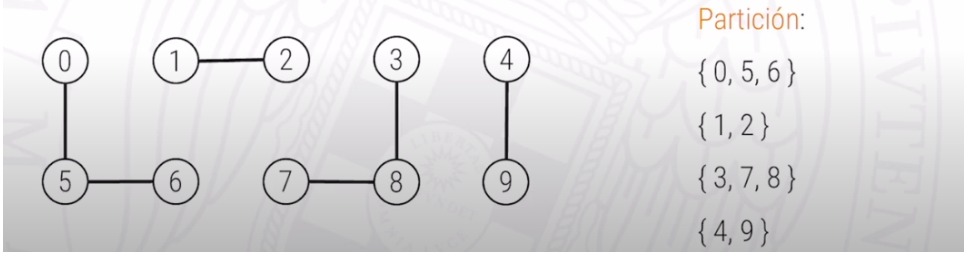
Semana 6.- Conjuntos disjuntos, grafos valorados y árboles de recubrimiento de coste mínimo.

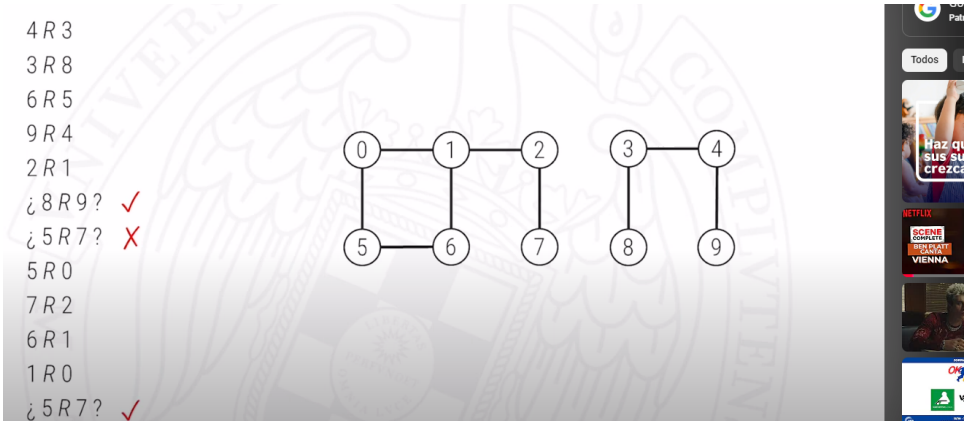
Relación de equivalencia

Si queremos representar una relación de equivalencia R:

- Reflexiva: aRa
- Simétrica: aRb → bRa
- Transitiva: aRb y bRc → aRc



Equivalencia dinámica



TAD de conjuntos disjuntos

Posible implementación: búsqueda rápida

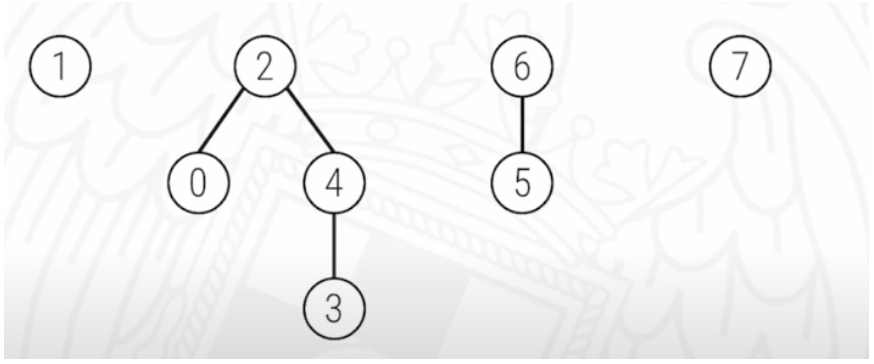
El siguiente vector:

	0	1	2	3	4	5	6	7	8	9
id[]	0	1	1	8	8	0	0	1	8	8

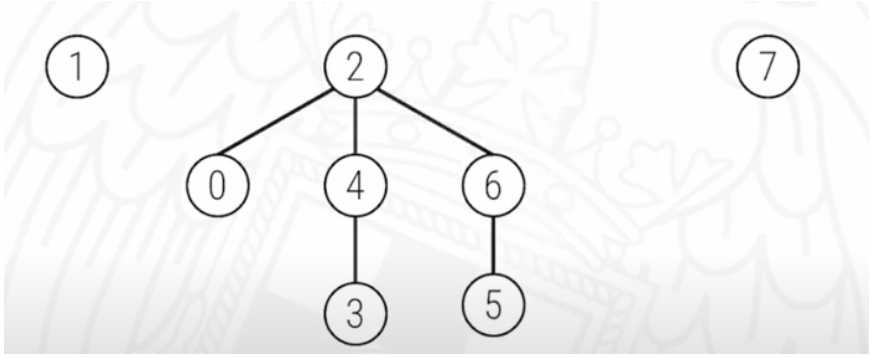
representa la partición: {0,5,6}, {1,2,7}, {3,4,8,9}

