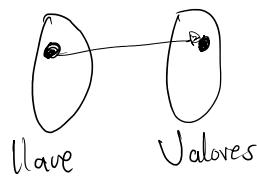
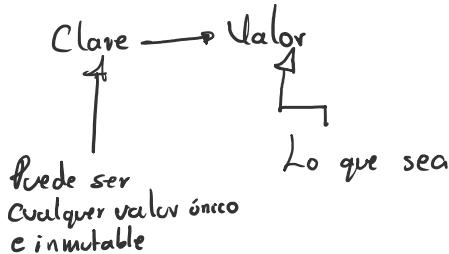
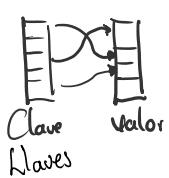


Mapas

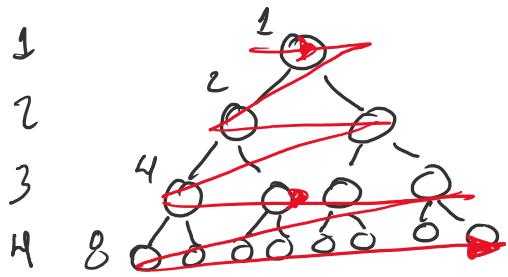
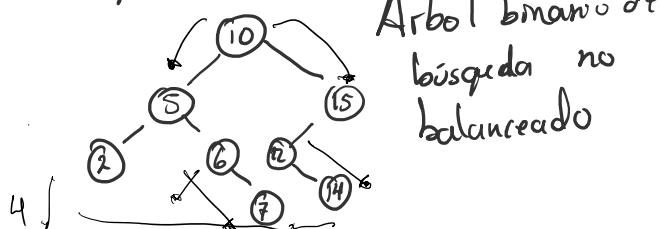
¿Qué son los mapas?

**Implementaciones**

- * Árboles autobalanceados
- * Tablas hash

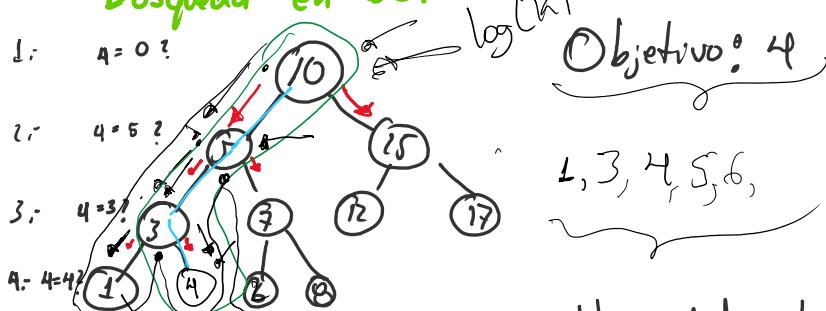
Árboles binarios de búsqueda auto-balanceado (Árbol AVL)

- * Todos los elementos a la izquierda de un nodo son menores
- * Todos los elementos a lo derecho son mayores



Árbol binario de búsqueda balanceado

$$\begin{aligned} h &= \lceil \log_2(c) \rceil + 1 \\ 2^h &= c \end{aligned}$$

búsqueda en BST

Rotaciones

 $O(1)$

1, 3, 4, 5, 6,

Número de comparaciones necesarias = altura del árbol

- Búsqueda $O(\log(n))$
- Inserción $O(\log(n))$
- Eliminación $O(\log(n))$

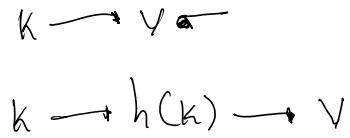
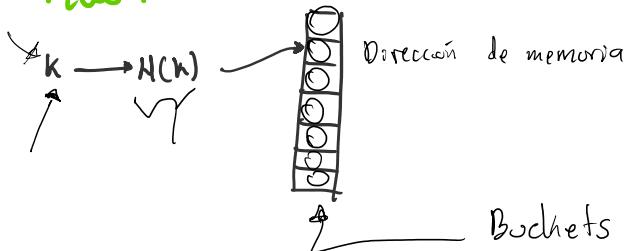
El valor que buscamos en el árbol es la clave de la información que estamos buscando

