Algoritmo de Kruskal

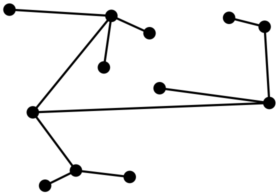
Curso de Teoría Algebraica de Grafos

Facultad de Ingeniería
Universidad de la República

14 de mayo de 2012

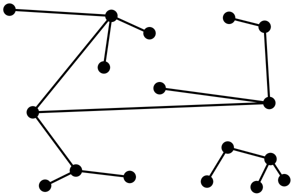
Árboles

- Un árbol es un grafo conexo y acíclico (sin ciclos).



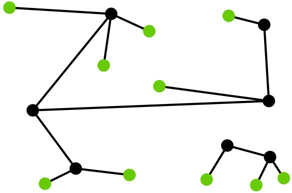
Árboles

- Un árbol es un grafo conexo y acíclico (sin ciclos).
- Un bosque es un grafo acíclico, o sea, una unión disjunta de árboles.



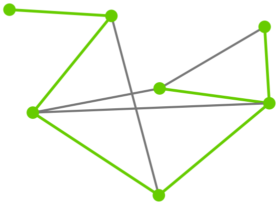
Árboles

- Un árbol es un grafo conexo y acíclico (sin ciclos).
- Un bosque es un grafo acíclico, o sea, una unión disjunta de árboles.
- Una hoja en un grafo es un vértice de grado 1.



Árboles

- Un árbol es un grafo conexo y acíclico (sin ciclos).
- Un bosque es un grafo acíclico, o sea, una unión disjunta de árboles.
- Una hoja en un grafo es un vértice de grado 1.
- Un árbol generador de un grafo G es un subgrafo generador de G que es un árbol.



Lema 1

Si $m_G > n_G - 1$ entonces G tiene un ciclo.

Lema 1

Si $m_G > n_G - 1$ entonces G tiene un ciclo.

Demo: Por inducción. Si $n_G = 1$ o 2, no puede pasar. Si $n_G = 3$, entonces G es un triángulo y tiene un ciclo.

Lema 1

Si $m_G > n_G - 1$ entonces G tiene un ciclo.

Demo: Por inducción. Si $n_G = 1$ o 2, no puede pasar. Si $n_G = 3$, entonces G es un triángulo y tiene un ciclo.
Sea G con $n_G > 3$ y $m_G > n_G - 1$. Si todo vértice de G tiene grado al menos 2, entonces G tiene un ciclo (ejercicio).

Lema 1

Si $m_G > n_G - 1$ entonces G tiene un ciclo.

Demo: Por inducción. Si $n_G = 1$ o 2, no puede pasar. Si $n_G = 3$, entonces G es un triángulo y tiene un ciclo.
Sea G con $n_G > 3$ y $m_G > n_G - 1$. Si todo vértice de G tiene grado al menos 2, entonces G tiene un ciclo (ejercicio).
Si no, saco un vértice v con $\mathrm{gr}(v) \leq 1$. Ahora $G' = G - v$ cumple

$$m_{G'} \geq m_G - 1 > n_G - 2 = n_{G'} - 1$$

Lema 1

Si $m_G > n_G - 1$ entonces G tiene un ciclo.

Demo: Por inducción. Si $n_G = 1$ o 2, no puede pasar. Si $n_G = 3$, entonces G es un triángulo y tiene un ciclo.
Sea G con $n_G > 3$ y $m_G > n_G - 1$. Si todo vértice de G tiene grado al menos 2, entonces G tiene un ciclo (ejercicio).
Si no, saco un vértice v con $\mathrm{gr}(v) \leq 1$. Ahora $G' = G - v$ cumple

$$m_{G'} \geq m_G - 1 > n_G - 2 = n_{G'} - 1$$

Por hipótesis inductiva G' contiene un ciclo, y como G' es un subgrafo de G, es también un ciclo en G. $\square$

Lema 2

Si G es conexo, entonces $m_G \geq n_G - 1$.

Lema 2

Si G es conexo, entonces $m_G \geq n_G - 1$.

Demo: Por inducción. Si $n_G = 1$ o 2, se verifica.

Lema 2

Si G es conexo, entonces $m_G \geq n_G - 1$.

Demo: Por inducción. Si $n_G = 1$ o 2, se verifica.
Sea G conexo, $n_G \geq 3$. Si todo vértice de G tiene grado al menos 2, entonces

$$2m_G = \sum_{v \in G} \mathrm{gr}(v) \geq 2 \sum_{v \in G} 1 = 2n_G$$

y $2m_G \geq 2n_G$, implica $m_G \geq n_G - 1$.

Lema 2

Si G es conexo, entonces $m_G \geq n_G - 1$.

Demo: Por inducción. Si $n_G = 1$ o 2, se verifica.
Sea G conexo, $n_G \geq 3$. Si todo vértice de G tiene grado al menos 2, entonces

$$2m_G = \sum_{v \in G} \mathrm{gr}(v) \geq 2 \sum_{v \in G} 1 = 2n_G$$

y $2m_G \geq 2n_G$, implica $m_G \geq n_G - 1$.
Si no, sea v de grado 1 (no puede haber vértices de grado cero por ser G conexo no trivial). Como v no puede ser punto de corte, $G' = G - v$ es conexo.

Lema 2

Si G es conexo, entonces $m_G \geq n_G - 1$.

Demo: Por inducción. Si $n_G = 1$ o 2, se verifica.
Sea G conexo, $n_G \geq 3$. Si todo vértice de G tiene grado al menos 2, entonces

$$2m_G = \sum_{v \in G} \mathrm{gr}(v) \geq 2 \sum_{v \in G} 1 = 2n_G$$

y $2m_G \geq 2n_G$, implica $m_G \geq n_G - 1$.
Si no, sea v de grado 1 (no puede haber vértices de grado cero por ser G conexo no trivial). Como v no puede ser punto de corte, $G' = G - v$ es conexo.
Por hipótesis inductiva $m_{G'} \geq n_{G'} - 1$. Entonces

$$m_G = m_{G'} + 1 \geq n_{G'} = n_G - 1$$

$\square$

Teorema

Son equivalentes:

1. G es un árbol.
2. Todo par de vértices de G está unido por un único camino.
3. G es conexo y $m_G = n_G - 1$.
4. G es acíclico y $m_G = n_G - 1$.

G es un árbol $\Leftrightarrow$ todo par de vértices de G está unido por un único camino

G es un árbol $\Leftrightarrow$ todo par de vértices de G está unido por un único camino

Demo: ($\Leftarrow$) Si todo par de vértices de G está unido por un único camino, claramente G es conexo. Además, si hubiera un ciclo, cualquier par de vértices del ciclo estaría unido por al menos dos caminos distintos. Luego G es un árbol.

G es un árbol $\Leftrightarrow$ todo par de vértices de G está unido por un único camino

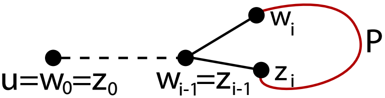
Demo: ($\Leftarrow$) Si todo par de vértices de G está unido por un único camino, claramente G es conexo. Además, si hubiera un ciclo, cualquier par de vértices del ciclo estaría unido por al menos dos caminos distintos. Luego G es un árbol.

($\Rightarrow$) Si G es un árbol, es conexo, luego todo par de vértices está unido por al menos un camino. Supongamos que u y v están unidos por al menos dos caminos distintos, $P_1 : u = w_0, w_1, \ldots, w_k = v$ y $P_2 : u = z_0, z_1, \ldots, z_r = v$. Sea i el primer índice tal que $w_i \neq z_i$. Entonces $i > 0$, y $w_{i-1} = z_{i-1}$.

G es un árbol $\Leftrightarrow$ todo par de vértices de G está unido por un único camino

Demo: ($\Leftarrow$) Si todo par de vértices de G está unido por un único camino, claramente G es conexo. Además, si hubiera un ciclo, cualquier par de vértices del ciclo estaría unido por al menos dos caminos distintos. Luego G es un árbol.

($\Rightarrow$) Si G es un árbol, es conexo, luego todo par de vértices está unido por al menos un camino. Supongamos que u y v están unidos por al menos dos caminos distintos, $P_1 : u = w_0, w_1, \ldots, w_k = v$ y $P_2 : u = z_0, z_1, \ldots, z_r = v$. Sea i el primer índice tal que $w_i \neq z_i$. Entonces $i > 0$, y $w_{i-1} = z_{i-1}$.



Además, $w_i, \ldots, w_k = v = z_r, \ldots, z_i$ inducen en G un subgrafo conexo. Sea P un camino mínimo entre w_i y z_i en ese subgrafo inducido. Entonces $w_{i-1}Pw_{i-1}$ es un ciclo en G, absurdo. $\square$

$$G \text{ es un árbol} \Leftrightarrow G \text{ es conexo y } m_G = n_G - 1$$

$$G \text{ es un árbol} \Leftrightarrow G \text{ es conexo y } m_G = n_G - 1$$

Demo: ($\Rightarrow$) Si G es un árbol entonces es conexo y por Lema 2, $m_G \geq n_G - 1$. Además, como es acíclico, por Lema 1, $m_G \leq n_G - 1$. Luego $m_G = n_G - 1$.

