Árbol mínimo cobertor

Guillermo Palma

Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información

Plan

1. El problema del árbol mínimo cobertor
2. Crecimiento en el árbol mínimo cobertor
3. Algoritmo de Kruskal
4. Algoritmo de Prim

El problema del árbol mínimo cobertor

Árbol cobertor (*spanning tree*)

Definición de árbol cobertor

Dado un grafo $G = (V, E)$ no dirigido y conectado, un **árbol cobertor** es un subconjunto acíclico de lados $T \subseteq E$ que conecta a todos los vértices V del grafo.

Ejemplo de un árbol cobertor

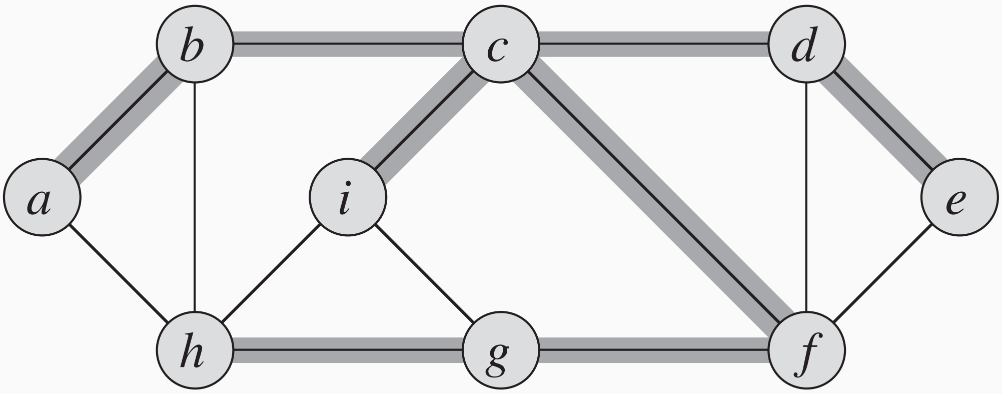

Figura 1: Grafo no dirigido con costos en los lados. Los lados resaltados en gris corresponden a un **árbol cobertor**. Fuente [1].

Costo de un árbol cobertor

Definición del costo de un árbol cobertor

Dado un grafo $G = (V, E)$ no dirigido y conectado, en donde que cada lado $(u, v) \in E$ tiene un atributo numérico $w(u, v)$ llamado costo, definimos el **costo de un árbol cobertor** T, como la suma de los costos de los lados que conforman a T. Esto es:

$$w(T) = \sum_{(u,v) \in T} w(u, v)$$

Problema del árbol mínimo cobertor

Definición del problema del árbol mínimo cobertor

El problema del **árbol mínimo cobertor**, en inglés *minimum spanning tree* (MST), consiste en encontrar el árbol cobertor de costo mínimo.

Ejemplo de un árbol mínimo cobertor

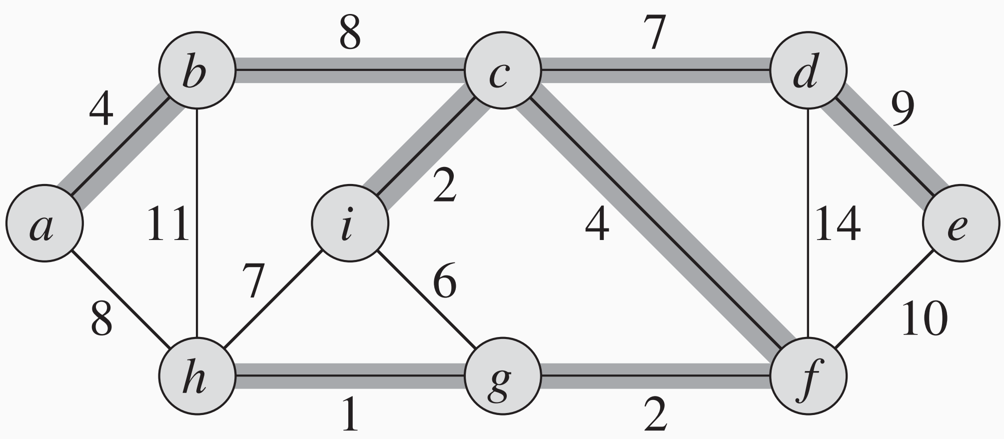

Figura 2: Grafo no dirigido con costos en los lados. Los lados en resaltados en gris corresponde a un **árbol mínimo cobertor** de costo 37. Fuente [1].

Crecimiento en el árbol mínimo cobertor

Sobre los algoritmos del árbol mínimo cobertor

- Se tiene un grafo G no dirigido y conectado.
- Se tiene una función de costo sobre los lados, esto es $w : E \to \mathbb{R}$.
- Se desea un algoritmo para determinar el árbol mínimo cobertor.
- Los algoritmos de Prim y Kruskal siguen una estrategia ávida de escoger el mejor lado en cada iteración y lo agregan a un conjunto A.
- Los algoritmo mantienen el invariante: **antes de cada iteración, A es un subconjunto de algún árbol de mínimo cobertor**.
- Se escoge en cada iteración un lado (u, v) que no viole el invariante, que llamamos **lado seguro**.
- Se agrega el **lado seguro** (u, v) a A.
- Se continúa agregando **lados seguros** hasta que se tenga el árbol mínimo cobertor.

Algoritmo genérico del árbol mínimo cobertor

Función ArbMimCobertor-generico(G, w)

inicio
- $A \leftarrow \emptyset$;
- **mientras** A *no sea un árbol mínimo cobertor* **hacer**
  - encontrar un **lado seguro** (u, v) para A ;
  - $A \leftarrow A \cup (u, v)$
- **devolver** A

Invariante del algoritmo genérico del árbol mínimo cobertor

Invariante de ciclo

Antes de cada iteración del ciclo, A es un subconjunto de algún árbol mínimo cobertor.

Invariante del algoritmo genérico del árbol mínimo cobertor, cont.

Invariante de ciclo

Antes de cada iteración del ciclo *mientras*, A es un subconjunto de algún árbol mínimo cobertor.

Inicialización: Después de la inicialización de la línea 2, A satisface el invariante de ciclo.

Mantenimiento: El ciclo *mientras* de las líneas 3-5 mantiene el invariante porque solo agrega lados seguros.

Terminación: Todos los lados de A pertenecen al *árbol mínimo cobertor*, por lo tanto el conjunto A que se retorna, debe ser un *árbol mínimo cobertor*.

