

Árboles

Algoritmos y Estructuras de Datos III

Árboles

Definición:

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

Árboles

Teorema: Dado un grafo $G = (V, X)$ son equivalentes:

1. G es un árbol.
2. G es un grafo sin circuitos simples, pero si se agrega una arista e a G resulta un grafo con exactamente un circuito simple, y ese circuito contiene a e .
3. Existe exactamente un camino simple entre todo par de nodos.
4. G es conexo, pero si se quita cualquier arista a G queda un grafo no conexo.

Lema 1: Sea $G = (V, X)$ un grafo conexo y $e \in X$. $G - e$ es conexo si y solo si e pertenece a un circuito simple de G .

Árboles

Definición:

- ▶ Una **hoja** es un nodo de grado 1.

Árboles

Definición:

- Una **hoja** es un nodo de grado 1.

Lema 2: Todo árbol no trivial tiene al menos dos hojas.

Lema 3: Sea $G = (V, X)$ árbol. Entonces $m = n - 1$.

Corolario 1: Sea $G = (V, X)$ sin circuitos simples y c componentes conexas. Entonces $m = n - c$.

Corolario 2: Sea $G = (V, X)$ con c componentes conexas. Entonces $m \geq n - c$.

Árboles

Teorema: Dado un grafo G son equivalentes:

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

Árboles orientados

Definiciones:

- ▶ Un **árbol orientado** es un grafo orientado G tal que:
 1. El grafo no orientado subyacente es un árbol.
 2. En G existe un nodo r tal que existe un camino orientado desde r a todos los demás nodos (cualquier nodo es alcanzable desde r por un camino orientado).

Árboles enraizados

Definiciones:

- ▶ En un árbol no orientado podemos definir un nodo cualquiera como raíz.
- ▶ El **nivel** de un nodo de un árbol es la distancia de ese nodo a la raíz.
- ▶ La **altura** h de un árbol es la longitud desde la raíz al nodo más lejano.
- ▶ Un árbol se dice (exactamente) **m -ario** (con $m \geq 2$) si todos sus nodos, salvo las hojas y la raíz tienen grado (exactamente) a lo sumo $m + 1$ y la raíz (exactamente) a lo sumo m .
- ▶ Un árbol se dice **balanceado** si todas sus hojas están a nivel h o $h - 1$.
- ▶ Un árbol se dice **balanceado completo** si todas sus hojas están a nivel h .

Árboles enraizados

Definiciones:

- ▶ Los nodos **internos** de un árbol son aquellos que no son ni hojas ni la raíz.

Árboles enraizados

Definiciones:

- Los nodos **internos** de un árbol son aquellos que no son ni hojas ni la raíz.

¿Cuántos nodos tiene en total un árbol exactamente m -ario que tiene i nodos internos?

Árboles enraizados

Teorema:

- ▶ Un árbol m -ario de altura h tiene a lo sumo m^h hojas. Alcanza esta cota si es un árbol exactamente m -ario balanceado completo con $h \geq 1$.
- ▶ Un árbol m -ario con l hojas tiene $h \geq \lceil \log_m l \rceil$.
- ▶ Si T es un árbol exactamente m -ario balanceado no trivial entonces $h = \lceil \log_m l \rceil$.

Recorrido de árboles

- ▶ **BFS** (Breadth-First Search): Lista implementada como cola.
- ▶ **DFS** (Depth-First Search): Lista implementada como pila.

Algoritmo *Recorrer* (para grafos y árboles no orientados)

```
Para todo nodo  $i$ ,  $\text{pred}(i) := \infty$ ,  $\text{order}(i) := \infty$   
 $\text{next} := 0$   
 $LIST := \{(s, 0)\}$   
mientras  $LIST \neq \emptyset$  hacer  
    sacar un elemento  $(i, \text{padre})$  de  $LIST$   
    si  $\text{order}(i) = \infty$  entonces  
        (marcar nodo  $i$ )  
         $\text{pred}(i) := \text{padre}$   
         $\text{next} := \text{next} + 1$   
         $\text{order}(i) := \text{next}$   
        Para cada arista  $(i, j)$  hacer  
             $LIST := LIST \cup \{(j, i)\}$   
retornar  $\text{order}$  y  $\text{pred}$ 
```

Algoritmo *Recorrer* (para grafos y árboles no orientados)

