BST → Árboles Binarios de Búsqueda

Características de los árboles binarios:

- A lo sumo dos hijos.
- Existen los siguientes tipos:
  - **Árbol Binario Auténtico** → Salvo las hojas, cada nodo tiene dos hijos.
  - **Árbol Binario Completo** → Todos los niveles son completos salvo el último nivel, en el que todos los nodos aparecen a la izquierda.
  - **Árbol Binario Perfecto** → Son auténticos con todas las hojas en el nivel máximo.

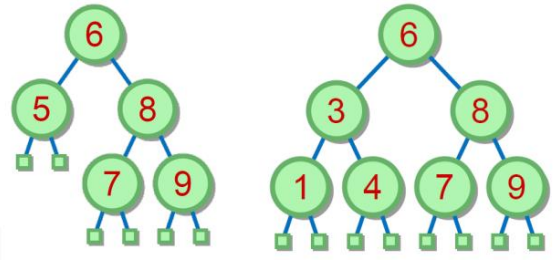
Órdenes de visita más usuales en los árboles binarios:

- Pre-orden → Raíz -> hijo izquierda -> hijo derecha.
- En-Orden (o en profundidad) → hijo izq -> raíz -> hijo dcha
- Post-orden → hijo izq -> hijo dcha -> raíz
- En anchura (o por niveles).

Binary Search Trees (BST)

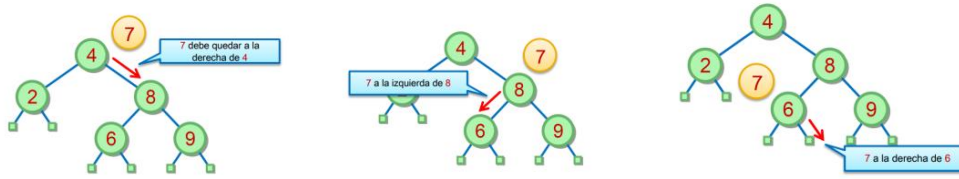
Para cada nodo v:

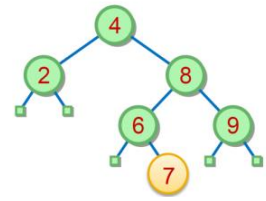
- Todos los elementos del subárbol izquierdo son menores que v.
- Todos los elementos del subárbol derecho son mayores que v.



¿Cómo se insertaría un nuevo elemento en un BST de forma que el resultado sea otro BST?

Si por ejemplo queremos insertar un 7 en el siguiente árbol, debemos tener en cuenta que debe quedar a la derecha de la raíz que es 4 y, a su vez, a la izquierda de 8 y a la derecha de 6.

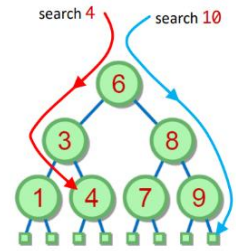




El número de pasos será proporcional a la altura del árbol. O(n) en el peor caso, pero podría conseguirse para cierto tipo de árboles O(log n).

Búsqueda de un elemento en un BST

- Si el número que queremos buscar es menor al nodo en el que estamos, miramos en el subárbol izquierdo (3, 1, 4).
- Si el número que queremos buscar es mayor al nodo en el que estamos, miramos en el subárbol derecho (8, 7, 9).



Elemento mínimo en un BST → se encuentra en la posición más a la izquierda del último nivel.

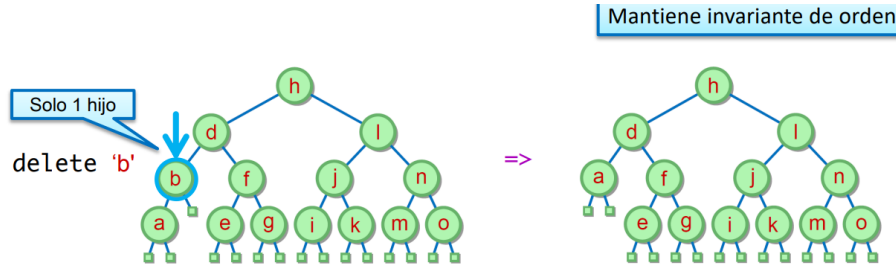
Elemento máximo en un BST → se encuentra en la posición más a la derecha del último nivel.