**Black-red Tree**

Black-red Tree

$\map<K, V>$ myMap = $\{ \{k_1, v_1\}, \{k_2, v_2\}, \dots, \{k_n, v_n\} \}$

Hash table



Función hash

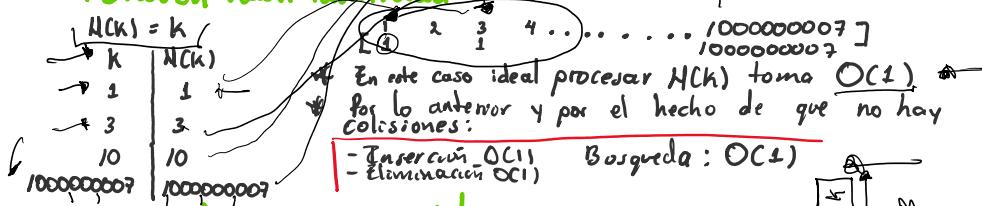
Es una función que mapea una llave K con una dirección de memoria

* Si $k_1 = k_2$, entonces $H(k_1) = H(k_2)$ pero no necesariamente al centro

$$H(k_1) = H(k_2)$$

$$k_1 \neq k_2$$

Función hash idealidad



$H(K) = K$

En este caso ideal procesar $H(K)$ toma $O(1)$. Por lo anterior y por el hecho de que no hay colisiones:

- Insertión: $O(1)$ Búsqueda: $O(1)$

Función hash con módulo

$$H(K) = K \% m$$

$$H(K) = K \% 10$$

$$\begin{array}{|c|c|} \hline K & H(K) \\ \hline 1 & 1 \\ 3 & 3 \\ 7 & 7 \\ 1000000007 & ? \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} \hline & 1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ \hline \end{array}$$

$$abc \quad O(m)$$

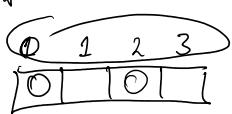
Implementación en C++

unordered_map<K, V> myMap

1: arreglo (La función identidad debe ser viable).

2: unordered_map<K,T> (Debe haber una implementación de hash<key_type>).

3: map<K,T> (Más lento, pero funciona casi con cualquier tipo de dato).



$$H(K) = K \% 1000$$

long long
Storage

$$K \rightarrow \text{hash} \leftarrow S$$

$$\{ K, V \}$$

$$\{ \}$$

Sets

Estructura de datos donde no se aceptan repetidos.

$$\{ 2, 4, 6 \} \neq \{ -1, 0, 1 \}$$

Implementaciones

* Árboles binarios de búsqueda平衡树

* Tablas hash

En un mapa tenemos $\{ K, V \}$

2 Tablas hash

En un mapa tenemos pares $\{k, v\}$
 En un set solo tenemos la clave $\{k\}$

Aplicación: Two sum

Dado un arreglo de enteros $nums$ y un entero $target$, devolver los índices de los dos números en $nums$ que suman $target$.
 Se garantiza que la solución es única y no se puede usar el mismo elemento 2 veces.

$2 \leq n \leq 10^4$

Entrada:

$nums = [2, 7, 15, 11]$ $target = 9$

Salida: $[0, 1]$

$$Map = \{ \{2, 0\}, \{7, 1\}, \{15, 2\}, \{11, 3\} \}$$

$$\text{Suponemos } a = nums[i] \quad a + b = T \\ a + b = c \rightarrow b = c - a \quad b = T - a$$

Verificamos si b existe en el mapa.

