Árboles de cobertura mínima

José Ortiz Bejar

Facultad de Ingeniería Eléctrica

Mayo 11, 2022

Outline

Árbol de cobertura mínima

- Dado un grafo pesado $G = (V, E)$
  - con con una función de "peso" $w : E \longrightarrow \mathbb{R}$

Árbol de cobertura mínima

- Dado un grafo pesado $G = (V, E)$
  - con con una función de "peso" $w : E \longrightarrow \mathbb{R}$
- Encontrar un sub-conjunto acíclico $T \subseteq E$
  - un ***árbol***

Árbol de cobertura mínima

- Dado un grafo pesado $G = (V, E)$
  - con con una función de "peso" $w : E \longrightarrow \mathbb{R}$
- Encontrar un sub-conjunto acíclico $T \subseteq E$
  - un ***árbol***
- T "cubre" todo el grafo G (e.i., conecta todos los vértices)
  - un ***árbol de cobertura***

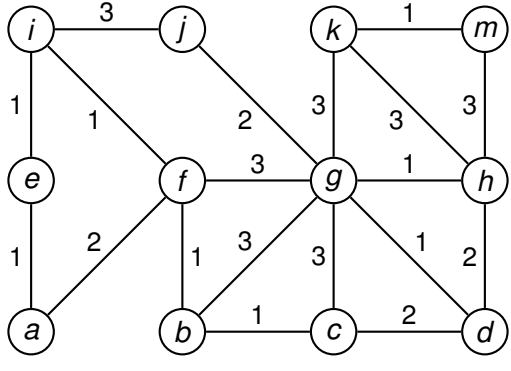
Árbol de cobertura mínima

- Dado un grafo pesado $G = (V, E)$
  - con con una función de "peso" $w : E \to \mathbb{R}$
- Encontrar un sub-conjunto acíclico $T \subseteq E$
  - un ***árbol***
- T "cubre" todo el grafo G (e.i., conecta todos los vértices)
  - un ***árbol de cobertura***
- el peso total del árbol T es mínimo

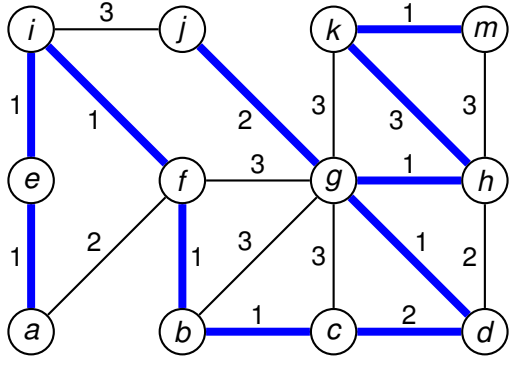
$$w(T) = \sum_{(u,v) \in T} w(u, v)$$

  - un ***árbol de cobertura con peso mínimo***, o "árbol de cobertura mínima"

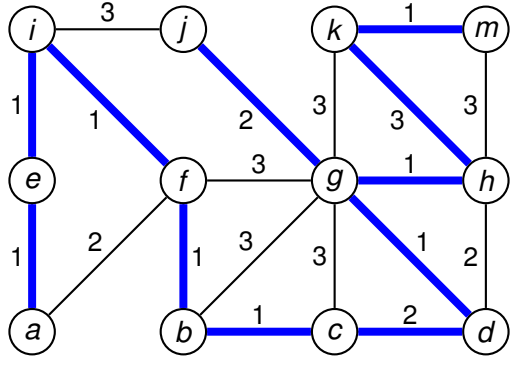
Ejemplo



Ejemplo

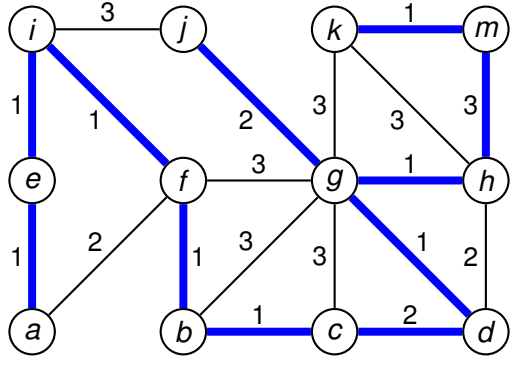


Ejemplo



- Puede haber muchos árboles de cobertura mínima.

Ejemplo



- Puede haber muchos árboles de cobertura mínima.

Idea General

- Construir T agregando un eje a la vez

Idea General

- Construir T agregando un eje a la vez
- En cada paso, agregar $e \in E$ tal que
  - e es la arista de ***costo mínimo***
  - e ***que no genere un ciclo***
- Terminar cuando todos lo vértices estén en T

Idea General

- Construir T agregando un eje a la vez
- En cada paso, agregar $e \in E$ tal que
  - e es la arista de ***costo mínimo***
  - e ***que no genere un ciclo***
- Terminar cuando todos lo vértices estén en T
- Es un *algoritmo "voraz" (ávido)*

Idea General

- Construir T agregando un eje a la vez
- En cada paso, agregar $e \in E$ tal que
  - e es la arista de ***costo mínimo***
  - e ***que no genere un ciclo***
- Terminar cuando todos lo vértices estén en T
- Es un *algoritmo "voraz" (ávido)*
- ¿Funciona?

```
Generic-MST(G, w) 1  A = ∅
                  2  while A no sea un árbol de cobertura
                  3      identificar una arista segura e = (u, v)
                  4      A = A ∪ {e}
```

```
Generic-MST(G, w) 1  A = ∅
                  2  while A no sea un árbol de cobertura
                  3      identificar una arista segura e = (u, v)
                  4      A = A ∪ {e}
```

- ***Invariante:*** A es un sub-conjunto de un árbol de cobertura mínima.