G es un árbol $\Leftrightarrow$ G es conexo y $m_G = n_G - 1$

Demo: ($\Rightarrow$) Si G es un árbol entonces es conexo y por Lema 2, $m_G \geq n_G - 1$. Además, como es acíclico, por Lema 1, $m_G \leq n_G - 1$. Luego $m_G = n_G - 1$.

($\Leftarrow$) G es conexo, probemos por inducción que es un árbol. Si $n_G = 1$, vale. Supongamos $n_G > 1$. Por $2m_G = \sum_{v \in G} \mathrm{gr}(v)$, G tiene al menos un vértice v de grado menor o igual que uno. Como es conexo, v tiene grado 1, y entonces no es punto de corte. Luego $G' = G - v$ es conexo y

G es un árbol $\Leftrightarrow$ G es conexo y $m_G = n_G - 1$

Demo: $(\Rightarrow)$ Si G es un árbol entonces es conexo y por Lema 2, $m_G \geq n_G - 1$. Además, como es acíclico, por Lema 1, $m_G \leq n_G - 1$. Luego $m_G = n_G - 1$.

$(\Leftarrow)$ G es conexo, probemos por inducción que es un árbol. Si $n_G = 1$, vale. Supongamos $n_G > 1$. Por $2m_G = \sum_{v \in G} \mathrm{gr}(v)$, G tiene al menos un vértice v de grado menor o igual que uno. Como es conexo, v tiene grado 1, y entonces no es punto de corte. Luego $G' = G - v$ es conexo y

$$n_{G'} - 1 = n_G - 2 = m_G - 1 = m_{G'}.$$

Por hipótesis inductiva G' es un árbol, y entonces G era un árbol también (v tiene grado 1, no puede pertenecer a un ciclo). $\square$

G es un árbol $\Leftrightarrow$ G es acíclico y $m_G = n_G - 1$

$$G \text{ es un árbol} \Leftrightarrow G \text{ es acíclico y } m_G = n_G - 1$$

Demo: ($\Rightarrow$) Si G es un árbol entonces es acíclico y por Lema 1, $m_G \leq n_G - 1$. Además, como es conexo, por Lema 2, $m_G \geq n_G - 1$. Luego $m_G = n_G - 1$.

G es un árbol $\Leftrightarrow$ G es acíclico y $m_G = n_G - 1$

Demo: ($\Rightarrow$) Si G es un árbol entonces es acíclico y por Lema 1, $m_G \leq n_G - 1$. Además, como es conexo, por Lema 2, $m_G \geq n_G - 1$. Luego $m_G = n_G - 1$.

($\Leftarrow$) G es acíclico. Supongamos que tiene t componentes conexas $G_1, \ldots, G_t$. Cada una de ellas es un árbol, y por la equivalencia anterior, $m_{G_i} = n_{G_i} - 1$. Pero entonces

G es un árbol $\Leftrightarrow$ G es acíclico y $m_G = n_G - 1$

Demo: ($\Rightarrow$) Si G es un árbol entonces es acíclico y por Lema 1, $m_G \leq n_G - 1$. Además, como es conexo, por Lema 2, $m_G \geq n_G - 1$. Luego $m_G = n_G - 1$.

($\Leftarrow$) G es acíclico. Supongamos que tiene t componentes conexas $G_1, \ldots, G_t$. Cada una de ellas es un árbol, y por la equivalencia anterior, $m_{G_i} = n_{G_i} - 1$. Pero entonces

$$m_G = \sum_{i=1}^{t} m_{G_i} = \sum_{i=1}^{t} (n_{G_i} - 1) = \sum_{i=1}^{t} n_{G_i} - t = n_G - t.$$

Luego $t = 1$, por lo tanto G es conexo. $\square$

Teorema

Todo árbol no trivial tiene al menos dos hojas.

Teorema

Todo árbol no trivial tiene al menos dos hojas.

Demo: Si T es un árbol no trivial, por ser conexo tiene una arista v_0w_0. O v_0 es una hoja, o puedo elegir un vecino v_1 de v_0, tal que $v_1 \neq w_0$. En cada paso, o v_i es una hoja o tiene un vecino distinto de v_{i-1} y también distinto del resto de los vértices del camino, porque T es acíclico. Como los vértices son finitos, hay algún v_k que es una hoja. Con el mismo argumento a partir de w_0, hay algún w_t que es una hoja, y es distinto de todos los v_i. $\square$

Teorema

1. Toda arista de un árbol es un puente.
2. Un vértice de un árbol no trivial es un punto de corte si y sólo si no es una hoja.

Teorema

1. Toda arista de un árbol es un puente.
2. Un vértice de un árbol no trivial es un punto de corte si y sólo si no es una hoja.

Demo: 1. Sea T un árbol y vw una arista de T. Como en T hay un único camino entre v y w (es v-e-w), no existe camino que una v y w en $T - e$.

Teorema

1. Toda arista de un árbol es un puente.
2. Un vértice de un árbol no trivial es un punto de corte si y sólo si no es una hoja.

Demo: 1. Sea T un árbol y vw una arista de T. Como en T hay un único camino entre v y w (es v-e-w), no existe camino que una v y w en $T - e$.
2. En cualquier grafo, una hoja nunca puede ser punto de corte. Sea T un árbol no trivial y v vértice de T que no es hoja. Entonces tiene al menos dos vecinos, z y w. Como en T existe un único camino que une a z y w (es zvw), no existe camino entre z y w en $T - v$. $\square$

Teorema

Un grafo es conexo si y sólo si admite un árbol generador.

Teorema

Un grafo es conexo si y sólo si admite un árbol generador.

Idea de demo: ($\Leftarrow$) Sea T un a.g. de G. Sean v, w vértices de G. En T existe un camino de v a w. Como T es subgrafo de G, en particular es un camino en G de v a w. Por lo tanto G es conexo.

Teorema

Un grafo es conexo si y sólo si admite un árbol generador.

Idea de demo: ($\Leftarrow$) Sea T un a.g. de G. Sean v, w vértices de G. En T existe un camino de v a w. Como T es subgrafo de G, en particular es un camino en G de v a w. Por lo tanto G es conexo.

($\Rightarrow$) Por inducción. Si G es trivial, G es un a.g. de si mismo. Si no, sea v un vértice de G. Por hipótesis inductiva, cada componente conexa G_i de $G - v$ tiene un a.g. T_i. Como G era conexo, v tiene un vecino v_i en cada G_i. Sea T el grafo obtenido agregando a la unión de los T_i el vértice v y las aristas $v_i v$. T es un a.g. de G: es fácil ver que es conexo y que $m_T = n_T - 1$, sabiendo que los T_i son conexos y $m_{T_i} = n_{T_i} - 1$. $\square$

Corolario

Todo grafo conexo no trivial tiene al menos dos vértices tales que al sacar alguno de ellos, sigue siendo conexo.

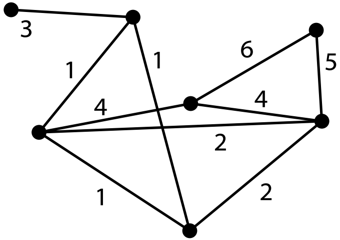
Corolario

Todo grafo conexo no trivial tiene al menos dos vértices tales que al sacar alguno de ellos, sigue siendo conexo.

Idea de demo: Si G es conexo, tiene un árbol generador T. Como G es no trivial, T también, y por lo tanto tiene al menos dos hojas v, w. Como $T - v$ y $T - w$ siguen siendo árboles, son árboles generadores de $G - v$ y $G - w$, respectivamente. Luego $G - v$ y $G - w$ son conexos. $\square$

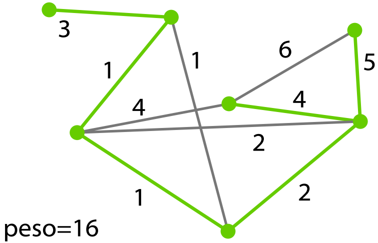
Árbol generador mínimo

- Un grafo pesado es un grafo tal que sus aristas tienen asociado un peso.



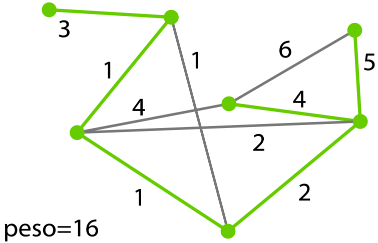
Árbol generador mínimo

- Un grafo pesado es un grafo tal que sus aristas tienen asociado un peso.
- El peso de un subgrafo es la suma de los pesos de sus aristas.



