Find all possible binary trees with given Inorder Traversal

Given an array that represents Inorder Traversal, find all possible Binary Trees with the given Inorder traversal and print their preorder traversals.

Examples:

```
Input:   in[] = {3, 2};
Output:  Preorder traversals of different possible Binary Trees are:
         3 2
         2 3
Below are different possible binary trees
    3        2
     \      /
      2    3

Input:   in[] = {4, 5, 7};
Output:  Preorder traversals of different possible Binary Trees are:
          4 5 7
          4 7 5
          5 4 7
          7 4 5
          7 5 4
Below are different possible binary trees
  4         4           5         7       7
   \         \       /   \      /       /
    5         7     4     7    4       5
     \       /                 \     /
      7     5                   5   4
```

We strongly recommend you to minimize your browser and try this yourself first.

Let given inorder traversal be **in[]**. In the given traversal, all nodes in left subtree of **in[i]** must appear before it and in right subtree must appear after it. So when we consider in[i] as root, all elements from in[0] to in[i-1] will be in left subtree and in[i+1] to n-1 will be in right subtree. If in[0] to in[i-1] can form **x** different trees and in[i+1]

to in[n-1] can from **y** different trees then we will have **x*y** total trees when in[i] as root. So we simply iterate from 0 to n-1 for root. For every node in[i], recursively find different left and right subtrees. If we take a closer look, we can notice that the count is basically n'th Catalan number. We have discussed different approaches to find n'th Catalan number here.

The idea is to maintain a list of roots of all Binary Trees. Recursively construct all possible left and right subtrees. Create a tree for every pair of left and right subtree and add the tree to list. Below is detailed algorithm.

```
1) Initialize list of Binary Trees as empty.
2) For every element in[i] where i varies from 0 to n-1,
    do following
......a)  Create a new node with key as 'arr[i]',
          let this node be 'node'
......b)  Recursively construct list of all left subtrees.
......c)  Recursively construct list of all right subtrees.
3) Iterate for all left subtrees
   a) For current leftsubtree, iterate for all right subtrees
        Add current left and right subtrees to 'node' and add
        'node' to list.
```

C++

```
// C++ program to find binary tree with given inorder
// traversal
#include <bits/stdc++.h>
using namespace std;

// Node structure
struct Node
{
    int key;
    struct Node *left, *right;
};

// A utility function to create a new tree Node
struct Node *newNode(int item)
{
    struct Node *temp = new Node;
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to do preorder traversal of BST
void preorder(Node *root)
{
    if (root != NULL)
    {
        printf("%d ", root->key);
        preorder(root->left);
        preorder(root->right);
    }
}
```

```
// Function for constructing all possible trees with
// given inorder traversal stored in an array from
// arr[start] to arr[end]. This function returns a
// vector of trees.
vector<Node *> getTrees(int arr[], int start, int end)
{
    // List to store all possible trees
    vector<Node *> trees;

    /* if start > end then subtree will be empty so
    returning NULL in the list */
    if (start > end)
    {
        trees.push_back(NULL);
        return trees;
    }

    /* Iterating through all values from start to end
        for constructing left and right subtree
        recursively */
    for (int i = start; i <= end; i++)
    {
        /* Constructing left subtree */
        vector<Node *> ltrees = getTrees(arr, start, i-1);

        /* Constructing right subtree */
        vector<Node *> rtrees = getTrees(arr, i+1, end);

        /* Now looping through all left and right subtrees
        and connecting them to ith root below */
        for (int j = 0; j < ltrees.size(); j++)
        {
            for (int k = 0; k < rtrees.size(); k++)
            {
                // Making arr[i] as root
                Node * node = newNode(arr[i]);

                // Connecting left subtree
                node->left = ltrees[j];

                // Connecting right subtree
                node->right = rtrees[k];

                // Adding this tree to list
                trees.push_back(node);
            }
        }
    }
    return trees;
}

// Driver Program to test above functions
int main()
{
    int in[] = {4, 5, 7};
    int n = sizeof(in) / sizeof(in[0]);

    vector<Node *> trees = getTrees(in, 0, n-1);

    cout << "Preorder traversals of different "
         << "possible Binary Trees are \n";
    for (int i = 0; i < trees.size(); i++)
    {
        preorder(trees[i]);
        printf("\n");
```

```
    }
    return 0;
}
```