Corte de un grafo (*cut*)

Definición de un corte de un grafo

Un **corte** de un grafo no dirigido $G = (V, E)$ es la partición del conjunto de vértices de un grafo en dos conjuntos disjuntos, uno llamado S y el otro es $V - S$. Se representa como $(S, V - S)$.

Cruce de un lado

Definición de cruce de un corte de un lado

Un lado $(u, v) \in E$ de un grafo no dirigido $G = (V, E)$ se dice que **cruza un corte** $(S, V - S)$ si $u \in S$ y $v \in V - S$.

Respeto de un corte

Definición de respeto de un corte

Un corte $(S, V - S)$ de un grafo no dirigido $G = (V, E)$ se dice **respeta** a un conjunto de lados $A \subseteq E$ si ninguno de los lados de A cruza el corte.

Lado ligero

Definición de lado ligero

Un lado $(u, v) \in E$ de un grafo no dirigido $G = (V, E)$ se dice que es un **lado ligero** si es el lado de menor costo que **cruza un corte** $(S, V - S)$ de G. Si hay empate, se dice que hay más de un **lado ligero**.

Ejemplo de un corte con lado ligero

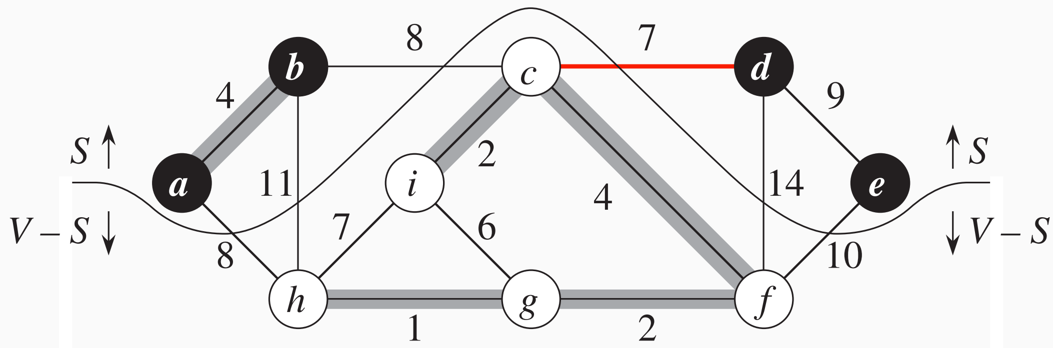

Figura 3: Corte $(S, V - S)$ de un grafo no dirigido. Los vértices negros están en S y los blancos en $V - S$. Los lados resaltados en gris corresponden a un conjunto A. El corte respeta a A. El lado en rojo es un lado ligero. Fuente [1].

Teorema del reconocimiento de un *lado seguro*

Teorema 1

Sea G un grafo no dirigido y conectado, y sea w una función de costo sobre los lados $w : E \to \mathbb{R}$. Sea A un subconjunto de E, tal que los lados de A pertenecen a algún árbol mínimo cobertor de G. Sea $(S, V - S)$ un corte cualquiera de G que respeta a A, y sea (u, v) un *lado ligero* que cruza $(S, V - S)$. Entonces, el lado (u, v) es un **lado seguro** para A.

Ilustración de la formación del árbol mínimo cobertor

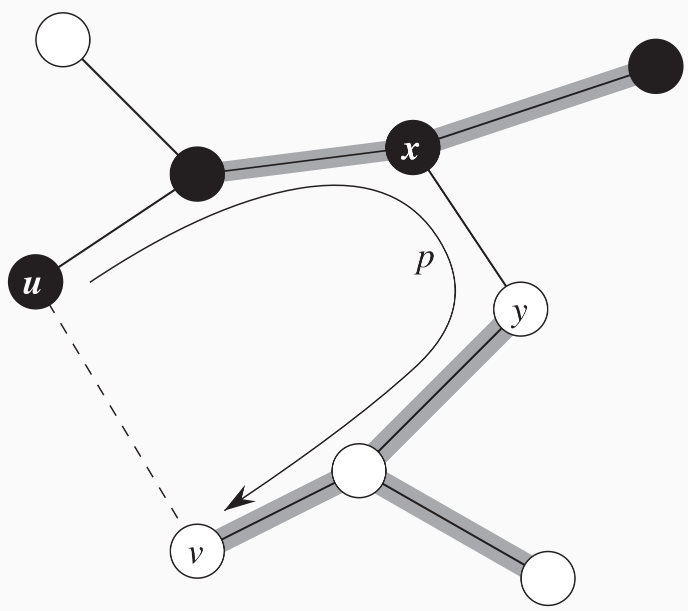

Figura 4: Árbol mínimo cobertor de un grafo. Se tiene que p es un camino simple de u hasta v. Observe que al agregar el lado (u, v) se forma un ciclo. En gris lados del subconjunto de lados A. Fuente [1].

Prueba del Teorema 1

Parte 1: se demuestra que (u, v) pertenece a un MST.

- Sea T el MST que contiene a A.
- Se asume que T no contiene a (u, v).
- Se construirá otro MST T' que contiene a $A \cup \{(u, v)\}$.
- Como $(u, v) \notin T$, (u, v) forma un ciclo con el camino p desde u hasta v en T (ver Figura 4).
- Como u y v están en lados opuestos del corte $(S, V - S)$, entonces tiene que haber un lado en T que pertenece a p y que cruza el corte.
- Sea $(x, y) \in T$ el lado que cruza el corte.
- $(x, y) \notin A$ porque A respeta el corte.
- Como $(x, y) \in T$ y está en p, su eliminación divide a T en dos.
- Se crea un nuevo árbol cobertor $T' = T - \{(x, y)\} \cup (u, v)$.
- Como (u, v) es un lado ligero del corte, entonces $w(u, v) \leq w(x, y)$.
- En consecuencia, $w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$.
- T es MST y $w(T') \leq w(T)$, por lo tanto T' debe ser un MST.

Continuación de la prueba del Teorema 1

Parte 2: se demuestra que (u, v) es un lado seguro.

- Se tiene que $A \subseteq T'$ y $A \subseteq T$.
- Se tiene que $(x, y) \notin A$.
- En consecuencia, $\{(u, v)\} \cup A \subseteq T'$.
- T' es un MST.
- Por lo tanto, (u, v) es un lado seguro para A.

$\square$

Reconocimiento de *lado seguro*

Corolario 1

Sea $G = (V, E)$ un grafo no dirigido y conectado, y sea w una función de costo sobre los lados $w : E \to \mathbb{R}$. Sea $A \subseteq E$ tal que los lados de A pertenecen a un árbol mínimo cobertor de G y sea $C = (V_c, E_c)$ una componente conexa (que es un árbol) en el bosque $G_A = (V, A)$. Si un lado $(u, v) \in E$ es un *lado ligero* que conecta a C con otra componentes conexa en G_A, entonces (u, v) es un *lado seguro* para A.

