

# Análisis y Diseño de Algoritmos II

*Jesús Alexander Aranda Ph.D   Robinson Duque, Ph.D  
Juan Francisco Díaz, Ph. D*

*Universidad del Valle*

[jesus.aranda@correounalvalle.edu.co](mailto:jesus.aranda@correounalvalle.edu.co)  
[robinson.duque@correounalvalle.edu.co](mailto:robinson.duque@correounalvalle.edu.co)  
[juanfco.diaz@correounalvalle.edu.co](mailto:juanfco.diaz@correounalvalle.edu.co)

*Programa de Ingeniería de Sistemas*

*Escuela de Ingeniería de Sistemas y Computación*



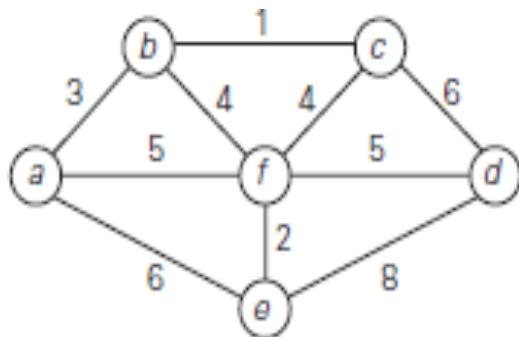
# Prog. Voraz- Árbol Recubrimiento Mínimo

Entrada : Un grafo conexo ponderado  $G = \langle V, E \rangle$

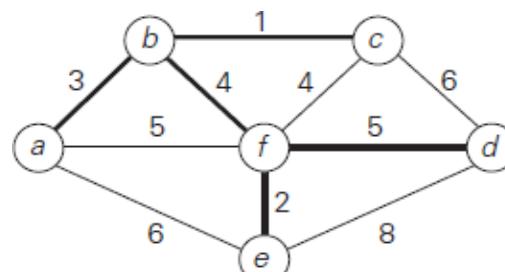
Salida : Un sub-grafo acíclico conexo de  $G$ ,  $G' = \langle V', E' \rangle$  tal que  $V = V'$  y la suma de los pesos de sus aristas ( $E'$ ) sea mínima. (Árbol de recubrimiento mínimo)

Ejemplo:

Entrada:



Salida:



# Prog. Voraz- Árbol Recubrimiento Mínimo

---

Estrategia Voraz:

Incrementalmente construir un árbol de recubrimiento mínimo

Propiedad de escogencia voraz:

Escoger la arista de menor peso que conecte el árbol parcial con los vértices que no hacen parte del árbol. Dicha arista con su respectivo vértice se agregan al árbol.

-- Se inicia con un árbol que tiene sólo un vértice hasta que incrementalmente se construya un árbol que tiene todos los vértices del grafo.