Árbol generador mínimo

- Un grafo pesado es un grafo tal que sus aristas tienen asociado un peso.
- El peso de un subgrafo es la suma de los pesos de sus aristas.
- Un árbol generador mínimo en un grafo pesado es un árbol generador de peso mínimo.



Los algoritmos de Kruskal y Prim

Los algoritmos de Kruskal y Prim

Hipótesis

Los algoritmos de Kuskal y Prim son algoritmos para grafos ponderados (con peso), no dirigidos, conexos y sin lazos.

Los algoritmos de Kruskal y Prim

Hipótesis

Los algoritmos de Kuskal y Prim son algoritmos para grafos ponderados (con peso), no dirigidos, conexos y sin lazos.

Si bien su aplicación es de la más diversa tenemos un ejemplo tipo que es el siguiente:

Los algoritmos de Kruskal y Prim

Hipótesis

Los algoritmos de Kuskal y Prim son algoritmos para grafos ponderados (con peso), no dirigidos, conexos y sin lazos.

Si bien su aplicación es de la más diversa tenemos un ejemplo tipo que es el siguiente:

Ejemplo

Tengo que construir caminos entre ciertos pares de ciudades, de modo que todo el país me quede conectado. ¿Cómo puedo hacerlo minimizando la longitud total del camino construido?

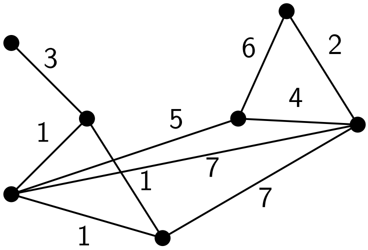
Algoritmo de Kruskal para AGM

Partir de un subgrafo generador cuyo conjunto de aristas es vacío, y en cada paso agregar una arista de peso mínimo que no forme ciclos con las demás aristas del conjunto, hasta haber agregado $n-1$ aristas.

Algoritmo de Kruskal para AGM

Partir de un subgrafo generador cuyo conjunto de aristas es vacío, y en cada paso agregar una arista de peso mínimo que no forme ciclos con las demás aristas del conjunto, hasta haber agregado $n-1$ aristas.

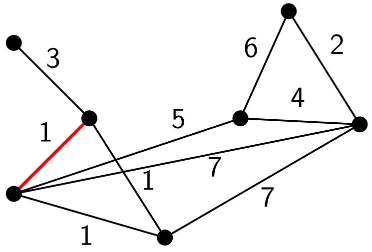
Ej:



Algoritmo de Kruskal para AGM

Partir de un subgrafo generador cuyo conjunto de aristas es vacío, y en cada paso agregar una arista de peso mínimo que no forme ciclos con las demás aristas del conjunto, hasta haber agregado $n-1$ aristas.

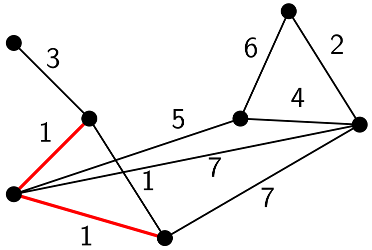
Ej:



Algoritmo de Kruskal para AGM

Partir de un subgrafo generador cuyo conjunto de aristas es vacío, y en cada paso agregar una arista de peso mínimo que no forme ciclos con las demás aristas del conjunto, hasta haber agregado $n-1$ aristas.

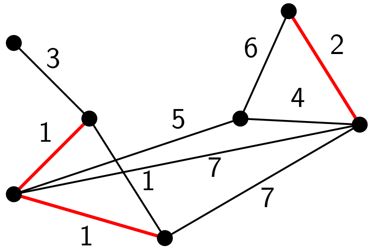
Ej:



Algoritmo de Kruskal para AGM

Partir de un subgrafo generador cuyo conjunto de aristas es vacío, y en cada paso agregar una arista de peso mínimo que no forme ciclos con las demás aristas del conjunto, hasta haber agregado $n-1$ aristas.

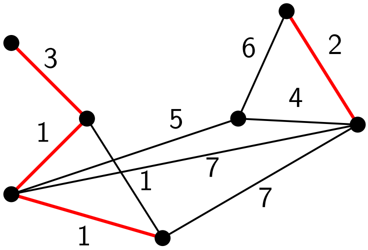
Ej:



Algoritmo de Kruskal para AGM

Partir de un subgrafo generador cuyo conjunto de aristas es vacío, y en cada paso agregar una arista de peso mínimo que no forme ciclos con las demás aristas del conjunto, hasta haber agregado $n-1$ aristas.

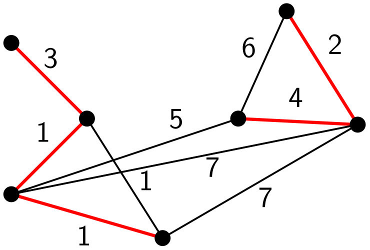
Ej:



Algoritmo de Kruskal para AGM

Partir de un subgrafo generador cuyo conjunto de aristas es vacío, y en cada paso agregar una arista de peso mínimo que no forme ciclos con las demás aristas del conjunto, hasta haber agregado $n - 1$ aristas.

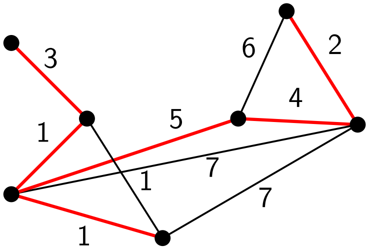
Ej:



Algoritmo de Kruskal para AGM

Partir de un subgrafo generador cuyo conjunto de aristas es vacío, y en cada paso agregar una arista de peso mínimo que no forme ciclos con las demás aristas del conjunto, hasta haber agregado $n-1$ aristas.

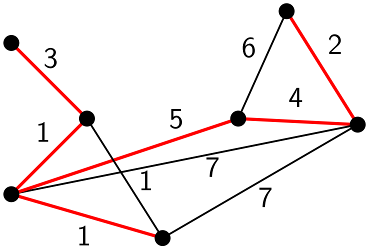
Ej:



Algoritmo de Kruskal para AGM

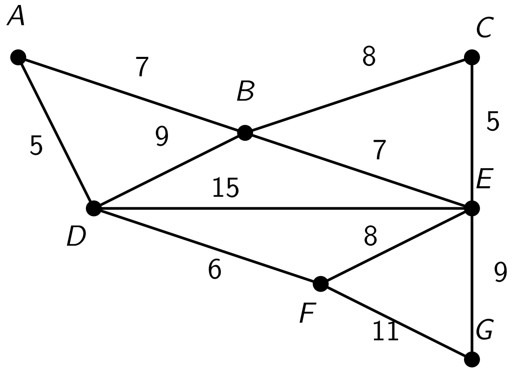
Partir de un subgrafo generador cuyo conjunto de aristas es vacío, y en cada paso agregar una arista de peso mínimo que no forme ciclos con las demás aristas del conjunto, hasta haber agregado $n-1$ aristas.

Ej:

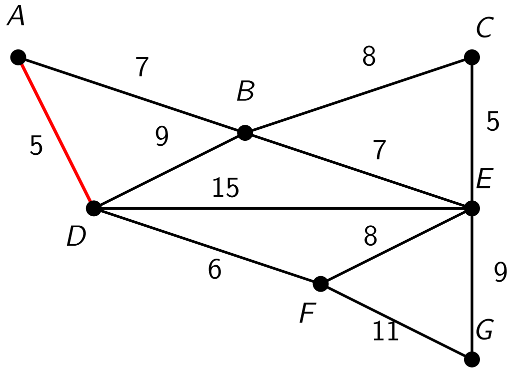


Veamos en la siguiente diapositiva otro ejemplo

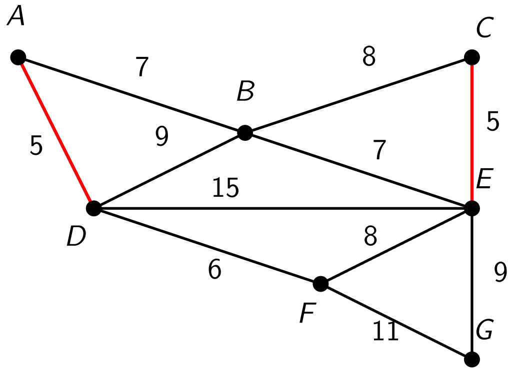
Sumando los pesos tenemos=



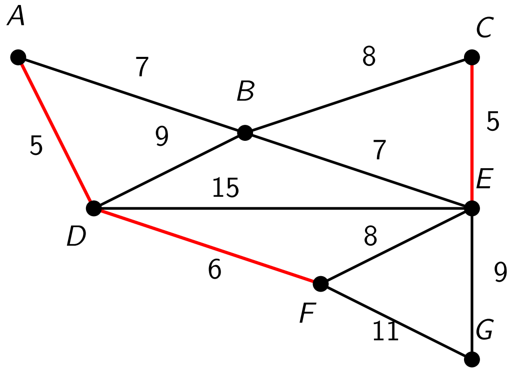
Sumando los pesos tenemos$= 5$



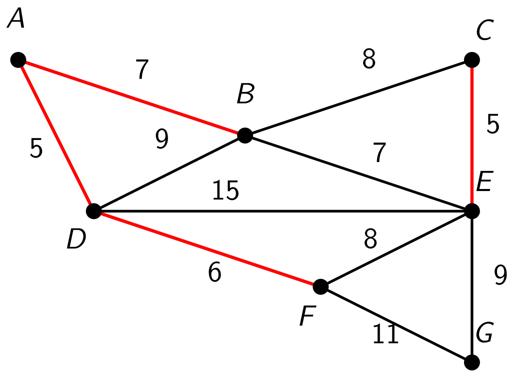
Sumando los pesos tenemos= 5+5



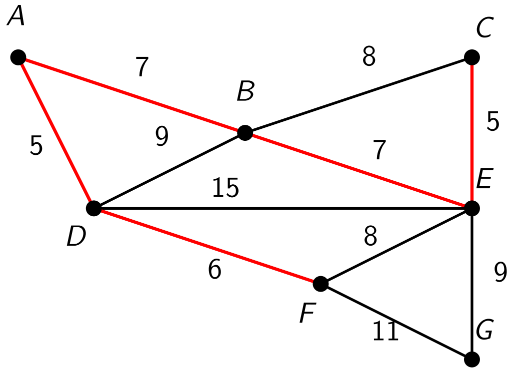
Sumando los pesos tenemos$= 5+5+6$



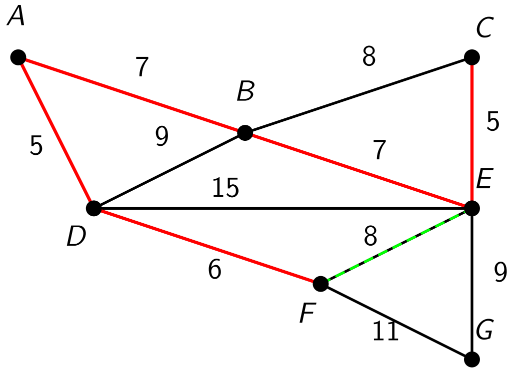
Sumando los pesos tenemos= 5+5+6+7



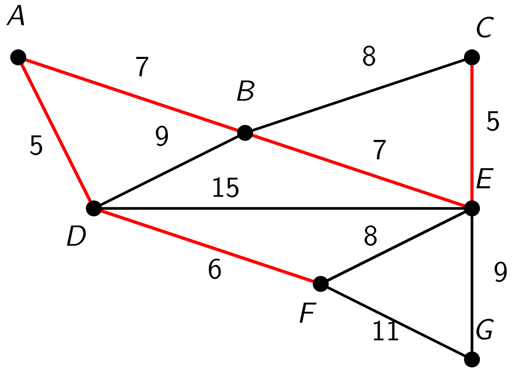
Sumando los pesos tenemos= 5+5+6+7+7



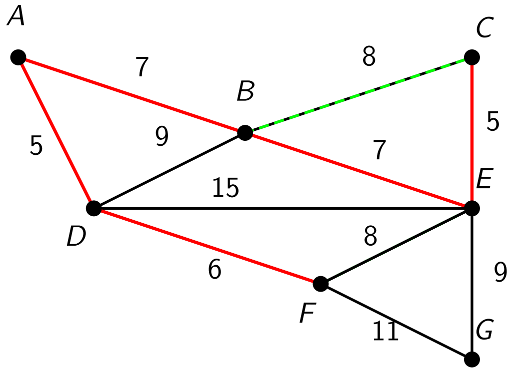
Sumando los pesos tenemos= 5+5+6+7+7



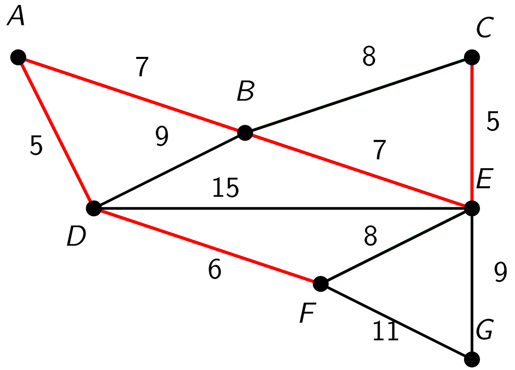
Sumando los pesos tenemos= 5+5+6+7+7



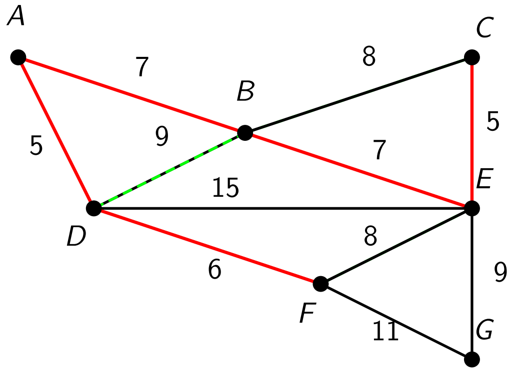
Sumando los pesos tenemos= 5+5+6+7+7



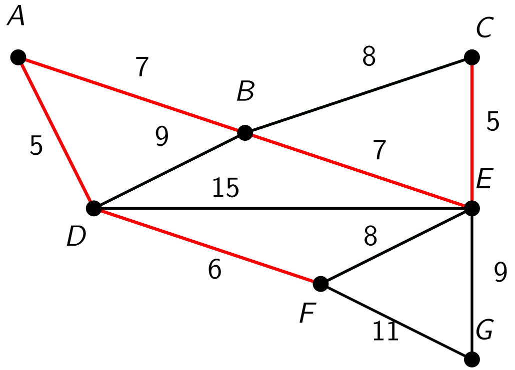
Sumando los pesos tenemos= 5+5+6+7+7



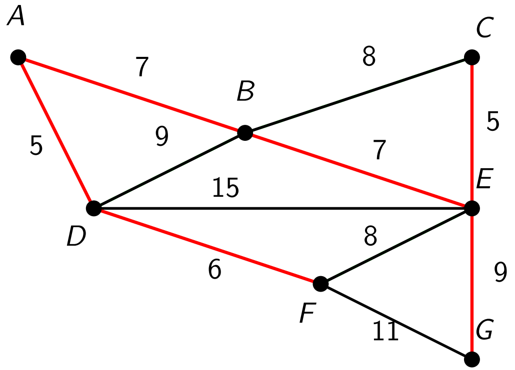
Sumando los pesos tenemos= 5+5+6+7+7



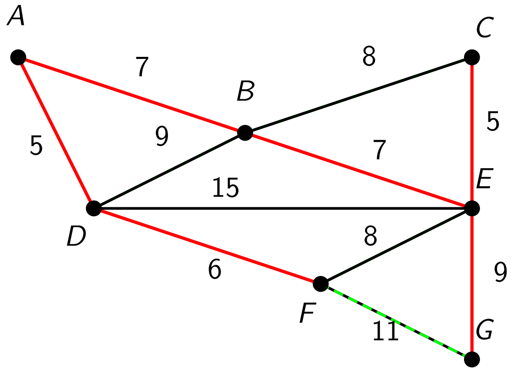
Sumando los pesos tenemos$=$ 5+5+6+7+7



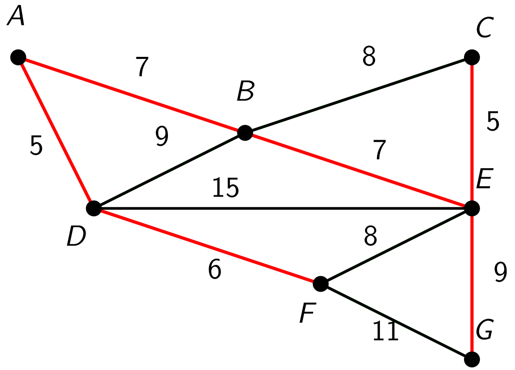
Sumando los pesos tenemos= 5+5+6+7+7+9



Sumando los pesos tenemos= 5+5+6+7+7+9



Sumando los pesos tenemos= 5+5+6+7+7+9=39



Demostración de que Kruskal construye un AGM

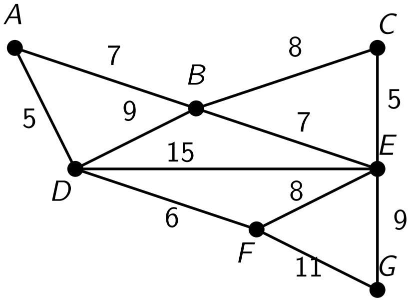
Demostración de que Kruskal construye un AGM

Para ver que el algoritmo construye un árbol generador, como en cada paso el subgrafo B elegido hasta el momento es generador y acíclico, basta ver que el algoritmo termina con $m_B = n_G - 1$. Si $m_B < n_G - 1$, B es no conexo. Sea B_1 una componente conexa de B. Como G es conexo, va a existir alguna arista con un extremo en B_1 y el otro en $V(G) - B_1$, que por lo tanto no forma ciclo con las demás aristas de B. Entonces, si $m_B < n_G - 1$, el algoritmo puede realizar un paso más.