Prueba del Corolario 1

- Se tiene que el corte $(V_c, V - V_c)$ respeta a A.
- Se tiene que (u, v) es lado ligero para el corte.
- Por Teorema 1, por lo tanto (u, v) es un lado seguro para A.

$\square$

Algoritmo de Kruskal

Algoritmo de Kruskal

Función ArbMimCobertor-Kruskal($G = (V, E), w$)

inicio
  $A \leftarrow \emptyset$;
  para cada $v \in V$ **hacer**
    `make-set`(v) ;
  Ordena los lados en E en orden no decreciente por su costo w ;
  para cada $(u, v) \in E$; `/* lados en` E `están ordenados */`
  hacer
    si `find-set`(u) $\neq$ `find-set`(v) **entonces**
      $A \leftarrow A \cup (u, v)$;
      `union`(u, v);
  devolver A ;

Ejemplo del algoritmo de Kruskal

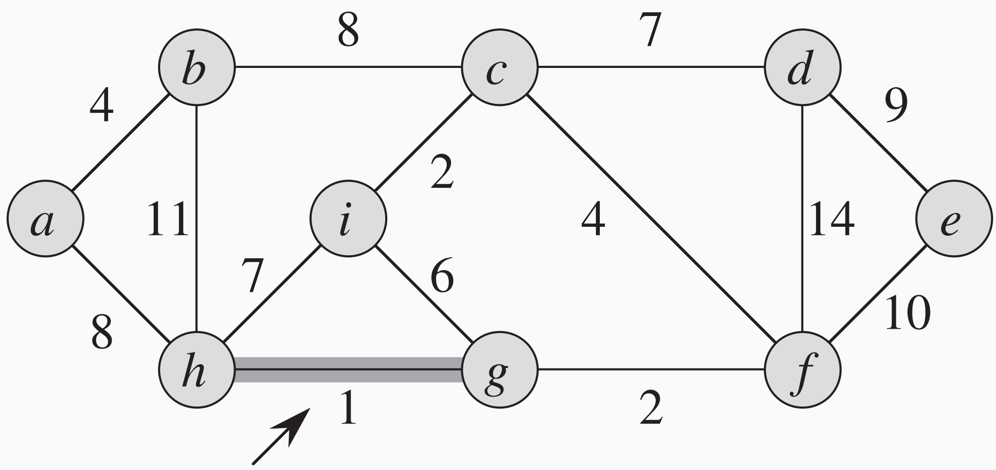

Figura 5: Se incluye el lado (h, g) en A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

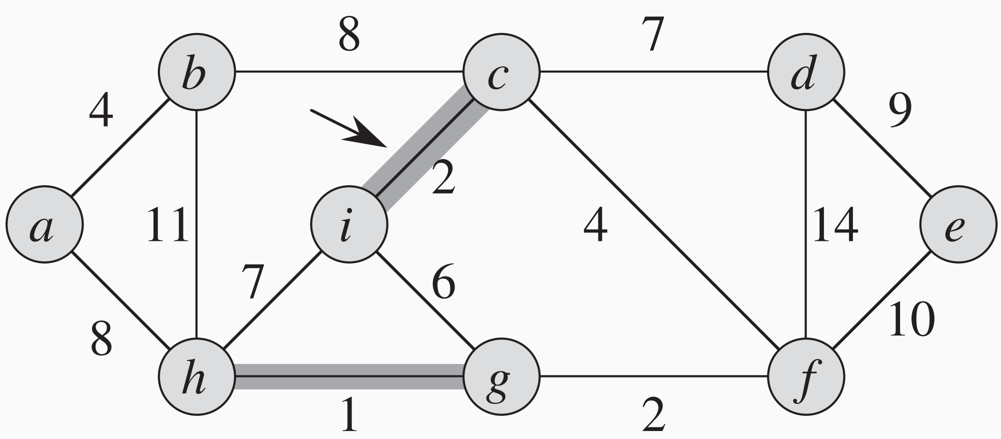

Figura 6: Se incluye el lado (i, c) en A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

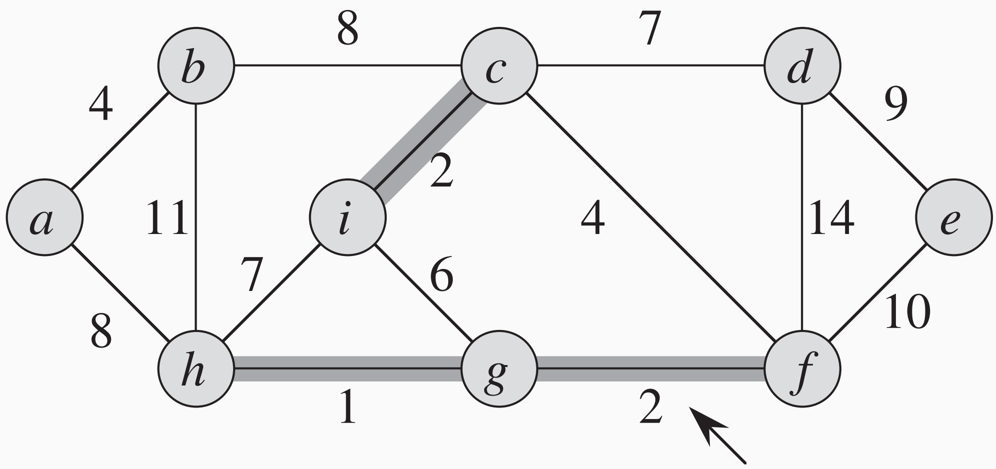

Figura 7: Se incluye el lado (g, f) en A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

