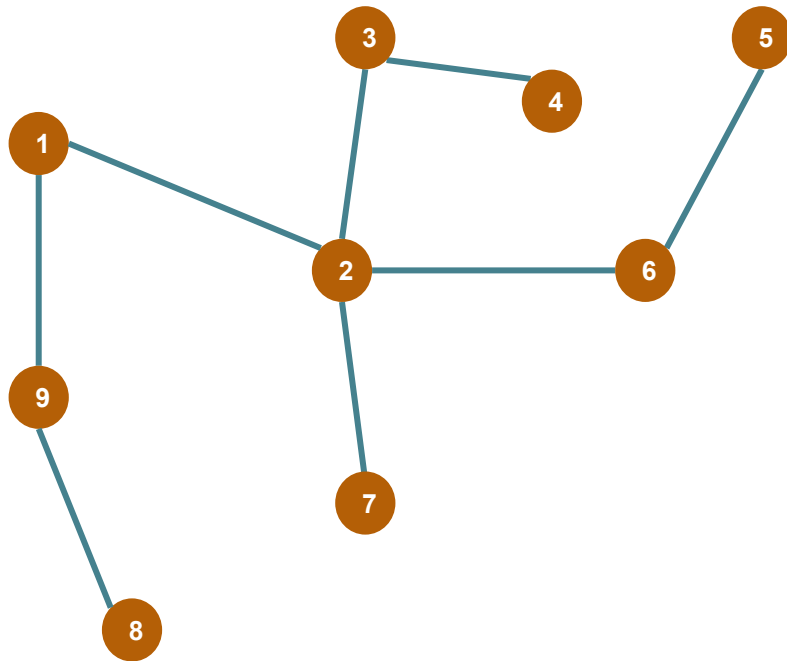


AED3 > Clase 6 > AGM

# Árboles: Repaso

## Definición 11: Más tipos de grafos: Árboles

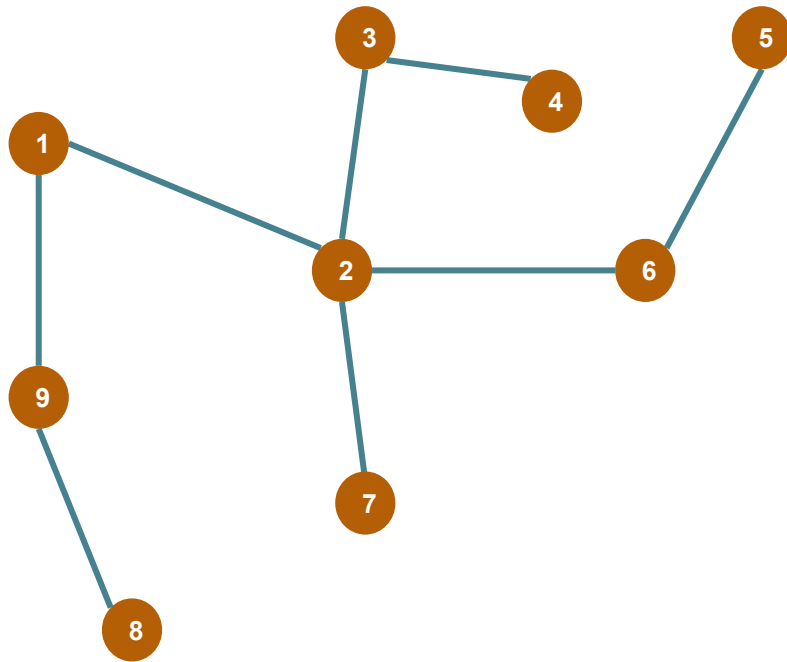
1. Grafo conectado y acíclico.
2. Si saco cualquier arista se desconecta.
3. Si agrego una arista cualquiera se forma un ciclo.



# Árboles: Repaso

## Definición 1:

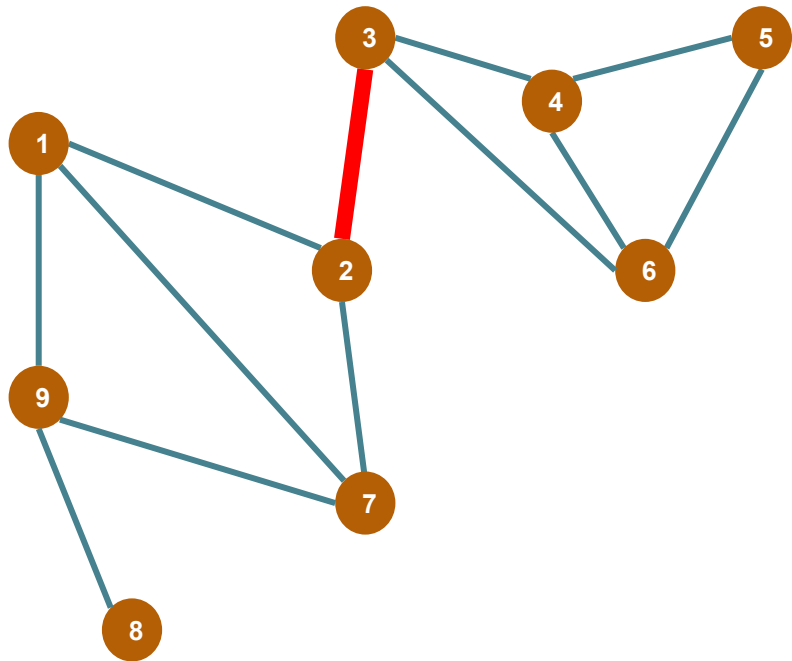
Un **árbol** es un grafo conexo sin circuitos simples.



# Árboles: Repaso

## Definición 2:

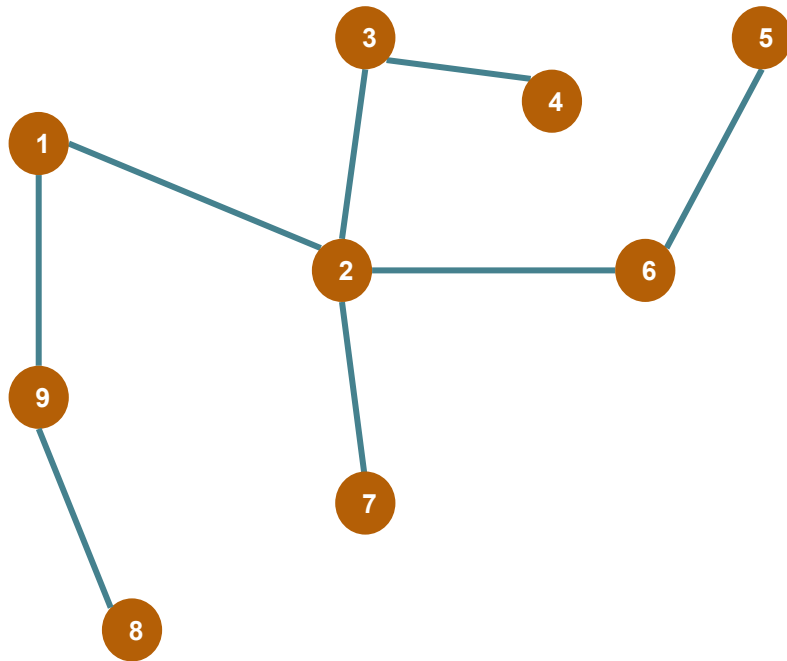
Una arista  $e$  de  $G$  es un **punte** si  $G - \{e\}$  tiene más componente que  $G$ . Es decir, si la saco desconecta.



# Árboles: Repaso

## Teorema: Equivalencias

1.  $G$  es un árbol (*grafo conexo sin circuitos simples*).
2.  $G$  es un grafo sin circuitos simples y  $e$  una arista tq  $e \notin E$ .  $G+e = (V, E+\{e\})$  tiene exactamente un circuito simple, y ese circuito contiene a  $e$ . *Es decir, si agrego una arista cualquiera se forma un ciclo.*
3.  $\exists$  exactamente un camino simple entre todo par de nodos.
4.  $G$  es conexo, pero si se quita cualquier arista queda un grafo no conexo. *Es decir, si saco cualquier arista se desconecta, o toda arista es puente.*



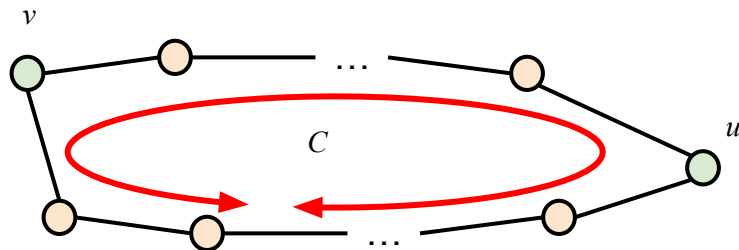
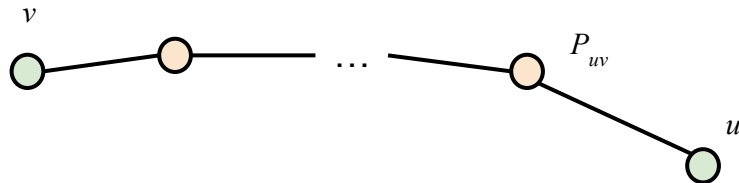
# Árboles: Repaso

## Lema 1:

La unión entre dos caminos simples distintos entre  $u$  y  $v$  contiene un circuito simple.

$P_{uv}, Q_{uv}$ : Caminos simples

$C : P_{uv} + Q_{vu}$ : Circuito simple



# Árboles: Repaso

**Lema 2:** Sea  $G = (V, E)$  un grafo conexo y  $e=(v,u) \in E$ .

$G - e = (V, E \setminus \{e\})$  es conexo  $\Leftrightarrow e \in C$  : circuito simple de  $G$ .

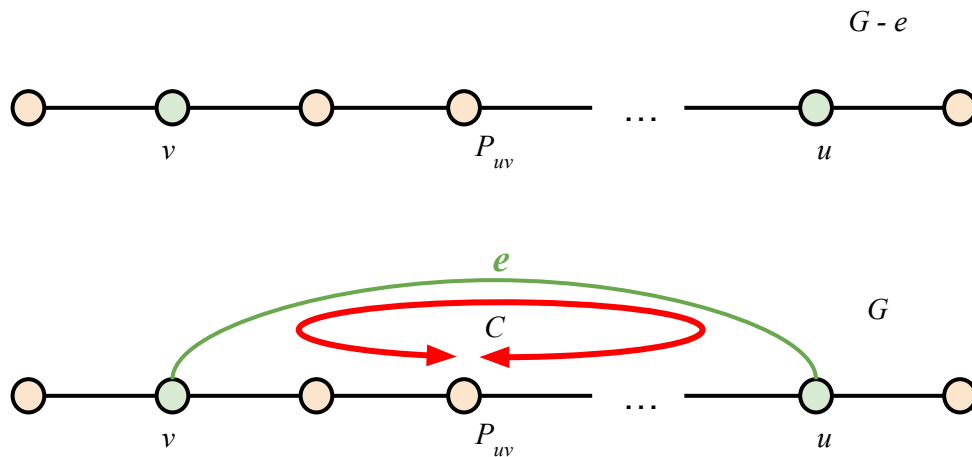
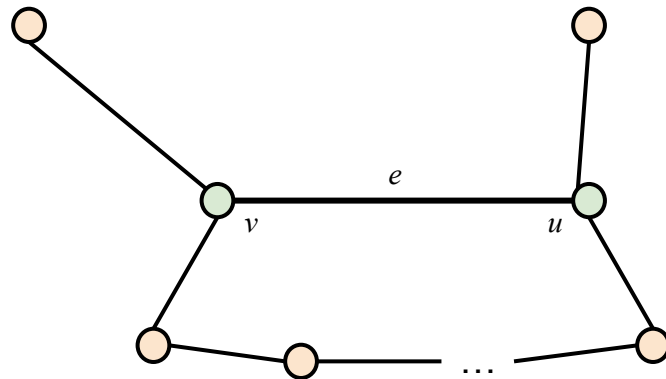
(  $e=(v,u) \in E$  es puente  $\Leftrightarrow e$  no pertenece a un circuito simple de  $G$  ).

**Demostración ( $\Rightarrow$ ):**

$G - e$  es conexo  $\Rightarrow$

Existe un camino simple entre  $u$  y  $v$  ( $P_{uv}$ ) que no usa  $e$ .

Si agrego  $e$  se forma un circuito simple  $C : P_{uv} + e$



# Árboles: Repaso

**Lema 2:** Sea  $G = (V, E)$  un grafo conexo y  $e = (v, u) \in E$ .

$G - e = (V, E \setminus \{e\})$  es conexo  $\Leftrightarrow e \in C$  : circuito simple de  $G$ .

(  $e = (v, u) \in E$  es puente  $\Leftrightarrow e$  no pertenece a un circuito simple de  $G$  ).

**Demostración ( $\Leftarrow$ ):**

Sea  $C$  un circuito simple que contiene a  $e = (u, v) \Rightarrow$

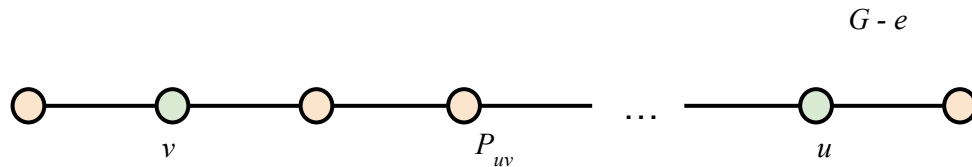
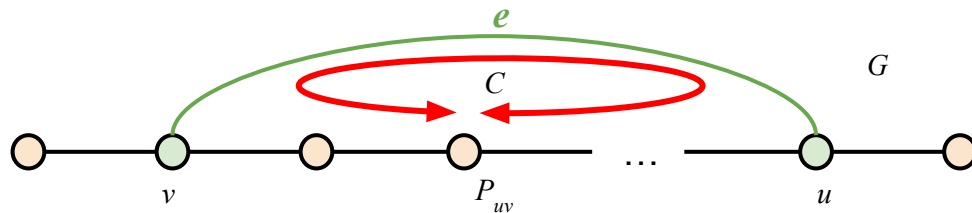
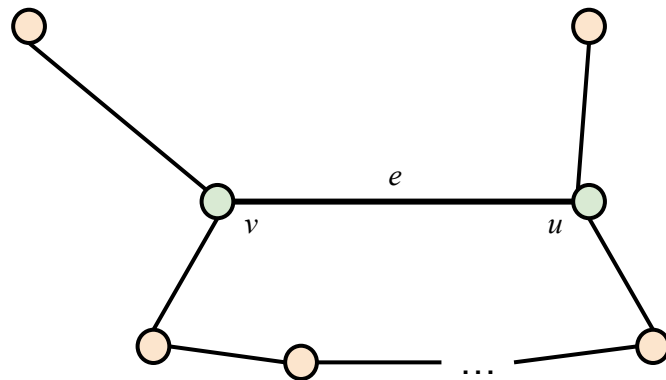
$C : P_{uv} + e$ , tq  $P_{uv}$  no usa  $e$ .

$G$  es conexo  $\Rightarrow$  Existe un camino entre todo par de vértices.

Si no usa  $e$  lo conservo en  $G - e$ .

Si usa  $e$ , la reemplazo por  $P_{uv}$  en  $G - e$ .  $\Rightarrow$

Existe un camino entre todo par de vértices en  $G - e$ .  $\Rightarrow G - e$  es conexo

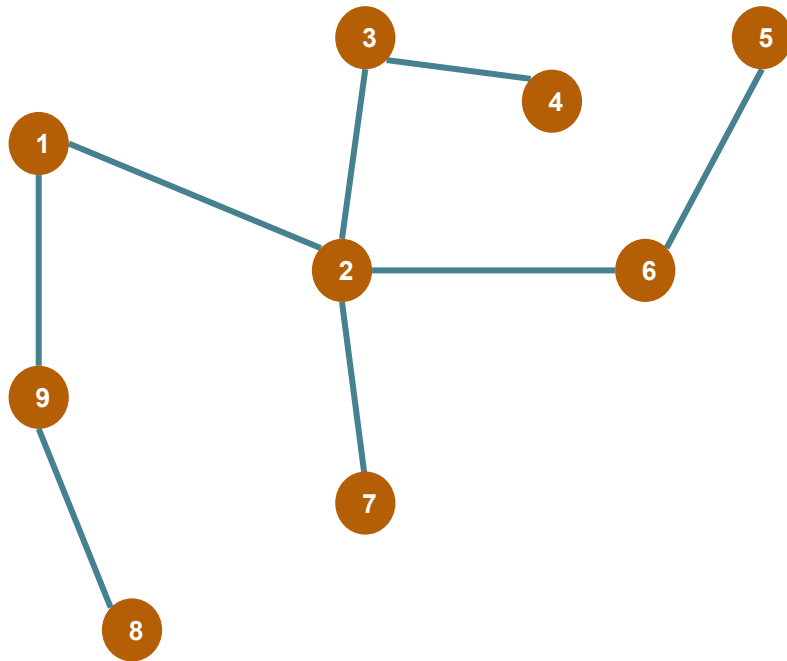




# Árboles: Repaso

## Teorema: Equivalencias

1.  $G$  es un árbol (*grafo conexo sin circuitos simples*).
2.  $G$  es un grafo sin circuitos simples y  $e$  una arista tq  $e \notin E$ .  $G+e = (V, E+\{e\})$  tiene exactamente un circuito simple, y ese circuito contiene a  $e$ . *Es decir, si agrego una arista cualquiera se forma un ciclo.*
3.  $\exists$  exactamente un camino simple entre todo par de nodos.
4.  $G$  es conexo, pero si se quita cualquier arista queda un grafo no conexo. *Es decir, si saco cualquier arista se desconecta, o toda arista es puente.*



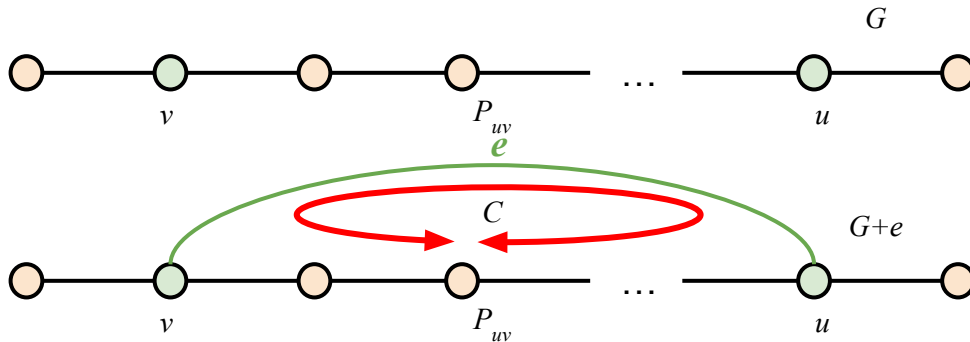
# Árboles: Repaso

**1  $\Rightarrow$  2)**  $G$  es un árbol (grafo conexo sin circuitos simples).  $\Rightarrow G$  es un grafo sin circuitos simples y  $e$  una arista tq  $e \notin E$ .  $G+e = (V, E+\{e\})$  tiene exactamente un circuito simple, y ese circuito contiene a  $e$ . Es decir, si agrego una arista cualquiera se forma un ciclo.

## Demostración (1 $\Rightarrow$ 2):

Como  $G$  es conexo  $\Rightarrow$  Existe algún camino  $P_{uv}$  entre  $u$  y  $v$ .

Como  $G+e$  es conexo  $\Rightarrow$  Existe algún circuito  $C$ :  $P_{uv} + e$ .



# Árboles: Repaso

$1 \Rightarrow 2$ )  $G$  es un árbol (grafo conexo sin circuitos simples).  $\Rightarrow G$  es un grafo sin circuitos simples y  $e$  una arista tq  $e \notin E$ .  $G+e = (V, E+\{e\})$  tiene exactamente un circuito simple, y ese circuito contiene a  $e$ . Es decir, si agrego una arista cualquiera se forma un ciclo.

## Demostración ( $1 \Rightarrow 2$ ):

Como  $G$  es conexo  $\Rightarrow$  Existe algún camino  $P_{uv}$  entre  $u$  y  $v$ .

Como  $G+e$  es conexo  $\Rightarrow$  Existe algún circuito  $C: P_{uv} + e$ .

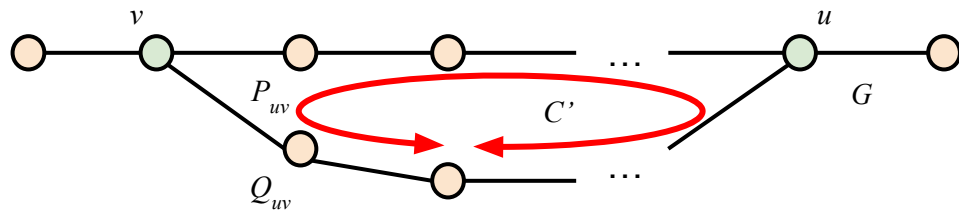
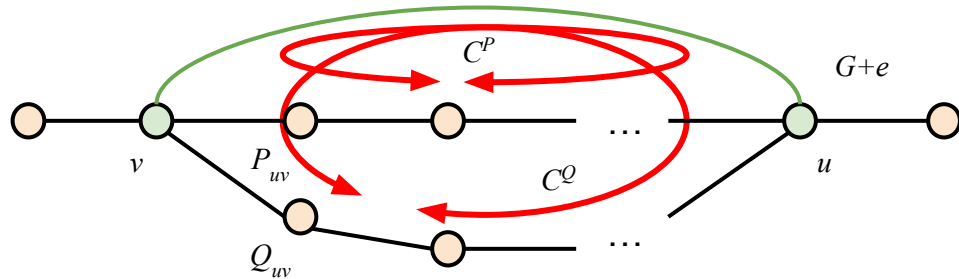
¿Por no existen más?

Supongo que existen dos  $C^P: P_{uv} + e$  y  $C^Q: Q_{uv} + e$  en  $G+e$

$\Rightarrow$  Existe algún circuito  $C': P_{uv} + Q_{vu}$  en  $G+e$  no usa  $e$

$\Rightarrow$  Existe algún circuito  $C': P_{uv} + Q_{vu}$  en  $G$

¡Absurdo!



# Árboles: Repaso

**2  $\Rightarrow$  3)**  $G$  es un grafo sin circuitos simples y  $e$  una arista tq  $e \notin E$ .  $G+e = (V, E+\{e\})$  tiene exactamente un circuito simple, y ese circuito contiene a  $e$ . Es decir, si agrego una arista cualquiera se forma un ciclo.  $\Rightarrow \exists$  exactamente un camino simple entre todo par de nodos.

## Demostración (2 $\Rightarrow$ 3):

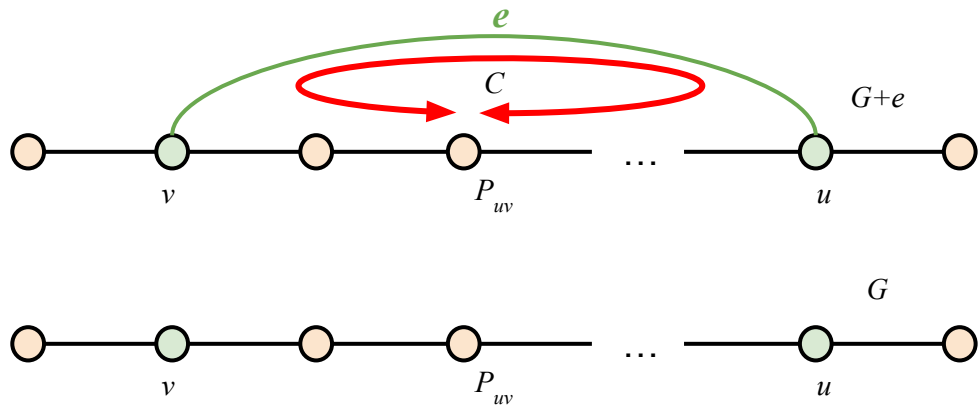
Existe algún circuito  $C: P_{uv} + e$  en  $G+e$

$\Rightarrow$  Existe algún camino  $P_{uv}$  entre  $u$  y  $v$  en  $G+e-e$

$\Rightarrow$  Existe  $P_{uv}$  entre todo par de vértices

¿Por no existen más?

