CI2613: Algoritmos y Estructuras III

Blai Bonet

Universidad Simón Bolívar, Caracas, Venezuela

Enero-Marzo 2015

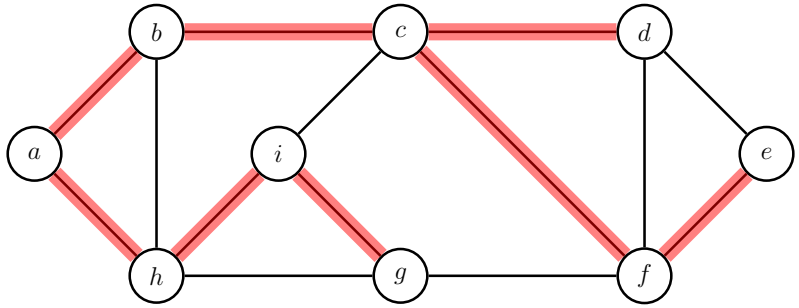
Árbol cobertor de peso mínimo

Árbol cobertor

Dado un grafo **no dirigido** $G = (V, E)$, un árbol cobertor de G es un **subgrafo** $T = (V, E')$ tal que:

- $E' \subseteq E$
- T es un árbol; equivalentemente:
  1. T es conectado y $|E'| = |V| - 1$
  2. Para cada par $u, v \in V$, existe un único camino de u a v en T

Árbol cobertor: Ejemplo



Árbol cobertor de peso mínimo

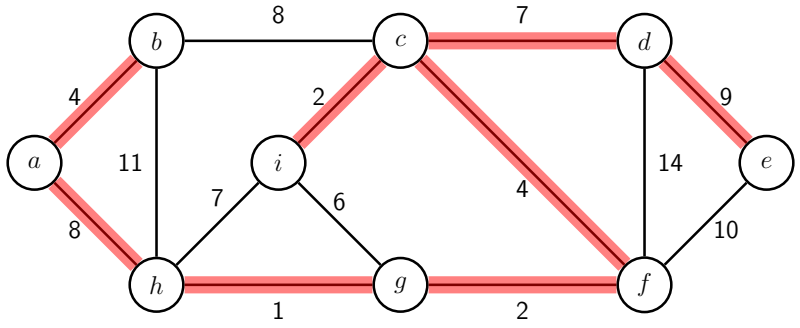
Considere un grafo no dirigido $G = (V, E)$ y **pesos** $w(u, v) \in \mathbb{R}$ para cada arista $(u, v) \in E$

Dado un árbol cobertor $T = (V, E')$ de G:

- El **peso de** T es $w(T) = \sum_{(u,v)\in E'} w(u, v)$
- Decimos que T es **mínimo**, de **peso mínimo** ó **MST** si $w(T) \leq w(T')$ para cualquier otro árbol cobertor T' de G

El árbol cobertor de mínimo peso es también llamado **Minimum(-weight) Spanning Tree (MST)**

Árbol cobertor de peso mínimo: Ejemplo



Algoritmos para el cálculo del MST

Veremos dos algoritmos **greedy** para el cálculo del MST:

- Algoritmo de Kruskal
- Algoritmo de Prim

Ambos algoritmos pueden facilmente implementarse en tiempo $O(E \log V)$, pero el tiempo puede mejorarse para ambos algoritmos

Ambos algoritmos son instancias de un **método general** para MST

Método general para el cálculo de MST

Considere un grafo no dirigido $G = (V, E)$ con pesos $w : E \to \mathbb{R}$

La idea es construir un MST de forma iterativa:

- Inicialmente el **pseudo-MST** es vacío
- En cada paso se **agrega o descarta** una arista para el pseudo-MST
- Se termina cuando el pseudo-MST es un árbol

El método garantiza el siguiente **invariante** al inicio de cada iteración:

Si A son las aristas del pseudo-MST, existe un MST $T = (V, E')$ tal que $A \subseteq E'$

Aristas "seguras" y esquema general

Considere una iteración del método general en donde ya hemos calculado un conjunto A de aristas