Para todo nodo i , $\text{pred}(i) := \infty$, $\text{order}(i) := \infty$

$\text{next} := 0$

$LIST := \{(s, 0)\}$

mientras $LIST \neq \emptyset$ **hacer**

 sacar un elemento (i, padre) de $LIST$

si $\text{order}(i) = \infty$ **entonces**

 (marcar nodo i)

$\text{pred}(i) := \text{padre}$

$\text{next} := \text{next} + 1$

$\text{order}(i) := \text{next}$

 Para cada arista (i, j) con $j \neq \text{padre}$ hacer

$LIST := LIST \cup \{(j, i)\}$

retornar order y pred

Algoritmo *Recorrer* (para grafos y árboles no orientados)

```
Para todo nodo  $i$ ,  $\text{pred}(i) := \infty$ ,  $\text{order}(i) := \infty$   
 $\text{next} := 0$   
 $\text{hayciclo} := \text{false}$   
 $LIST := \{(s, 0)\}$   
mientras  $LIST \neq \emptyset$  hacer  
    sacar un elemento  $(i, \text{padre})$  de  $LIST$   
    si  $\text{order}(i) = \infty$  entonces  
        (marcar nodo  $i$ )  
         $\text{pred}(i) := \text{padre}$   
         $\text{next} := \text{next} + 1$   
         $\text{order}(i) := \text{next}$   
        Para cada arista  $(i, j)$  con  $j \neq \text{padre}$  hacer  
             $LIST := LIST \cup \{(j, i)\}$   
        sino  $\text{hayciclo} := \text{true}$ ,  $\text{semillaciclo} := (i, \text{padre})$   
retornar  $\text{order}$ ,  $\text{pred}$ ,  $\text{hayciclo}$ ,  $\text{semillaciclo}$ 
```

Algoritmo *Recorrer* (para grafos y árboles orientados)

```
Para todo nodo  $i$ ,  $\text{pred}(i) := \infty$ ,  $\text{order}(i) := \infty$   
 $\text{next} := 0$   
 $LIST := \{(s, 0)\}$   
mientras  $LIST \neq \emptyset$  hacer  
    sacar un elemento  $(i, \text{padre})$  de  $LIST$   
    si  $\text{order}(i) = \infty$  entonces  
        (marcar nodo  $i$ )  
         $\text{pred}(i) := \text{padre}$   
         $\text{next} := \text{next} + 1$   
         $\text{order}(i) := \text{next}$   
        Para cada arista  $i \rightarrow j$  hacer  
             $LIST := LIST \cup \{(j, i)\}$   
retornar  $\text{order}$  y  $\text{pred}$ 
```

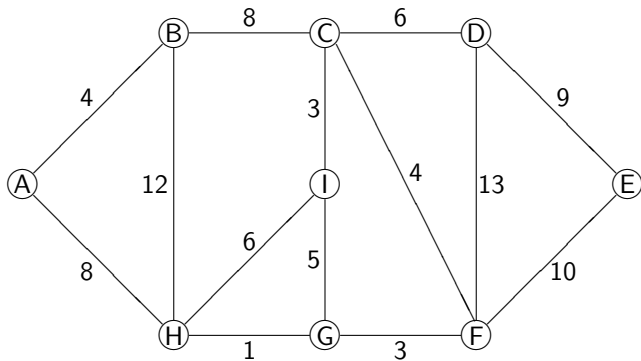

Árbol generador mínimo

Definiciones:

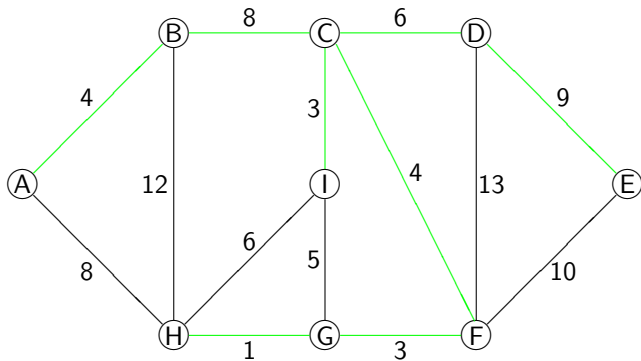
- ▶ Dado un grafo conexo G , un **árbol generador** de G es un subgrafo de G que es un árbol y tiene el mismo conjunto de nodos que G .
- ▶ Sea $T = (V, X)$ un árbol y $l: X \rightarrow R$ una función que asigna longitudes (o pesos) a las aristas de T . Se define la **longitud** de T como $l(T) = \sum_{e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

