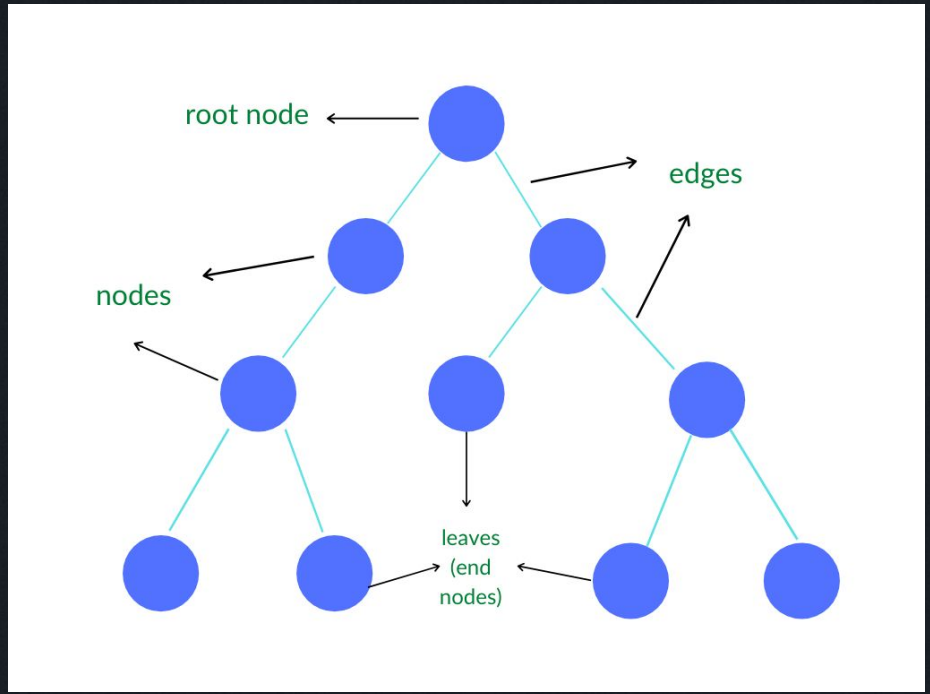
Árvores

Finally, after years of search I found a real tree

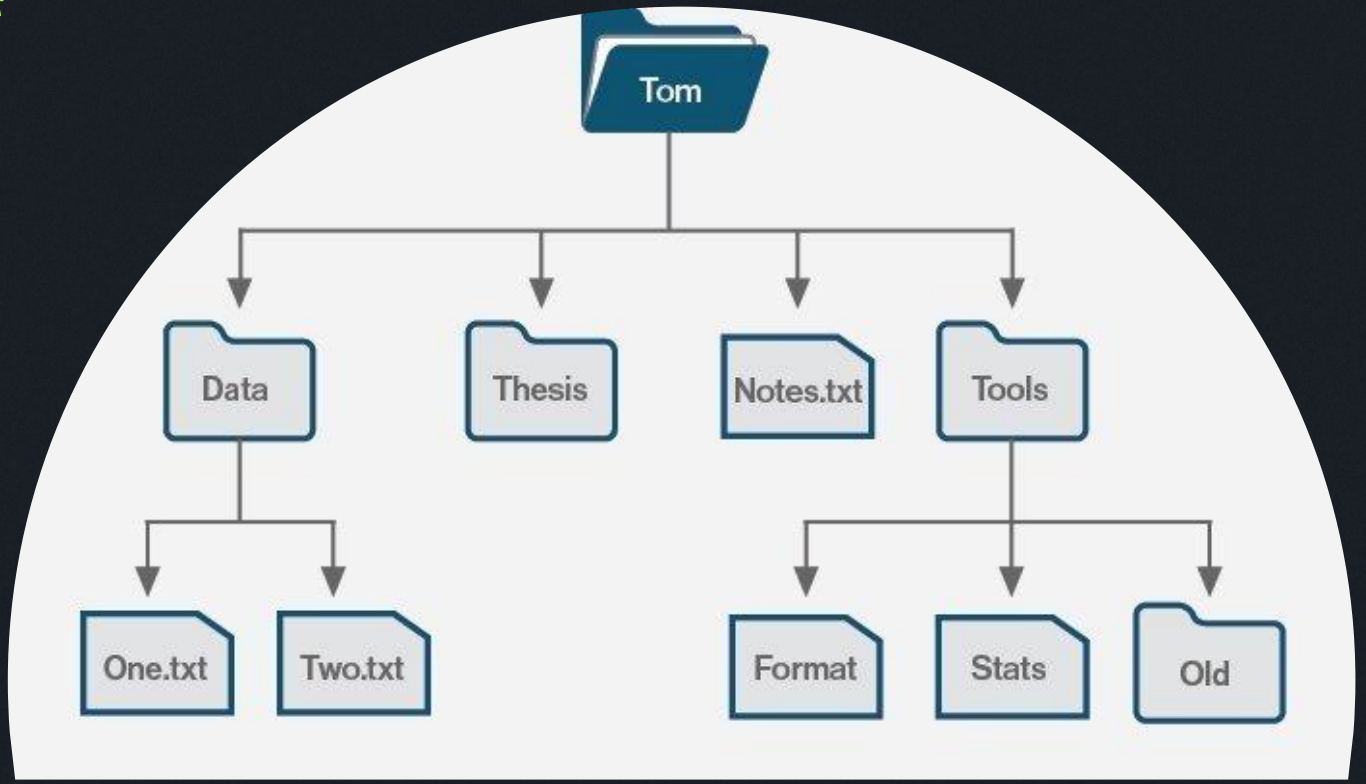
Introdução às Árvores

- Estrutura de dados hierárquicas
- Nós (representando valores) e arestas (ligação entre os nós)
- Componentes:
  - Nodes
  - Root
  - Edges
  - Children nodes
  - Parent Node
  - Leaves

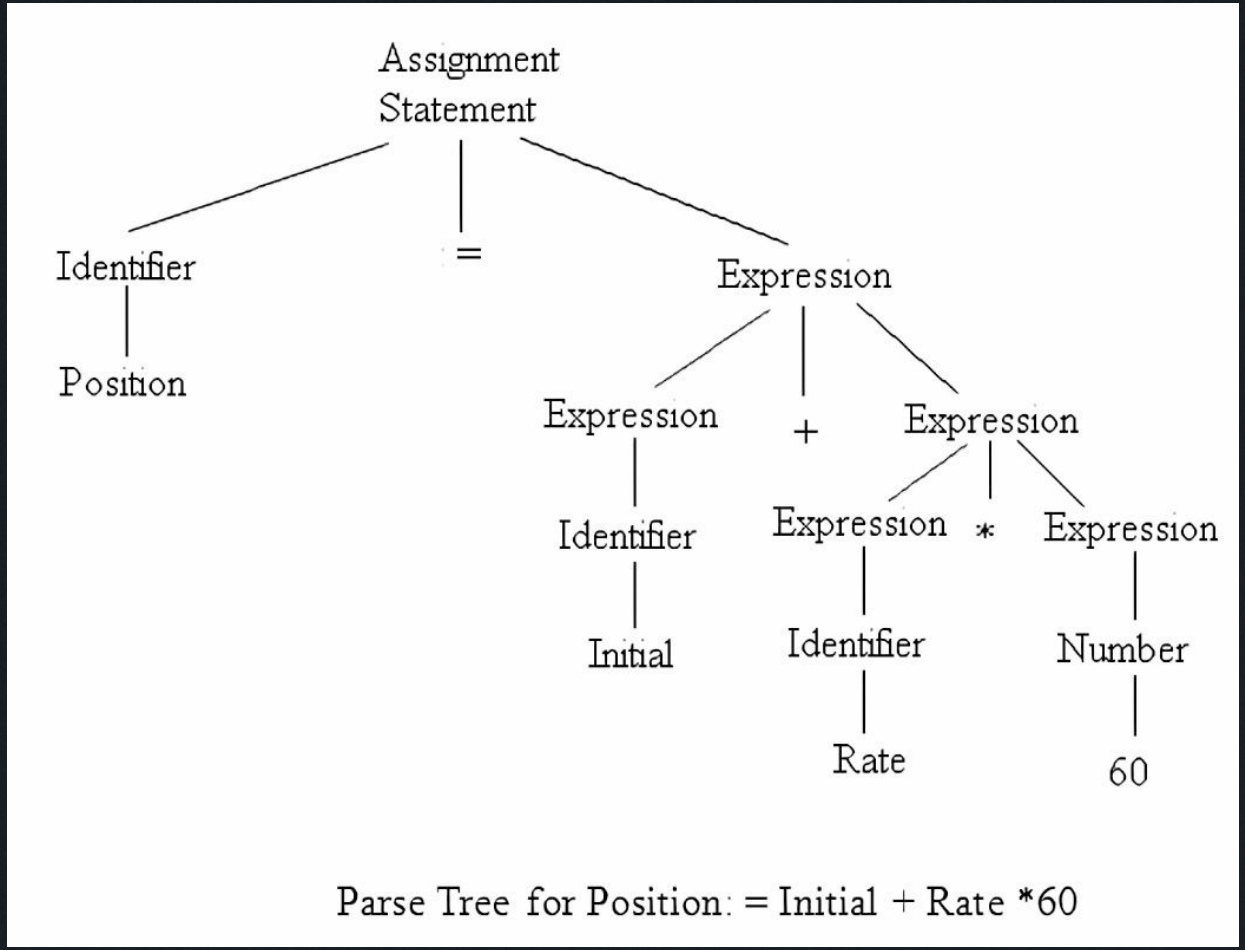


ada

Sistema de Arquivos

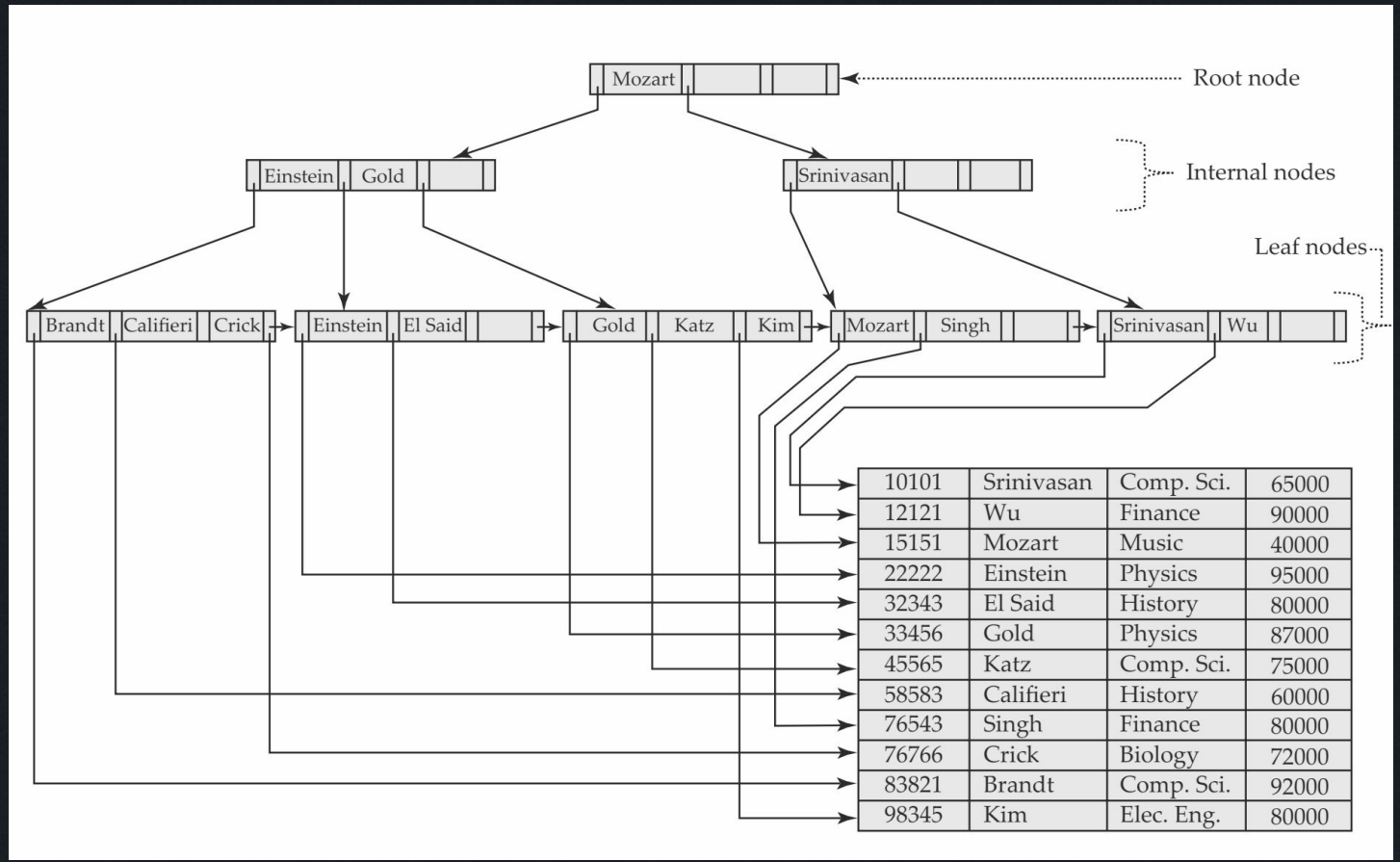


Análise sintática



Parse Tree for Position: = Initial + Rate *60

Banco de dados



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

ada

Implementando uma árvore…

Implementação

```
class Node:
    def __init__(self, val):
        self.val = val
        self.children = [] # Node array
```

Árvores são estruturas inerentemente hierárquicas, em que geralmente os nós são definidos em termos de um valor armazenado e, recursivamente, outros nós filhos