Idea refinada

```
Generic-MST(G, w) 1  A = ∅
                  2  while A no sea un árbol de cobertura
                  3      identificar una arista segura e = (u, v)
                  4      A = A ∪ {e}
```

- ***Invariante:*** A es un sub-conjunto de un árbol de cobertura mínima.
- Una arista ***segura*** es aquella que mantiene el invariante
  - e es tal que, si A es un sub-conjunto de árbol de cobertura mínima, entonces $A \cup \{e\}$ es también un sub-conjunto de un árbol de cobertura.

Idea refinada

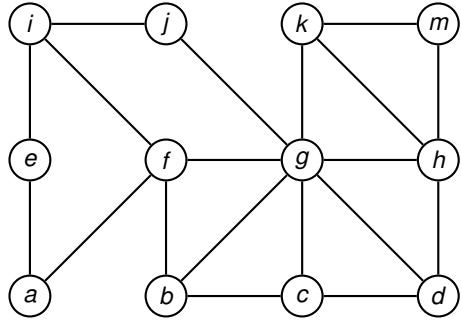
```
Generic-MST(G, w) 1  A = ∅
                  2  while A no sea un árbol de cobertura
                  3      identificar una arista segura e = (u, v)
                  4      A = A ∪ {e}
```

- ***Invariante:*** A es un sub-conjunto de un árbol de cobertura mínima.
- Una arista ***segura*** es aquella que mantiene el invariante
  - e es tal que, si A es un sub-conjunto de árbol de cobertura mínima, entonces $A \cup \{e\}$ es también un sub-conjunto de un árbol de cobertura.
  - similar a la *definición* de un algoritmo voraz

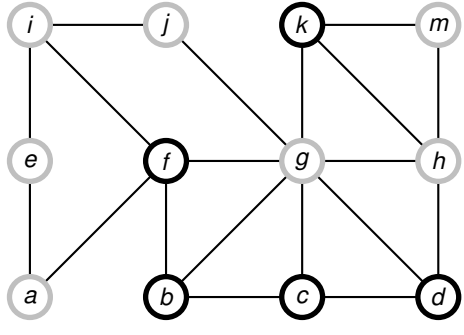
Definiciones preliminares

- Un ***cut*** $(S, V - S)$ es un grafo no dirigido $G = (V, E)$ es una *partición* de V

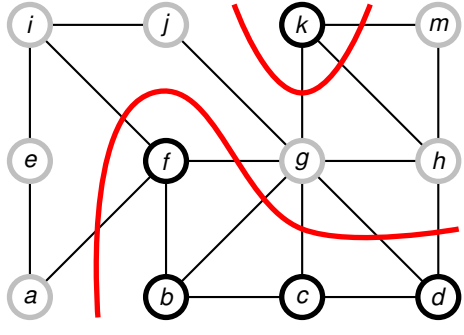
- Un ***cut*** $(S, V - S)$ es un grafo no dirigido $G = (V, E)$ es una *partición* de V



- Un ***cut*** $(S, V - S)$ es un grafo no dirigido $G = (V, E)$ es una *partición* de V

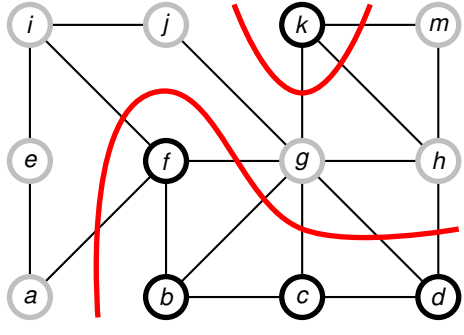


- Un ***cut*** $(S, V - S)$ es un grafo no dirigido $G = (V, E)$ es una *partición* de V



Definiciones preliminares

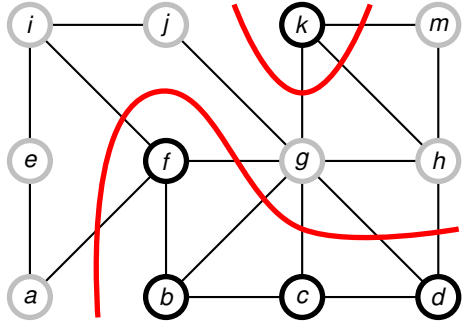
- Un ***cut*** $(S, V - S)$ es un grafo no dirigido $G = (V, E)$ es una *partición* de V



- Una arista $e = (u, v)$ *cruza* el cut $(S, V - S)$ si $u \in S$ y $v \in V - S$, o vice-versa

Definiciones preliminares

- Un ***cut*** $(S, V - S)$ es un grafo no dirigido $G = (V, E)$ es una *partición* de V



- Una arista $e = (u, v)$ *cruza* el cut $(S, V - S)$ si $u \in S$ y $v \in V - S$, o vice-versa

- Un cut $(S, V - S)$ *respeta* un conjunto de aristas A si ninguna arista en A cruza el corte

Determinando una arista segura

```
Generic-MST(G, w) 1  A = ∅
                  2  while A no sea un árbol de cobertura
                  3      encontrar una arista segura e = (u, v)
                  4      A = A ∪ {e}
```

Determinando una arista segura

```
Generic-MST(G, w) 1  A = ∅
                  2  while A no sea un árbol de cobertura
                  3      encontrar una arista segura e = (u, v)
                  4      A = A ∪ {e}
```

- Sea $(S, V - S)$ un cut de G que respeta A

Determinando una arista segura

```
Generic-MST(G, w) 1  A = ∅
                  2  while A no sea un árbol de cobertura
                  3      encontrar una arista segura e = (u, v)
                  4      A = A ∪ {e}
```

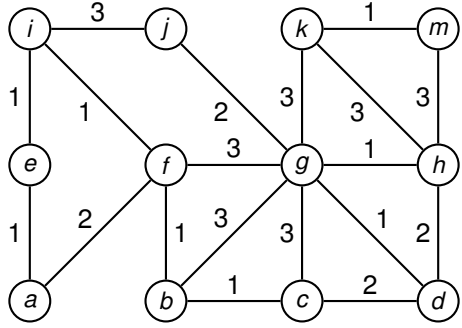
- Sea $(S, V - S)$ un cut de G que respeta A
- Una arista de costo mínimo e que cruza $(S, V - S)$ es segura.