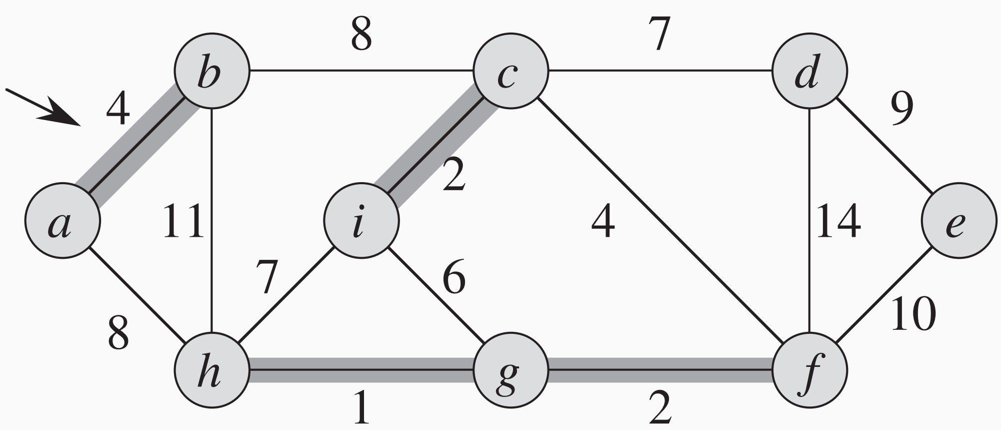

Figura 8: Se incluye el lado (a, b) en A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

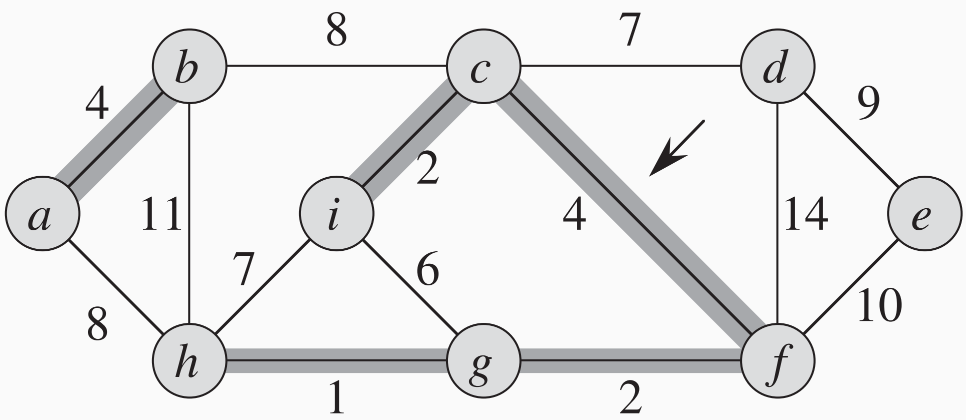

Figura 9: Se incluye el lado (c, f) en A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

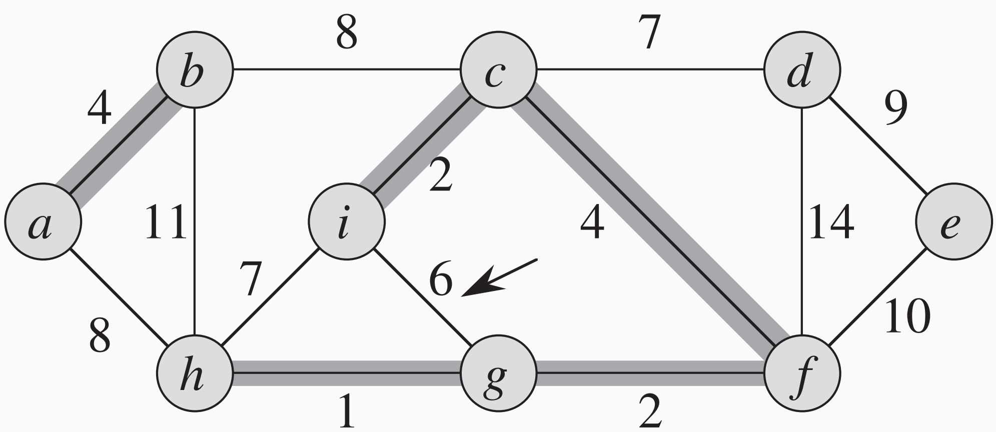

Figura 10: El lado (i, g) no se agrega a A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

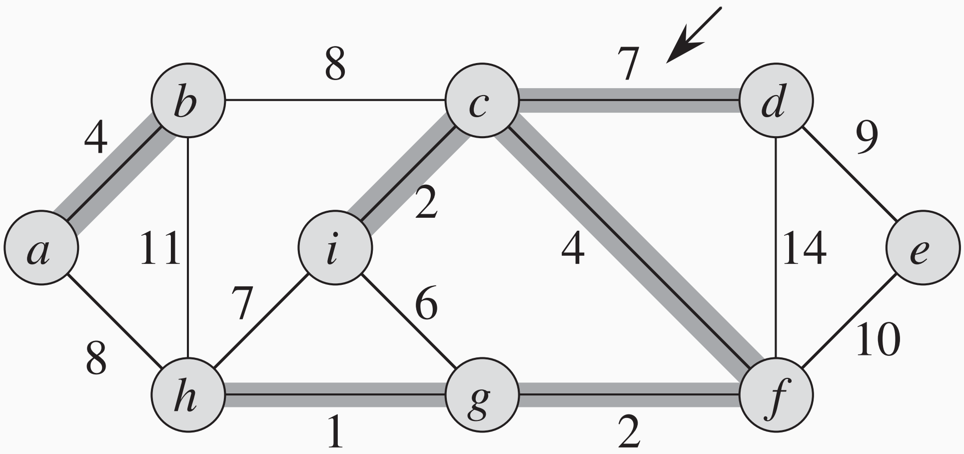

Figura 11: Se incluye el lado (c, d) en A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

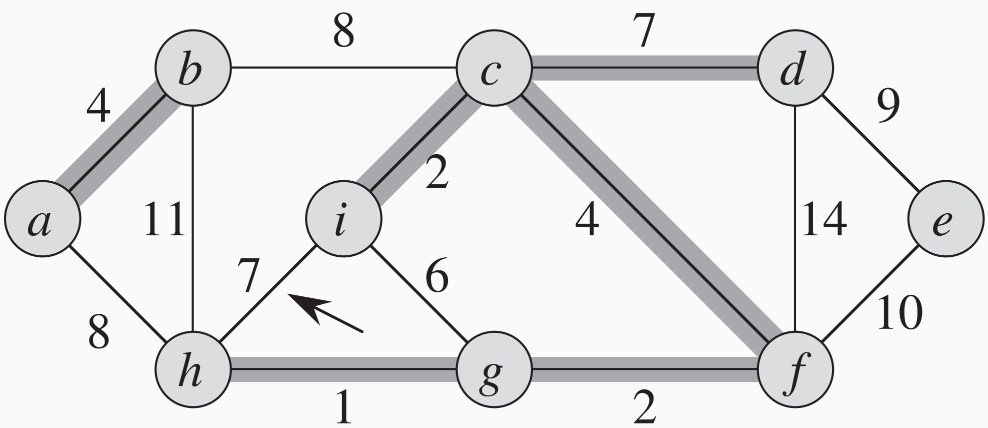

Figura 12: El lado (h, i) no se agrega a A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

