-1) & \text{si } G \text{ es dirigido} \end{cases}$$

En cambio, en un grafo planar las ramas son relativamente escasas como lo muestra la siguiente proposición.

Proposición 3.7. Un grafo planar conexo con $m \geq 3$ vértices tiene a lo sumo 3(m-2) ramas.

Demostración: Cada cara del grafo está delimitada por un circuito que tiene por lo menos tres ramas (salvo el caso $\#E = 2$ en cuyo caso debe ser $m = 3$ y vale trivialmente lo que queremos probar).
Para cada cara R_i sea x_i la cantidad de ramas que son un borde de R_i. Entonces

$$\sum_{i=1}^{\#F} x_i \geq 3\#F$$

Por otra parte, como cada rama es el borde de a lo sumo dos caras entonces

$$\sum_{i=1}^{\#F} x_i \leq 2\#E$$

Luego, $2\#E \geq 3\#F$. Ahora, aplicando el teorema 3.6. de Euler, se tiene que

$$2\#E \geq 3(2 + \#E - \#V) = 6 + 3\#E - 3\#V$$

de donde $\#E \leq 3\#V - 6 = 3(\#V - 2) = 3(m - 2)$ como habíamos afirmado. □

4. Buscando un camino.

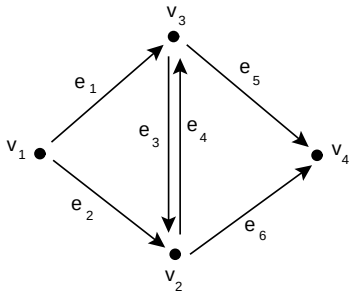
Dado un grafo $G = (V, E)$ y dado un vértice s de G, vamos a describir un algoritmo que, para cada $t \in V$ tal que $t \neq s$, encuentra (cuando existe) un camino en G de s a t. Describiremos dos maneras de hacer esto,

llamadas *breadth-first search* y *depth-first search*. Para poder aplicar estos algoritmos necesitaremos conocer cierta información sobre el grafo dada en lo que llamaremos la *tabla de adyacencia* de G.

Definición 4.1. Sea $G = (V, E)$ un grafo. Dados $u, v \in V$ diremos que u y v son *adyacentes* si $(u, v) \in E$ o $(v, u) \in E$.

Dado un grafo $G = (V, E)$, la tabla de adyacencia de G es una tabla que contiene, para cada vértice u, el conjunto $A(u) = \{v \in V \,/\, u \text{ y } v \text{ son adyacentes } \}$.

Ejemplo 4.2. La tabla de adyacencia del grafo



es la tabla

u	$A(u)$
v_1	v_3, v_2
v_2	v_1, v_3, v_4
v_3	v_1, v_2, v_4
v_4	v_2, v_3

Descripción del algoritmo breadth-first search.
1. Hacer una lista de los elementos de V.
2. Marcar s en la lista, poner $Q = \{s\}$ y $p(u) = -1$ para cada $u \in V$.
3. Si $u \in Q$ es, de todos los elementos el primero que ingresó a Q, $Q = Q - \{u\}$.
4. Para cada $v \in A(u)$ que no esté marcado en la lista, marcarlo y reemplazar Q por $Q \cup \{v\}$ y $p(v)$ por u. Llevar un registro del orden en que los elementos ingresan a Q.
5. Si $Q = \emptyset$ STOP.
6. goto 3.

Al terminar el algoritmo se tiene un subconjunto de vértices que están marcados. Dado $t \neq s$ se verifica que existe un camino de s a t si y sólo si t está marcado y en tal caso, $p(t)$ es el nodo anteriror a t en ese camino. De esta manera, el camino a t hallado por el algoritmo puede reconstruírse usando el predecesor $p(v)$ de cada nodo v de ese camino.
El criterio usado para extraer cada vértice u de Q en este algoritmo es "first in, first out". En este caso decimos que Q es una *cola* (queue) y la búsqueda se denomina *breadth-first search*. Si, en cambio, utilizamos el criterio "last in, first out" para extraer los elementos de Q entonces la búsqueda se llama *depth-first search* y Q se dice una *pila* (stack). Es decir, el algoritmo depth-first search resulta de cambiar en el algoritmo anterior el paso 3. por

3'. Si $u \in Q$ es, de todos los elementos de el último que ingresó a Q, $Q = Q - \{u\}$.

Observación 4.3. El grafo G puede ser dirigido o no dirigido. Si el grafo es dirigido el camino no necesariamente es dirigido. Si quisiéramos resolver el problema de determinar (cuando existe) un camino dirigido

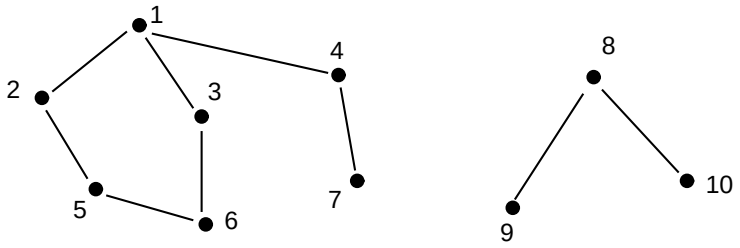
de s a t deberemos utilizar, en lugar de la tabla de adyacencia, una tabla que contenga, para cada vértice u, el conjunto

$$A'(u) = \{v \in V \,/\, (u, v) \in E\}$$

Dejamos a cargo del lector verificar que, en ambos casos, los caminos hallados por el algoritmo search son simples.

A continuación aplicaremos estos algoritmos en un par de ejemplos. En el primer caso aplicaremos el algoritmo breadth-first search a un grafo no dirigido y usaremos la tabla de adyacencia del grafo para encontrar, fijado un vértice s, un camino de s a t, para aquellos t tales que existe un tal camino. En el segundo, aplicaremos el algoritmo depth-first search a un grafo dirigido y encontraremos, para un determinado s, un camino dirigido de s a t, para aquellos t tales que existe un tal camino. En este caso usaremos, en lugar de la tabla de adyacencia, la tabla que contiene, para cada u, el conjunto $A'(u)$.

Ejemplo 4.4. Dado el grafo



su tabla de adyacencia es

u	$A(u)$
1	$2, 3, 4$
2	$1, 5$
3	$1, 6$
4	$1, 7$
5	$2, 6$
6	$3, 5$
7	4
8	$9, 10$
9	8
10	8

Sea $s = 1$. Determinaremos, usando breadth-first search, para cuáles t existe un camino de s a t. Reconstruiremos ese camino para uno de esos valores de t y dejaremos a cargo del lector la reconstrucción de los caminos a los restantes t hallados.

1. $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
2. $V = \{1^*, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, $Q = \{1\}$, $p = (p(1), p(2), \ldots, p(10)) = (-1, -1, \ldots, -1)$

Primera iteración

3. $u = 1$, $Q = Q - \{1\} = \emptyset$
4. $A(u) = \{2, 3, 4\}$, marcamos 2, 3 y 4, los agregamos a la cola y reemplazamos $p(2)$, $p(3)$ y $p(4)$ por 1. El estado actual de la lista y de la cola es
$V = \{1^*, 2^*, 3^*, 4^*, 5, 6, 7, 8, 9, 10\}$ y $Q = \{2, 3, 4\}$.
El valor actual del vector p es $(-1, 1, 1, 1, -1, -1, -1, -1, -1, -1)$
5. $Q \neq \emptyset$

6. volvemos a 3.

Segunda iteración

3. $u = 2$, $Q = Q - \{2\} = \{3, 4\}$
4. $A(u) = \{1, 5\}$, marcamos el 5, lo agregamos a la cola y reemplazamos $p(5)$ por 2. El estado actual de la lista y de la cola es $V = \{1^*, 2^*, 3^*, 4^*, 5^*, 6, 7, 8, 9, 10\}$ y $Q = \{3, 4, 5\}$.
El valor actual del vector p es $(-1, 1, 1, 1, 2, -1, -1, -1, -1, -1)$
5. $Q \neq \emptyset$
6. volvemos a 3.

Tercera iteración

3. $u = 3$, $Q = Q - \{3\} = \{4, 5\}$
4. $A(u) = \{1, 6\}$, marcamos el 6, lo agregamos a la cola y reemplazamos $p(6)$ por 3. El estado actual de la lista y de la cola es $V = \{1^*, 2^*, 3^*, 4^*, 5^*, 6^*, 7, 8, 9, 10\}$ y $Q = \{4, 5, 6\}$.
El valor actual del vector p es $(-1, 1, 1, 1, 2, 3, -1, -1, -1, -1)$
5. $Q \neq \emptyset$
6. volvemos a 3.

Cuarta iteración

3. $u = 4$, $Q = Q - \{4\} = \{5, 6\}$
4. $A(u) = \{1, 7\}$, marcamos el 7, lo agregamos a la cola y reemplazamos $p(7)$ por 4. El estado actual de la lista y de la cola es
$V = \{1^*, 2^*, 3^*, 4^*, 5^*, 6^*, 7^*, 8, 9, 10\}$ y $Q = \{5, 6, 7\}$.
El valor actual del vector p es $(-1, 1, 1, 1, 2, 3, 4, -1, -1, -1)$
5. $Q \neq \emptyset$
6. volvemos a 3.

Quinta iteración

3. $u = 5$, $Q = Q - \{5\} = \{6, 7\}$
4. $A(u) = \{2, 6\}$, no marcamos ningún vértice porque el 2 y el 6 ya están marcados. El estado actual de la lista y de la cola es
$V = \{1^*, 2^*, 3^*, 4^*, 5^*, 6^*, 7^*, 8, 9, 10\}$ y $Q = \{6, 7\}$.
El valor actual del vector p es $(-1, 1, 1, 1, 2, 3, 4, -1, -1, -1)$
5. $Q \neq \emptyset$
6. volvemos a 3.

Sexta iteración

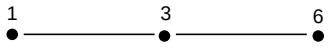
3. $u = 6$, $Q = Q - \{6\} = \{7\}$
4. $A(u) = \{3, 5\}$, no marcamos ningún vértice porque el 3 y el 5 ya están marcados. El estado actual de la lista y de la cola es
$V = \{1^*, 2^*, 3^*, 4^*, 5^*, 6^*, 7^*, 8, 9, 10\}$ y $Q = \{7\}$.
El valor actual del vector p es $(-1, 1, 1, 1, 2, 3, 4, -1, -1, -1)$
5. $Q \neq \emptyset$
6. volvemos a 3.

Séptima iteración

3. $u = 7$, $Q = Q - \{7\} = \emptyset$
4. $A(u) = \{4\}$, no marcamos ningún vértice porque el 4 ya está marcado. El estado actual de la lista y de la cola es
$V = \{1^*, 2^*, 3^*, 4^*, 5^*, 6^*, 7^*, 8, 9, 10\}$ y $Q = \emptyset$.

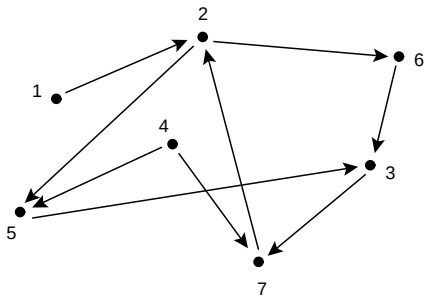
El valor actual del vector p es $(-1, 1, 1, 1, 2, 3, 4, -1, -1, -1)$
5. $Q = \emptyset$. STOP

Dado $t \neq 1$, existe un camino de 1 a t sii $t = 2, 3, 4, 5, 6, 7$.
Como $p(6) = 3$ y $p(3) = 1$ entonces el camino de $s = 1$ a $t = 6$ hallado por el algoritmo es



Dejamos a cargo del lector la reconstrucción de los caminos a los restantes t hallados.

Ejemplo 4.5. Dado el grafo dirigido



construyamos la tabla

u	$A'(u)$
1	2
2	5, 6
3	7
4	5, 7
5	3
6	3
7	2

Sea $s = 1$. Determinaremos, usando depdth-first search, para cuáles t existe un camino dirigido de s a t. Reconstruiremos ese camino para uno de esos valores de t y dejaremos a cargo del lector la reconstrucción de los caminos a los restantes t hallados.

1. $V = \{1, 2, 3, 4, 5, 6, 7\}$
2. $V = \{1^*, 2, 3, 4, 5, 6, 7\}$, $Q = \{1\}$, $p = (p(1), p(2), \ldots, p(7)) = (-1, -1, \ldots, -1)$

Primera iteración

3. $u = 1$, $Q = Q - \{1\} = \emptyset$
4. $A'(u) = \{2\}$, marcamos 2, lo ponemos en la pila y reemplazamos $p(2)$ por 1. El estado actual de la lista y de la pila es
$V = \{1^*, 2^*, 3, 4, 5, 6, 7\}$ y $Q = \{2\}$.
El valor actual del vector p es $(-1, 1, -1, -1, -1, -1, -1)$
5. $Q \neq \emptyset$
6. volvemos a 3.

Segunda iteración

3. $u = 2$, $Q = Q - \{2\} = \emptyset$
4. $A'(u) = \{5, 6\}$, marcamos 5, 6, los ponemos en la pila y reemplazamos $p(5)$ y $p(6)$ por 2. El estado actual de la lista y de la pila es
$V = \{1^*, 2^*, 3, 4, 5^*, 6^*, 7\}$ y $Q = \{5, 6\}$

El valor actual del vector p es $(-1, 1, -1, -1, 2, 2, -1)$
5. $Q \neq \emptyset$
6. volvemos a 3.

Tercera iteración

3. $u = 6$, $Q = Q - \{6\} = \{5\}$
4. $A'(u) = \{3\}$, marcamos el 3, lo ponemos en la pila y reemplazamos $p(3)$ por 6. El estado actual de la lista y de la pila es
$V = \{1^*, 2^*, 3^*, 4, 5^*, 6^*, 7\}$ y $Q = \{5, 3\}$
El valor actual del vector p es $(-1, 1, 6, -1, 2, 2, -1)$
5. $Q \neq \emptyset$
6. volvemos a 3.

Cuarta iteración

3. $u = 3$, $Q = Q - \{3\} = \{5\}$
4. $A'(u) = \{7\}$, marcamos 7, lo ponemos en la pila y reemplazamos $p(7)$ por 3. El estado actual de la lista y de la pila es
$V = \{1^*, 2^*, 3^*, 4, 5^*, 6^*, 7^*\}$ y $Q = \{5, 7\}$
El valor actual del vector p es $(-1, 1, 6, -1, 2, 2, 3)$
5. $Q \neq \emptyset$
6. volvemos a 3.

Quinta iteración

3. $u = 7$, $Q = Q - \{7\} = \{5\}$
4. $A'(u) = \{2\}$, no marcamos ningún vértice porque 2 ya está marcado. El estado actual de la lista y de la pila es
$V = \{1^*, 2^*, 3^*, 4, 5^*, 6^*, 7^*\}$ y $Q = \{5\}$
El valor actual del vector p es $(-1, 1, 6, -1, 2, 2, 3)$.
5. $Q \neq \emptyset$
6. volvemos a 3.

Sexta iteración

3. $u = 5$, $Q = Q - \{5\} = \emptyset$
4. $A'(u) = \{3\}$, no marcamos ningún vértice porque 3 ya está marcado. El estado actual de la lista y de la pila es
$V = \{1^*, 2^*, 3^*, 4, 5^*, 6^*, 7^*\}$ y $Q = \emptyset$
El valor actual del vector p es $(-1, 1, 6, -1, 2, 2, 3)$.
5. $Q = \emptyset$. STOP.

Dado $t \neq 1$, existe un camino dirigido de 1 a t sii $t = 2, 3, 5, 6, 7$.
Como $p(7) = 3$, $p(3) = 6$, $p(6) = 2$ y $p(2) = 1$ entonces el camino de $s = 1$ a $t = 7$ hallado por el algoritmo es $1 \longrightarrow 2 \longrightarrow 6 \longrightarrow 3 \longrightarrow 7$.
Como antes, dejamos a cargo del lector la reconstrucción de los caminos a los restantes t hallados.

Complejidad del algoritmo.