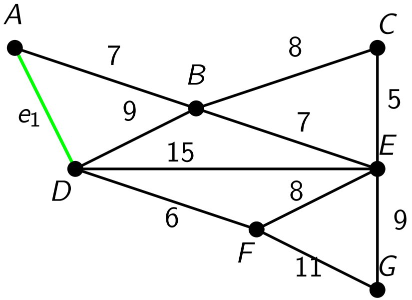
Demostración de que Kruskal construye un AGM

Para ver que el algoritmo construye un árbol generador, como en cada paso el subgrafo B elegido hasta el momento es generador y acíclico, basta ver que el algoritmo termina con $m_B = n_G - 1$. Si $m_B < n_G - 1$, B es no conexo. Sea B_1 una componente conexa de B. Como G es conexo, va a existir alguna arista con un extremo en B_1 y el otro en $V(G) - B_1$, que por lo tanto no forma ciclo con las demás aristas de B. Entonces, si $m_B < n_G - 1$, el algoritmo puede realizar un paso más.
La demostración siguiente está basada en le existencia de árboles generadores minimales para grafos con peso, no dirigidos, conexos y sin lazos, demostrada por Otakar Borusvka cuyo artículo se puede consultar en
http://www.cmat.edu.uy/ marclan/TAG/Sellanes/boruvka.pdf

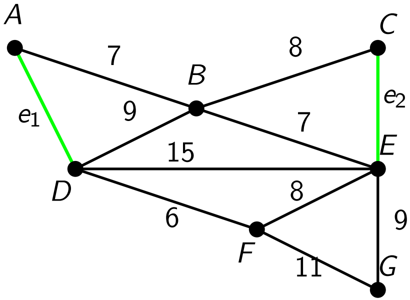
Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.



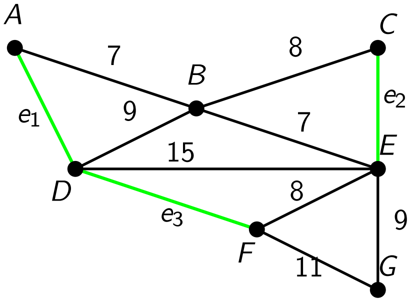
Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.



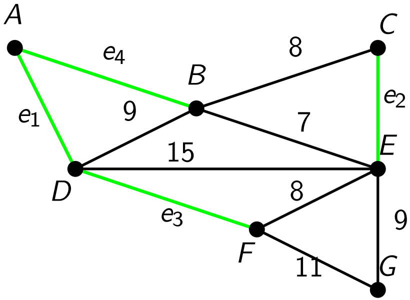
Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.



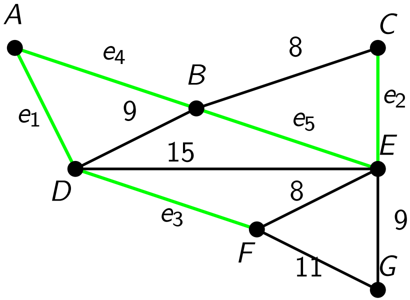
Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.



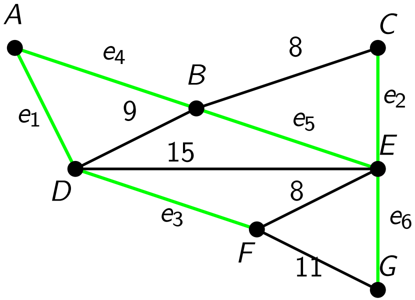
Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.



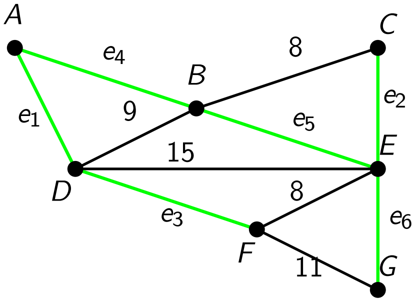
Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.



Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.

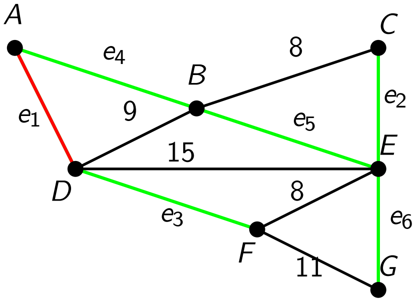


Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.
Para cada árbol generador T de G definimos $\ell(T)$ como el máximo $k \leq n$ tal que $\forall j < k$, $e_j \in T$.



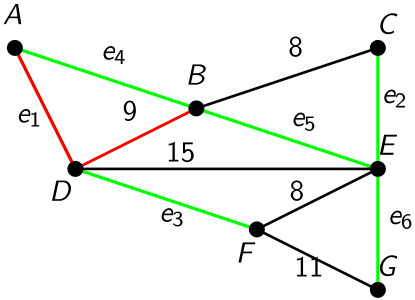
Por ejemplo para el árbol generador, determinado por la secuencia de aristas

Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.
Para cada árbol generador T de G definimos $\ell(T)$ como el máximo $k \leq n$ tal que $\forall j < k$, $e_j \in T$.



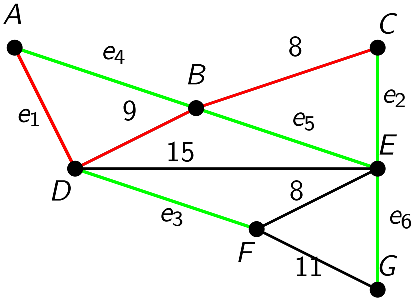
Por ejemplo para el árbol generador, determinado por la secuencia de aristas AD

Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.
Para cada árbol generador T de G definimos $\ell(T)$ como el máximo $k \leq n$ tal que $\forall j < k$, $e_j \in T$.



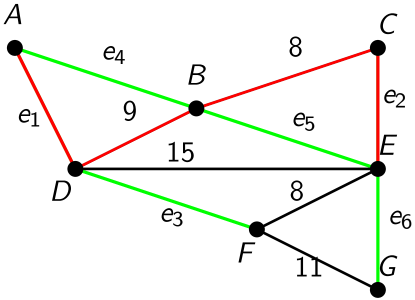
Por ejemplo para el árbol generador, determinado por la secuencia de aristas AD DB

Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.
Para cada árbol generador T de G definimos $\ell(T)$ como el máximo $k \leq n$ tal que $\forall j < k$, $e_j \in T$.



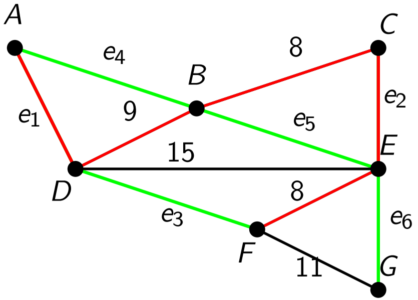
Por ejemplo para el árbol generador, determinado por la secuencia de aristas AD DB BC

Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.
Para cada árbol generador T de G definimos $\ell(T)$ como el máximo $k \leq n$ tal que $\forall j < k$, $e_j \in T$.



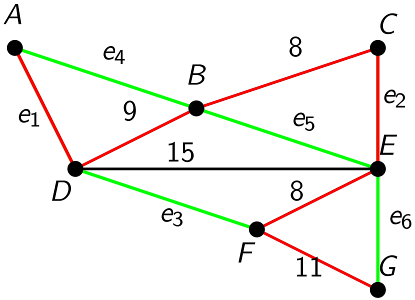
Por ejemplo para el árbol generador, determinado por la secuencia de aristas AD DB BC CE

Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.
Para cada árbol generador T de G definimos $\ell(T)$ como el máximo $k \leq n$ tal que $\forall j < k$, $e_j \in T$.



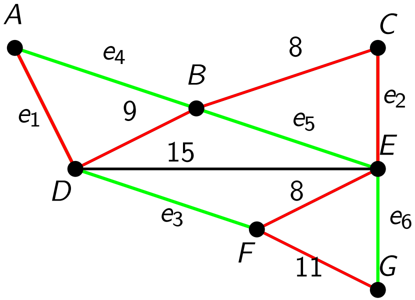
Por ejemplo para el árbol generador, determinado por la secuencia de aristas AD DB BC CE EF

Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.
Para cada árbol generador T de G definimos $\ell(T)$ como el máximo $k \leq n$ tal que $\forall j < k$, $e_j \in T$.