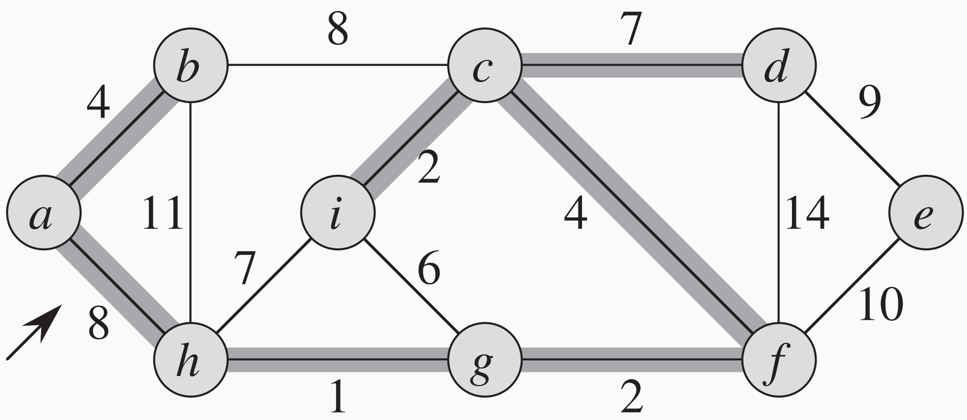

Figura 13: Se incluye el lado (a, h) en A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

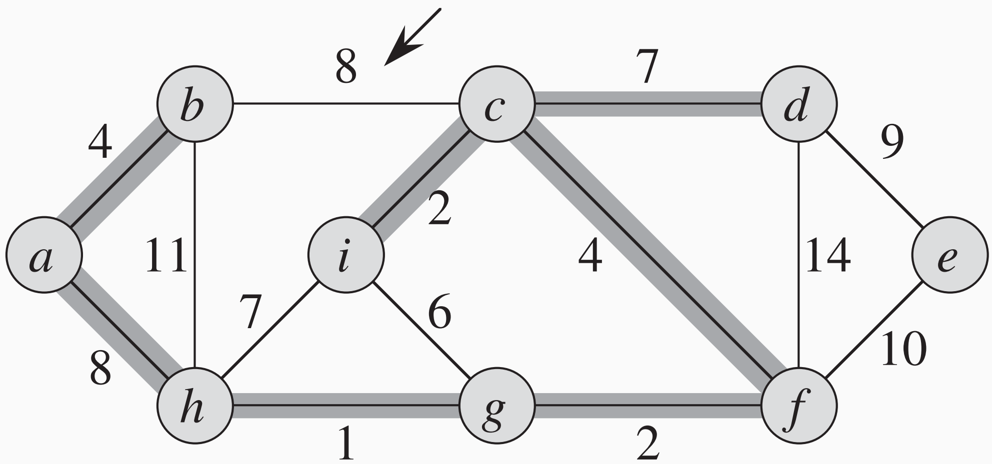

Figura 14: El lado (b, c) no se agrega a A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

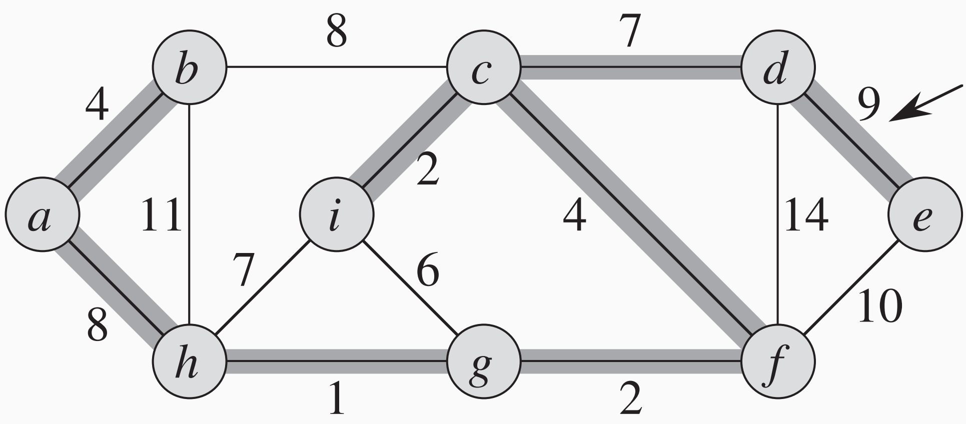

Figura 15: Se incluye el lado (d, e) en A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

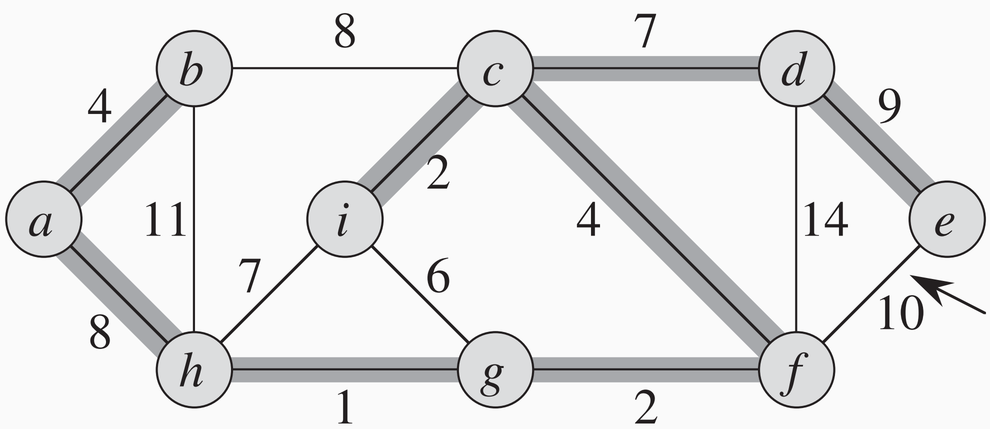

Figura 16: El lado (e, f) no se agrega a A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

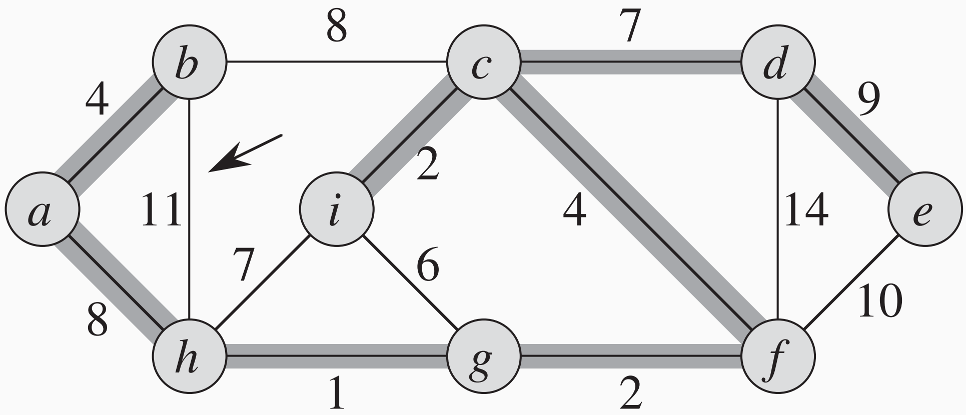

Figura 17: El lado (b, h) no se agrega a A. Fuente [1].