Posible implementación: unión rápida

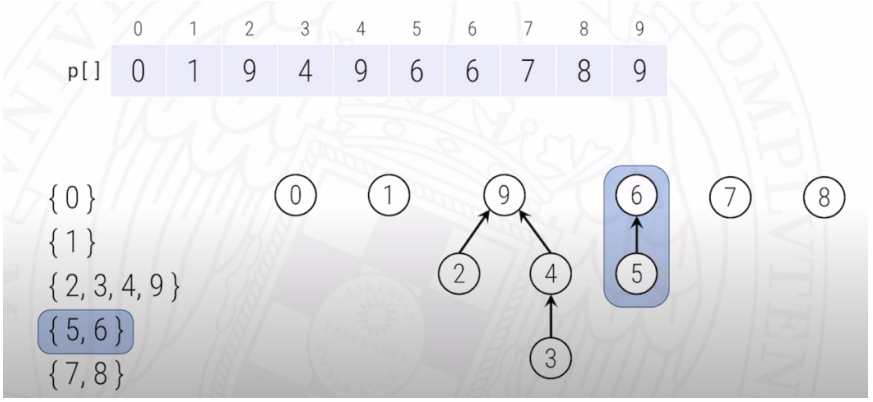
Cada conjunto se representa mediante un árbol, aprovecha que es fácil unir árboles ya que solo habría que poner uno como hijo del otro. Un ejemplo es el siguiente:



Se une el árbol de raíz 2 con el de raíz 6.

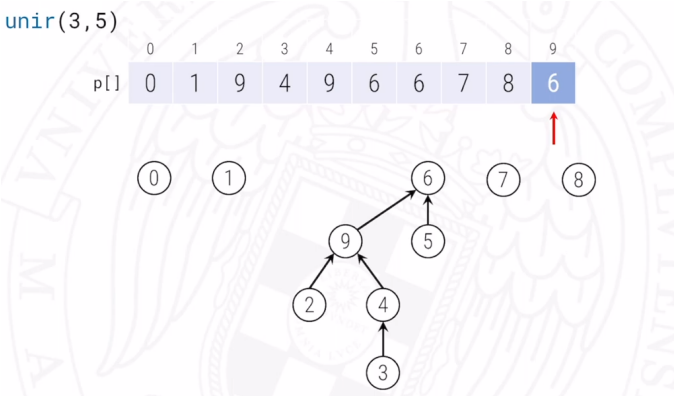


Cada conjunto se representa mediante un árbol. Donde cada elemento se almacena con quién es su padre.



¿Qué pasaría si quisiéramos realizar una unión?

Imaginemos que queremos unir el 3 con el 5. Para ello primero debemos buscar la raíz donde se encuentran esos elementos, en este caso 9 y 6 y después hacer que uno pase a ser hijo del otro de la siguiente forma:



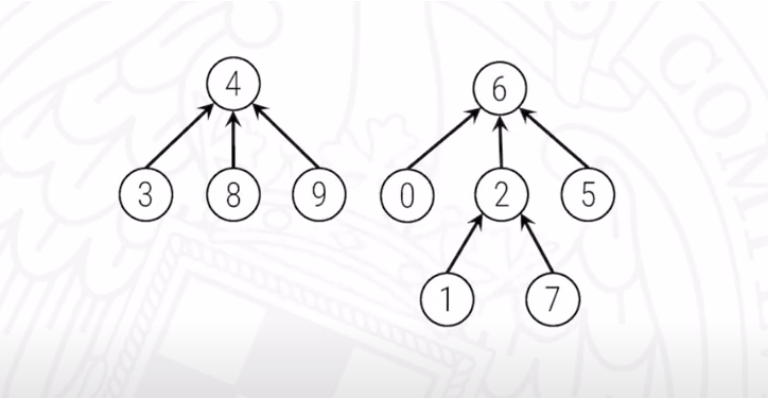
Esto tiene un coste de **O(N)** ya que hay que buscar en todo el vector los elementos que queremos unir.

