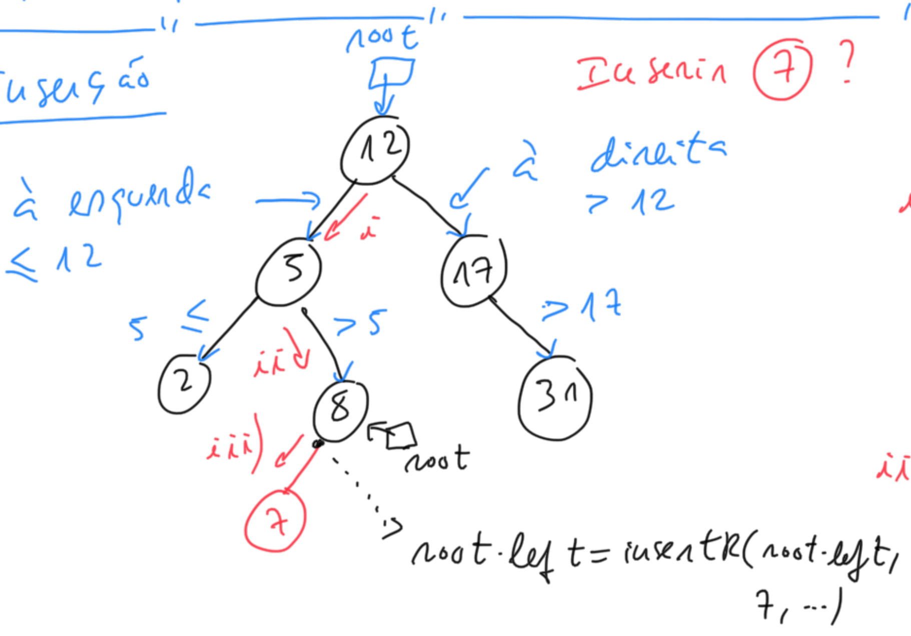
BST - Binary Search Trees

BST são árvores binárias de pesquisa. São estruturas ordenadas (também existem árvores não ordenadas, mas as BST são ordenadas).

$\Rightarrow$ BST permitem realizar pesquisa binária com complexidade temporal $O(\log_2 N)$.

As bases de dados são implementadas usando um tipo especial de árvore ordenada: B-tree.

BST - Inserção



Inserir $\boxed{7}$?

i) $7 < 12$: insere na subárvore esquerda

ii) $7 > 5$: insere na subárvore direita

iii) 8 é folha:
- cria nova folha
- $7 < 8$: a nova folha é subárvore esquerda

BST - Procura

```
class Node<E> {
    var item : E
    var left : Node<E>? = null
    var right : Node<E>? = null
    constructor( elem : E ) {
        item = elem
    }
}
```

```
fun <E> searchR(
        root : Node<E>?, elem : E,
        cmp : Comparator<E>) : Node<E>?
{
        if ( root == null )
            return null
        val c : Int = cmp.compare( elem, root.item )
        // c == 0 , se  elem == root.item
        // c < 0 ,  "     "   <   "
        // c > 0 ,  "     "   >   "
        if ( c == 0 )
            return root
        if ( c < 0 )
            return searchR( root.left, elem, cmp )
        else
            return searchR( root.right, elem, cmp )
}
```

Pesquisa iterativa

```
fun <E> searchIterative( root : Node<E>?, elem : E,
                         cmp : Comparator<E> ) : Node<E>? {

    while (root != null ) {
        val c : cmp.compare( elem, root.item)
        if ( c ==0)
            break
        if (c < 0)
            root = root.left
        else
            root = root.right
    }
    return root
}
```

BST - inserção (recursivo)

```
fun <E> insertR ( root : Node<E>?, elem : E,
                  cmp : Comparator<E>) : Node<E>?
{
        val newNode = Node<E>(elem)
        if (root == null)
            return newNode
      val c = cmp.compare(elem, root.item)
      if (c <= 0)
        root.left = insertR(root.left, elem,
                                     cmp)
      else
        root.right = insertR (root.right,
                              elem, cmp)

     return root
}
```

devolve nova árvore após inserção

```
main:
var tree : Node<Int>?
              = null
                    null
tree = insertR (tree,
                    7, cmp)
```