Por ejemplo para el árbol generador, determinado por la secuencia de aristas AD DB BC CE EF FG

Sea G un grafo, T_K el árbol generado por el algoritmo de Kruskal y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de T_K en el orden en que fueron elegidas por el algoritmo de Kruskal.
Para cada árbol generador T de G definimos $\ell(T)$ como el máximo $k \leq n$ tal que $\forall j < k$, $e_j \in T$.



Por ejemplo para el árbol generador, determinado por la secuencia de aristas AD DB BC CE EF FG $\ell(T) = 3$

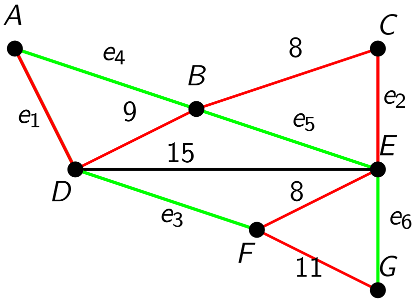
Ahora, sea T_0 un AGM para el que $\ell(T_0) = r$ es maximal. Si $r = n$, entonces T_0 coincide con T_K, con lo cual T_K resulta ser solución óptima (minimal).

Ahora, sea T_0 un AGM para el que $\ell(T_0) = r$ es maximal. Si $r = n$, entonces T_0 coincide con T_K, con lo cual T_K resulta ser solución óptima (minimal).

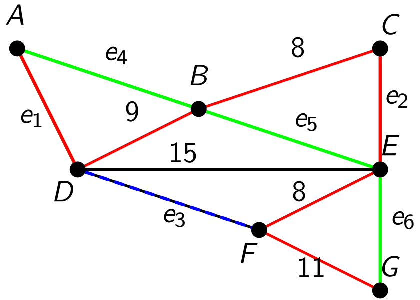
De otra forma $r \leq n-1$, y $e_r \notin T_0$,

Ahora, sea T_0 un AGM para el que $\ell(T_0) = r$ es maximal. Si $r = n$, entonces T_0 coincide con T_K, con lo cual T_K resulta ser solución óptima (minimal).

De otra forma $r \leq n - 1$, y $e_r \notin T_0$, como T_0 es conexo hay un camino C en T_0, que une los extremos de e_r.

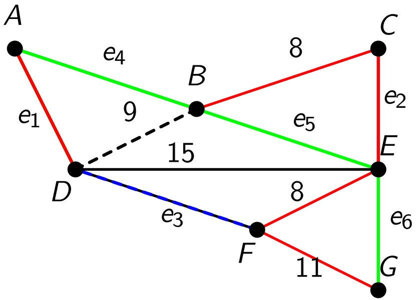


Ahora, sea T_0 un AGM para el que $\ell(T_0) = r$ es maximal. Si $r = n$, entonces T_0 coincide con T_K, con lo cual T_K resulta ser solución óptima (minimal).
De otra forma $r \leq n - 1$, y $e_r \notin T_0$, como T_0 es conexo hay un camino C en T_0, que une los extremos de e_r.



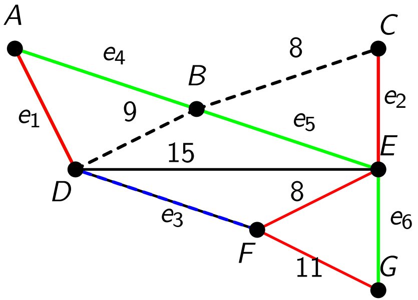
En el caso de la figura es el camino

Ahora, sea T_0 un AGM para el que $\ell(T_0) = r$ es maximal. Si $r = n$, entonces T_0 coincide con T_K, con lo cual T_K resulta ser solución óptima (minimal).
De otra forma $r \leq n - 1$, y $e_r \notin T_0$, como T_0 es conexo hay un camino C en T_0, que une los extremos de e_r.



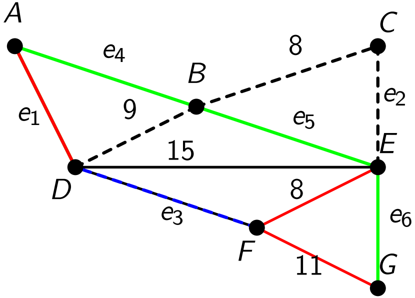
En el caso de la figura es el camino DB,

Ahora, sea T_0 un AGM para el que $\ell(T_0) = r$ es maximal. Si $r = n$, entonces T_0 coincide con T_K, con lo cual T_K resulta ser solución óptima (minimal).
De otra forma $r \leq n - 1$, y $e_r \notin T_0$, como T_0 es conexo hay un camino C en T_0, que une los extremos de e_r.



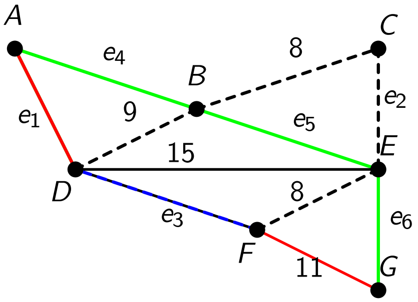
En el caso de la figura es el camino DB, BC,

Ahora, sea T_0 un AGM para el que $\ell(T_0) = r$ es maximal. Si $r = n$, entonces T_0 coincide con T_K, con lo cual T_K resulta ser solución óptima (minimal).
De otra forma $r \leq n - 1$, y $e_r \notin T_0$, como T_0 es conexo hay un camino C en T_0, que une los extremos de e_r.



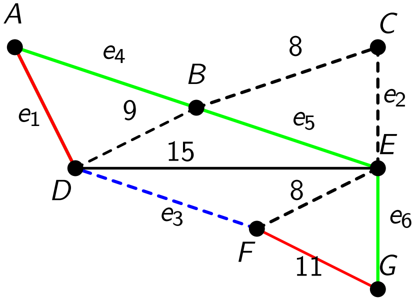
En el caso de la figura es el camino DB, BC, CE,

Ahora, sea T_0 un AGM para el que $\ell(T_0) = r$ es maximal. Si $r = n$, entonces T_0 coincide con T_K, con lo cual T_K resulta ser solución óptima (minimal).
De otra forma $r \leq n - 1$, y $e_r \notin T_0$, como T_0 es conexo hay un camino C en T_0, que une los extremos de e_r.

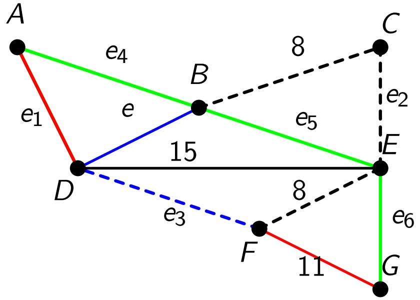


En el caso de la figura es el camino DB, BC, CE, EF

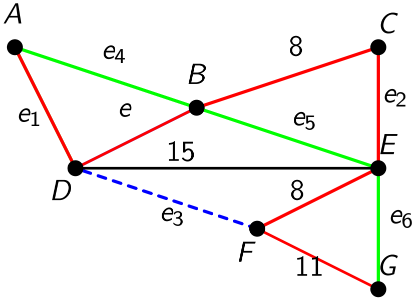
Como T_K es acíclico, hay alguna arista e en $C \subset T_0$ tal que $e \notin T_K$. Como $e_1, \ldots, e_{r-1} \in T_0$ y T_0 es acíclico, e no forma ciclos con $e_1, \ldots, e_{r-1}$.



Como T_K es acíclico, hay alguna arista e en $C \subset T_0$ tal que $e \notin T_K$. Como $e_1, \ldots, e_{r-1} \in T_0$ y T_0 es acíclico, e no forma ciclos con $e_1, \ldots, e_{r-1}$. Además por la forma en que fue elegida e_r por el algoritmo de Kruskal, $peso(e_r) \leq peso(e)$.

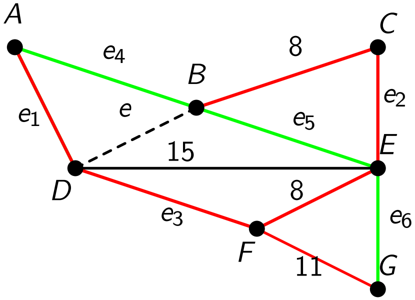


Como T_K es acíclico, hay alguna arista e en $C \subset T_0$ tal que $e \notin T_K$. Como $e_1, \ldots, e_{r-1} \in T_0$ y T_0 es acíclico, e no forma ciclos con $e_1, \ldots, e_{r-1}$. Además por la forma en que fue elegida e_r por el algoritmo de Kruskal, $peso(e_r) \leq peso(e)$.