Posible implementación: Unión por tamaños

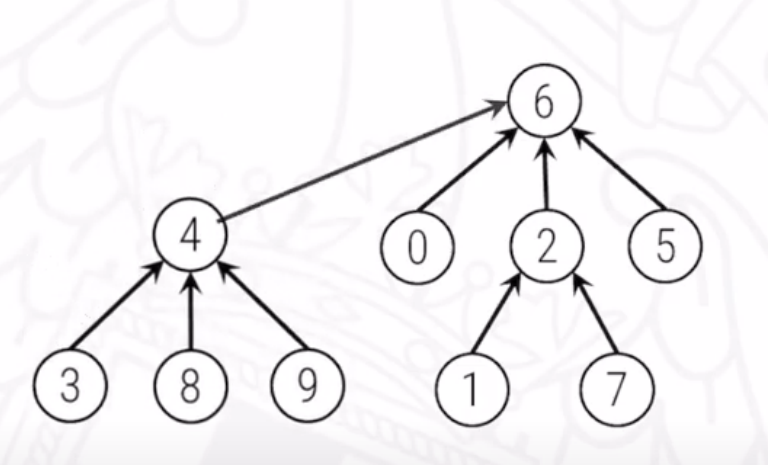
Se tienen que mantener los tamaños de los árboles (número de elementos).

Al unir, **el árbol más pequeño pasa a ser hijo del árbol mayo**r.

Imaginemos que quisiéramos hacer la unión (7,3) para los siguientes árboles:



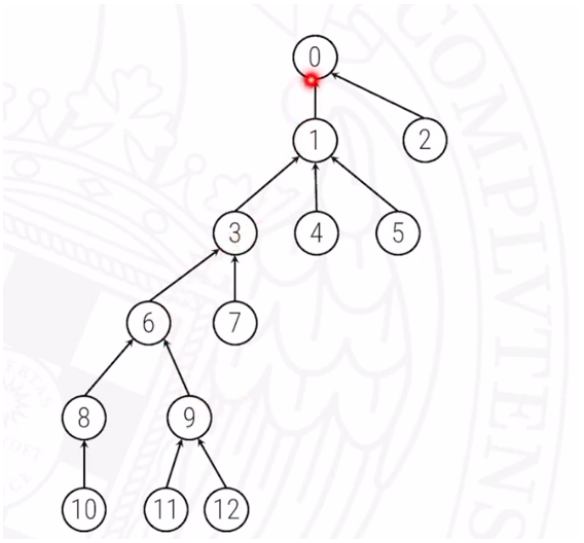
Quedaría tal que así debido a que el árbol que contiene al 7 es mayor que el que contiene al 3:

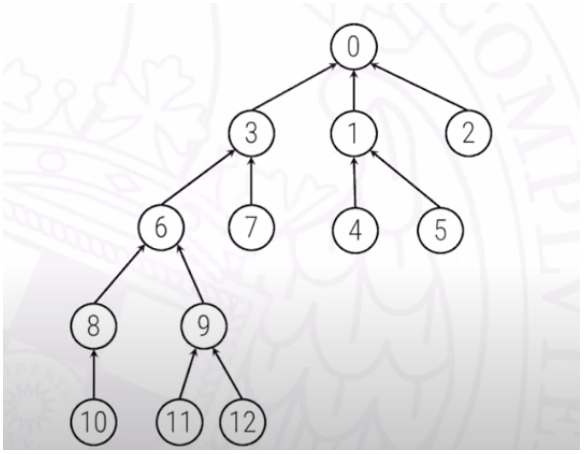


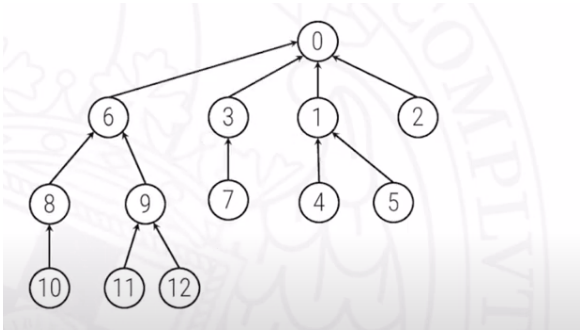
- La unión de raíces es constante.
- La búsqueda es proporcional a la profundidad del elemento búsqueda.
- Al hacer la unión por tamaños, la profundidad acotados es **O(logN).**

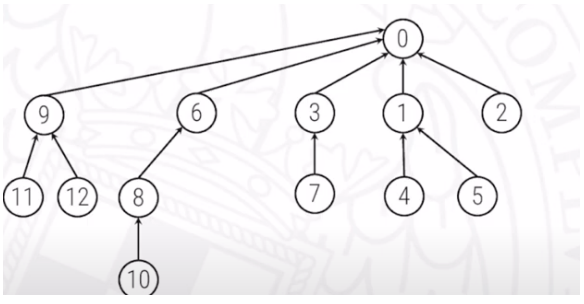
Posible implementación: Compresión de caminos.

