MST: dos algoritmos glotones

comp-420

Dos algoritmos glotones para MST

- Algoritmo de Kruskal.
- Algoritmo de Prim.
- Ambos pueden ejecutarse en un tiempo O(E log V) utilizando montículos binarios.
- Ambos algoritmos son glotones o greedy.
- En cada paso el algoritmo toma la mejor opción en ese momento.
- Generalmente esta estrategia no garantiza encontrar una solución globalmente óptima al problema.
- En el caso de los MST estos algoritmos glotones si nos llevan al árbol con menor costo.

Dos algoritmos glotones para MST

- Ambos algoritmos utilizan una regla específica para encontrar aristas seguras.
- En el Algoritmo de Kruskal, el conjunto T es un bosque.
- La arista segura que se agrega siempre es la arista con menor peso que conecta dos componentes distintas.
- En el Algorimo de Prim el conjunto T es un sólo árbol.
- La arista segura que se agrega es siempre aquella con menor peso que conecte el árbol a un vértice que todavía no esté en el MST.

MST: Algoritmo de Prim (Jarník)

- Es otro caso especial del algoritmo genérico para encontrar MST.
- Las aristas de T siempre forman un solo árbol.
- El algoritmo empieza de un vértice arbitrario r y el árbol crece hasta que conecta a todos los vértices en V.
- En cada paso se agrega una arista segura al árbol T que conecta a T a un vértice aislado de $G_T = (V,T)$.
- Cuando el algoritmo termina, T es un árbol generador mínimo.
- Esta es una estratégia greedy ya que el árbol crece en cada paso con una arista que contribuye lo mínimo posible al costo total del árbol.

MST: Algoritmo de Prim

MST-PRIM(G, w, r)

```
1   for each u ∈ V[G]
2       do key[u] ← ∞
3          π[u] ← NIL
4   key[r] ← 0
5   Q ← V[G]
6   while Q ≠ ∅
7       do u ← EXTRACT-MIN(Q)
8          for each v ∈ Adj[u]
9              do if v ∈ Q and w(u, v) < key[v]
10                then π[v] ← u
11                     key[v] ← w(u, v)
```

MST: Algoritmo de Prim

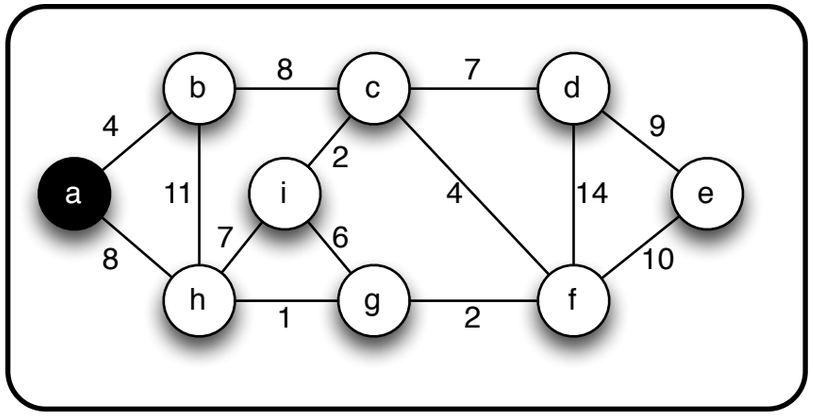
- Mientras se ejecuta el algoritmo los vértices que no están en el árbol viven en una cola de prioridad min Q respecto a su campo key.
- Para cada vértice v, v.key es el peso mínimo de cualquier arista que conecte a v con un vértice en el árbol. Por convención v.key = ∞ si no existe tal arista.
- El campo v.π se refiere al padre de v en el árbol.
- Durante la ejecución del algoritmo el conjunto T se mantiene implícitamente como:

