Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return *the binary tree*.

**Example 1:**



**Input:** preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]

**Output:** [3,9,20,null,null,15,7]

**Example 2:**

**Input:** preorder = [-1], inorder = [-1]

**Output:** [-1]

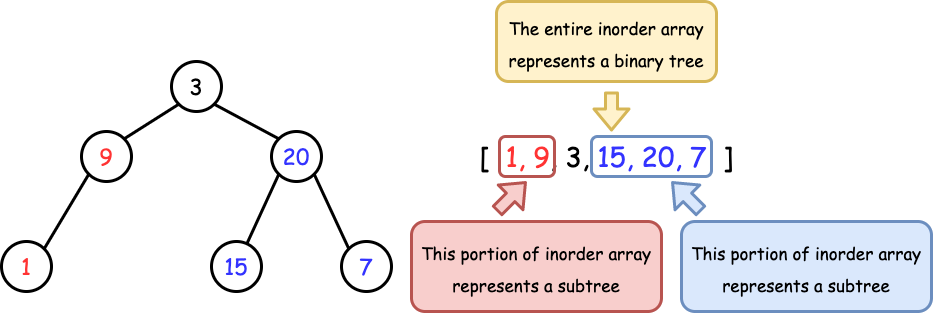
Solution:

Solution

Overview

This problem examines your understanding of preorder and inorder binary tree traversals. If you are not familiar with them, feel free to visit our [Explore Cards](https://leetcode.com/explore/learn/card/data-structure-tree/) where you will see all the ways to traverse a binary tree including preorder, inorder, postorder, and level-order traversals :)

A tree has a recursive structure because it has subtrees which are trees themselves. Let's take a look at the inorder traversal of a binary tree, and you will see the built-in recursive structure.



*Figure 1. The recursive structure in a Tree.*

Henceforth, we will leverage this property and find a way to recursively construct the tree.

Approach 1: Recursion

**Intuition**

The two key observations are:

1. Preorder traversal follows Root -> Left -> Right, therefore, given the preorder array preorder, we have easy access to the root which is preorder[0].
2. Inorder traversal follows Left -> Root -> Right, therefore if we know the position of Root, we can recursively split the entire array into two subtrees.

Now the idea should be clear enough. We will design a recursion function: it will set the first element of preorder as the root, and then construct the entire tree. To find the left and right subtrees, it will look for the root in inorder, so that everything on the left should be the left subtree, and everything on the right should be the right subtree. Both subtrees can be constructed by making another recursion call.

It is worth noting that, while we recursively construct the subtrees, we should choose the next element in preorder to initialize as the new roots. This is because the current one has already been initialized to a parent node for the subtrees.

**Algorithm**

* Build a hashmap to record the relation of value -> index for inorder, so that we can find the position of root in constant time.
* Initialize an integer variable preorderIndex to keep track of the element that will be used to construct the root.
* Implement the recursion function arrayToTree which takes a range of inorder and returns the constructed binary tree:
  + if the range is empty, return null;
  + initialize the root with preorder[preorderIndex] and then increment preorderIndex;
  + recursively use the left and right portions of inorder to construct the left and right subtrees.
* Simply call the recursion function with the entire range of inorder.

**Question:**

Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree.

Example 1:

Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]

Output: [3,9,20,null,null,15,7]

Example 2:

Input: preorder = [-1], inorder = [-1]

Output: [-1]

Solution:

----------

/\*\*

\* 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 Solution {

int preorderIndex;

Map<Integer, Integer> inorderIndexMap;

public TreeNode buildTree(int[] preorder, int[] inorder) {

preorderIndex = 0;

// build a hashmap to store value -> its index relations

inorderIndexMap = new HashMap<>();

for (int i = 0; i < inorder.length; i++) {

inorderIndexMap.put(inorder[i], i);

}

return arrayToTree(preorder, 0, preorder.length - 1);

}

private TreeNode arrayToTree(int[] preorder, int left, int right) {

// if there are no elements to construct the tree

if (left > right) return null;

// select the preorder\_index element as the root and increment it

int rootValue = preorder[preorderIndex++];

TreeNode root = new TreeNode(rootValue);

// build left and right subtree

// excluding inorderIndexMap[rootValue] element because it's the root

root.left = arrayToTree(preorder, left, inorderIndexMap.get(rootValue) - 1);

root.right = arrayToTree(preorder, inorderIndexMap.get(rootValue) + 1, right);

return root;

}

}