Grau

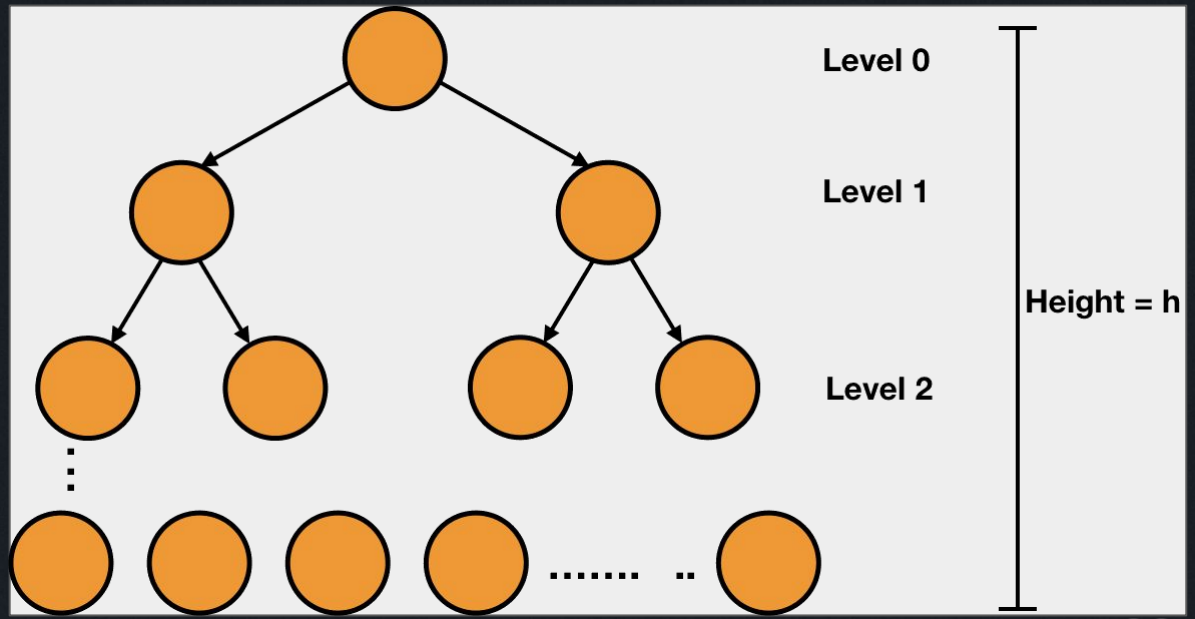
- **Grau de um nó**
  - Define o número de filhos que um nó possui
  - Se um nó tem 3 filhos, seu grau é 3

- **Grau de uma árvore**
  - O grau de uma árvore é o maior grau entre todos os nós da árvore
  - Reflete a máxima ramificação da árvore

Altura

- Número de arestas no caminho mais longo da raiz até uma folha

- Indica o nível máximo de aninhamento da árvore



ada

Exercício

https://leetcode.com/problems/maximum-depth-of-n-ary-tree/

559. Maximum Depth of N-ary Tree

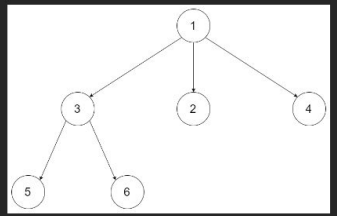
Easy | Topics | Companies

Given a n-ary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

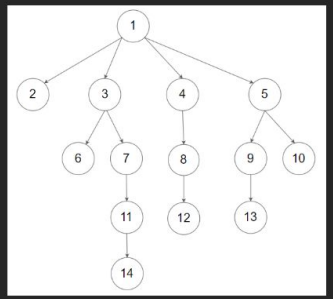
Nary-Tree input serialization is represented in their level order traversal, each group of children is separated by the null value (See examples).

Example 1:




```
Input: root = [1,null,3,2,4,null,5,6]
Output: 3
```

Example 2:




```
Input: root = [1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null,13,null,null,14]
Output: 5
```

Exercício

https://leetcode.com/problems/maximum-depth-of-n-ary-tree/

559. Maximum Depth of N-ary Tree

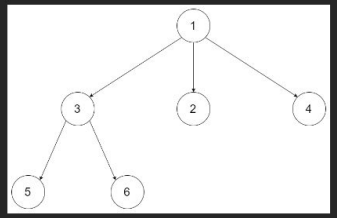
Easy | Topics | Companies

Given a n-ary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

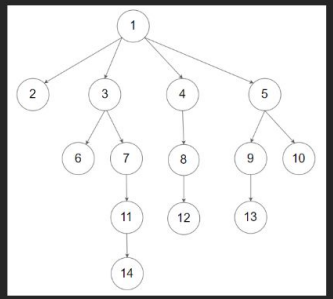
Nary-Tree input serialization is represented in their level order traversal, each group of children is separated by the null value (See examples).

Example 1:




```
Input: root = [1,null,3,2,4,null,5,6]
Output: 3
```

Example 2:




```
Input: root = [1,null,2,3,4,5,null,null,6,7,null,8,null,9,10,null,null,11,null,12,null,13,null,null,14]
Output: 5
```

Estratégia: calcular recursivamente a altura da árvore.

Racional: a altura de qualquer nó **não-folha** vai ser igual ao maior valor da altura de seus filhos + 1.

Caso base: folhas tem altura 1

Para todo nó **não-folha**, calcular a maior altura entre os nós filhos (`max_depth`). Retorna `max_depth + 1`.

ada

Exercício

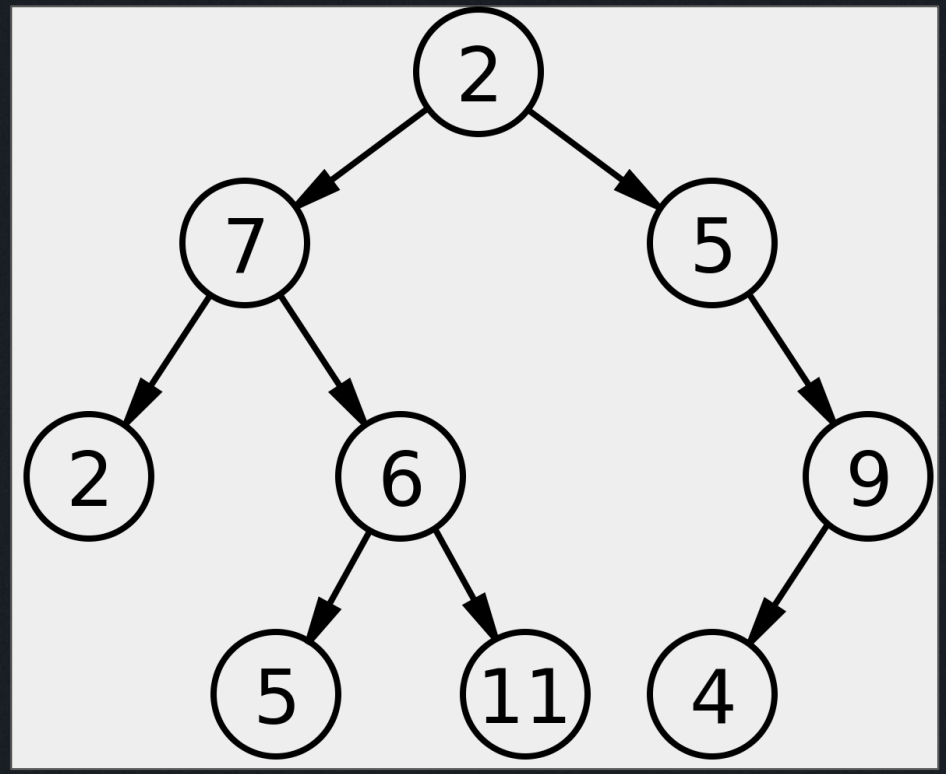
https://leetcode.com/problems/maximum-depth-of-n-ary-tree/

```python
def maxDepth(self, root):
    return self.maxDepthAux(root)

def maxDepthAux(self, root):
    if not root: # árvore vazia
        return 0

    if len(root.children) == 0: # folha
        return 1

    max_depth = 0
    for child in root.children:  # não folha
        depth = self.maxDepthAux(child)   # calcula a altura dos filhos
        max_depth = max(max_depth, depth) # armazena a maior altura entre os filhos

    return max_depth + 1 # retorna a maior altura + 1
```

Árvores Binárias

- Árvore com grau 2
  - Cada nó pode ter no máximo 2 filhos

- Comumente usamos a nomenclatura **left** e **right** para diferenciar os nós filhos de cada nó



ada

Exercício

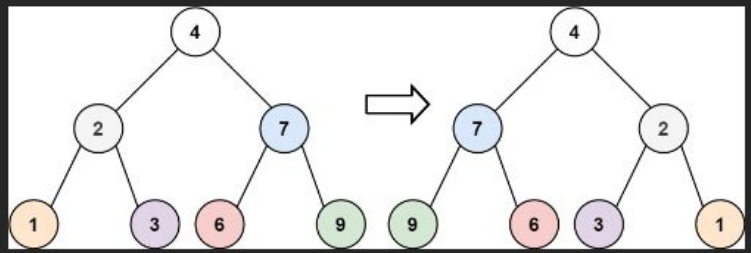
https://leetcode.com/problems/invert-binary-tree/description

226. Invert Binary Tree

Easy | Topics | Companies

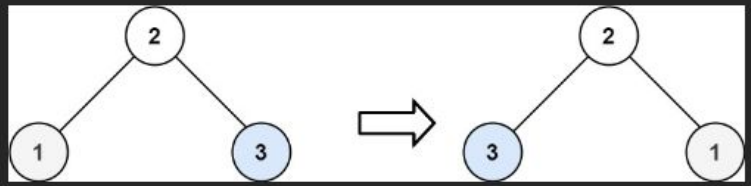
Given the `root` of a binary tree, invert the tree, and return *its root*.

Example 1:




```
Input: root = [4,2,7,1,3,6,9]
Output: [4,7,2,9,6,3,1]
```

Example 2:




```
Input: root = [2,1,3]
Output: [2,3,1]
```

ada

Exercício

https://leetcode.com/problems/invert-binary-tree/description

```
def invertTree(self, root):
    if not root: return None

    root.left, root.right = root.right, root.left

    self.invertTree(root.left)
    self.invertTree(root.right)

    return root
```

Exercício

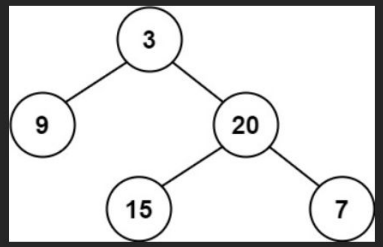
https://leetcode.com/problems/balanced-binary-tree

110. Balanced Binary Tree

Easy | Topics | Companies

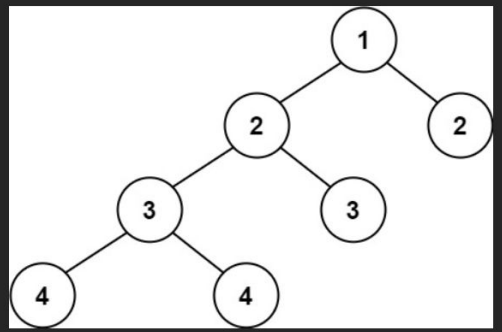
Given a binary tree, determine if it is height-balanced.

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: true
```

Example 2:




```
Input: root = [1,2,2,3,3,null,null,4,4]
Output: false
```

A **height-balanced** binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

ada

Exercício

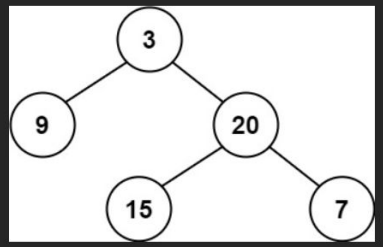
https://leetcode.com/problems/balanced-binary-tree

110. Balanced Binary Tree

Easy | Topics | Companies

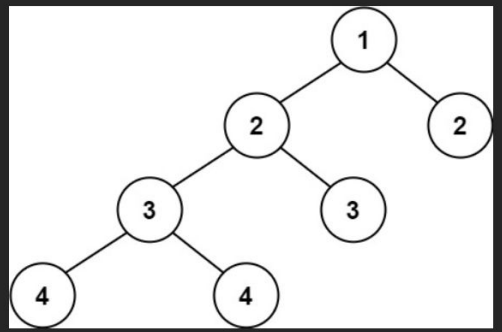
Given a binary tree, determine if it is height-balanced.

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: true
```

Example 2:




```
Input: root = [1,2,2,3,3,null,null,4,4]
Output: false
```

Estratégia: calcular recursivamente se todos os nós estão balanceados

Caso base: um nó nulo está balanceado e tem altura `0`.

Para todo nó **não nulo**, checa se os filhos da esquerda e direita estão balanceados, e armazena as respectivas alturas.

Se algum dos filhos não estiver balanceado, retorna `False`.

Se ambos estão balanceados, retorna `True` e a altura do nó atual (a maior altura entre os nós filhos `+1`.

