

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

\* TreeNode(int val, TreeNode left, TreeNode right) {

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class CBTInserter {

List<TreeNode> tree;

public CBTInserter(TreeNode root) {

tree = new ArrayList<>();

tree.add(root);

for (int i = 0; i < tree.size(); ++i) {

if (tree.get(i).left != null) tree.add(tree.get(i).left);

if (tree.get(i).right != null && tree.get(i).left != null) tree.add(tree.get(i).right);

}

}

public int insert(int val) {

TreeNode node = new TreeNode(val);

int tamanio = tree.size();

tree.add(node);

if(tamanio%2 == 0){

tree.get((tamanio-1)/2).right = node;

}else{

tree.get((tamanio-1)/2).left = node;

}

return tree.get((tamanio-1)/2).val;

}

public TreeNode get\_root() {

return tree.get(0);

}

}

/\*\*

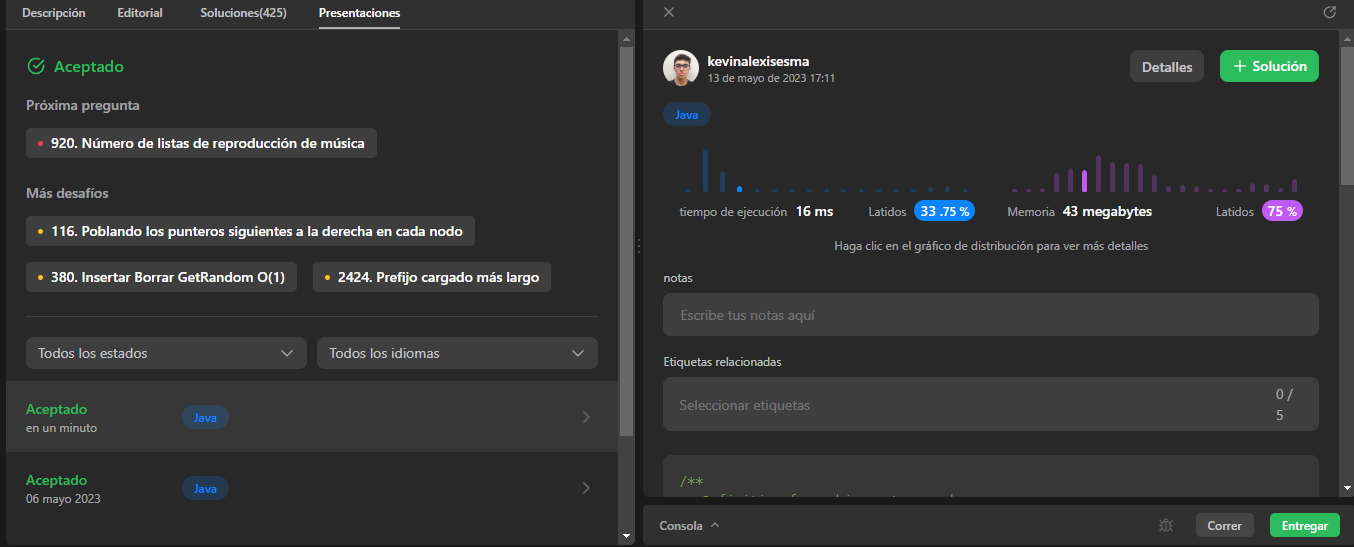
\* Your CBTInserter object will be instantiated and called as such:

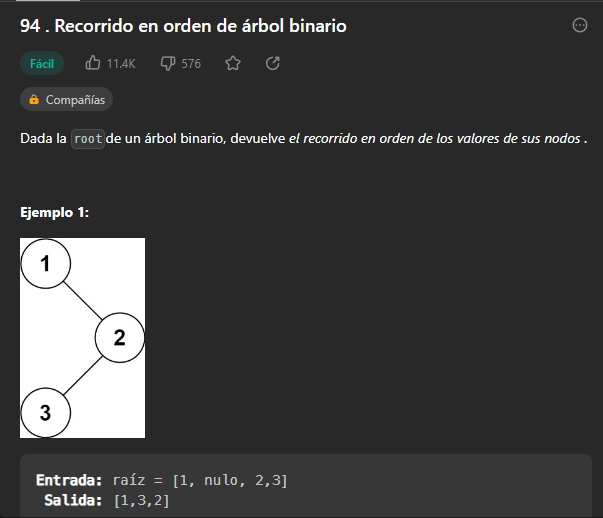
\* CBTInserter obj = new CBTInserter(root);

\* int param\_1 = obj.insert(val);

\* TreeNode param\_2 = obj.get\_root();