Example

- Sea $(S, V - S)$ un cut de G que respeta A
- Una arista de costo mínimo e que cruza $(S, V - S)$ es segura.

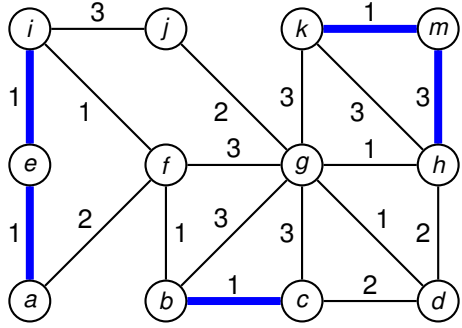
Example

- Sea $(S, V - S)$ un cut de G que respeta A
- Una arista de costo mínimo e que cruza $(S, V - S)$ es segura.



Example

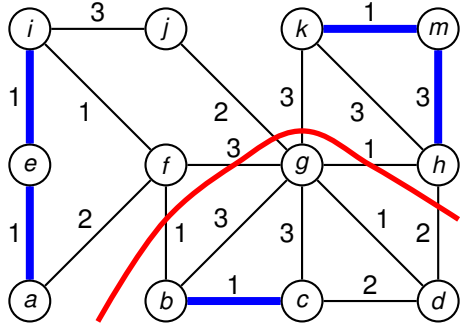
- Sea $(S, V - S)$ un cut de G que respeta A
- Una arista de costo mínimo e que cruza $(S, V - S)$ es segura.



- $A = \{(a, e), (e, i), (b, c), (k, m), (m, h)\}$

Example

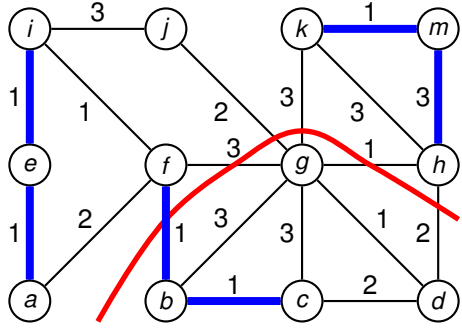
- Sea $(S, V - S)$ un cut de G que respeta A
- Una arista de costo mínimo e que cruza $(S, V - S)$ es segura.



- $A = \{(a, e), (e, i), (b, c), (k, m), (m, h)\}$
- Seleccionado el cut $(\{a, e, f, h, i, j, k, m\}, \{b, c, d, g\})$

Example

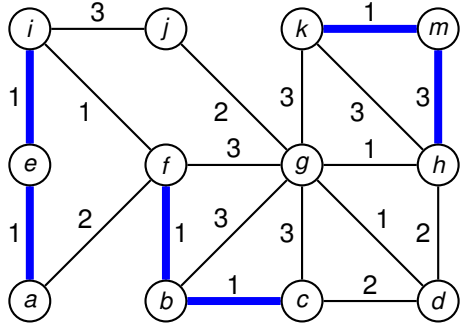
- Sea $(S, V - S)$ un cut de G que respeta A
- Una arista de costo mínimo e que cruza $(S, V - S)$ es segura.



- $A = \{(a, e), (e, i), (b, c), (k, m), (m, h)\}$
- Seleccionado el cut $(\{a, e, f, h, i, j, k, m\}, \{b, c, d, g\})$
- (f, b) es una arista segura

Example

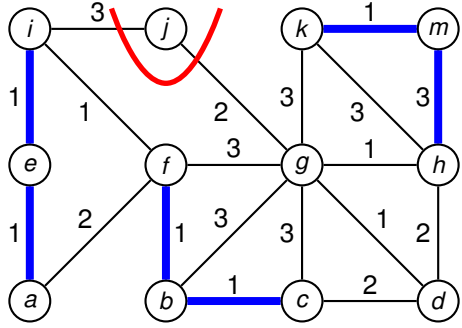
- Sea $(S, V - S)$ un cut de G que respeta A
- Una arista de costo mínimo e que cruza $(S, V - S)$ es segura.



- $A = \{(a, e), (e, i), (b, c), (k, m), (m, h)\}$
- Seleccionado el cut $(\{a, e, f, h, i, j, k, m\}, \{b, c, d, g\})$
- (f, b) es una arista segura ; (g, h) también es una arista segura

Example

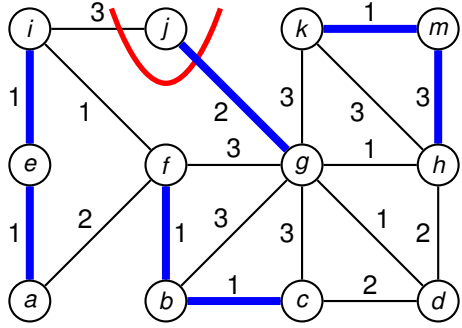
- Sea $(S, V - S)$ un cut de G que respeta A
- Una arista de costo mínimo e que cruza $(S, V - S)$ es segura.



- $A = \{(a, e), (e, i), (b, c), (k, m), (m, h)\}$
- Seleccionado el cut $(\{a, e, f, h, i, j, k, m\}, \{b, c, d, g\})$
- (f, b) es una arista segura ; (g, h) también es una arista segura

Example

- Sea $(S, V - S)$ un cut de G que respeta A
- Una arista de costo mínimo e que cruza $(S, V - S)$ es segura.