ada

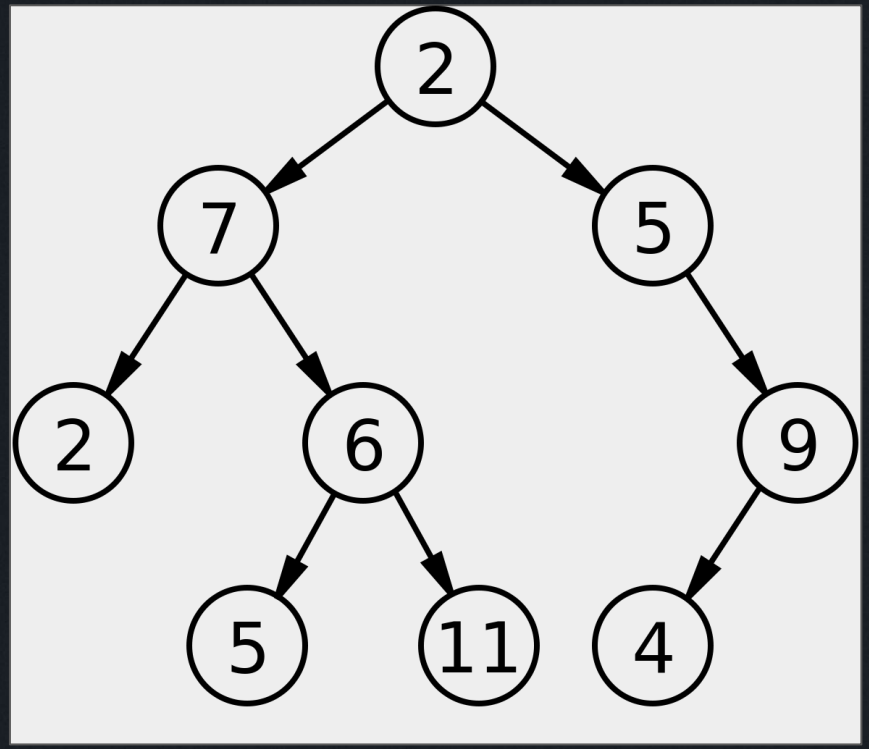
Exercício

https://leetcode.com/problems/balanced-binary-tree

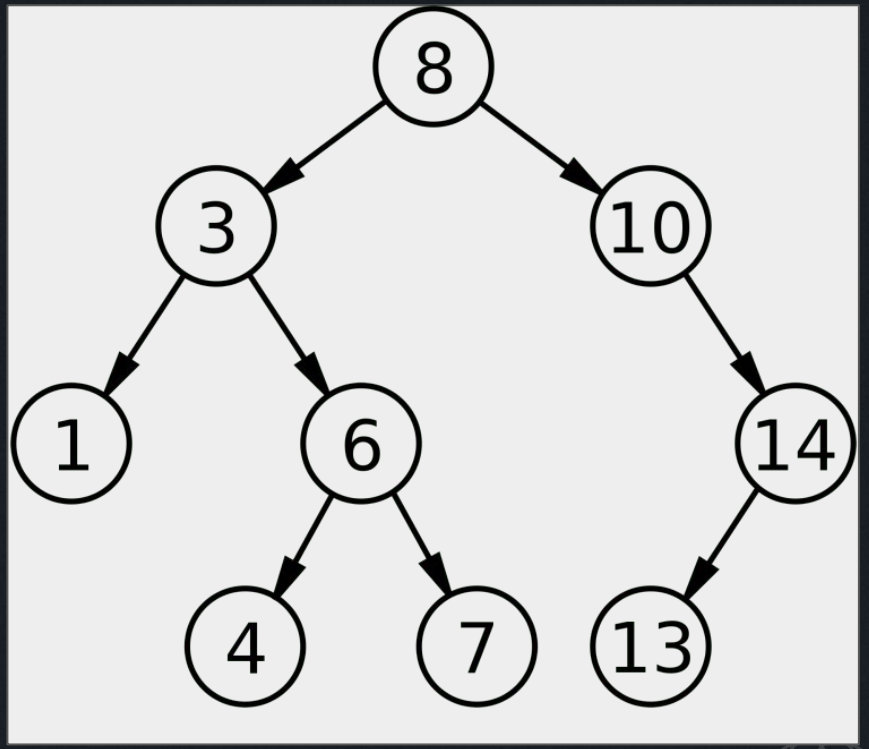
```python
def isBalanced(self, root: Optional[TreeNode]) -> bool:
    ans, _ = self.isBalancedAux(root)
    return ans


# retorna uma tupla (boolean , int)
# boolean: True se o nó estiver balanceado
# int: altura do nó
def isBalancedAux(self, root):
    if not root:
        return (True, 0)

    isLeftBalanced, leftHeight = self.isBalancedAux(root.left)
    isRightBalanced, rightHeight = self.isBalancedAux(root.right)

    rootHeight = max (leftHeight, rightHeight) + 1

    if isLeftBalanced == False or isRightBalanced == False:
        return (False, rootHeight)

    rootIsBalanced = abs(leftHeight - rightHeight) < 2

    return (rootIsBalanced, rootHeight)
```

Binary Tree vs Binary Search Tree (BST)

Não é BST



BST



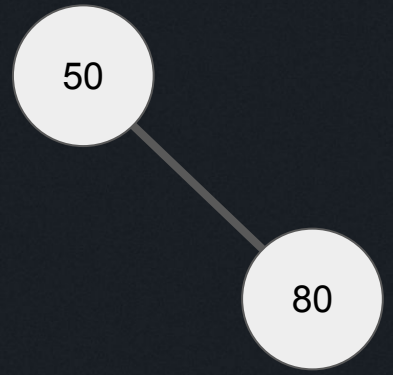
Insert

Insert

```
Insert (80)
```

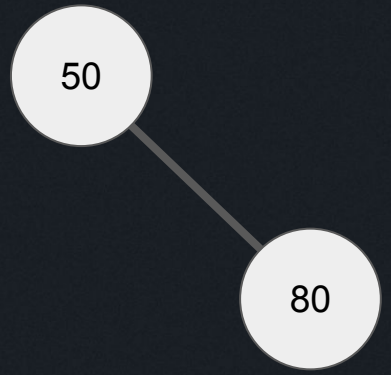
Insert

Insert (80)



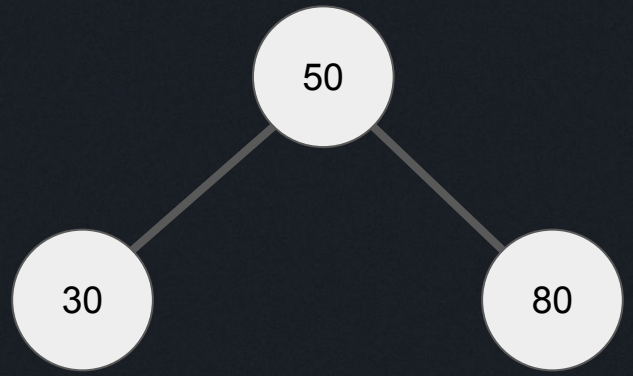
Insert

Insert (30)



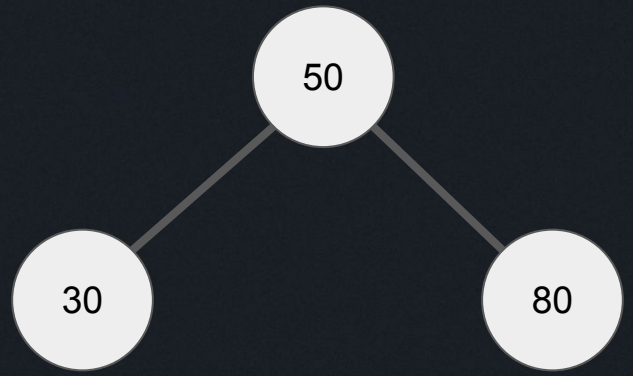
Insert

Insert (30)



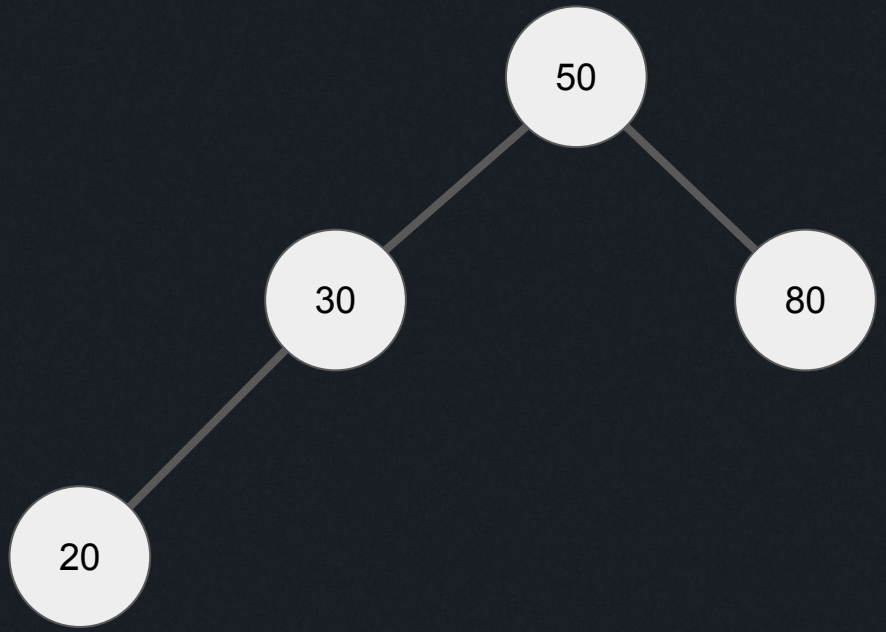
Insert

Insert (20)



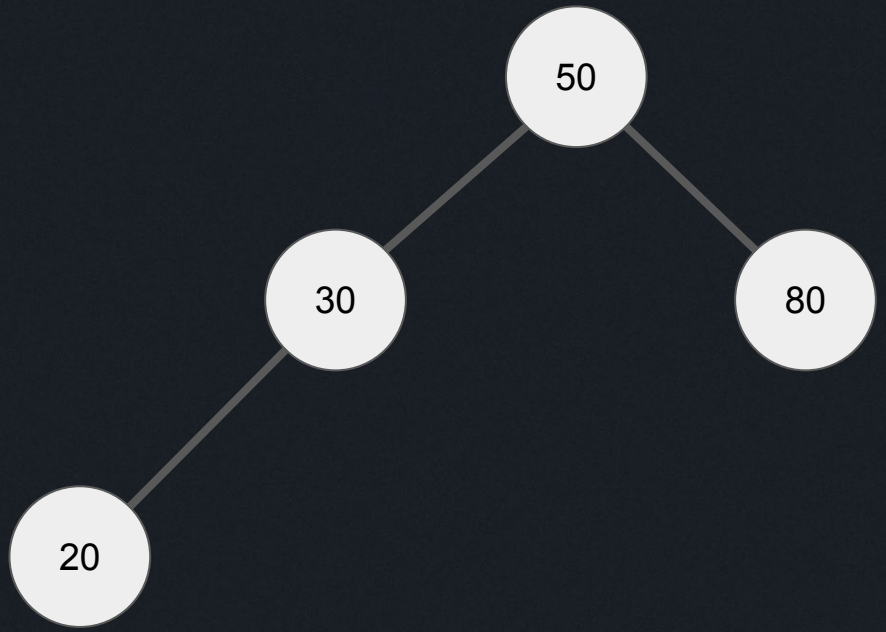
Insert

Insert (20)



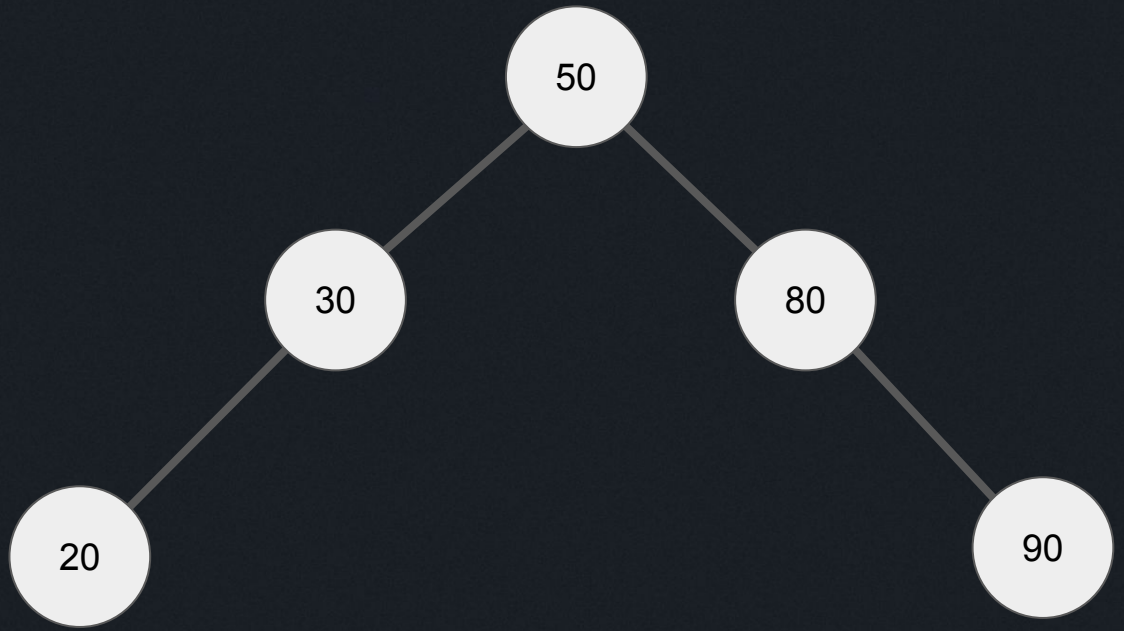
Insert

Insert (90)



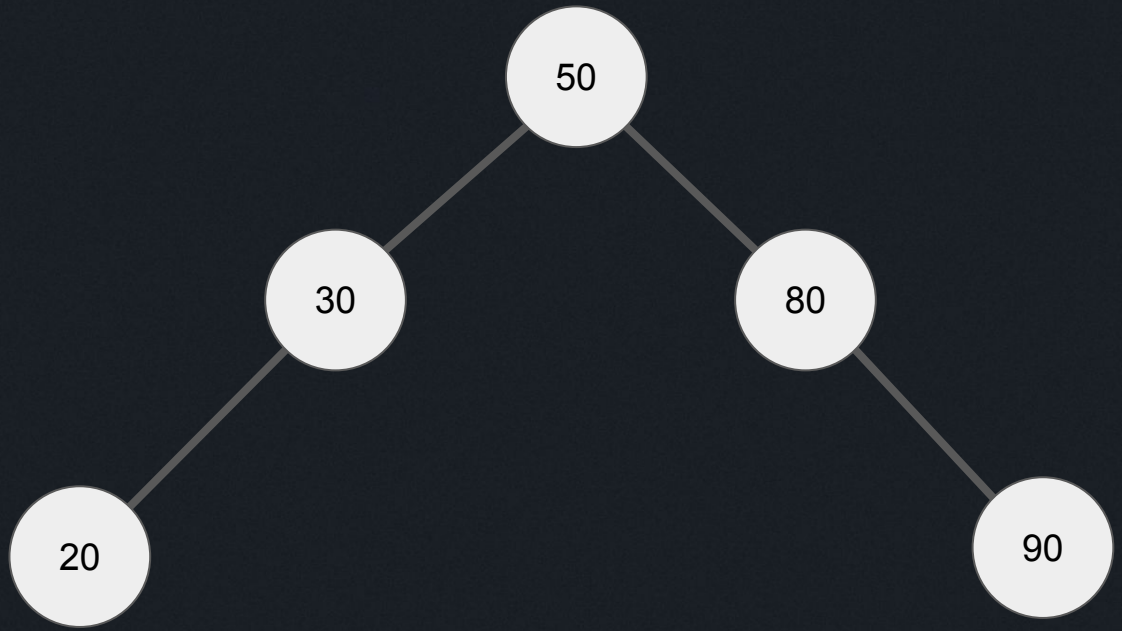
Insert

Insert (90)



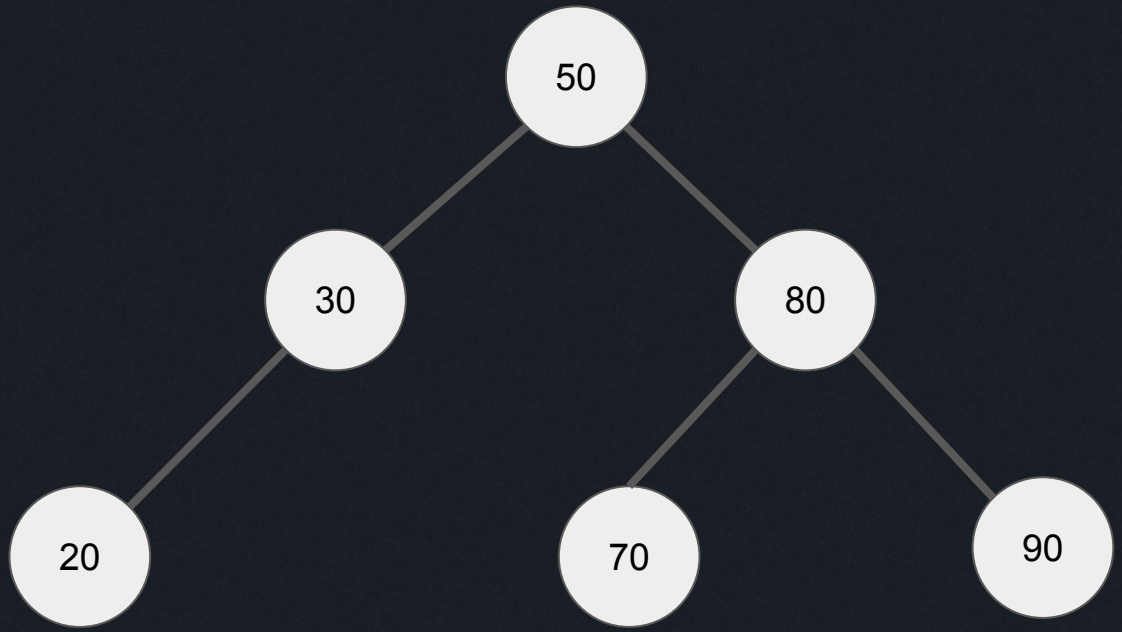
Insert

Insert (70)



