



Grafos: caminos mínimos y árbol expansión mínimo

MSc Edson Ticona Zegarra

Campamento de Programación



Contenido

Minimum Spanning Tree

- Conjuntos Disjuntos

- Kruskal

- Prim

- Comparación

Shortest Paths

- Algoritmo de Dijkstra

- Algoritmo de Bellman-Ford

- Algoritmo de Floyd-Warshall



Contenido

Minimum Spanning Tree

Conjuntos Disjuntos

Kruskal

Prim

Comparación

Shortest Paths

Algoritmo de Dijkstra

Algoritmo de Bellman-Ford

Algoritmo de Floyd-Warshall



Minimum Spanning Tree (MST)

- ▶ Dado un grafo no direccionado con pesos, se conoce como *árbol de expansión mínimo* a aquel grafo cuyas aristas sean un subconjunto del grafo original tal que todos los vértices estén conectados y la suma de sus pesos sea mínimo
- ▶ Dicho grafo es un árbol.
- ▶ Para solucionar este problema se tiene dos algoritmos ampliamente conocidos: el algoritmo de Kruskal y el algoritmo de Prim



Conjuntos Disjuntos

- ▶ Para el algoritmo de Kruskal se necesita una estructura de datos especial conocida como *conjuntos disjuntos*
- ▶ Un conjunto disjunto representa a varios conjuntos que no tienen elementos en común y se define un par de operaciones básicas, FIND y UNION
- ▶ Se necesita que ambas operaciones sean eficientes
- ▶ Cuando se busca un elemento se debe retornar el conjunto en el que se encuentra
- ▶ Cuando dos conjuntos se unen, todos los elementos pasan a estar en el mismo conjunto



Conjuntos Disjuntos

- ▶ La manera más sencilla de hacer esto es con un arreglo. Se tiene un arreglo C del mismo tamaño que la cantidad de elementos
- ▶ El valor $C[i]$ indica a que conjunto pertenece el i -ésimo elemento
- ▶ Esta implementación es directa y sin mayores complicaciones
- ▶ Entonces $\text{FIND}(i)$ será únicamente retornar el valor de $C[i]$



Union

input : a y b son los elementos que deben quedar en un mismo conjunto

$k \leftarrow Find(b)$;

for $i \in C$ **do**

if $Find(i) == k$ **then**

$C[i] \leftarrow a$

end

end

- FIND tiene una complejidad $O(1)$ y UNION tiene una complejidad $O(n)$



Balancing (union by rank)

- ▶ Si consideramos almacenar árboles por cada elemento, entonces podemos mejorar la complejidad de la operación UNION
- ▶ Los árboles son estructuras eficientes cuando se encuentra balanceados, por ello es conveniente almacenar en el nodo raíz la altura del árbol
- ▶ Así, al unir dos árboles, escogemos que el árbol de menor altura para que sea un nodo hijo del otro árbol



Path compression

- ▶ En `FIND`, podemos recorrer dos veces el árbol para que los elementos apunten directamente al nodo raíz, reduciendo el tamaño del árbol
- ▶ Si bien es cierto la operación de `FIND` se complica un poco más y su complejidad deja de ser constante, la complejidad de `UNION` es reducida ampliamente compensando lo perdido en `FIND`



Kruskal

- ▶ Cada vértice es un conjunto disjunto
- ▶ Se va agregando aristas de menor peso, uniendo diversos componentes
- ▶ Se termina cuando se tiene un único conjunto



Prim

- ▶ Se inicia de un vértice arbitrario
- ▶ Se va agregando vértices de menor peso que estén conectados al componente actual
- ▶ Se termina cuando todos los vértices estan conectados



Comparación

- ▶ La complejidad de ambos algoritmos dependen de la estructura de datos que se utilice.
- ▶ La complejidad de Prim va desde $O(V^2)$ hasta un $O(E \log V)$ si se utiliza *Fibonacci heaps*
- ▶ La complejidad de Kruskal es $O(E \log V)$ si se utiliza una estructura de conjuntos disjuntos eficiente
- ▶ El algoritmo de Kruskal es más rápido para grafos esparsos
- ▶ El algoritmo de Prim es más rápido para grafos densos



Contenido

Minimum Spanning Tree

Conjuntos Disjuntos

Kruskal

Prim

Comparación

Shortest Paths

Algoritmo de Dijkstra

Algoritmo de Bellman-Ford

Algoritmo de Floyd-Warshall



Single Source Shortest Paths (SSSP)

- ▶ Dado un grafo, se busca el camino mínimo entre un par de vértices s, t .
- ▶ Si consideramos un grafo sin pesos, se busca el camino con menor cantidad de aristas y tal problema se puede solucionar con el BFS
- ▶ Si consideramos un grafo con pesos, se busca el camino tal que la suma de los pesos de las aristas sea mínimo, se puede solucionar con el algoritmo de Dijkstra
- ▶ Si consideramos un grafos con pesos negativos, tal que no existan ciclos negativos, se puede solucionar con el algoritmo de Bellman-Ford
- ▶ En todos los casos anteriores, los algoritmos nos retornan las distancias mínimas entre un vértice s y el resto de vértices en el grafo



Relajación

input : u, v y $w(u, v)$ son un par de vértices y el peso de la arista que los conecta

```
if  $dist[v] > dist[u] + w(e, v)$  then  
     $dist[v] \leftarrow dist[u] + w(e, v)$   
     $parent[v] \leftarrow u$   
    return true  
end  
return false
```



Algoritmo de Dijkstra

- ▶ Se considera un vector de distancias, tal que cada vértice está a una distancia infinita del vértice inicial s .
- ▶ Se tiene que procesar cada vértice del grafo, almacenandolos en una cola de prioridades
- ▶ La prioridad está dada por la distancia mínima calculada hasta el momento
- ▶ Para cada vértice u de la cola, relajar cada uno de sus vértices v adyacentes. Si el vértice fue relajado, quiere decir que la distancia fue modificada, por tanto se debe actualizar la cola de prioridades



Algoritmo de Dijkstra

input : G grafo

$Q \leftarrow V$; // Q cola de prioridades en base a la
distancia mínima

while $!Q.empty()$ **do**

$u \leftarrow Q.top()$

for $v \leftarrow Adj(u)$ **do**

$r \leftarrow Relax(u, v, w)$

if r **then**

$Update(Q)$

end

end

end



Algoritmo de Dijkstra

- Complejidad: $O(|E| + |V| \log |V|)$



Algoritmo de Bellman-Ford

- ▶ Es más genérico que el algoritmo de Dijkstra, pero es más lento
- ▶ Utiliza también la noción de “relajar” aristas
- ▶ Notar que un grafo con un ciclo negativo no tiene solución
- ▶ El algoritmo relaja todas las aristas, en un orden arbitrario, $|V| - 1$ veces



Algoritmo de Bellman-Ford

```

input :  $G = (V, E)$  grafo

for  $u \in V$  do
  | for  $e = (u, v) \in E$  do
  | |  $Relax(u, v, w)$ 
  | end
end

for  $e = (u, v) \in E$  do
  | if  $dist[v] > dist[u] + w(u, v)$  then
  | | return false
  | end
end

return true

```



All Pairs Shortest Paths (APSP)

- ▶ El algoritmo Floyd-Warshall resuelve calcula la distancia mínima entre cualquier par de vértices de un grafo
- ▶ Este algoritmo provee una solución más eficiente que llamar $|V|$ veces al algoritmo de Dijkstra



Algoritmo de Floyd-Warshall