Ídem 1  $\Rightarrow$  2



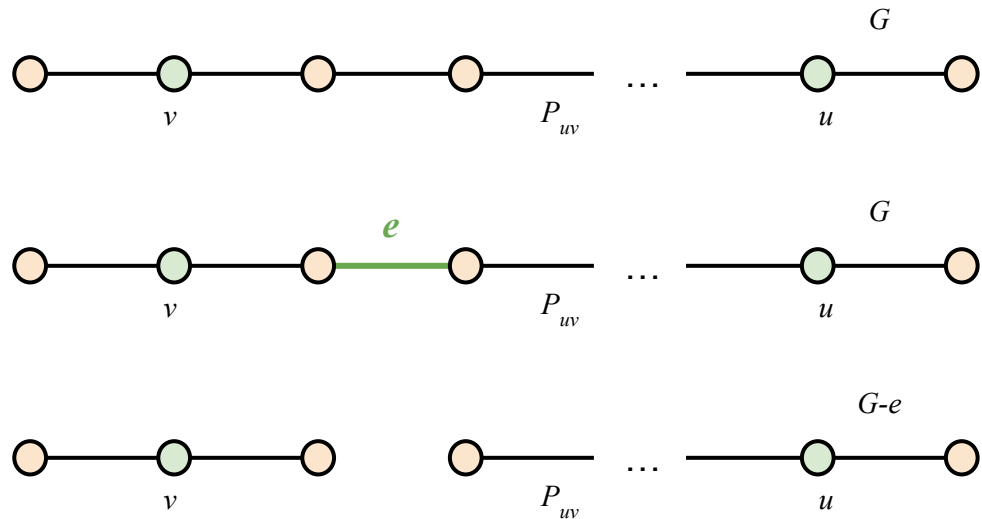
# Árboles: Repaso

$3 \Rightarrow 4)$   $\exists$  exactamente un camino simple entre todo par de nodos.  $\Rightarrow G$  es conexo, pero si se quita cualquier arista queda un grafo no conexo. *Es decir, si saco cualquier arista se desconecta, o toda arista es puente.*

## Demostración ( $3 \Rightarrow 4$ ):

Existe  $P_{uv}$  entre todo par de vértices  $\Rightarrow G$  es conexo

$P_{uv}$  es único  $\Rightarrow$  Si saco cualquier arista  $e \in P_{uv}$  se desconecta



# Árboles: Repaso

$4 \Rightarrow 1$ )  $G$  es conexo, pero si se quita cualquier arista queda un grafo no conexo. Es decir, si saco cualquier arista se desconecta, o toda arista es puente.  $\Rightarrow G$  es un árbol (grafo conexo sin circuitos simples).

## Demostración ( $4 \Rightarrow 1$ ):

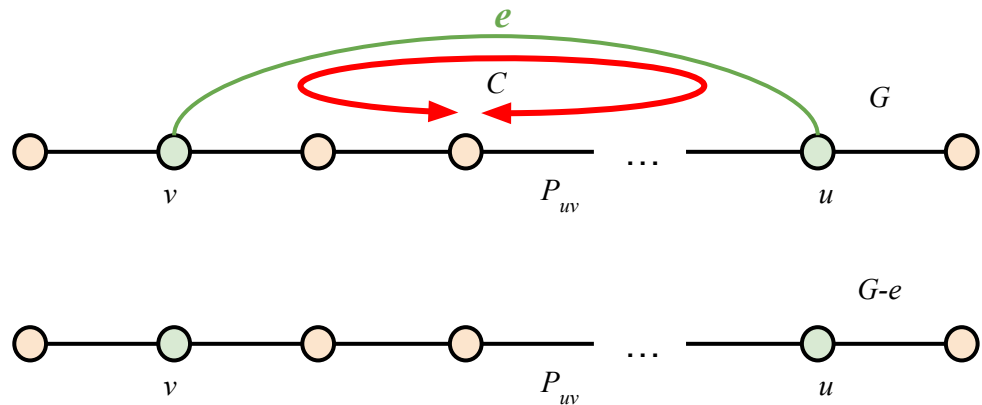
$G$  es conexo

Si existe  $e$  tq  $C : P_{uv} + e$  es circuito simple en  $G$

$\Rightarrow$  Si saco  $e$  no se desconecta.

**¡Absurdo!**

$\Rightarrow G$  es conexo y sin circuitos simples (un árbol)



# Árboles: Definiciones

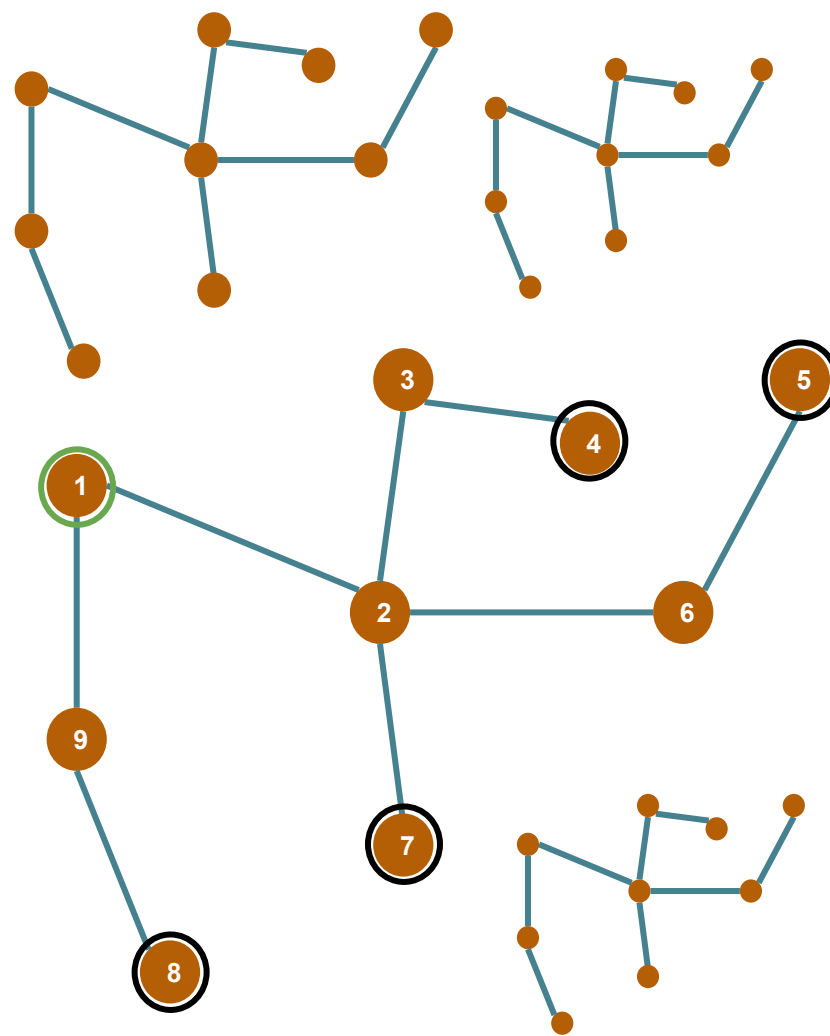
**Árbol :**  $T$

**Hoja:**  $u$  tq  $d(u) = 1$

**Raíz:** Algún vértice elegido

**Bosque:** Conjunto de árboles

**Árbol trivial:**  $T$  con  $n=1$  y  $m=0$



# Árboles: Repaso

**Lema 3:** Todo árbol no trivial tiene al menos dos hojas

**Demostración:**

$P : v_l \dots v_k$  es un camino simple maximal en  $T$  (no lo puedo extender más).

q.v.q.  $v_l$  y  $v_k$  son hojas, es decir que  $d(v_l) = 1$  y  $d(v_k) = 1$

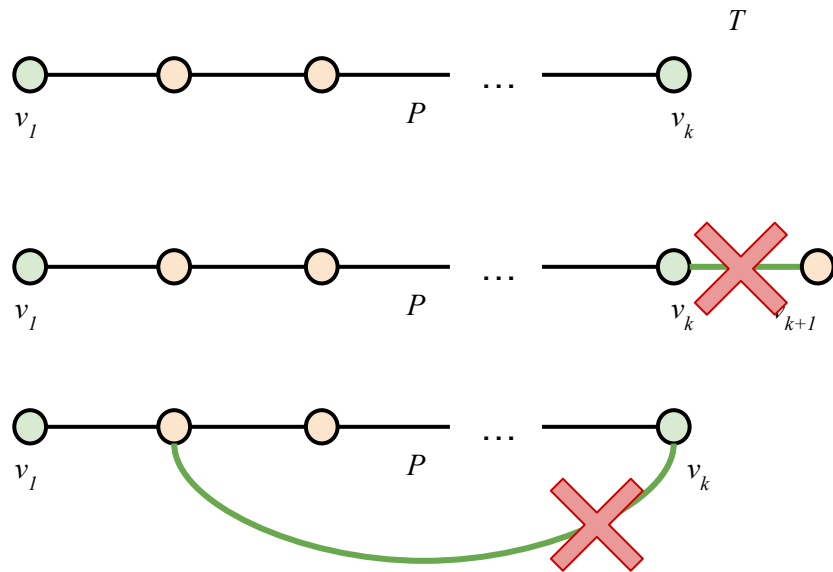
$d(v_k) > 0$  porque conecta con  $v_{k-1}$

$d(v_k) > 1$  ??

No puedo agregar un vértice porque era maximal

No puedo ir a uno existente porque formo un circuito.

$\Rightarrow d(v_k) = 1$  (idem  $v_l$ )





# Árboles: Repaso

**Lema 4:** Sea  $G = (V, E)$  un árbol  $\Rightarrow m = n - 1$

**Demostración:** Inducción en  $n$ .

Caso base:  $n=1$  y  $m=0$

Hipótesis inductiva:  $T'$  con  $k'$  vértices ( $k' < k$ ) tiene  $k' - 1$  aristas

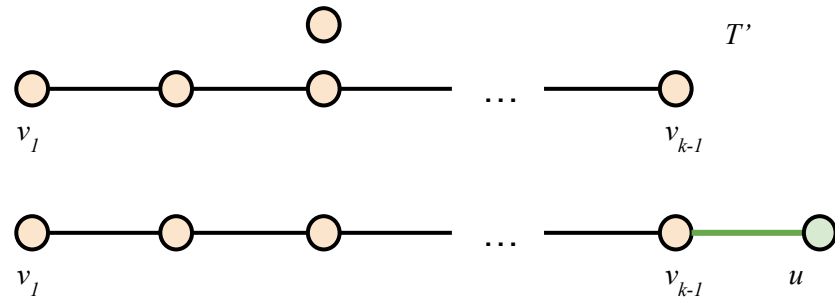
Paso inductivo: Sea  $u$  una hoja (sabemos que tiene por Lema 2).

$$T' = T - u = (V \setminus \{u\}, E \setminus \{(u, v) \in E, \forall v \in V\})$$

$T'$  es conexo y sin circuitos con  $k' = k - 1$  vértices  $< k$

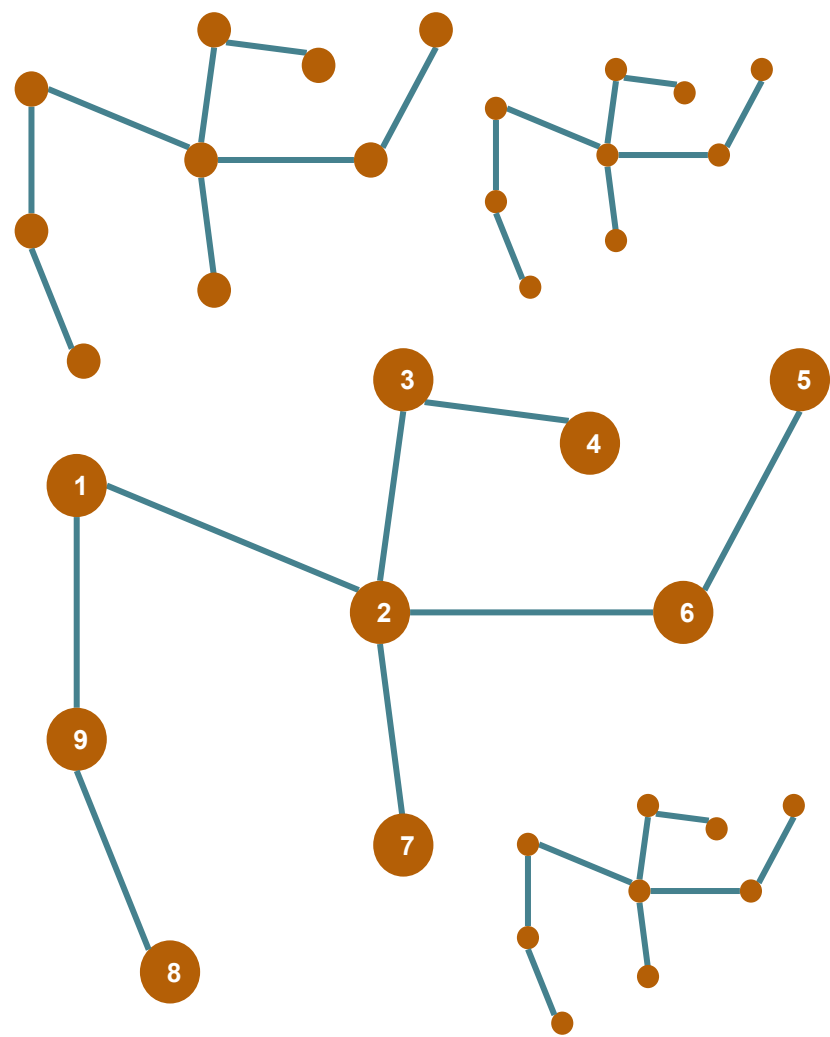
$\Rightarrow$  (Hip. ind.) tiene  $k - 2$  aristas

Como  $u$  era una hoja  $\Rightarrow d(u)=1 \Rightarrow T$  tiene  $k - 2 + 1 = k - 1$  aristas



# Árboles: Definiciones

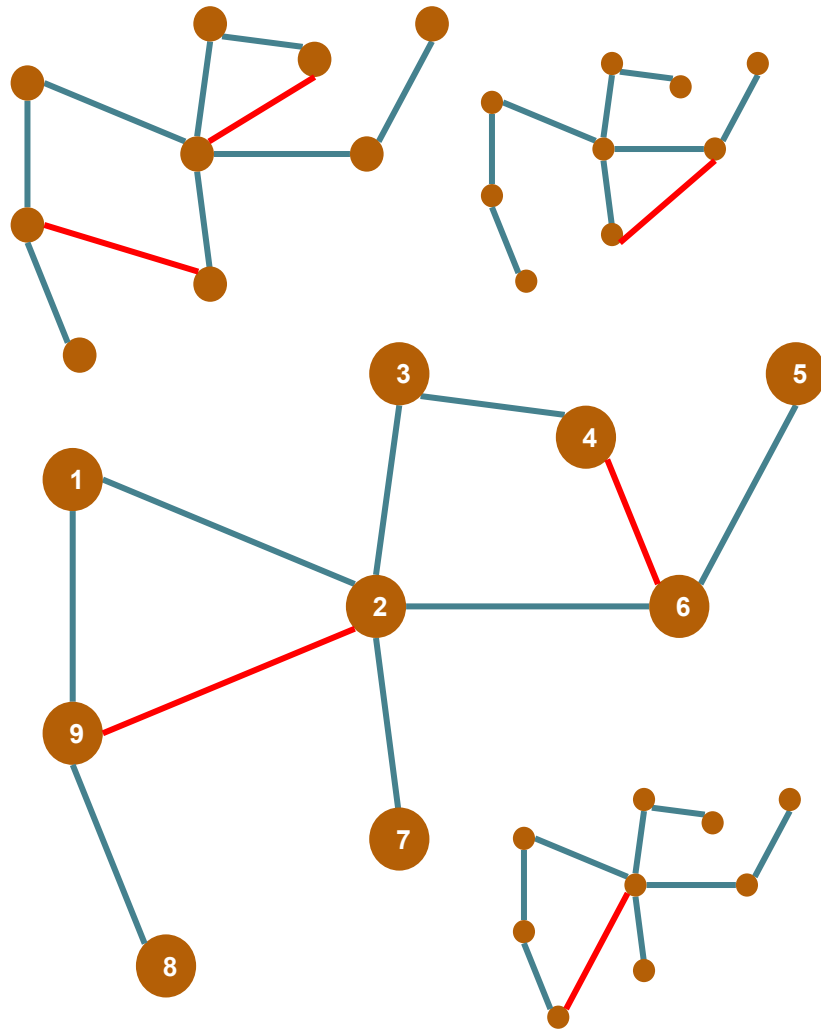
**Corolario 1:** Sea  $G$  un bosque con  $c$  c.c.  $\Rightarrow m = n - c$



# Árboles: Definiciones

**Corolario 1:** Sea  $G$  un bosque con  $c$  c.c.  $\Rightarrow m = n - c$

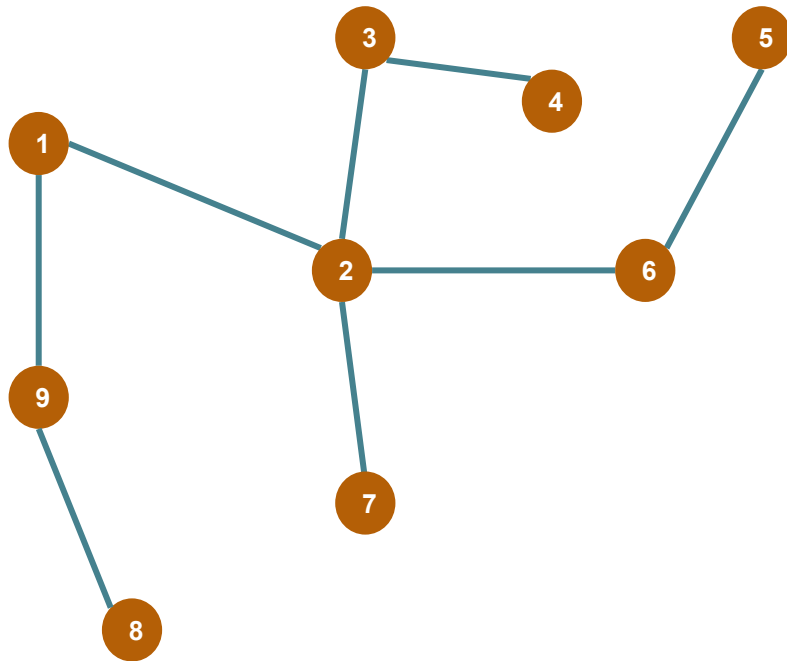
**Corolario 2:** Sea  $G$  un grafo con  $c$  c.c.  $\Rightarrow m \geq n - c$



# Árboles: Repaso

## Teorema 2: Equivalencias

1.  $G$  es un árbol (*grafo conexo sin circuitos simples*).
2.  $G$  es un grafo sin circuitos simples y  $m = n - 1$
3.  $G$  es un grafo conexo y  $m = n - 1$



**1  $\Rightarrow$  2)**  $G$  es un árbol (*grafo conexo sin circuitos simples*).  $\Rightarrow G$  es un grafo sin circuitos simples y  $m = n - 1$

# Árboles: Repaso

**Demostración (1  $\Rightarrow$  2):**

(Por Lema 4)

$2 \Rightarrow 3)$   $G$  es un grafo sin circuitos simples y  $m = n - 1 \Rightarrow G$  es un grafo conexo y  $m = n - 1$

# Árboles: Repaso

**Demostración ( $2 \Rightarrow 3$ ):**

Si tiene  $c$  c.c.  $\Rightarrow m = n - c = n - 1 \Rightarrow c = 1 \Rightarrow G$  conexo

$3 \Rightarrow 1$ )  $G$  es un grafo conexo y  $m = n - 1 \Rightarrow G$  es un árbol

# Árboles: Repaso

**Demostración ( $3 \Rightarrow 1$ ):**

Si  $G$  tiene un circuito simple,  $G$  conexo

$\Rightarrow$  (por Lema 2)  $G-e$  conexo y  $m_{G-e} = n - 2$  (porque  $m_G = n - 1$ )

**¡Absurdo!**

$G$  no tiene un circuito simple,  $G$  conexo (es un árbol)


**Lema 2:** Sea  $G = (V, E)$  un grafo conexo y  $e = (v, u) \in E$ .

$G - e = (V, E \setminus \{e\})$  es conexo  $\Leftrightarrow e \in C$  : circuito simple de  $G$ .

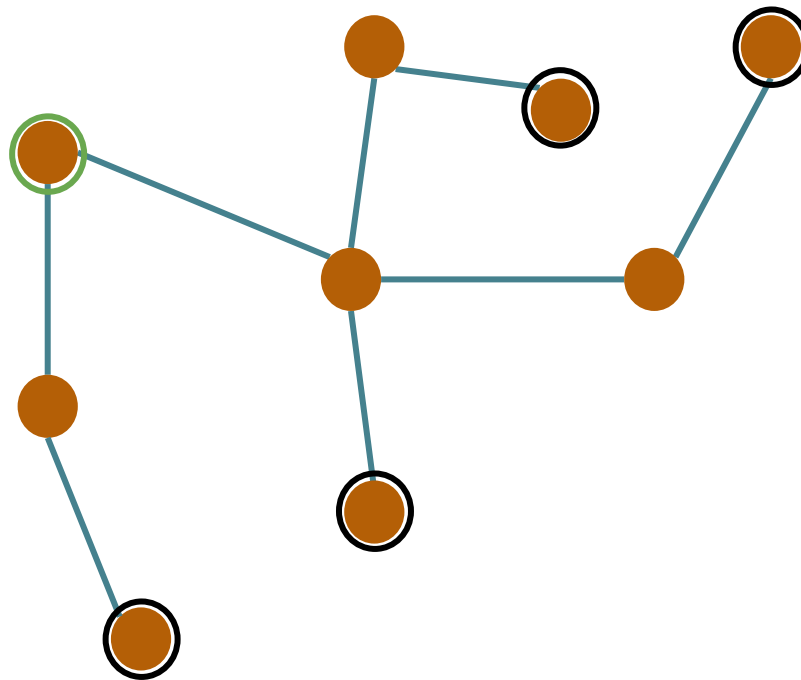
(  $e = (v, u) \in E$  es puente  $\Leftrightarrow e$  no pertenece a un circuito simple de  $G$  ).

# Árboles con raíz (enraizados...)

 **Raíz:** Algún vértice elegido

 **Hoja:**  $u$  tq  $d(u) = 1$


**Árbol enraizado:** Árbol con raíz





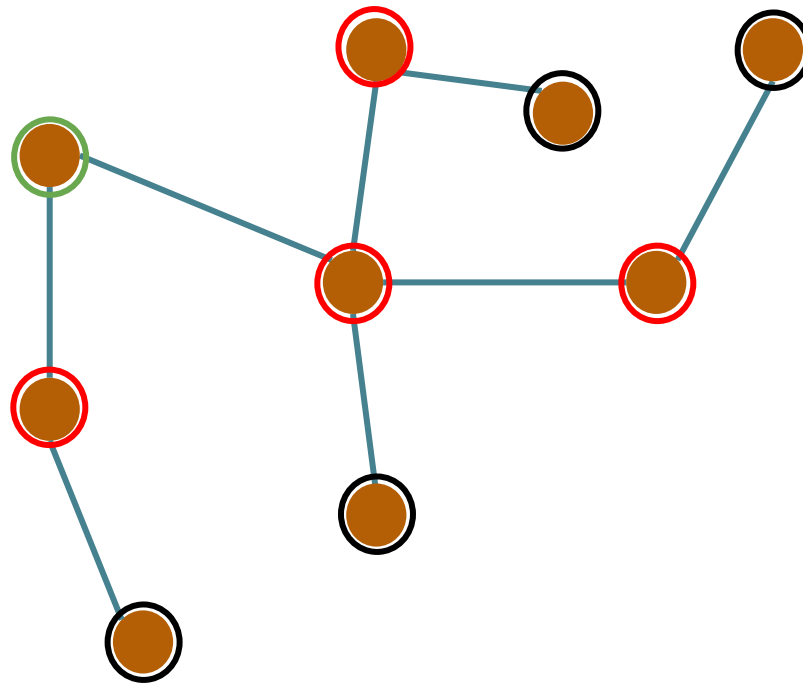
# Árboles con raíz (enraizados...)