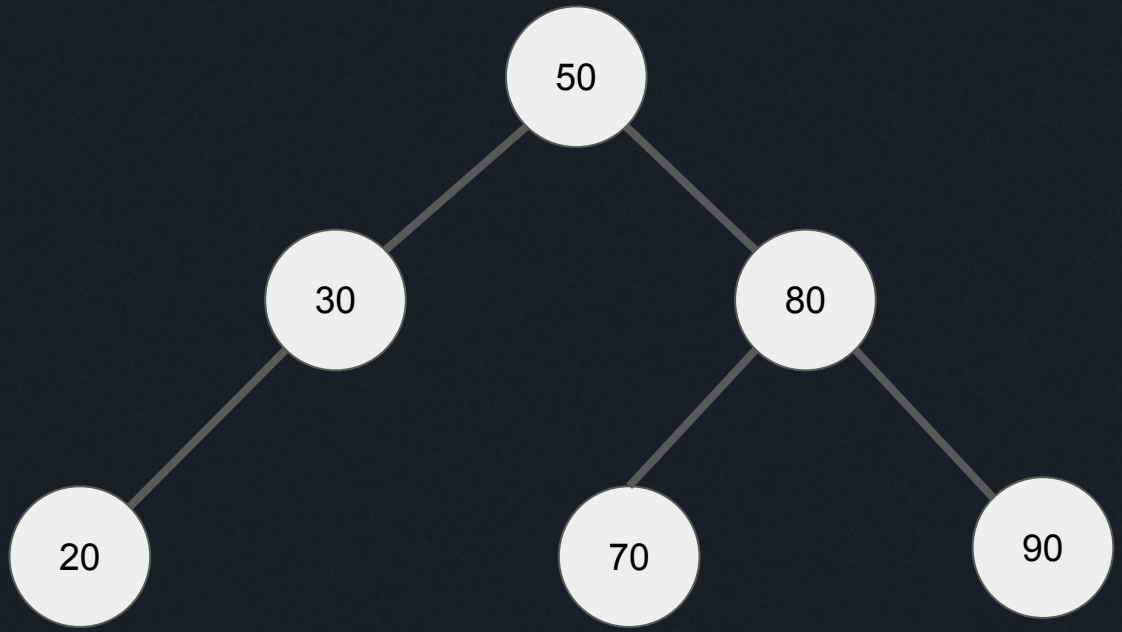
Insert

Insert (70)



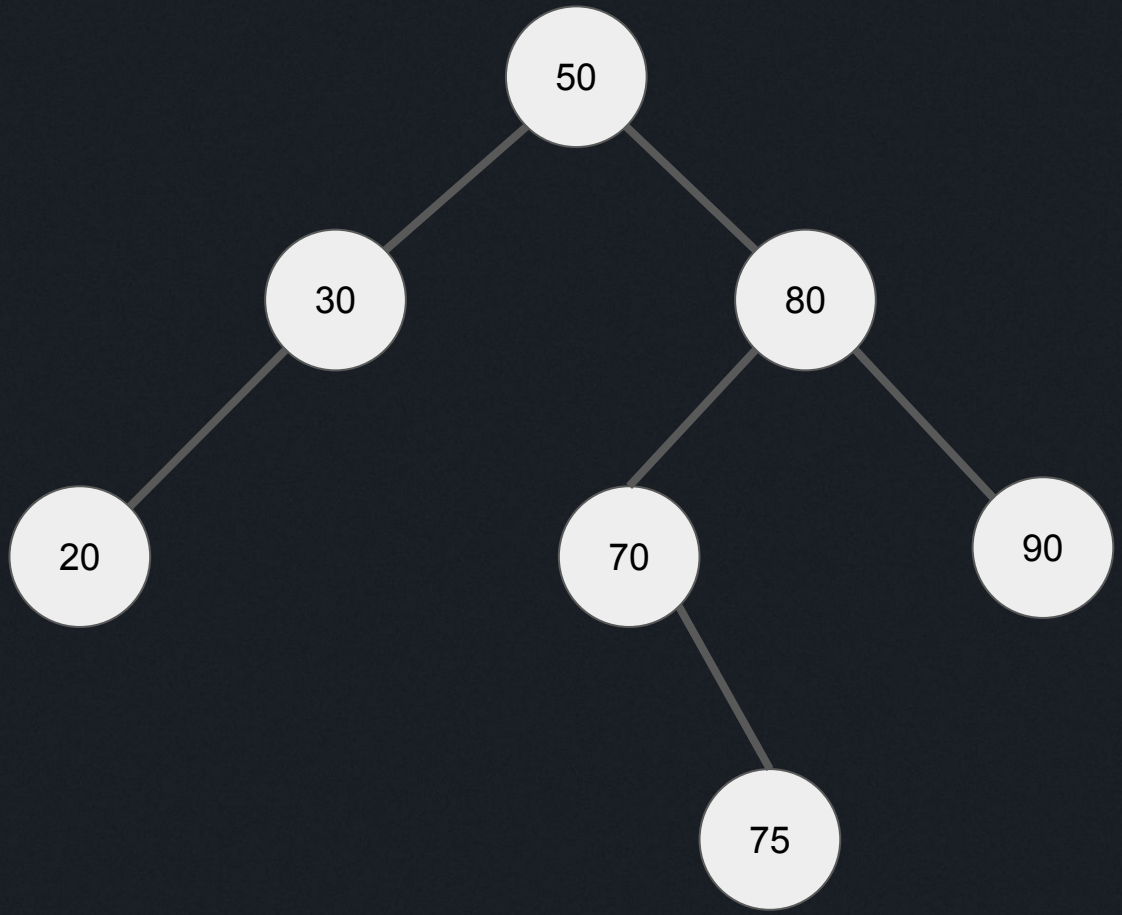
Insert

Insert (75)



Insert

Insert (75)



Insert

https://leetcode.com/problems/insert-into-a-binary-search-tree/

701. Insert into a Binary Search Tree

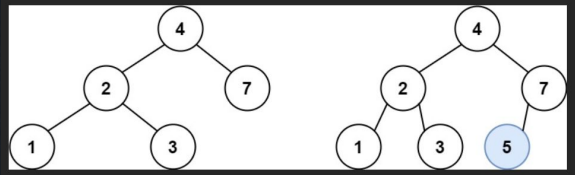
Solved

Medium | Topics | Companies

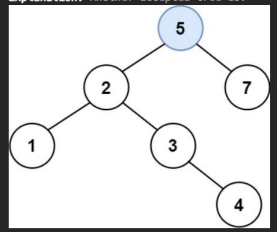
You are given the `root` node of a binary search tree (BST) and a `value` to insert into the tree. Return *the root node of the BST after the insertion*. It is **guaranteed** that the new value does not exist in the original BST.

Notice that there may exist multiple valid ways for the insertion, as long as the tree remains a BST after insertion. You can return **any of them**.

Example 1:




```
Input: root = [4,2,7,1,3], val = 5
Output: [4,2,7,1,3,5]
Explanation: Another accepted tree is:
```



Example 2:

```
Input: root = [40,20,60,10,30,50,70], val = 25
Output: [40,20,60,10,30,50,70,null,null,25]
```

Example 3:

```
Input: root = [4,2,7,1,3,null,null,null,null,null,null], val = 5
Output: [4,2,7,1,3,5]
```

Constraints:

- The number of nodes in the tree will be in the range `[0, 10^4]`.
- `-10^8 <= Node.val <= 10^8`
- All the values `Node.val` are **unique**.
- `-10^8 <= val <= 10^8`
- It's **guaranteed** that `val` does not exist in the original BST.

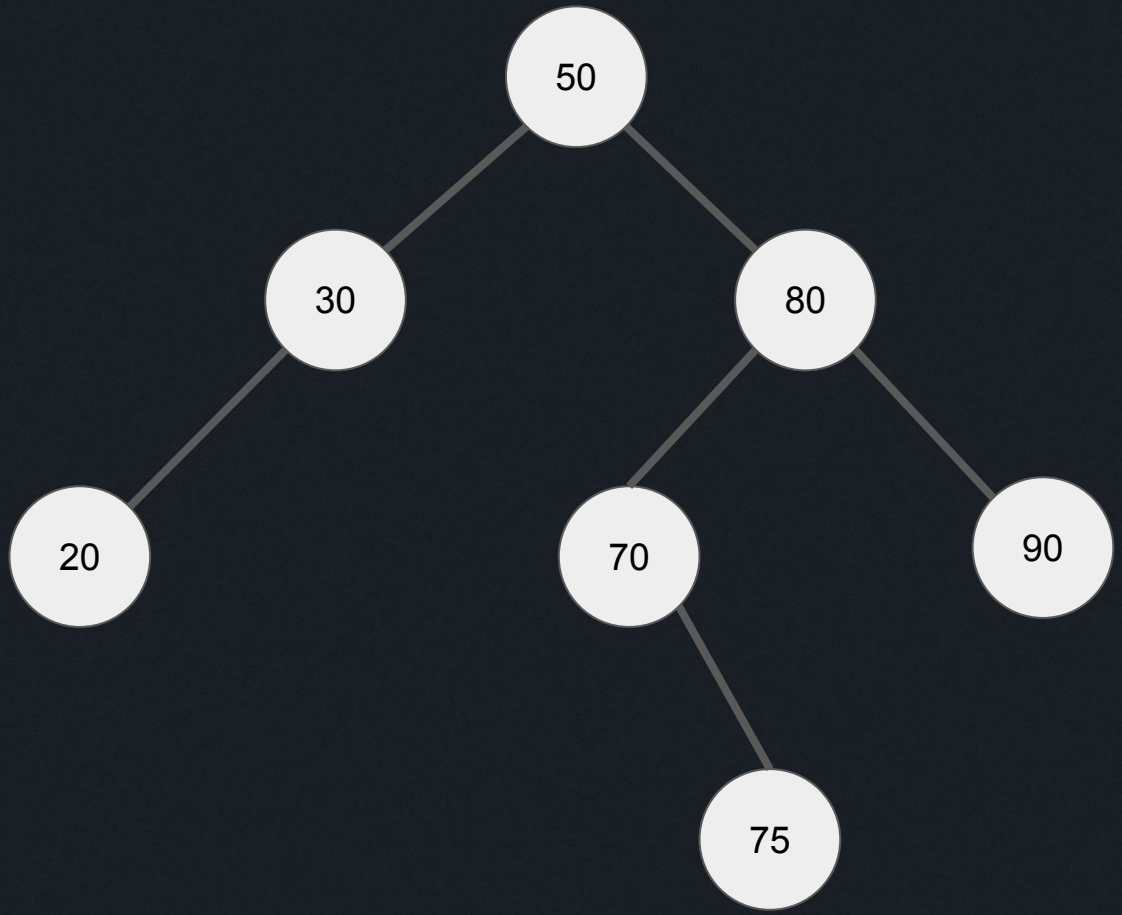
ada

Insert

https://leetcode.com/problems/insert-into-a-binary-search-tree/

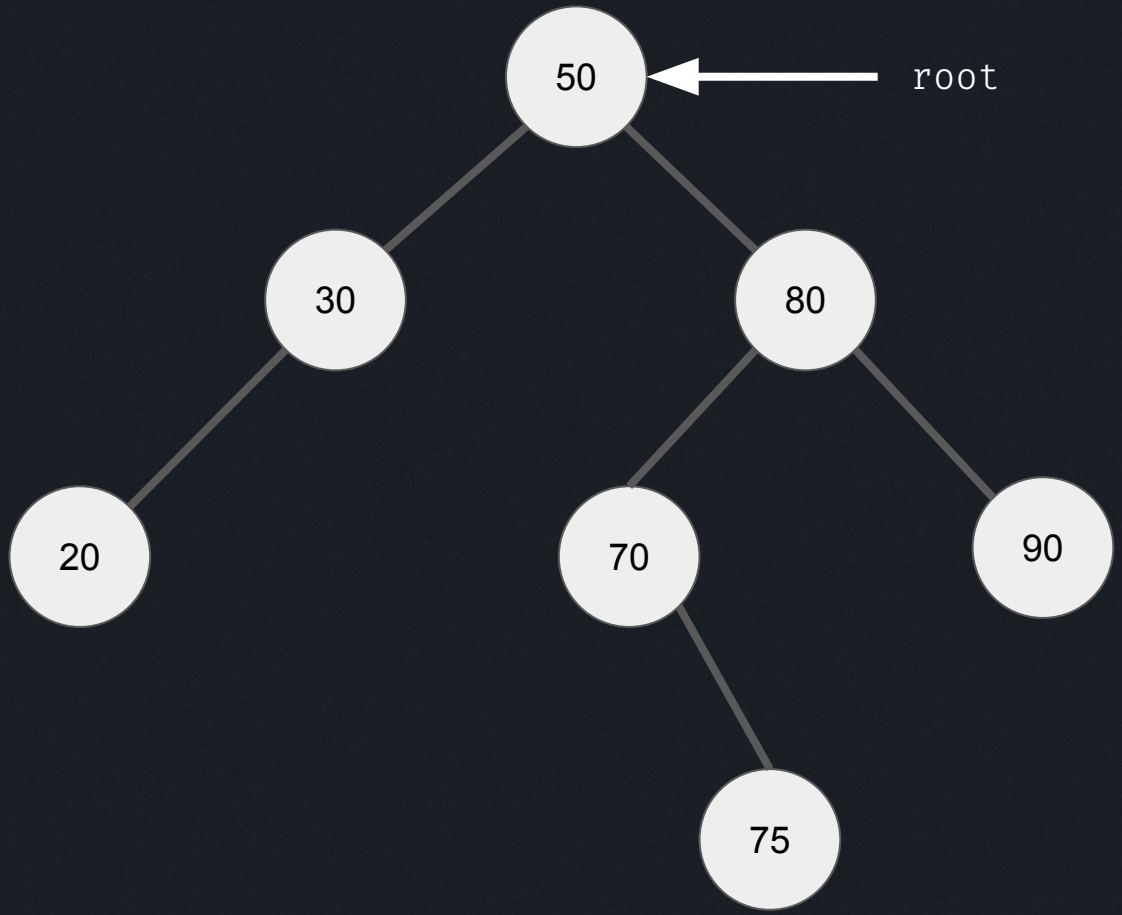
```
class Solution:
    def insertIntoBST(self, root, val):
        if not root:
            return TreeNode(val)

        if val > root.val:
            root.right = self.insertIntoBST(root.right, val)
        else:
            root.left = self.insertIntoBST(root.left, val)

        return root
```