Run on IDE

Java

```
// Java program to find binary tree with given inorder
// traversal
import java.util.Vector;

/* Class containing left and right child of current
 node and key value*/
class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = null;
        right = null;
    }
}

/* Class to print Level Order Traversal */
class BinaryTree {

    Node root;

    // A utility function to do preorder traversal of BST
    void preOrder(Node node) {
        if (node != null) {
            System.out.print(node.data + " "    );
            preOrder(node.left);
            preOrder(node.right);
        }
    }

    // Function for constructing all possible trees with
    // given inorder traversal stored in an array from
    // arr[start] to arr[end]. This function returns a
    // vector of trees.
    Vector<Node> getTrees(int arr[], int start, int end) {

        // List to store all possible trees
        Vector<Node> trees= new Vector<Node>();

        /* if start > end then subtree will be empty so
         returning NULL in the list */
        if (start > end) {
            trees.add(null);
            return trees;
        }

        /* Iterating through all values from start to end
         for constructing left and right subtree
         recursively */
        for (int i = start; i <= end; i++) {
            /* Constructing left subtree */
            Vector<Node> ltrees = getTrees(arr, start, i - 1);
```

```
            /* Constructing right subtree */
            Vector<Node> rtrees = getTrees(arr, i + 1, end);

            /* Now looping through all left and right subtrees
             and connecting them to ith root below */
            for (int j = 0; j < ltrees.size(); j++) {
                for (int k = 0; k < rtrees.size(); k++) {

                    // Making arr[i] as root
                    Node node = new Node(arr[i]);

                    // Connecting left subtree
                    node.left = ltrees.get(j);

                    // Connecting right subtree
                    node.right = rtrees.get(k);

                    // Adding this tree to list
                    trees.add(node);
                }
            }
        }
        return trees;
    }

    public static void main(String args[]) {
        int in[] = {4, 5, 7};
        int n = in.length;
        BinaryTree tree = new BinaryTree();
        Vector<Node> trees = tree.getTrees(in, 0, n - 1);
        System.out.println("Preorder traversal of different "+
                           " binary trees are:");
        for (int i = 0; i < trees.size(); i++) {
            tree.preOrder(trees.get(i));
            System.out.println("");
        }
    }
}
```

Run on IDE

Python

```
# Python program to find binary tree with given
# inorder traversal

# Node Structure
class Node:

    # Utility to create a new node
    def __init__(self , item):
        self.key = item
        self.left = None
        self.right = None

# A utility function to do preorder traversal of BST
def preorder(root):
    if root is not None:
        print root.key,
        preorder(root.left)
        preorder(root.right)
```

```
# Function for constructing all possible trees with
# given inorder traversal stored in an array from
# arr[start] to arr[end]. This function returns a
# vector of trees.
def getTrees(arr , start , end):

    # List to store all possible trees
    trees = []

    """ if start > end then subtree will be empty so
    returning NULL in the list """
    if start > end :
        trees.append(None)
        return trees


    """ Iterating through all values from start to end
        for constructing left and right subtree
        recursively """
    for i in range(start , end+1):

        # Constructing left subtree
        ltrees = getTrees(arr , start , i-1)

        # Constructing right subtree
        rtrees = getTrees(arr , i+1 , end)

        """ Looping through all left and right subtrees
        and connecting to ith root below"""
        for j in ltrees :
            for k in rtrees :

                # Making arr[i]  as root
                node  = Node(arr[i])

                # Connecting left subtree
                node.left = j

                # Connecting right subtree
                node.right = k

                # Adding this tree to list
                trees.append(node)
    return trees

# Driver program to test above function
inp = [4 , 5, 7]
n = len(inp)

trees = getTrees(inp , 0 , n-1)

print "Preorder traversals of different possible\
 Binary Trees are "
for i in trees :
    preorder(i);
    print ""

# This program is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Run on IDE

Output:

```
Preorder traversals of different possible Binary Trees are
4 5 7
4 7 5
5 4 7
7 4 5
7 5 4
```

Thanks to Utkarsh for suggesting above solution.

This problem is similar to the problem discussed here.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

11 Comments Category: Trees

Related Posts:

- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree
- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack
- Check if leaf traversal of two Binary Trees is same?
- Closest leaf to a given node in Binary Tree
- Locking and Unlocking of Resources arranged in the form of n-ary Tree

(Login to Rate and Mark)

3.5 Average Difficulty : **3.5/5.0**
Based on **18** vote(s)

Add to TODO List

Mark as DONE

Like Share 10 people like this.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

11 Comments GeeksforGeeks 1 Login

Recommend Share Sort by Newest

Join the discussion…

Ahmad Najeeb • 2 months ago
For the {3,2} example, why are only 2 of the possible 4 binary tree arrangements considered ?
• Reply • Share ›

Dinesh Agarwal • 2 months ago
time complexity is n^3 in this ? and isn't there any more efficient way possibly without recursion ?
• Reply • Share ›

Vasanthakumar Soundararajan • 3 months ago
So, redundant entries are removed?
I mean mirror images cause same pre orders in trees.
• Reply • Share ›

Akash Devkumar • 3 months ago
1st example examples contradict the behaviour, how come {3,2} has 2 possible BST's ??
1 • Reply • Share ›

Manohar Singh Gour Akash Devkumar • 3 months ago
3 as a root and 2 as a root
• Reply • Share ›

yoyo • 3 months ago
Number of trees is N! I order traversal can be generated using algorithm for all combination of digits.

• Reply • Share ›

Nanu yoyo • 2 months ago

This is not true. If your inorder array is [4, 5, 7], then you only get 5 possibilities. The preorder permutation that is not valid is 5 7 4.

• Reply • Share ›

Aaron Chung • 3 months ago

#Python 2.7 Implementation
`````` represents 4 spaces

```
import itertools

class Node(object):
````def __init__(self, val):
````````self.val = val
````````self.left = None
````````self.right = None

def preorder(root):
````def preorder_helper(node):
````````if node is None:
````````````return
````````print node.val,
````````preorder_helper(node.left)
````````preorder_helper(node.right)
````preorder_helper(root)
```

see more

• Reply • Share ›

Vishwa Mohan • 3 months ago

JAVA implementation:

```
--------------------------
import java.util.ArrayList;
import java.util.List;

public class PrintAllTreeFromInorder {

public List<treenode> getListOfTrees(int[] inOrder , int start , int end){
List<treenode> trees = new ArrayList<treenode>() ;
if(start >end){
trees.add(null);
return trees ;
}
for(int i=start ;i<=end ;i++){
```
```

```
for(int i=start;i<=end;i++){
List<treenode> leftTrees = getListOfTrees(inOrder, start, i-1);
List<treenode> rightTrees = getListOfTrees(inOrder, i+1, end);
//Here inOrder[i] is taken as the node

//Below for loop will create all the combinations of the tree
```

see more

• Reply • Share ›

anubhav singh ➦ Vishwa Mohan • 3 months ago

anyone with implementation in C..??

• Reply • Share ›

GanesH AvacharE ➦ anubhav singh • 3 months ago

```
#include <stdio.h>
#include <stdlib.h>

typedef struct tree {
int data;
struct tree *left, *right;
}tree;

typedef struct return_value {
tree **list;
int no;
}return_value;

return_value gettrees(int *array, int start, int end) {
return_value trees, left_subtrees, right_subtrees;
tree *node;
int i, j, k;
// initialize trees
trees.list = NULL;
```

see more

• Reply • Share ›