- $A = \{(a, e), (e, i), (b, c), (k, m), (m, h)\}$
- Seleccionado el cut $(\{a, e, f, h, i, j, k, m\}, \{b, c, d, g\})$
- (f, b) es una arista segura ; (g, h) también es una arista segura

- Algoritmo de Kruskal (1956)
  - basado en el algoritmo genérico
  - Construye un ***bosque*** A de forma incremental

Algoritmos

- Algoritmo de Kruskal (1956)
  - basado en el algoritmo genérico
  - Construye un ***bosque*** *A* de forma incremental
- Algoritmo de Prim (1957)
  - basado en el algoritmo genérico
  - Construye un ***árbol*** *A* de forma incremental

Estructura de datos para conjuntos disjuntos

- *Make-Set*(x) crear un conjunto que contenga x

Estructura de datos para conjuntos disjuntos

- *Make-Set*(x) crear un conjunto que contenga x
- *Find-Set*(x) regresa un *representante* del conjunto que contiene a x
  - $x, y \in S \Rightarrow$ *Find-Set*$(x) =$ *Find-Set*(y)
  - $x \in S_1 \wedge y \in S_2 \wedge S_1 \neq S_2 \Rightarrow$ *Find-Set*$(x) \neq$ *Find-Set*(y)

Estructura de datos para conjuntos disjuntos

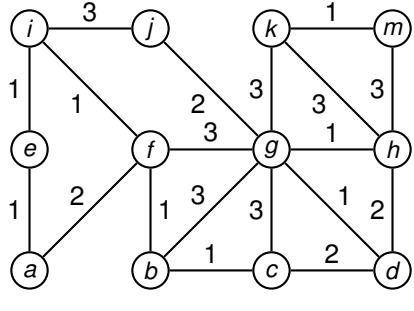
- *Make-Set*(x) crear un conjunto que contenga x
- *Find-Set*(x) regresa un *representante* del conjunto que contiene a x
  - $x, y \in S \Rightarrow$ *Find-Set*$(x) =$ *Find-Set*(y)
  - $x \in S_1 \wedge y \in S_2 \wedge S_1 \neq S_2 \Rightarrow$ *Find-Set*$(x) \neq$ *Find-Set*(y)
- *Union*(x, y) une los conjuntos que contienen a x y y

Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3      Make-Set(v)                    // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6      if Find-Set(u) ≠ Find-Set(v)
                  7          A = A ∪ {(u, v)}
                  8          Union(u, v)
```

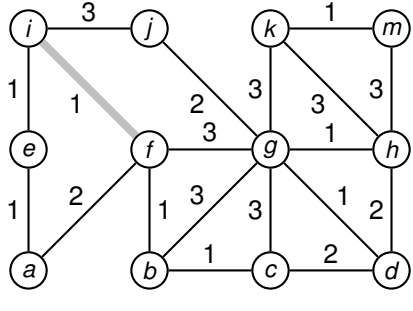
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



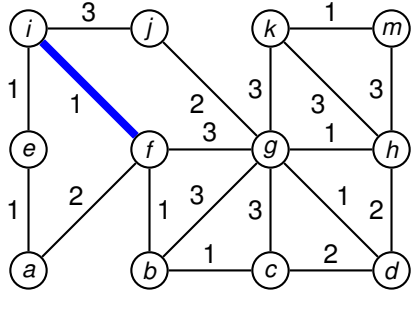
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



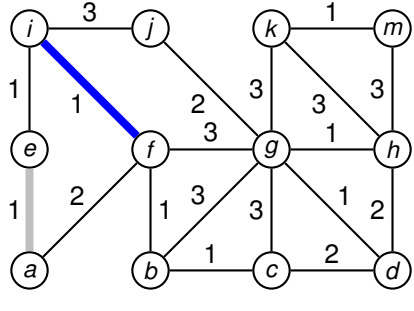
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



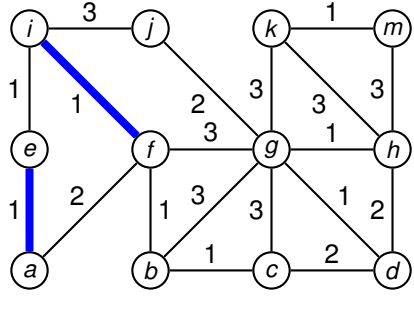
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



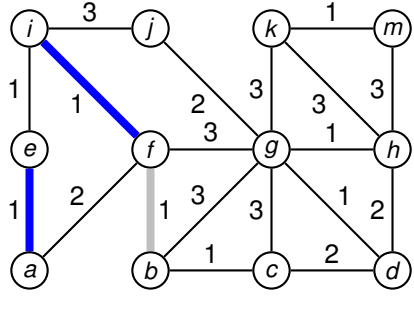
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



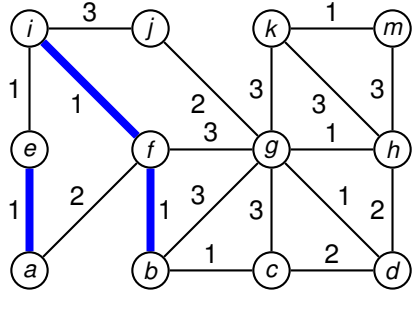
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



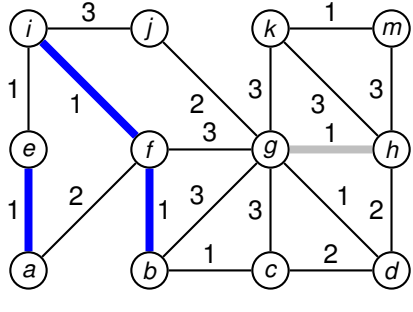
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



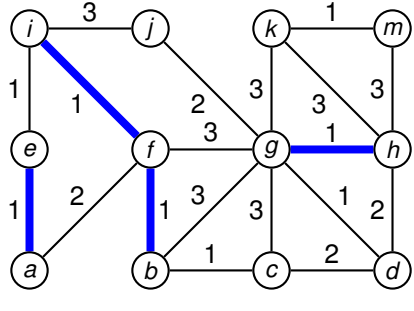
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



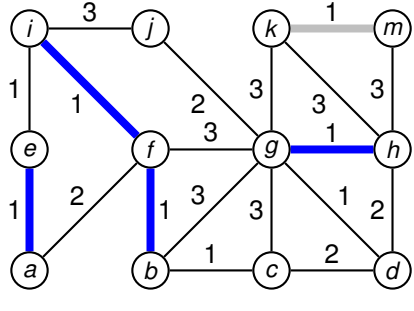
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



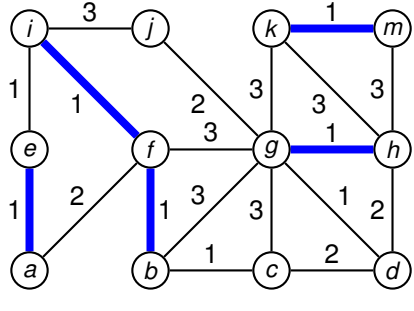
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



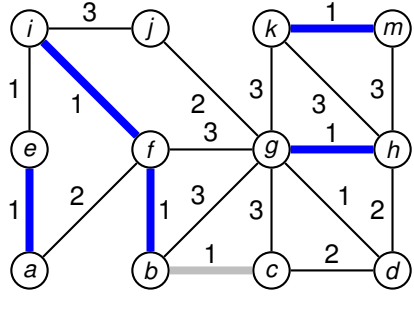
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



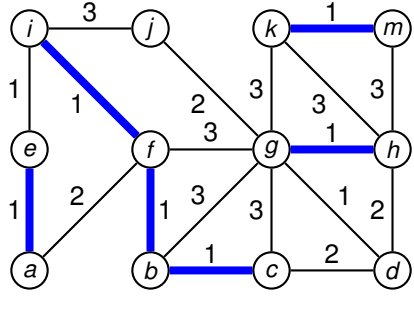
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



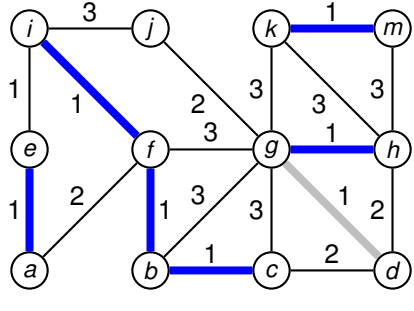
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



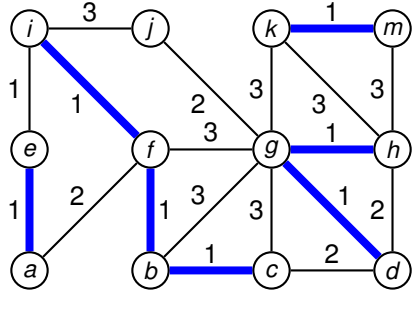
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



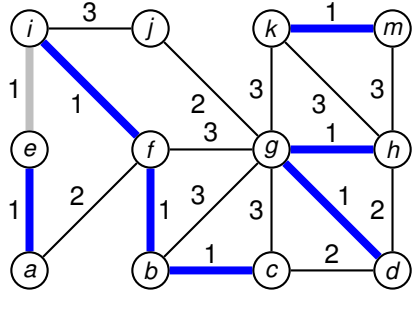
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



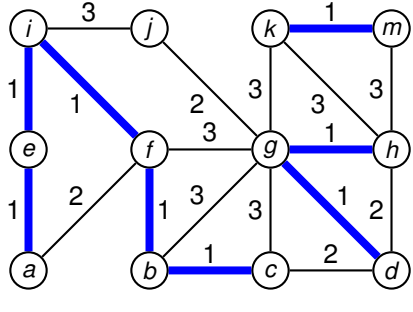
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



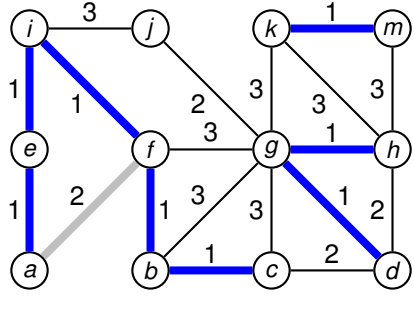
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



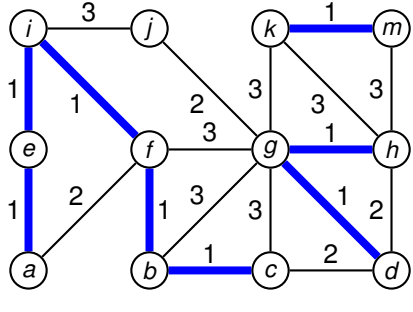
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



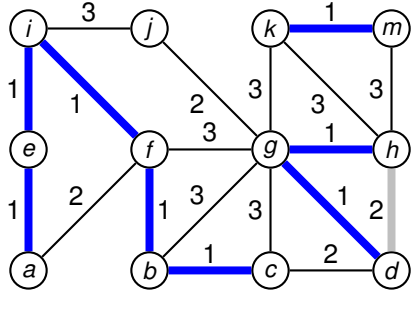
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