Estimaremos ahora la complejidad del algoritmo search, es decir, la cantidad de operaciones que realiza el algoritmo para obtener el output a partir de los datos de entrada.
Sea $m = \#V$. El algoritmo realiza a lo sumo m iteraciones (pues cada vértice ingresa en Q a lo sumo una vez: al ingresar un vértice, se lo marca y los vértices que están marcados no se ingresan) y, en cada una de ellas, se realizan a lo sumo $c\,m$ operaciones, para una constante $c > 0$. Luego, el algoritmo realiza en total a

lo sumo $c\,m^2$ operaciones y por lo tanto la complejidad del algoritmo search es menor o igual que $c\,m^2$ para una constante c. Diremeos entonces que la complejidad es del orden de m^2 o también que es $O(m^2)$.

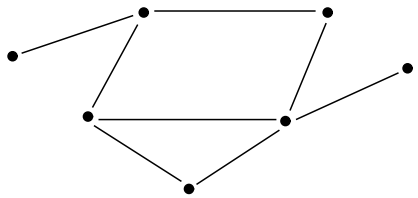
Definición 4.5. Dado un algoritmo, sea N el tamaño del input, es decir, la cantidad de bits que se necesitan para describir el input. Diremos que la complejidad del algoritmo es *polinomial* si es menor o igual que $P(N)$ para algún polinomio P.

En el caso del algoritmo search, se tiene que $m \leq N$, de donde $c\,m^2 \leq c.N^2$. Luego, la complejidad del algoritmo search es menor o igual que $P(N)$ donde P es el polinomio $c\,X^2$. Hemos probado entonces que la complejidad de este algorimto es polinomial.

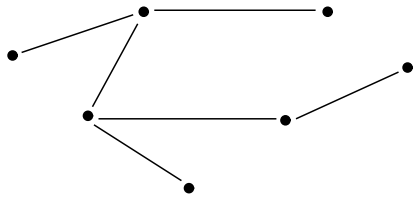
5. Spanning trees.

Definición 5.1. Sea $G = (V, E)$ un grafo conexo. Un *spanning tree* de G es un árbol $T = (V', E')$ tal que $V' = V$ y E' es un subconjunto de E.

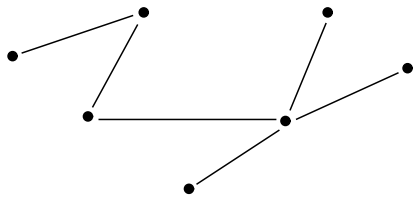
Ejemplo 5.2. Si G es el grafo



entonces el grafo



es un spanning tree de G. También lo es el grafo



Observación 5.3. Sea G un grafo conexo. Si G es un árbol, entonces G es un spanning tree de G y es el único posible. Pero si G no es un árbol, entonces contiene por lo menos un ciclo. Siguiendo la misma idea que en la proposición 2.10., podemos quitar una rama del ciclo de manera que el subgrafo resultante siga siendo conexo y tenga un ciclo menos. Si este no es un grafo acíclico, repetimos el procedimiento. De esta manera luego de un número finito de pasos obtenemos un subgrafo conexo acíclico de G. Como este grafo fue obtenido quitando sólo ramas y no vértices, entonces es un spanning tree de G. Esto muestra que cualquier grafo conexo tiene un spanning tree.

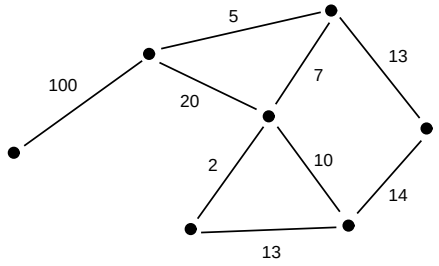
Observación 5.4. Sea $G = (V, E)$ un grafo conexo con m vértices. Si $G' = (V', E')$ es un subgrafo conexo y acíclico de G con $m - 1$ ramas entonces G' es un spannig tree de G. En efecto, como G' es un árbol con

$m-1$ ramas, entonces $\#V' = m$ por la proposición 2.9. Pero $V' \subseteq V$, $\#V' = m = \#V$ y $E' \subseteq E$. Luego $V' = V$ y $E' \subseteq E$.

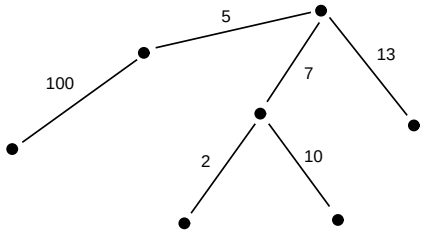
Sea G un grafo no dirigido y conexo donde a cada rama e de G se le ha asignado un *peso* $\omega(e)$.

Definición 5.5. Diremos que un spanning tree T de G es *mínimo* sii la suma de los pesos de sus ramas es menor o igual que la suma de los pesos de las ramas de cualquier otro spanning tree de G.

Ejemplo 5.6. Consideremos el siguiente grafo donde en cada rama indicamos el peso

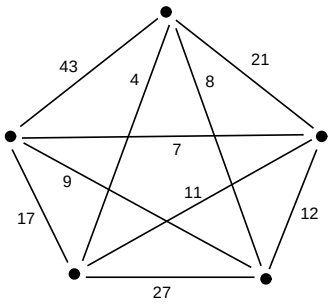


Entonces T dado por



es un spanning tree mínimo de G. La suma de los pesos de sus ramas es 137.

Ejemplo 5.7. Supongamos que deseamos interconectar cinco localidades con una red de caminos a costo mínimo. Supongamos conocido el costo de conectar las localidades i y j e ilustremos la situación con esos datos en un grafo G con pesos, donde a cada rama (i, j) le asignamos como peso el costo de conectar i y j.



Entonces la solución del problema consiste en hallar un spanning tree mínimo de G.

En las próximas dos secciones veremos un par de algoritmos que hallan, dado un grafo G no dirigido, conexo y con pesos, un spanning tree mínimo de G.

6. El algoritmo de Kruskal

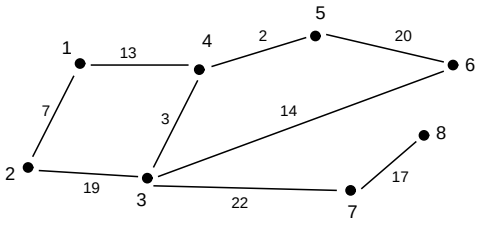
Sea $G = (V, E)$ un grafo no dirigido, conexo, donde cada rama $e \in E$ tiene asignado un peso $\omega(e)$. Sea

$m = \#V$. El siguiente algoritmo, conocido como el algoritmo de Kruskal encuentra, en tiempo polinomial, un spanning tree mínimo de G.

Descripción del algoritmo.

1. Ordenar las ramas de G de menor a mayor según el peso.
2. Elegir una rama de mínimo peso entre aquellas que todavía no fueron elegidas y que no forman un ciclo con las ya elegidas.
3. repetir 2. si el número de ramas elegidas es menor que $m - 1$.

Ejemplo 6.1. Apliquemos el algoritmo al grafo G con $m = 8$ vértices dado por

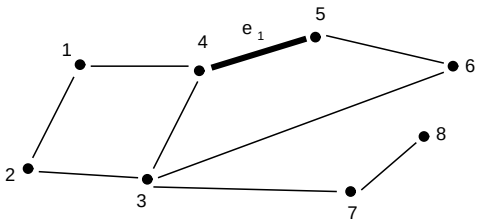


Para poder tener en claro en el gráfico cuál es el conjunto de ramas elegidas, dibujaremos las ramas que han sido elegidas hasta el momento con trazo grueso y las que no con trazo fino.

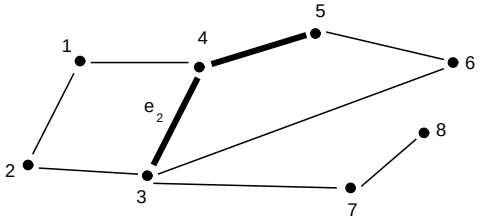
1. Ordenamos las ramas de menor a mayor según el peso tal como se indica en la tabla

e	$\omega(e)$
$e_1 = (4, 5)$	$\omega(e_1) = 2$
$e_2 = (4, 3)$	$\omega(e_2) = 3$
$e_3 = (1, 2)$	$\omega(e_3) = 7$
$e_4 = (1, 4)$	$\omega(e_4) = 13$
$e_5 = (3, 6)$	$\omega(e_5) = 14$
$e_6 = (7, 8)$	$\omega(e_6) = 17$
$e_7 = (2, 3)$	$\omega(e_7) = 19$
$e_8 = (5, 6)$	$\omega(e_8) = 20$
$e_9 = (3, 7)$	$\omega(e_9) = 22$

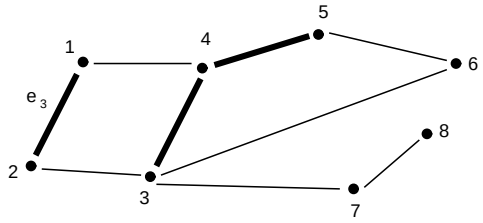
2. Elegimos e_1.



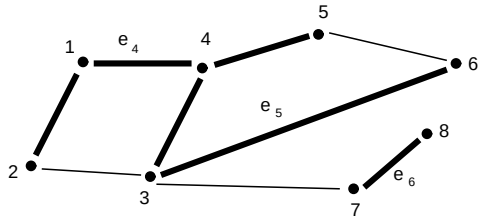
Como $\#\{$ ramas elegidas $\} = 1 < 8 - 1 = m - 1$, repetimos 2. Ahora elegimos e_2.



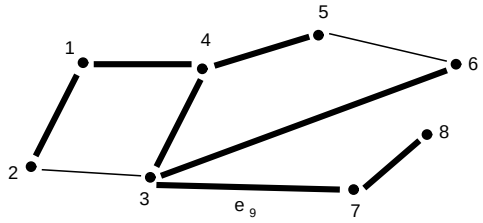
Como #{ ramas elegidas } $= 2 < 8 - 1 = m - 1$, repetimos 2.
Elegimos e_3 ya que no forma ciclo con las ya elegidas.



De la misma manera luego elegimos e_4, e_5 y e_6. Hasta ahora el gráfico es



Siendo que #{ ramas elegidas } $= 6 < 8 - 1 = m - 1$, repetimos 2. Como e_7 y e_8 forman ciclo con las ya elegidas y e_9 no, elegimos e_9.



Como #{ ramas elegidas } $= 7 = 8 - 1 = m - 1$, STOP.
El grafo determinado por las líneas gruesas es un spanning tree mínimo de G.

Para demostrar la validez del algoritmo necesitaremos probar antes un resultado, que también utilizaremos luego para probar la validez del algoritmo de Prim que veremos en la sección 7.

Definición 6.2. Sea $G = (V, E)$ un grafo. Dado un subconjunto A de V definimos el conjunto

$$\partial A = \{(u, v) \in E \; u \in A \text{ y } v \in V - A\}$$

El conjunto ∂A se llama el *corte* definido por A.

Observación 6.3. Si G es no dirigido entonces

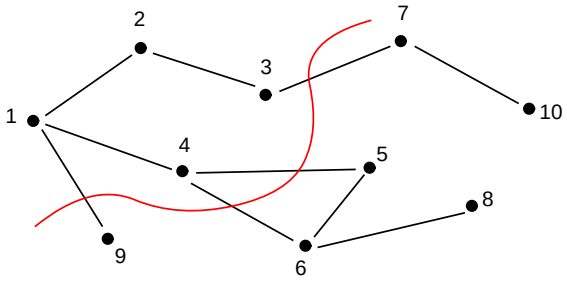
$$\partial A = \{e \in E \,/\, \text{ un vértice de } e \text{ pertenece a } A \text{ y el otro a } V - A\}$$

mientras que cuando G es dirigido

$$\partial A = \{e \in E \,/\, \text{ la cola de } e \text{ pertenece a } A \text{ y la punta de } e \text{ pertenece a } V - A\}$$

Definición 6.4. Si ∂A es un corte de G y U es un subconjunto de ramas de G, diremos que U *no cruza al corte* ∂A si $U \cap \partial A = \emptyset$.

Ejemplo 6.5. El conjunto $A = \{1, 2, 3, 4\}$ define un corte en el grafo

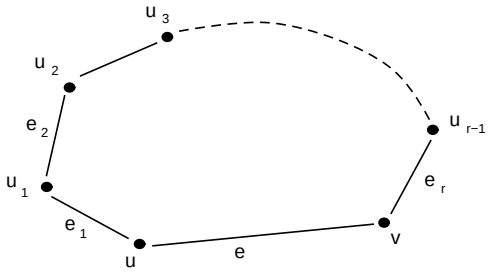


En la figura, $\partial A = \{(3, 7), (4, 5), (4, 6), (1, 9)\}$ es el conjunto de ramas intersecadas por la curva. El conjunto $U = \{(1, 2), (5, 6), (6, 8)\}$ no cruza el corte.

Teorema 6.6. Sea $G = (V, E)$ un grafo no dirigido conexo, donde cada rama $e \in E$ tiene asignado un peso $\omega(e)$, y sea ∂A un corte. Sea U un subconjunto de ramas de un mínimo spanning tree T de G que no cruza el corte.
Si $e \in \partial A$ es una rama de mínimo peso entonces $U \cup \{e\}$ es un subconjunto de ramas de un mínimo spanning tree T' de G.

Demostración: Si e es una rama de T no hay nada que probar. Supongamos entonces que no lo es.

Como T es conexo, si $e = (u, v)$ entonces existe un camino en T de u a v. Sean $e_1, \ldots, e_r$ las ramas de ese camino.
Ilustremos esta situación en un gráfico



Entonces existe k tal que $e_k \in \partial A$. En efecto, como $e = (u, v) \in \partial A$ entonces $u \in A$ y $v \in V - A$. Luego
si $u_1 \notin A$ se sigue que $e_1 \in \partial A$
si $u_1 \in A$ y $u_2 \notin A$ entonces $e_2 \in \partial A$
$\vdots$
si $u_{r-2} \in A$ y $u_{r-1} \notin A$ entonces $e_{r-1} \in \partial A$
y, finalmente, si $u_{r-1} \in A$ entonces $e_r \in \partial A$ pues $v \notin A$.

Notemos que como $e_k \in \partial A$ y $U \cap \partial A = \emptyset$ (U no cruza el corte) entonces se tiene que $e_k \notin U$.

Sea T' el grafo que resulta de suprimir en T la rama e_k y agregar la rama e. Veamos que T' es un spanning tree de G. En efecto, como T es conexo entonces T' también lo es. Sea $m = \#V$. Como T es un spanning tree de G entonces tiene $m - 1$ ramas. Luego T' tiene $m - 1$ ramas y m vértices. Luego T' es un árbol por la proposición 2.10. y por lo tanto es un spanning tree de G. Además es mínimo porque $\omega(e) \leq \omega(e_k)$ (recordemos que e era de mínimo peso en ∂A y que $e_k \in \partial A$).

Así, $U \cup \{e\}$ es un subconjunto de ramas del spanning tree mínimo T' pues la rama e_k que suprimimos no pertenecía a U. □

Validez del algoritmo.

Para cada k, consideremos el conjunto $U_k = \{$ ramas elegidas hasta la iteración $k\}$. Probaremos que, cualquiera sea k, U_k es un subconjunto de ramas de un spanning tree mínimo T_k.

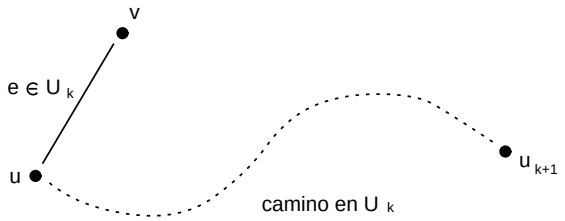
Iteración $k = 0$: En este caso $U_0 = \emptyset$ y basta tomar T_0 como un spanning tree mínimo cualquiera.
Iteración $k = 1$: sea $U = U_0$ y sea $e_1 = (u_1, v_1) \in E$ de mínimo peso. Entonces $U_1 = U \cup \{e_1\}$.
Sea ∂A_1 el corte definido por $A_1 = \{u_1\}$. Como U un subconjunto de ramas de un mínimo spanning tree T_0 de G que no cruza el corte y $e_1 \in \partial A_1$ es de mínimo peso, entonces $U_1 = U_0 \cup \{e_1\}$ es un subconjunto de ramas de un spanning tree mínimo T_1 por el teorema 6.6.