 **Raíz:** Algún vértice elegido

 **Hoja:**  $u$  tq  $d(u) = 1$


**Árbol enraizado:** Árbol con raíz

 **Vértices internos:** Ni hojas ni raíces



# Árboles con raíz (enraizados...)

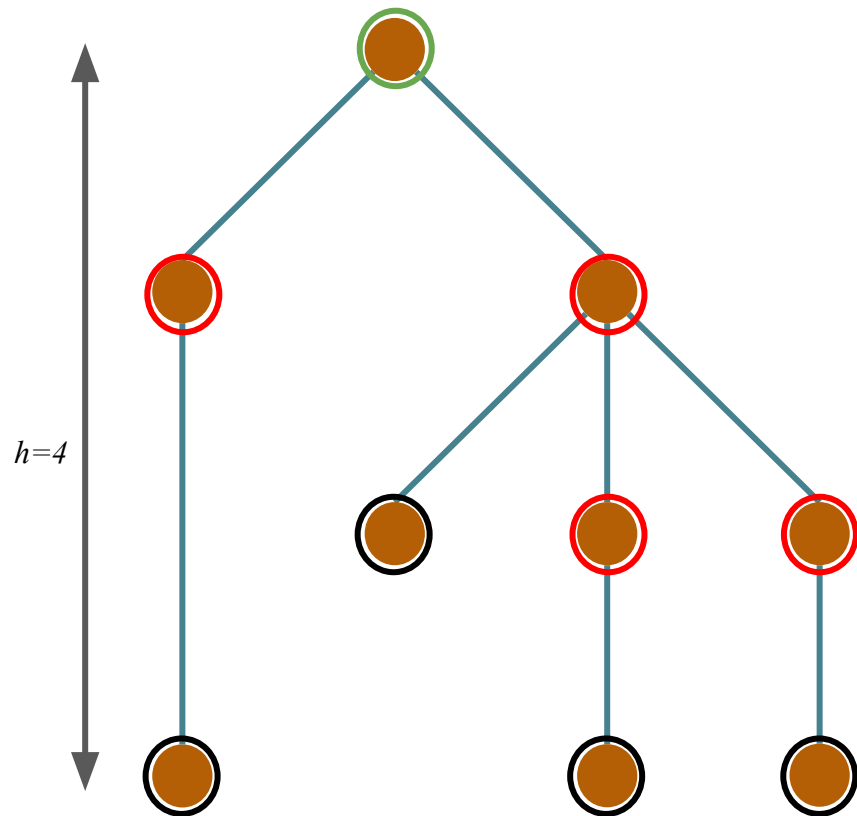
 **Raíz:** Algún vértice elegido

 **Hoja:**  $u$  tq  $d(u) = l$

**Árbol enraizado:** Árbol con raíz

 **Vértices internos:** Ni hojas ni raíces

**Altura (h):** De la raíz a la hoja más lejana.



# Árboles con raíz (enraizados...)

 **Raíz:** Algún vértice elegido

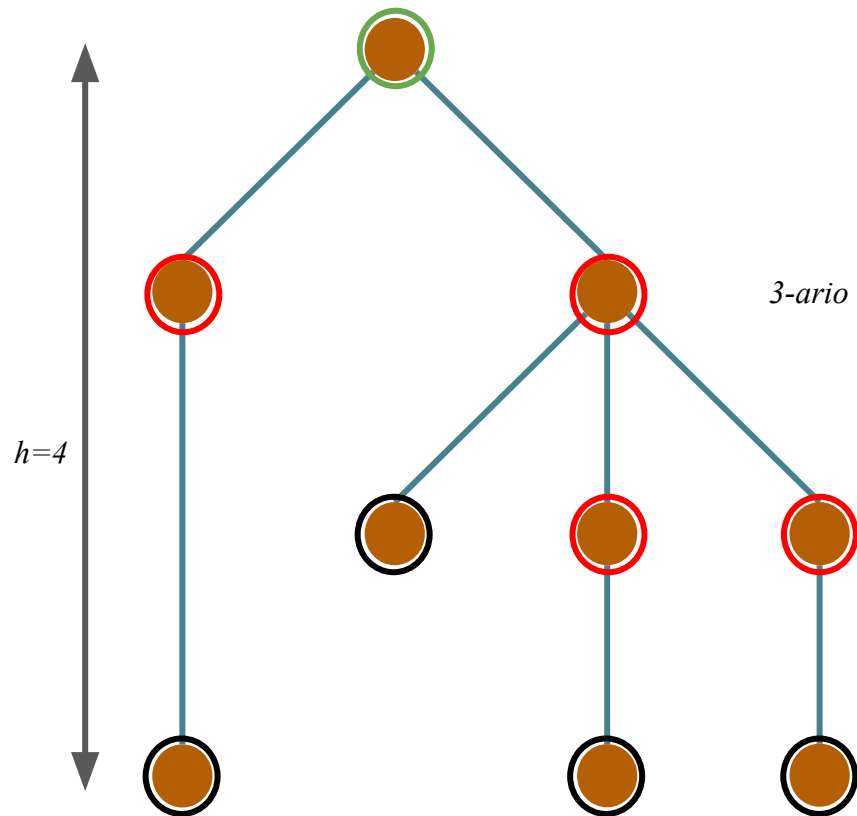
 **Hoja:**  $u$  tq  $d(u) = l$

**Árbol enraizado:** Árbol con raíz

 **Vértices internos:** Ni hojas ni raíces


**Altura (h):** De la raíz a la hoja más lejana.

**Árbol m-ario:** Donde  $m$  es el número máximo de hijos un nodo (si todos los vértices  $v$  tienen  $d(v) \leq m + 1$  y la raíz  $r$  tiene  $d(r) \leq m$ ).



# Árboles con raíz (enraizados...)

 **Raíz:** Algún vértice elegido

 **Hoja:**  $u$  tq  $d(u) = 1$

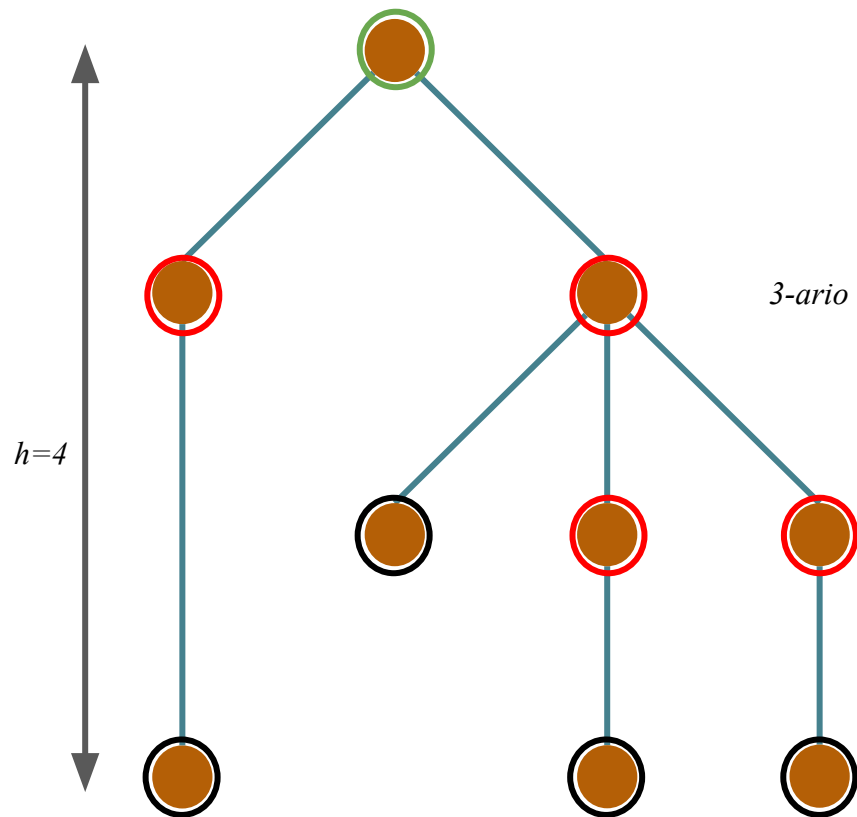
**Árbol enraizado:** Árbol con raíz

 **Vértices internos:** Ni hojas ni raíces

**Altura (h):** De la raíz a la hoja más lejana.


**Árbol m-ario:** Donde  $m$  es el número máximo de hijos un nodo (si todos los vértices  $v$  tienen  $d(v) \leq m + 1$  y la raíz  $r$  tiene  $d(r) \leq m$ ).

**Nivel:** “Altura” de un vértice o distancia a la raíz.



# Árboles con raíz (enraizados...)

 **Raíz:** Algún vértice elegido

 **Hoja:**  $u$  tq  $d(u) = l$

**Árbol enraizado:** Árbol con raíz

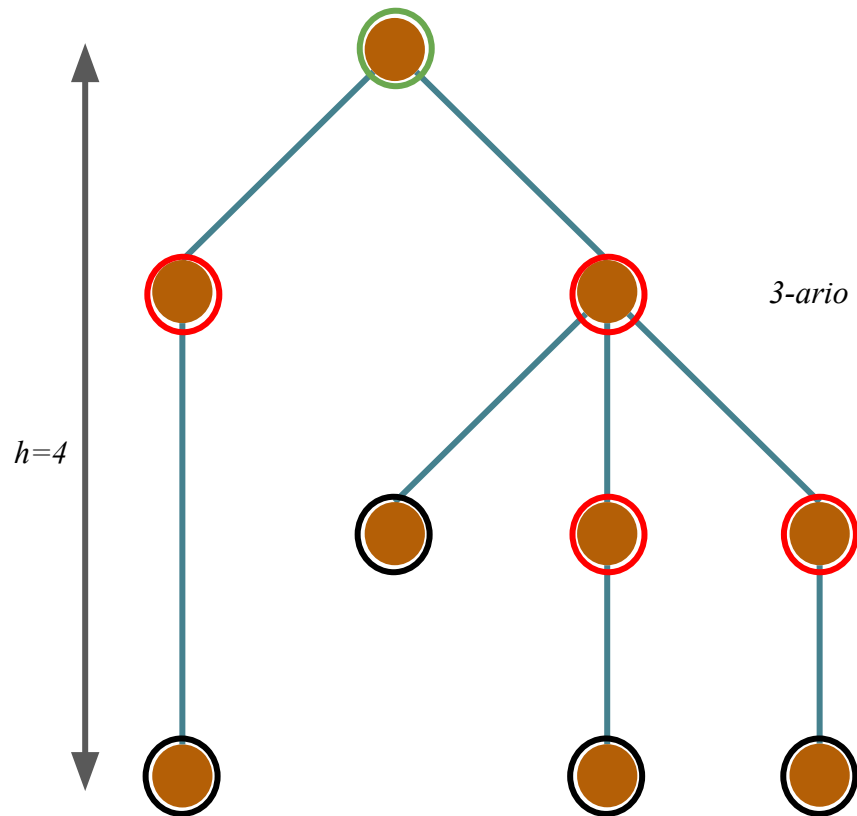
 **Vértices internos:** Ni hojas ni raíces

**Altura (h):** De la raíz a la hoja más lejana.

**Árbol m-ario:** Donde  $m$  es el número máximo de hijos un nodo (si todos los vértices  $v$  tienen  $d(v) \leq m + 1$  y la raíz  $r$  tiene  $d(r) \leq m$ ).

**Nivel:** “Altura” de un vértice o distancia a la raíz.

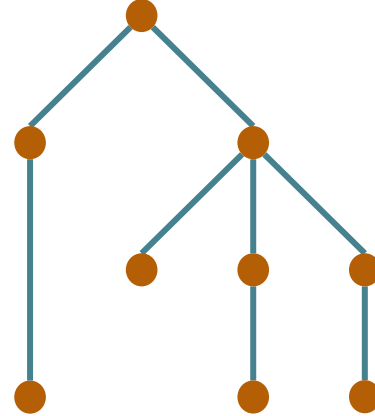
**Árbol balanceado:** Todas sus hojas están a nivel  $h$  (o  $h-1$ ).



# Árboles con raíz (enraizados...)

## Teorema 3:

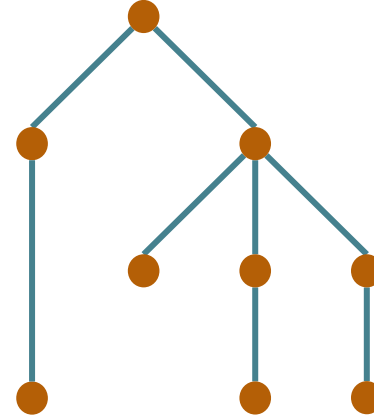
1.  $T$  es  $m$ -ario de altura  $h \Rightarrow$  tiene a lo sumo  $l=m^h$  hojas
2.  $T$  es  $m$ -ario con  $l$  hojas  $\Rightarrow$  tiene  $h \geq \lceil \log_m(l) \rceil$  hojas



$T$  es  $m$ -ario de altura  $h \Rightarrow$  tiene a lo sumo  $l=m^h$  hojas

# Árboles con raíz (enraizados...)

**Demostración (1):**



$$h=1 \Rightarrow l \leq m = m^1$$

$$h=2 \Rightarrow l \leq m * m^1 = m^2$$

$$h=3 \Rightarrow l \leq m * m^2 = m^3$$

$$h=4 \Rightarrow l \leq m * m^3 = m^4$$

$$\Rightarrow l \leq m * m^{h-1} = m^h$$

$T$  es  $m$ -ario con  $l$  hojas  $\Rightarrow$  tiene  $h \geq \lceil \log_m(l) \rceil$  hojas

# Árboles con raíz (enraizados...)

**Demostración (2):**

$$l \leq m^h$$

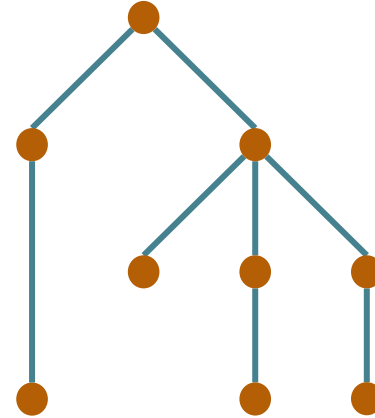
$$\log(l) \leq \log(m^h)$$

$$\log(l) \leq h * \log(m)$$

$$\log(l) / \log(m) \leq h$$

$$\log_m(l) \leq h$$

$$\lceil \log_m(l) \rceil \leq h \text{ (por ser entero)}$$



$$h=1 \Rightarrow l \leq m = m^1$$

$$h=2 \Rightarrow l \leq m * m^1 = m^2$$

$$h=3 \Rightarrow l \leq m * m^2 = m^3$$

$$h=4 \Rightarrow l \leq m * m^3 = m^4$$

$$\Rightarrow l \leq m * m^{h-1} = m^h$$

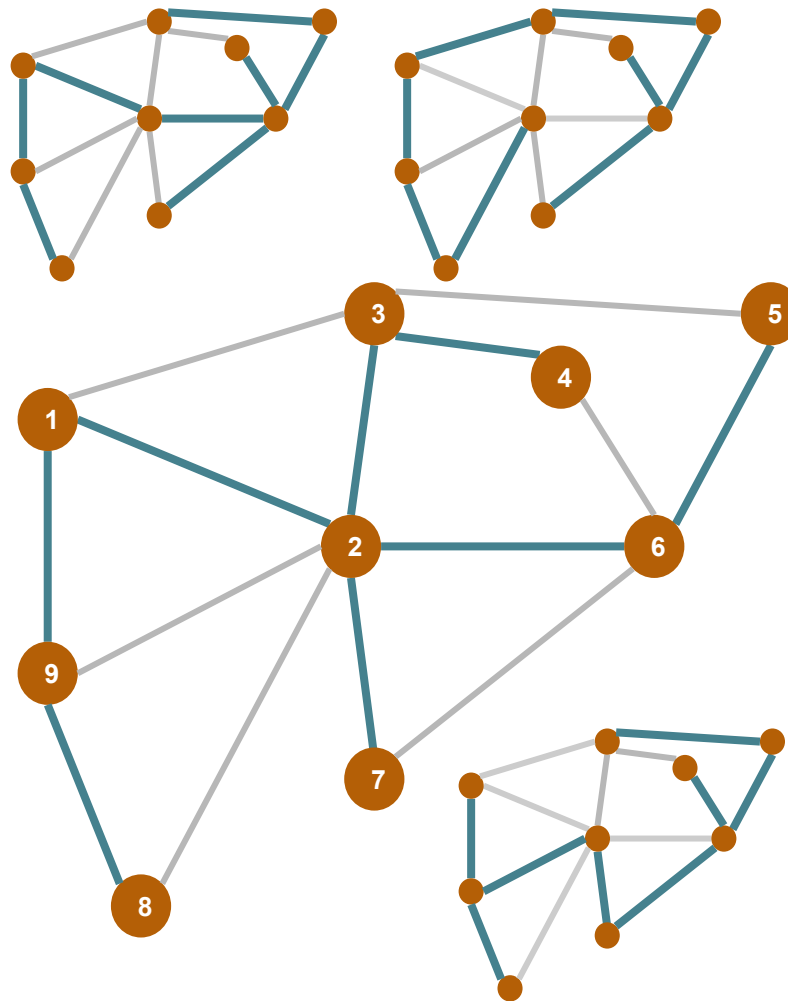


AED3 > Clase 6 > AGM

# Árbol Generador (AG)

## Definición:

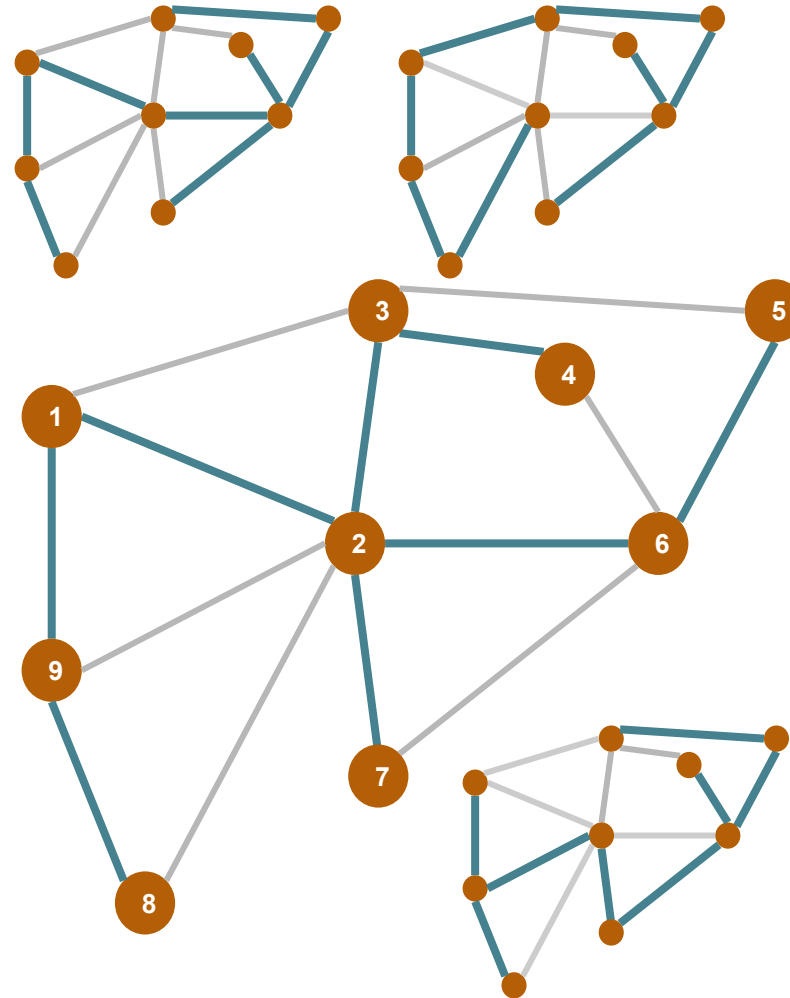
Un árbol generador (AG) de un grafo  $G$  es un subgrafo que tiene el mismo conjunto de vértices y es un árbol.



# Árbol Generador (AG)

## Teorema 4:

1. Todo  $G$  conexo tiene al menos un  $AG$ .
2. Si  $G$  conexo.  $G$  tiene un sólo  $AG$  sii  $G$  es un árbol.
3.  $T=(V, E_T)$  es  $AG$  de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol) tq  $T' = T + e - f = (V, E \cup \{e\} \setminus \{f\})$  con  $f$  una arista del único circuito que se forma al agregar  $e$  (de  $T+e$ )  $\Rightarrow T'$  es otro  $AG$  de  $G$ .

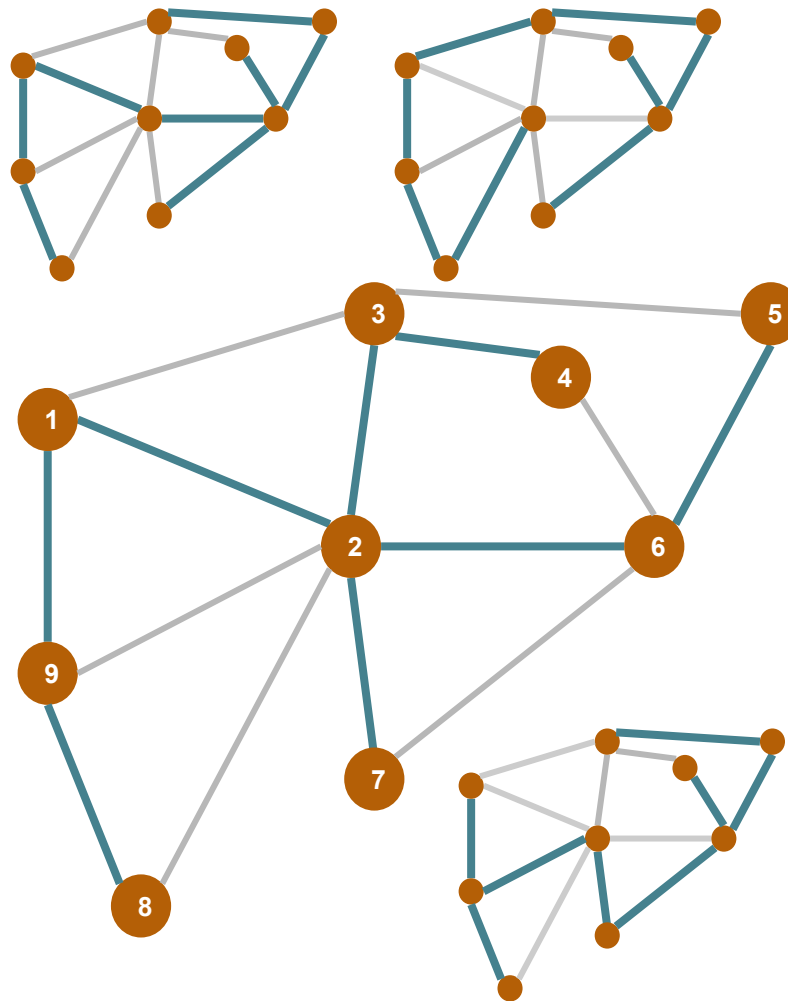


# Árbol Generador (AG)

## Demostración (1):

Por construcción,

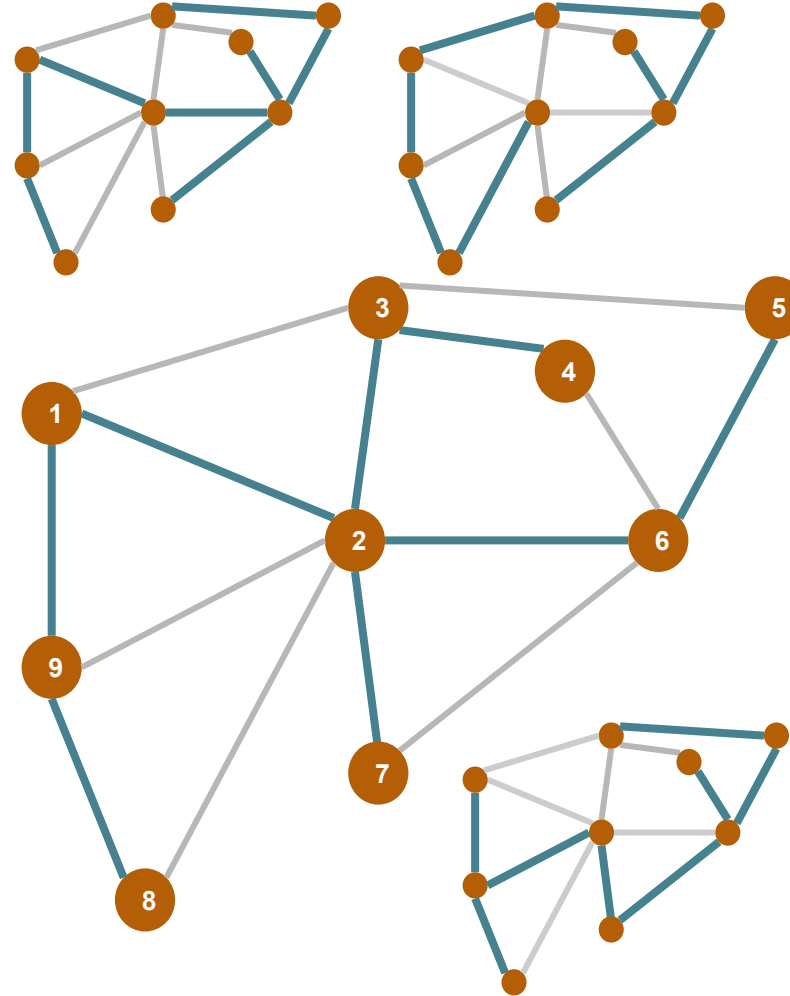
$G$  conexo, ejecuto *BFS* o *DFS* y obtengo un *AG*.