Search



Search

Search (75)

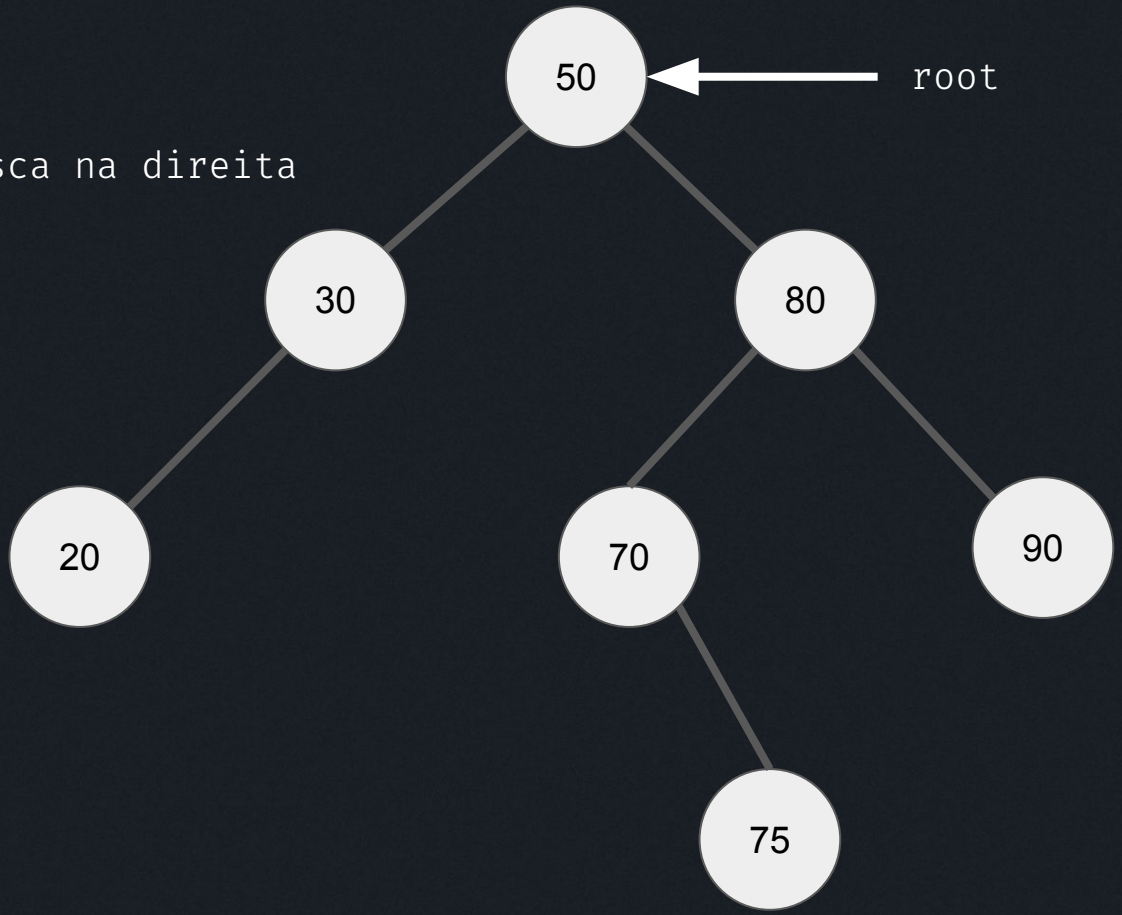


ada

Search

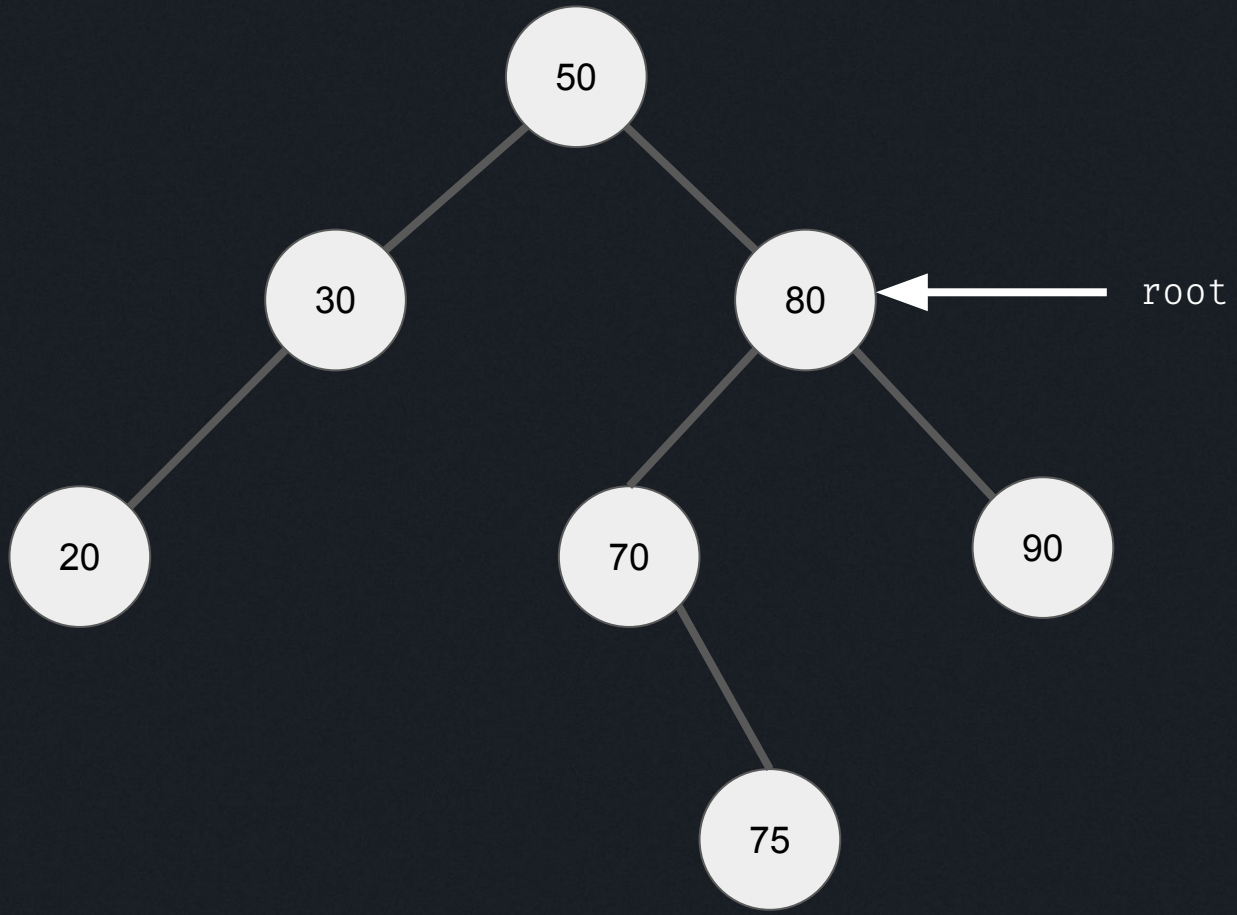
Search (75)

75 > 50, busca na direita



Search

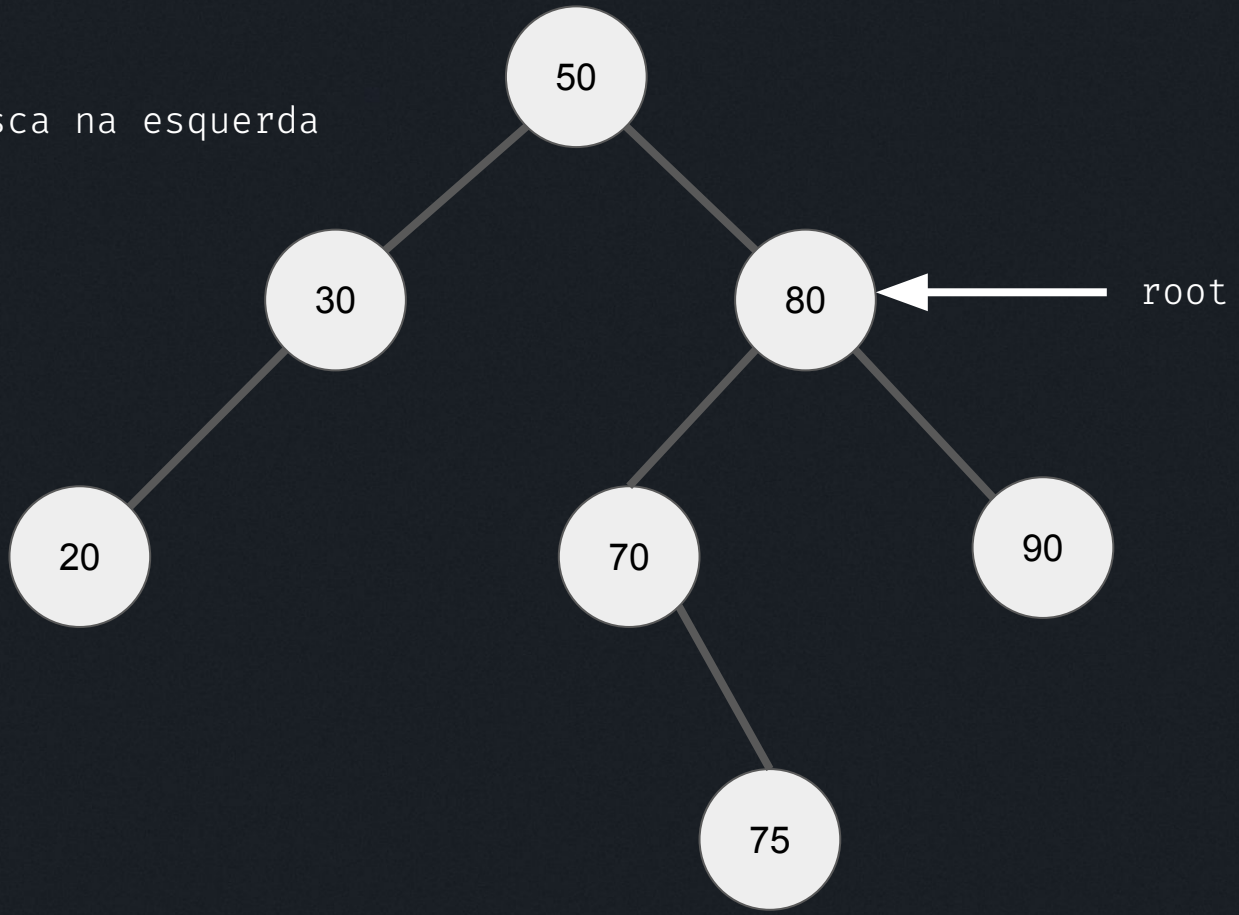
Search (75)