Supongamos ahora que nuestra afirmación es cierta para k, es decir, que U_k es un subconjunto de ramas de un spanning tree mínimo T_k.
Iteración $k + 1$: Sea $U = U_k$ y sea $e_{k+1} = (u_{k+1}, v_{k+1}) \notin U_k$ de mínimo peso entre las ramas que no pertenecen a U_k y que no forman ciclo con las ya elegidas. Entonces $U_{k+1} = U \cup \{e_{k+1}\}$.
Sea

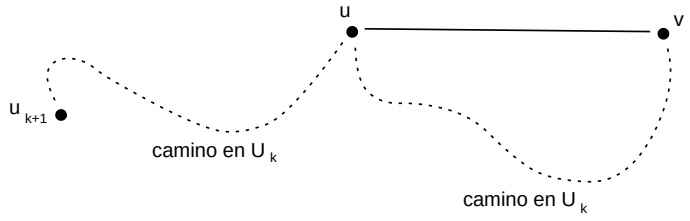
$$A_{k+1} = \{v \in V \,/\, \exists \text{ un camino en } U_k \text{ de } v \text{ a } u_{k+1}\} \cup \{u_{k+1}\}$$

y sea ∂A_{k+1} el corte definido por A_{k+1}.

Veamos que U no cruza el corte, es decir, que $\partial A_{k+1} \cap U_k = \emptyset$. Supongamos que no. Sea $e \in \partial A_{k+1} \cap U_k$. Luego $e = (u, v)$ donde $u \in A_{k+1}$, $v \notin A_{k+1}$ y $(u, v) = e \in U_k$. Como $u \in A_{k+1}$ entonces $u = u_{k+1}$ o existe un camino en U_k de u a u_{k+1}.



Luego, existe un camino en U_k de v a u_{k+1}. Absurdo, pues $v \notin A_{k+1}$.
Luego, U es un subconjunto de ramas de un spanning tree mínimo T_k que no cruza el corte.
Por otra parte, $e_{k+1} \in \partial A_{k+1}$. En efecto, $u_{k+1} \in A_{k+1}$ y $v_{k+1} \notin A_{k+1}$ ya que si existiera un camino en U_k uniendo v_{k+1} y u_{k+1} entonces e_{k+1} formaría un ciclo con las ramas ya elegidas.
Ahora veamos que e_{k+1} es de mínimo peso en ∂A_{k+1}. Como e_{k+1} fue elegida de mínimo peso entre las ramas que no pertenecían a U_k y que no formaban ciclo con las ya elegidas, basta ver que toda rama en ∂A_{k+1} no pertenece a U_k y no forma ciclo con las ya elegidas.
Si $e \in \partial A_{k+1}$, entonces claramente e no pertenece a U_k pues $U_k = U$ no cruza el corte. Veamos que no forma ciclo con las ramas ya elegidas.
Si $e = (u, v)$ con $u \in A_{k+1}$, $v \notin A_{k+1}$ formara ciclo, entonces existiría un camino en U_k uniendo u y v. Además, existiría un camino en U_k uniendo u_{k+1} y u pues $u \in A_{k+1}$.



Por lo tanto existiría un camino en U_k uniendo v y u_{k+1}, lo que contradice que $v \notin A_{k+1}$.

Luego, por el teorema 6.6., $U_{k+1} = U_k \cup \{e_{k+1}\}$ es un subconjunto de ramas de un spanning tree mínimo T_{k+1}.

Luego, para todo k, el conjunto de ramas elegidas hasta la k-ésima iteración es un subconjunto de ramas de un spanning tree mínimo T_k. Sea $m = \#V$. Como $\#U_k = k$ para todo k ya que en cada iteración se agrega una rama ($U_{k+1} = U_k \cup \{e_{k+1}\}$ con $e_{k+1} \notin U_k$) y el algoritmo se detiene cuando hay $m-1$ ramas elegidas, entonces el algoritmo se detiene cuando $k = m-1$. Además, U_{m-1} será un conjunto de $m-1$ ramas que es un subconjunto de ramas de un spanning tree mínimo T_{m-1}. Pero como T_{m-1} también tiene $m-1$ ramas por ser un spanning tree entonces se tiene que U_{m-1} es igual al conjunto de ramas de T_{m-1}. Esto significa que si E' es el conjunto de las $m-1$ ramas elegidas por el algoritmo entonces $(V, E') = T_{m-1}$ y por lo tanto es un mínimo spanning tree de G.

Para aplicar el algoritmo de Kruskal debemos ordenar las ramas de menor a mayor costo. Hay muchos algoritmos que se pueden utilizar para ordenar. Veremos uno conocido como '*divide and conquer*'.

Divide and conquer.

Supongamos que queremos ordenar de menor a mayor los números $a_1, a_2, \ldots, a_n$.

1. Partimos el conjunto $\{a_1, \ldots, a_n\}$ en los subconjuntos de $2 = 2^1$ elementos cada uno:

$$\{a_1, a_2\} \quad \{a_3, a_4\} \quad \{a_5, a_6\} \quad \cdots\cdots \quad \cdots\cdots \quad \cdots\cdots \quad \cdots\cdots$$

Ordenamos los elementos de cada uno de los subconjuntos de dos elementos.

2. Partimos el conjunto $\{a_1, \ldots, a_n\}$ en los subconjuntos de $4 = 2^2$ elementos cada uno:

$$\{a_1, a_2, a_3, a_4\} \quad \{a_5, a_6, a_7, a_8\} \quad \cdots\cdots \quad \cdots\cdots \quad \cdots\cdots$$

Ordenamos los elementos de cada uno de los subconjuntos de cuatro elementos teniendo en cuenta el orden obtenido en el paso previo.

3. Partimos el conjunto $\{a_1, \ldots, a_n\}$ en los subconjuntos de $8 = 2^3$ elementos cada uno:

$$\{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\} \quad \{a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}\} \quad \cdots\cdots \quad \cdots\cdots$$

Ordenamos los elementos de cada uno de los subconjuntos de ocho elementos teniendo en cuenta el orden obtenido en el paso previo.

Seguimos de este modo hasta tener un solo subconjunto con los n elementos ordenados.

Si $\{b_1, \ldots, b_r\}$ y $\{c_1, \ldots, c_r\}$ están ordenados, para ordenar $\{b_1, \ldots, b_r, c_1, \ldots, c_r\}$ (es decir, para ordenar cada subconjunto al pasar del paso k al paso $k+1$) necesitamos hacer, a lo sumo, $2r$ comparaciones. Veamos esto en un par de ejemplos y luego veamos el caso general.

Ejemplo 6.7 Supongamos que sabemos que $b_1 \leq b_2 \leq b_3 \leq b_4 \leq b_5 \leq b_6$ y que $c_1 \leq c_2 \leq c_3 \leq c_4 \leq c_5 \leq c_6$ y supongamos que el orden final sería

$$b_1, b_2, c_1, b_3, c_2, b_4, b_5, c_3, b_6, c_4, c_5, c_6$$

Entonces, para encontrar el orden final habremos tenido que comparar

$$c_1 \text{ con } b_1, b_2, b_3, \quad c_2 \text{ con } b_3, b_4, \quad c_3 \text{ con } b_4, b_5, b_6 \quad \text{y} \quad c_4 \text{ con } b_6$$

Necesitamos $9 \leq 2.6$ comparaciones.

Ejemplo 6.8 Supongamos que sabemos que

$$b_1 \leq b_2 \leq b_3 \leq b_4 \leq b_5$$

y que

$$c_1 \leq c_2 \leq c_3 \leq c_4 \leq c_5$$

y supongamos que el orden final debe ser

$$b_1, c_1, b_2, c_2, b_3, c_3, b_4, c_4, b_5, c_5$$

Entonces, para encontrar el orden final habremos tenido que comparar

$$c_1 \text{ con } b_1, b_2, \quad c_2 \text{ con } b_2, b_3, \quad c_3 \text{ con } b_3, b_4, \quad c_4 \text{ con } b_4, b_5 \quad \text{y} \quad c_5 \text{ con } b_5$$

Necesitamos $9 \leq 2.5$ comparaciones. Veamos ahora el caso general.

Veremos que para ordenar el conjunto $\{b_1, \ldots, b_r, c_1, \ldots, c_r\}$, donde $b_1 \leq b_2 \leq \cdots \leq b_r$ y $c_1 \leq c_2 \leq \cdots \leq c_r$, se necesitan a lo sumo $2r$ comparaciones. Sea $b_0 = -\infty$ y sea x_j definido por

$$x_j = \begin{cases} r & \text{si } c_j > b_r \\ 1 & \text{si } j = 0 \\ s & \text{si } b_{s-1} < c_j \leq b_s, \ (1 \leq s \leq r) \end{cases}$$

Entonces par ordenar c_j se necesitan a lo sumo $x_j - x_{j-1} + 1$ comparaciones ya que c_j debe ser comparado con los $x_j - x_{j-1} + 1$ números $b_{x_{j-1}}, \ldots, b_{x_j}$. En el ejemplo 6.7., para $j = 3$ resulta que $x_{j-1} = x_2 = 4$ (pues $b_3 < c_2 \leq b_4$) y $x_j = x_3 = 6$ pues $b_5 < c_3 \leq b_6$ y por lo tanto $c_j = c_3$ debe ser comparado con $b_{x_{j-1}}, \ldots, b_{x_j} = b_4, b_5, b_6$.

Luego, en total necesitaremos a lo sumo

$$x_1 + (x_2 - x_1 + 1) + (x_3 - x_2 + 1) + \cdots + (x_r - x_{r-1} + 1) = x_r + r - 1 \leq 2r$$

comparaciones para ordenar cada subconjunto de $2r$ elementos generado a partir de dos subconjuntos de r elementos ordenados.

Por último, veamos que el algoritmo 'divide and conquer' para ordenar n elementos tiene complejidad $O(n.\log n)$.

Para el primer paso necesitamos hacer $\frac{n}{2}$ comparaciones. Además, en cada paso k tenemos $\frac{n}{2^k}$ subconjuntos de 2^k elementos cada uno que queremos ordenar, y cada uno de estos subconjuntos es la unión de dos conjuntos de $r = 2^{k-1}$ elementos que ya están ordenados. Luego, necesitamos hacer $2r = 2.2^{k-1}$ comparaciones por cada uno de los $\frac{n}{2^k}$ subconjuntos, es decir, un total de $2^k.\frac{n}{2^k} = n$

Si $2^{t-1} < n \leq 2^t$ entonces en $t \leq 1+\log n$ pasos terminamos ya que en el paso k ordenamos subconjuntos de 2^k elementos. Como en cada paso hacemos a lo sumo n comparaciones y se realizan a lo sumo $1+\log n \leq 2\log n$ pasos entonces la complejidad de este algoritmo es $O(n.\log n)$, donde log denota el logaritmo en base 2.

Implementación del algoritmo de Kruskal.

Supongamos que $V = \{v_1, \ldots, v_m\}$ y $E = \{e_1, \ldots, e_n\}$, donde hemos ordenado las ramas de manera que e_1 sea la de menor peso, e_2 a la que sigue, etc.

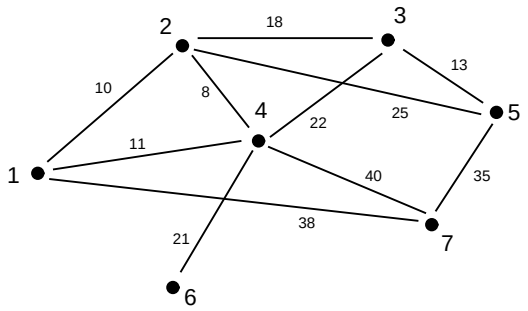
Ahora, en cada iteración debemos elegir una rama de mínimo peso entre las no elegidas que no forman ciclo con las que tenemos hasta ese momento. Para hacer esto en la forma en que una máquina pudiera hacerlo,

notemos que en la iteración k del algoritmo, el grafo (V, U_k), donde U_k denota el conjunto de ramas elegidas hasta la k-ésima iteración, constituye una foresta con $m-k$ árboles. En cada iteración asignaremos a cada uno de estos árboles un número entre 1 y m y llevaremos un registro de cuáles son los nodos y las ramas que lo forman.
Utilizaremos dos vectores $x = (x_1, \ldots, x_m)$ e $y = (y_1, \ldots, y_n)$ y una variable t. El valor de la coordenada x_i de x nos dirá el número de árbol al pertenece el nodo v_i y el valor de la coordenada y_j de y nos dirá el número de árbol al que pertenece la rama e_j, o valdrá -1 si e_j no fue examinada aún, o valdrá 0 si e_j fue examinada y descartada porque formaba ciclo con las ramas ya elegidas. El valor de la variable t nos indicará cuál es la rama que nos toca examinar. Además, utilizaremos un contador c que nos indicará cuál es la iteración que hemos realizado. El algoritmo se detendrá cuando c valga $m-1$.

En la iteración cero inicializamos las variables c y t poniendo $c = 0$, $t = 1$ y los vectores x e y en la forma $x_1 = 1$, $x_2 = 2, \ldots, x_m = m$, $y_1 = -1$, $y_2 = -1, \ldots, y_n = -1$. Esta asignación corresponde a la foresta inicial formada por los m vértices y ninguna rama.

Supongamos que hemos realizado las primeras k iteraciones. En este momento $c = k$ y la foresta puede describirse como sigue: cada árbol de la foresta está formado por las ramas y vértices cuyas correspondientes coordenadas en los vectores x e y tienen un mismo número positivo. Si $y_j > 0$ significa que la rama e_j fue elegida, si $y_j = 0$ significa que la rama e_j fue examinada y descartada porque formaba ciclo con las ya elegidas y si $y_j = -1$ significa que la rama e_j aún no fue examinada. El valor que tiene t en este momento nos indica cuál es la primera rama que debemos examinar al comenzar la iteración $k+1$.
Iteración $k+1$: examinamos la rama e_t. Supongamos que $e_t = (v_i, v_j)$. Para ver si forma un ciclo con las ramas de la foresta actual comparamos los dos valores de x_i y x_j correspondientes a los vértices v_i y v_j de e_t. Si son distintos, entonces estos vértices pertenecen a distintos árboles y por lo tanto agregar la rama fusionará esos árboles formando uno nuevo y no habrá ciclo. Pero si son iguales, entonces pertenecen a un mismo árbol. Como el árbol es conexo existe un camino en él de v_i a v_j y por lo tanto agregar la rama formará ciclo con las ya elegidas.
Si vemos que e_t forma ciclo, actualizamos el valor de y_t poniendo $y_t = 0$ para indicar que la rama e_t fue examinada y descartada, actualizamos t en la forma $t = t+1$ y pasamos a examinar la rama e_t. Si en cambio no forma ciclo, el agregar esa rama fusionará los árboles x_i y x_j. Supongamos que el valor de x_i es r y el de x_j es s, con $s < r$. Entonces actualizamos los vectores x e y poniendo $y_t = s$, $x_l = s \, \forall l \,/\, x_l = r$, $y_l = s \, \forall l \,/\, y_l = r$. Esto indica que ahora el árbol s está formado por todos los vértices y ramas que tenía hasta ese momento, la rama e_t y todos los vértices y ramas que antes pertenecían al árbol s. Además, actualizamos t y c en la forma $t = t+1$ y $c = c+1$.
Repetimos este procedimiento hasta que $c = m-1$, es decir, hasta que haya $m-1$ ramas elegidas (notemos que en cada iteración elegimos una rama). En ese momento todas las coordenadas de x valdrán 1 y las coordenadas de y valdrán 0, 1 o -1. Más aún, habrá exactamente $m-1$ coordenadas de y que tengan el valor 1. Esas son las coordenadas que corresponden a las $m-1$ ramas del mínimo spanning tree buscado.

