

# Árboles Generadores Mínimos

Técnicas de Diseño de Algoritmos (Ex Algoritmos y Estructuras de Datos III)

Segundo cuatrimestre 2024

# Repaso: Árboles generadores

## Definición:

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

## Teorema:

- ▶ Todo grafo conexo tiene (al menos) un árbol generador.
- ▶  $G$  conexo.  $G$  tiene un único árbol generador  $\iff G$  es árbol.
- ▶ Sea  $T = (V, X_T)$  un AG de  $G = (V, X)$  y  $e \in X \setminus X_T$ . Entonces  $T' = T + e - f = (V, X_T \cup \{e\} \setminus \{f\})$ , con  $f$  una arista del único circuito de  $T + e$ ,  $T'$  es árbol generador de  $G$ .

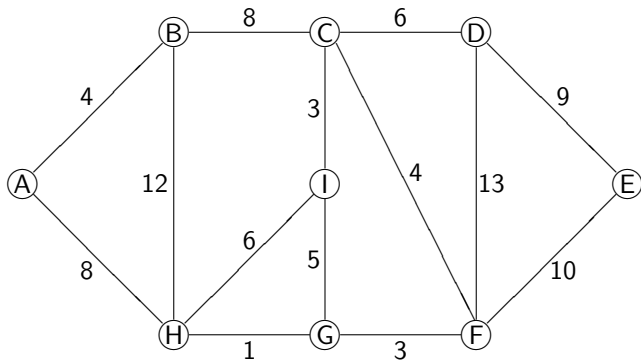
# Árbol generador mínimo

## Definiciones:

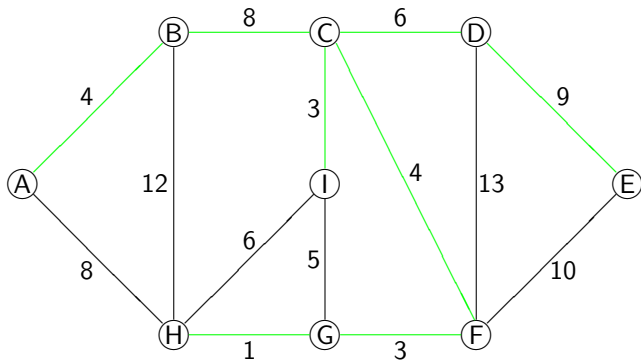
- ▶ Sea  $T = (V, X)$  un árbol y  $l : X \rightarrow R$  una función que asigna longitudes (o pesos) a las aristas de  $T$ . Se define la **longitud** de  $T$  como  $l(T) = \sum_{e \in T} l(e)$ .
- ▶ Dado un grafo conexo  $G = (V, X)$  con una función  $l$  que asigna longitudes (o pesos) a sus aristas, un **árbol generador mínimo** de  $G$ ,  $T$ , es un árbol generador de  $G$  de mínima longitud, es decir

$$l(T) \leq l(T') \quad \forall T' \text{ árbol generador de } G.$$

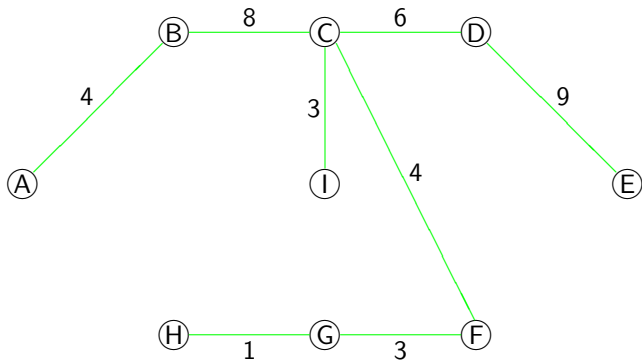
## Ejemplo de AGM



## Ejemplo de AGM



## Ejemplo de AGM



## Algoritmo de *Prim*

Entrada:  $G = (V, X)$  grafo conexo con una función  $l : X \rightarrow R$ .

$V_T := \{u\}$  ( $u$  cualquier nodo de  $G$ )

$X_T := \emptyset$

$i := 1$

**mientras**  $i \leq n - 1$  **hacer**

    elegir  $e = (u, v) \in X$  tal que  $l(e)$  sea mínima  
    entre las aristas que tienen un extremo