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

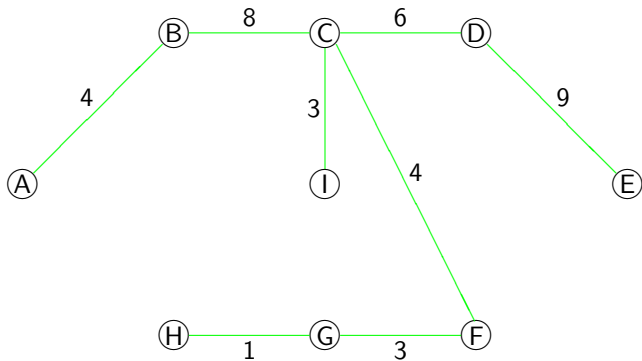
Ejemplo de AGM



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 $T = (V_T, X_T)$

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 .

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.

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

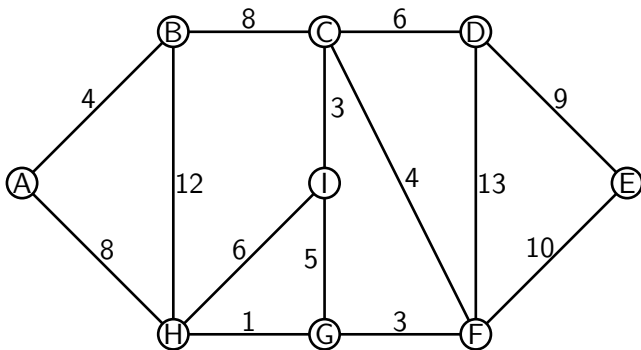
$X_T := X_T \cup \{e\}$

$i := i + 1$

retornar $T = (V, X_T)$

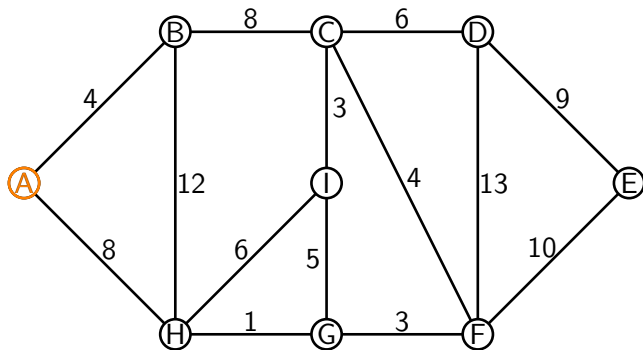
Ejemplo de AGM - Algoritmo de *Prim*

Iteración k : Subgrafo de un AGM con k aristas



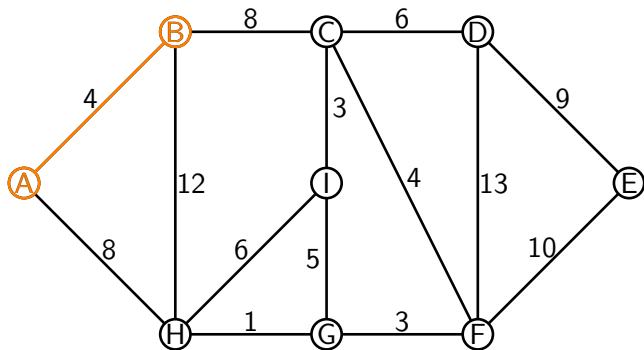
Ejemplo de AGM - Algoritmo de *Prim*

Iteración k : Subgrafo de un AGM conexo con k aristas



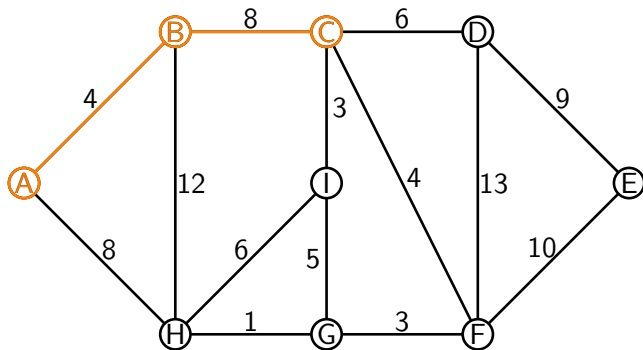