Ejemplo 6.9. Aplicaremos el algoritmo de Kruskal al grafo con $m = 7$ vértices



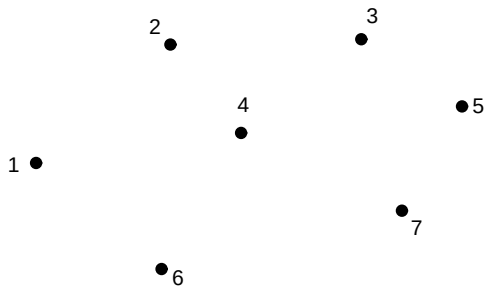
para obtener un mínimo spanning tree. Primero ordenamos las ramas de menor a mayor según el peso:

e	$\omega(e)$
$e_1 = (2,4)$	$\omega(e_1) = 8$
$e_2 = (1,2)$	$\omega(e_2) = 10$
$e_3 = (1,4)$	$\omega(e_3) = 11$
$e_4 = (3,5)$	$\omega(e_4) = 13$
$e_5 = (2,3)$	$\omega(e_5) = 18$
$e_6 = (6,4)$	$\omega(e_6) = 21$
$e_7 = (4,3)$	$\omega(e_7) = 22$
$e_8 = (2,5)$	$\omega(e_8) = 25$
$e_9 = (5,7)$	$\omega(e_9) = 35$
$e_{10} = (1,7)$	$\omega(e_{10}) = 38$
$e_{11} = (4,7)$	$\omega(e_{11}) = 40$

En la siguiente tabla se ven las actualizaciones de los vectores x e y, donde x_i corresponde al vértice i e y_j a la rama e_j y el estado del contador c.

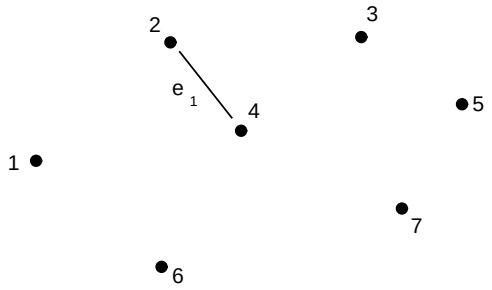
c	0	1	2	3	4	5	6
x_1	1	1	1	1	1	1	1
x_2	2	2	1	1	1	1	1
x_3	3	3	3	3	1	1	1
x_4	4	2	1	1	1	1	1
x_5	5	5	5	3	1	1	1
x_6	6	6	6	6	6	1	1
x_7	7	7	7	7	7	7	1
y_1	-1	2	1	1	1	1	1
y_2	-1	-1	1	1	1	1	1
y_3	-1	-1	-1	0	0	0	0
y_4	-1	-1	-1	3	1	1	1
y_5	-1	-1	-1	-1	1	1	1
y_6	-1	-1	-1	-1	-1	1	1
y_7	-1	-1	-1	-1	-1	-1	0
y_8	-1	-1	-1	-1	-1	-1	0
y_9	-1	-1	-1	-1	-1	-1	1
y_{10}	-1	-1	-1	-1	-1	-1	-1
y_{11}	-1	-1	-1	-1	-1	-1	-1

Ahora analicemos cómo es la foresta en cada iteración del algoritmo. En la interación cero la foresta inicial consiste de siete árboles, cada uno de ellos formado por un único vértice y ninguna rama.



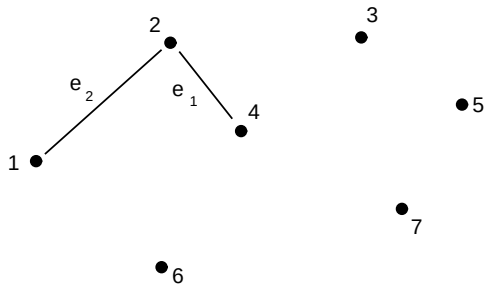
En este momento se tiene que $t = 1$ y $c = 0$.

Primera iteración: como $t = 1$ examinamos la rama e_1. Se comparan los valores de x_2 y x_4 correspondientes a los vértices 2 y 4 que son los extremos de la rama de menor peso e_1.
Como $x_2 = 2 \neq 4 = x_4$ entonces actualizamos los vectores x e y poniendo $x_2 = 2 = x_4$ e $y_1 = 2$. Esto significa que la rama e_1 fue elegida y, al agregarla, los árboles 2 y 4 se fusionaron formando un solo árbol, el árbol 2.
Ahora la foresta tiene seis árboles, cinco de ellos (los árboles 1, 3, 5, 6 y 7) formados por un vértice y ninguna rama y el otro (el árbol 2) por los vértices 2 y 4 y la rama e_1.



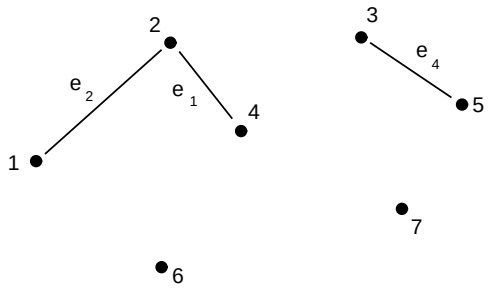
Actualizamos los valores de t y c poniendo $t = 2$ y $c = 1$.

Segunda iteración: como $t = 2$ se comparan los valores de x_1 y x_2 correspondientes a los vértices 1 y 2 que son los extremos de la rama e_2 (que es la de menor peso entre las no elegidas hasta el momento).
Como $x_1 = 1 \neq 2 = x_2$ entonces actualizamos los vectores x e y poniendo $x_2 = 1$, $x_4 = 1$, $y_1 = 1$ e $y_2 = 1$. Esto significa que la rama e_2 fue elegida y al agregarla se fusionaron los árboles 1 y 2 formando un solo árbol, el árbol 1.
Ahora la foresta tiene cinco árboles. Cuatro de ellos (los árboles 3, 5, 6 y 7) están formados por un vértice y ninguna rama y el otro (el árbol 1) está formado por los vértices 1, 2 y 4 y las ramas e_1 y e_2.



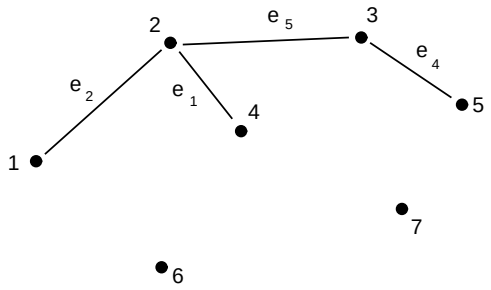
Actualizamos los valores de t y c poniendo $t = 3$ y $c = 2$.

Tercera iteración: como $t = 3$, se comparan los valores de x_1 y x_4 correspondientes a los vértices 1 y 4 que son los extremos de la rama e_3 (que es la de menor peso entre las no elegidas hasta el momento).
Como $x_1 = 2 = x_4$ entonces actualizamos los vectores x e y poniendo $y_3 = 0$. Esto significa que la rama e_3 fue descartada pues formaba ciclo con las ya elegidas. Ahora actualizamos t poniendo $t = 4$. Como $t = 4$, se comparan los valores de x_3 y x_5 correspondientes a los vértices 3 y 5 que son los extremos de la rama e_4 (que es la de menor peso entre las no elegidas hasta el momento). Como $x_3 = 3 \neq 5 = x_5$ entonces actualizamos los vectores x e y poniendo $x_3 = 3$, $x_5 = 3$ e $y_4 = 3$. Esto significa que la rama e_4 fue elegida y al agregarla se fusionaron los árboles 3 y 5 fomando un solo árbol, el árbol 3.
Ahora la foresta tiene cuatro árboles. Dos de ellos (los árboles 6 y 7) formados por un vértice y ninguna rama, el árbol 1 formado por los vértices 1, 2 y 4 y las ramas e_1 y e_2 y el árbol 3 por los vértices 3 y 5 y la rama e_4.

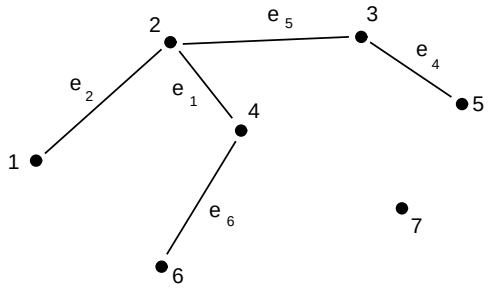


Actualizamos los valores de t y c poniendo $t = 5$ y $c = 3$.

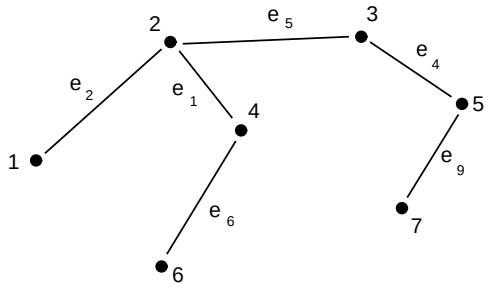
De manera análoga se ve que la foresta en las restantes iteraciones es como sigue
Foresta en la cuarta iteración, donde se fusionan los árboles 1 y 3 al agregar la rama e_5



Foresta en la quinta iteración, donde se fusionan los árboles 1 y 6 al agregar la rama e_6



Foresta en la sexta iteración, donde las ramas e_7 y e_8 son descartadas y se fusionan los árboles 1 y 7 al agregar la rama e_9



Este es el mínimo spanning tree de G buscado.

Complejidad del algoritmo.

Sean $m = \#V$ y $n = \#E$. Como en la primera iteración se ordenan las n ramas y además, en cada iteración

se hacen a lo sumo $c.n$ comparaciones y actualizaciones (donde c es una constante) y se realizan a lo sumo m iteraciones, entonces la complejidad de este algoritmo es $O(n.\log n) + m.O(n)$.
Dado que $n \leq \binom{m}{2} \leq m^2$, entonces $n.\log n \leq n.\log m^2 = 2n.\log m \leq 2n.m$. Por lo tanto la compeljidad es $O(n.m)$, es decir, es menor o igual que $k.n.m$ para una constante k. Como en el caso del algorimto search, este algoritmo es polinomial. En efecto, si N es el tamaño del input, se tiene que $n + m \leq N$. Luego $n.m \leq (n+m)^2 \leq N^2$ de donde la complejidad del algoritmo es menor o igual que $P(N)$ donde P es el polinomio $k\,X^2$.

7. El algoritmo de Prim.

Sea $G = (V, E)$ un grafo no dirigido, conexo, donde cada rama $e \in E$ tiene asignado un peso $\omega(e)$ y sea $m = \#V$. Describiremos ahora otro algoritmo, conocido como el algoritmo de Prim que también encuentra, en tiempo polinomial, un spanning tree mínimo de G. Dejamos como ejercicio el cálculo de su complejidad.

Descripción del algoritmo.

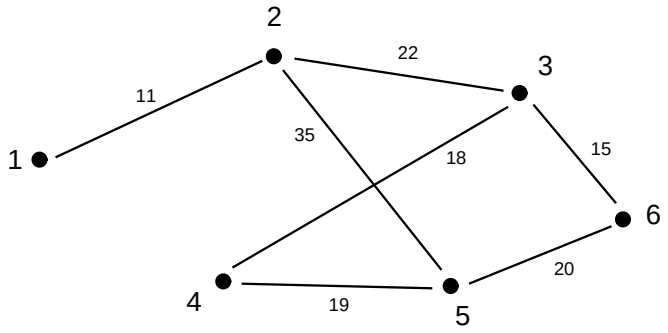
1. Ordenar las ramas de G de menor a mayor según el peso.
2. Elegir un vértice inicial cualquiera u_0.
3. Sea U el subárbol formado hasta ahora. Añadir una rama (u, v) a U de peso mínimo entre aquellas ramas no pertenecientes a U que agregadas a U forman un subárbol de G.
4. repetir 3. si el número de ramas del subárbol formado hasta ahora es menor que $m - 1$.

Validez del algoritmo.

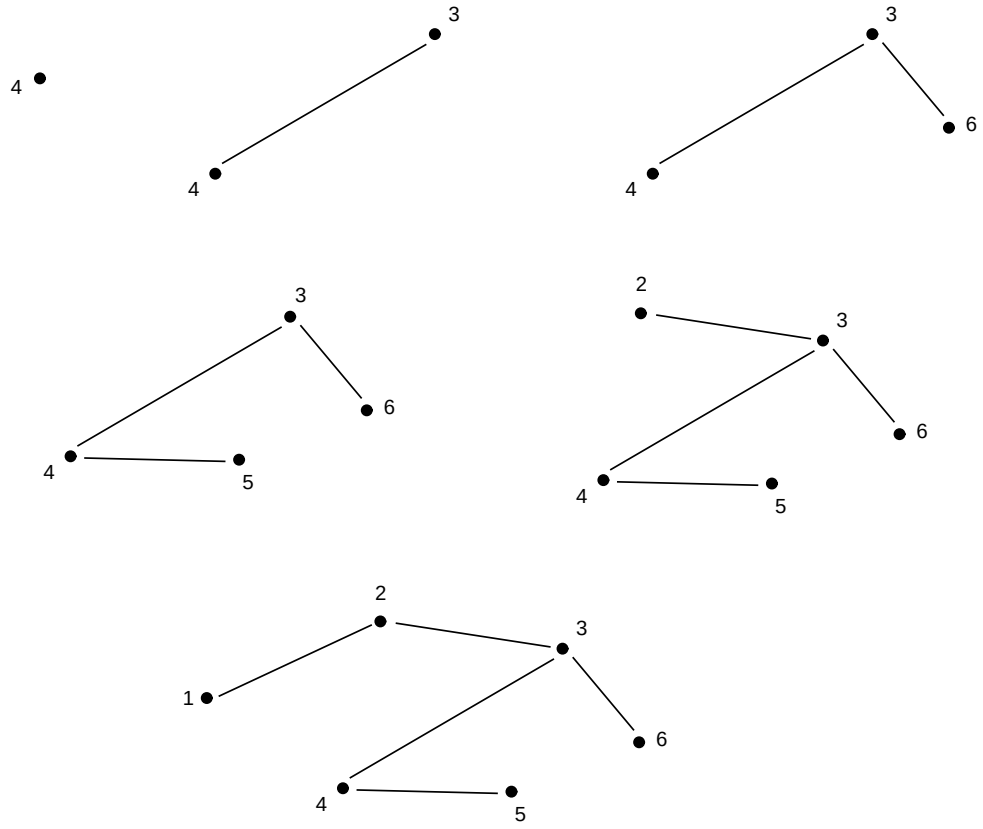
Dejamos a cargo del lector justificar la validez de este algoritmo. La demostración es análoga a la de la validez del algoritmo de Kruskal. Debe probarse que, para cada k, $U_k = \{$ ramas elegidas hasta el paso k $\}$ es un subconjunto de un spanning tree mínimo T_k.

Sugerencia: para el paso cero tomar $U_0 = \emptyset$, T_0 un spanning tree mínimo y, para el paso inductivo, aplicar el teorema 6.6. a $U = U_k = \{$ ramas del subárbol formado hasta ahora $\}$ y al corte definido por el conjunto $A_{k+1} = \{$ vértices del subárbol formado hasta ahora $\}$.
Notar que entonces $\partial A_{k+1} = \{e \in E - U \,/ U \cup \{e\}$ es un subárbol de G $\}$.

Ejemplo 7.1. Apliquemos el algoritmo de Prim al siguiente grafo, donde en cada rama hemos indicado su peso



Se tiene

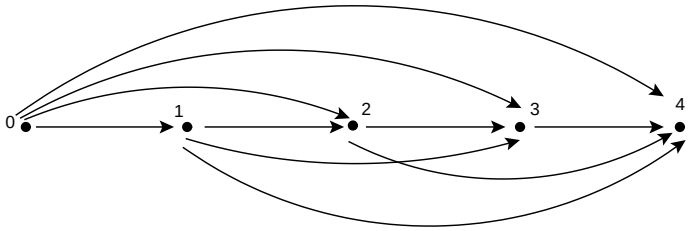


8. El camino óptimo en un grafo dirigido.

Sea $G = (V, E)$ un grafo dirigido y supongamos que para cada rama $e = (u, v) \in E$ tenemos definido un costo (o distancia) $c_e = c_{uv}$.
Si $\mathcal{P}$ es un camino dirigido en G definimos el costo de $\mathcal{P}$ como la suma de los costos de las ramas que lo forman. Fijado un vértice s queremos encontrar, para cada vértice v de G, un camino dirigido óptimo (i.e., de mínimo costo o distancia) de s a v. En esta sección analizaremos varios algoritmos que resuelven este problema, pero antes veamos su utilidad en un ejemplo.