# Árbol Generador (AG)

## Demostración (2):

$(\Leftarrow)$   $G$  es un árbol. (Hip. Abs.) Si tuviese dos AG significa que hay dos formas de conectar  $u$  y  $v$  en  $G \Rightarrow$  hay un circuito en  $G \Rightarrow G$  no es un árbol. **¡Absurdo!**

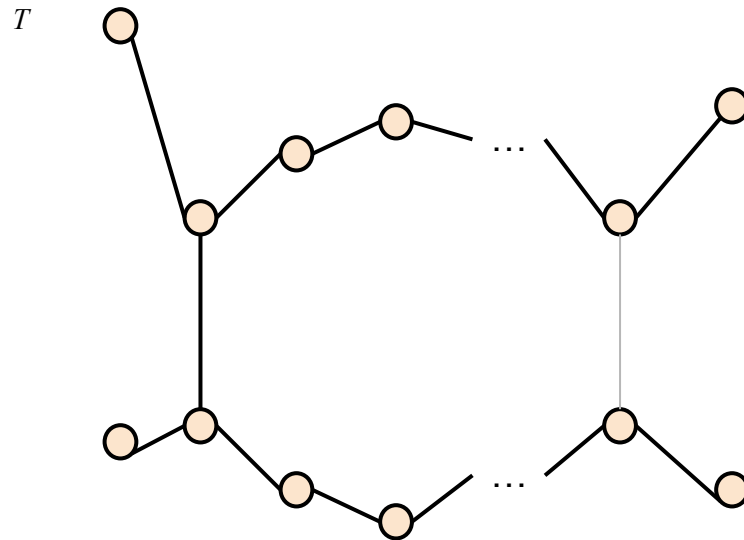


# Árbol Generador (AG)

## Demostración (3):

$T=(V, E_T)$  es AG de  $G=(V, E)$ .

$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol) tq  $T' = T+e-f = (V, E \cup \{e\} \setminus \{f\})$  con  $f$  una arista del único circuito que se forma al agregar  $e$  (de  $T+e$ )  $\Rightarrow T'$  es otro AG de  $G$ .

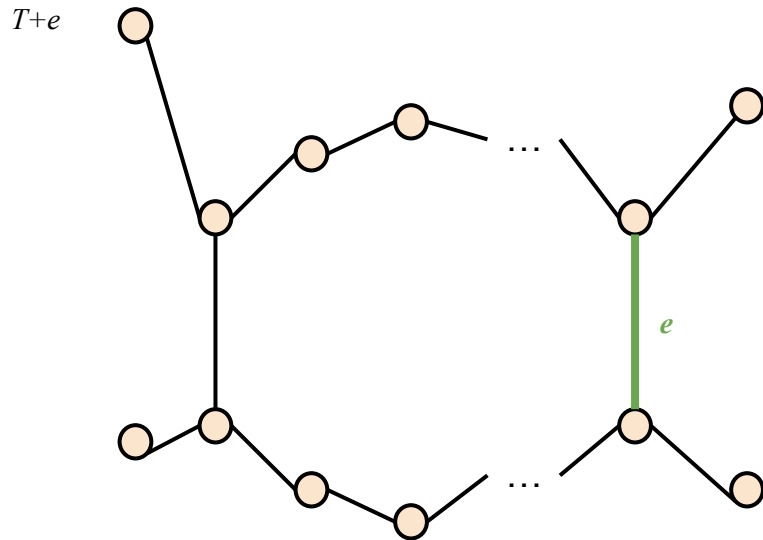


# Árbol Generador (AG)

## Demostración (3):

$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol)

$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol) tq  $T' = T+e-f = (V, E \cup \{e\} \setminus \{f\})$  con  $f$  una arista del único circuito que se forma al agregar  $e$  (de  $T+e$ )  $\Rightarrow T'$  es otro AG de  $G$ .



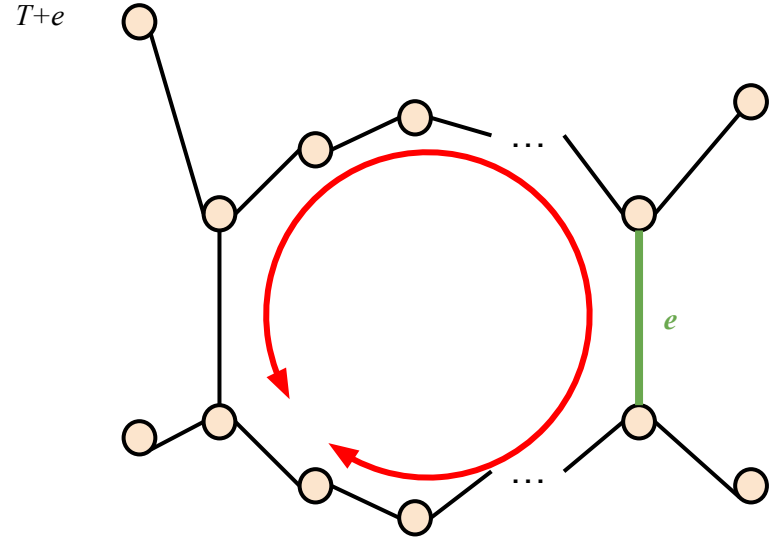
# Árboles Generador (AG)

## Demostración (3):

$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol).

Por Teorema 1, se forma un único circuito.

$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol) tq  $T' = T+e-f = (V, E \cup \{e\} \setminus \{f\})$  con  $f$  una arista del único circuito que se forma al agregar  $e$  (de  $T+e$ )  $\Rightarrow T'$  es otro AG de  $G$ .





# Árbol Generador (AG)

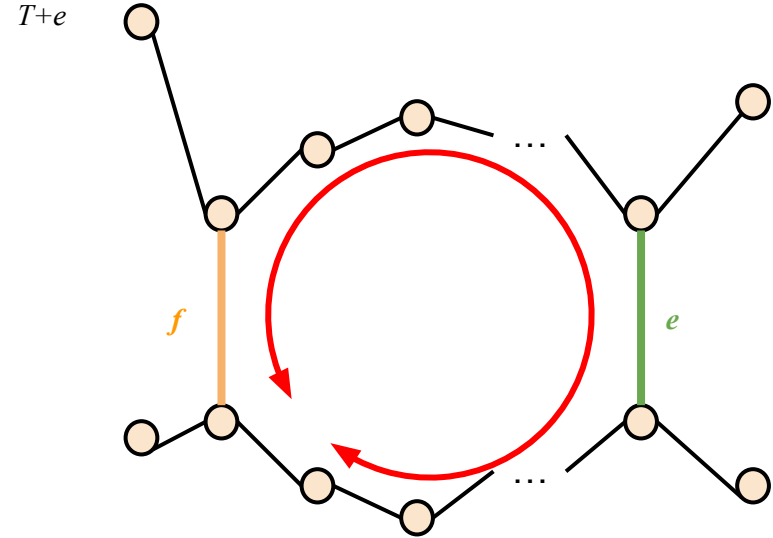
## Demostración (3):

$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol).

Por Teorema 1, se forma un único circuito.

Sea  $f$  una arista del único circuito que se forma al agregar  $e$  (de  $T+e$  )

$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol) tq  $T' = T+e-f = (V, E \cup \{e\} \setminus \{f\})$  con  $f$  una arista del único circuito que se forma al agregar  $e$  (de  $T+e$  )  $\Rightarrow T'$  es otro AG de  $G$ .



# Árbol Generador (AG)

## Demostración (3):

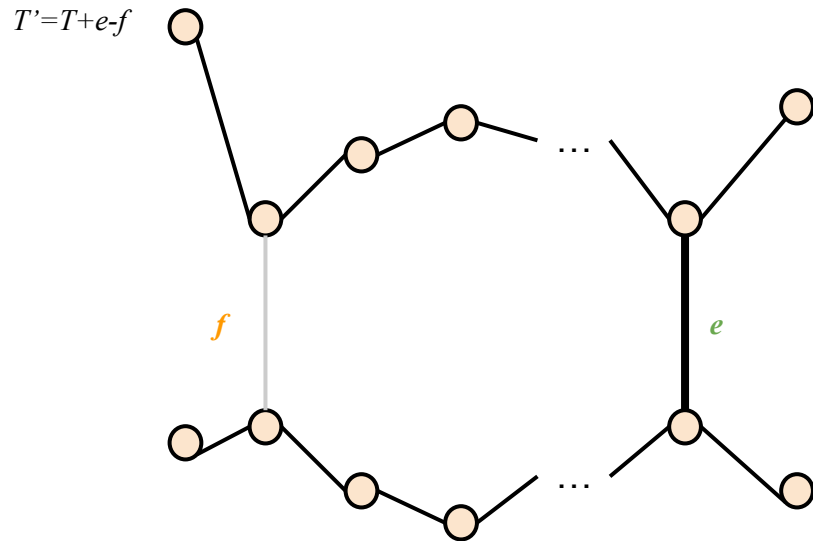
$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol).

Por Teorema 1, se forma un único circuito.

Sea  $f$  una arista del único circuito que se forma al agregar  $e$  (de  $T+e$  )

$$T' = T+e-f = (V, E \cup \{e\} \setminus \{f\})$$

$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol) tq  $T' = T+e-f = (V, E \cup \{e\} \setminus \{f\})$  con  $f$  una arista del único circuito que se forma al agregar  $e$  (de  $T+e$  )  $\Rightarrow T'$  es otro AG de  $G$ .



# Árbol Generador (AG)

## Demostración (3):

$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol).

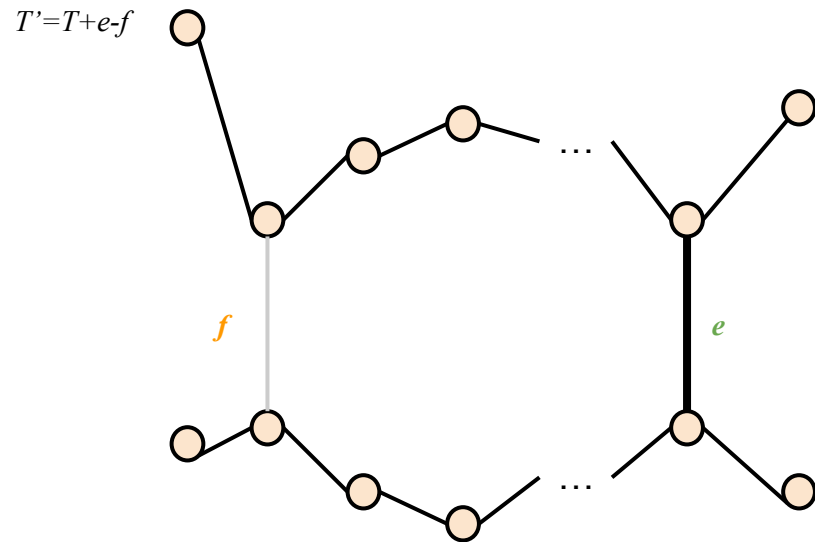
Por Teorema 1, se forma un único circuito.

Sea  $f$  una arista del único circuito que se forma al agregar  $e$  (de  $T+e$  )

$$T' = T+e-f = (V, E \cup \{e\} \setminus \{f\})$$

$T'$  es conexo por Lema 2

$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol) tq  $T' = T+e-f = (V, E \cup \{e\} \setminus \{f\})$  con  $f$  una arista del único circuito que se forma al agregar  $e$  (de  $T+e$  )  $\Rightarrow T'$  es otro AG de  $G$ .



**Lema 2:** Sea  $G = (V, E)$  un grafo conexo y  $e=(v,u) \in E$ .

$G - e = (V, E \setminus \{e\})$  es conexo  $\Leftrightarrow e \in C$  : circuito simple de  $G$ .

(  $e=(v,u) \in E$  es puente  $\Leftrightarrow e$  no pertenece a un circuito simple de  $G$  ).

# Árbol Generador (AG)

## Demostración (3):

$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol).

Por Teorema 1, se forma un único circuito.

Sea  $f$  una arista del único circuito que se forma al agregar  $e$  (de  $T+e$  )

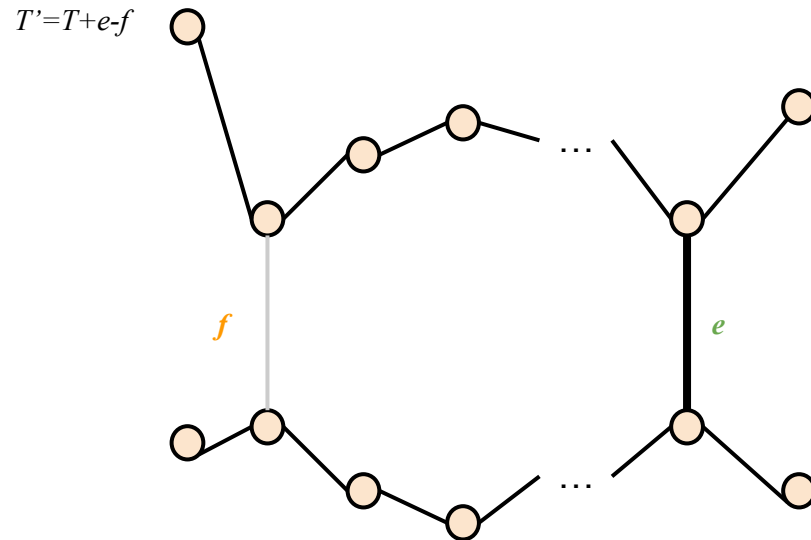
$$T' = T+e-f = (V, E \cup \{e\} \setminus \{f\})$$

$T'$  es conexo por Lema 2

$T'$  tiene los mismos vértices que  $G \Rightarrow$  es subgrafo generador

$T'$  tiene  $n-1$  aristas  $\Rightarrow$  es árbol generador (AG)

$T=(V, E_T)$  es AG de  $G=(V, E)$ . Sea  $e=E \setminus E_T$  (no está en el árbol) tq  $T' = T+e-f = (V, E \cup \{e\} \setminus \{f\})$  con  $f$  una arista del único circuito que se forma al agregar  $e$  (de  $T+e$  )  $\Rightarrow T'$  es otro AG de  $G$ .



## Teorema 2: Equivalencias

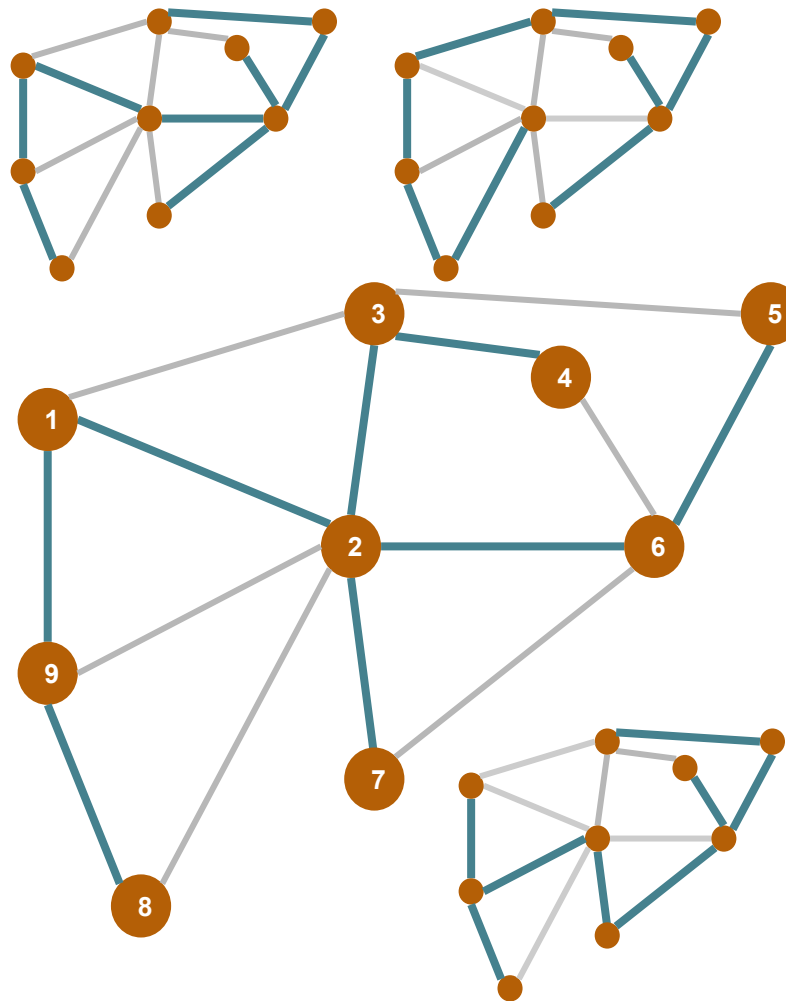
1.  $G$  es un árbol (grafo conexo sin circuitos simples).
2.  $G$  es un grafo sin circuitos simples y  $m = n - 1$
3.  $G$  es un grafo conexo y  $m = n - 1$

# Árbol Generador Mínimo (AGM)

Árbol Generador Mínimo (AGM) = Minimum Spanning Tree (MST).

Dado un grafo  $G=(V, E, w)$  con  $w: E \rightarrow R$

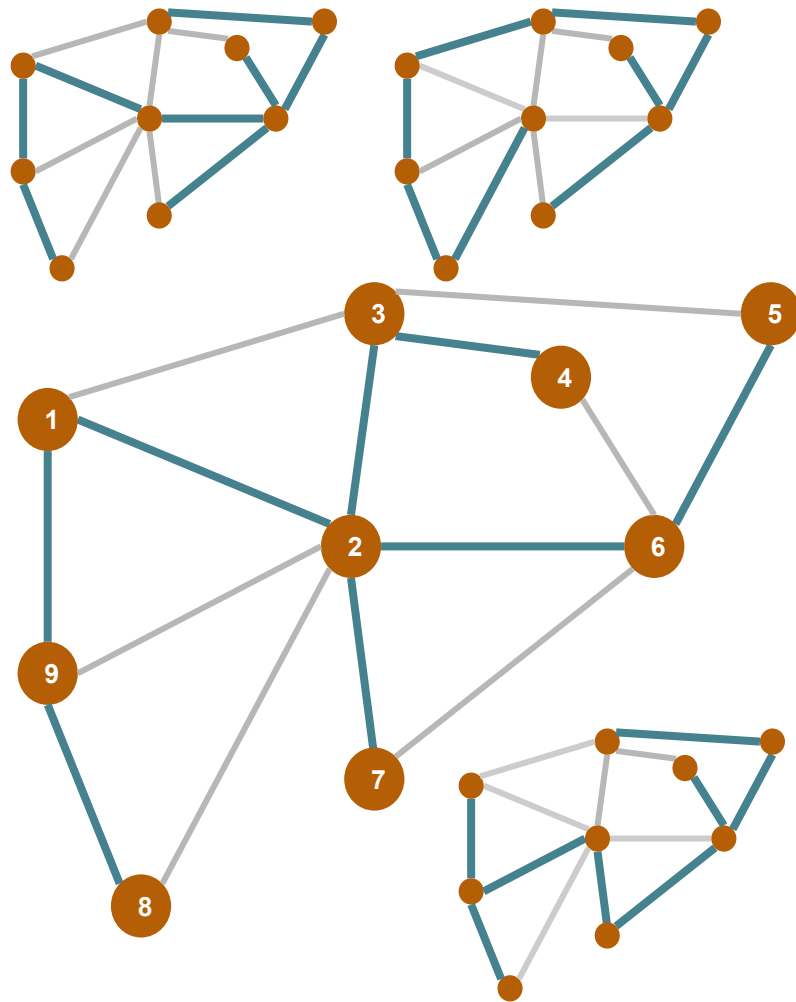
- Costo:  $w(T) = \sum_T w(e) \dots$  Como un abuso de notación se usa  $w$  tanto para el costo de una arista como de todo el árbol.
- AGM es el AG para el cual  $\sum_T w$  es mínima.
- Para los grafos no pesados todo AG es AGM porque  $w=1 \Rightarrow \sum_T w = m = n-1$
- También puede haber varios AGM.



# Árbol Generador Mínimo (AGM)

Prim (1957)

Kruskal (1956)



# Algoritmos golosos (CLRS cap. 16)

## 1. Subestructura óptima.

La solución óptima del problema contiene las soluciones óptimas a los subproblemas.

## 2. Elección golosa.

En cada paso se busca el óptimo local...

podría no ser el óptimo global

(en general no lo es  $\Rightarrow$  heurísticas)

# Algoritmos golosos (CLRS cap. 16)

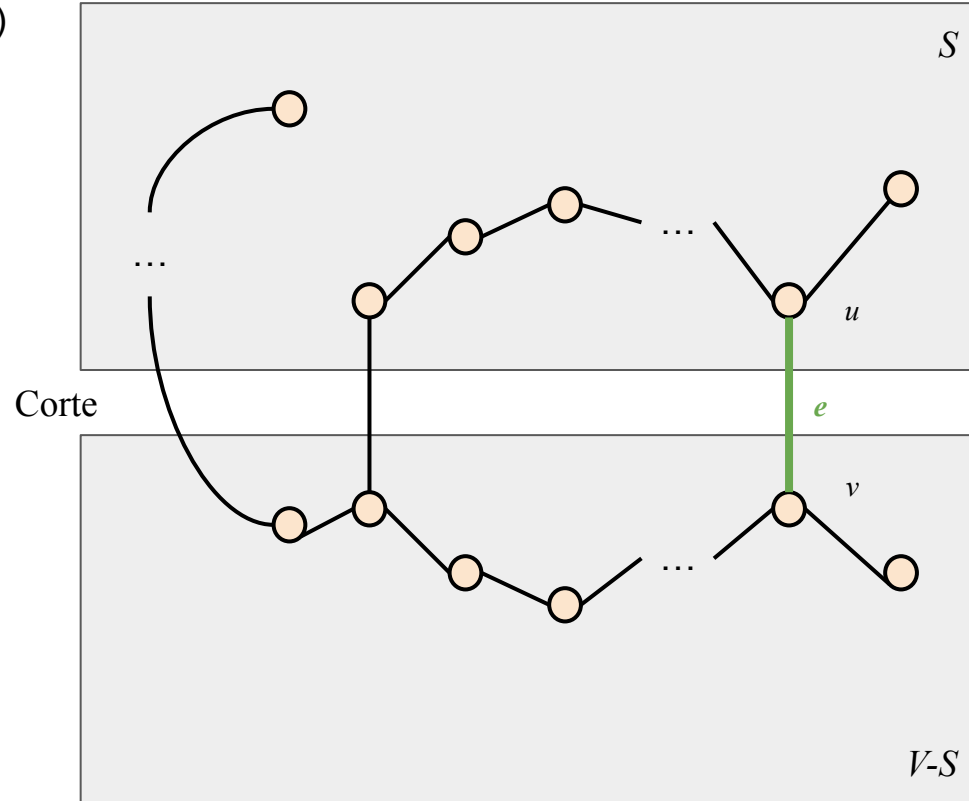
## 1. Subestructura óptima.

