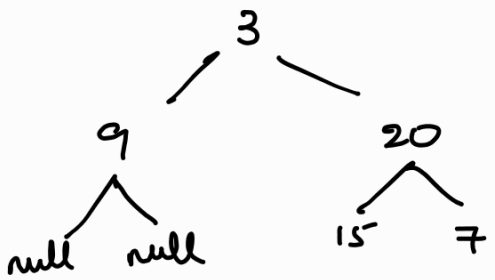
Leetcode Problem #105 (Medium) Construct Binary Tree from PreOrder and InOrder traversal

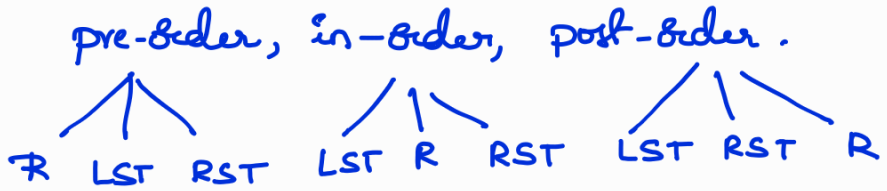
https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/description/

PRE-ORDER : {3, 9, 20, 15, 7}

IN-ORDER : {9, 3, 15, 20, 7}.

OUTPUT : {3, 9, 20, NULL, NULL, 15, 7}.





R — root LST — Left Sub Tree RST - Right Sub Tree.

APPROACH :-

(1) Find root from pre-order which is first element, mark the index as preStart.

(2) Locate same root in in-order, mark the index as inRoot

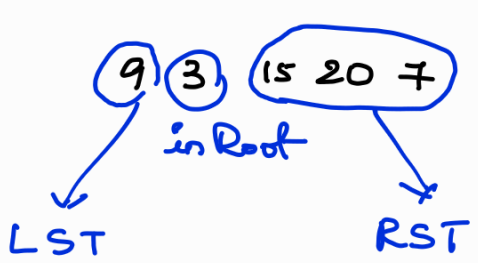
(3) Elements to left of inRoot form LST
Elements to right of inRoot form RST

PRE

(3) 9 20 15 7

preStart

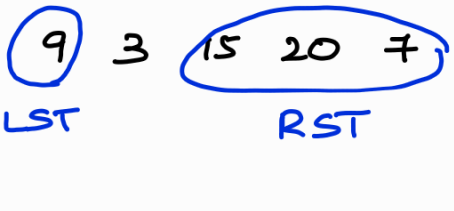
IN

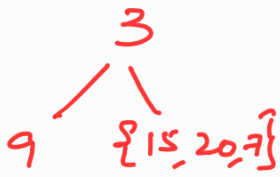


④ Locate LST : {9} & RST = {15, 20, 7} in pre-order.

PRE

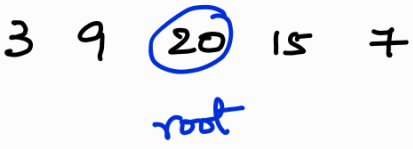
IN



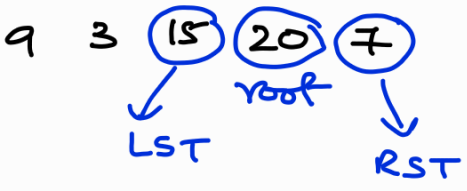


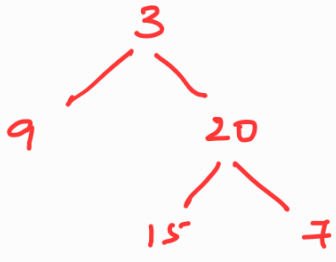
⑤ Repeat step ① to ④ for Left SubTree & Right SubTree individually