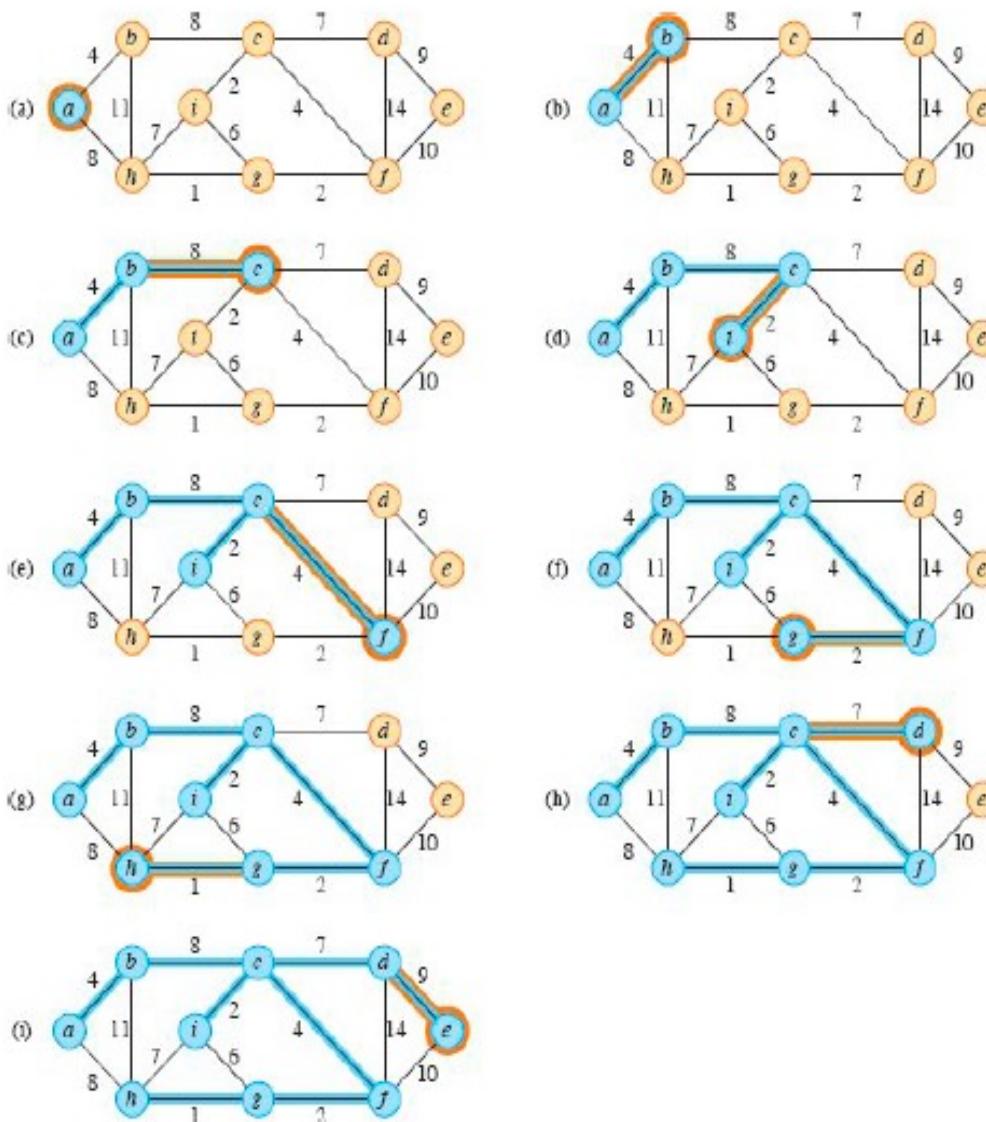
Invariante a lo largo de proceso: El árbol parcial en todo momento es parte de un árbol de recubrimiento mínimo.

Al algoritmo que implementa esta estrategia voraz se le conoce como el algoritmo de Prim.

# Prog. Voraz- Árbol Recubrimiento Mínimo

Estrategia Voraz:

Algoritmo PRIM



Tomado de [0]

# Prog. Voraz- Árbol Recubrimiento Mínimo

El algoritmo que plantea esta estrategia voraz se conoce como el algoritmo de Prim:

$Vt = \{VO\}$  // Se escoge arbitrariamente el vértice  $VO$  para ser parte de  $G'$

$E^+ = \{ \}$

for  $i = 1$  to  $|V| - 1$  do

-- Encuentre la arista de peso mínimo  $e^* = (v^*, u^*)$  entre todas las aristas  $(v, u)$  donde  $v \in V_t$  y  $u \in V - V_t$

$$V^t = V^t \cup \{u^*\}$$

$$E^* = E^* \cup \{e^*\}$$

Return (Vt,Et)

# Prog. Voraz- Árbol Recubrimiento Mínimo

---

El algoritmo de Prim siempre encuentra un árbol de recubrimiento mínimo.

Idea de la prueba :

Asumamos que hay un árbol de recubrimiento mínimo  $T$  diferente al arrojado por el algoritmo Prim,  $T'$ , por lo tanto durante el proceso de construcción del algoritmo va a existir una arista  $e^* = (v^*, u^*)$  con el mínimo peso entre todas las aristas que conectan los vértices de  $V_t$  y los vértices de  $V - V_t$  que es escogida por el algoritmo Prim para ser parte de  $G'$  y que no es parte del árbol  $T$ , por lo tanto si se agregara  $e^*$  a  $T$  se formaría un ciclo formado por las aristas de  $T$  y  $e^*$ .