$u \in V_T$  y el otro  $v \in V \setminus V_T$

$X_T := X_T \cup \{e\}$

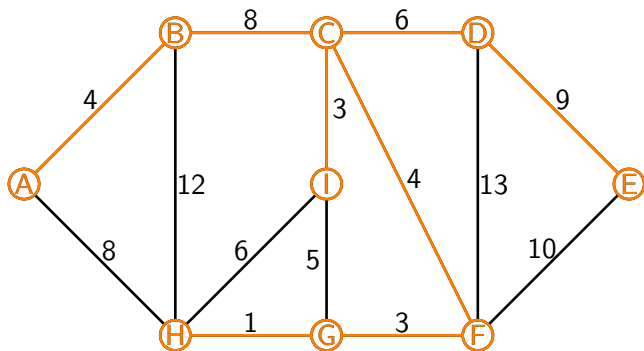
$V_T := V_T \cup \{v\}$

$i := i + 1$

**retornar**  $T = (V_T, X_T)$

## Ejemplo de AGM - Algoritmo de *Prim*

Iteración  $k$ : Subgrafo de un AGM con  $k$  aristas





# Algoritmo de *Prim*

## Lema:

Sea  $T = (V, X_T)$  un árbol generador de  $G = (V, X)$ . Si  $e \in X$  y  $e \notin X_T$  y  $f \in X_T$  una arista del ciclo de  $T + e$ . Entonces  $T' = (V, X_T \cup \{e\} \setminus \{f\})$  es árbol generador de  $G$ .

## Proposición:

Sea  $G$  un grafo conexo. Sea  $T_k = (V_{T_k}, X_{T_k})$  el árbol que el algoritmo de *Prim* determina en la iteración  $k$ , para  $0 \leq k \leq n - 1$ .  $T_k$  es un subárbol de un árbol generador mínimo de  $G$ .

## Teorema:

El algoritmo de *Prim* es correcto, es decir dado un grafo  $G$  conexo determina un árbol generador mínimo de  $G$ .

El algoritmo *Prim* es un algoritmo goloso.

## Prueba del lema

Claramente,  $T$  tiene exactamente  $|V| - 1$  aristas porque es árbol y  $T'$  tiene la misma cantidad de aristas que  $T$  ya que las aristas de  $T'$  son las mismas de  $T$  salvo  $f$  (arista de  $T$ ) que fue sustituida por  $e$  (no es arista de  $T$ ).

$T'$  no tiene ciclos porque  $T$  no los tiene, con el agregado de la arista  $e$  se forma un único ciclo en  $T + e$  y luego se quita una arista  $f$  de dicho ciclo. Consecuentemente, en  $T'$  ya no queda ningún ciclo.

$T'$  tiene exactamente  $|V| - 1$  aristas y no tiene ciclos entonces  $T'$  es árbol. Como  $T$  tiene todos los vértices de  $G$ ,  $T'$  es AG de  $G$ .

# Prueba de la proposición

Sea  $e_i$  la arista elegida en la iteración  $i$ -ésima de la ejecución del algoritmo de *Prim* para  $1 \leq i \leq n - 1$ .

Probaremos por inducción en  $k$  que  $T_k$  es un subárbol de un generador mínimo de  $G$ .

- Caso base,  $k = 0$ . Claramente,  $T_0$  (árbol de un solo nodo) es un subárbol de cualquier árbol generador de  $G$  y particularmente de cualquier AGM  $T$  de  $G$  ya que todo subgrafo generador de  $G$  debe tener a todos los nodos de  $G$ .

## Prueba de la proposición: continuación

- Es cierta la proposición para  $T_{k-1}$ , queremos ver que también es cierta para  $T_k$ . Supongamos que no, entonces existe un AGM  $T$  tal que  $T_{k-1}$  es un subárbol suyo y  $T_k$  no lo es. Por lo tanto,  $e_1, \dots, e_{k-1}$  son aristas de  $T$  y  $e_k = (a, b)$  no es arista de  $T$ . Podemos suponer sin pérdida de generalidad que  $a \in V_{T_{k-1}}$  y  $b \notin V_{T_{k-1}}$ . Sea  $P_{ab}$  el camino de  $T$  que une  $a$  con  $b$  y  $C = P_{ab} \cup \{e_k\}$  sería el único ciclo de  $T + e_k$ . Existe una arista  $f = (c, d) \in P_{ab}$  tal que  $c \in V_{T_{k-1}}$  y  $d \notin V_{T_{k-1}}$  ya que el camino  $P_{ab}$  comienza en  $a$  (un nodo de  $V_{T_{k-1}}$ ) y termina en  $b$  (un nodo que no está en  $V_{T_{k-1}}$ ). Claramente,  $l(f) \geq l(e_k)$  ya que ambas aristas estaban disponibles para competir en la iteración  $k$  de la ejecución del algoritmo y  $e_k$  fue la arista elegida de esa iteración. Es más,  $l(f) = l(e_k)$ , porque si  $l(f) > l(e_k)$  entonces  $l(T) > l(T' = T \cup \{e_k\} \setminus \{f\})$  y  $T$  no sería mínimo, lo cual es una contradicción. Por lo tanto,  $T' = T \cup \{e_k\} \setminus \{f\}$  es un AGM que tiene a  $T_k$  como subárbol.

# Complejidad del Algoritmo de *Prim*

- ▶  $O(n^2)$ , implementación estándar.
- ▶  $O((m + n) \log n)$ , usando heap binario.
- ▶  $O(m + n \log n)$ , usando heap Fibonacci.

## Algoritmo de *Kruskal*

Entrada:  $G = (V, X)$  grafo conexo con una función  $l : X \rightarrow R$ .