Si $(u, v) \in E$ es una arista tal que $A' = A \cup \{(u, v)\}$ safisface el invariante (i.e., existe un MST $T = (V, E')$ con $A' \subseteq E'$), entonces decimos que la arista (u, v) es **segura para** A

```
1  generic-MST(G, w):
2      A = ∅
3      while A no es MST
4          Encontrar una arista (u,v) segura para A
5          A = A ∪ { (u,v) }
6      return A
```

Para implementar el método tenemos que encontrar de forma **eficiente** una arista segura para un conjunto dado A

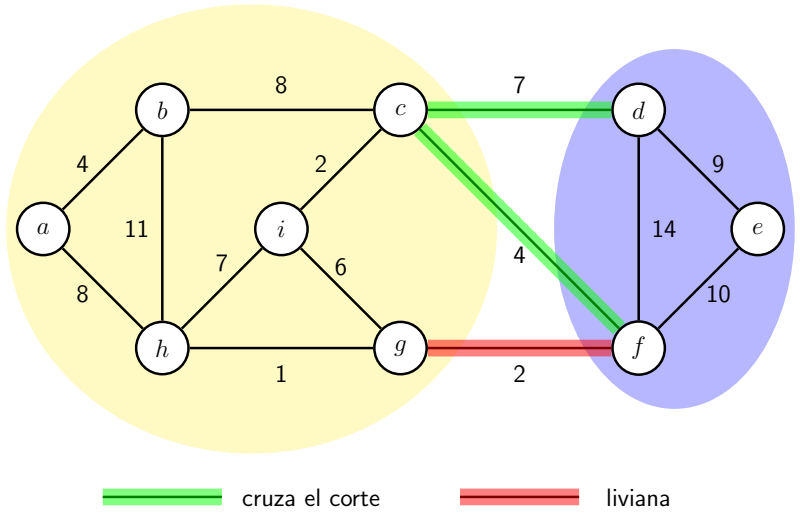
Cortes de un grafo

Considere un grafo no dirigido $G = (V, E)$ con pesos $w : E \to \mathbb{R}$

- Un **corte** de G es una **partición** $(S, V \setminus S)$ de sus vértices
- Una arista (u, v) **cruza** el corte $(S, V \setminus S)$ ssi $u \in S$ y $v \in V \setminus S$
- Un corte $(S, V \setminus S)$ **respeta** un conjunto de aristas A si ninguna arista en A cruza el corte
- Una arista que cruza un corte $(S, V \setminus S)$ es **liviana** si su peso es mínimo entre todas las aristas que cruzan el corte

En general, una arista que satisface una propiedad φ es liviana si su peso es mínimo entre todas las aristas que satisfacen φ

Cortes de un grafo: Ejemplo



Encontrando aristas seguras

Teorema

Considere:

- *un grafo $G = (V, E)$ conectado y no dirigido con pesos $w : E \to \mathbb{R}$*
- *un subconjunto $A \subseteq E$ contenido en un MST $T = (V, E')$ de G*
- *un corte $(S, V \setminus S)$ de G que respeta A*
- *una arista liviana (u, v) que cruza $(S, V \setminus S)$*

Entonces, la arista (u, v) es segura para A

Demostración del Teorema

- un grafo $G = (V, E)$ conectado y no dirigido con pesos $w : E \to \mathbb{R}$
- un subconjunto $A \subseteq E$ contenido en un MST $T = (V, E')$ de G
- un corte $C = (S, V \setminus S)$ de G que respeta A
- una arista liviana (u, v) que cruza C

Sea $T = (V, E')$ un MST con $A \subseteq E'$. Debemos encontrar un MST $T' = (V, E'')$ con $A \cup \{(u, v)\} \subseteq E''$ para mostrar que (u, v) es segura

Suponga que $A \cup \{(u, v)\} \nsubseteq E'$ (en otro caso no hay nada que mostrar)