Ejemplo 8.1. Una empresa es contratada para reparar 100 torres eléctricas, para lo cual necesitará comprar barras de acero de distintas longitudes $L_1 < L_2 < \cdots < L_n$ cuya única diferencia es la longitud. La empresa compra estas barras a un fabricante, quien está dispuesto a venderle barras de cualquier longitud siempre y cuando compre un mínimo de 100 barras de cada longitud. Por lo tanto, a la empresa no le conviene comprar barras de todas las n longitudes. En lugar de esto, le conviene comprar barras de ciertas longitudes y obtener las barras de cada longitud L_i que no tiene cortando las barras de longitud más chica entre aquellas que tiene y cuya longitud es mayor que L_i. Por ejemplo, si necesita 10 barras de longitud L_1, 5 de longitud L_2 y 90 de longitud L_3, le conviene comprar 105 barras de longitud L_3 y cortar 10 de ellas para obtener las barras de longitud L_1 y 5 de ellas para obtener las barras de longitud L_2 en lugar de comprar 100 barras de longitud L_1, 100 de longitud L_2 y 100 de longitud L_3. En cambio, si necesita 99 barras de longitud L_1, 20 de longitud L_2 y 90 de longitud L_3, entonces le conviene comprar 100 de longitud L_1 y 110 de longitud L_3. Se plantea entonces el problema de elegir cuáles longitudes conviene comprar de manera de poder obtener las restantes por recorte, a un costo mínimo. Como $L_n > L_i \ \forall i$, la longitud L_n debe ser una de las que se compren ya que no puede obtenerse por recorte de ninguna otra.
Para cada i entre 1 y n sea c_i el costo de una barra de longitud L_i y sea p_i la cantidad de barras de longitud L_i que necesitará la empresa para reparar las 100 torres.

Consideremos el grafo dirigido $G = (V, E)$, donde $V = \{0, 1, 2, \ldots, n\}$ y $E = \{(i, j) \,/\, 0 \leq i < j \leq n\}$. Por ejemplo, para $n = 4$ el grafo sería



Asignemos a cada rama (i, j) $(0 \leq i < j \leq n)$ el costo $c_{ij} = \max\{100, p_{i+1} + p_{i+2} + \cdots + p_j\}\, c_j$. Supongamos que $\mathcal{P}$ es un camino dirigido de 0 a n y que los vértices de ese camino son $0, j_1, \ldots, j_r, n$. Entonces, si elegimos comprar las longitudes $L_{j_1}, \ldots, L_{j_r}$ y L_n y obtener las restantes por recorte, resulta que el costo total de la compra determinado por esta elección es el costo de $\mathcal{P}$. En efecto, si $\mathcal{P}$ es el camino dirigido $(0, j_1), (j_1, j_2), \ldots, (j_{r-1}, j_r), (j_r, n)$ el costo de $\mathcal{P}$ es

$$c_{0j_1} + c_{j_1 j_2} + \cdots + c_{j_{r-1} j_r} + c_{j_r n}$$

Dado que de cada longitud deben comprarse un mínimo de 100 barras, entonces el costo de comprar k barras de una dada longitud L_j es $\max\{100, k\}\, c_j$. Luego, el costo de $\mathcal{P}$ es el costo de comprar $p_1 + p_2 + \cdots + p_{j_1}$ barras de longitud L_{j_1} + el costo de comprar $p_{j_1+1} + p_{j_1+2} + \cdots + p_{j_2}$ barras de longitud L_{j_2} + $\cdots$ + el costo de comprar $p_{j_{r-1}+1} + p_{j_{r-1}+2} + \cdots + p_{j_r}$ barras de longitud L_{j_r} + el costo de comprar $p_{1+j_r} + p_{2+j_r} + \cdots + p_n$ barras de longitud L_n. Luego el problema se resuelve hallando el camino dirigido de mínimo costo de 0 a n en G. Si el camino de mínimo costo es

$$0 \longrightarrow j_1 \longrightarrow j_2 \longrightarrow \cdots \longrightarrow j_r \longrightarrow n$$

entonces a la empresa le conviene comprar $\max\{100, p_1 + p_2 + \cdots + p_{j_1}\}$ barras de longitud L_{j_1} y obtener las de longitudes menores que L_{j_1} recortando las de longitud L_{j_1}, $\max\{100, p_{j_1+1} + p_{j_1+2} + \cdots + p_{j_2}\}$ barras de longitud L_{j_2} y obtener las de longitudes mayores que L_{j_1} y menores que L_{j_2} recortando las de longitud L_{j_2}, …, $\max\{100, p_{j_{r-1}+1} + p_{j_{r-1}+2} + \cdots + p_{j_r}\}$ barras de longitud L_{j_r} y obtener las de longitudes mayores que $L_{j_{r-1}}$ y menores que L_{j_r} recortando las de longitud L_{j_r} y $\max\{100, p_{1+j_r} + p_{2+j_r} + \cdots + p_n\}$ barras de longitud L_n y obtener las de longitudes mayores que L_{j_r} y menores que L_n recortando las de longitud L_n.

Sea G un grafo dirigido donde cada rama (u, v) tiene asignado un costo c_{uv} y donde se ha fijado un vértice s.
Describiremos a continuación varios algoritmos que encuentran (cuando existe) el camino dirigido óptimo de s a u y su costo, para cualquier vértice $u \neq s$.

Observación 8.2. Sea $\mathcal{R}$ es un camino óptimo de s a u y sea v el vértice anterior a u en ese camino. Si $\mathcal{P}$ la parte de ese camino que va de s a v entonces $\mathcal{P}$ es un camino óptimo a v.
En efecto, si hubiera un camino más "barato" de s a v, entonces ese camino seguido de la rama (v, u) sería un camino más "barato" que el camino que resulta de agregar a $\mathcal{P}$ la rama (v, u), es decir, más "barato" que el camino óptimo $\mathcal{R}$.

Método de programación dinámica.

Para poder aplicar este método necesitaremos que el grafo G no contenga ciclos dirigidos. Sea entonces $G = (V, E)$ un grafo que no contiene ciclos dirigidos, donde cada rama (u, v) tiene asignado un costo c_{uv} y

supongamos que el vértice fijado s de G es tal que de él sólo salen flechas. A este vértice s lo llamaremos la *fuente*.
Claramente podemos suponer que $V = \{1, 2, \ldots, m\}$. Para cada $i < j$ tal que $(i, j) \notin E$ tomaremos $c_{ij} = \infty$.
Supondremos además que los vértices están numerados de 1 a m de forma tal que valga

$$(i, j) \in E \Longrightarrow i < j$$

Dejamos como tarea para el lector probar que si G no contiene ciclos dirigidos siempre es posible renumerar los vértices de manera que esto se verifique.
Luego, $s = 1$ y vale $\{i \,/\, (i, j) \in E\} \subseteq \{1, 2, \ldots, j-1\}$. Sea

$$y_j = \begin{cases} 0 & \text{si } j = 1 \\ \infty & \text{si } j \geq 2 \text{ y } \nexists \text{ camino dirigido de 1 a } j \\ \text{costo de un camino dirigido óptimo de 1 a } j & \text{en otro caso} \end{cases}$$

(Notar que, como el grafo no contiene ciclos dirigidos, si existe un camino dirigido a j entonces existe un camino dirigido óptimo a j)

Luego, para todo $i < j$ es $y_j \leq y_i + c_{ij}$. En efecto, esto es obvio si $(i, j) \in E$. Pero si $(i, j) \notin E$ entonces también vale pues en ese caso $c_{ij} = \infty$. Más aún, por la observación 8.2., si i es el vértice anterior a j en un camino dirigido óptimo de 1 a j, entonces $y_j = y_i + c_{ij}$.
Esto muestra que

$$y_j = \min_{1 \leq i \leq j-1} \{y_i + c_{ij}\}$$

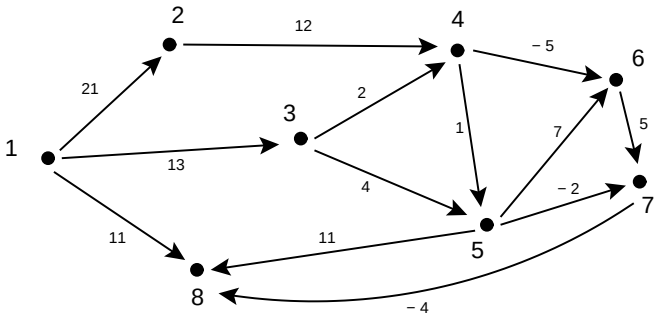
y que si k es el índice que realiza el mínimo entonces k es el vértice anterior a j en un camino óptimo de 1 a j.

Descripción del algoritmo.

1. $y_1 = 0$, j=2.
2. $y_j = \min_{1 \leq i \leq j-1} \{y_i + c_{ij}\}$. Si $y_k + c_{kj} = \min_{1 \leq i \leq j-1} \{y_i + c_{ij}\}$ poner $p_j = k$.
3. $j = j + 1$. Si $j \leq m$ ir a 2.
4. STOP

Como p_j es el índice que realiza el mínimo entonces es el predecesor a j en un camino óptimo. Luego, conociendo p_j para cada $2 \leq j \leq m$ podemos reconstruír ese camino.

Ejemplo 8.3. Apliquemos el algoritmo al grafo



donde para cada rama (i, j) hemos indicado su costo c_{ij}. Notemos que este grafo no contiene ciclos dirigidos y satisface $(i, j) \in E \Longrightarrow i < j$

$y_1 = 0$
$y_2 = 21$, $p_2 = 1$
$y_3 = \min\{y_1 + c_{13}, y_2 + c_{23}\}$. Como $y_1 + c_{13} = 13$ e $y_2 + c_{23} = \infty$ entonces $y_3 = 13$, $p_3 = 1$.
$y_4 = \min\{y_1 + c_{14}, y_2 + c_{24}, y_3 + c_{34}\}$. Como $y_1 + c_{14} = \infty$, $y_2 + c_{24} = 33$ e $y_3 + c_{34} = 15$ entonces $y_4 = 15$, $p_4 = 3$.
$y_5 = \min\{y_1 + c_{15}, y_2 + c_{25}, y_3 + c_{35}, y_4 + c_{45}\}$. Como $y_1 + c_{15} = \infty$, $y_2 + c_{25} = \infty$, $y_3 + c_{35} = 17$ e $y_4 + c_{45} = 16$ entonces $y_5 = 16$, $p_5 = 4$.
$y_6 = \min\{y_1 + c_{16}, y_2 + c_{26}, y_3 + c_{36}, y_4 + c_{46}, y_5 + c_{56}\}$. Como $y_1 + c_{16} = \infty$, $y_2 + c_{26} = \infty$, $y_3 + c_{36} = \infty$, $y_4 + c_{46} = 10$ e $y_5 + c_{56} = 23$ entonces $y_6 = 10$, $p_6 = 4$.
Análogamente $y_7 = 14$, $p_7 = 5$, $y_8 = 10$ y $p_8 = 7$.

En el camino óptimo de 1 a 7 el predecesor de 7 es $p_7 = 5$, el de 5 es $p_5 = 4$, el de 4 es $p_4 = 3$ y el de 3 es $p_3 = 1$. Luego, el camino dirigido óptimo de 1 a 7 es $1 \longrightarrow 3 \longrightarrow 4 \longrightarrow 5 \longrightarrow 7$ con costo $y_7 = 14$. Análogamente el camino dirigido óptimo de 1 a 6 es $1 \longrightarrow 3 \longrightarrow 4 \longrightarrow 6$ con costo $y_6 = 10$.

Método de Dijkstra.

Este método sólo se podrá aplicar en el caso en que los costos $c_e = c_{uv}$ son no negativos para toda rama $e = (u, v)$. Si el grafo $G = (V, E)$ no es completo ponemos $c_{ij} = \infty$ si $(i, j) \notin E$.

Descripción del algoritmo.
1. Poner

$$y_u = \begin{cases} 0 & \text{si } u = s \\ \infty & \text{si } u \neq s \end{cases}, \qquad p_u = \begin{cases} -1 & \text{si } u = s \\ s & \text{si } u \neq s \end{cases}, \qquad A = \{s\}$$

2. Sea v el último vértice que ingresó a A. Para todo $u \notin A$ tal que $y_v + c_{vu} < y_u$ actualizar y_u y p_u en la forma

$$y_u = y_v + c_{vu} \qquad p_u = v$$

3. Sea $u \notin A$ tal que $y_u = \min_{w \notin A}\{y_w\}$. Poner $A = A \cup \{u\}$.
4. Si $A \neq V$ goto 2.

Al finalizar el algoritmo, para todo $u \neq s$ vale: si $y_u < \infty$ entonces y_u es el costo de un camino dirigido óptimo de s a u y p_u es el vértice anterior a u en ese camino y si $y_u = \infty$ entonces no existe un camino dirigido de s a u.

Ejemplo 8.4. Apliquemos el algoritmo al grafo completo de seis vértices, para hallar el camino dirigido de mínimo costo desde el nodo $s = 1$ hasta cada uno de los restantes nodos, donde los costos de cada rama están dados por la matriz

$$||c_{ij}|| = \begin{pmatrix} \infty & 5 & 14 & 3 & 7 & 6 \\ 13 & \infty & 3 & 7 & 4 & 1 \\ 2 & 2 & \infty & 2 & 7 & 2 \\ 1 & 1 & 12 & \infty & 21 & 10 \\ 5 & 3 & 1 & 1 & \infty & 2 \\ 2 & 7 & 3 & 2 & 1 & \infty \end{pmatrix}$$

Primera iteración:

1. $y_1 = 0$, $p_1 = -1$, $y_u = \infty$, $p_u = 1$ $(u \neq 1)$, $A = \{1\}$.
Luego $(y_1, y_2, y_3, y_4, y_5, y_6) = (0, \infty, \infty, \infty, \infty, \infty)$ y $(p_1, p_2, p_3, p_4, p_5, p_6) = (-1, 1, 1, 1, 1, 1)$
2. $v = 1$
Para $u = 2$, $y_v + c_{vu} = y_1 + c_{12} = 5 < \infty = y_2 = y_u$. Luego ponemos $y_u = y_v + c_{vu}$, es decir, $y_2 = y_1 + c_{12} = 5$ y $p_u = v$, es decir, $p_2 = 1$.

Para $u = 3$, $y_v + c_{vu} = y_1 + c_{13} = 14 < \infty = y_3 = y_u$. Luego ponemos $y_u = y_v + c_{vu}$, es decir, $y_3 = y_1 + c_{13} = 14$ y $p_u = v$, es decir, $p_3 = 1$.
Para $u = 4$, $y_v + c_{vu} = y_1 + c_{14} = 3 < \infty = y_4 = y_u$. Luego ponemos $y_u = y_v + c_{vu}$, es decir, $y_4 = y_1 + c_{14} = 3$ y $p_u = v$, es decir, $p_4 = 1$.
Para $u = 5$, $y_v + c_{vu} = y_1 + c_{15} = 7 < \infty = y_5 = y_u$. Luego ponemos $y_u = y_v + c_{vu}$, es decir, $y_5 = y_1 + c_{15} = 7$ y $p_u = v$, es decir, $p_5 = 1$.
Para $u = 6$, $y_v + c_{vu} = y_1 + c_{16} = 6 < \infty = y_6 = y_u$. Luego ponemos $y_u = y_v + c_{vu}$, es decir, $y_6 = y_1 + c_{16} = 6$ y $p_u = v$, es decir, $p_6 = 1$.
Ahora $(y_1, y_2, y_3, y_4, y_5, y_6) = (0, 5, 14, 3, 7, 6)$ y $(p_1, p_2, p_3, p_4, p_5, p_6) = (-1, 1, 1, 1, 1, 1)$.
3. $\min_{w \notin A}\{y_w\} = y_4$. Luego $A = \{1, 4\}$
4. $A \neq V$, goto 2.