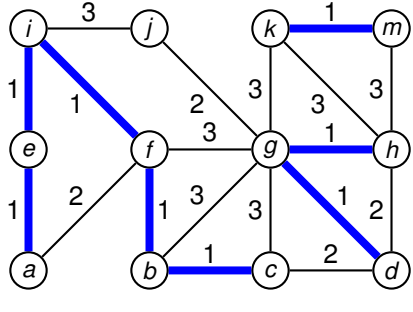
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



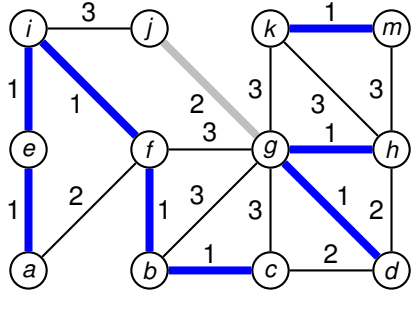
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



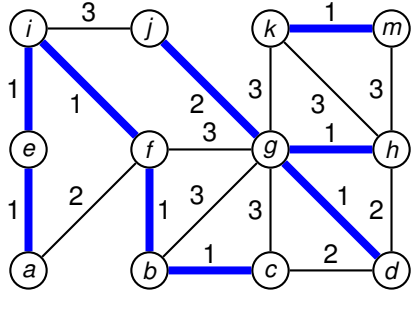
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



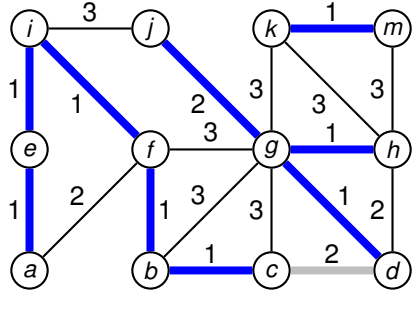
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



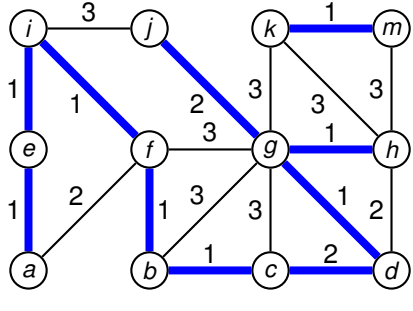
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



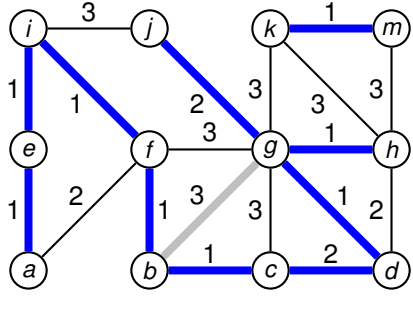
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



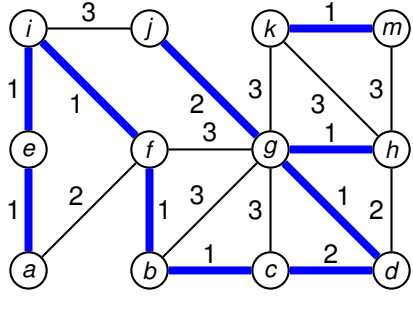
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



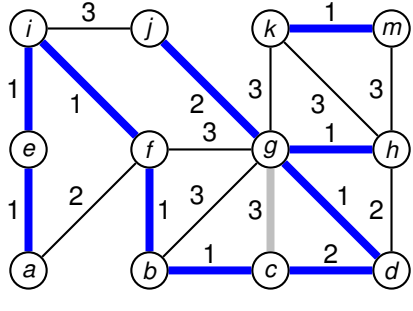
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



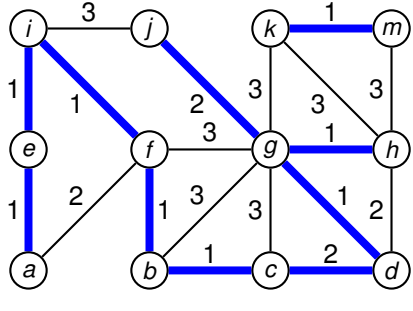
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



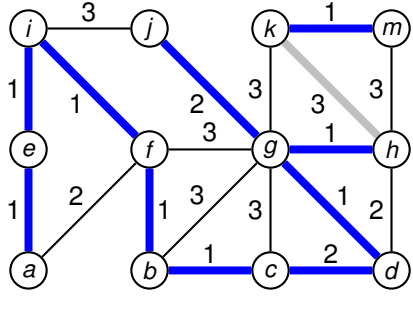
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



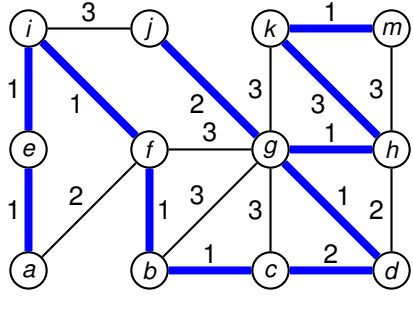
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



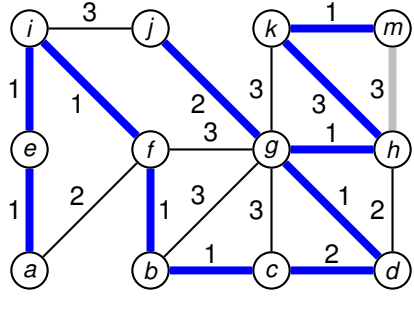
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



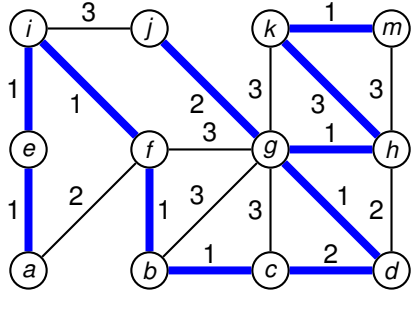
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



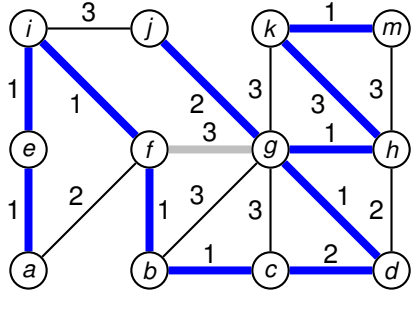
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



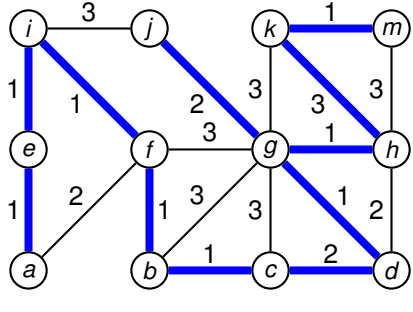
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



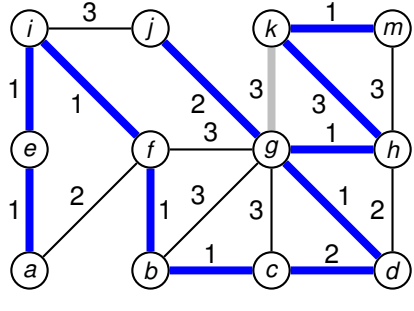
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



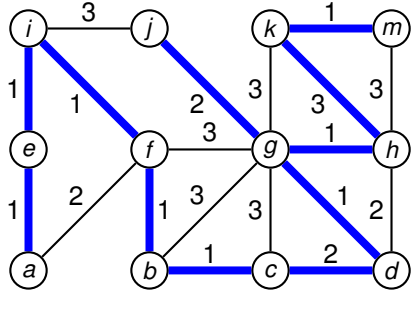
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



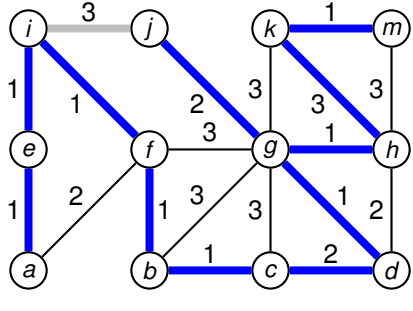
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



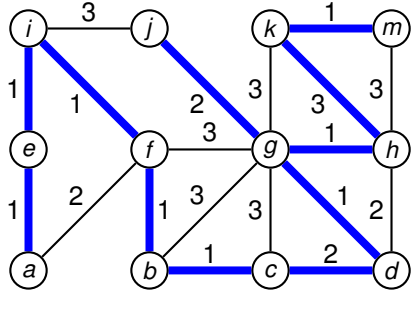
Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



Algoritmo de Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3       Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6       if Find-Set(u) ≠ Find-Set(v)
                  7            A = A ∪ {(u, v)}
                  8            Union(u, v)
```