PRE



IN





ALGORITHM :-

preStart → index of root in preOrder. (0)
inStart → start index of tree in inOrder (0)
inEnd → end index of tree in inOrder (4)

constructTree (preStart, inStart, inEnd, preOrder[], inOrder[]) {

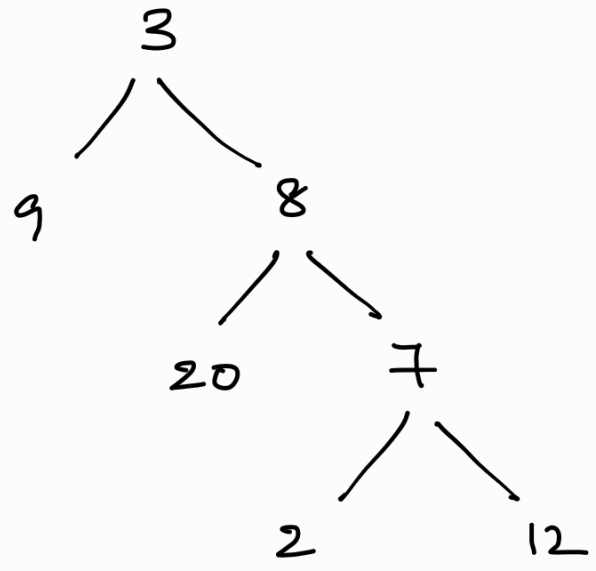
1) find root from preOrder => root = preOrder[preStart]
2) find index of root in inOrder =>
inRoot = inMap.get(root.val)

3) form leftsubtree & return root.

⇒ constructTree (preStart+1, inStart, inRoot-1, preOrder[], inOrder[])

4) form rightSubTree & return root.

⇒ constructTree (preStart + inRoot - inStart+1, inRoot+1, inEnd, preOrder[], inOrder[])



PREORDER: 3, 9, 8, 20, 7, 2, 12

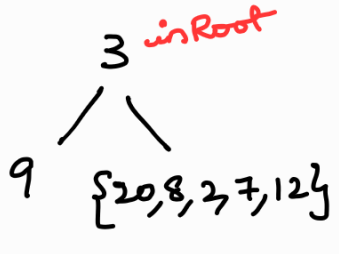
INORDER: 9, 3, 20, 8, 2, 7, 12.

(preStart, inStart, inEnd).

↓

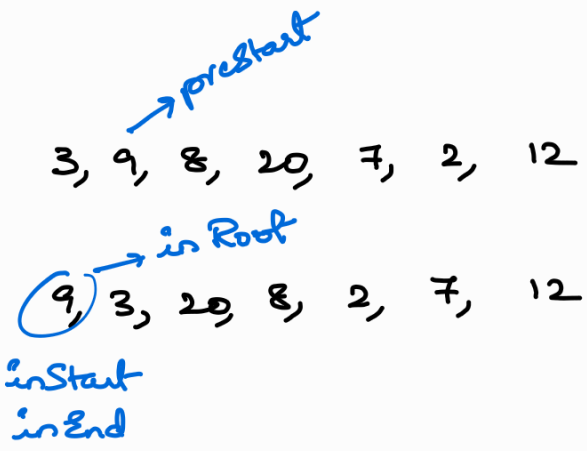
① preStart
(3), 9, 8, 20, 7, 2, 12.

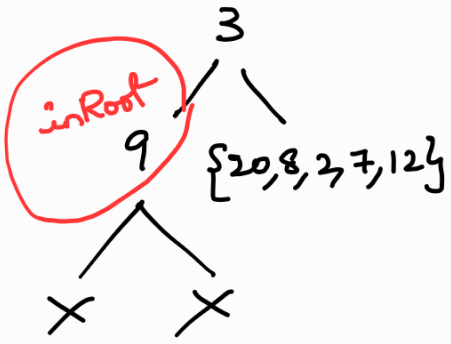
② (9), (3), 20, 8, 2, 7, (12)
inStart, inRoot, inEnd.