Es más eficiente que el resto. Después de buscar la raíz del árbol donde se encuentra x, cambiamos su padre para que ese sea raíz.







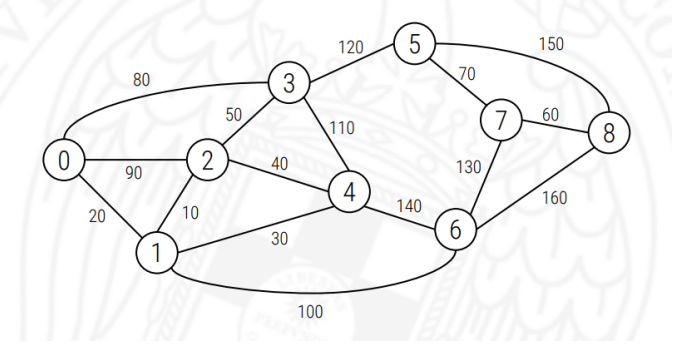


Esto tiene un coste de O(N+M*logN).

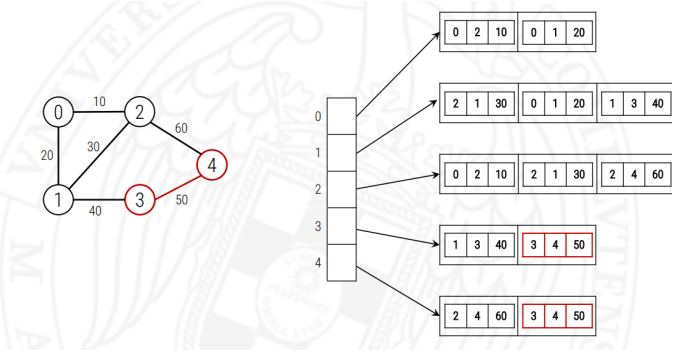
Implementación	complejidad en el caso peor
búsqueda rápida	$N\ M$
unión rápida	$N\ M$
unión rápida por tamaños	$N + M \log N$
unión rápida con compresión de caminos	$N + M \log N$
unión rápida por tamaños y con compresión de caminos	$N + M \lg^* N$

Grafos valorados

Grafos cuyas aristas tienen asociado un valor (peso, coste).



En este caso la lista de adyacencia quedaría de la siguiente forma:



Árbol de recubrimiento de coste mínimo

Un árbol de recubrimiento de un grafo G es un subgrafo T tal que:

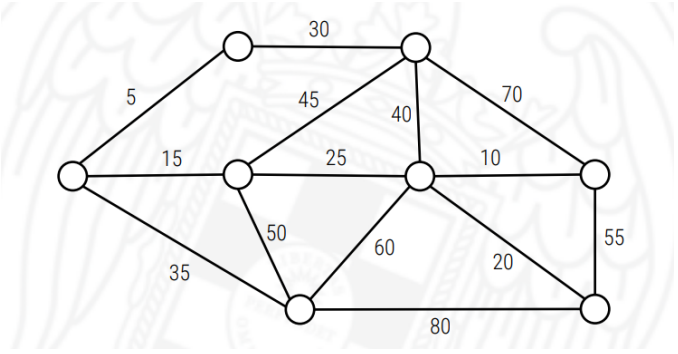
- T es un árbol conexo.
- T es de recubrimiento: alcanza todos los vértices de G.

Características:

Si T es un árbol de recubrimiento:

- Contiene exactamente V-1 aristas.
- Al eliminar cualquier arista T deja de ser conexo.
- Añadir cualquier arista T crea un ciclo.

Encontrar un árbol de recubrimiento



Basicamente se empieza desde la arista con menor coste y se van cogiendo las aristas que tengan menor coste mientras no creen un ciclo.

La propiedad del corte

Un corte de un grafo es una partición de sus vértices en dos conjuntos no vacíos.

Una arista cruza el corte si tiene un extremo en cada conjunto.

La propiedad del corte dicta que para cualquier corte, la arista de menor peso que lo cruza pertenece al ARM.