→ AGM

## 2. Elección golosa.

→ Elijo  $e=(u,v)$  tq  $u \in S$ ,  $v \in V-S$ , y  $w(e)$  es el mínimo de las aristas que cruzan el corte.

$\Rightarrow e \in (\text{algún}) \text{ AGM}$





# Algoritmos golosos (CLRS cap. 16)

## Demostración (Elección golosa).

Sea  $T$  AGM de  $G$ .

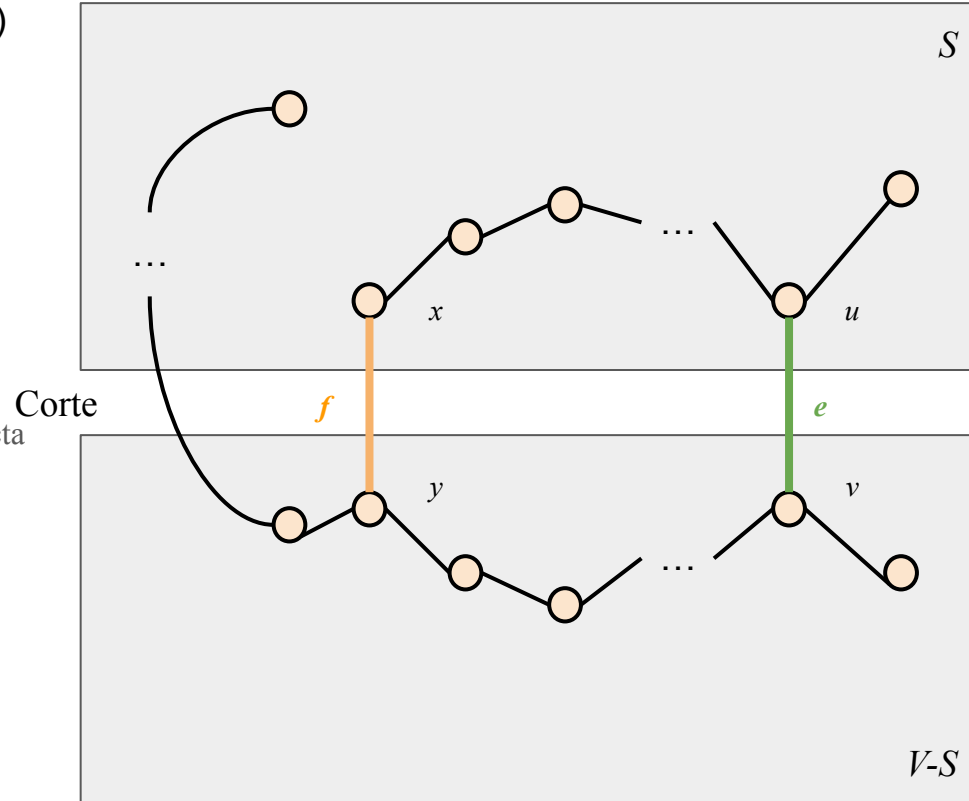
1. Si  $e \in T \Rightarrow$  LISTO

2. Si  $e \notin T \Rightarrow$

elijo  $f=(x,y)$  tq  $x \in S, y \in V-S$ , y  $f \in P_{uv}$  (está en la rama que conecta  $u$  con  $v$ )

$\Rightarrow T' = T - f + e$  también es AG de  $G$  (Teorema 4)

¿ $T'$  es AGM?



# Algoritmos golosos (CLRS cap. 16)

## Demostración (Correctitud).

Sea  $T$  AGM de  $G$ .

1. Si  $e \in T \Rightarrow$  LISTO
2. Si  $e \notin T \Rightarrow \dots \Rightarrow T'$  también es AG de  $G$

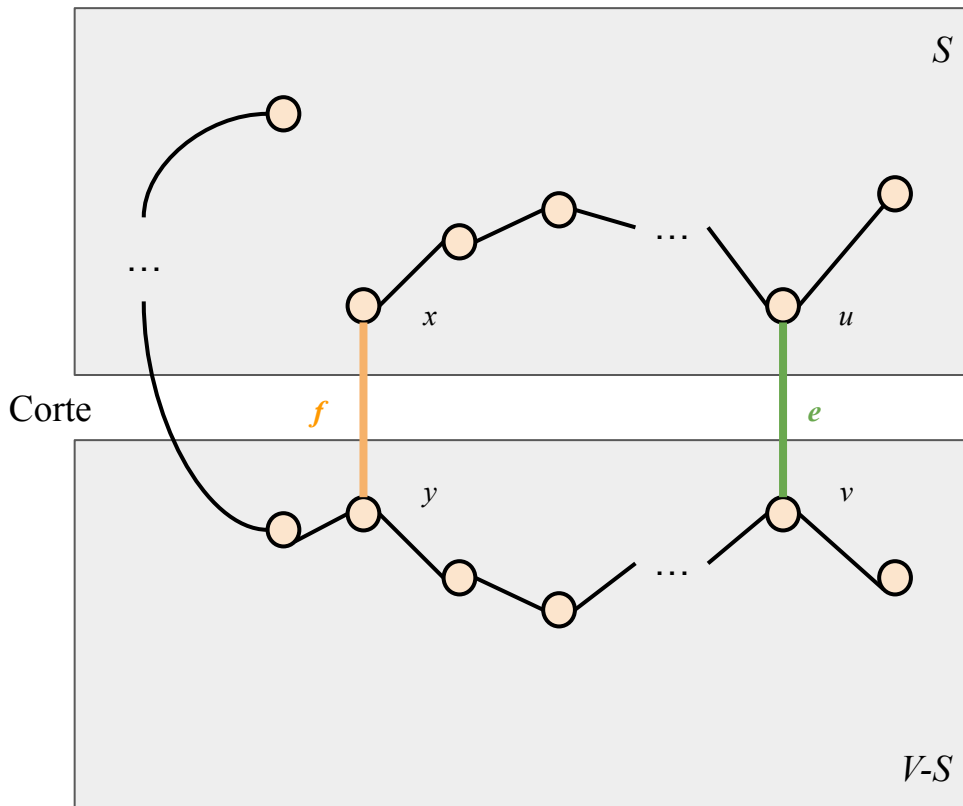
¿ $T'$  es AGM?

$$w(T') = \sum_{T'} w = \sum_T w - w(f) + w(e) = w(T) - w(f) + w(e)$$

Como  $w(e) \leq w(f)$  (elección golosa)

$$w(T') = \sum_{T'} w = \sum_T w - w(f) + w(e) = w(T) - w(f) + w(e) \leq w(T)$$

$\Rightarrow T'$  es AGM



# Prim (1957; versión CLRS cap. 21)

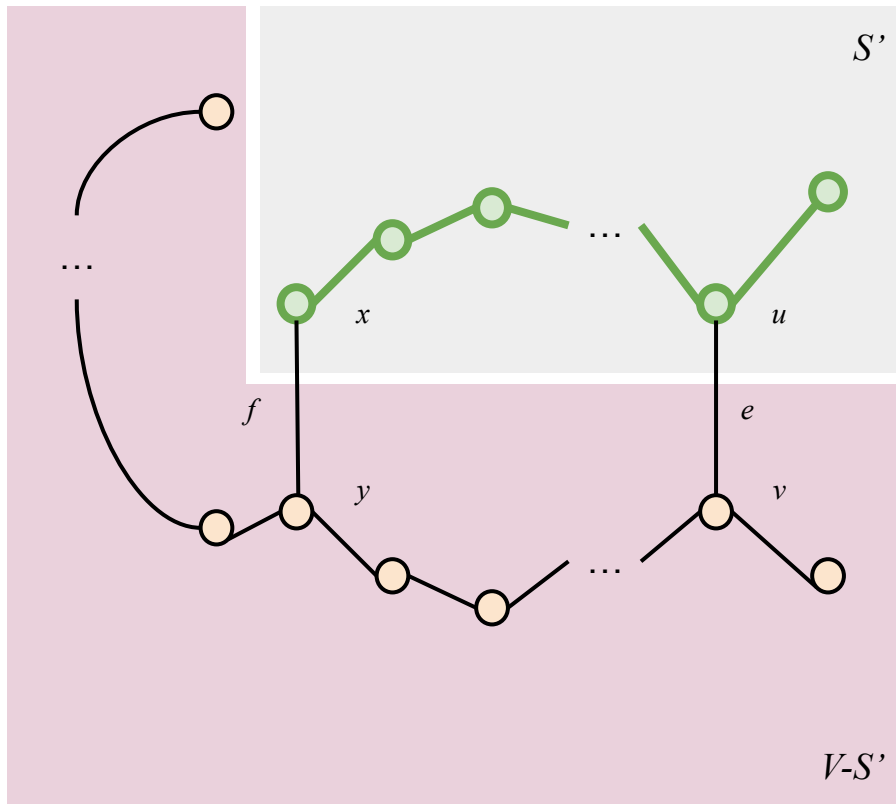
```
PRIM ( r , G ) :  
|   for u in V:  
|   |   u.key      = Inf  
|   |   u.parent   = None  
|   r.key = 0  
|   Q = 0  
|   for u in G.V  
|   |   INSERT(Q,u) # Cola de prioridad (key)  
|   while Q :  
|   |   u = EXTRACT-MIN(Q)  
|   |   for v in Adj[u] :  
|   |   |   if (v in Q) AND (w(u,v) < v.key):  
|   |   |   |   v.parent = u  
|   |   |   |   v.key = w(u,v)  
|   |   |   |   DECREASE-KEY(Q, v, w(u,v))
```

# Prim (1957; versión CLRS cap. 21)

**Demostración (Correctitud).** (Inducción sobre las iteraciones)

Caso base:  $S = [r]$

Hipótesis inductiva:  $T_{S'}$  es AGM de  $S'$  (INVARIANTE)



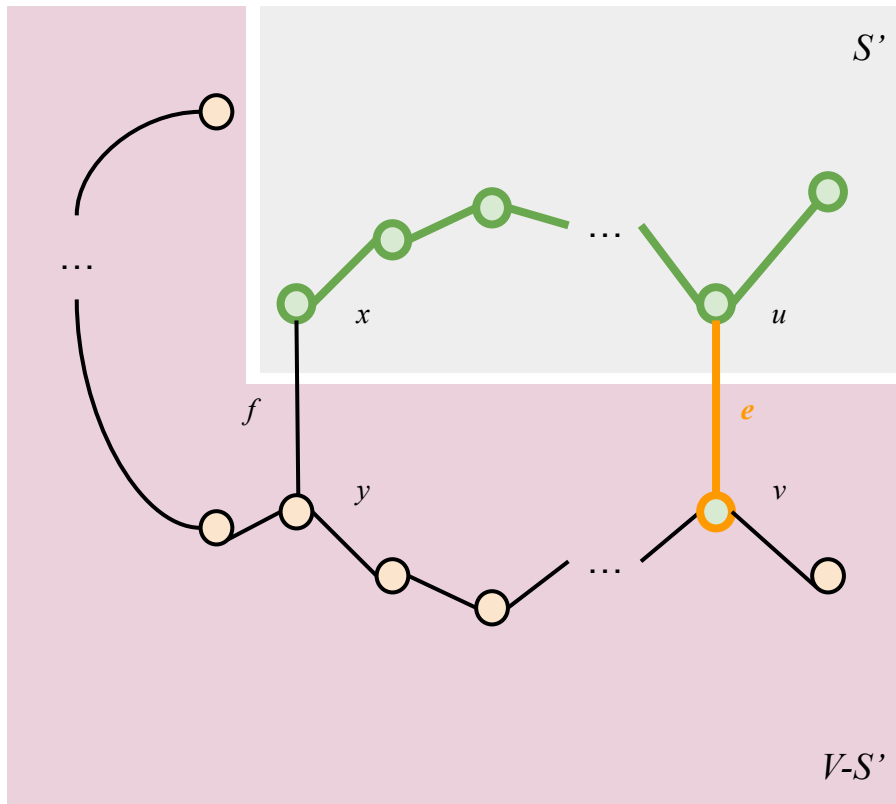
# Prim (1957; versión CLRS cap. 21)

**Demostración (Correctitud).** (Inducción sobre las iteraciones)

Caso base:  $S = [r]$

Hipótesis inductiva:  $T_{S'}$  es AGM de  $S'$  (INVARIANTE)

Paso inductivo: Agrego  $e=(u,v)$  tq  $u \in S'$ ,  $v \in V-S'$ , y  $w(e)$  es el mínimo de las aristas que cruzan el corte.



# Prim (1957; versión CLRS cap. 21)

**Demostración (Correctitud).** (Inducción sobre las iteraciones)

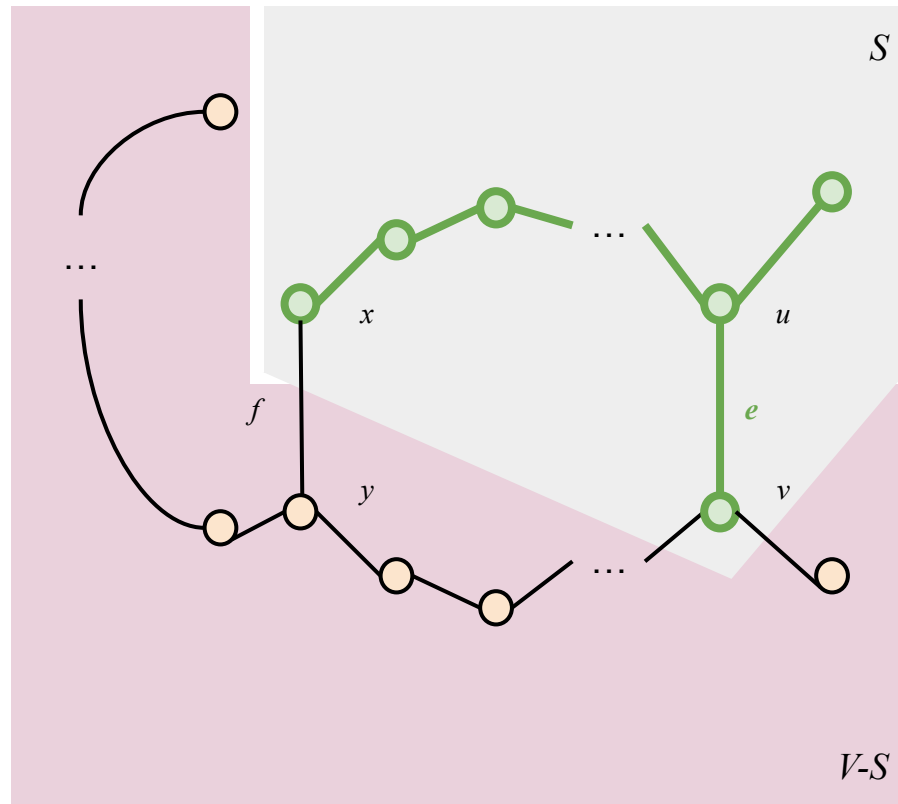
Caso base:  $S = [r]$

Hipótesis inductiva:  $T_{S'}$  es *AGM* de  $S'$  (INVARIANTE)

Paso inductivo: Agrego  $e=(u,v)$  tq  $u \in S$ ,  $v \in V-S$ , y  $w(e)$  es el mínimo de las aristas que cruzan el corte.

$T_S$  es *AGM* de  $S$  por propiedad golosa (demo anterior)

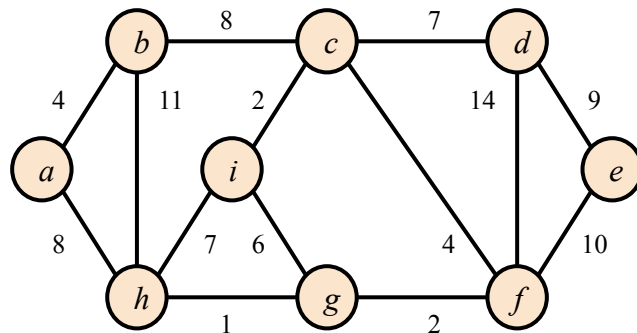
... Hasta completar el grafo.



# Prim (1957; versión CLRS cap. 21)

key = {a:0, b:Inf, c:Inf, d:Inf, e:Inf, f:Inf, g:Inf, h:Inf, i:Inf}  
parent = {a:None, b:None, c:None, d:None, e:None, f:None, g:None, h:None, i:None}  
Q = [a, b, c, d, e, f, g, h, i]

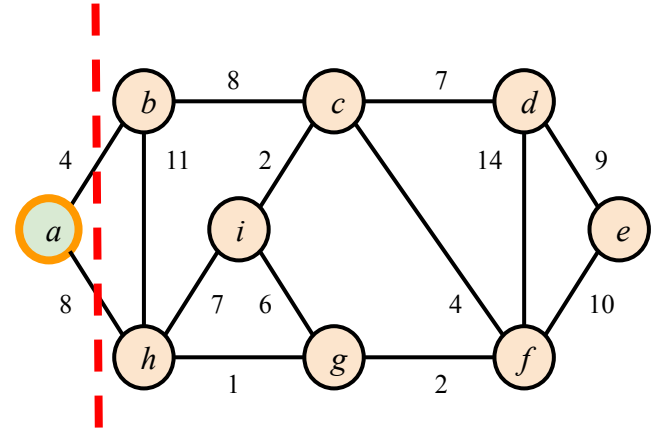
```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
|   r.key = 0  
|   Q = ∅  
|   for u in G.V  
|       INSERT(Q,u) # Cola de prioridad (key)  
|   while Q :  
|       u = EXTRACT-MIN(Q)  
|       for v in Adj[u] :  
|           if (v in Q) AND (w(u,v) < v.key):  
|               v.parent = u  
|               v.key = w(u,v)  
|               DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = a  
key = {a:0, b:Inf, c:Inf, d:Inf, e:Inf, f:Inf, g:Inf, h:Inf, i:Inf}  
parent = {a:None, b:None, c:None, d:None, e:None, f:None, g:None, h:None, i:None}  
Q = [b, c, d, e, f, g, h, i]

```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
|   r.key = 0  
|   Q = ∅  
|   for u in G.V  
|       INSERT(Q,u) # Cola de prioridad (key)  
|   while Q :  
|       u = EXTRACT-MIN(Q)  
|       for v in Adj[u] :  
|           if (v in Q) AND (w(u,v) < v.key):  
|               v.parent = u  
|               v.key = w(u,v)  
|               DECREASE-KEY(Q, v, w(u,v))
```

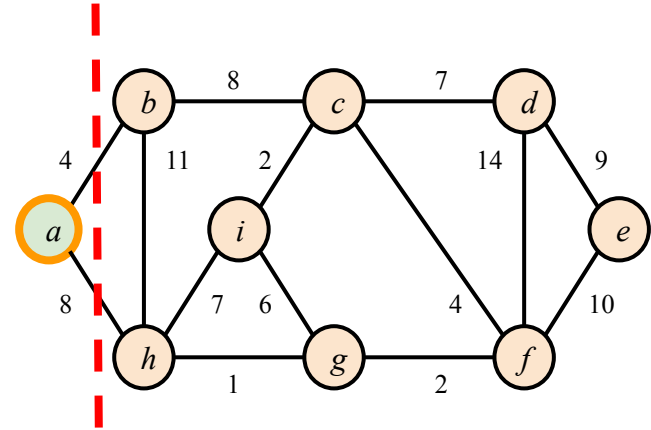




# Prim (1957; versión CLRS cap. 21)

u = a  
key = {a:0, b:4, c:Inf, d:Inf, e:Inf, f:Inf, g:Inf, h:8, i:Inf}  
parent = {a:None, b:a, c:None, d:None, e:None, f:None, g:None, h:a, i:None}  
Q = [b, h, c, d, e, f, g, i]

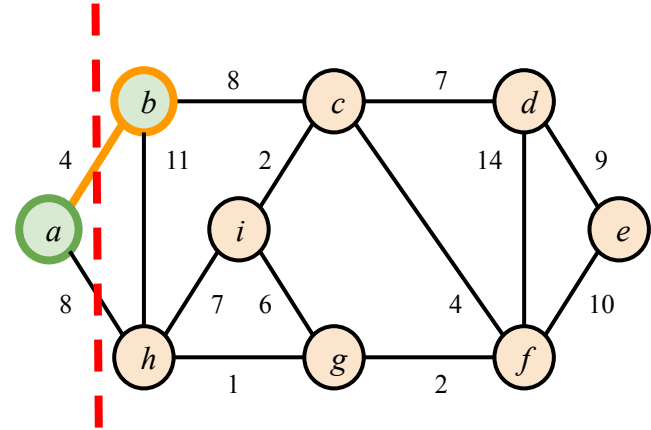
```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
|   r.key = 0  
|   Q = ∅  
|   for u in G.V  
|       INSERT(Q,u) # Cola de prioridad (key)  
|   while Q :  
|       u = EXTRACT-MIN(Q)  
|       for v in Adj[u] :  
|           if (v in Q) AND (w(u,v) < v.key):  
|               v.parent = u  
|               v.key = w(u,v)  
|               DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = b  
key = {a:0, b:4, c:Inf, d:Inf, e:Inf, f:Inf, g:Inf, h:8, i:Inf}  
parent = {a:None, b:a, c:None, d:None, e:None, f:None, g:None, h:a, i:None}  
Q = [h, c, d, e, f, g, i]

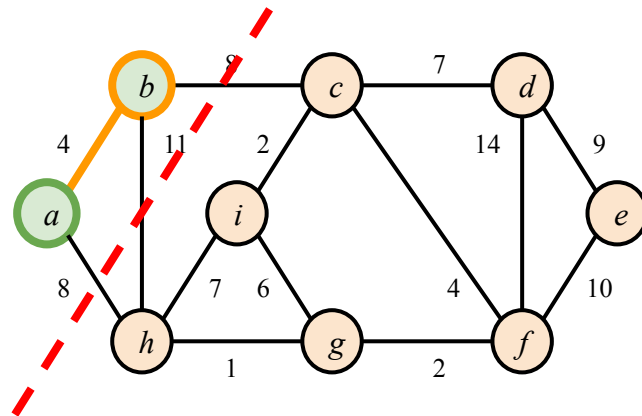
```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
|   r.key = 0  
|   Q = ∅  
|   for u in G.V  
|       INSERT(Q,u) # Cola de prioridad (key)  
|   while Q :  
|       u = EXTRACT-MIN(Q)  
|       for v in Adj[u] :  
|           if (v in Q) AND (w(u,v) < v.key):  
|               v.parent = u  
|               v.key = w(u,v)  
|               DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = b  
key = {a:0, b:4, c:8, d:Inf, e:Inf, f:Inf, g:Inf, h:8, i:Inf}  
parent = {a:None, b:a, c:b, d:None, e:None, f:None, g:None, h:a, i:None}  
Q = [c, h, d, e, f, g, i]

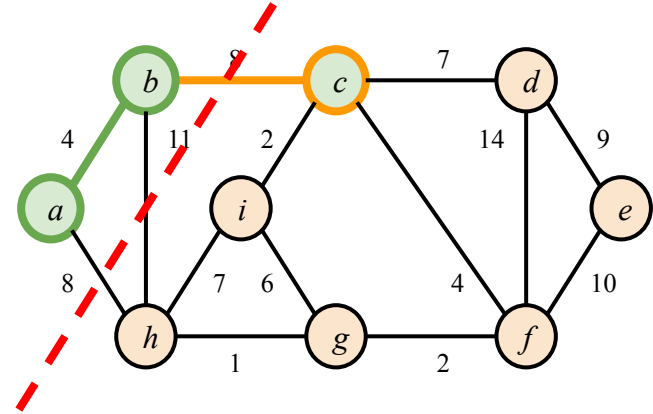
```
Prim ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
|   r.key = 0  
|   Q = ∅  
|   for u in G.V  
|       INSERT(Q,u) # Cola de prioridad (key)  
|   while Q :  
|       u = EXTRACT-MIN(Q)  
|       for v in Adj[u] :  
|           if (v in Q) AND (w(u,v) < v.key):  
|               v.parent = u  
|               v.key = w(u,v)  
|               DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = c  
key = {a:0, b:4, c:8, d:Inf, e:Inf, f:Inf, g:Inf, h:8, i:Inf}  
parent = {a:None, b:a, c:b, d:None, e:None, f:None, g:None, h:a, i:None}  
Q = [h, d, e, f, g, i]