Ejemplo de AGM - Algoritmo de *Prim*

Iteración k : Subgrafo de un AGM con k aristas



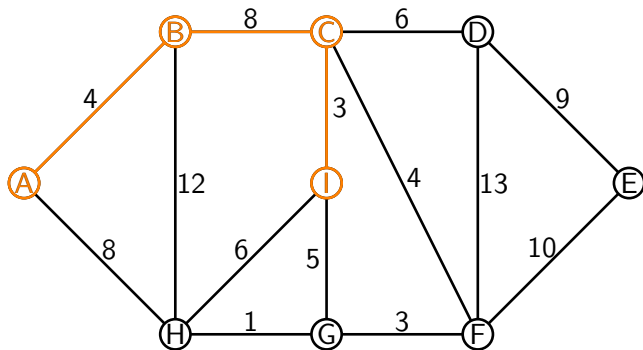
Ejemplo de AGM - Algoritmo de *Prim*

Iteración k : Subgrafo de un AGM conexo con k aristas



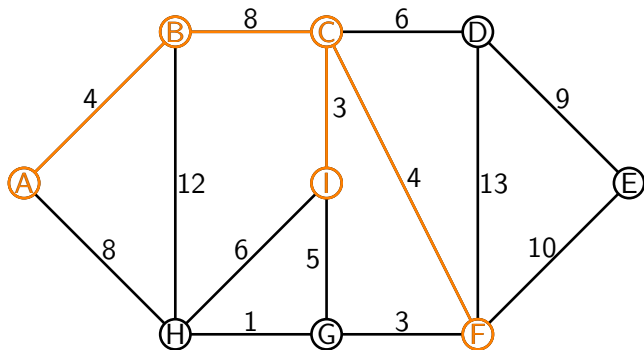
Ejemplo de AGM - Algoritmo de *Prim*

Iteración k : Subgrafo de un AGM conexo con k aristas



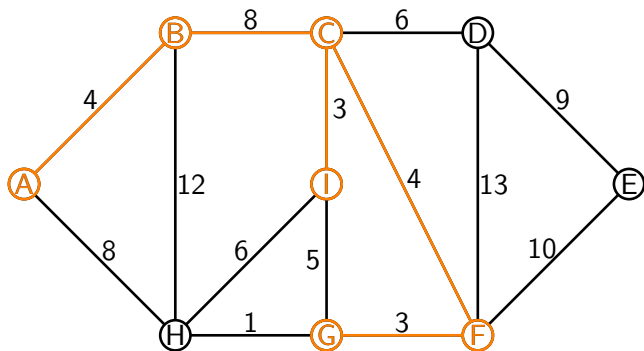
Ejemplo de AGM - Algoritmo de *Prim*

Iteración k : Subgrafo de un AGM con k aristas



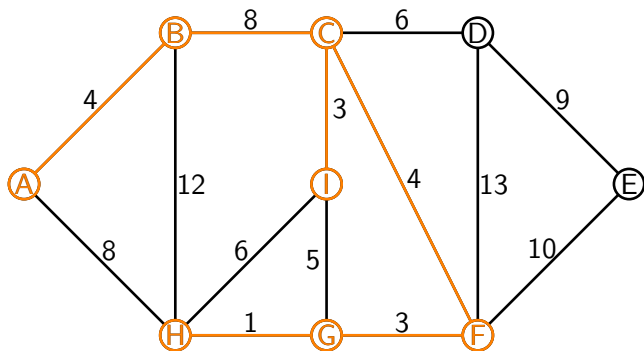
Ejemplo de AGM - Algoritmo de *Prim*

Iteración k : Subgrafo de un AGM con k aristas



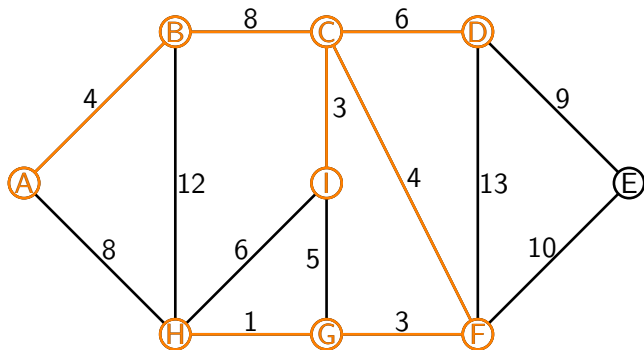
Ejemplo de AGM - Algoritmo de *Prim*

Iteración k : Subgrafo de un AGM con k aristas



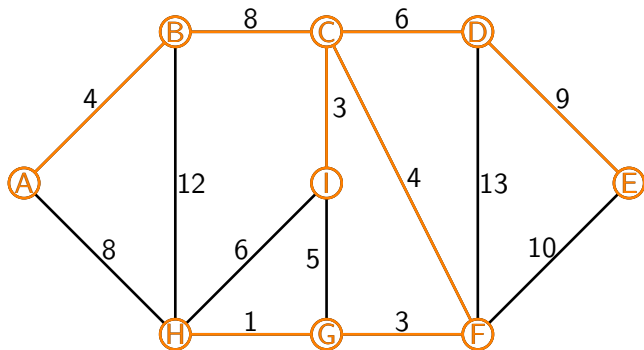