$X_T := \emptyset$

$i := 1$

**mientras**  $i \leq n - 1$  **hacer**

    elegir  $e \in X$  tal que  $l(e)$  sea mínima entre las  
    aristas que no forman circuito con las  
    aristas que ya están en  $X_T$

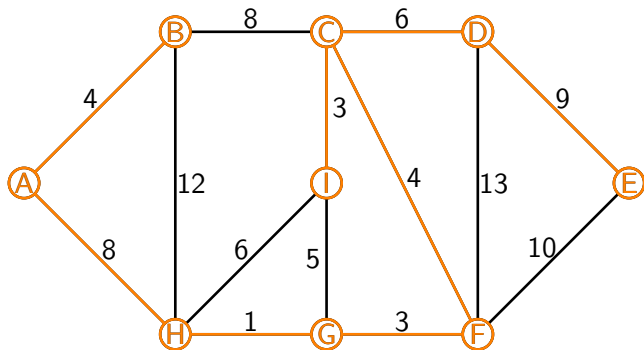
$X_T := X_T \cup \{e\}$

$i := i + 1$

**retornar**  $T = (V, X_T)$

## Ejemplo de AGM - Algoritmo de *Kruskal*

Iteración  $k$ : Subgrafo de un AGM con  $k$  aristas sin ciclos



# Algoritmo de *Kruskal*

## Proposición:

Sea  $G$  un grafo conexo. Sea  $B_k = (V, X_{T_k})$  el bosque que el algoritmo de *Kruskal* genera en algún momento con exactamente  $k$  aristas,  $0 \leq k \leq n - 1$ .  $B_k$  es un subgrafo generador sin ciclos de un árbol generador mínimo de  $G$ .

## Teorema:

El algoritmo de *Kruskal* es correcto, es decir dado un grafo  $G$  conexo determina un árbol generador mínimo de  $G$ .

El algoritmo *Kruskal* es un algoritmo goloso.



# Prueba de la proposición

Sean  $e_1, \dots, e_m$  las aristas ordenadas de menor a mayor de acuerdo a la función  $l$  ( $l(e_1) \leq \dots \leq l(e_m)$ ) y  $e'_1 = e_{j_1}, \dots, e'_{n-1} = e_{j_{n-1}}$ , las  $n - 1$  aristas agregadas durante la ejecución del algoritmo de *Kruskal* siguiendo este orden de aristas, es decir que  $1 = j_1 < \dots < j_{n-1}$ . Claramente, por la forma de elegir las aristas del algoritmo,  $B_k = (V, X_{T_k} = \{e'_1, \dots, e'_k\})$  no contiene ciclos. A continuación, vamos a probar por inducción en  $k$  que  $B_k$  es subgrafo generador de algún AGM  $T$ .

- Caso base,  $k = 0$ . Claramente,  $B_0$  es un subgrafo generador sin aristas y sin ciclos de cualquier árbol generador y particularmente de cualquier AGM  $T$ .

## Prueba de la proposición: continuación

- Es cierta la proposición para  $B_{k-1}$ , queremos ver que también es cierta para  $B_k$ . Supongamos que no, entonces existe un AGM  $T$  tal que  $B_{k-1}$  es un subgrafo generador suyo y  $B_k$  no lo es. Por lo tanto,  $e'_1 = e_{j_1}, \dots, e'_{k-1} = e_{j_{k-1}}$  son aristas de  $T$  y  $e'_k = e_{j_k} = (a, b)$  no es arista de  $T$ . Sea  $P_{ab}$  el camino de  $T$  que une  $a$  con  $b$  y  $C = P_{ab} \cup \{e'_k\}$  sería el único ciclo de  $T + e'_k$ . Cualquier arista  $e_p \in P_{ab}$ ,  $l(e_p) \leq l(e'_k)$ , caso contrario  $l(T' = T \cup \{e'_k\} \setminus \{e_p\}) < l(T)$  y como  $T'$  es AG entonces  $T$  no sería mínimo, contradicción. Hay una arista  $e_p \in P_{ab}$  tal que  $e_p$  no está en  $B_k$  sino  $C$  es un ciclo de  $B_k$  y sería otra contradicción. Si  $p < j_k$ ,  $e_p$  formaba un ciclo con las aristas de  $B_{k-1}$  ya que fue descartada por el algoritmo y ese ciclo estaría en  $T$  y es contradicción. Por lo tanto,  $p > j_k$  y  $l(e_p) \geq l(e'_k)$ . Consecuentemente,  $l(e_p) = l(e'_k)$  y  $T' = T \cup \{e'_k\} \setminus \{e_p\}$  es un AGM que tiene a  $B_k$  como subgrafo generador.

# Complejidad del Algoritmo de *Kruskal*

- ▶  $O(m * n)$ , implementación trivial.
- ▶  $O(m \log n + m \log n)$ , union and find (por rango).
- ▶  $O(m \log n + m * \alpha(n))$ , union and find (por rango + compresión de camino).