\*/





class Solution {

List<Integer> inorder = new ArrayList<>();

public List<Integer> inorderTraversal(TreeNode root) {

inorderT(root);

return inorder;

}

public void inorderT(TreeNode root) {

if(root != null){

if(root.left != null){

inorderT(root.left);

}

inorder.add(root.val);

if(root.right != null){

inorderT(root.right);

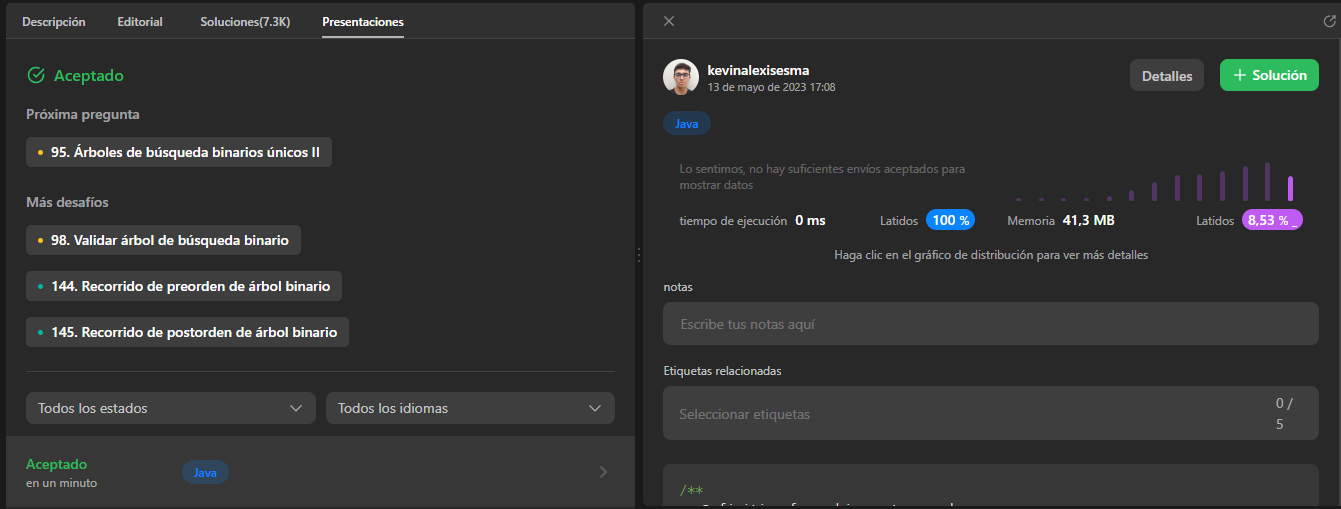
}

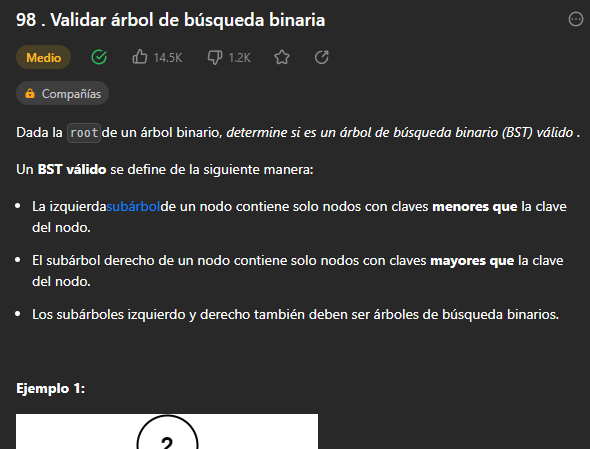
}

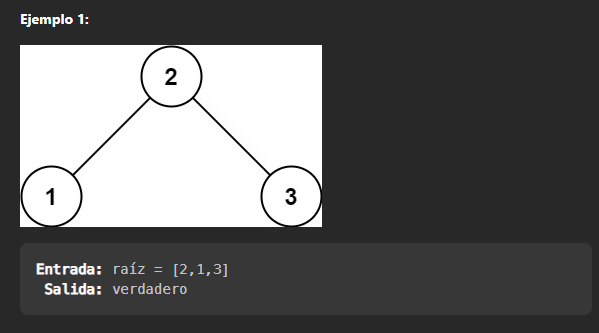
return;

}

}







class Solution {

boolean bst = true;

TreeNode previous = null;

public void isValid(TreeNode root){

if(root==null){

return;

}

isValid(root.left);

if(previous!=null && previous.val>=root.val){

bst = false;

}

previous = root;

isValid(root.right);

}

public boolean isValidBST(TreeNode root) {

isValid(root);

return bst;

}

}