Segunda iteración:

2. $v = 4$
Para $u = 2$, $y_v + c_{vu} = y_4 + c_{42} = 4 < 5 = y_2 = y_u$. Luego ponemos $y_u = y_v + c_{vu}$, es decir, $y_2 = y_4 + c_{42} = 4$ y $p_u = v$, es decir, $p_2 = 4$.
Para $u = 3$, $y_v + c_{vu} = y_4 + c_{43} = 15 > 14 = y_3 = y_u$.
Para $u = 5$, $y_v + c_{vu} = y_4 + c_{45} = 24 > 7 = y_5 = y_u$.
Para $u = 6$, $y_v + c_{vu} = y_4 + c_{46} = 13 > 6 = y_6 = y_u$.
Ahora $(y_1, y_2, y_3, y_4, y_5, y_6) = (0, 4, 14, 3, 7, 6)$ y $(p_1, p_2, p_3, p_4, p_5, p_6) = (-1, 4, 1, 1, 1, 1)$.
3. $\min_{w \notin A}\{y_w\} = y_2$. Luego $A = \{1, 4, 2\}$
4. $A \neq V$, goto 2.

Tercera iteración:

2. $v = 2$
Para $u = 3$, $y_v + c_{vu} = y_2 + c_{23} = 7 < 14 = y_3 = y_u$. Ponemos $y_3 = 7$, $p_3 = 2$.
Para $u = 5$, $y_v + c_{vu} = y_2 + c_{25} = 8 > 7 = y_5 = y_u$.
Para $u = 6$, $y_v + c_{vu} = y_2 + c_{26} = 5 < 6 = y_6 = y_u$. Ponemos $y_6 = 5$, $p_6 = 2$
Ahora $(y_1, y_2, y_3, y_4, y_5, y_6) = (0, 4, 7, 3, 7, 5)$ y $(p_1, p_2, p_3, p_4, p_5, p_6) = (-1, 4, 2, 1, 1, 2)$.
3. $\min_{w \notin A}\{y_w\} = y_6$. Luego $A = \{1, 4, 2, 6\}$
4. $A \neq V$, goto 2.

Cuarta iteración:

2. $v = 6$
Para $u = 3$, $y_v + c_{vu} = y_6 + c_{63} = 8 > 7 = y_3 = y_u$.
Para $u = 5$, $y_v + c_{vu} = y_6 + c_{65} = 6 < 7 = y_5 = y_u$. Ponemos $y_5 = 6$, $p_5 = 6$.
Ahora $(y_1, y_2, y_3, y_4, y_5, y_6) = (0, 4, 7, 3, 6, 5)$ y $(p_1, p_2, p_3, p_4, p_5, p_6) = (-1, 4, 2, 1, 6, 2)$.
3. $\min_{w \notin A}\{y_w\} = y_5$. Luego $A = \{1, 4, 2, 6, 5\}$
4. $A \neq V$, goto 2.

Quinta iteración:

2. $v = 5$
Para $u = 3$, $y_v + c_{vu} = y_5 + c_{53} = 7 \geq 7 = y_3 = y_u$.
Ahora $(y_1, y_2, y_3, y_4, y_5, y_6) = (0, 4, 7, 3, 6, 5)$ y $(p_1, p_2, p_3, p_4, p_5, p_6) = (-1, 4, 2, 1, 6, 2)$.
3. $\min_{w \notin A}\{y_w\} = y_3$. Luego $A = \{1, 4, 2, 6, 5, 3\}$
4. $A = V$, STOP.

En la siguiente tabla se ven cómo son las actualizaciones correspondientes a cada iteración de los vectores $y = (y_1, \ldots, y_6)$ y $p = (p_1, \ldots, p_6)$ y cuál es el elemento que ingresa en el conjunto A en esa iteración.

	0	1	2	3	4	5
y_1	0	0	0	0	0	0
y_2	∞	5	4	4	4	4
y_3	∞	14	14	7	7	7
y_4	∞	3	3	3	3	3
y_5	∞	7	7	7	6	6
y_6	∞	6	6	5	5	5
p_1	-1	-1	-1	-1	-1	-1
p_2	1	1	4	4	4	4
p_3	1	1	1	2	2	2
p_4	1	1	1	1	1	1
p_5	1	1	1	1	6	6
p_6	1	1	1	2	2	2
A	1	4	2	6	5	3

Luego, para cada $u \neq 1$, el camino dirigido óptimo de 1 a u y su costo y_u es
Si $u = 2$: $1 \longrightarrow 4 \longrightarrow 2$, $y_2 = 4$ (en el camino óptimo de 1 a 2 el antecesor de 2 es $p_2 = 4$ y el antecesor de 4 es $p_4 = 1$).
Si $u = 3$: $1 \longrightarrow 4 \longrightarrow 2 \longrightarrow 3$, $y_3 = 7$
Si $u = 4$: $1 \longrightarrow 4$, $y_4 = 3$
Si $u = 5$: $1 \longrightarrow 4 \longrightarrow 2 \longrightarrow 6 \longrightarrow 5$, $y_5 = 6$
Si $u = 6$: $1 \longrightarrow 4 \longrightarrow 2 \longrightarrow 6$, $y_6 = 5$

Lema 8.5. En cada iteración del algoritmo se verifica
Para todo $w \in A$, si $u \notin A$ o si u es un vértice que ha ingresado a A después que w entonces $y_u \leq y_w + c_{wu}$.

Demostración: Sea $w \in A$. Si $u \notin A$ o si u es un vértice que ha ingresado a A después que w entonces en la iteración siguiente a aquella en la que ingresamos w a A comparamos $y_w + c_{wu}$ con y_u, ya que en ese momento u no pertenece a a A y w es el último vértice que ingresó en A. Si $y_w + c_{wu} < y_u$, entonces ponemos $y_u = y_w + c_{wu}$. Luego, en ese momento vale $y_u \leq y_w + c_{wu}$. Teniendo en cuenta que en cada iteración el valor de y_u o no se cambia o se reemplaza por algo menor, entonces el valor de y_u en todas las iteraciones siguientes será menor o igual que $y_w + c_{wu}$. □

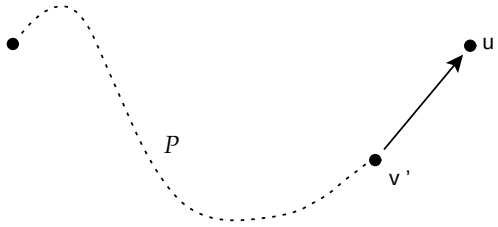
Validez del algoritmo.

De ahora en adelante la palabra camino significará camino dirigido, a menos que se indique lo contrario.
Veremos que para cada u que ingresa en A, si $y_u = \infty$ entonces no existe ningún camino de s a u y si $y_u < \infty$ entonces y_u es el costo de un camino óptimo de s a u y que p_u es el predecesor a u en ese camino o $p_u = -1$ si $u = s$. Esto es claro para s que es el primer vértice que ingresa en A.
Supongamos que han pasado varias iteraciones del algoritmo y sea u el último vértice que ingresó a A. Demostraremos que nuestra afirmación vale para u suponiendo que vale para todo vértice que ingresara a A antes que u.
Si $y_u < \infty$, como al iniciarse el algoritmo era $y_u = \infty$, entonces el valor de y_u fue modificado. Sea v el vértice que ingresó a A antes que u que nos hizo modificar y_u y p_u la última vez, es decir, tal que

$$y_u = y_v + c_{vu} \qquad p_u = v$$

Como por hipótesis inductiva lo que queremos demostrar vale para v entonces y_v es el costo de un camino óptimo $\mathcal{C}$ de s a v. Luego $y_u = y_v + c_{vu}$ es el costo del camino de s a u que resulta de agregar a $\mathcal{C}$ la rama (v, u). Veamos que y_u es el costo de un camino óptimo a u.

Sea v' el vértice anterior a u en un camino óptimo $\mathcal{R}$ a u y sea $\mathcal{P}$ la parte de este camino que va de s a v'. La situación es

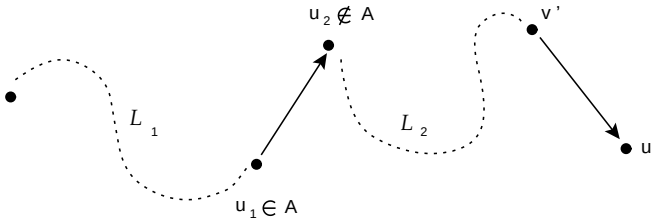


Por la observación 8.2. se tiene que $\mathcal{P}$ es un camino óptimo a v'. Luego

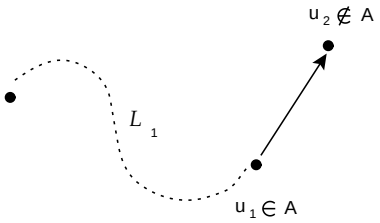
i) Si $v' \in A$ entonces $y_{v'}$ es el costo de un camino óptimo a v' entonces $y_{v'}$ es el costo de cualquier camino óptimo a v', en particular $y_{v'}$ es el costo de $\mathcal{P}$. Por lo tanto el costo de $\mathcal{R}$ es $y_{v'} + c_{v'u}$. Pero $y_u \leq y_w + c_{wu}$ para todo otro $w \in A$ por el lema 8.5., de donde resulta que $y_u \leq y_{v'} + c_{v'u}$. Como y_u es el costo de un camino a u y como $y_{v'} + c_{v'u}$ es el costo de $\mathcal{R}$, que es óptimo, entonces debe valer la igualdad y por lo tanto y_u es el costo de un camino óptimo a u como queríamos probar.

ii) Si $v' \notin A$, como $s \in A$ entonces existen dos vértices sucesivos u_1, u_2 en $\mathcal{P}$ tales que $u_1 \in A$ y $u_2 \notin A$. Sean $\mathcal{L}_1$ la parte del camino $\mathcal{P}$ que va de s a u_1 y $\mathcal{L}_2$ la que va de u_2 a v'. Luego, como u_1 y u_2 son vértices sucesivos, el camino óptimo $\mathcal{R}$ está formado por el camino $\mathcal{L}_1$ seguido de la rama (u_1, u_2), luego seguido del camino $\mathcal{L}_2$ y finalmente seguido de la rama (v', u).

Ahora la situacón es



Como $\mathcal{R}$ era óptimo, la parte de este camino que va de s a u_2, es decir, el camino $\mathcal{L}_1$ seguido de la rama (u_1, u_2)



es un camino óptimo a u_2.

Como por hipótesis $c_e \geq 0$ para toda rama $e \in E$, el costo de un camino óptimo de s a u es mayor o igual que el costo de un camino óptimo de s a u_2 y este último es igual al costo del camino $\mathcal{L}_1$ más el costo $c_{u_1 u_2}$ de la rama (u_1, u_2).

Como $u_1 \in A$, $u_1 \neq u$ entonces, por hipótesis inductiva, el costo de $\mathcal{L}_1$ es y_{u_1} ya que $\mathcal{L}_1$ es un camino óptimo a u_1 (ver observación 8.2.). Además, como $y_{u_2} \notin A$ entonces $y_{u_2} \leq y_{u_1} + c_{u_1 u_2}$ por el lema 8.5.

Por otra parte, como u era el último vértice que ingresó a A entonces $y_u = \min_{w \notin A}\{y_w\}$ de donde $y_u \leq y_w$ para todo $w \notin A$. En particular, $y_u \leq y_{u_2}$

Luego el costo de un camino óptimo de s a $u \geq$ costo de un camino óptimo de s a u_2 = costo del camino $\mathcal{L}_1 + c_{u_1u_2} = y_{u_1} + c_{u_1u_2} \geq y_{u_2} \geq y_u$.
Esto prueba que y_u es menor o igual que el costo de un camino óptimo de s a u y, como y_u es el costo de un camino a u, entonces debe valer la igualdad.

Por último, veamos que si $u \neq s$ entonces p_u es el vértice que precede a u en un camino óptimo de s a u.
Por construcción, $y_u = y_v + c_{vu}$ para algún v y, por lo que acabamos de probar, y_u es el costo de cualquier camino óptimo a u e y_v es el costo de cualquier camino óptimo a v.
Sea $\mathcal{P}$ un camino óptimo a v y sea $\mathcal{R}$ el camino a u que resulta de agregarle a $\mathcal{P}$ la rama (v, u). Luego, el costo de $\mathcal{R}$ = costo de $\mathcal{P} + c_{vu} = y_v + c_{vu} = y_u$. Entonces $\mathcal{R}$ es un camino óptimo a u y en este camino v es el predecesor a u.

Complejidad del algoritmo de Dijkstra.

En la iteración k-ésima se realizan a lo sumo $c.(m-k)$ operaciones, donde $m = \#V$ y c es una constante que no depende del grafo. Como hay $m-1$ iteraciones, entonces la complejidad es menor o igual que

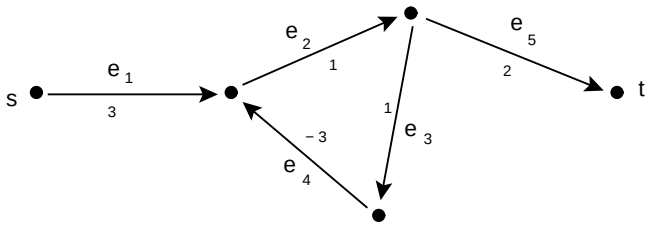
$$\sum_{k=1}^{m-1} c.(m-k) = c.\frac{m(m-1)}{2}$$

es decir, la complejidad de este algoritmo es polinomial del orden de m^2.

Método de Ford-Bellman.

Este método es el más general: el grafo puede contener ciclos dirigidos y ramas con costos negativos. Podría ocurrir, entonces, que para algún t no exista un camino mínimo de s a t tal como se ve en el siguiente ejemplo.

Ejemplo 8.6. El grafo



contiene un ciclo dirigido de costo negativo. En efecto, el ciclo (e_2, e_3, e_4) tiene costo -1. Luego no existe un camino óptimo de s a t ya que hay caminos de costo tan pequeño como se quiera: el camino que resulta de recorrer primero la rama e_1, luego n veces el ciclo (e_2, e_3, e_4) y finalmente la rama e_5 es un camino de s a t de costo $3 + n.(-1) + 2 = 5 - n$.

Supongamos que y_u sea el costo de un camino (no necesariamente óptimo) de s a u, para cada $u \in V$. Dado v, si para algún $u \neq v$ fuese $y_u + c_{uv} < y_v$ entonces es mejor el camino de s a u cuyo costo es y_u seguido de la rama (u, v) que el camino cuyo costo es y_v. Luego, si y_v fuese el costo de un camino óptimo entonces debería ser $y_v \leq y_w + c_{wv}$ para todo $w \neq v$. Esto sugiere el siguiente algoritmo

Descripción del algoritmo.

1. Poner

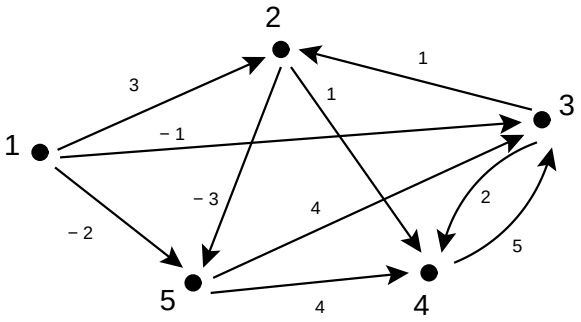
$$y_u = \begin{cases} 0 & \text{si } u = s \\ \infty & \text{si } u \neq s \end{cases}, \qquad p_u = -1, \qquad i = 1$$

2. Para cada v poner $y'_v = y_v$, si $y_v \leq y_u + c_{uv}$ para todo u tal que $(u,v) \in E$ o poner $y'_v = y_w + c_{wv}$ y $p_v = w$, donde $y_w + c_{wv} = \min\{y_u + c_{uv} \,/\, (u,v) \in E \text{ e } y_v > y_u + c_{uv}\}$, si $y_v > y_u + c_{uv}$ para algún u tal que $(u,v) \in E$

3. $y_v = y'_v$, $i = i + 1$

4. Si $i < \#V$ goto 2. Si no, STOP.

Ejemplo 8.7. Apliquemos el algoritmo de Ford-Bellman al grafo



para encontrar el camino óptimo de $s = 1$ a cada uno de los restantes vértices.