Ejemplo del algoritmo de Kruskal, cont.

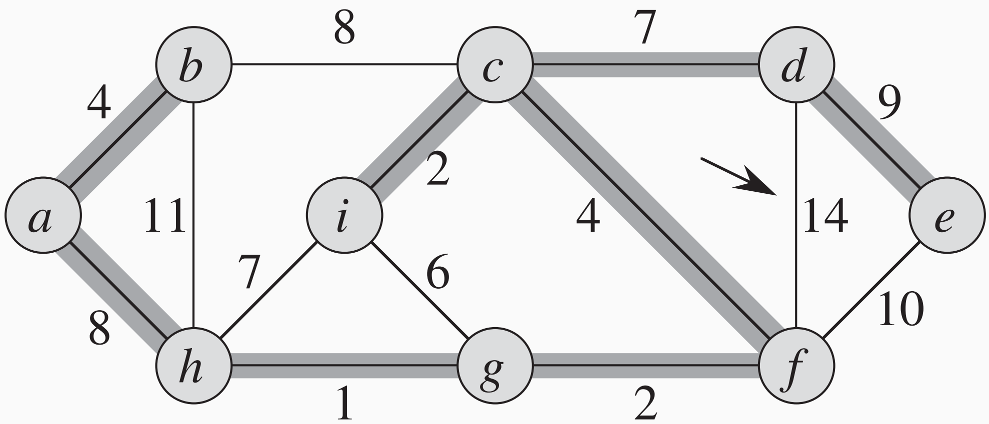

Figura 18: El lado (d, f) no se agrega a A. Fuente [1].

Análisis del tiempo del algoritmo de Kruskal

- Inicializar cada conjuntos es $O(1)$, entonces en total es $O(|V|)$.
- Ordenar los lados es $O(|E| \log |E|)$.
- Recorrer todos los lados es $O(|E|)$, en línea 6.
- FIND-SET y UNION son operaciones de conjuntos disjuntos.
- El tiempo de las operaciones de conjuntos disjuntos con $|V|$ operaciones MAKE-SET es $O((|V| + |E|)\alpha(|V|))$.
- Como el grafo es conexo $|E| > |V| - 1$, entonces $O(|E|\alpha(V))$.
- $\alpha(|V|) = O(\log |V|) = O(\log |E|)$, entonces $O(|E|\alpha(|V|)) = O(|E| \log |E|)$.
- Como $|E| < |V|^2$, entonces $\log |E| = O(\log |V|)$.
- En consecuencia, el algoritmo de Kruskal es $O(|E| \log |V|)$.

Algoritmo de Prim

Algoritmo de Prim

Procedimiento ArbMimCobertor-Prim($G = (V, E)$, w, r)

inicio
  para cada $v \in V$ **hacer**
    $v.key \leftarrow \infty$;
    $v.pred \leftarrow NIL$;
  $r.key \leftarrow 0$;
  $Q \leftarrow V$;
  mientras $Q \neq \emptyset$ **hacer**
    $u \leftarrow$ `extraerMin`(Q);
    para cada $v \in G.adyacentes[u]$ **hacer**
      si $v \in Q \wedge w(u, v) < v.key$ **entonces**
        $v.pred \leftarrow u$;
        $v.key \leftarrow w(u, v)$; `/* Decrecimiento de clave */`

Ejemplo del algoritmo de Prim

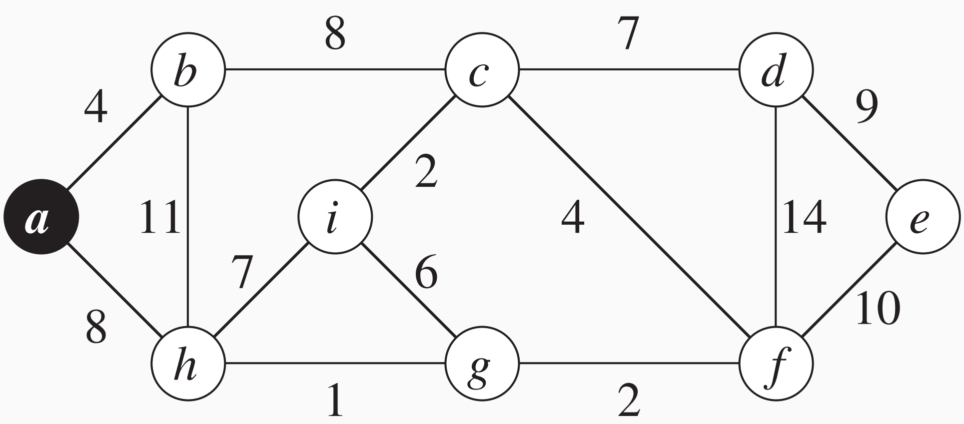

Figura 19: Comienza el algoritmo con raíz a. Fuente [1].

Ejemplo del algoritmo de Prim, cont.