Agreguemos (u, v) a T para **formar un ciclo**. El ciclo contiene una arista $(x, y) \neq (u, v)$ que cruza C (¿por qué?). Sea $E'' = (E \cup \{(u, v)\}) \setminus \{(x, y)\}$

Veamos que $T' = (V, E'')$ es un MST y $A \cup \{(u, v)\} \subseteq E''$:

1. T' es conectado ya que existe camino $x \rightsquigarrow y$ y por lo tanto es árbol
2. $w(T') = w(T) + w(u, v) - w(x, y) \leq w(T)$ y por lo tanto T' es MST
3. $(x, y) \notin A$ porque C respeta A y (x, y) cruza C
4. Como $A \subseteq E'$, tenemos $A \cup \{(u, v)\} \subseteq E''$ □

Encontrando aristas seguras

Corolario

Considere:

- *un grafo $G = (V, E)$ conectado y no dirigido con pesos $w : E \to \mathbb{R}$*
- *un subconjunto $A \subseteq E$ contenido en un MST $T = (V, E')$ de G*
- *el grafo $C = (V_C, E_C)$ de componentes del bosque $G_A = (V, A)$*
- *una arista liviana (u, v) que conecta componentes distintas en C*

Entonces, la arista (u, v) es segura para A

Prueba: sean C_1 y C_2 las dos componentes de G_A tal que $u \in C_1$ y $v \in C_2$. Considere el corte $(C_1, V \setminus C_1)$.

Ninguna arista de A cruza el corte. Por otro lado, la arista (u, v) cruza el corte y es liviana entre aquellas que lo cruza.

Por el Teorema, la arista (u, v) es segura para A □

Algoritmo de Kruskal

El algoritmo de Kruskal es una implementación del método general

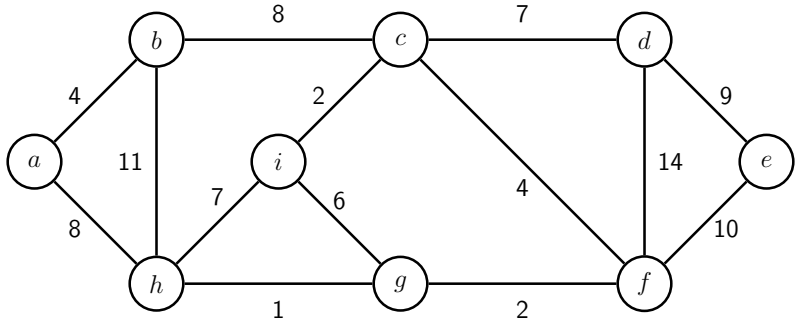
El conjunto A de aristas define un bosque $G_A = (V, A)$

Por el Corolario, cualquier arista liviana (u, v) que conecte dos componentes de G_A es segura para A y puede ser agregada sin violar el invariante

Algoritmo de Kruskal: Pseudocódigo

```
1  MST-Kruskal(G, w):
2      % inicialización
3      A = ∅
4      foreach Vertice u: make-set(u)
5
6      % algoritmo de Kruskal
7      Ordenar las aristas E de menor a mayor peso
8      foreach arista (u,v) en orden de peso
9          if find(u) != find(v)
10             A = A ∪ { (u,v) }
11             union(u,v)
12     return A
```