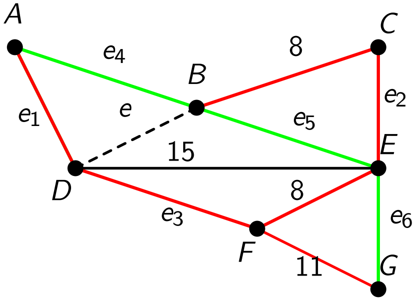
Pero entonces $T_1 = T_0 - e \cup \{e_r\}$ es un árbol generador de G de peso menor o igual a T_0 y $\ell(T_1) > \ell(T_0)$, absurdo.

Como T_K es acíclico, hay alguna arista e en $C \subset T_0$ tal que $e \notin T_K$. Como $e_1, \ldots, e_{r-1} \in T_0$ y T_0 es acíclico, e no forma ciclos con $e_1, \ldots, e_{r-1}$. Además por la forma en que fue elegida e_r por el algoritmo de Kruskal, $peso(e_r) \leq peso(e)$.



Pero entonces $T_1 = T_0 - e \cup \{e_r\}$ es un árbol generador de G de peso menor o igual a T_0 y $\ell(T_1) > \ell(T_0)$, absurdo.

Como T_K es acíclico, hay alguna arista e en $C \subset T_0$ tal que $e \notin T_K$. Como $e_1, \ldots, e_{r-1} \in T_0$ y T_0 es acíclico, e no forma ciclos con $e_1, \ldots, e_{r-1}$. Además por la forma en que fue elegida e_r por el algoritmo de Kruskal, $peso(e_r) \leq peso(e)$.



Pero entonces $T_1 = T_0 - e \cup \{e_r\}$ es un árbol generador de G de peso menor o igual a T_0 y $\ell(T_1) > \ell(T_0)$, absurdo.

Luego T_K es un árbol generador mínimo. $\square$

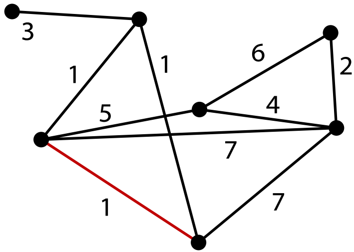
Algoritmo de Prim para AGM

Partir de un conjunto de aristas $A = \{e\}$ y un conjunto de vértices $W = \{v, w\}$, donde e es una arista de peso mínimo en G y v y w son sus extremos. En cada paso, agregar a A una arista f de peso mínimo con un extremo en W y el otro en $V(G) - W$. Agregar a W el extremo de f que no estaba en W, hasta que $W = V(G)$.

Algoritmo de Prim para AGM

Partir de un conjunto de aristas $A = \{e\}$ y un conjunto de vértices $W = \{v, w\}$, donde e es una arista de peso mínimo en G y v y w son sus extremos. En cada paso, agregar a A una arista f de peso mínimo con un extremo en W y el otro en $V(G) - W$. Agregar a W el extremo de f que no estaba en W, hasta que $W = V(G)$.

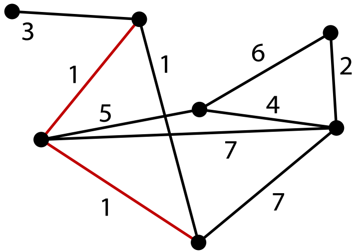
Ej:



Algoritmo de Prim para AGM

Partir de un conjunto de aristas $A = \{e\}$ y un conjunto de vértices $W = \{v, w\}$, donde e es una arista de peso mínimo en G y v y w son sus extremos. En cada paso, agregar a A una arista f de peso mínimo con un extremo en W y el otro en $V(G) - W$. Agregar a W el extremo de f que no estaba en W, hasta que $W = V(G)$.

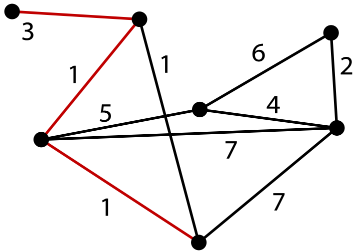
Ej:



Algoritmo de Prim para AGM

Partir de un conjunto de aristas $A = \{e\}$ y un conjunto de vértices $W = \{v, w\}$, donde e es una arista de peso mínimo en G y v y w son sus extremos. En cada paso, agregar a A una arista f de peso mínimo con un extremo en W y el otro en $V(G) - W$. Agregar a W el extremo de f que no estaba en W, hasta que $W = V(G)$.

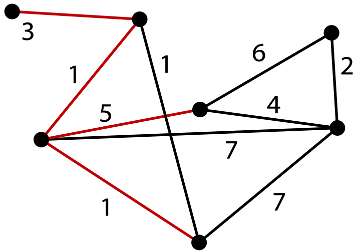
Ej:



Algoritmo de Prim para AGM

Partir de un conjunto de aristas $A = \{e\}$ y un conjunto de vértices $W = \{v, w\}$, donde e es una arista de peso mínimo en G y v y w son sus extremos. En cada paso, agregar a A una arista f de peso mínimo con un extremo en W y el otro en $V(G) - W$. Agregar a W el extremo de f que no estaba en W, hasta que $W = V(G)$.

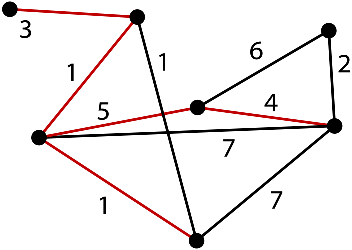
Ej:



Algoritmo de Prim para AGM

Partir de un conjunto de aristas $A = \{e\}$ y un conjunto de vértices $W = \{v, w\}$, donde e es una arista de peso mínimo en G y v y w son sus extremos. En cada paso, agregar a A una arista f de peso mínimo con un extremo en W y el otro en $V(G) - W$. Agregar a W el extremo de f que no estaba en W, hasta que $W = V(G)$.

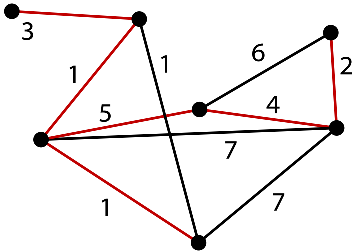
Ej:



Algoritmo de Prim para AGM

Partir de un conjunto de aristas $A = \{e\}$ y un conjunto de vértices $W = \{v, w\}$, donde e es una arista de peso mínimo en G y v y w son sus extremos. En cada paso, agregar a A una arista f de peso mínimo con un extremo en W y el otro en $V(G) - W$. Agregar a W el extremo de f que no estaba en W, hasta que $W = V(G)$.

Ej:



Demostración de que Prim construye un AGM

Para ver que construye un árbol generador, se puede ver que en cada paso del algoritmo, el subgrafo elegido hasta el momento es conexo y con $m = n - 1$. Finalmente, como el grafo es conexo, mientras $W \neq V(G)$ va a existir alguna arista de W a $V(G) - W$ con lo cual el algoritmo termina construyendo un árbol generador del grafo.

Demostración de que Prim construye un AGM

Para ver que construye un árbol generador, se puede ver que en cada paso del algoritmo, el subgrafo elegido hasta el momento es conexo y con $m = n - 1$. Finalmente, como el grafo es conexo, mientras $W \neq V(G)$ va a existir alguna arista de W a $V(G) - W$ con lo cual el algoritmo termina construyendo un árbol generador del grafo.

Sea G un grafo, P el árbol generado por el algoritmo de Prim y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de P en el orden en que fueron elegidas por el algoritmo de Prim. Para cada árbol generador minimal T de G definimos $\ell(T)$ como el máximo $k \leq n$ tal que $\forall j < k$, $e_j \in T$.

Demostración de que Prim construye un AGM

Para ver que construye un árbol generador, se puede ver que en cada paso del algoritmo, el subgrafo elegido hasta el momento es conexo y con $m = n - 1$. Finalmente, como el grafo es conexo, mientras $W \neq V(G)$ va a existir alguna arista de W a $V(G) - W$ con lo cual el algoritmo termina construyendo un árbol generador del grafo.

Sea G un grafo, P el árbol generado por el algoritmo de Prim y $\{e_1, e_2, \ldots, e_{n-1}\}$ la secuencia de aristas de P en el orden en que fueron elegidas por el algoritmo de Prim. Para cada árbol generador minimal T de G definimos $\ell(T)$ como el máximo $k \leq n$ tal que $\forall j < k$, $e_j \in T$.

Ahora, sea T_0 un AGM que maximiza ℓ. Si $\ell(T_0) = r = n$, entonces T_0 coincide con P, con lo cual P resulta ser la solución óptima (minimal). Si $r \leq n - 1$, entonces $e_r \notin T$. Como T_0 es conexo, en T_0 hay un camino C que une los extremos de e_r.

Demostración de que Prim construye un AGM

- Si $r = 1$, como e_1 es de peso mínimo, $peso(e_1) \leq peso(e)\ \forall e \in C$. Luego $T_1 = T_0 - e \cup \{e_r\}$ es un árbol generador de G de peso menor o igual a T_0 y $\ell(T_1) > \ell(T_0)$, absurdo.