II. LST

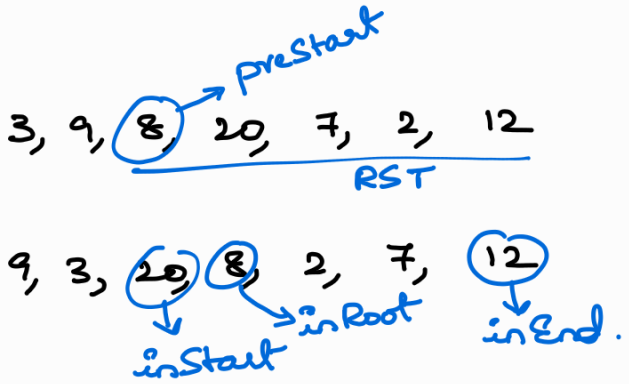
(preStart+1, inStart, inRoot-1)





V. RST.

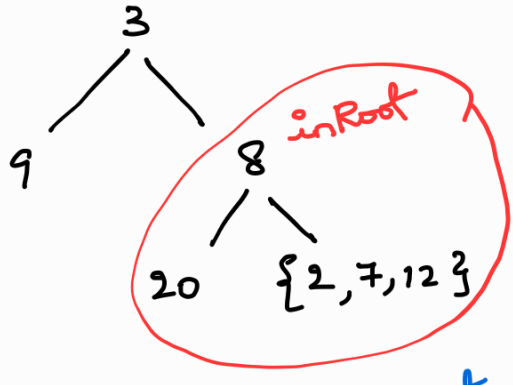
(preStart + inRoot − inStart + 1, inRoot + 1, inEnd)



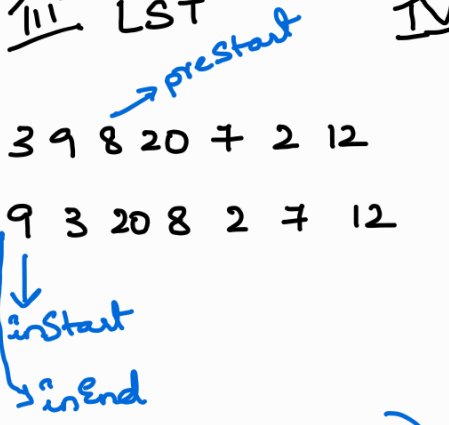
[Note: preStart of RST =

preStart + sizeOf LST + 1

= preStart + (inRoot − inStart) + 1



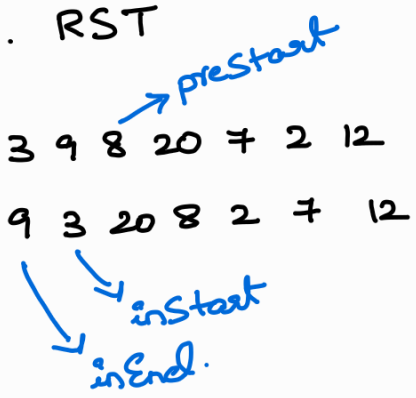
III. LST



(inStart > inEnd)

terminate

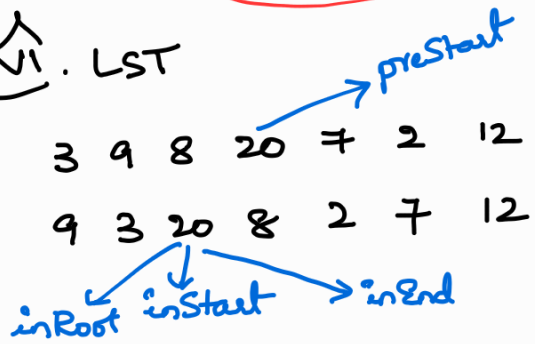
IV. RST

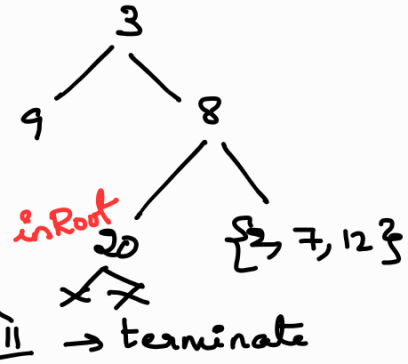


(inStart > inEnd)