ada

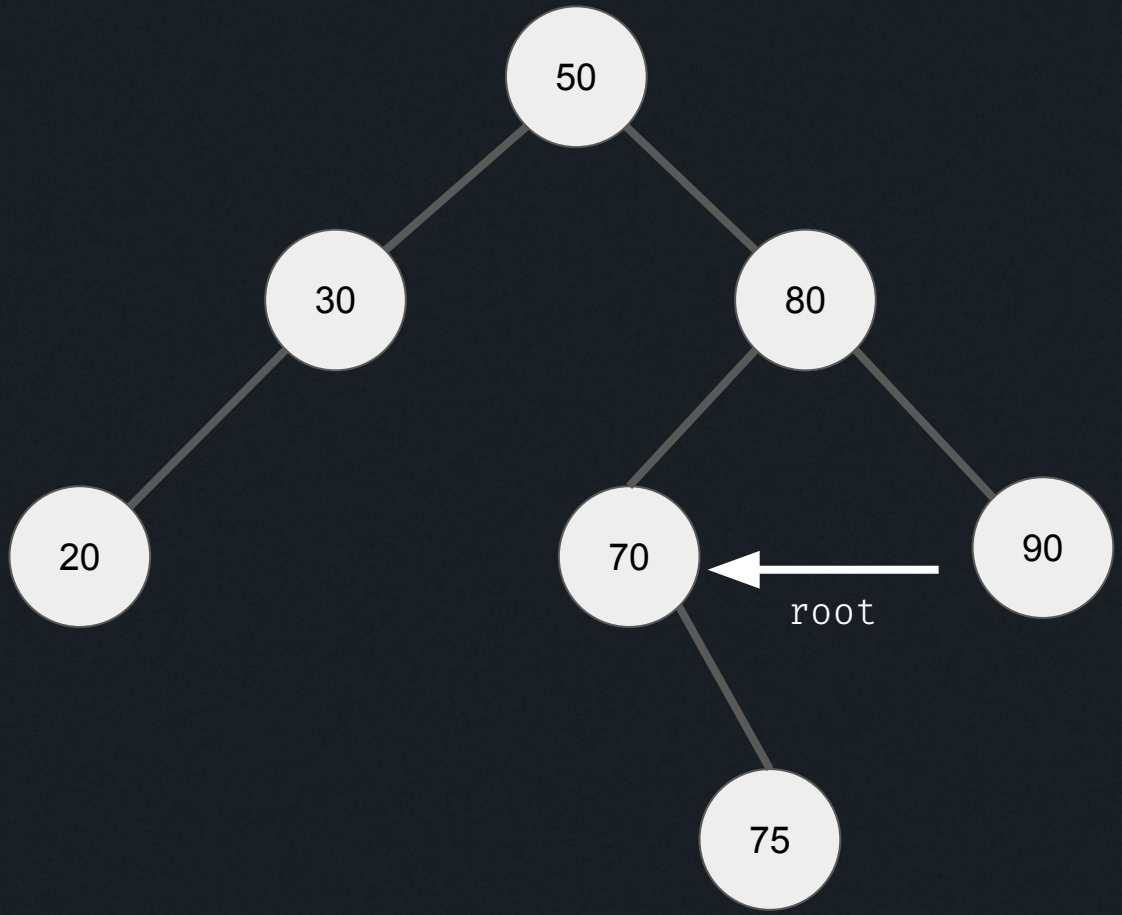
Search

Search (75)
75 < 80, busca na esquerda



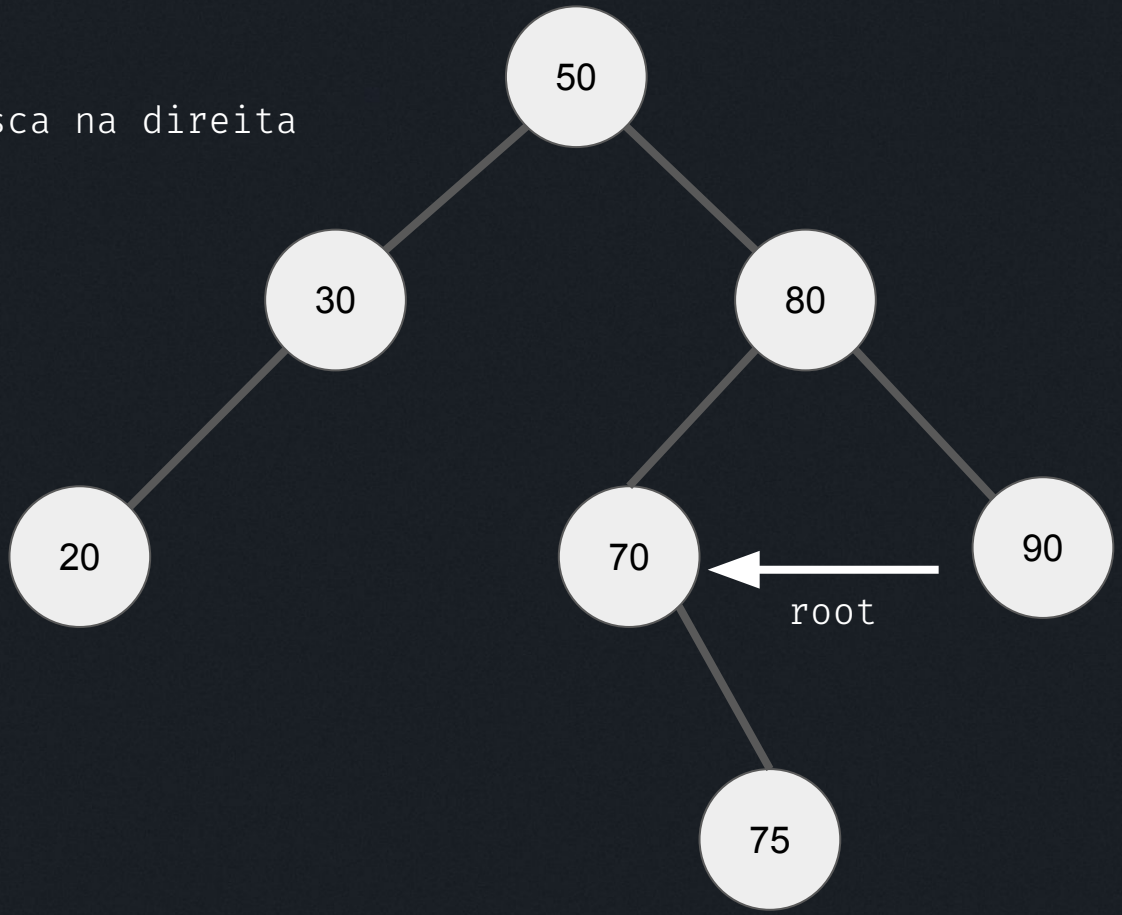
Search

Search (75)



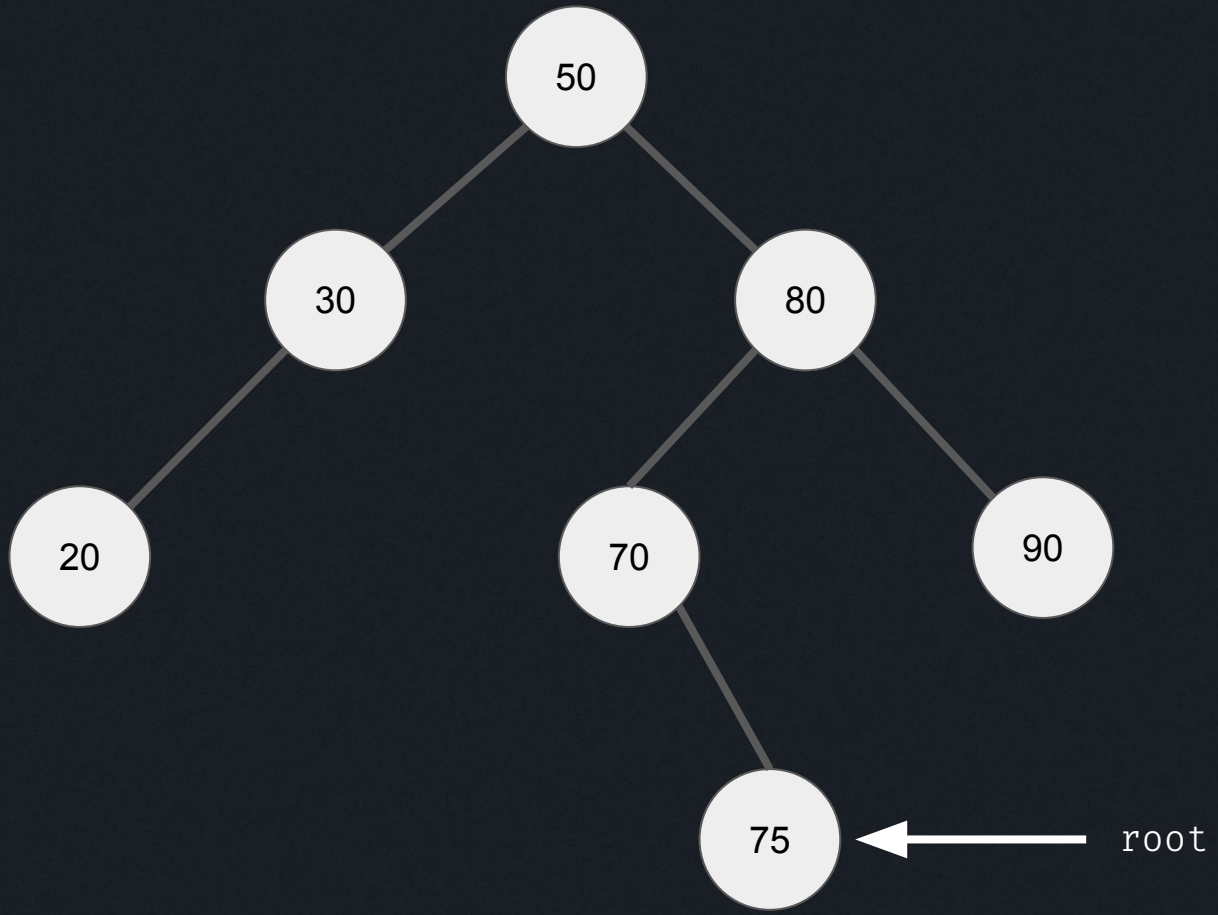
Search

Search (75)
75 > 70, busca na direita



Search

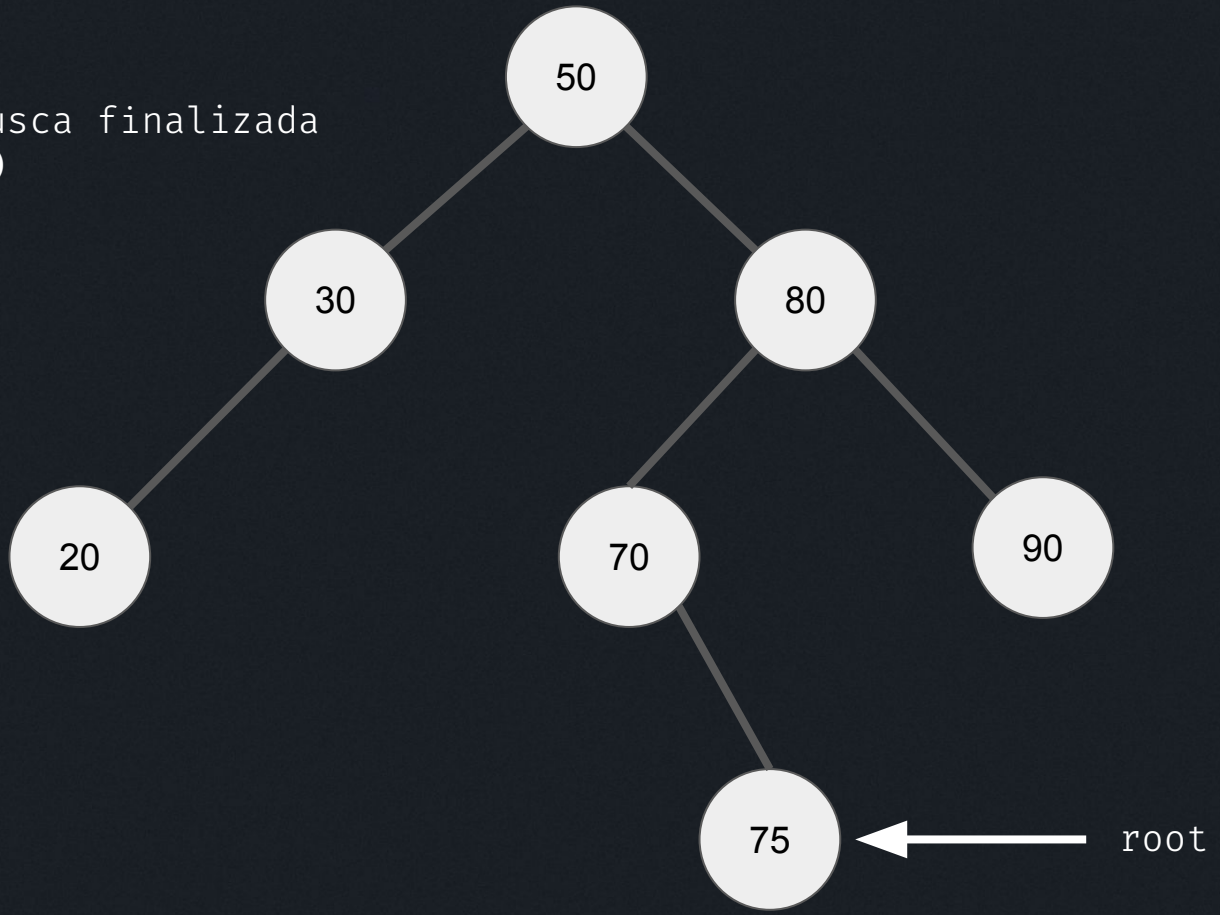
Search (75)



ada

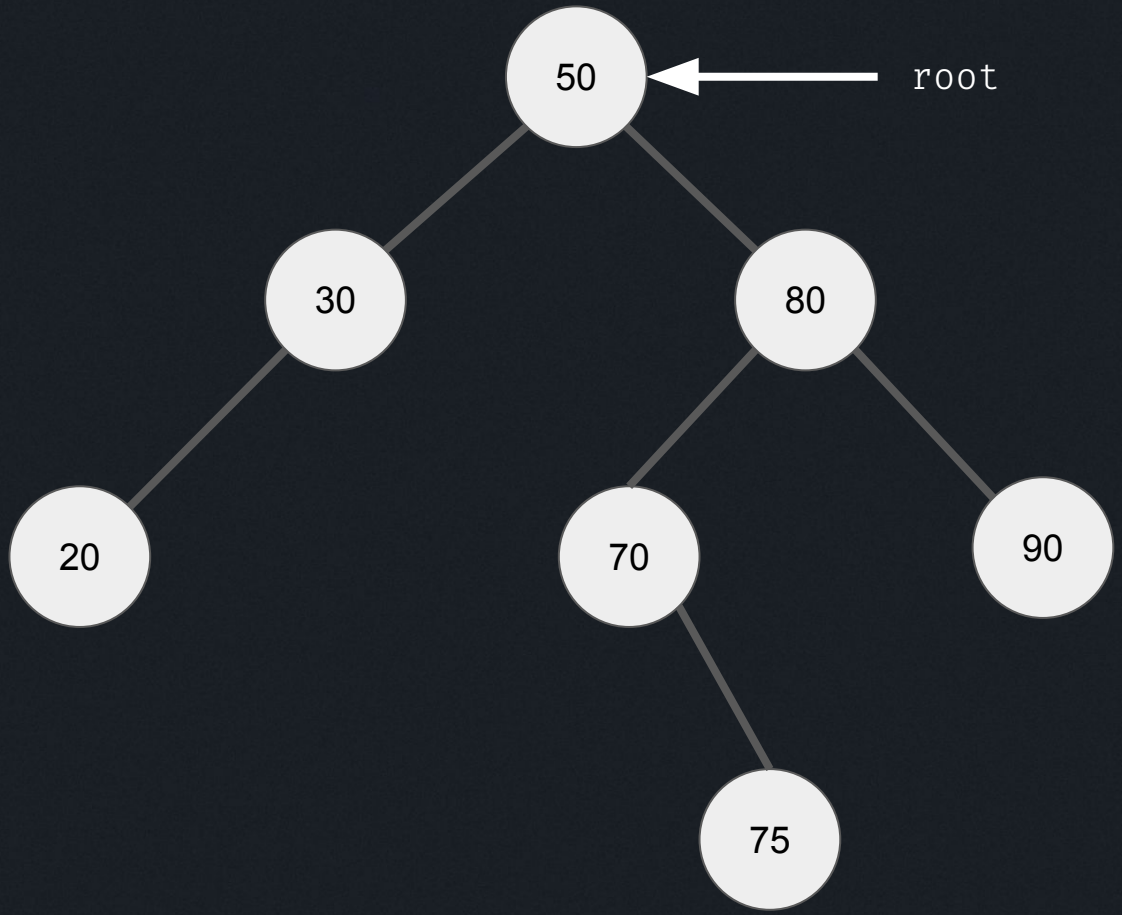
Search

```
Search (75)
75 == 75, busca finalizada
(encontrado)
```