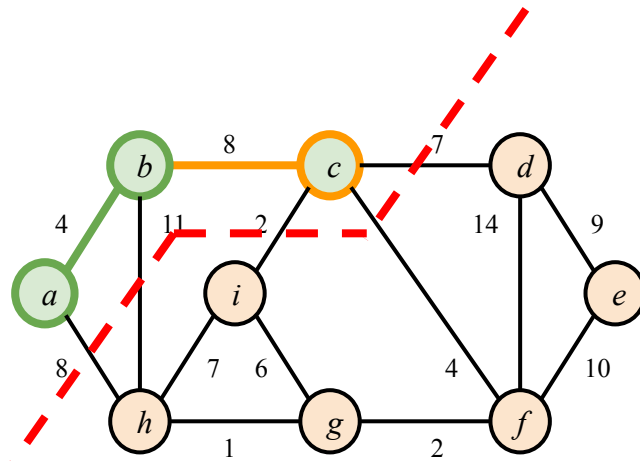
```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
|   r.key = 0  
|   Q = ∅  
|   for u in G.V  
|       INSERT(Q,u) # Cola de prioridad (key)  
|   while Q :  
|       u = EXTRACT-MIN(Q)  
|       for v in Adj[u] :  
|           if (v in Q) AND (w(u,v) < v.key):  
|               v.parent = u  
|               v.key = w(u,v)  
|               DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = c  
key = {a:0, b:4, c:8, d:7, e:Inf, f:4, g:Inf, h:8, i:2}  
parent = {a:None, b:a, c:b, d:c, e:None, f:c, g:None, h:a, i:c}  
Q = [i, f, d, h, e, g]

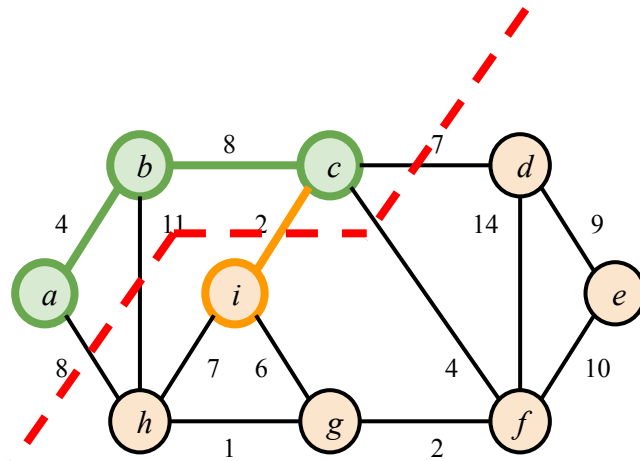
```
PRIM ( r , G ) :  
    for u in V:  
        | u.key      = Inf  
        | u.parent   = None  
    r.key = 0  
    Q = ∅  
    for u in G.V  
        | INSERT(Q,u) # Cola de prioridad (key)  
    while Q :  
        | u = EXTRACT-MIN(Q)  
        | for v in Adj[u] :  
            | if (v in Q) AND (w(u,v) < v.key):  
                | v.parent = u  
                | v.key = w(u,v)  
                | DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = i  
key = {a:0, b:4, c:8, d:7, e:Inf, f:4, g:Inf, h:8, i:2}  
parent = {a:None, b:a, c:b, d:c, e:None, f:c, g:None, h:a, i:c}  
Q = [f, d, h, e, g]

```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
r.key = 0  
Q = ∅  
for u in G.V  
|   INSERT(Q,u) # Cola de prioridad (key)  
while Q :  
|   u = EXTRACT-MIN(Q)  
|   for v in Adj[u] :  
|       if (v in Q) AND (w(u,v) < v.key):  
|           v.parent = u  
|           v.key = w(u,v)  
|           DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

```

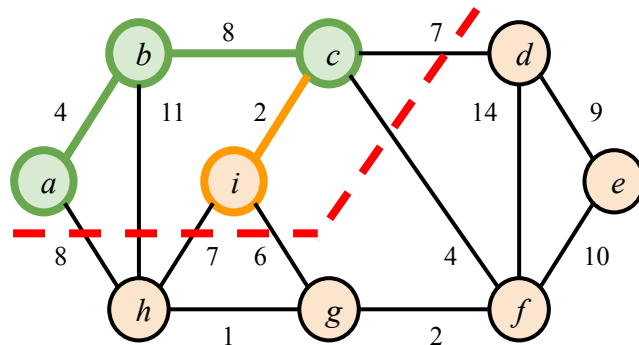
u      = i
key    = {a:0, b:4, c:8, d:7, e:Inf, f:4, g:6, h:7, i:2}
parent = {a:None, b:a, c:b, d:c, e:None, f:c, g:i, h:i, i:c}
Q      = [f, g, h, d, e]

```

```

PRIM ( r , G ) :
|   for u in V:
|       |   u.key      = Inf
|       |   u.parent   = None
|   r.key = 0
|   Q = ∅
|   for u in G.V
|       |   INSERT(Q,u) # Cola de prioridad (key)
|   while Q :
|       |   u = EXTRACT-MIN(Q)
|       |   for v in Adj[u] :
|           |       if (v in Q) AND (w(u,v) < v.key):
|               |       |   v.parent = u
|               |       |   v.key = w(u,v)
|               |       |   DECREASE-KEY(Q, v, w(u,v))

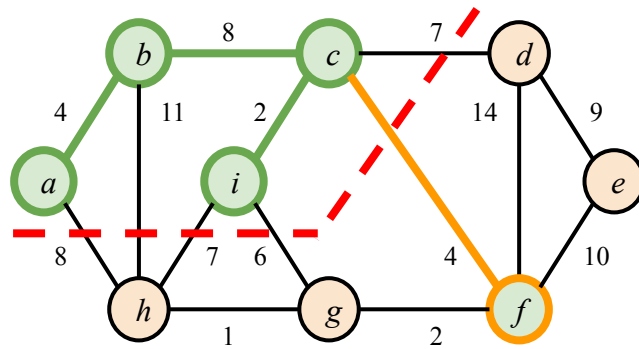
```



# Prim (1957; versión CLRS cap. 21)

u = f  
key = {a:0, b:4, c:8, d:7, e:Inf, f:4, g:6, h:7, i:2}  
parent = {a:None, b:a, c:b, d:c, e:None, f:c, g:i, h:i, i:c}  
Q = [g, h, d, e]

```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
|   r.key = 0  
|   Q = ∅  
|   for u in G.V  
|       INSERT(Q,u) # Cola de prioridad (key)  
|   while Q :  
|       u = EXTRACT-MIN(Q)  
|       for v in Adj[u] :  
|           if (v in Q) AND (w(u,v) < v.key):  
|               v.parent = u  
|               v.key = w(u,v)  
|               DECREASE-KEY(Q, v, w(u,v))
```

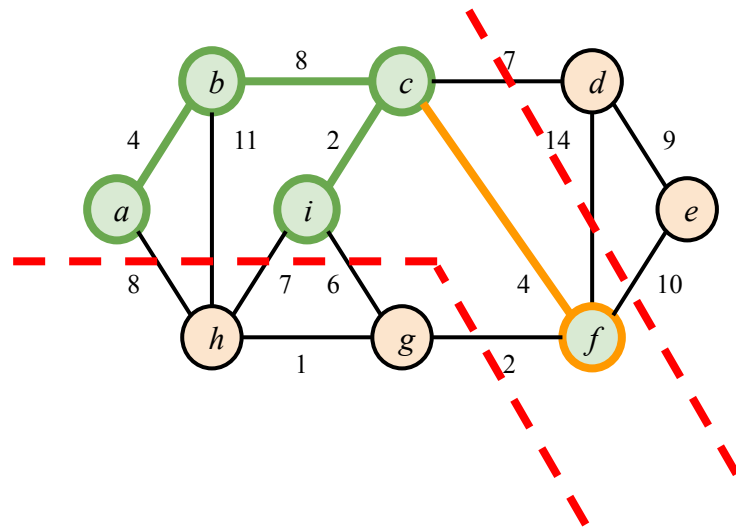




# Prim (1957; versión CLRS cap. 21)

u = f  
key = {a:0, b:4, c:8, d:7, e:10, f:4, g:2, h:7, i:2}  
parent = {a:None, b:a, c:b, d:c, e:f, f:c, g:f, h:i, i:c}  
Q = [g, h, d, e]

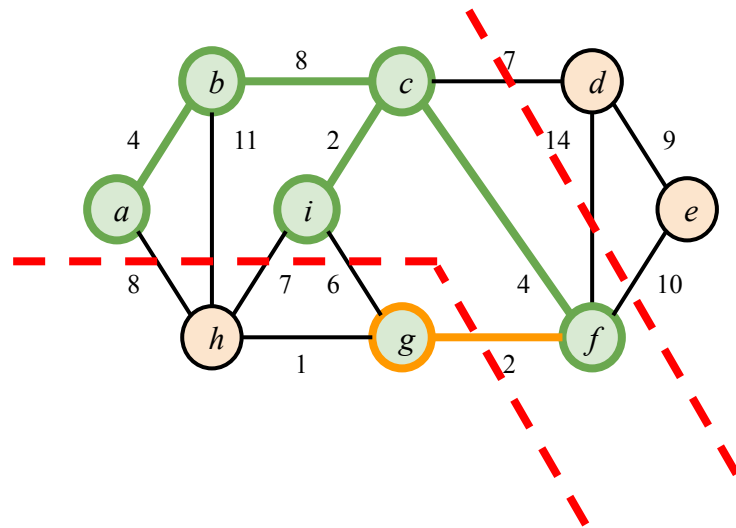
```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
r.key = 0  
Q = ∅  
for u in G.V  
|   INSERT(Q,u) # Cola de prioridad (key)  
while Q :  
|   u = EXTRACT-MIN(Q)  
|   for v in Adj[u] :  
|       if (v in Q) AND (w(u,v) < v.key):  
|           v.parent = u  
|           v.key = w(u,v)  
|           DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = g  
key = {a:0, b:4, c:8, d:7, e:10, f:4, g:2, h:7, i:2}  
parent = {a:None, b:a, c:b, d:c, e:f, f:c, g:f, h:i, i:c}  
Q = [h, d, e]

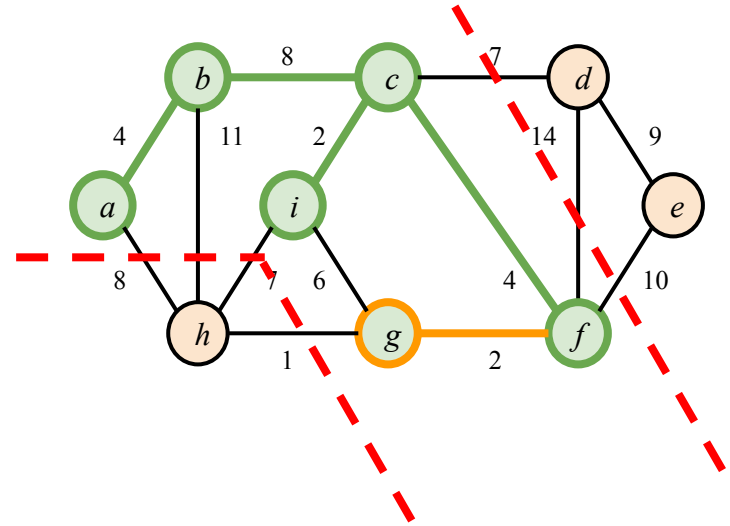
```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
r.key = 0  
Q = ∅  
for u in G.V  
|   INSERT(Q,u) # Cola de prioridad (key)  
while Q :  
|   u = EXTRACT-MIN(Q)  
|   for v in Adj[u] :  
|       |   if (v in Q) AND (w(u,v) < v.key):  
|       |       |   v.parent = u  
|       |       |   v.key = w(u,v)  
|       |       |   DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = g  
key = {a:0, b:4, c:8, d:7, e:10, f:4, g:2, h:1, i:2}  
parent = {a:None, b:a, c:b, d:c, e:f, f:c, g:f, h:g, i:c}  
Q = [h, d, e]

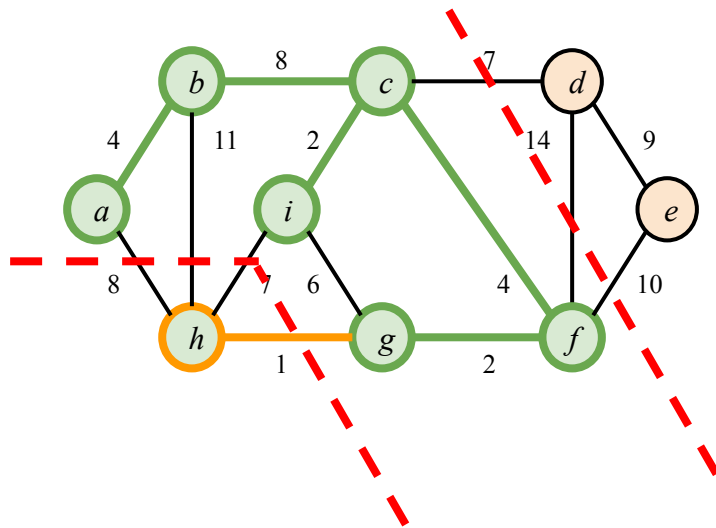
```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
r.key = 0  
Q = ∅  
for u in G.V  
|   INSERT(Q,u) # Cola de prioridad (key)  
while Q :  
|   u = EXTRACT-MIN(Q)  
|   for v in Adj[u] :  
|       if (v in Q) AND (w(u,v) < v.key):  
|           v.parent = u  
|           v.key = w(u,v)  
|           DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = h  
key = {a:0, b:4, c:8, d:7, e:10, f:4, g:2, h:1, i:2}  
parent = {a:None, b:a, c:b, d:c, e:f, f:c, g:f, h:g, i:c}  
Q = [d, e]

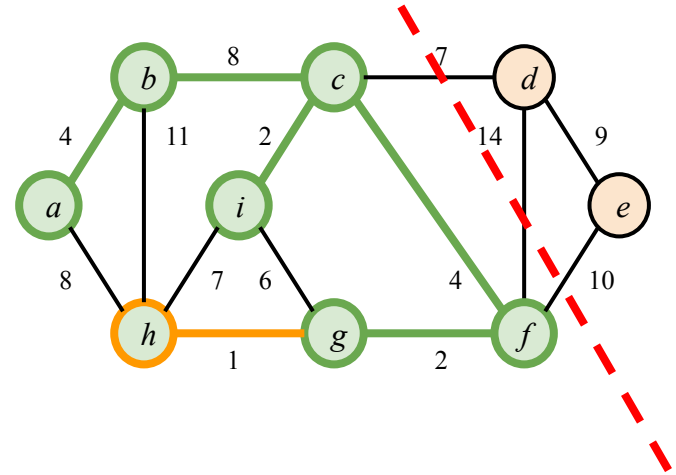
```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
r.key = 0  
Q = ∅  
for u in G.V  
|   INSERT(Q,u) # Cola de prioridad (key)  
while Q :  
|   u = EXTRACT-MIN(Q)  
|   for v in Adj[u] :  
|       |   if (v in Q) AND (w(u,v) < v.key):  
|       |       |   v.parent = u  
|       |       |   v.key = w(u,v)  
|       |       |   DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = h  
key = {a:0, b:4, c:8, d:7, e:10, f:4, g:2, h:1, i:2}  
parent = {a:None, b:a, c:b, d:c, e:f, f:c, g:f, h:g, i:c}  
Q = [d, e]

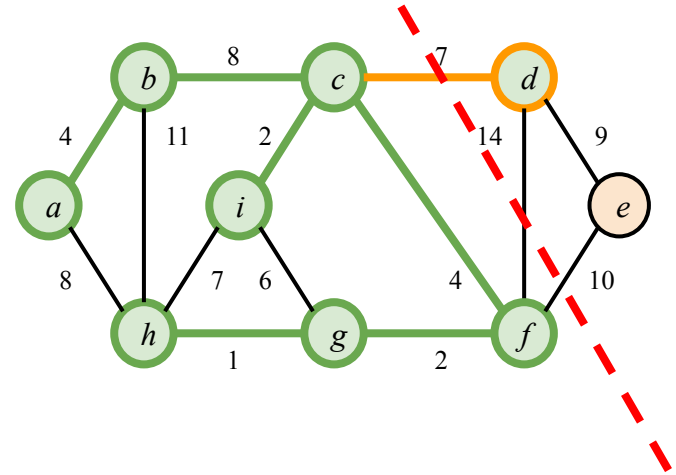
```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
r.key = 0  
Q = ∅  
for u in G.V  
|   INSERT(Q,u) # Cola de prioridad (key)  
while Q :  
|   u = EXTRACT-MIN(Q)  
|   for v in Adj[u] :  
|       if (v in Q) AND (w(u,v) < v.key):  
|           v.parent = u  
|           v.key = w(u,v)  
|           DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = d  
key = {a:0, b:4, c:8, d:7, e:10, f:4, g:2, h:1, i:2}  
parent = {a:None, b:a, c:b, d:c, e:f, f:c, g:f, h:g, i:c}  
Q = [e]

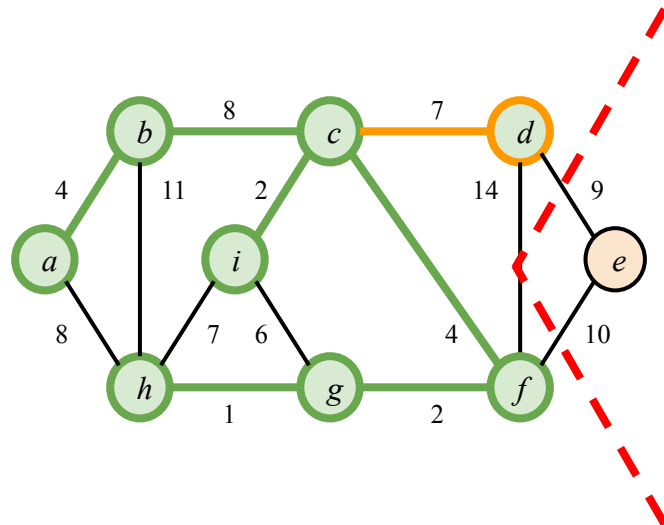
```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
|   r.key = 0  
|   Q = ∅  
|   for u in G.V  
|       INSERT(Q,u) # Cola de prioridad (key)  
|   while Q :  
|       u = EXTRACT-MIN(Q)  
|       for v in Adj[u] :  
|           if (v in Q) AND (w(u,v) < v.key):  
|               v.parent = u  
|               v.key = w(u,v)  
|               DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = d  
key = {a:0, b:4, c:8, d:7, e:9, f:4, g:2, h:1, i:2}  
parent = {a:None, b:a, c:b, d:c, e:d, f:c, g:f, h:g, i:c}  
Q = [e]

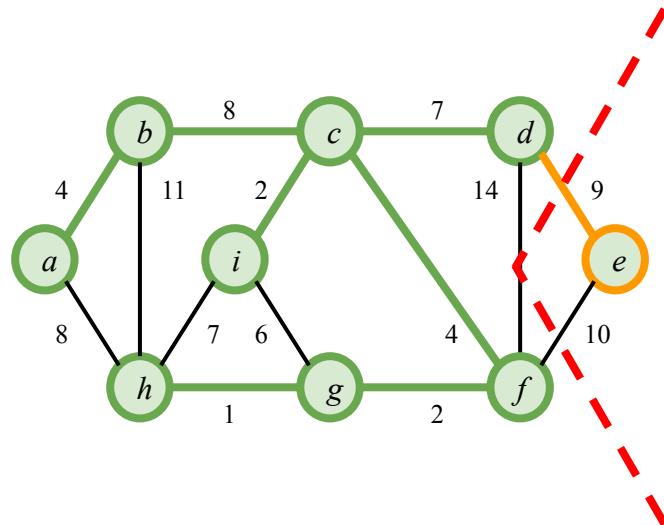
```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
r.key = 0  
Q = ∅  
for u in G.V  
|   INSERT(Q,u) # Cola de prioridad (key)  
while Q :  
|   u = EXTRACT-MIN(Q)  
|   for v in Adj[u] :  
|       if (v in Q) AND (w(u,v) < v.key):  
|           v.parent = u  
|           v.key = w(u,v)  
|           DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

u = e  
key = {a:0, b:4, c:8, d:7, e:9, f:4, g:2, h:1, i:2}  
parent = {a:None, b:a, c:b, d:c, e:d, f:c, g:f, h:g, i:c}  
Q = []

```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
r.key = 0  
Q = ∅  
for u in G.V  
|   INSERT(Q,u) # Cola de prioridad (key)  
while Q :  
|   u = EXTRACT-MIN(Q)  
|   for v in Adj[u] :  
|       |   if (v in Q) AND (w(u,v) < v.key):  
|       |       |   v.parent = u  
|       |       |   v.key = w(u,v)  
|       |       |   DECREASE-KEY(Q, v, w(u,v))
```

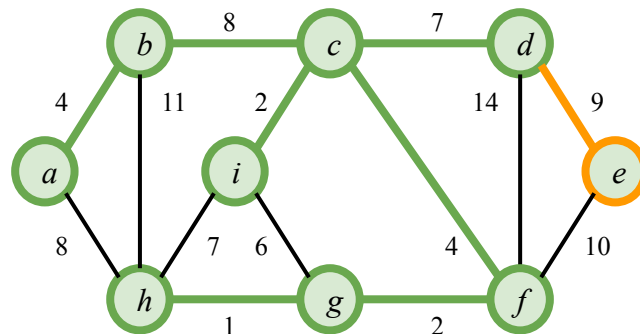




# Prim (1957; versión CLRS cap. 21)

u = e  
key = {a:0, b:4, c:8, d:7, e:9, f:4, g:2, h:1, i:2}  
parent = {a:None, b:a, c:b, d:c, e:d, f:c, g:f, h:g, i:c}  
Q = []

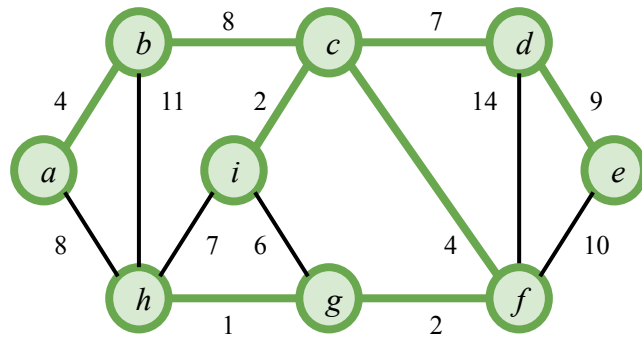
```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
r.key = 0  
Q = ∅  
for u in G.V  
|   INSERT(Q,u) # Cola de prioridad (key)  
while Q :  
|   u = EXTRACT-MIN(Q)  
|   for v in Adj[u] :  
|       if (v in Q) AND (w(u,v) < v.key):  
|           v.parent = u  
|           v.key = w(u,v)  
|           DECREASE-KEY(Q, v, w(u,v))
```



# Prim (1957; versión CLRS cap. 21)

key = {a:0, b:4, c:8, d:7, e:9, f:4, g:2, h:1, i:2}  
parent = {a:None, b:a, c:b, d:c, e:d, f:c, g:f, h:g, i:c}  
Q = []

```
PRIM ( r , G ) :  
|   for u in V:  
|       u.key      = Inf  
|       u.parent   = None  
r.key = 0  
Q = ∅  
for u in G.V  
|   INSERT(Q,u) # Cola de prioridad (key)  
while Q :  
|   u = EXTRACT-MIN(Q)  
|   for v in Adj[u] :  
|       |   if (v in Q) AND (w(u,v) < v.key):  
|       |       |   v.parent = u  
|       |       |   v.key = w(u,v)  
|       |       |   DECREASE-KEY(Q, v, w(u,v))
```