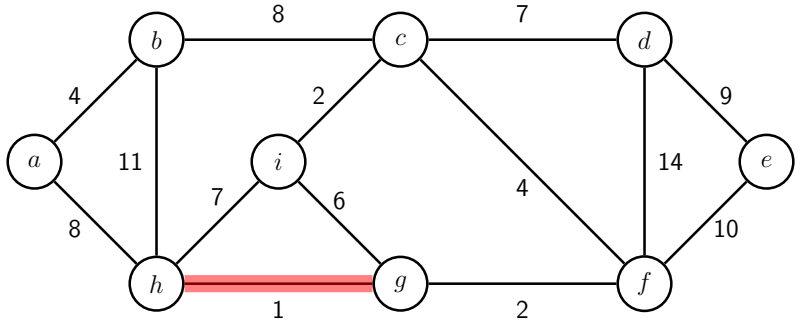
Algoritmo de Kruskal: Ejemplo



$\{a\},\ \{b\},\ \{c\},\ \{d\},\ \{e\},\ \{f\},\ \{g\},\ \{h\},\ \{i\}$

MST final

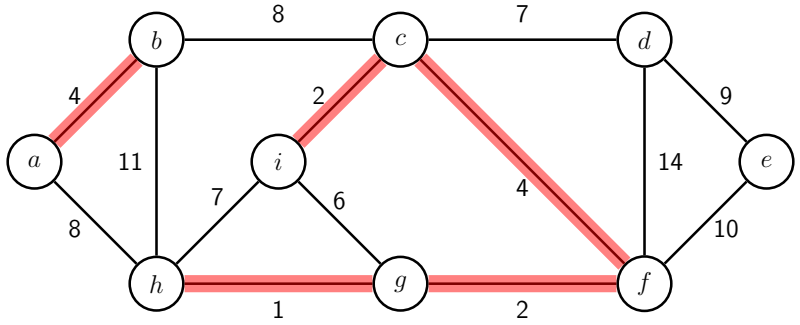
Algoritmo de Kruskal: Ejemplo



$\{a\},\ \{b\},\ \{c\},\ \{d\},\ \{e\},\ \{f\},\ \{g,h\},\ \{i\}$

MST final

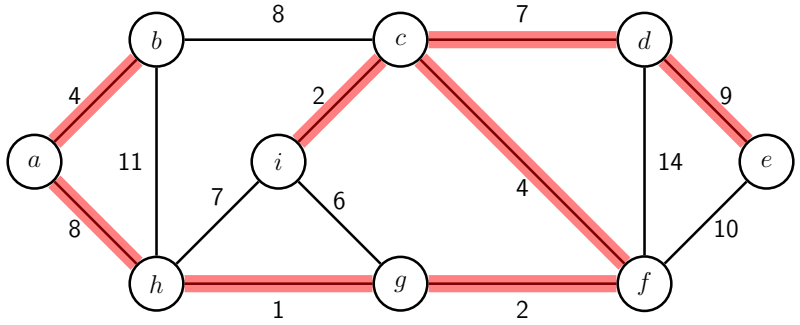
Algoritmo de Kruskal: Ejemplo



$\{a,b\},\ \{c,f,g,h,i\},\ \{d\},\ \{e\}$

MST final

Algoritmo de Kruskal: Ejemplo



$\{a,b,c,d,e,f,g,h,i\}$

MST final

Algoritmo de Kruskal: Pseudocódigo

```
1  MST-Kruskal(G, w):
2      % inicialización
3      A = ∅
4      foreach Vertice u: make-set(u)
5
6      % algoritmo de Kruskal
7      Ordenar las aristas E de menor a mayor peso
8      foreach arista (u,v) en orden de peso
9          if find(u) != find(v)
10             A = A ∪ { (u,v) }
11             union(u,v)
12     return A
```


Análisis del algoritmo de Kruskal

1. El algoritmo es correcto por el Corolario
2. Tiempo para ordenar aristas: $O(E \log E)$
3. Tiempo para todas las operaciones sobre la ED: $O(E \log^* V)$
4. Tiempo total:

$$O(E \log E) + O(E \log^* V) = O(E \log V)$$

   ya que $E = O(V^2)$ y $\log^* V = O(\log V)$

5. Si las aristas ya están ordenadas o se pueden ordenan en tiempo lineal (e.g. con radix-sort si los **pesos son enteros**), el tiempo total es $O(E \log^* V)$ que en la **práctica** es $O(E)$