Como  $e^*$  no es parte de  $T$  entonces debe existir alguna otra arista  $(v^{**}, u^{**})$  que conecte un vértice de  $V_t$  con un vértice de  $V - V_t$  y que sea parte de  $T$ .

Si eliminaramos  $(v^{**}, u^{**})$  del ciclo formado por las aristas de  $ARM$  y  $e^*$  aún tendríamos un árbol de recubrimiento (¿por qué?) pero con menor o igual peso por lo tanto  $T$  también debe ser un árbol de recubrimiento mínimo

# Prog. Voraz- Árbol Recubrimiento Mínimo

---

## Algoritmo de Prim

-- Algoritmo Prim (G)

$V_t = \{V_0\}$  //  $O(1)$

$E_t = \{ \}$  //  $O(1)$

for  $i = 1$  to  $|V| - 1$  do //

-- Encuentre la arista de peso mínimo  $e^* = (v^*, u^*)$  entre todas las aristas  $(v, u)$  donde  $v \in V_t$  y  $u \in V - V_t$  //  $O(|V|)^*$  ?

$V_t = V_t \cup \{u^*\}$  //  $O(|V|)^* O(1)$

$E_t = E_t \cup \{e^*\}$  //  $O(|V|)^* O(1)$

Return  $(V_t, E_t)$

# Prog. Voraz- Árbol Recubrimiento Mínimo

## Complejidad del algoritmo de Prim

```
MST-PRIM( $G, w, r$ )
1 for each vertex  $u \in G.V$ 
2    $u.key = \infty$ 
3    $u.\pi = \text{NIL}$ 
4  $r.key = 0$ 
5  $Q = \emptyset$ 
6 for each vertex  $u \in G.V$ 
7   INSERT( $Q, u$ )
8 while  $Q \neq \emptyset$ 
9    $u = \text{EXTRACT-MIN}(Q)$  // add  $u$  to the tree
10  for each vertex  $v$  in // update keys of  $u$ 's non-tree neighbors
11     $G.Adj[u]$ 
12    if  $v \in Q$  and  $w(u, v) < v.key$ 
13       $v.\pi = u$ 
14       $v.key = w(u, v)$ 
15      DECREASE-KEY( $Q, v, w(u, v)$ )
```

Tomado de [0]

Utiliza colas de  
prioridad  
(implementación  
montículos)