## Prim (1957; versión CLRS cap. 21)

```

PRIM ( r , G ) :
|   for u in V:
|       |   u.key      = Inf
|       |   u.parent   = None
|   r.key = 0
|   Q = ∅
|   for u in G.V
|       |   INSERT(Q,u) # Cola de prioridad (key)
|   while Q :
|       |   u = EXTRACT-MIN(Q)
|       |   for v in Adj[u] :
|           |   if (v in Q) AND (w(u,v) < v.key):
|               |   v.parent = u
|               |   v.key = w(u,v)
|               |   DECREASE-KEY(Q, v, w(u,v))

```

*Complexity Analysis:*

- Initialization (lines 4-6):  $O(V)$
- Inserting all vertices into the priority queue (line 8):  $O(V)$  (Min-Heap)
- Extracting the minimum vertex (line 10):  $O(\log(V))$  (Min-Heap)
- Processing all edges (lines 12-14):  $O(E \log(V))$  (Min-Heap)

**Total Complexity:  $O(V \log(V) + E \log(V))$**

$$\text{Min-Heap: } O( V + V^* \log(V) + E^* \log(V) ) = O( E^* \log(V) )$$

# Prim (1957; versión CLRS cap. 21)

```

PRIM ( r , G ) :
|   for u in V:
|       |   u.key      = Inf
|       |   u.parent   = None
|   r.key = 0
|   Q = ∅
|   for u in G.V
|       |   INSERT(Q,u) # Cola de prioridad (key)
|   while Q :
|       |   u = EXTRACT-MIN(Q)
|       |   for v in Adj[u] :
|           |   if (v in Q) AND (w(u,v) < v.key):
|               |   v.parent = u
|               |   v.key = w(u,v)
|               |   DECREASE-KEY(Q, v, w(u,v))

```

*$O(V)$*

*$\text{Fibonacci-Heap: } O(1)$*

*$\text{Fibonacci-Heap: } O(\log(V))$*

*$\text{Fibonacci-Heap: } O(V \cdot \log(V))$*

*$\text{Fibonacci-Heap: } O(E)$*

*$\text{Fibonacci-Heap: } O(1)$*

*Min-Heap:  $O(V + V \cdot \log(V) + E \cdot \log(V)) = O(E \cdot \log(V))$*

*Fibonacci-Heap:  $O(V + V \cdot \log(V) + E) = O(V \cdot \log(V) + E)$*

*Grafos raros  $E \sim V$ : Min-Heap  $\sim$  Fibonacci-Heap:  $O(V \cdot \log(V))$*

*Grafos densos  $E \sim V^2$  Fibonacci-Heap:  $O(V^2)$ , Min-Heap  $O(V^2 \cdot \log(V))$*

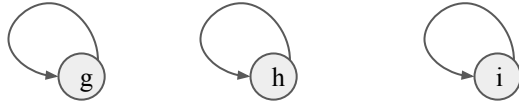
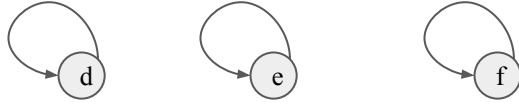
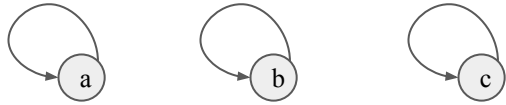
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           |   A = A  $\cup$  {e}  
|           |   UNION(u,v)
```

# Disjoint-set / Union-Find (CLRS cap. 19)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           |   A = A  $\cup$  {e}  
|           |   UNION(u,v)
```

# Disjoint-set / Union-Find (CLRS cap. 19)



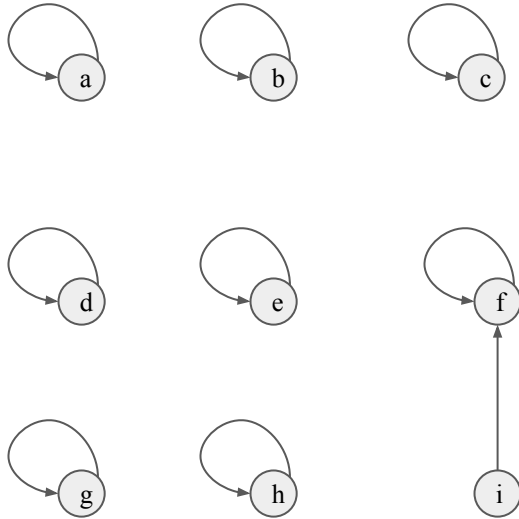
MAKE-SET (  $x$  ) , crea un SET con un único elemento  $x$ .

UNION (  $x, y$  ) , asigna un único representante a ambos SET  $S_x$  y  $S_y$ . Puede o no ser uno de los representantes anteriores.

FIND-SET (  $x$  ) , devuelve el representante de  $x$ .

MAKE-SET (  $x$  )

# Disjoint-set / Union-Find (CLRS cap. 19)



MAKE-SET (  $x$  ) , crea un SET con un único elemento  $x$ .

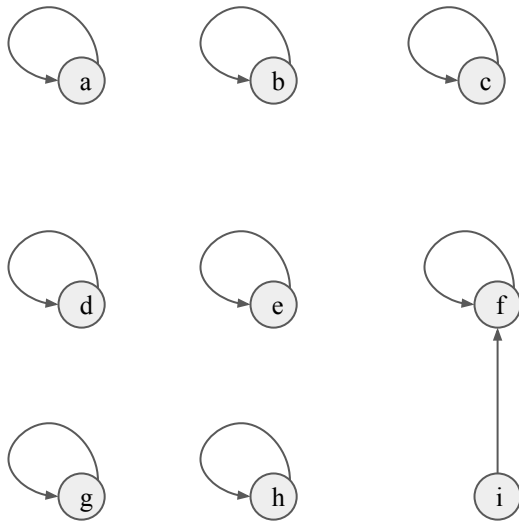
UNION (  $x, y$  ) , asigna un único representante a ambos SET  $S_x$  y  $S_y$ . Puede o no ser uno de los representantes anteriores.

FIND-SET (  $x$  ) , devuelve el representante de  $x$ .

UNION (  $f, i$  )



# Disjoint-set / Union-Find (CLRS cap. 19)



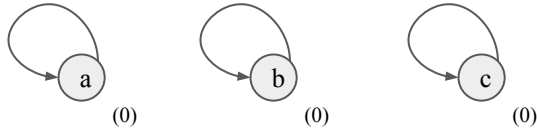
`MAKE-SET ( x )` , crea un SET con un único elemento  $x$ .

`UNION ( x, y )` , asigna un único representante a ambos SET  $S_x$  y  $S_y$ . Puede o no ser uno de los representantes anteriores.

`FIND-SET ( x )` , devuelve el representante de  $x$ .

`FIND-SET ( i )  $\rightarrow$  f (representante)`  
`FIND-SET ( i ) == FIND-SET ( f )  $\rightarrow$  VERDADERO`  
`FIND-SET ( i ) != FIND-SET ( e )  $\rightarrow$  VERDADERO`

# Disjoint-set Opt. *Union by rank* (CLRS cap. 19)

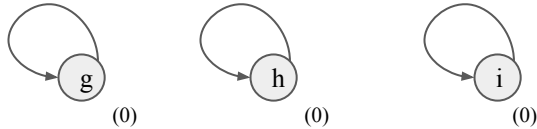
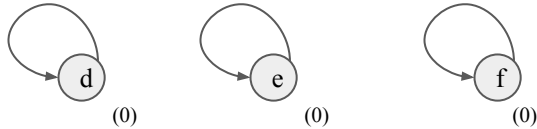


MAKE-SET (  $x$  ) , crea un SET con un único elemento  $x$ .

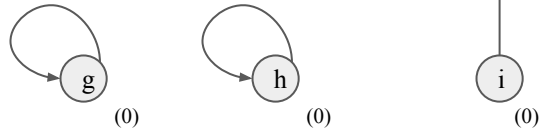
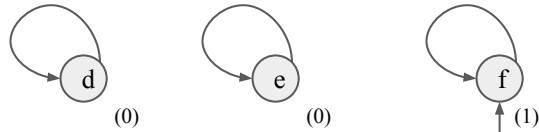
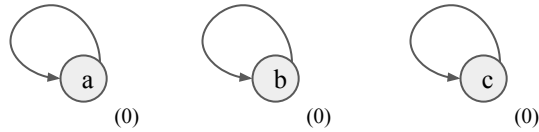
UNION (  $x, y$  ) , asigna un único representante a ambos SET  $S_x$  y  $S_y$ . Puede o no ser uno de los representantes anteriores.

FIND-SET (  $x$  ) , devuelve el representante de  $x$ .

MAKE-SET (  $x$  )



# Disjoint-set Opt. *Union by rank* (CLRS cap. 19)



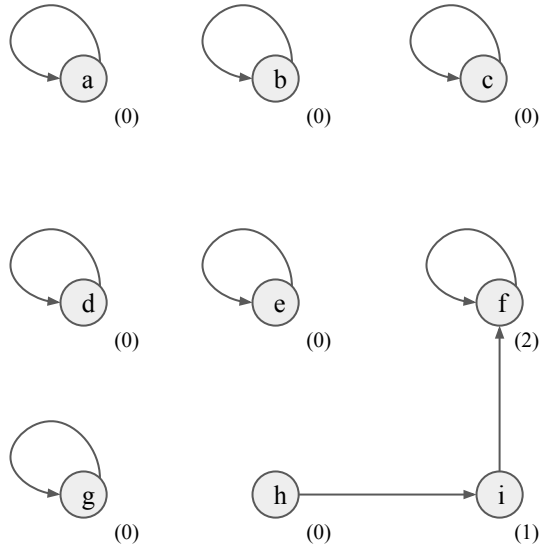
MAKE-SET (  $x$  ) , crea un SET con un único elemento  $x$ .

UNION (  $x, y$  ) , asigna un único representante a ambos SET  $S_x$  y  $S_y$ . Puede o no ser uno de los representantes anteriores.

FIND-SET (  $x$  ) , devuelve el representante de  $x$ .

UNION (  $f, i$  )

# Disjoint-set Opt. *Union by rank* (CLRS cap. 19)



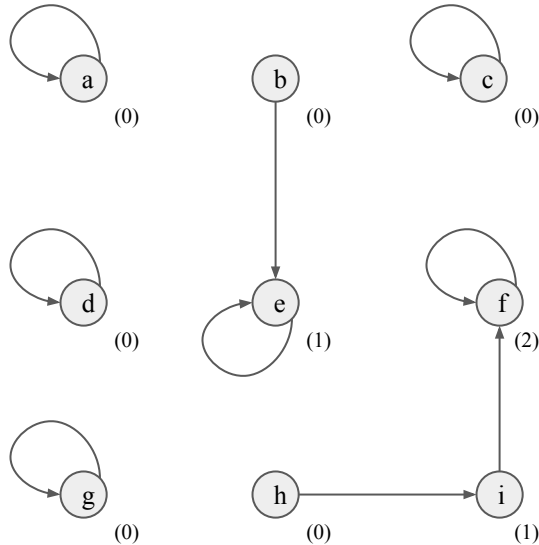
MAKE-SET (  $x$  ) , crea un SET con un único elemento  $x$ .

UNION (  $x, y$  ) , asigna un único representante a ambos SET  $S_x$  y  $S_y$ . Puede o no ser uno de los representantes anteriores.

FIND-SET (  $x$  ) , devuelve el representante de  $x$ .

UNION (  $f, i$  )  
UNION (  $h, i$  )

# Disjoint-set Opt. *Union by rank* (CLRS cap. 19)



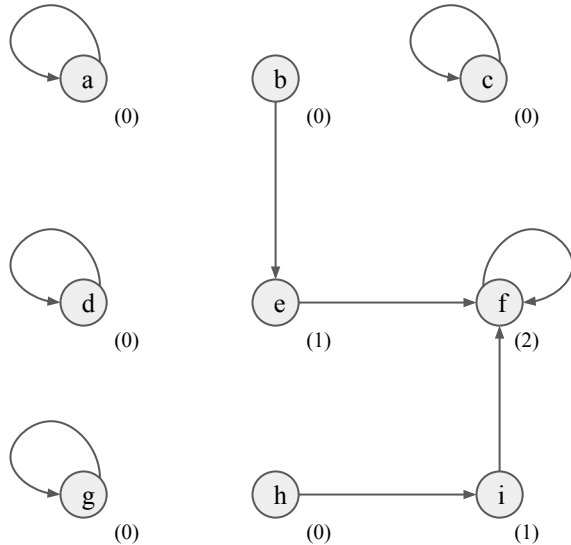
MAKE-SET (  $x$  ) , crea un SET con un único elemento  $x$ .

UNION (  $x, y$  ) , asigna un único representante a ambos SET  $S_x$  y  $S_y$ . Puede o no ser uno de los representantes anteriores.

FIND-SET (  $x$  ) , devuelve el representante de  $x$ .

UNION (  $f, i$  )  
UNION (  $h, i$  )  
UNION (  $b, e$  )

# Disjoint-set Opt. *Union by rank* (CLRS cap. 19)



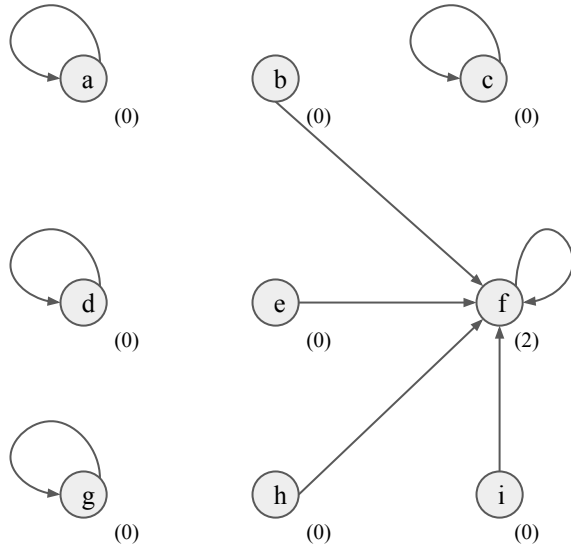
MAKE-SET (  $x$  ) , crea un SET con un único elemento  $x$ .

UNION (  $x, y$  ) , asigna un único representante a ambos SET  $S_x$  y  $S_y$ . Puede o no ser uno de los representantes anteriores.

FIND-SET (  $x$  ) , devuelve el representante de  $x$ .

UNION (  $f, i$  )  
UNION (  $h, i$  )  
UNION (  $b, e$  )  
UNION (  $f, e$  )

# Disjoint-set Opt. *Path Compression* (CLRS cap. 19)



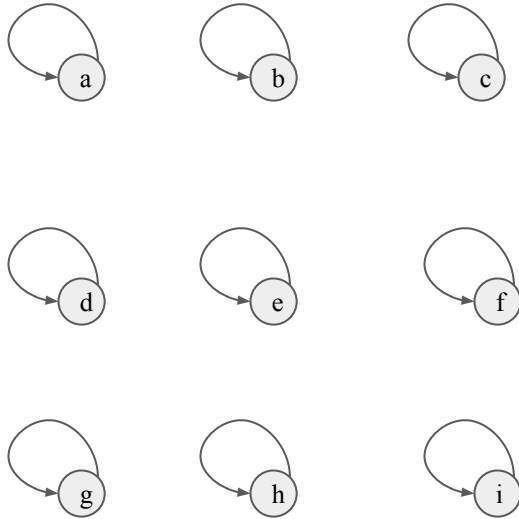
MAKE-SET (  $x$  ) , crea un SET con un único elemento  $x$ .

UNION (  $x, y$  ) , asigna un único representante a ambos SET  $S_x$  y  $S_y$ . Puede o no ser uno de los representantes anteriores.

FIND-SET (  $x$  ) , devuelve el representante de  $x$ .

UNION (  $f, i$  )  
UNION (  $h, i$  )  
UNION (  $b, e$  )  
UNION (  $f, e$  )

# Disjoint-set / Union-Find (CLRS cap. 19)



`MAKE-SET ( x )` , crea un SET con un único elemento  $x$ .

`UNION ( x, y )` , asigna un único representante a ambos SET  $S_x$  y  $S_y$ . Puede o no ser uno de los representantes anteriores.

`FIND-SET ( x )` , devuelve el representante de  $x$ .