Algoritmo de Prim

El algoritmo de Prim es una implementación del método general

El conjunto A define un **único** árbol que "crece" a partir de una **raíz**

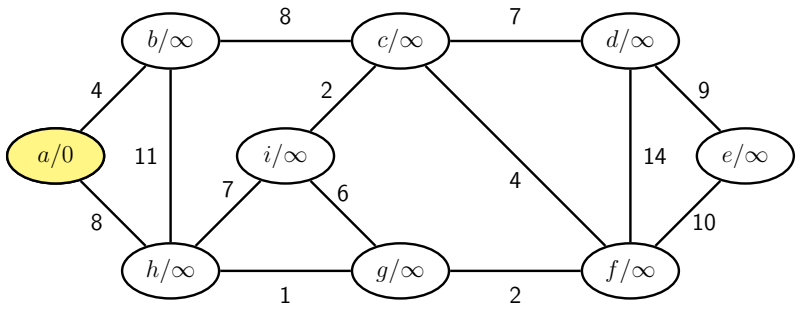
Cada arista que se agrega a A es una **arista liviana** que conecta A con algún **vértice aislado**

Por el Teorema, dichas aristas son seguras para A

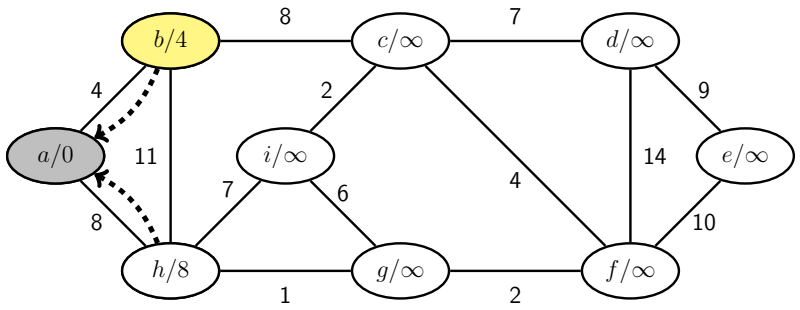
Algoritmo de Prim: Pseudocódigo

```
1  MST-Prim(G, w, r):
2      % inicialización
3      foreach Vertice u
4          key[u] = ∞
5          π[u] = null
6      key[r] = 0
7
8      % cola de prioridad ordenada por min key[.]
9      PriorityQueue q
10     foreach Vertice u
11         q.insert(u)
12
13     while q != ∅
14         u = q.extract-min()
15         foreach Vertice v in adyacentes[u]
16             if v ∈ q && w(u,v) < key[v]
17                 key[v] = w(u,v)  % involucra decresase-key
18                 π[v] = u
```