Si es así, retornamos $[map[a], map[b]]$

Si NO, repetimos con el siguiente número.

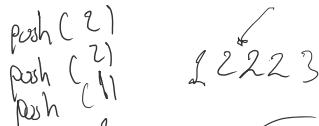
$[0, 1]$

Es una estructura de datos donde los elementos se ordenan automáticamente



Las operaciones soportadas son:

- * empty $O(1)$
- * size $O(1)$
- * top $O(1)$
- * push $O(\log n)$
- * pop $O(\log n)$



3 2 2 2 1

3 2 2 2 1

3 2 2 2 1

3 2 2 2 1

3 2 2 2 1

Implementaciones en C++

- * set<T> \rightarrow Árbol binario de búsqueda balanceado
- * priority_queue<T> \rightarrow Heap

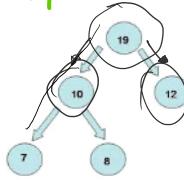
Red-Black

Árbol de búsqueda

AVL

Heap

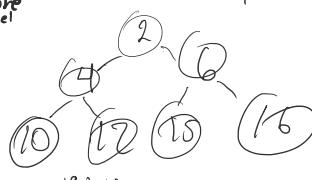
Max heap



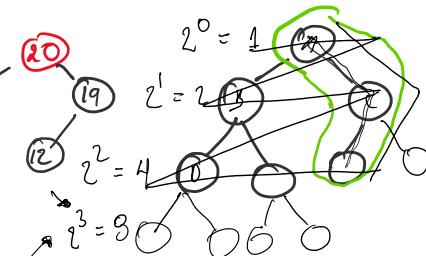
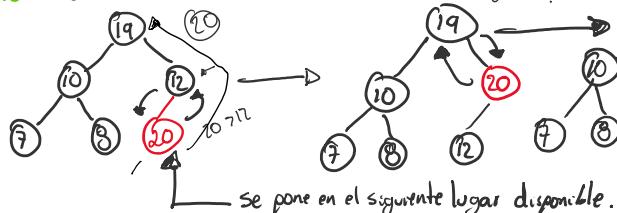
Max heap

Los nodos hijos siempre tendrán valor menor al del padre

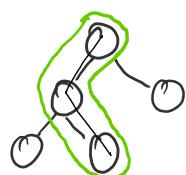
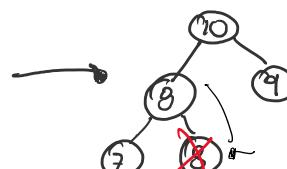
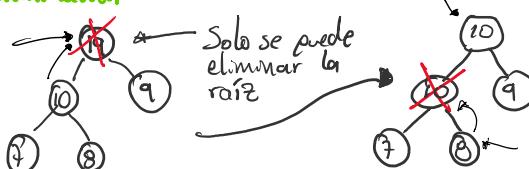
Min heap



Inserción (push)



Eliminación



$\log(k)$

$$2^n = k$$

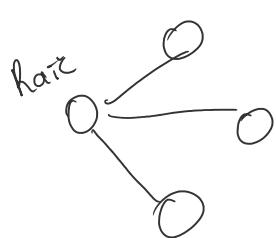
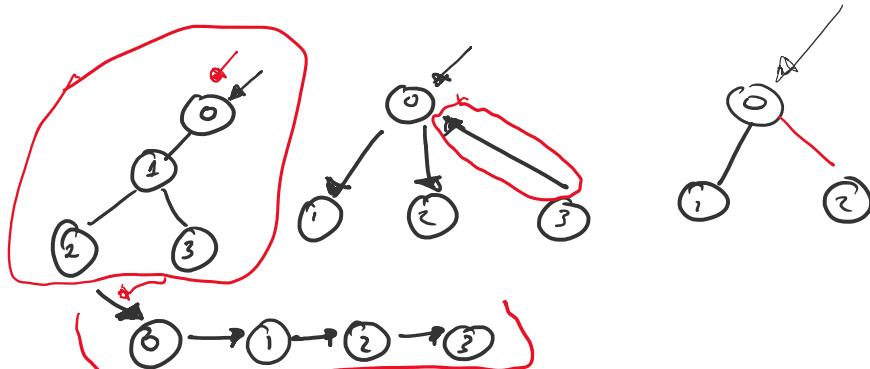
$$n = \log(k)$$