$$O(V \lg V + E \lg V) = O(E \lg V)$$

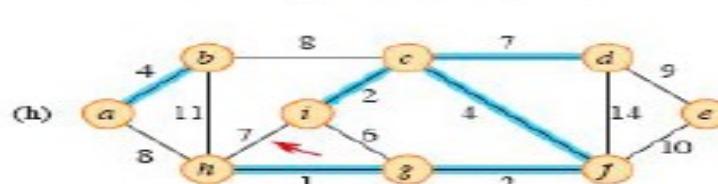
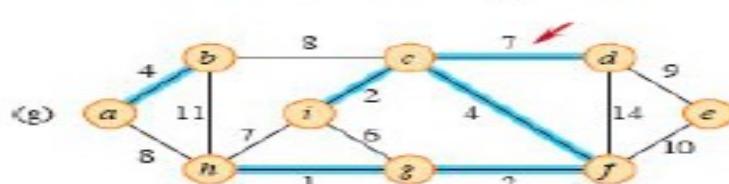
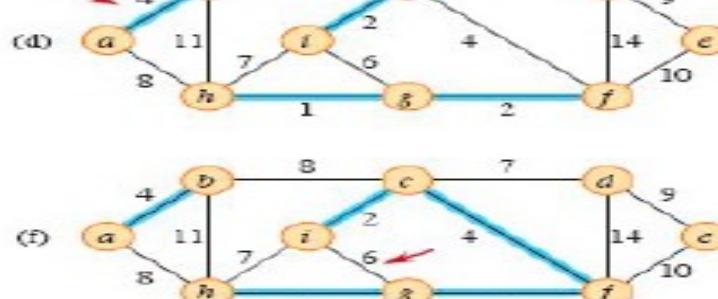
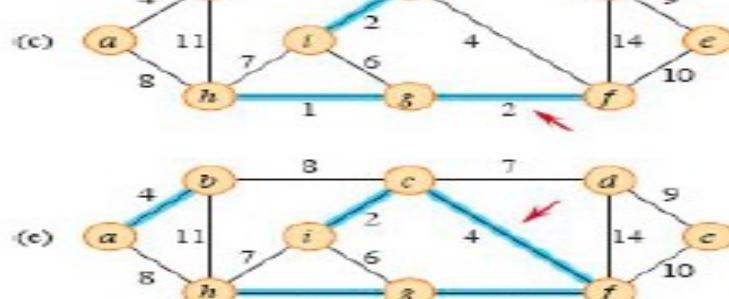
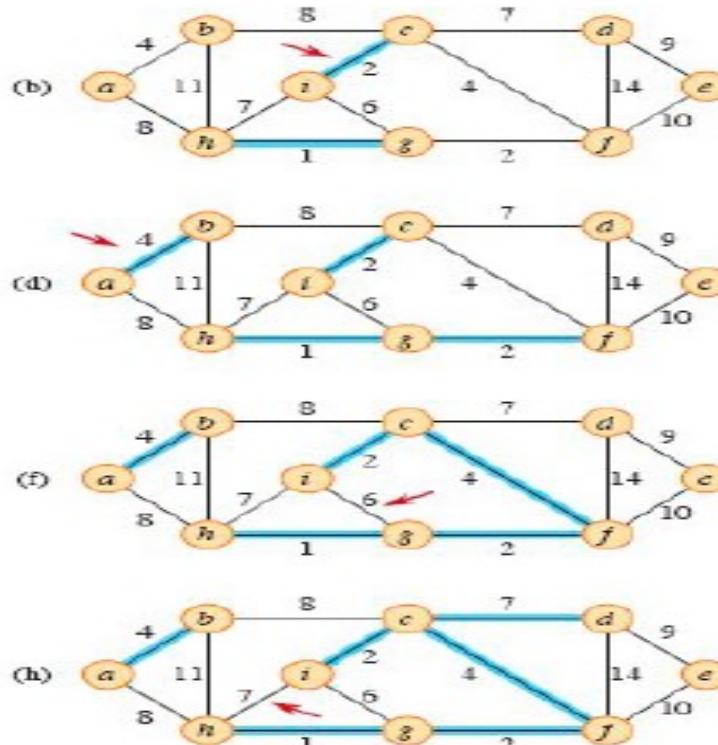
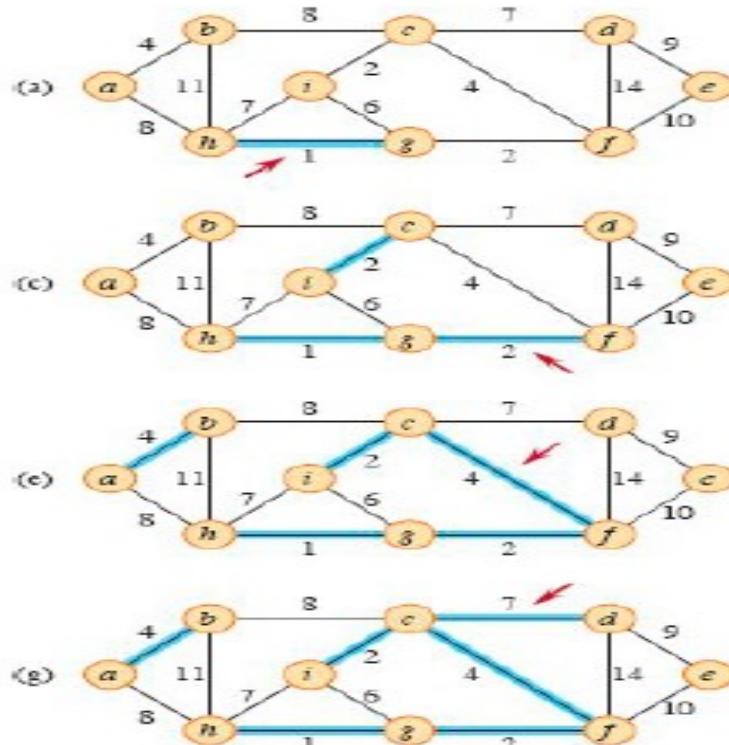
# Prog. Voraz- Árbol Recubrimiento Mínimo