Ejemplo de AGM - Algoritmo de *Prim*

Iteración k : Subgrafo de un AGM con k aristas



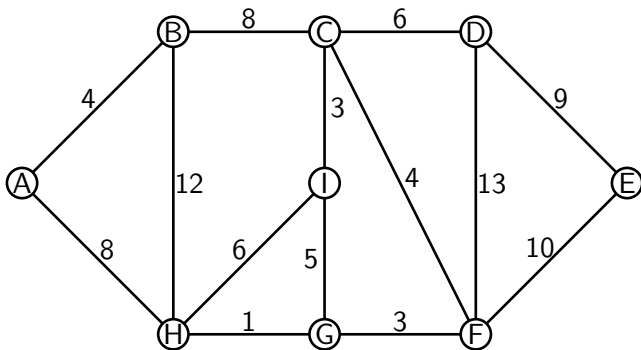
Ejemplo de AGM - Algoritmo de *Prim*

Iteración k : Subgrafo de un AGM con k aristas



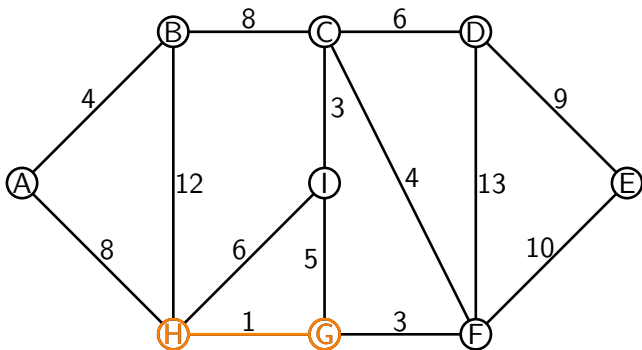
Ejemplo de AGM - Algoritmo de *Kruskal*

Iteración k : Subgrafo de un AGM con k aristas sin circuitos



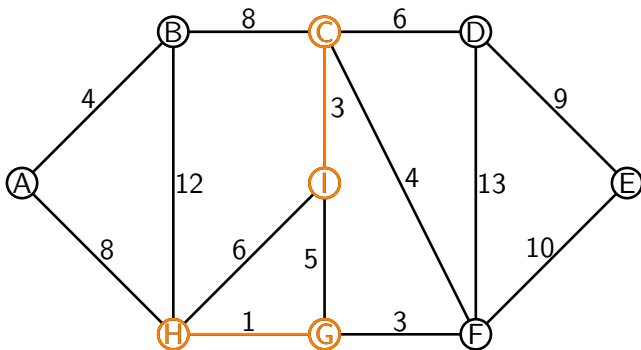
Ejemplo de AGM - Algoritmo de *Kruskal*

Iteración k : Subgrafo de un AGM con k aristas sin circuitos



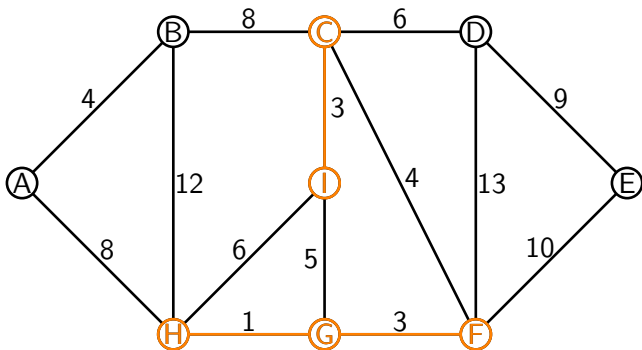
Ejemplo de AGM - Algoritmo de *Kruskal*

Iteración k : Subgrafo de un AGM con k aristas sin circuitos



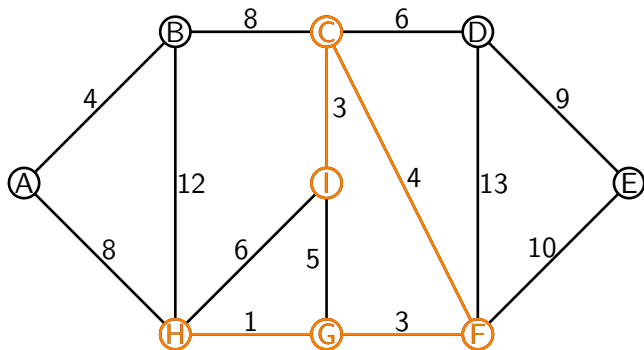
Ejemplo de AGM - Algoritmo de *Kruskal*

Iteración k : Subgrafo de un AGM con k aristas sin circuitos