Árboles

martes, enero 24, 2023 5:16 PM

Un árbol es un grafo acíclico conectado

* Se debe poder llegar a todos los nodos desde la raíz



Terminología usada con árboles

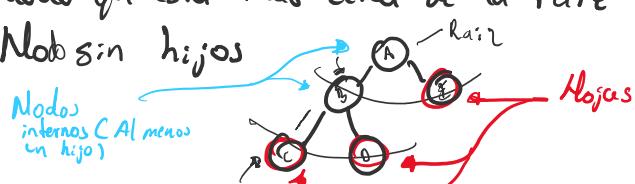
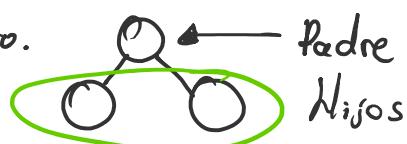
Raíz: El punto de acceso al árbol



Hijo: Nodo que está en nivel más lejos de la raíz desde otro.

Padre: Nodo que está más cerca de la raíz desde otro.

Hoja: Nodo sin hijos

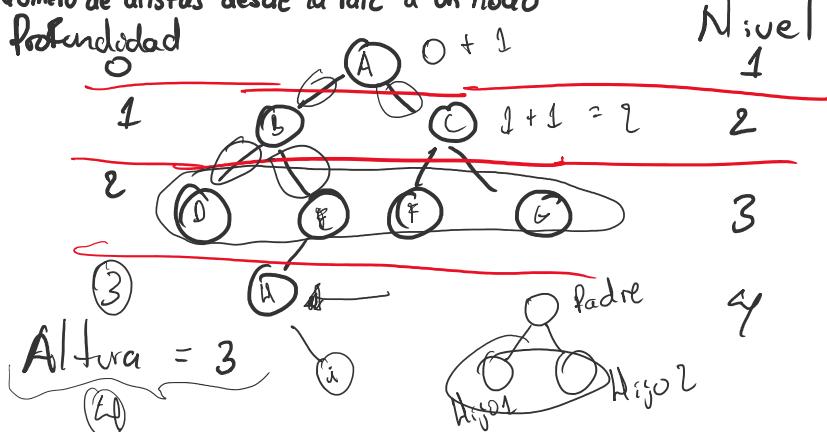


Nivel: Número de aristas recorridas desde la raíz más 1

Altura del árbol: Número de aristas en el camino más largo entre la raíz y una de las hojas.