Primera iteración:
1. $(y_1, y_2, y_3, y_4, y_5) = (0, \infty, \infty, \infty, \infty)$
$(p_1, p_2, p_3, p_4, p_5) = (-1, -1, -1, -1, -1)$
$i = 1$.

2. Para $v = 1$ ponemos $y'_1 = y_1 = 0$ pues no existe u tal que $(u,1) \in E$ e $y_1 > y_u + c_{u1}$.
Para $v = 2$ el único u tal que $(u,2) \in E$ e $y_2 > y_u + c_{u2}$ es $u = 1$. En este caso $y_1 + c_{12} = 3$ de modo que ponemos $y'_2 = 3$ y $p_2 = 1$.
Para $v = 3$ el único u tal que $(u,3) \in E$ e $y_3 > y_u + c_{u3}$ es $u = 1$. En este caso $y_1 + c_{13} = -1$ de modo que ponemos $y'_3 = -1$ y $p_3 = 1$.
Para $v = 4$ no existe u tal que $(u,4) \in E$ e $y_4 > y_u + c_{u4}$. Luego ponemos $y'_4 = y_4 = \infty$.
Para $v = 5$ el único u tal que $(u,5) \in E$ e $y_5 > y_u + c_{u5}$ es $u = 1$. En este caso $y_1 + c_{15} = -2$ de modo que ponemos $y'_5 = -2$ y $p_5 = 1$.

3. $(y_1, y_2, y_3, y_4, y_5) = (y'_1, y'_2, y'_3, y'_4, y'_5) = (0, 3, -1, \infty, -2)$
$(p_1, p_2, p_3, p_4, p_5) = (-1, 1, 1, -1, 1)$
$i = 2$.

4. Como $i = 2 < 5 = \#V$ goto 2.

Segunda iteración:
2. Para $v = 1$ ponemos $y'_1 = y_1 = 0$ pues no existe u tal que $(u,1) \in E$ e $y_1 > y_u + c_{u1}$.

Para $v = 2$ el único u tal que $(u,2) \in E$ e $y_2 > y_u + c_{u2}$ es $u = 3$. En este caso $y_3 + c_{32} = 0$ de modo que ponemos $y'_2 = 0$ y $p_2 = 3$.

Para $v = 3$ no existe u tal que $(u,3) \in E$ e $y_3 > y_u + c_{u3}$ así que ponemos $y'_3 = y_3 = -1$.

Para $v = 4$ se tiene que para $u = 2, 3, 5$ vale que $(u,4) \in E$ e $y_4 > y_u + c_{u4}$.
Luego ponemos $y'_4 = \min\{y_2 + c_{24}, y_3 + c_{34}, y_5 + c_{54}\} = y_3 + c_{34} = 1$ y $p_4 = 3$.

Para $v = 5$ no existe u tal que $(u,5) \in E$ e $y_5 > y_u + c_{u5}$. Luego ponemos $y'_5 = y_5 = -2$.

3. $(y_1, y_2, y_3, y_4, y_5) = (y'_1, y'_2, y'_3, y'_4, y'_5) = (0, 0, -1, 1, -2)$
$(p_1, p_2, p_3, p_4, p_5) = (-1, 3, 1, 3, 1)$

$i = 3$.

4. Como $i = 3 < 5 = \#V$ goto 2.

Continuamos de esta manera hasta que tengamos $i = 5$. La siguiente tabla resume la información de las iteraciones

	0	1	2	3	4
y_1	0	0	0	0	0
y_2	∞	3	0	0	0
y_3	∞	-1	-1	-1	-1
y_4	∞	∞	1	1	1
y_5	∞	-2	-2	-3	-3
p_1	-1	-1	-1	-1	-1
p_2	-1	1	3	3	3
p_3	-1	1	1	1	1
p_4	-1	-1	3	3	3
p_5	-1	1	1	2	2

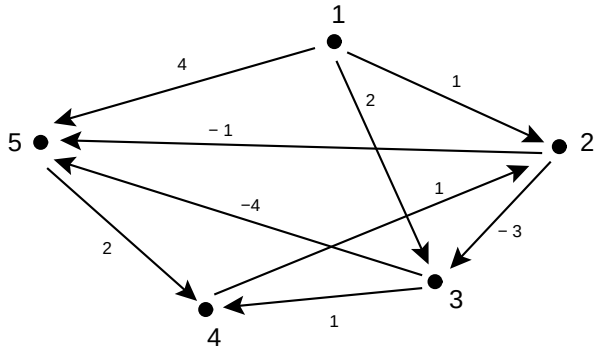
Dejamos a cargo del lector verificar los valores de y_v y de p_v en las restantes iteraciones y también que los valores finales de y_v ($v \in V$) satisfacen que para todo $(u, v) \in E$ es $y_v \leq y_u + c_{uv}$. Veremos luego que al terminar el algoritmo se verifican

i) Si $y_w < \infty$ para todo w entonces el grafo no contiene ciclos dirigidos de costo negativo sii $y_v \leq y_u + c_{uv}$ $\forall (u, v) \in E$. En tal caso, para cada v se tiene que p_v es el vértice predecesor a v en un camino óptimo de s a v cuyo costo es y_v.

ii) $y_v = \infty$ sii no existe un camino dirigido de s a v.

Luego, en el ejemplo anterior resulta que el camino óptimo de 1 a 2 es $1 \longrightarrow 3 \longrightarrow 2$ con costo $y_2 = 0$, de 1 a 3 es $1 \longrightarrow 3$ con costo $y_3 = -1$, de 1 a 4 es $1 \longrightarrow 3 \longrightarrow 4$ con costo $y_4 = 1$ y de 1 a 5 es $1 \longrightarrow 3 \longrightarrow 2 \longrightarrow 5$ con costo $y_5 = -3$.

Ejemplo 8.8. Aplicando el algoritmo de Ford-Bellman al grafo

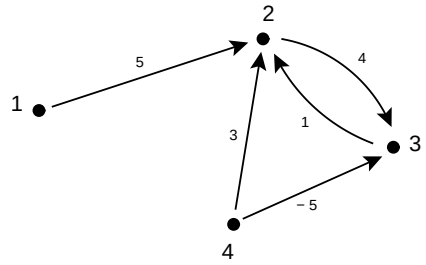


para encontrar el camino óptimo de $s = 1$ a cada uno de los restantes vértices obtenemos la siguiente tabla que muestra los valores de y_v y de p_v para cada vértice v.

	0	1	2	3	4
y_1	0	0	0	0	0
y_2	∞	1	1	1	0
y_3	∞	2	-2	-2	-2
y_4	∞	∞	3	-1	-4
y_5	∞	4	-2	-6	-6
p_1	-1	-1	-1	-1	-1
p_2	-1	1	1	1	4
p_3	-1	1	2	2	2
p_4	-1	-1	3	3	5
p_5	-1	1	3	3	3

Ahora analizamos si se verifica que $y_v \leq y_u + c_{uv}\ \forall (u,v) \in E$.
En este caso vemos que esto no vale. En efecto, $y_3 = -2 > -3 = y_2 + c_{23}$. Esto significa que el grafo contiene un ciclo dirigido de costo negativo. En efecto, el ciclo $2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 2$ es un ciclo dirigido de costo -1.

Ejemplo 8.9. Aplicando el algoritmo de Ford-Bellman al grafo



para encontrar el camino óptimo de $s = 1$ a cada uno de los restantes vértices obtenemos la siguiente tabla que muestra los valores de y_v y de p_v para cada vértice v.

	0	1	2	3
y_1	0	0	0	0
y_2	∞	5	5	5
y_3	∞	∞	9	9
y_4	∞	∞	∞	∞
p_1	-1	-1	-1	-1
p_2	-1	1	1	1
p_3	-1	-1	2	2
p_4	-1	-1	-1	-1

En este caso al terminar el algoritmo $y_4 = \infty$ ya que no existe camino de 1 a 4.

Observación 8.10. Dado v, si en la k-ésima iteración del algoritmo el valor de y_v es finito entonces existe un camino de s a v cuyo costo es y_v. Esto se debe a que, cada vez que el algoritmo actualiza el valor de y_v, lo reemplaza por el costo de un camino de s a v. Notar que esto seguiría valiendo si el algoritmo, en lugar de detenerse cuando $i = \#V$, continuara haciendo cualquier cantidad de iteraciones.

Validez del algoritmo.

Sea $m = \#V$. Para cada $0 \leq r \leq m-1$ denotemos por $y_v^{(r)}$ el valor de y_v al terminar la r-ésima iteración del algoritmo.

La validez del algoritmo resultará como corolario del siguiente

Teorema 8.11. Sea $k \leq m-1$. Si existe un camino óptimo de s a v con a lo sumo k ramas entonces $y_v^{(k)}$ es el costo de ese camino.

Demostración: Si $k=1$, supongamos que $\mathcal{P}=(s,v)$ es un camino óptimo con una sola rama. Entonces, en la primera iteración resulta que $y_v^{(0)} > y_u^{(0)} + c_{uv}$ sii $u=s$ porque $y_v^{(0)}=\infty$ e

$$y_u^{(0)} = \begin{cases} 0 & \text{si } u=s \\ \infty & \text{si } u \neq s \end{cases}$$

Luego, al terminar la primera iteración tendremos que $y_v^{(1)} = c_{sv} =$ costo de $\mathcal{P}$.

Supongamos que el teorema vale para k y sea v un vértice tal que existe un camino óptimo $\mathcal{R}$ de s a v con a lo sumo $k+1$ ramas. Sea u el vértice anterior a v en ese camino. Sea $\mathcal{C}$ el camino de s a u que resulta de suprimir en $\mathcal{R}$ la rama (u,v). Entonces, por la observación 8.2. se tiene que $\mathcal{C}$ es un camino óptimo de s a u con a lo sumo k ramas. Luego, por hipótesis inductiva, $y_u^{(k)} =$ costo de $\mathcal{C}$.
Luego, el costo de $\mathcal{R} = y_u^{(k)} + c_{uv}$. Esto significa que $y_u^{(k)} + c_{uv}$ es el costo de un camino óptimo de s a v. Como $y_v^{(k)}$ es el costo de un camino de s a v o es infinito entonces

$$y_v^{(k)} \geq y_u^{(k)} + c_{uv} \qquad (1)$$

y como $y_w^{(k)} + c_{wv}$ también es el costo de un camino de s a v para todo w tal que $(w,v) \in E$ entonces se tiene que

$$y_u^{(k)} + c_{uv} \leq y_w^{(k)} + c_{wv} \quad \forall\, w \,/\, (w,v) \in E \qquad (2)$$

Si en (1) valiera la igualdad, entonces de (2) resulta que $y_v^{(k)} \leq y_w^{(k)} + c_{wv} \quad \forall\, w \,/\, (w,v) \in E$. Por lo tanto no existe w tal que $(w,v) \in E$ e $y_v^{(k)} > y_w^{(k)} + c_{wv}$. En ese caso, en la iteración $k+1$ pondremos $y_v^{(k+1)} = y_v^{(k)} = y_u^{(k)} + c_{uv} =$ costo de $\mathcal{R}$.
Si en cambio en (1) no valiera la igualdad, entonces $(u,v) \in E$ y vale $y_v^{(k)} > y_u^{(k)} + c_{uv}$. Por lo tanto en la iteración $k+1$ pondremos

$$y_v^{(k+1)} = \min\{y_w^{(k)} + c_{wv} \,/\, w \text{ satisface } (w,v) \in E \text{ e } y_v^{(k)} > y_w^{(k)} + c_{wv}\}$$

Pero de (2) resulta que $\min\{y_w^{(k)} + c_{wv} \,/\, w \text{ satisface } (w,v) \in E \text{ e } y_v^{(k)} > y_w^{(k)} + c_{wv}\} = y_u^{(k)} + c_{uv}$, de donde también en este caso es $y_v^{(k+1)} = y_u^{(k)} + c_{uv} =$ costo de $\mathcal{R}$. □

Corolario 8.12. Al terminar el algoritmo se verifican

i) Si $y_w < \infty$ para todo w entonces el grafo no contiene ciclos dirigidos de costo negativo sii $y_v \leq y_u + c_{uv}$ $\forall (u,v) \in E$. En tal caso, para cada v se tiene que p_v es el vértice predecesor a v en un camino óptimo de s a v cuyo costo es y_v.
ii) $y_v = \infty$ sii no existe un camino dirigido de s a v.

Demostración: i) Supongamos que para todo v se satisface

$$y_v \leq y_u + c_{uv} \; \forall (u,v) \in E$$

Si el grafo contiene algún ciclo dirigido $\mathcal{C}$ de costo negativo, sean (u_1,u_2),(u_2,u_3), $\ldots$, (u_{n-1},u_n), (u_n,u_1) las ramas de ese ciclo.
Entonces

$$\begin{aligned} y_{u_2} &\leq y_{u_1} + c_{u_1u_2} \\ y_{u_3} &\leq y_{u_2} + c_{u_2u_3} \\ &\vdots \\ y_{u_n} &\leq y_{u_{n-1}} + c_{u_{n-1}u_n} \\ y_{u_1} &\leq y_{u_n} + c_{u_nu_1} \end{aligned}$$

de donde

$$\begin{aligned} y_{u_1} &\leq y_{u_n} + c_{u_n u_1} \\ &\leq y_{u_{n-1}} + c_{u_{n-1} u_n} + c_{u_n u_1} \\ &\leq \ldots \leq y_{u_2} + c_{u_2 u_3} + \cdots + c_{u_{n-1} u_n} + c_{u_n u_1} \\ &\leq y_{u_1} + c_{u_1 u_2} + c_{u_2 u_3} + \cdots + c_{u_{n-1} u_n} + c_{u_n u_1} \end{aligned}$$

Luego $y_{u_1} \leq y_{u_1} + c_{u_1 u_2} + c_{u_2 u_3} + \cdots + c_{u_{n-1} u_n} + c_{u_n u_1}$ y por lo tanto, como $y_{u_1} < \infty$, $0 \leq c_{u_1 u_2} + c_{u_2 u_3} + \cdots + c_{u_{n-1} u_n} + c_{u_n u_1} =$ costo de $\mathcal{C}$, lo que contradice que el costo de $\mathcal{C}$ era negativo. Luego el grafo no contiene ciclos dirigidos de costo negativo.

Recíprocamente, supongamos que el grafo no contiene ciclos dirigidos de costo negativo. Dado v, como $y_v < \infty$ entonces y_v es el costo de un camino a v (ver observación 8.10.). Como el grafo no contiene ciclos dirigidos de costo negativo entonces existe un camino óptimo de s a v. Además, podemos suponer que este camino es simple ya que si contiene un ciclo de costo nulo entonces el camino simple evitando el ciclo también es óptimo (y un camino óptimo no puede contener un ciclo de costo positivo ya que el camino evitando el ciclo sería más "barato"). Como un camino simple no puede tener más de $m-1$ ramas entonces existe un camino óptimo $\mathcal{P}$ a v con a lo sumo $m-1$ ramas. Como el algoritmo realiza $m-1$ iteraciones, por el teorema 8.11., el valor de y_v al terminar el algoritmo es $y_v^{(m-1)} =$ costo de $\mathcal{P}$. Por lo tanto, y_v es el costo de un camino óptimo. Luego, como $y_u + c_{uv}$ es el costo de un camino a v entonces debe ser $y_v \leq y_u + c_{uv}$ $\forall (u,v) \in E$.

Dejamos como tarea al lector completar la demostración verificando que p_v es el vértice anterior a v en un camino óptimo cuyo costo es y_v y probando ii). □

Observación 8.13. En el caso particular en que $c_{uv} = 1$ para todo (u,v) resulta que camino óptimo significa camino más corto e y_v resulta ser el mínimo número de ramas que tiene un camino de s a v.