Estrategia Voraz: (Kruskal)

Incrementalmente construir un árbol de recubrimiento mínimo

Propiedad de escogencia voraz:

Escoger la arista de menor peso que conecte diferentes componentes



Tomado de [0]

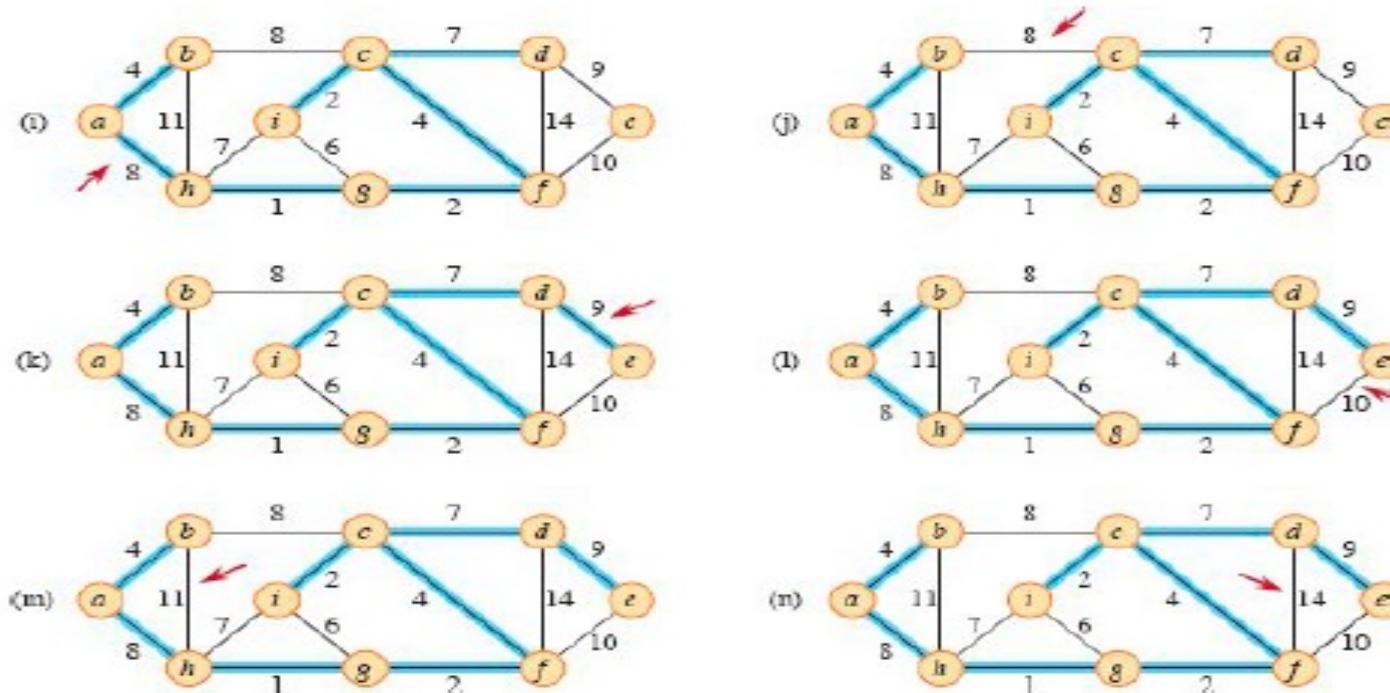
# Prog. Voraz- Árbol Recubrimiento Mínimo

Estrategia Voraz: (Kruskal)

Incrementalmente construir un árbol de recubrimiento mínimo

Propiedad de escogencia voraz:

Escoger la arista de menor peso que conecte diferentes componentes



Tomado de [0]

Ambas estrategias voraces (Prim y Kruskal) son óptimas.