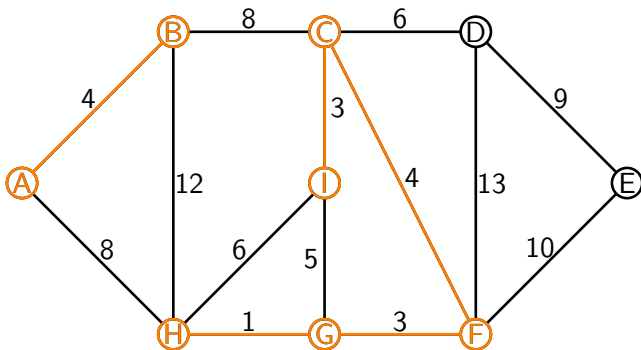
Ejemplo de AGM - Algoritmo de *Kruskal*

Iteración k : Subgrafo de un AGM con k aristas sin circuitos



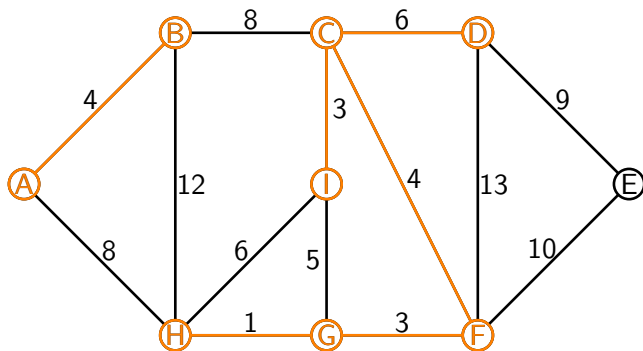
Ejemplo de AGM - Algoritmo de *Kruskal*

Iteración k : Subgrafo de un AGM con k aristas sin circuitos



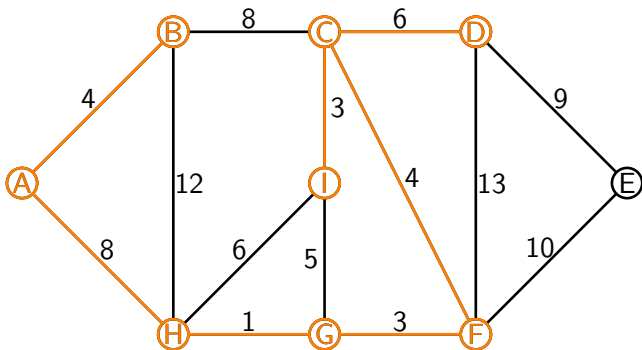
Ejemplo de AGM - Algoritmo de *Kruskal*

Iteración k : Subgrafo de un AGM con k aristas sin circuitos



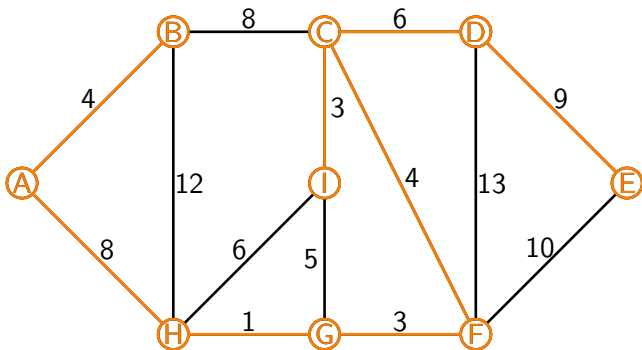
Ejemplo de AGM - Algoritmo de *Kruskal*

Iteración k : Subgrafo de un AGM con k aristas sin circuitos



Ejemplo de AGM - Algoritmo de *Kruskal*

Iteración k : Subgrafo de un AGM con k aristas sin circuitos



Enumeración

- ▶ ¿Cuántos grafos diferentes hay que tengan ciertas características?