$$A = \{(v, \pi[v]) : v \in V - \{r\} - Q\}.$$

- Cuando el algoritmo termina, la cola de prioridad min Q está vacía y

$$A = \{(v, \pi[v]) : v \in V - \{r\}\}.$$

MST: Algoritmo de Prim



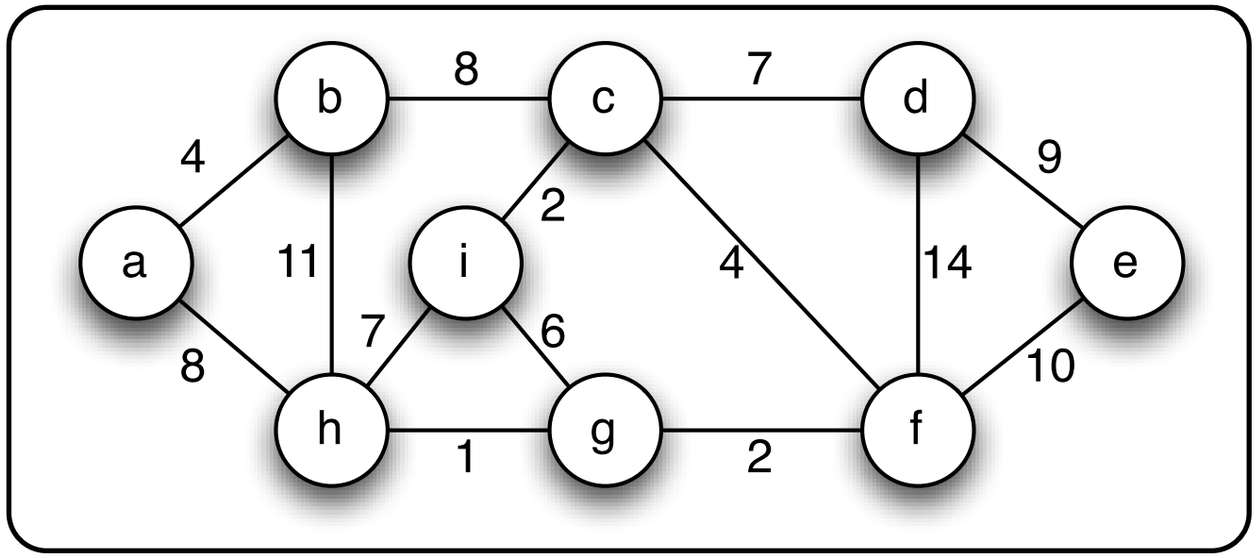
MST: Algoritmo de Kruskal

- Descubierto por Kruskal en 1956.
- Encuentra una arista segura para agregar al bosque T encontrando entre todas las aristas que conecten dos árboles cualquiera en el bosque, la arista (u,v) con menor costo.
- Sean C_1 y C_2 dos árboles conectados por (u,v). Como (u,v) deben ser aristas claras conectando C_1 a otro árbol, sabemos que es una arista segura para C_1.
- El algoritmo de Kruskal es glotón porque en cada paso agregamos al bosque T la arista clara con el menor costo posible.

MST: Algoritmo de Kruskal

```
MST-KRUSKAL(G, w)
1  A ← ∅
2  for each vertex v ∈ V[G]
3      do MAKE-SET(v)
4  sort the edges of E into nondecreasing order by weight w
5  for each edge (u, v) ∈ E, taken in nondecreasing order by weight
6      do if FIND-SET(u) ≠ FIND-SET(v)
7          then A ← A ∪ {(u, v)}
8               UNION(u, v)
9  return A
```

MST: Algoritmo de Kruskal



arista	costo
{h,g}	1
{i,c}	2
{g,f}	2
{a,b}	4
{c,f}	4
{i,g}	6
{c,d}	7
{h,i}	7
{a,h}	8
{b,c}	8
{d,e}	9
{f,e}	10
{b,h}	11
{d,f}	14

Aristas procesadas	Colección de conjuntos disjuntos: *A*								
aristas iniciales	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}
{h,g}	{a}	{b}	{c}	{d}	{e}	{f}	{g,h}		{i}
{i,c}	{a}	{b}	{c,i}		{d}	{e}	{f}	{g,h}	
…	…	…	…	…	…	…	…	…	…	…
{d,f}	{a,b,c,d,e,f,g,h,i}								

Kruskal: tiempo de ejecución

- Depende de la implementación elegida para conjuntos disjuntos.
- Suponemos representación por árboles con heurísticas.
- Inicializar en la línea 1 toma:
  - $O(1)$
- Ordenar las aristas en la línea 4 toma:
  - $O(E \log E)$
- El cíclo for de las líneas 5 a 8 realiza $O(E)$ operaciones FIND-SET y UNION en bósques disjuntos.
- Junto con las V operaciones MAKE-SET de las líneas 2 a 3 esto toma un tiempo de $O((V+E)\alpha(V))$.
- Como G está conectado, tenemos $E \geq V-1$ y las operaciones de conjuntos disjuntos toman $O(E\alpha(V))$.

Kruskal: tiempo de ejecución

- Ya que $\alpha(V) = O(\log V) = O(\log E)$, el tiempo de cálculo total del algoritmo de Kruskal es de $O(E \log E)$.
- Observando que $E \leq V^2$, tenemos $\log E = O(\log V)$ que nos permite reescribir el tiempo de ejecución total como:
  - $O(E \log V)$.