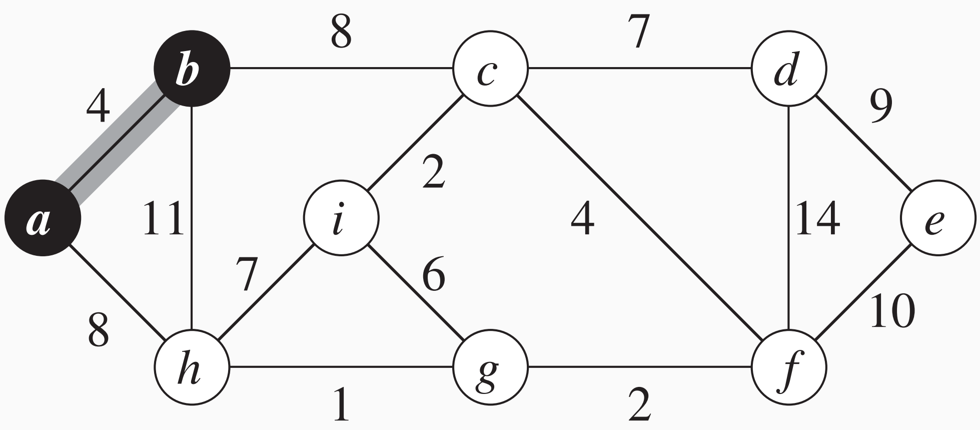

Figura 20: Se agrega el vértice *b* al árbol. Fuente [1].

Ejemplo del algoritmo de Prim, cont.

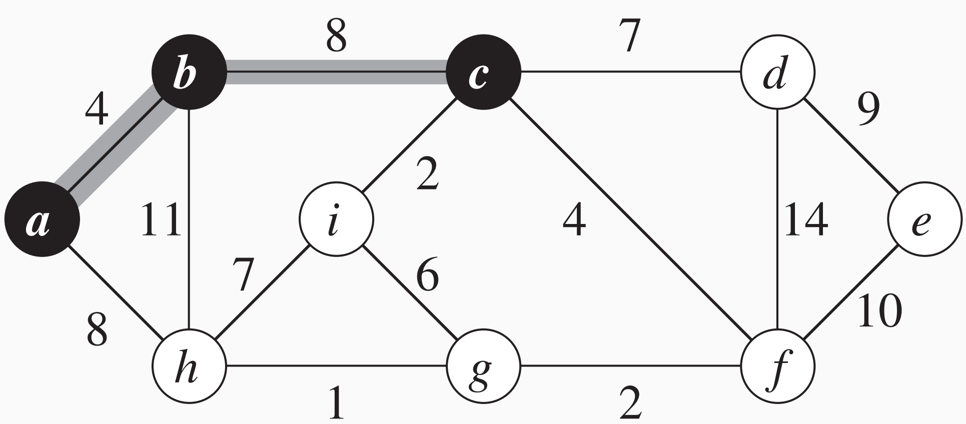

Figura 21: Se agrega el vértice *c* al árbol. Fuente [1].

Ejemplo del algoritmo de Prim, cont.

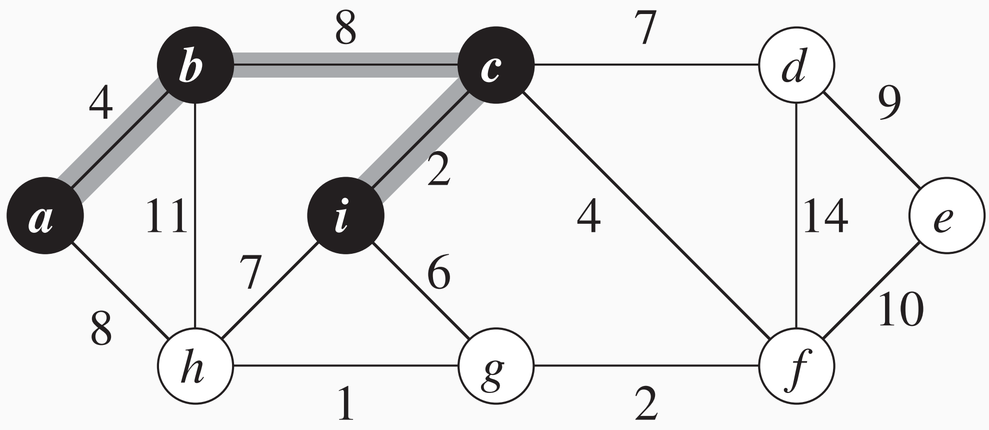

Figura 22: Se agrega el vértice *i* al árbol. Fuente [1].

Ejemplo del algoritmo de Prim, cont.

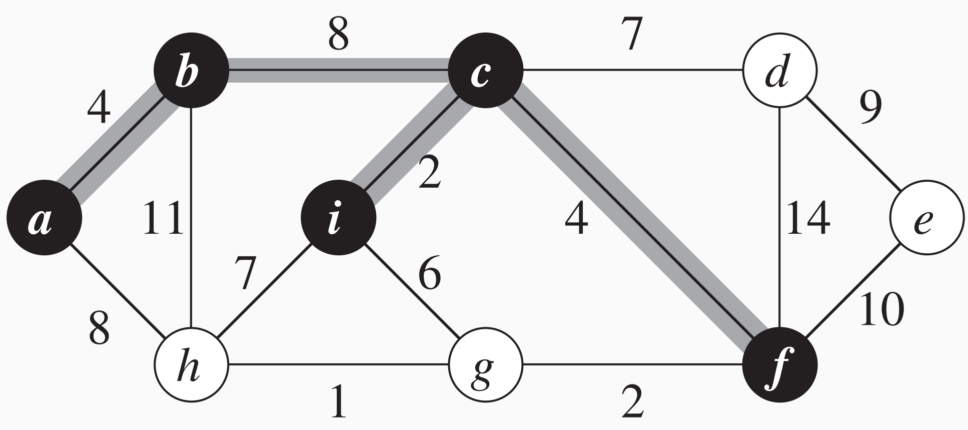

Figura 23: Se agrega el vértice *f* al árbol. Fuente [1].

Ejemplo del algoritmo de Prim, cont.

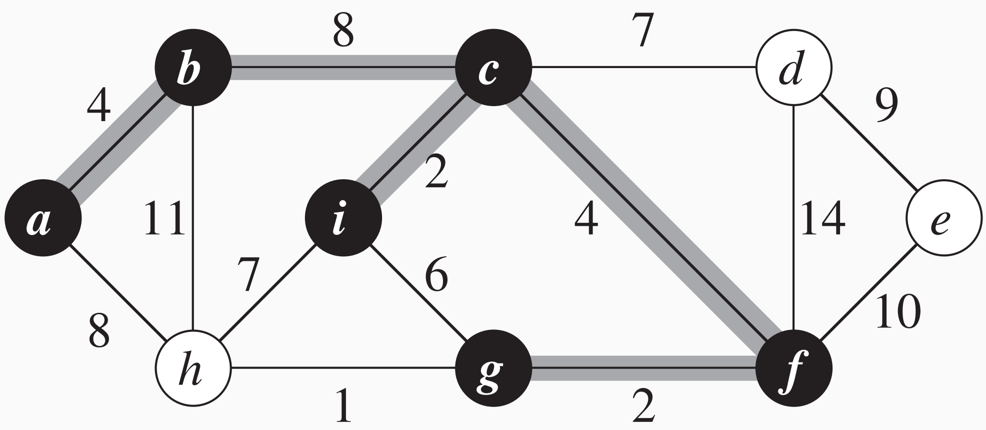

Figura 24: Se agrega el vértice g al árbol. Fuente [1].