Complejidad de MST-Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3      Make-Set(v)                    // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6      if Find-Set(u) ≠ Find-Set(v)
                  7          A = A ∪ {(u, v)}
                  8          Union(u, v)
```

Complejidad de MST-Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3      Make-Set(v)                    // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6      if Find-Set(u) ≠ Find-Set(v)
                  7          A = A ∪ {(u, v)}
                  8          Union(u, v)
```

- El procedimiento **Make-Set** se ejecuta $|V|$ veces (ciclo en las lineas 2–3)

Complejidad de MST-Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3      Make-Set(v)                    // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6      if Find-Set(u) ≠ Find-Set(v)
                  7          A = A ∪ {(u, v)}
                  8          Union(u, v)
```

- El procedimiento **Make-Set** se ejecuta $|V|$ veces (ciclo en las lineas 2–3)
- Ordenar E consume $O(|E| \log |E|)$ (linea 4)

Complejidad de MST-Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3      Make-Set(v)                    // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6      if Find-Set(u) ≠ Find-Set(v)
                  7          A = A ∪ {(u, v)}
                  8          Union(u, v)
```

- El procedimiento **Make-Set** se ejecuta $|V|$ veces (ciclo en las lineas 2–3)
- Ordenar E consume $O(|E| \log |E|)$ (linea 4)
- El procedimiento **Find-Set** se ejecuta $2|E|$ veces

Complejidad de MST-Kruskal

```
MST-Kruskal(G, w) 1  A = ∅
                  2  for cada vértice v ∈ V(G)
                  3      Make-Set(v)              // estructura para disjoint-set
                  4  ordenar E por peso w de forma ascendente
                  5  for cada vértice (u, v) ∈ E, seleccionado en orden ascendente de w
                  6      if Find-Set(u) ≠ Find-Set(v)
                  7          A = A ∪ {(u, v)}
                  8          Union(u, v)
```

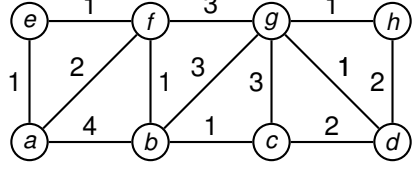
- El procedimiento **Make-Set** se ejecuta $|V|$ veces (ciclo en las lineas 2–3)
- Ordenar E consume $O(|E| \log |E|)$ (linea 4)
- El procedimiento **Find-Set** se ejecuta $2|E|$ veces
- **Union** se ejecuta $O(|E|)$ veces.