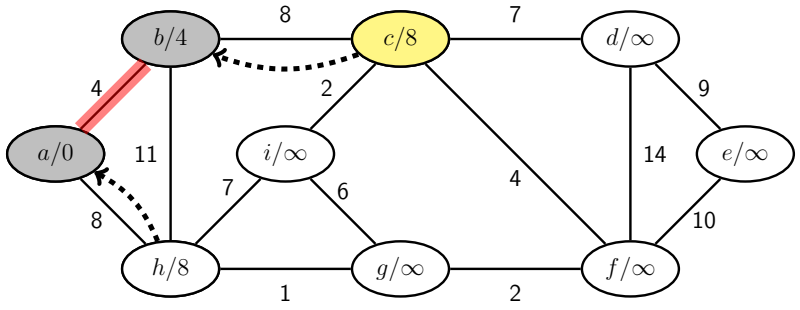
Algoritmo de Prim: Ejemplo



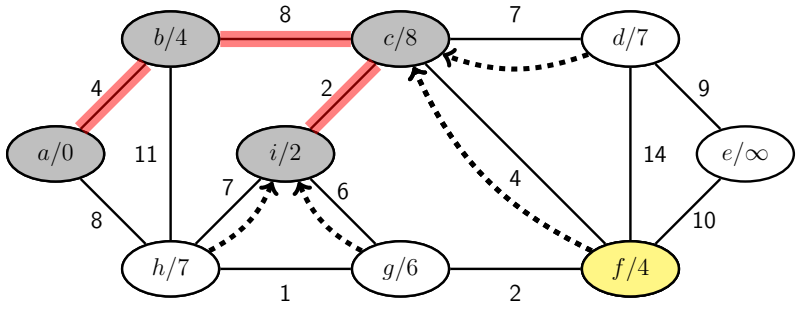
Algoritmo de Prim: Ejemplo



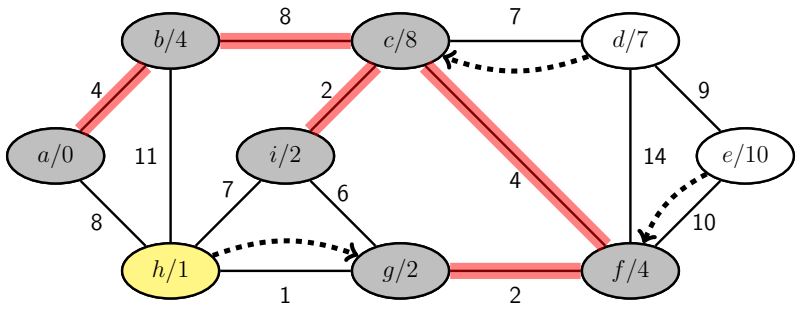
Algoritmo de Prim: Ejemplo



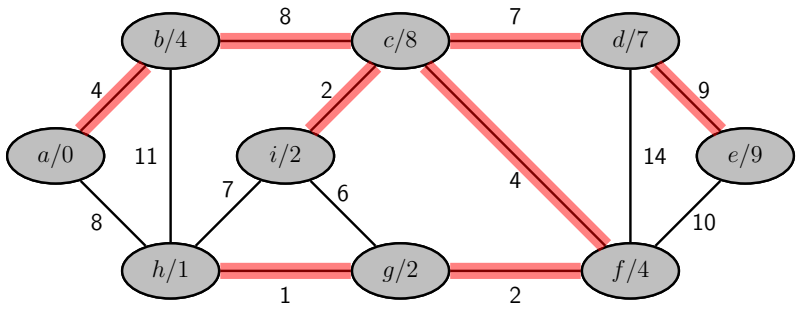
Algoritmo de Prim: Ejemplo



Algoritmo de Prim: Ejemplo



Algoritmo de Prim: Ejemplo



Invariantes en el algoritmo de Prim

Al inicio de cada iteración, el siguiente invariante se cumple:

Invariante:

1. El conjunto de aristas es $A = \{(v, \pi[v]) : v \in V \setminus (Q \cup \{r\})\}$
2. Los vértices en el pseudo-MST son aquellos que no están en la cola
3. Para todo vértice v: si $\pi[v] \neq$ **`null`**,
   - $key[v] < \infty$
   - Si $v \in Q$, $key[v]$ es el peso de una arista liviana que conecta v con el pseudo-MST

Por lo tanto, al inicio de cada iteración: si $\pi[u] \neq$ **`null`**, entonces la arista $(u, \pi[u])$ es una arista liviana que conecta el pseudo-MST con un vértice aislado

Análisis del algoritmo de Prim

1. El algoritmo es correcto por el Teorema
2. Tiempo en inicialización: $O(V) + O(V \log V) = O(V \log V)$
3. Tiempo en ops. de cola: $O(V \log V) + O(E \log V) = O(E \log V)$
4. Tiempo agregado para el lazo interno: $O(E)$
5. Tiempo total:

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

6. Con un **heap de Fibonacci**, la operación `decrease-key` toma tiempo $O(1)$ amortizado y obtenemos un tiempo total $O(E + V \log V)$

Resumen

- Queremos calcular un MST T para un grafo no dirigido y conectado $G = (V, E)$
- Existe un método general iterativo basado en la idea de aristas seguras y un invariante
- Dos implementaciones del método general:
  - **Algoritmo de Kruskal:** mantiene un bosque de componentes las cuales pueden crecer de forma independiente. Si las aristas pueden ordenarse de forma eficiente, se puede implementar en tiempo $O(E \log^* V)$ que en la práctica es $O(E)$
  - **Algoritmo de Prim:** se crece una componente a partir de un vértice raíz. Una implementación con heap de Fibonacci se logra en tiempo $O(E + V \log V)$