Search

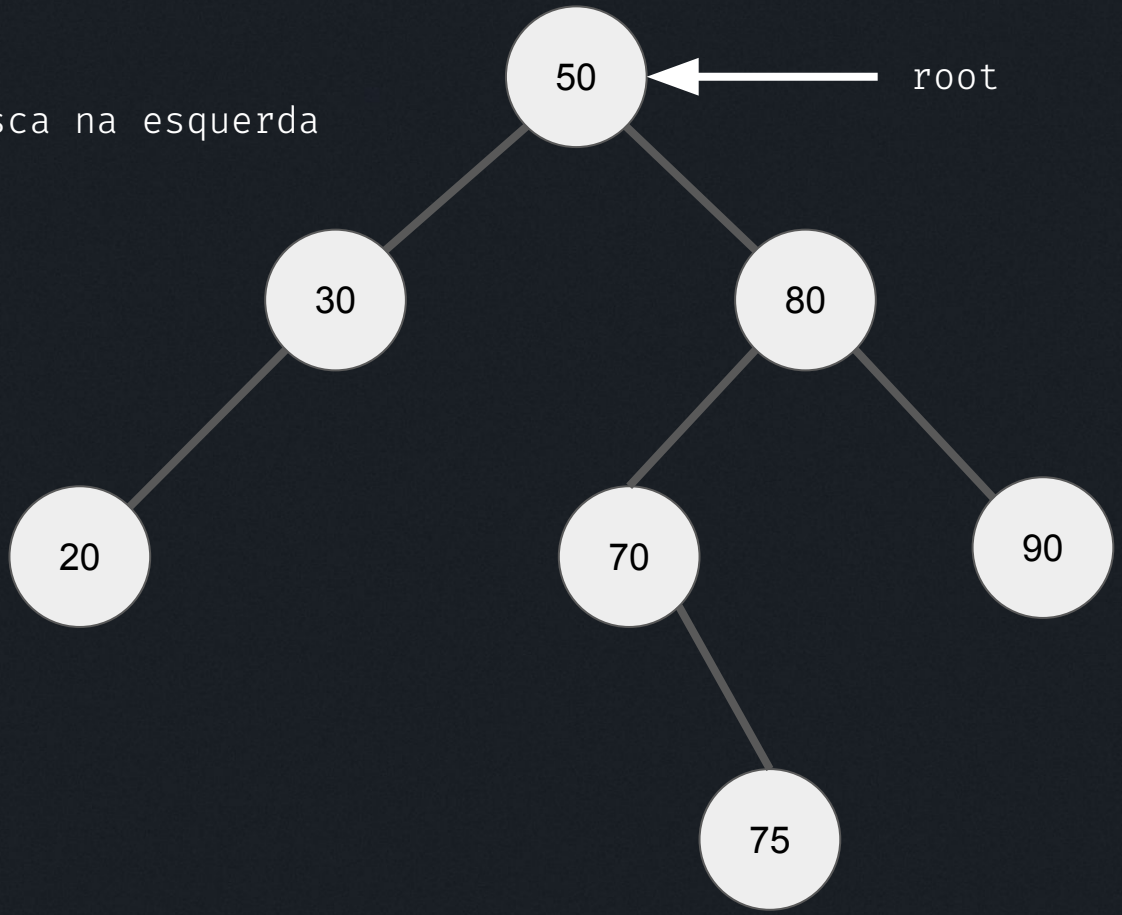
Search (15)



ada

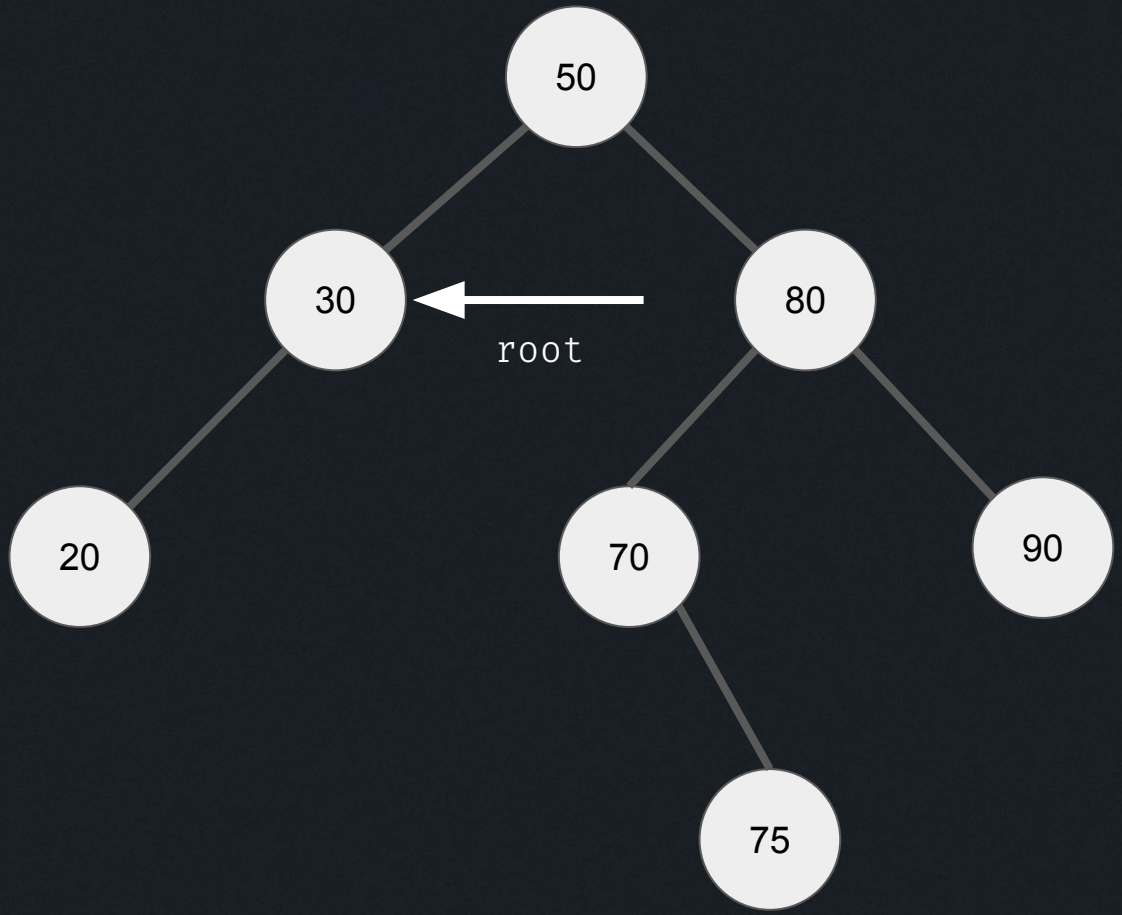
Search

Search (15)
15 < 50, busca na esquerda



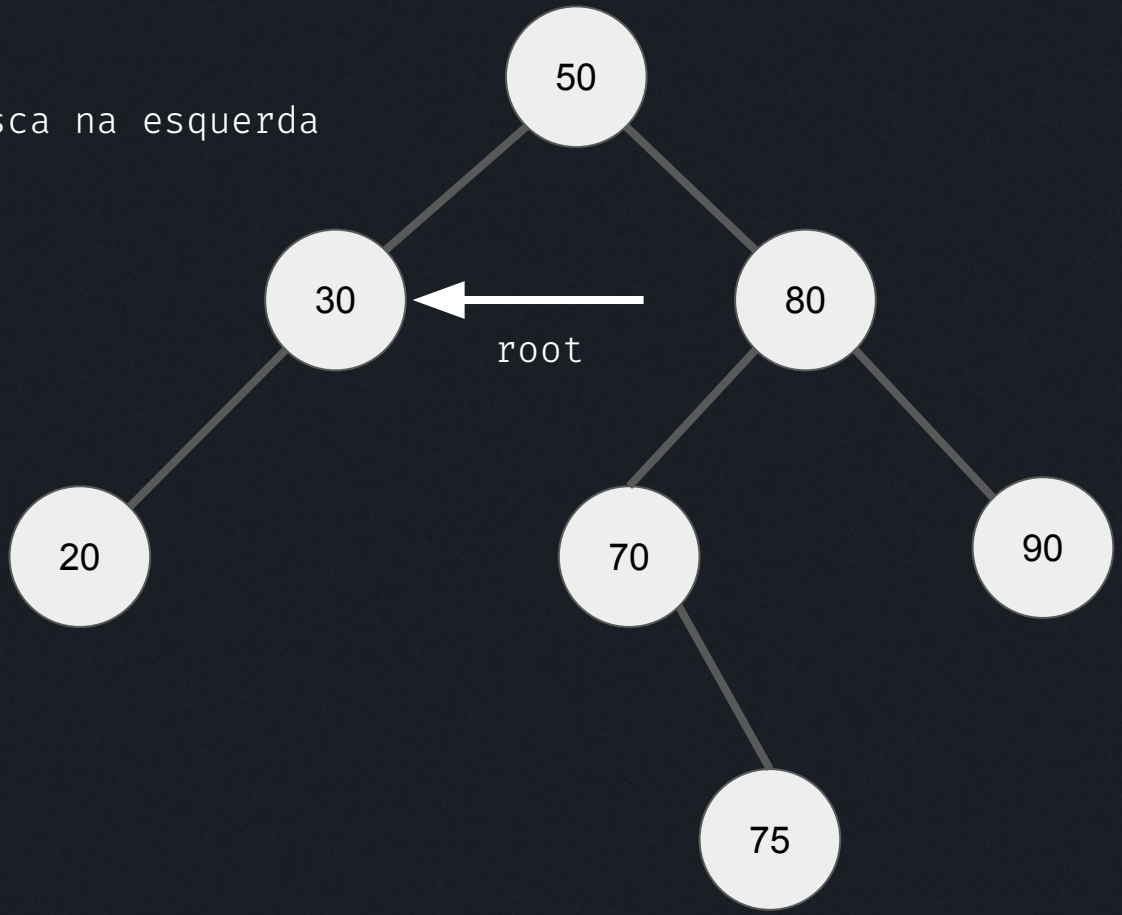
Search

Search (15)



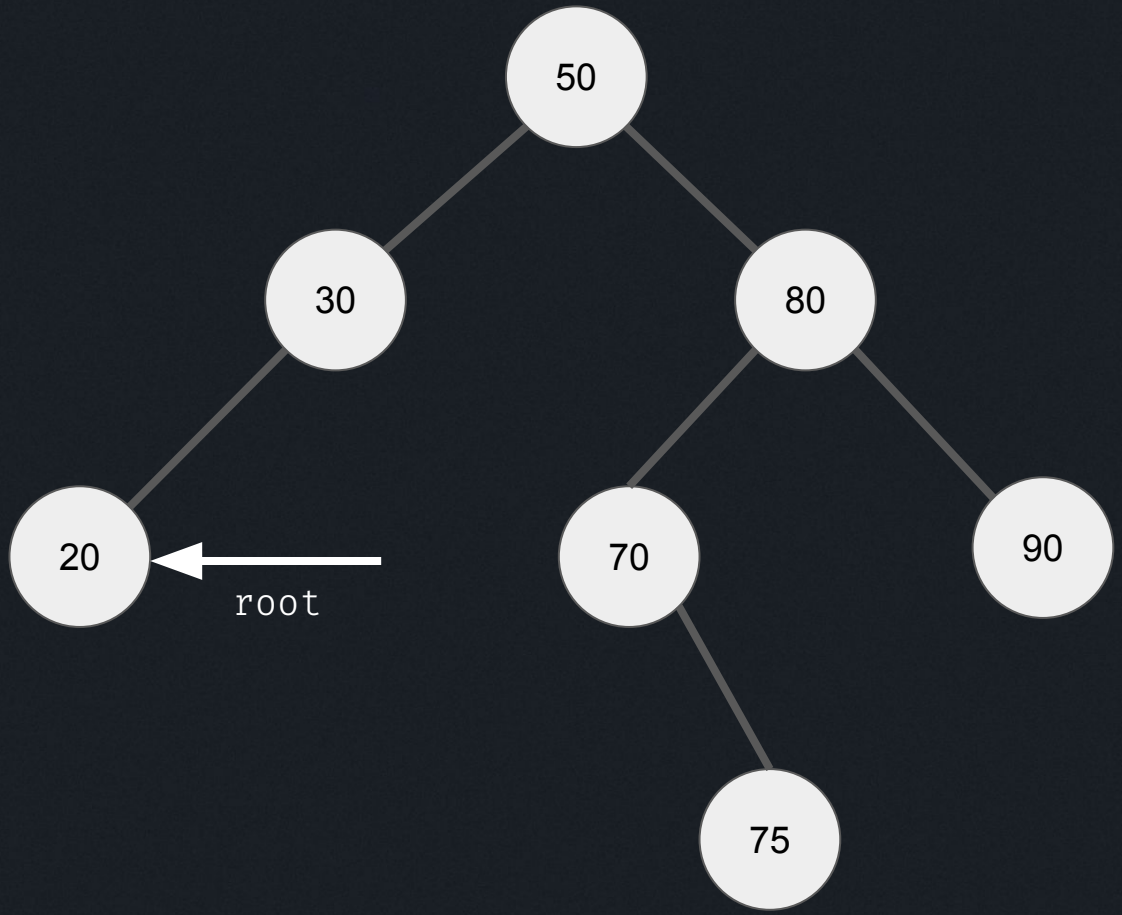
Search

Search (15)
15 < 30, busca na esquerda



Search

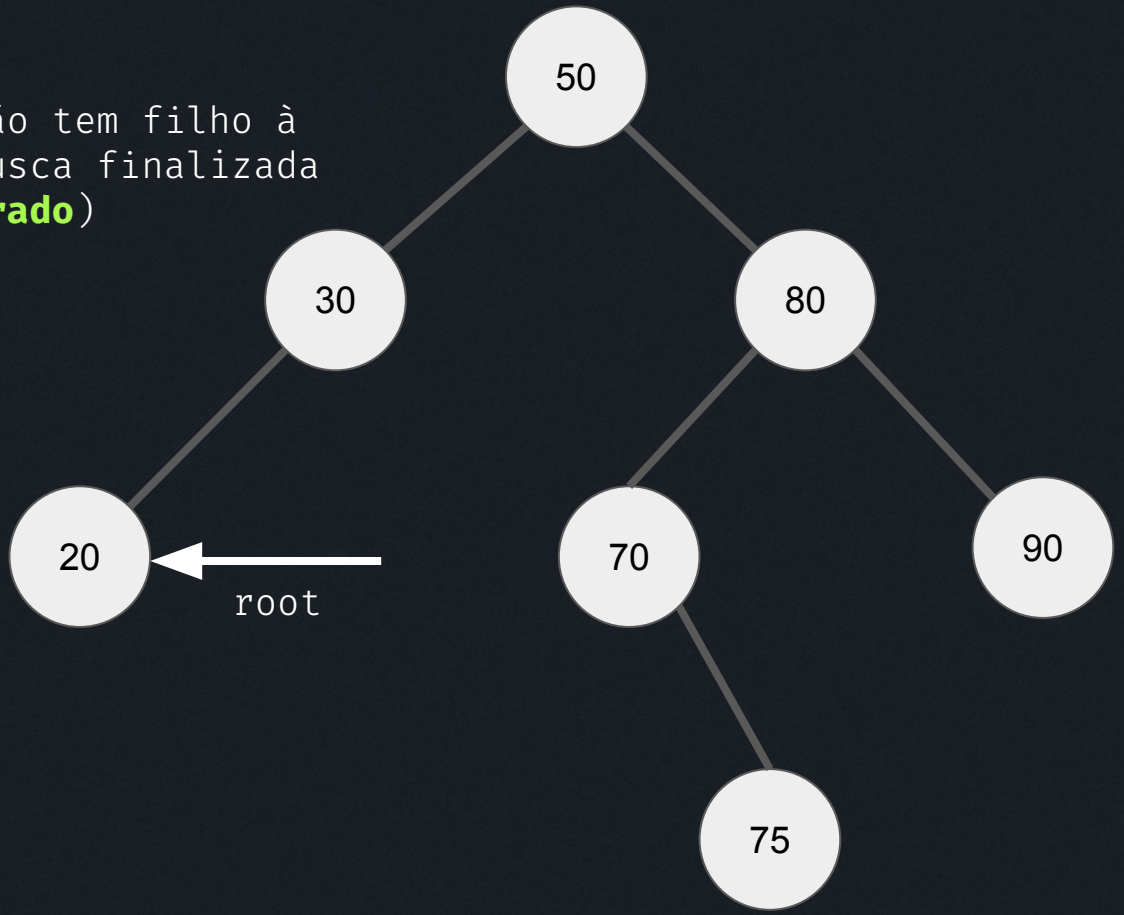
Search (15)



ada

Search

Search (15)
15 < 20 e não tem filho à esquerda, busca finalizada (**não encontrado**)



Search

https://leetcode.com/problems/search-in-a-binary-search-tree

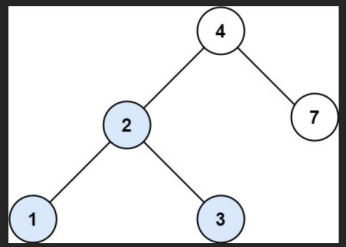
700. Search in a Binary Search Tree

Easy | Topics | Companies

You are given the `root` of a binary search tree (BST) and an integer `val`.