Enumeración

- ▶ ¿Cuántos grafos diferentes hay que tengan ciertas características?

¿Cuántos grafos conexos con 8 nodos y 2 circuitos simples?

Enumeración

- ▶ ¿Cuántos grafos diferentes hay que tengan ciertas características?

¿Cuántos grafos conexos con 8 nodos y 2 circuitos simples?

- ▶ ¿Cuántos subgrafos de un cierto tipo de un grafo dado hay?

Enumeración

- ▶ ¿Cuántos grafos diferentes hay que tengan ciertas características?

¿Cuántos grafos conexos con 8 nodos y 2 circuitos simples?

- ▶ ¿Cuántos subgrafos de un cierto tipo de un grafo dado hay?

¿Cuántos árboles generadores tiene un grafo?

¿Cuántos circuitos simples?

¿Cuántos subgrafos completos maximales?

Enumeración

- ▶ ¿Cuántos grafos diferentes hay que tengan ciertas características?

¿Cuántos grafos conexos con 8 nodos y 2 circuitos simples?

- ▶ ¿Cuántos subgrafos de un cierto tipo de un grafo dado hay?

¿Cuántos árboles generadores tiene un grafo?

¿Cuántos circuitos simples?

¿Cuántos subgrafos completos maximales?

En general son problemas difíciles.

Ejemplos de algunas cosas que se *pueden* calcular

¿Cuántos grafos **rotulados** distintos de n nodos y m aristas hay?

¿Cuántos grafos **rotulados** distintos de n nodos hay?

Teorema de Cayley (1875)

El número de árboles rotulados distintos de n nodos es n^{n-2} , para $n \geq 2$.

Demostración:

Por inducción en n usando los algoritmos inversos de codificación y decodificación de *Prüfer* para establecer una biyección entre el conjunto de árboles rotulados de n nodos y el conjunto de palabras de $n - 2$ símbolos usando el alfabeto $\{1, \dots, n\}$.

Algoritmo de codificación de *Prüfer*

Dado un árbol de n nodos rotulado con los números de 1 a n , construye una *palabra* de longitud $n - 2$ usando el *alfabeto* $\{1, \dots, n\}$.

Entrada: Un árbol T rotulado de n nodos, $n \geq 2$.

```
para  $i = 1, \dots, n - 2$  hacer  
    elegir  $v$  la hoja de menor rótulo de  $T$   
    poner como  $s_i$  el rótulo del nodo adyacente a  $v$   
     $T := T \setminus \{v\}$   
     $i := i + 1$   
retornar palabra  $s_1 \dots s_{n-2}$ 
```

Algoritmo de codificación de *Prüfer*

Proposición:

Si d_k es el número de ocurrencias del número k en la palabra retornada por el algoritmo de codificación de *Prüfer*, entonces el grado del nodo con rótulo k en el árbol T es $d_k + 1$.

Algoritmo de decodificación de *Prüfer*

Dada una *palabra* de longitud $n - 2$ usando el *alfabeto* $\{1, \dots, n\}$ construye un árbol de n nodos rotulado con los números de 1 a n .

Entrada: Una palabra $P = s_1 \dots s_{n-2}$.

$L := \{1, \dots, n\}$

$V_T := \{1, \dots, n\}$

$X_T := \emptyset$

$P := s_1 \dots s_{n-2}$

para $i = 1, \dots, n - 2$ **hacer**

 Sea k el menor número de L que no está en P

 Sea j el primer elemento de P

$X_T := X_T \cup \{(k, j)\}$

$L := L \setminus \{k\}$

$P := s_{i+1} \dots s_{n-2}$

$X_T := X_T \cup \{(u, v)\}$, con $u, v \in L$

retornar $T = (V_T, X_T)$

Algoritmo de decodificación de *Prüfer*

Proposición:

El algoritmo de decodificación de *Prüfer* define una función del conjunto de palabras de longitud $n - 2$ usando el alfabeto $\{1, \dots, n\}$ sobre el conjunto de árboles rotulados de n nodos.