Algoritmo de Prim

```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```

Algoritmo de Prim

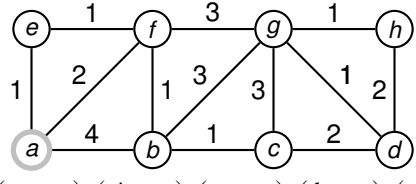
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(a, 0, \cdot), (b, \infty, \cdot), (c, \infty, \cdot), (d, \infty, \cdot), (e, \infty, \cdot), (f, \infty, \cdot), (g, \infty, \cdot), (h, \infty, \cdot)\}$

Algoritmo de Prim

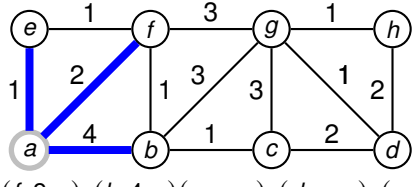
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(b, \infty, \cdot), (c, \infty, \cdot), (d, \infty, \cdot), (e, \infty, \cdot), (f, \infty, \cdot), (g, \infty, \cdot), (h, \infty, \cdot)\}$

Algoritmo de Prim

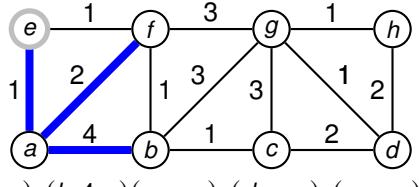
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(e, 1, a), (f, 2, a), (b, 4, a)(c, \infty, \cdot), (d, \infty, \cdot), (g, \infty, \cdot), (h, \infty, \cdot)\}$

Algoritmo de Prim

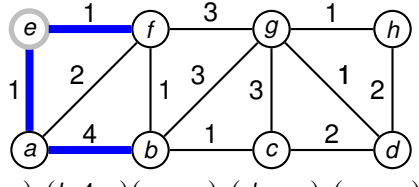
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(f, 2, a), (b, 4, a)(c, \infty, \cdot), (d, \infty, \cdot), (g, \infty, \cdot), (h, \infty, \cdot)\}$

Algoritmo de Prim

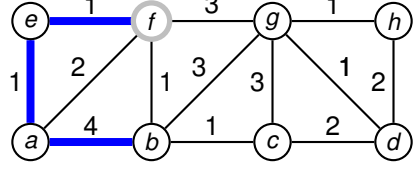
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(f, 1, e), (b, 4, a)(c, \infty, \cdot), (d, \infty, \cdot), (g, \infty, \cdot), (h, \infty, \cdot)\}$

Algoritmo de Prim

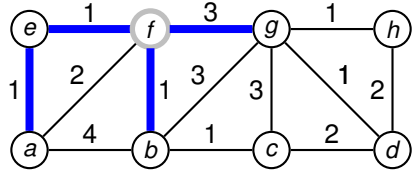
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(b, 4, a)(c, \infty, \cdot), (d, \infty, \cdot), (g, \infty, \cdot), (h, \infty, \cdot)\}$

Algoritmo de Prim

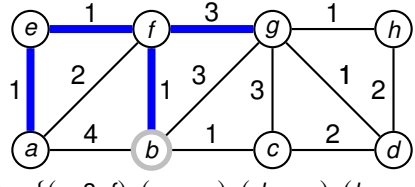
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(b, 1, f), (g, 3, f), (c, \infty, \cdot), (d, \infty, \cdot), (h, \infty, \cdot)\}$

Algoritmo de Prim

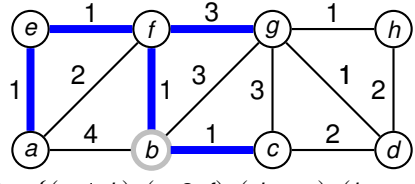
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(g, 3, f), (c, \infty, \cdot), (d, \infty, \cdot), (h, \infty, \cdot)\}$

Algoritmo de Prim

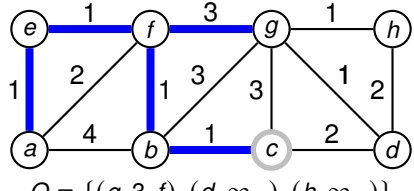
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(c, 1, b), (g, 3, f), (d, \infty, \cdot), (h, \infty, \cdot)\}$

Algoritmo de Prim

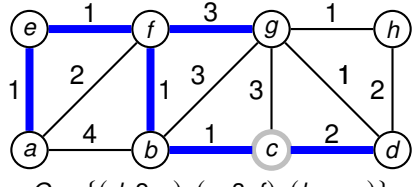
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(g, 3, f), (d, \infty, \cdot), (h, \infty, \cdot)\}$

Algoritmo de Prim

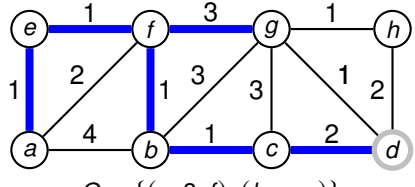
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(d, 2, c), (g, 3, f), (h, \infty, \cdot)\}$

Algoritmo de Prim

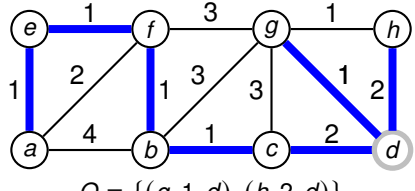
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(g, 3, f), (h, \infty, \cdot)\}$

Algoritmo de Prim

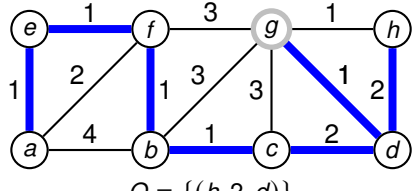
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(g, 1, d), (h, 2, d)\}$

Algoritmo de Prim

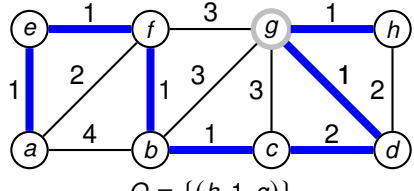
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(h, 2, d)\}$

Algoritmo de Prim

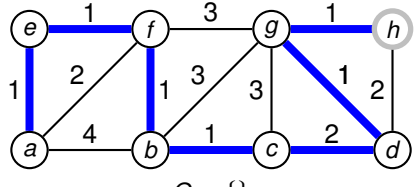
```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{(h, 1, g)\}$

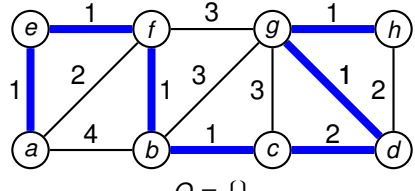
Algoritmo de Prim

```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



Algoritmo de Prim

```
MST-Prim(G, w, r)
 1  for each vertex u ∈ V(G)
 2      key[u] = ∞
 3      π(u) = nil
 4  key[r] = 0
 5  Q = V(G)
 6  while Q ≠ ∅
 7      u = Extract-Min(Q) // min by key[u]
 8      for each v ∈ Adj[u]
 9          if v ∈ Q ∧ w(u, v) < key[v]
10              π(v) = u
11              key[v] = w(u, v)
```



$Q = \{\}$