Profundidad: Número de aristas desde la raíz a un nodo

Nivel
1

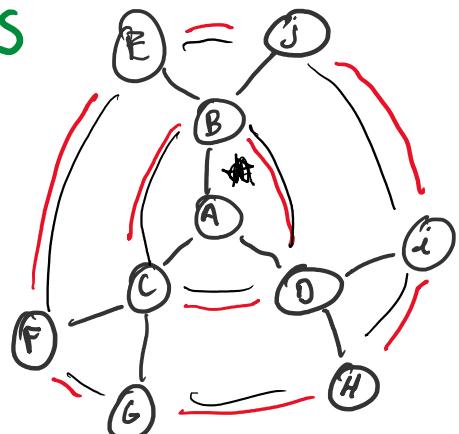


Ejemplo

BFS y DFS

miércoles, enero 25, 2023 3:41 PM

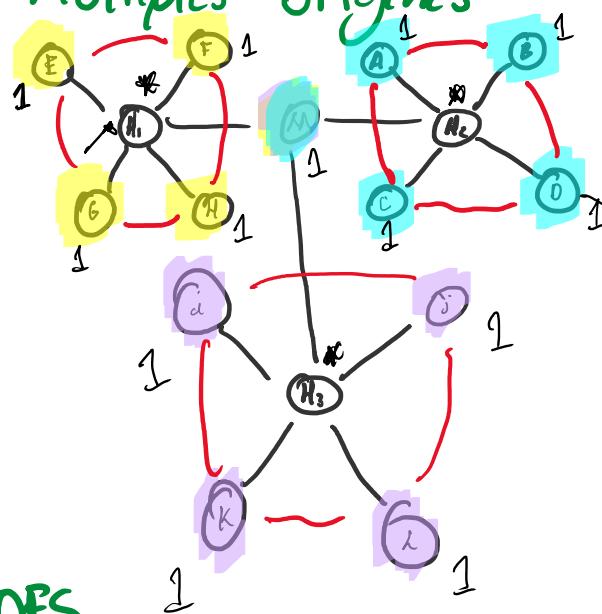
BFS



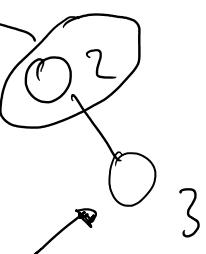
* Para generar este comportamiento se requiere de una cola.



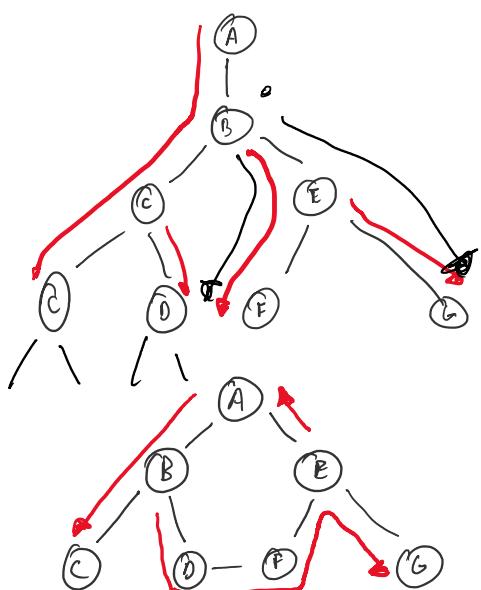
Múltiples orígenes



Para obtener este comportamiento se deben agregar todos los orígenes a la cola al inicio del algoritmo



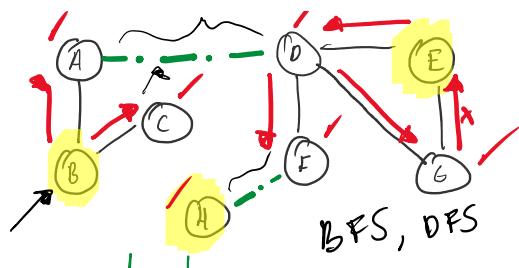
DFS



Aplicación: Deteccción de componentes conexas

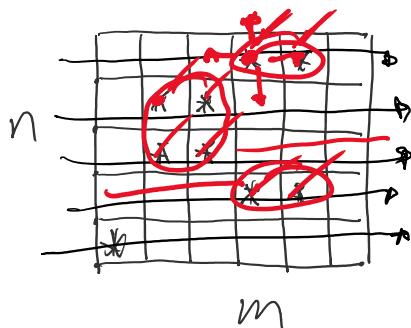


- El número de componentes conexas será igual al número de veces que inicianos un recorrido.



- El número de componentes conexas será igual al número de veces que iniciemos un recorrido.
- El número de puentes necesarios para unir todas las componentes conexas es igual al número de componentes conexas menos 1.
 $n \rightsquigarrow (n-1)$ puente

Problema de las k islas



- Cada isla es una componente conexa.