Complejidad del algoritmo de Ford-Bellman.
Sean $m = \#V$ y $n = \#E$. Como se realizan $m-1$ iteraciones y en cada una de ellas se hacen a lo sumo $c.n$ operaciones (donde c es una constante), este algoritmo es polinomial con complejidad $O(m.n)$.

9. Ciclos dirigidos de costo negativo.

En la sección anterior hemos visto que podemos determinar si G contiene ciclos dirigidos de costo negativo utilizando el algoritmo de Ford-Bellman. Veremos ahora cómo podemos hallar efectivamente un tal ciclo, modificando convenientemente este algoritmo.

Lema 9.1. Sea $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ un grafo dirigido donde cada rama $e \in \mathcal{E}$ tiene asignado un costo $\overline{c}_e$ y sea $C = \max\{|\,\overline{c}_e|\,/\,e \in \mathcal{E}\}$. Si $\mathcal{P}$ es un camino dirigido en $\mathcal{G}$ que no contiene ciclos dirigidos de costo negativo entonces $c(\mathcal{P}) \geq -(\#\mathcal{V} - 1)C$.

Demostración: Supongamos primero que $\mathcal{P}$ no contiene ciclos dirigidos. Entonces $\mathcal{P}$ es de la forma

donde $u_1, u_2, \ldots, u_k$ son todos vértices distintos. Luego, $k \leq \#\mathcal{V}$ y, como $\overline{c}_e \geq -C$ para toda rama e entonces

$$c(\mathcal{P}) = \overline{c}_{u_1 u_2} + \overline{c}_{u_2 u_3} + \cdots + \overline{c}_{u_{k-1} u_k} \geq \underbrace{-C - C - \ldots - C}_{k-1} \geq -(k-1)C \geq -(\#\mathcal{V} - 1)C$$

Supongamos ahora que $\mathcal{P}$ contiene algunos ciclos dirigidos $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_r$. Entonces esos ciclos deben tener costo mayor o igual que cero. Sea $\mathcal{P}'$ el camino simple que resulta de eliminar en $\mathcal{P}$ estos ciclos. Luego,

$$c(\mathcal{P}) = c(\mathcal{P}') + c(\mathcal{C}_1) + c(\mathcal{C}_2) + \cdots + c(\mathcal{C}_r) \geq c(\mathcal{P}')$$

y como $\mathcal{P}'$ es un camino que no contiene ciclos entonces $c(\mathcal{P}') \geq -(\#\mathcal{V}-1)C$, de donde resulta que $c(\mathcal{P}) \geq -(\#\mathcal{V}-1)C$. ▫

Sea $G = (V, E)$ un grafo dirigido donde cada rama $e \in E$ tiene asignado un costo c_e. Describiremos un algoritmo que halla, cuando existe, un ciclo dirigido de costo negativo en G. Sea $m = \#V$ y sea $C = \max\{|\, c_e \,| \,/\, e \in E\}$.
A partir de G construímos el nuevo grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ agregando a G un vértice s y ramas (s, v) para cada $v \in V$. A cada rama $e \in \mathcal{E}$ le asignamos un costo $\overline{c}_e$ en la forma

$$\overline{c}_e = \begin{cases} c_e & \text{si } e \in E \\ 0 & \text{si } e = (s, v) \text{ para algún } v \in V \end{cases}$$

Notar que $\#\mathcal{V} = m + 1$ y que $\max\{|\, \overline{c}_e \,| \,/\, e \in \mathcal{E}\} = C$. Notar además que para todo $v \in \mathcal{V}$, $v \neq s$, existe un camino de s a v (la rama (s, v)).

Observación 9.2. $\mathcal{C}$ es un ciclo dirigido de costo negativo en $\mathcal{G}$ si y sólo si $\mathcal{C}$ es un ciclo dirigido de costo negativo en G. En efecto, sea $\mathcal{C}$ un ciclo dirigido de costo negativo en $\mathcal{G}$. Como ningún ciclo de $\mathcal{G}$ puede pasar por el vértice s, ya que de s sólo salen flechas, entonces ese ciclo debe ser un ciclo en G, es decir, $e \in E$ para toda rama e de $\mathcal{C}$. Luego, $\overline{c}_e = c_e$ para toda rama e de $\mathcal{C}$. Por lo tanto, $\mathcal{C}$ es un ciclo dirigido de costo negativo en G. Recíprocamente, si $\mathcal{C}$ es un ciclo dirigido de costo negativo en G entonces claramente es un ciclo dirigido en $\mathcal{G}$ del mismo costo.

Para hallar un ciclo dirigido de costo negativo en G, primero aplicamos el algoritmo de Ford-Bellman para hallar un camino de mínimo costo en $\mathcal{G}$ de s a cada $v \in V$.
Como este algoritmo realiza $\#\mathcal{V} - 1 = m$ iteraciones entonces, por el corolario 8.12., se tiene que al terminar la iteración m-ésima $y_v < \infty$ para todo v (ya que para todo $v \in V$ existe un camino de s a v) y además, $y_v \leq y_u + \overline{c}_{uv}\ \forall (u, v) \in \mathcal{E}$ si y sólo si $\mathcal{G}$ no contiene ciclos dirigidos de costo negativo.
Luego, por la observación 9.2., al terminar la iteración m-ésima se tiene que $y_v \leq y_u + \overline{c}_{uv}\ \forall (u, v) \in \mathcal{E}$ si y sólo si G no contiene ciclos dirigidos de costo negativo. En ese caso el algoritmo termina. En caso contrario, continuamos con el algoritmo (es decir, seguimos haciendo iteraciones) hasta que para algún v valga $y_v < -mC = -(\#\mathcal{V}-1)C$.
Como $y_v < \infty$ entonces, por la observación 8.10., y_v es el costo de un camino $\mathcal{P}$ en $\mathcal{G}$.
Luego $c(\mathcal{P}) = y_v < -(\#\mathcal{V}-1)C$ y, como $C = \max\{|\, \overline{c}_e \,| \,/\, e \in \mathcal{E}\}$, $\mathcal{P}$ debe contener un ciclo dirigido de costo negativo por el lema 9.1. Por la observación 9.2., ese ciclo debe ser un ciclo dirigido de costo negativo en G. Para hallarlo, reconstruímos el camino usando los predecesores.

Descripción del algoritmo.

1. A partir de G construímos el nuevo grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ agregando a G un vértice s y ramas (s, v) para cada $v \in V$. A cada rama $e \in \mathcal{E}$ le asignamos un costo $\overline{c}_e$ en la forma

$$\overline{c}_e = \begin{cases} c_e & \text{si } e \in E \\ 0 & \text{si } e = (s, v) \text{ para algún } v \in V \end{cases}$$

2. Poner

$$y_u = \begin{cases} 0 & \text{si } u = s \\ \infty & \text{si } u \neq s \end{cases}, \qquad p_u = -1, \qquad i = 1$$

3. Para cada v poner $y'_v = y_v$, si $y_v \leq y_u + \overline{c}_{uv}$ para todo u tal que $(u, v) \in \mathcal{E}$ o poner $y'_v = y_w + \overline{c}_{wv}$ y $p_v = w$, donde $y_w + \overline{c}_{wv} = \min\{y_u + \overline{c}_{uv} \,/\, (u, v) \in \mathcal{E} \text{ e } y_v > y_u + \overline{c}_{uv}\}$, si $y_v > y_u + \overline{c}_{uv}$ para algún u tal que $(u, v) \in \mathcal{E}$

4. $y_v = y'_v$. Poner $i = i + 1$

5. Si $i < m+1$ ir a 3.

6. Si $i = m+1$ y vale $y_v \leq y_u + \overline{c}_{uv}\ \forall (u,v) \in \mathcal{E}$ STOP (en ese caso G no contiene ciclos dirigidos de costo negativo).

7. Si $y_v \geq -mC$ para todo v ir a 3. Si no, hallar v tal que $y_v < -mC$ y reconstruír el camino $\mathcal{P}$ cuyo costo es y_v usando los predecesores.

8. Hallar un ciclo dirigido de costo negativo contenido en $\mathcal{P}$. STOP.

Notemos que, en principio, este algoritmo podría no detenerse nunca. El objetivo de lo que sigue es dar condiciones que garantizan un STOP (proposición 9.5.) y mostrar que, en realidad, estas condiciones no son realmente restrictivas en la práctica (observación 9.6.).

Observación 9.3. Cualquiera sea v, al terminar la primera iteración del algoritmo el valor de y_v presente en ese momento satisface $y_v \leq 0$. En efecto como $(s,v) \in \mathcal{E}$ entonces en el paso 3. de la primera iteración se tiene que $y_v > y_s + \overline{c}_{sv}$ ya que $y_v = \infty$ e $y_s = 0 = \overline{c}_{sv}$. Luego, al terminar la primera iteración el valor de y_v es reemplazado por $\min\{y_u + \overline{c}_{uv} \,/\, (u,v) \in \mathcal{E} \text{ e } y_v > y_u + \overline{c}_{uv}\} \leq y_s + \overline{c}_{sv} = 0$. En particular, el valor de y_v presente al terminar la primera iteración es finito. Por lo tanto, como en cada iteración el valor de y_v no cambia o se reemplaza por algo menor entonces al terminar cualquier iteración del algoritmo, el valor de y_v presente en ese momento es finito para todo v.

Lema 9.4. Si al terminar alguna iteración del algoritmo se tiene que $y_v \leq y_u + \overline{c}_{uv}$ para todo $(u,v) \in \mathcal{E}$, entonces $\mathcal{G}$ no contiene ciclos dirigidos de costo negativo.

Demostración: Supongamos que al terminar alguna iteración del algoritmo se tenga que $y_v \leq y_u + \overline{c}_{uv}$ para todo $(u,v) \in \mathcal{E}$.
Si $\mathcal{C}$ es un ciclo en $\mathcal{G}$ y (u_1,u_2),(u_2,u_3), ... , (u_{n-1},u_n), (u_n,u_1) son sus ramas entonces

$$\begin{aligned} y_{u_2} &\leq y_{u_1} + \overline{c}_{u_1u_2} \\ y_{u_3} &\leq y_{u_2} + \overline{c}_{u_2u_3} \\ &\vdots \\ y_{u_n} &\leq y_{u_{n-1}} + \overline{c}_{u_{n-1}u_n} \\ y_{u_1} &\leq y_{u_n} + \overline{c}_{u_nu_1} \end{aligned}$$

de donde

$$\begin{aligned} y_{u_1} &\leq y_{u_n} + \overline{c}_{u_nu_1} \\ &\leq y_{u_{n-1}} + \overline{c}_{u_{n-1}u_n} + \overline{c}_{u_nu_1} \\ &\leq \ldots \leq y_{u_2} + \overline{c}_{u_2u_3} + \cdots + \overline{c}_{u_{n-1}u_n} + \overline{c}_{u_nu_1} \\ &\leq y_{u_1} + \overline{c}_{u_1u_2} + \overline{c}_{u_2u_3} + \cdots + \overline{c}_{u_{n-1}u_n} + \overline{c}_{u_nu_1} \end{aligned}$$

Luego $y_{u_1} \leq y_{u_1} + \overline{c}_{u_1u_2} + \overline{c}_{u_2u_3} + \cdots + \overline{c}_{u_{n-1}u_n} + \overline{c}_{u_nu_1}$ y por lo tanto, como $y_{u_1} < \infty$ por la observación 9.3., $0 \leq \overline{c}_{u_1u_2} + \overline{c}_{u_2u_3} + \cdots + \overline{c}_{u_{n-1}u_n} + \overline{c}_{u_nu_1} =$ costo de $\mathcal{C}$. □

Proposición 9.5. Si c_e es entero para todo $e \in E$ entonces valen
i) si G no contiene ciclos dirigidos de costo negativo entonces el algoritmo termina en la iteración m-ésima.
ii) si G contiene ciclos dirigidos de costo negativo entonces el algoritmo termina en a lo sumo $2 + m^2C$ iteraciones.

Demostración: i) Si G no contiene ciclos dirigidos de costo negativo entonces, al terminar la iteración m-ésima vale $y_v \leq y_u + c_{uv}\ \forall (u,v) \in \mathcal{E}$ y por lo tanto el algoritmo se detiene.

ii) Supongamos ahora que G contiene ciclos dirigidos de costo negativo. Luego, por la observación 9.2., $\mathcal{G}$ también. Para cada v denotemos por $y_v^{(k)}$ al valor que tiene y_v al terminar la k-ésima iteración.

Por el lema 9.4., cualquiera sea k, existe $(u,v) \in \mathcal{E}$ tal que $y_v^{(k)} > y_u^{(k)} + \overline{c}_{uv}$. Por lo tanto, en la siguiente iteración $y_v^{(k)}$ será reemplazado por $y_w^{(k)} + \overline{c}_{wv}$ para algún w tal que $y_w^{(k)} + \overline{c}_{wv} < y_v^{(k)}$. Esto muestra que, para todo k, existe v tal que $y_v^{(k+1)} < y_v^{(k)}$.

Afirmación: $\sum_v y_v^{(1)} \leq 0$ y, para todo k, $\sum_v y_v^{(k+1)} \leq \sum_v y_v^{(k)} - 1$.

Demostración: Por la observación 9.3., se tiene que $y_v^{(1)} \leq 0$ para todo v, de donde resulta que $\sum_v y_v^{(1)} \leq 0$.

Por otra parte, también por la observación 9.3., cualquiera sea r se verifica que $y_v^{(r)} < \infty$ para todo v. Luego, por la observación 8.10., para todo v se tiene que $y_v^{(r)}$ es el costo de un camino de s a v en $\mathcal{G}$ y por lo tanto es entero ya que $\overline{c}_e$ es entero para todo $e \in \mathcal{E}$. Luego, $\sum_v y_v^{(r)}$ es entero para todo r. Esto muestra que, cualquiera sea k, $\sum_v y_v^{(k+1)}$ y $\sum_v y_v^{(k)}$ son enteros. Como $y_v^{(k+1)} \leq y_v^{(k)}$ para todo v (ya que en cada iteración el valor de y_v no cambia o se reemplaza por algo menor) y además, para algún v se verifica $y_v^{(k+1)} < y_v^{(k)}$ entonces se tiene que $\sum_v y_v^{(k+1)} < \sum_v y_v^{(k)}$. Por lo tanto, siendo ambos miembros de la desigualdad números enteros, resutla que $\sum_v y_v^{(k+1)} \leq \sum_v y_v^{(k)} - 1$ tal como habíamos afirmado.

Probemos ahora que el algoritmo termina en a lo sumo $2 + m^2C$ iteraciones. Se tiene

$$\begin{aligned}
\sum_v y_v^{(1)} &\leq 0 \\
\sum_v y_v^{(2)} &\leq \sum_v y_v^{(1)} - 1 \leq -1 \\
\sum_v y_v^{(3)} &\leq \sum_v y_v^{(2)} - 1 \leq -2 \\
\vdots \qquad &\qquad \vdots \qquad\qquad \vdots \\
\sum_v y_v^{(k)} &\leq \sum_v y_v^{(k-1)} - 1 \leq -(k-1)
\end{aligned}$$

Por lo tanto, para $k = 2 + m^2C$ se tiene que $\sum_v y_v^{(k)} \leq -(1 + m^2C) = -1 - m^2C < -m^2C$. Si fuera $y_v^{(k)} \geq -mC$ para todo v entonces se tendría que $\sum_v y_v^{(k)} \geq -m^2C$. Luego, para $k = 2 + m^2C$, existe v tal que $y_v^{(k)} < -mC$ y por lo tanto el algoritmo termina a lo sumo en esa iteración. □

Observación 9.6. En la práctica, los costos no siempre son enteros. Sin embargo, siempre son racionales. Veamos que siempre podemos utilizar este algoritmo para hallar un ciclo dirigido de costo negativo en G. En efecto, si c_e es un número racional para todo $e \in E$, entonces podemos hallar a entero, $a > 0$, tal que $c'_e = a.c_e$ sea entero para todo $e \in E$. Por lo tanto basta resolver el problema equivalente de hallar un ciclo dirigido de costo negativo en G utilizando c'_e como costo de cada rama.