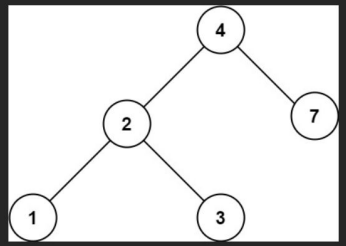
Find the node in the BST that the node's value equals `val` and return the subtree rooted with that node. If such a node does not exist, return `null`.

Example 1:




```
Input: root = [4,2,7,1,3], val = 2
Output: [2,1,3]
```

Example 2:




```
Input: root = [4,2,7,1,3], val = 5
Output: []
```

Constraints:

- The number of nodes in the tree is in the range `[1, 5000]`.
- `1 <= Node.val <= 10^7`
- `root` is a binary search tree.
- `1 <= val <= 10^7`

ada

Search

https://leetcode.com/problems/search-in-a-binary-search-tree

```python
class Solution:
    def searchBST(self, root, val):
        if not root:
            return None

        if val == root.val:
            return root

        if val > root.val:
            return self.searchBST(root.right, val)
        else:
            return self.searchBST(root.left, val)
```

Validate Binary Search Tree

https://leetcode.com/problems/validate-binary-search-tree/description/

98. Validate Binary Search Tree

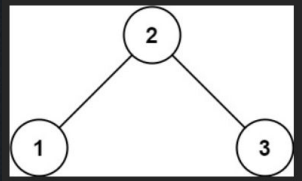
Medium | Topics | Companies

Given the `root` of a binary tree, *determine if it is a valid binary search tree (BST).*

A **valid BST** is defined as follows:

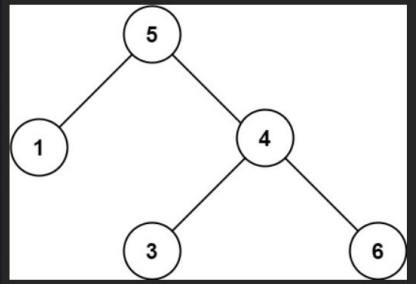
- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:




```
Input: root = [2,1,3]
Output: true
```

Example 2:




```
Input: root = [5,1,4,null,null,3,6]
Output: false
Explanation: The root node's value is 5 but its right child's value is 4.
```

ada

Validate Binary Search Tree

https://leetcode.com/problems/validate-binary-search-tree/description/

```
def isValidBST(self, root):
    return self.isValid(root, float('-inf'), float('inf'))

def isValid(self, root, minVal, maxVal):
    if not root:
        return True

    if root.left and root.left.val ⩾ root.val:
        return False

    if root.right and root.right.val ⩽ root.val:
        return False

    if root.val ⩽ minVal or maxVal ⩽ root.val:
        return False

    return (self.isValid(root.left, minVal, root.val)
            and self.isValid(root.right, root.val, maxVal))
```

Validate Binary Search Tree

https://leetcode.com/problems/validate-binary-search-tree/description/

```
public boolean isValidBST(TreeNode root) {
            return isValid(root, Long.MIN_VALUE, Long.MAX_VALUE);}

        private boolean isValid(TreeNode root, long minVal, long maxVal) {
            if (root == null)
                return true;

            if (root.left != null && root.left.val >= root.val)
                return false;

            if (root.right != null && root.right.val <= root.val)
                return false;

            if (root.val <= minVal || root.val >= maxVal)
                return false;

            return isValid(root.left, minVal, root.val) && isValid(root.right, root.val,
maxVal);
    }
```

Lowest Common Ancestor

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree

235. Lowest Common Ancestor of a Binary Search Tree

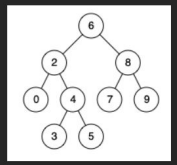
Solved

Medium | Topics | Companies

Given a binary search tree (BST), find the lowest common ancestor (LCA) node of two given nodes in the BST.

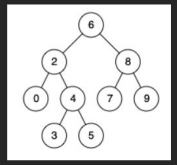
According to the definition of LCA on Wikipedia: "The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**)."

Example 1:




```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
Output: 6
Explanation: The LCA of nodes 2 and 8 is 6.
```

Example 2:




```
Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
Output: 2
Explanation: The LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.
```

Example 3:

```
Input: root = [2,1], p = 2, q = 1
Output: 2
```

Constraints:

- The number of nodes in the tree is in the range `[2, 10^5]`.
- `-10^9 <= Node.val <= 10^9`
- All `Node.val` are **unique**.
- `p != q`
- `p` and `q` will exist in the BST.

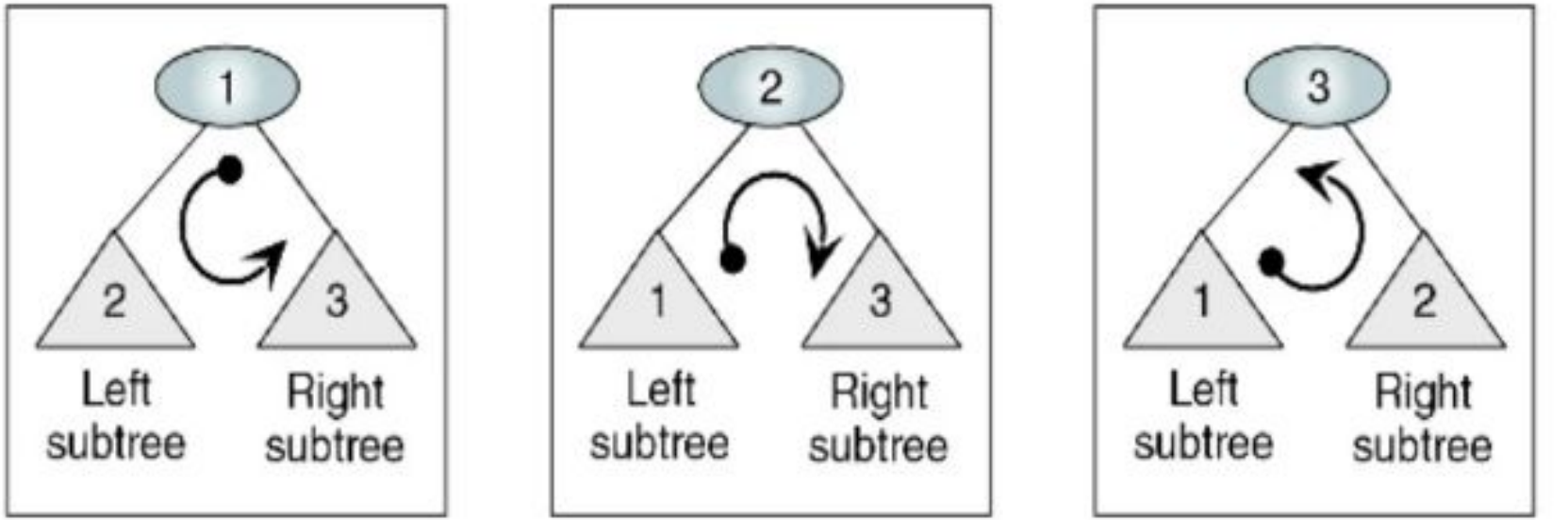
ada

Lowest Common Ancestor

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree

```
class Solution:
    def lowestCommonAncestor(self, root, p, q):
        if ((root.val ⩾ p.val and root.val ⩽ q.val) or
            (root.val ⩽ p.val and root.val ⩾ q.val)):
            return root

        if (root.val ⩾ p.val and root.val ⩾ q.val):
            return self.lowestCommonAncestor(root.left, p, q)
        else:
            return self.lowestCommonAncestor(root.right, p, q)
```

Traversal



(a) Preorder traversal (b) Inorder traversal (c) Postorder traversal

InOrder

https://leetcode.com/problems/binary-tree-inorder-traversal

94. Binary Tree Inorder Traversal

Easy | Topics | Companies

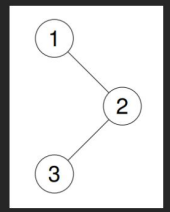
Given the `root` of a binary tree, return *the inorder traversal of its nodes' values.*

Example 1:

Input: `root = [1,null,2,3]`

Output: `[1,3,2]`

Explanation:

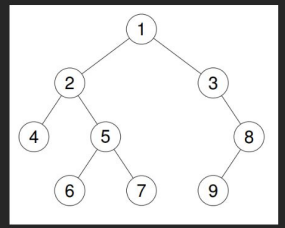


Example 2:

Input: `root = [1,2,3,4,5,null,8,null,null,6,7,9]`

Output: `[4,2,6,5,7,1,3,9,8]`

Explanation:



Example 3:

Input: `root = []`

Output: `[]`

Example 4:

Input: `root = [1]`

Output: `[1]`

ada

InOrder

https://leetcode.com/problems/binary-tree-inorder-traversal

```
class Solution:
    def inorderTraversal(self, root):
        l = []
        self.inorder(root, l)

        return l

    def inorder(self, root, l):
        if not root: return

        self.inorder(root.left, l)

        l.append(root.val)

        self.inorder(root.right, l)
```

InOrder

https://leetcode.com/problems/binary-tree-inorder-traversal

```
public List<Integer> inorderTraversal(TreeNode root) {

    List<Integer> result = new ArrayList<>();
        inorder(root, result);
        return result;
    }

    private void inorder(TreeNode root, List<Integer> result)
{
        if (root == null) {
            return;
        }

        inorder(root.left, result);
        result.add(root.val);
        inorder(root.right, result);
    }
```

PreOrder

https://leetcode.com/problems/binary-tree-preorder-traversal/

144. Binary Tree Preorder Traversal

Easy | Topics | Companies

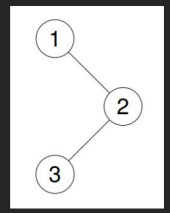
Given the `root` of a binary tree, return *the preorder traversal of its nodes' values.*

Example 1:

Input: `root = [1,null,2,3]`

Output: `[1,2,3]`

Explanation:

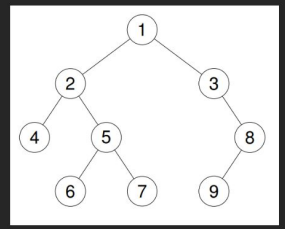


Example 2:

Input: `root = [1,2,3,4,5,null,8,null,null,6,7,9]`

Output: `[1,2,4,5,6,7,3,8,9]`

Explanation:



Example 3:

Input: `root = []`

Output: `[]`

Example 4:

Input: `root = [1]`

Output: `[1]`

ada

PreOrder

https://leetcode.com/problems/binary-tree-preorder-traversal/

```python
class Solution:
    def preorderTraversal(self, root):
        l = []
        self.preorder(root, l)

        return l

    def preorder(self, root, l):
        if not root: return

        l.append(root.val)

        self.preorder(root.left, l)

        self.preorder(root.right, l)
```

PreOrder

https://leetcode.com/problems/binary-tree-preorder-traversal/

```java
public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        preorder(root, result);
        return result;
    }

    private void preorder(TreeNode root, List<Integer> result) {
        if (root == null) {
            return;
        }

        result.add(root.val);
        preorder(root.left, result);
        preorder(root.right, result);
    }
```

PostOrder

https://leetcode.com/problems/binary-tree-postorder-traversal

145. Binary Tree Postorder Traversal

Easy | Topics | Companies

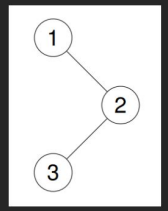
Given the `root` of a binary tree, return *the postorder traversal of its nodes' values.*

Example 1:

Input: `root = [1,null,2,3]`

Output: `[3,2,1]`

Explanation:

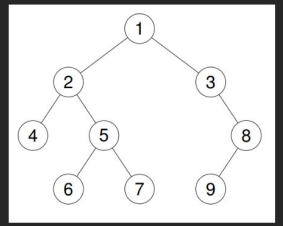


Example 2:

Input: `root = [1,2,3,4,5,null,8,null,null,6,7,9]`

Output: `[4,6,7,5,2,9,8,3,1]`

Explanation:



Example 3:

Input: `root = []`

Output: `[]`

Example 4:

Input: `root = [1]`

Output: `[1]`

ada

PostOrder

https://leetcode.com/problems/binary-tree-postorder-traversal

```
class Solution:
    def postorderTraversal(self, root):
        l = []
        self.postorder(root, l)

        return l

    def postorder(self, root, l):
        if not root: return

        self.postorder(root.left, l)

        self.postorder(root.right, l)

        l.append(root.val)
```

PostOrder

https://leetcode.com/problems/binary-tree-postorder-traversal

```java
public List<Integer> postorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        postorder(root, result);
        return result;
    }

    private void postorder(TreeNode root, List<Integer> result) {
        if (root == null) {
            return;
        }

        postorder(root.left, result);
        postorder(root.right, result);
        result.add(root.val);
    }
```

Obrigada