Demostración de que Prim construye un AGM

- Si $r = 1$, como e_1 es de peso mínimo, $peso(e_1) \leq peso(e)$ $\forall e \in C$. Luego $T_1 = T_0 - e \cup \{e_r\}$ es un árbol generador de G de peso menor o igual a T_0 y $\ell(T_1) > \ell(T_0)$, absurdo.
- Si $r > 1$, sea V_1 el conjunto de extremos de las aristas $e_1, \ldots, e_{r-1}$ y $V_2 = V - V_1$. Por la forma de elegir las aristas en Prim, e_r es de peso mínimo entre las que tienen un extremo en V_1 y otro en V_2. El camino C va de un vértice de V_1 a un vértice de V_2, con lo cual, existe $e \in C$ con un extremo en V_1 y otro en V_2 (sus vértices se pueden partir entre los de V_1 y los de V_2, ambos conjuntos son no vacíos y C es conexo). Entonces $peso(e_r) \leq peso(e)$ y $T_0 - e \cup \{e_r\}$ es un árbol generador de peso menor o igual a T_0 y $\ell(T_1) > \ell(T_0)$ (e no es ninguna de las e_i con $i < r$ porque esas tienen ambos extremos en V_1, por definición de V_1), absurdo.

Demostración de que Prim construye un AGM

- Si $r = 1$, como e_1 es de peso mínimo, $peso(e_1) \leq peso(e)\ \forall e \in C$. Luego $T_1 = T_0 - e \cup \{e_r\}$ es un árbol generador de G de peso menor o igual a T_0 y $\ell(T_1) > \ell(T_0)$, absurdo.
- Si $r > 1$, sea V_1 el conjunto de extremos de las aristas $e_1, \ldots, e_{r-1}$ y $V_2 = V - V_1$. Por la forma de elegir las aristas en Prim, e_r es de peso mínimo entre las que tienen un extremo en V_1 y otro en V_2. El camino C va de un vértice de V_1 a un vértice de V_2, con lo cual, existe $e \in C$ con un extremo en V_1 y otro en V_2 (sus vértices se pueden partir entre los de V_1 y los de V_2, ambos conjuntos son no vacíos y C es conexo). Entonces $peso(e_r) \leq peso(e)$ y $T_0 - e \cup \{e_r\}$ es un árbol generador de peso menor o igual a T_0 y $\ell(T_1) > \ell(T_0)$ (e no es ninguna de las e_i con $i < r$ porque esas tienen ambos extremos en V_1, por definición de V_1), absurdo.

Luego P es un árbol generador mínimo. $\square$

Complejidad del algoritmo de Kuskal

Sea un grafo G en la hipótesis establecidas y sea $n = |V|$ y $m = |E|$, con $m \geq 2$, podemos usar la ordenación por inserción para enumerar y volver a etiquetar las aristas (en caso necesario) como $e_1, \ldots, e_m$ donde $p(e_1) \leq \cdots \leq p(e_m)$. El número de comparaciones necesaria para hacer esto es $O(m \log(m))$. Luego una vez enumeradas las aristas en ese orden (de peso no decreciente), realizamos el paso 2 del algoritmo un máximo de $m - 1$ veces: una vez por cada una de las aristas $e_1, e_2, \ldots, e_m$

Complejidad del algoritmo de Kuskal

Sea un grafo G en la hipótesis establecidas y sea $n = |V|$ y $m = |E|$, con $m \geq 2$, podemos usar la ordenación por inserción para enumerar y volver a etiquetar las aristas (en caso necesario) como $e_1, \dots, e_m$ donde $p(e_1) \leq \cdots \leq p(e_m)$. El número de comparaciones necesaria para hacer esto es $O(m \log(m))$. Luego una vez enumeradas las aristas en ese orden (de peso no decreciente), realizamos el paso 2 del algoritmo un máximo de $m - 1$ veces: una vez por cada una de las aristas $e_1, e_2, \dots, e_m$ Para cada arista e_i, con $2 \leq i \leq m$, debemos determinar si e_i provoca la formación de un ciclo en el árbol o bosque desarrollado hasta ese momento (después de considerar las aristas $e_1, \dots, e_{i-1}$). Esto no puede hacerse en una cantidad constante de tiempo (es decir $O(1)$) para cada arista. Sin embargo todo el trabajo necesario para la determinación de ciclos puede hacerse como máximo en $O(n \log(n))$ pasos.

Complejidad del algoritmo de Kruskal

En consecuencia, definiremos la función de complejidad f en tiempo para el peor caso, con $m \geq 2$, como la suma siguiente:

- el número total de comparaciones necesaria para ordenar las aristas de G en orden no decreciente, y

Complejidad del algoritmo de Kruskal

En consecuencia, definiremos la función de complejidad f en tiempo para el peor caso, con $m \geq 2$, como la suma siguiente:

- el número total de comparaciones necesaria para ordenar las aristas de G en orden no decreciente, y
- el número total de pasos realizados en el paso 2 que determina la formación de un ciclo.

Complejidad del algoritmo de Kruskal

En consecuencia, definiremos la función de complejidad f en tiempo para el peor caso, con $m \geq 2$, como la suma siguiente:

- el número total de comparaciones necesaria para ordenar las aristas de G en orden no decreciente, y
- el número total de pasos realizados en el paso 2 que determina la formación de un ciclo.

A menos que G sea un árbol, se sigue que $n \leq m$, ya que G es conexo. Luego $n \log(n) \leq m \log(m)$ y $f \in O(m \log(m)) = O(m \log(n))$.

Complejidad del algoritmo de Kruskal

En consecuencia, definiremos la función de complejidad f en tiempo para el peor caso, con $m \geq 2$, como la suma siguiente:

- el número total de comparaciones necesaria para ordenar las aristas de G en orden no decreciente, y
- el número total de pasos realizados en el paso 2 que determina la formación de un ciclo.

A menos que G sea un árbol, se sigue que $n \leq m$, ya que G es conexo. Luego $n \log(n) \leq m \log(m)$ y $f \in O(m \log(m)) = O(m \log(n))$. También podemos dar una medida en función de n, el número de vértices de G; $n - 1 \leq m$, puesto que el grafo es conexo y $m \leq \binom{n}{2} = \frac{n(n-1)}{2}$ el número de aristas de K_n.

Complejidad del algoritmo de Kruskal

En consecuencia, definiremos la función de complejidad f en tiempo para el peor caso, con $m \geq 2$, como la suma siguiente:

- el número total de comparaciones necesaria para ordenar las aristas de G en orden no decreciente, y
- el número total de pasos realizados en el paso 2 que determina la formación de un ciclo.

A menos que G sea un árbol, se sigue que $n \leq m$, ya que G es conexo. Luego $n \log(n) \leq m \log(m)$ y $f \in O(m \log(m)) = O(m \log(n))$.
También podemos dar una medida en función de n, el número de vértices de G; $n - 1 \leq m$, puesto que el grafo es conexo y $m \leq \binom{n}{2} = \frac{n(n-1)}{2}$ el número de aristas de K_n.
En consecuencia $m \log(m) \leq n^2 \log(n^2) = 2n^2 \log(n)$ y podemos expresar la complejidad en tiempo del peor caso del algoritmo de Kruskal como $O(n^2 \log(n))$

Complejidad del algoritmo Prim y comparación

Acerca de la función de complejidad en tiempo para el peor caso para el algoritmo de Prim, cuando aplicamos dicho algoritmo a un grafo G como en las hipótesis ya señaladas, las implementaciones usuales requieren $O(n^2)$ pasos.

Complejidad del algoritmo Prim y comparación

Acerca de la función de complejidad en tiempo para el peor caso para el algoritmo de Prim, cuando aplicamos dicho algoritmo a un grafo G como en las hipótesis ya señaladas, las implementaciones usuales requieren $O(n^2)$ pasos.

Algunas implementaciones recientes de este algoritmo han mejorado la situación, de modo que ahora son necesarios $O(m \log(n))$ pasos.

Complejidad del algoritmo Prim y comparación

Acerca de la función de complejidad en tiempo para el peor caso para el algoritmo de Prim, cuando aplicamos dicho algoritmo a un grafo G como en las hipótesis ya señaladas, las implementaciones usuales requieren $O(n^2)$ pasos.

Algunas implementaciones recientes de este algoritmo han mejorado la situación, de modo que ahora son necesarios $O(m\log(n))$ pasos.

La pregunta que nos hacemos es ¿que algoritmo nos conviene usar? y la respuesta depende del número de aristas. Si casi todas las aristas posibles están en el grafo (es decir $m = O(n^2)$), la complejidad del algoritmo de Kruskal sería $O(n^2\log(n))$, y por lo tanto el algoritmo de Prim sería más eficiente. Por otro lado, si el número de aristas $m = O(n)$, el algoritmo de Kruscal correría en tiempo $O(n\log(n))$ que es menor que el tiempo necesario por el algoritmo de Prim en las implementaciones usuales.