Eliminación de un elemento en un BST

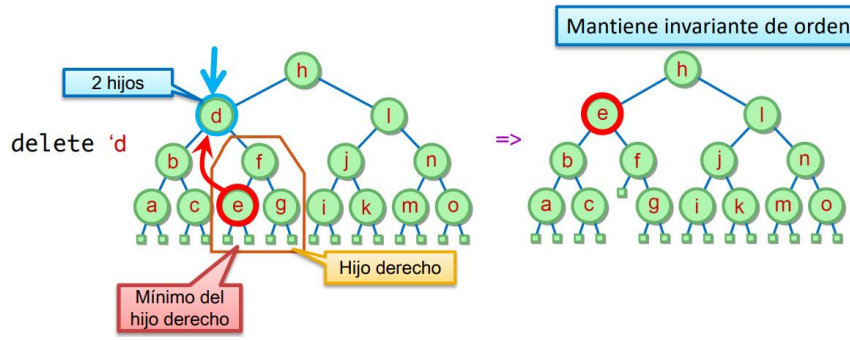
Si el nodo a eliminar es una hoja, se borra y ya.

Si no, localizamos el elemento siguiendo un algoritmo parecido al de inserción.

Si el nodo a eliminar tiene un solo hijo, el nodo padre a eliminar puede conectarse con el nodo hijo.



Si el nodo a borrar tiene dos hijos, el mínimo elemento del hijo derecho sustituirá al elemento a borrar.



De forma alternativa, también podemos usar el máximo elemento del hijo izquierdo para sustituir el elemento a borrar.

IMPORTANTE → Se ve mejor en el código java.

```
@Override
public void delete(K k) {
    root = deleteRec(root,k);
}

private Tree<K,V> deleteRec(Tree<K,V> node, K key) {
    if (node == null) {
        ; //si en nodo es nulo no se hace nada
    } else if (key.compareTo(node.key) == 0) {
        //estamos en el nodo que queremos borrar
        if (node.left == null) {
            node = node.right; //el nodo ahora se convierte en el nodo
de la derecha
        } else if (node.right == null) {
            node = node.left; //el nodo ahora se convierte en el nodo
de la izquierda
        } else { //en el caso de que tenga dos hijos
            node.right = split(node.right, node); //el node se
actualiza con los cambios realizados en split
            //SE ACTUALIZA NODE PORQUE LA CLASE SPLIT ES STATIC !!!!!
        }
        size--;
    } else if (key.compareTo(node.key) < 0) {
        //si la clave es menor a la clave del nodo que queremos borrar
        node.left = deleteRec(node.left,key);
    } else {
        node.right = deleteRec(node.right,key);
    }
    return node;
}

/*
si el nodo a borrar tiene dos hijos:
    --> el mínimo elemento del hijo derecho será el que sustituya al
nodo
 */

//eliminamos el minimo del arbol y lo ponemos donde iria el nodo, por
lo que se guarda la info en temp
private static <K extends Comparable<? super K>,V> Tree<K,V> split
(Tree<K,V> node, Tree<K,V> temp) {
    //si no tiene hijo izquierdo, quiere decir que el menor elemento
es el de la raiz
    if (node.left == null) {
```

```
        //guardamos el valor y la clave en temp y devolvemos el hijo
derecho (eliminamos la raiz)
        temp.key = node.key;
        temp.value = node.value;
        return node.right;
    } else {
        //el minimo estará en el hijo izquierdo
        node.left = split (node.left,temp); //actuailizo node.left
para quitar el minimo
        return node;
    }
}
```