## ¿Aplicaciones?

Ciclos,

Conjuntos conexos,

AGM,



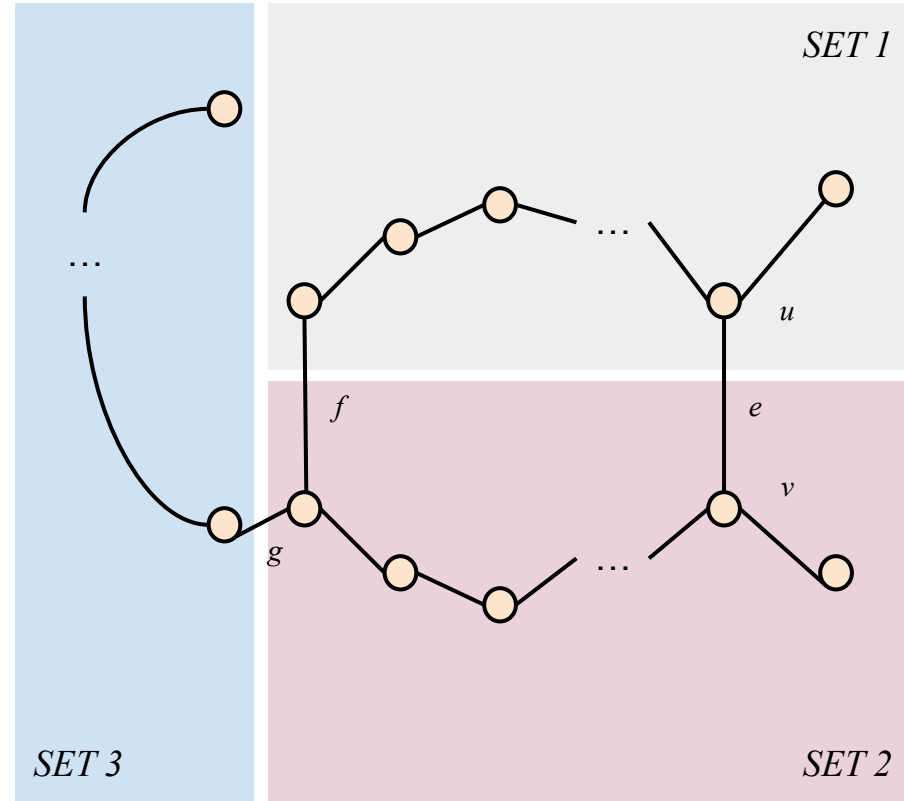
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           |   A = A  $\cup$  {e}  
|           |   UNION(u,v)
```

# Kruskal (1956; versión CLRS cap. 21)

DISJOINT-SET me genera muchos conjuntos,

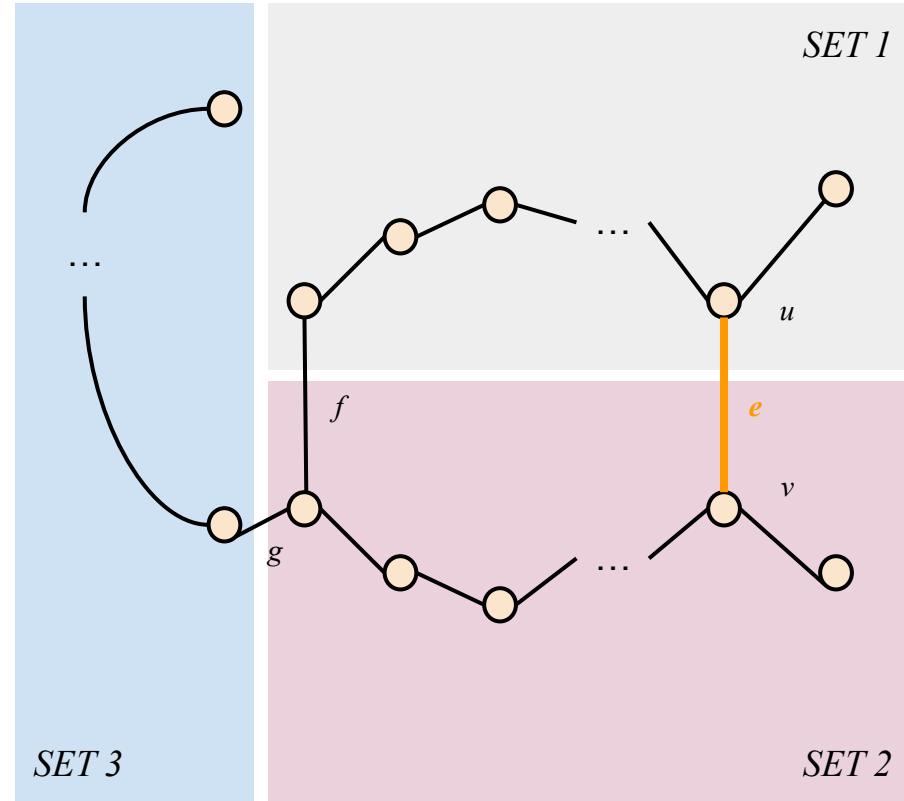
Busco la arista de menor  $w$  que una cualquiera dos conjuntos distintos.



# Kruskal (1956; versión CLRS cap. 21)

DISJOINT-SET me genera muchos conjuntos,

Busco la arista de menor  $w$  que una cualquiera dos conjuntos distintos.

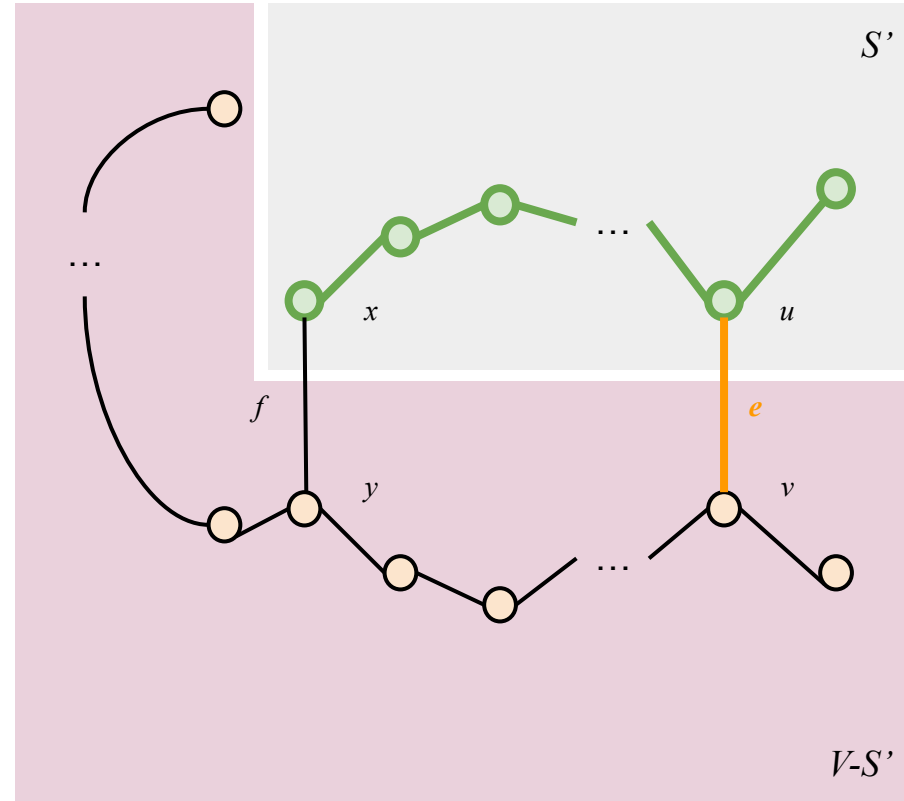


# Kruskal (1956; versión CLRS cap. 21)

DISJOINT-SET me genera muchos conjuntos,

Busco la arista de menor  $w$  que una cualquiera dos conjuntos distintos.

Y sigue la misma idea...



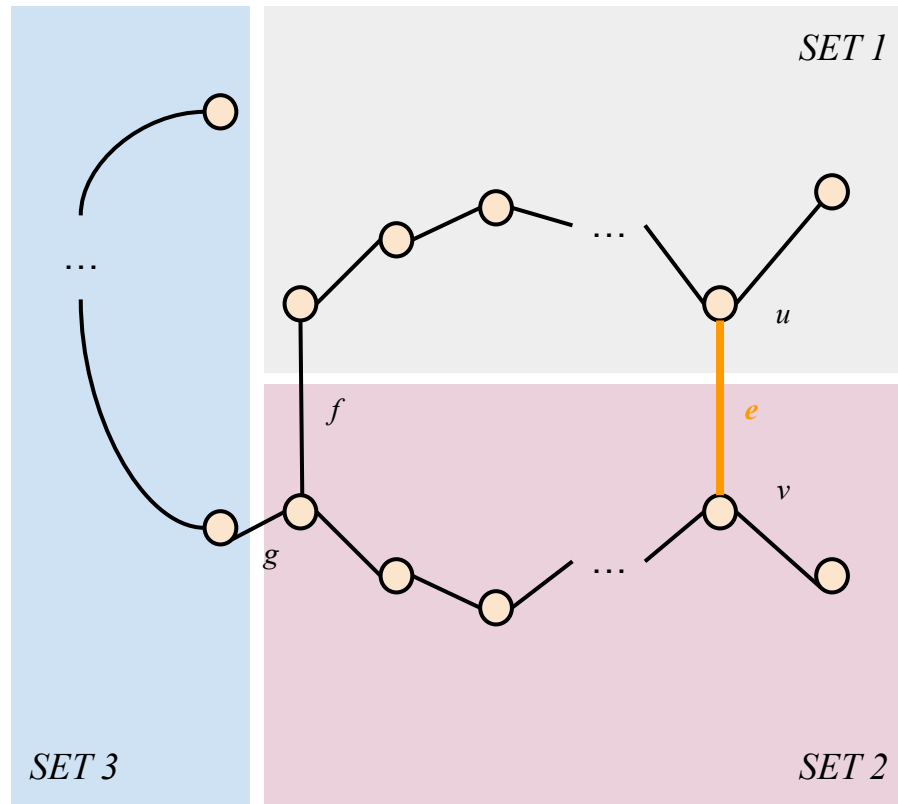
# Kruskal (1956; versión CLRS cap. 21)

**Demostración (Correctitud).** (Inducción sobre las iteraciones)

Caso base:  $S = [s]$

Hipótesis inductiva:  $T_k$  es bosque de  $AGM$  de  $G$  (INVARIANTE)

Paso inductivo: Agrego  $e=(u,v)$  y  $w(e)$  es el mínimo de las aristas que cruzan cualquier corte.



# Kruskal (1956; versión CLRS cap. 21)

**Demostración (Correctitud).** (Inducción sobre las iteraciones)

Caso base:  $S = [s]$

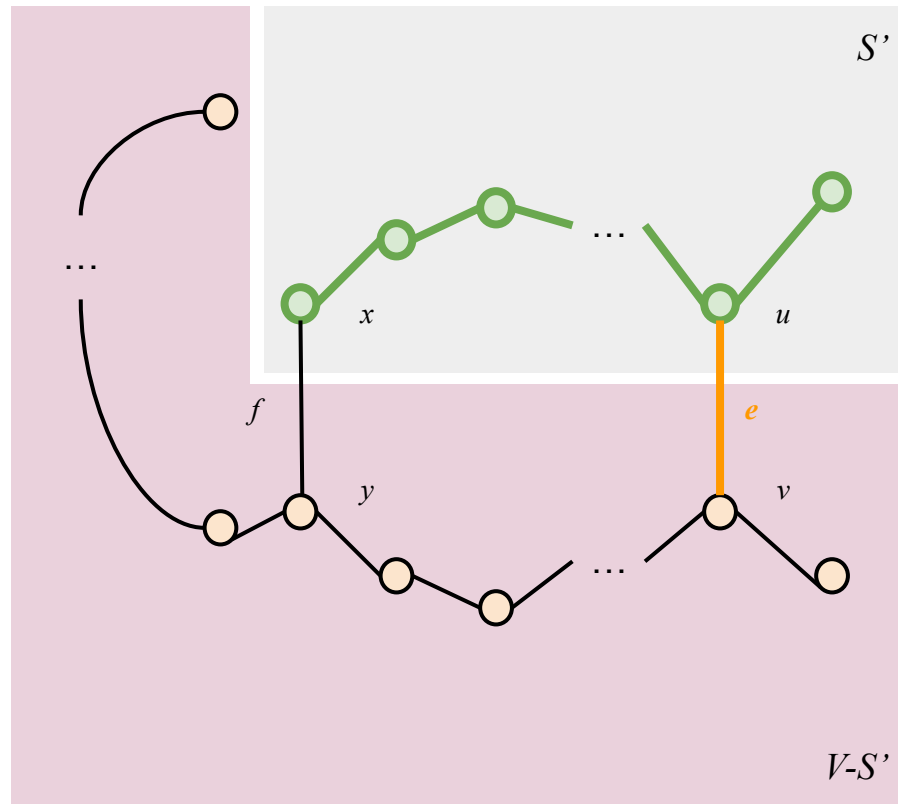
Hipótesis inductiva:  $T_k$  es bosque de  $AGM$  de  $G$  (INVARIANTE)

Paso inductivo: Agrego  $e=(u,v)$  y  $w(e)$  es el mínimo de las aristas que cruzan cualquier corte.

Defino  $S'$  tq  $u \in S'$ , y  $V-S'$  tq  $v \in V-S'$

$S = S' \cup \{v\}$ ,  $T_S = (V_S, E_{T_S} + e)$

$T_S$  es  $AGM$  de  $S$  por propiedad golosa (demo goloso)



# Kruskal (1956; versión CLRS cap. 21)

**Demostración (Correctitud).** (Inducción sobre las iteraciones)

Caso base:  $S = [s]$

Hipótesis inductiva:  $T_k$ , es bosque de  $AGM$  de  $G$  (INVARIANTE)

Paso inductivo: Agrego  $e=(u,v)$  y  $w(e)$  es el mínimo de las aristas que cruzan cualquier corte.

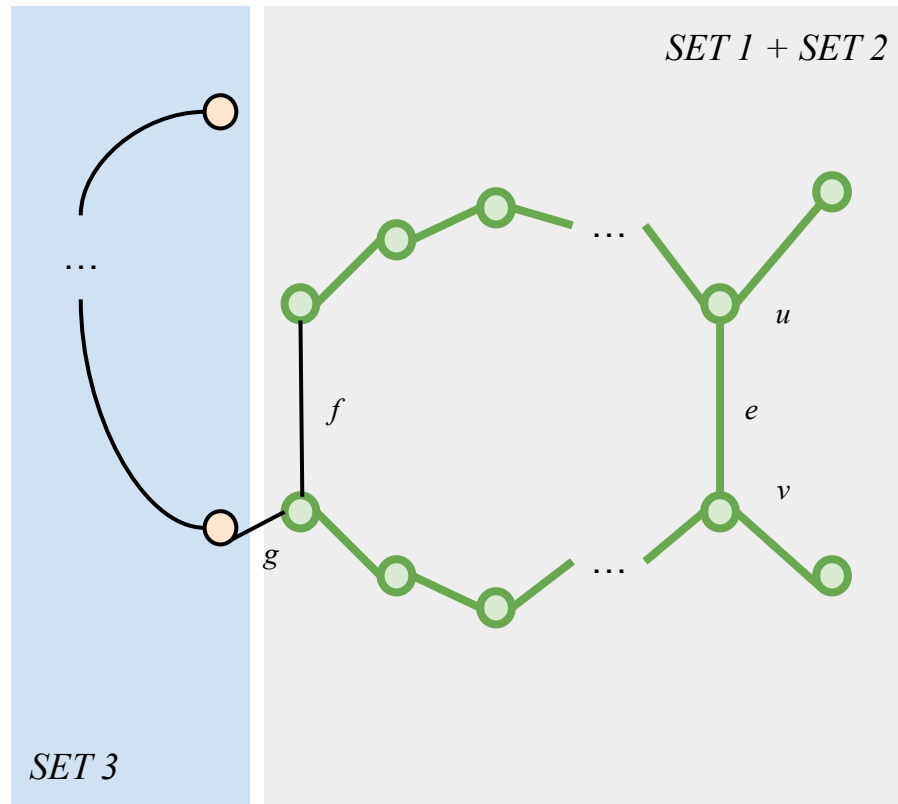
Defino  $S'$  tq  $u \in S'$ , y  $V-S'$  tq  $v \in V-S'$

$S = S \cup \{v\}$

$T_S$  es  $AGM$  de  $S$  por propiedad golosa (demo goloso)

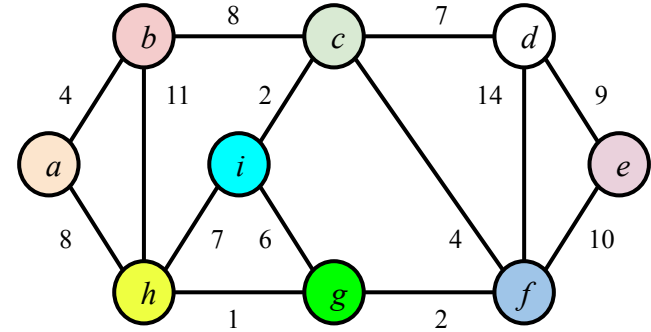
Si  $T_{SET-1} = (V_{SET-1}, E_{SET-1})$  era  $AGM$  de  $SET-1$  y  $T_{SET-2} = (V_{SET-2}, E_{SET-2})$  era  $AGM$  de  $SET-2 \Rightarrow$

$T = (V_{SET-1} + V_{SET-2}, E_{SET-1} + \{e\} + E_{SET-2})$  era  $AGM$



# Kruskal (1956; versión CLRS cap. 21)

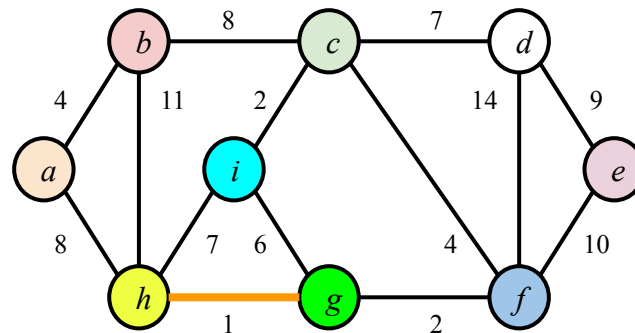
```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```





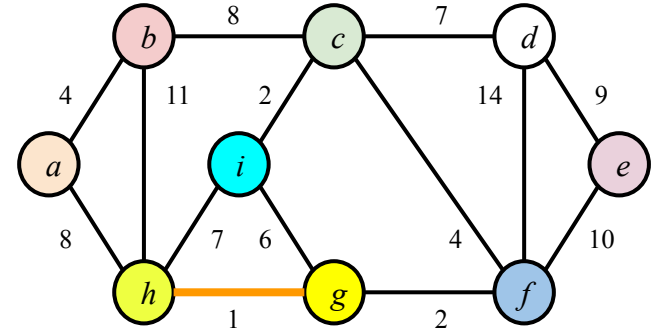
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



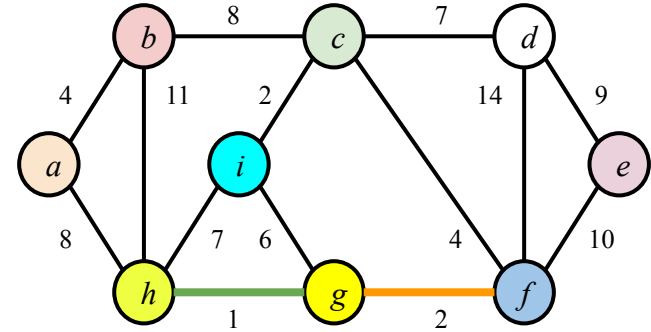
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



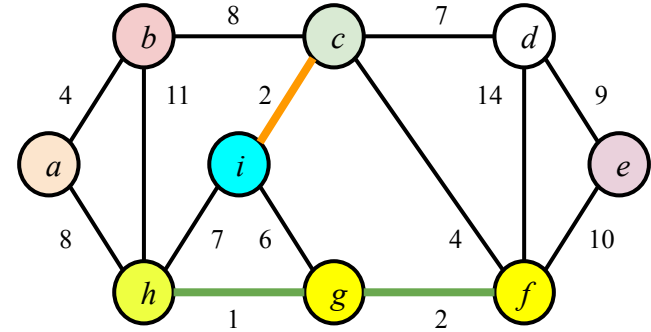
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



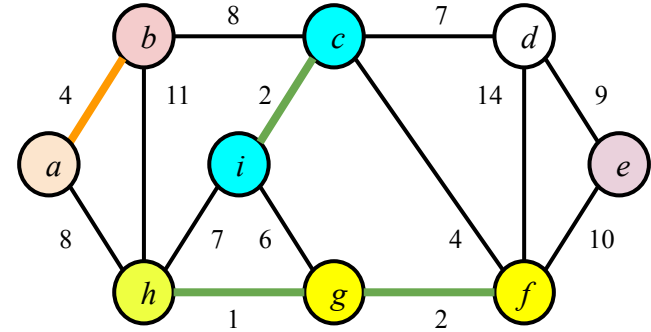
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



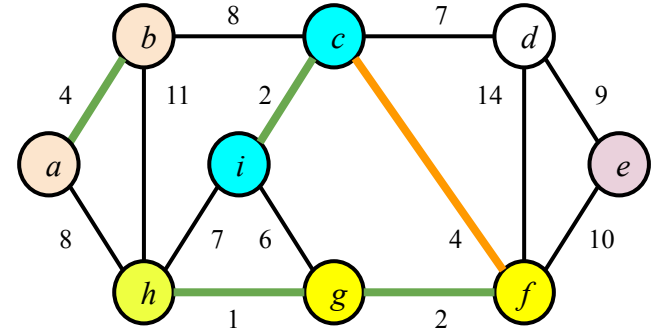
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



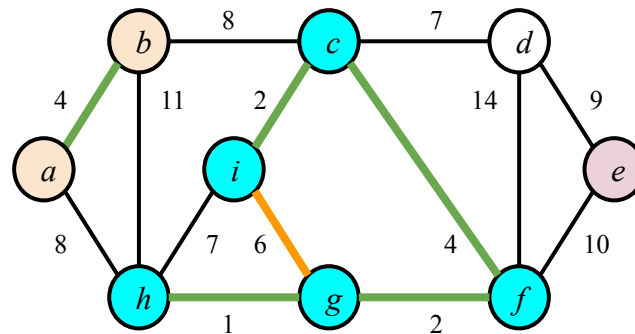
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



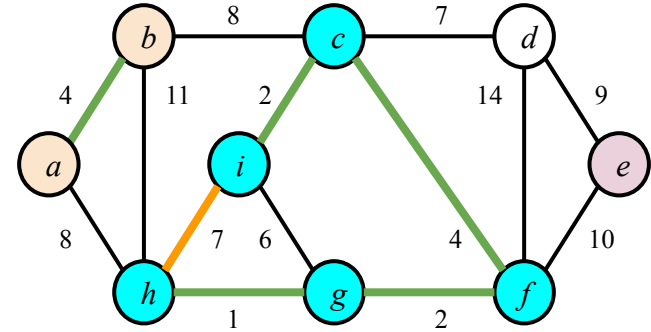
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



# Kruskal (1956; versión CLRS cap. 21)

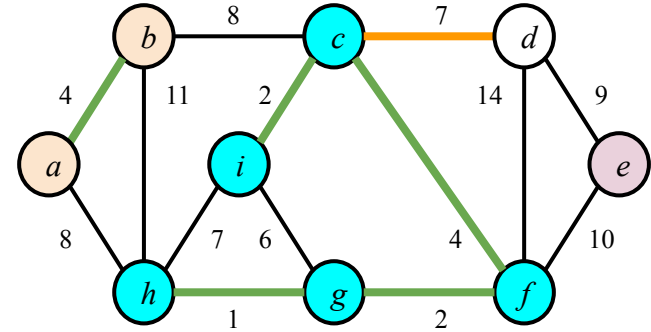
```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```





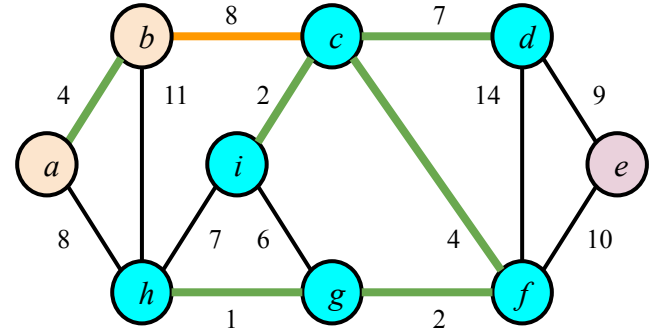
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



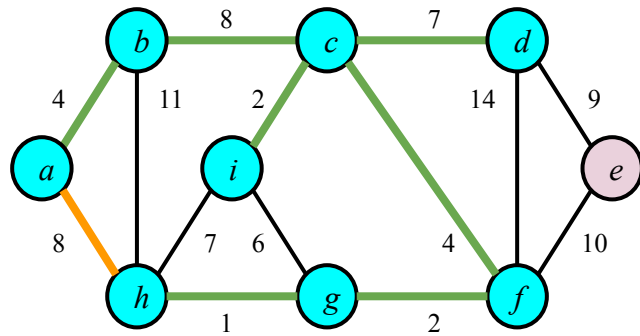
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



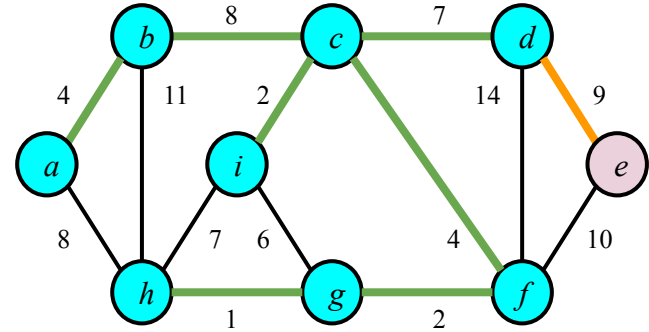
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



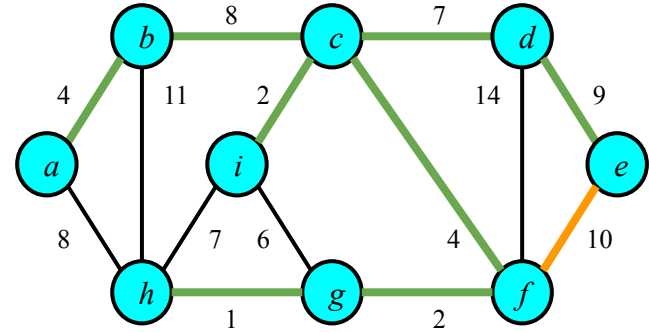
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



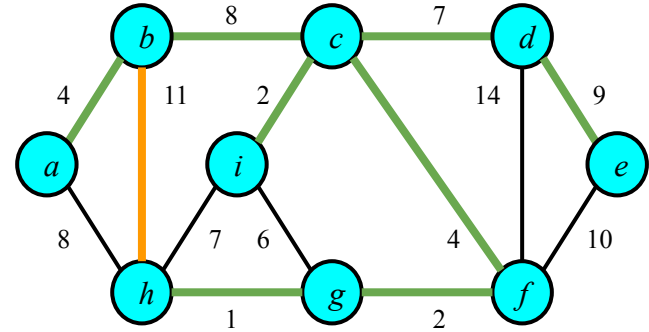
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



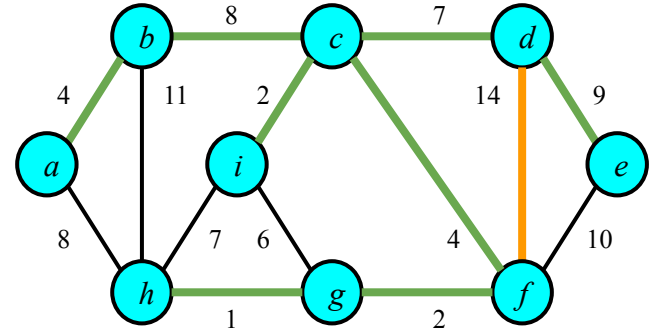
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



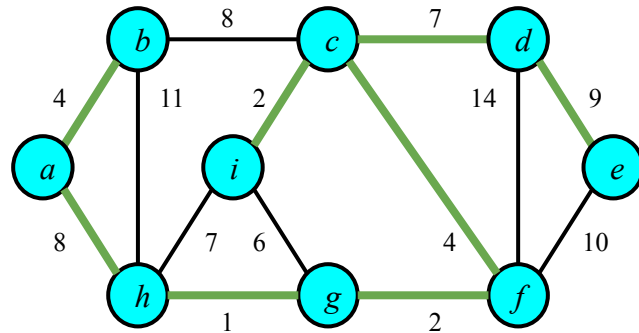
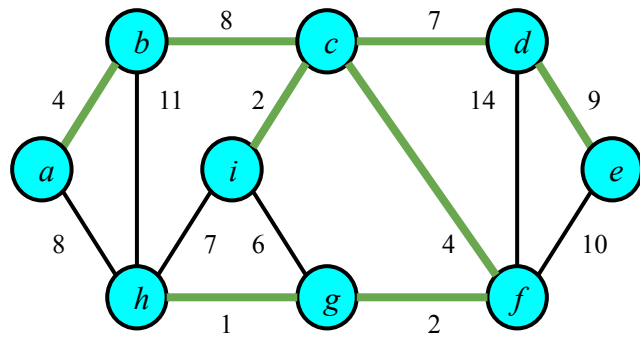
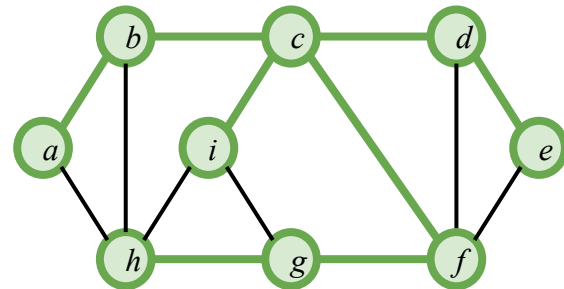
# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  
|   ORDENAR E de menor a mayor por w(e)  
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)
```



# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
  A =  $\emptyset$   
  for u in G.V:  
    | MAKE-SET( u )  
  ORDENAR E de menor a mayor por w(e)  
  for e in G.E: # en orden de w  
    | if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
    |   | A = A  $\cup$  {e}  
    |   | UNION(u,v)
```





# Kruskal (1956; versión CLRS cap. 21)

```
KRUSKAL ( G ) :  
|   A =  $\emptyset$   
|   for u in G.V:  
|       MAKE-SET( u )  $O(1)$   
|   ORDENAR E de menor a mayor por w(e)  $O(\text{sort}(E))$   
|   for e in G.E: # en orden de w  
|       if FIND-SET(u)  $\neq$  FIND-SET(v): # para e=(u,v)  
|           A = A  $\cup$  {e}  
|           UNION(u,v)  $O(\alpha(V))$ 
```

$O(V)$

$O(E * \alpha(V))$

$$O(V + \text{sort}(E) + E * \alpha(V)) \sim O(\text{sort}(E) + E * \log(V))$$

## Prim (1957; versión CLRS cap. 16, 21)

*Grafos raros  $E \sim V$ : Min-Heap  $\sim$  Fibonacci-Heap:  $O(V \log V)$*

*Grafos densos  $E \sim V^2$  Fibonacci-Heap:  $O(V^2)$ , Min-Heap  $O(V^2 \log V)$*

## Kruskal (1956; versión CLRS cap. 16, 19, 21)

*Grafos raros  $E \sim V$ :  $O(V \log V)$*

*Grafos densos  $E \sim V^2$ :  $O(V^2 \log V)$*