Ejemplo del algoritmo de Prim, cont.

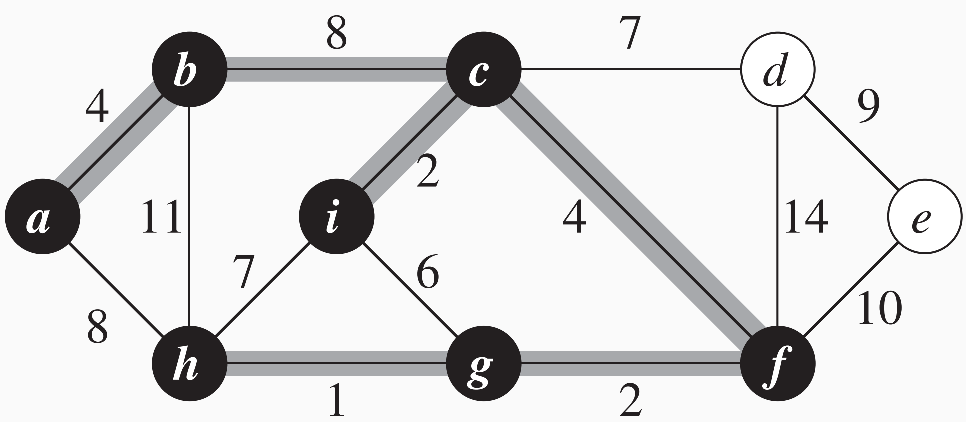

Figura 25: Se agrega el vértice h al árbol. Fuente [1].

Ejemplo del algoritmo de Prim, cont.

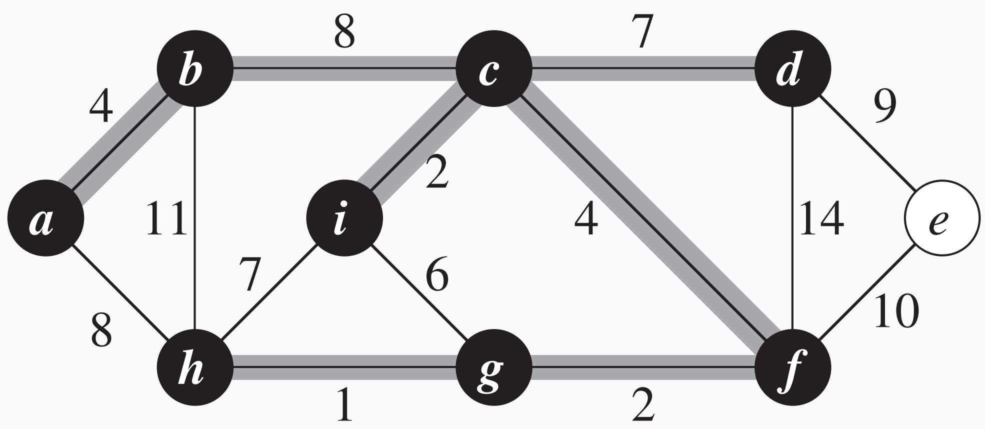

Figura 26: Se agrega el vértice *d* al árbol. Fuente [1].

Ejemplo del algoritmo de Prim, cont.

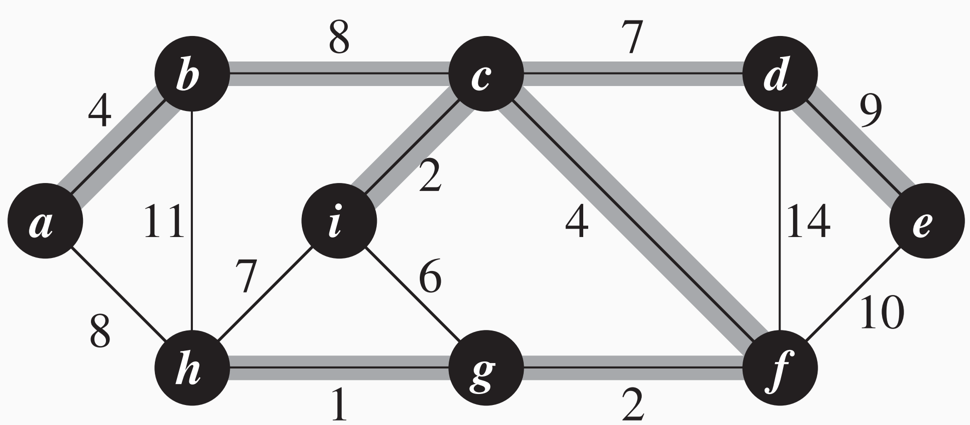

Figura 27: Se agrega el vértice *e* al árbol. Fuente [1].

Análisis del tiempo del algoritmo de Prim

Caso 1: usando como cola de prioridad un MIN-HEAP

- Construir el MIN-HEAP es $O(|V|)$.
- Cada operación de extraer el mínimo de la cola de prioridad es $O(\log|V|)$, entonces todas las llamadas es $O(|V|\log|V|)$.
- El ciclo for se ejecuta $O(|E|)$ porque se recorren todos los lados.
- El cambio de clave involucra un decrecimiento de la clave en la cola de prioridad esto es $O(\log|V|)$, entonces el tiempo total del ciclo for sería $O(|E|\log|V|)$.
- Entonces, el tiempo de Prim es $O(|V|\log|V| + |E|\log|V|) = O(|E|\log|V|)$.

Análisis del tiempo del algoritmo de Prim, cont.

Caso 2: usando como cola de prioridad un FIBONACCI HEAP

- En FIBONACCI HEAP extraer el mínimo es es $O(\log|V|)$, entonces todas las llamadas es $O(|V|\log|V|)$.
- En FIBONACCI HEAP el decrecimiento de la clave en la cola de prioridad es $O(1)$ en tiempo amortizado, entonces el tiempo total del ciclo `for` es $O(|E|)$.
- Entonces, el tiempo de Prim es $O(|V|\log|V| + |E|)$.

Referencias

[1] T. Cormen, C. Leirserson, R. Rivest, and C. Stein.
***Introduction to Algorithms*.**
McGraw Hill, 3ra edition, 2009.