terminate

VI. LST





VII, VIII → terminate

IX RST

3 9 8 20 7 2 12

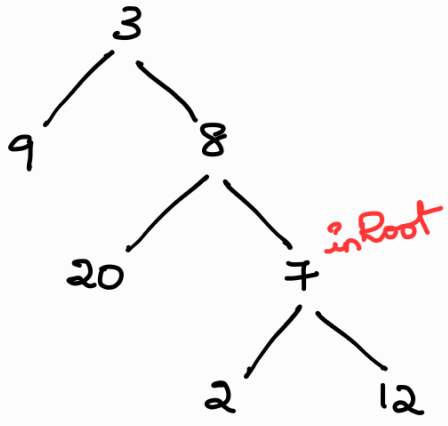
9 3 20 8 2 7 12

preStart → 7

inStart → 2

inRoot → 7

inEnd → 12



X. LST

3 9 8 20 7 2 12

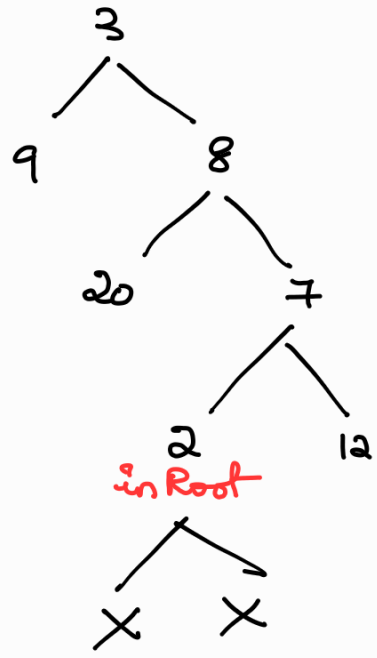
9 3 20 8 2 7 12

preStart → 7

inRoot → 2

inStart

inEnd.



XI, XII → terminate

XIII. RST

3 9 8 20 7 2 12

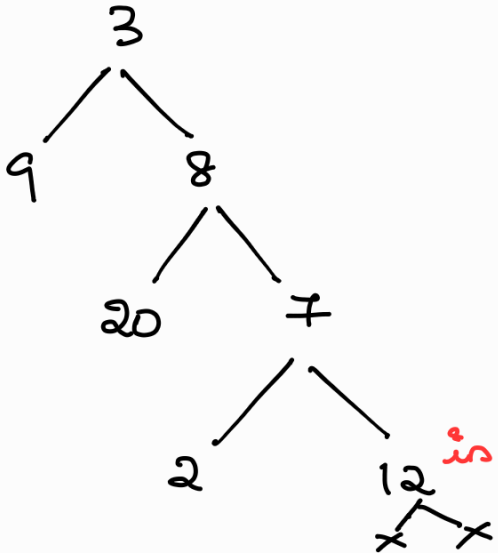
9 3 20 8 2 7 12

preStart → 12

inRoot → 12

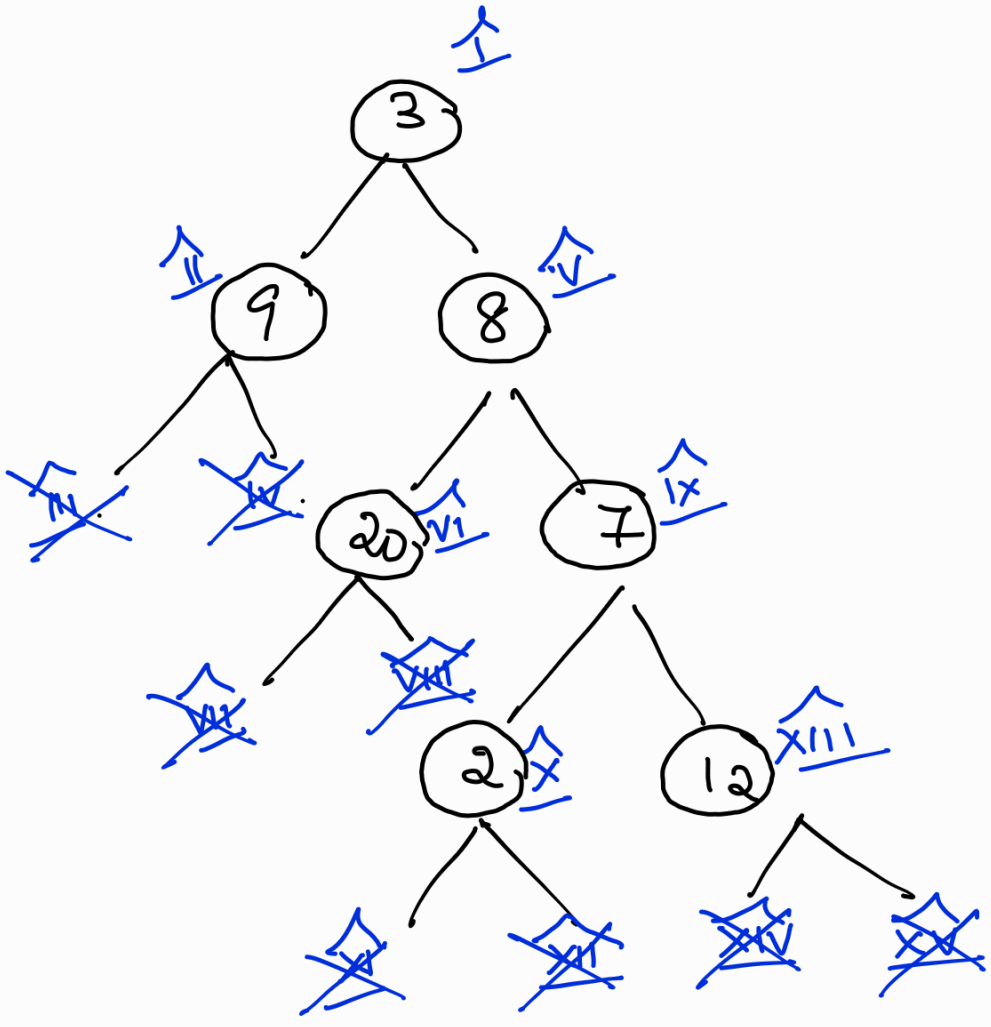
inStart

inEnd.



XIV, XV → terminate

You can view the above steps in a diagram below:



For example, at process VI, we compute new (preStart, inStart, inEnd) values based on values of process V.

at process V, preStart = 3

inRoot = 3

inStart = 2

inEnd = 6

at process VI, preStart = 4 (preStart+1)

inRoot = 2

inStart = 2 (inStart)

inEnd = 2 (inRoot-1).

CODE :-

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

    HashMap<Integer, Integer> inMap = new HashMap<>();
    for (int i = 0; i < inorder.length; i++)
        inMap.put(inorder[i], i);
    return constructTree(0, 0, preorder.length-1, preorder, inorder, inMap);
}

public TreeNode constructTree(int preStart, int inStart, int inEnd, int[] preorder,
int[] inorder, HashMap<Integer, Integer> inMap) {

    if (preStart > preorder.length - 1 || inStart > inEnd)
        return null;
    TreeNode root = new TreeNode(preorder[preStart]);
    int inRoot = inMap.get(root.val);
    root.left = constructTree(preStart + 1, inStart, inRoot - 1, preorder, inorder,
inMap);
    root.right = constructTree(preStart + inRoot - inStart + 1, inRoot + 1, inEnd,
preorder, inorder, inMap);
    return root;
}
```

Dry Run:

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

In below diagram, each node is represented as: **element** [preStart, inStart, inEnd]