Construct a special tree from given preorder traversal

Given an array ‘pre[]’ that represents Preorder traversal of a spacial binary tree where every node has either 0 or 2 children. One more array ‘preLN[]’ is given which has only two possible values ‘L’ and ‘N’. The value ‘L’ in ‘preLN[]’ indicates that the corresponding node in Binary Tree is a leaf node and value ‘N’ indicates that the corresponding node is non-leaf node. Write a function to construct the tree from the given two arrays.

Source: Amazon Interview Question

Example:

```
Input:  pre[] = {10, 30, 20, 5, 15},  preLN[] = {'N', 'N', 'L', 'L', 'L'}
Output: Root of following tree
          10
         /  \
        30   15
       /  \
      20   5
```

The first element in pre[] will always be root. So we can easily figure out root. If left subtree is empty, the right subtree must also be empty and preLN[] entry for root must be ‘L’. We can simply create a node and return it. If left and right subtrees are not empty, then recursively call for left and right subtrees and link the returned nodes to root.

C

```
/* A program to construct Binary Tree from preorder traversal */
#include<stdio.h>

/* A binary tree node structure */
struct node
{
    int data;
    struct node *left;
    struct node *right;
};
```

```
/* Utility function to create a new Binary Tree node */
struct node* newNode (int data)
{
    struct node *temp = new struct node;
    temp->data = data;
    temp->left = NULL;
    temp->right = NULL;
    return temp;
}

/* A recursive function to create a Binary Tree from given pre[]
   preLN[] arrays. The function returns root of tree. index_ptr is used
   to update index values in recursive calls. index must be initially
   passed as 0 */
struct node *constructTreeUtil(int pre[], char preLN[], int *index_ptr, int n)
{
    int index = *index_ptr; // store the current value of index in pre[]

    // Base Case: All nodes are constructed
    if (index == n)
        return NULL;

    // Allocate memory for this node and increment index for
    // subsequent recursive calls
    struct node *temp = newNode ( pre[index] );
    (*index_ptr)++;

    // If this is an internal node, construct left and right subtrees and link the subtrees
    if (preLN[index] == 'N')
    {
      temp->left  = constructTreeUtil(pre, preLN, index_ptr, n);
      temp->right = constructTreeUtil(pre, preLN, index_ptr, n);
    }

    return temp;
}

// A wrapper over constructTreeUtil()
struct node *constructTree(int pre[], char preLN[], int n)
{
    // Initialize index as 0. Value of index is used in recursion to maintain
    // the current index in pre[] and preLN[] arrays.
    int index = 0;

    return constructTreeUtil (pre, preLN, &index, n);
}


/* This function is used only for testing */
```

```
void printInorder (struct node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder (node->left);

    /* then print the data of node */
    printf("%d ", node->data);

    /* now recur on right child */
    printInorder (node->right);
}

/* Driver function to test above functions */
int main()
{
    struct node *root = NULL;

    /* Constructing tree given in the above figure
          10
         /  \
        30   15
       /  \
      20   5 */
    int pre[] = {10, 30, 20, 5, 15};
    char preLN[] = {'N', 'N', 'L', 'L', 'L'};
    int n = sizeof(pre)/sizeof(pre[0]);

    // construct the above tree
    root = constructTree (pre, preLN, n);

    // Test the constructed tree
    printf("Following is Inorder Traversal of the Constructed Binary Tree: \n");
    printInorder (root);

    return 0;
}
```

Java

```
// Java program to construct a binary tree from preorder traversal

// A Binary Tree node
class Node {

    int data;
    Node left, right;
```

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

class Index {

    int index = 0;
}

class BinaryTree {

    static Node root;
    Index myindex = new Index();

    /* A recursive function to create a Binary Tree from given pre[]
     preLN[] arrays. The function returns root of tree. index_ptr is used
     to update index values in recursive calls. index must be initially
     passed as 0 */
    Node constructTreeUtil(int pre[], char preLN[], Index index_ptr, int n, Node temp) {
        int index = index_ptr.index; // store the current value of index in pre[]

        // Base Case: All nodes are constructed
        if (index == n) {
            return null;
        }

        // Allocate memory for this node and increment index for
        // subsequent recursive calls
        temp = new Node(pre[index]);
        (index_ptr.index)++;

        // If this is an internal node, construct left and right subtrees and link the subtrees
        if (preLN[index] == 'N') {
            temp.left = constructTreeUtil(pre, preLN, index_ptr, n, temp.left);
            temp.right = constructTreeUtil(pre, preLN, index_ptr, n, temp.right);
        }

        return temp;
    }

    // A wrapper over constructTreeUtil()
    Node constructTree(int pre[], char preLN[], int n, Node node) {

        // Initialize index as 0. Value of index is used in recursion to maintain
        // the current index in pre[] and preLN[] arrays.
        int index = 0;
```

```
        return constructTreeUtil(pre, preLN, myindex, n, node);
    }

    /* This function is used only for testing */
    void printInorder(Node node) {
        if (node == null) {
            return;
        }

        /* first recur on left child */
        printInorder(node.left);

        /* then print the data of node */
        System.out.print(node.data + " ");

        /* now recur on right child */
        printInorder(node.right);
    }

    // driver function to test the above functions
    public static void main(String args[]) {
        BinaryTree tree = new BinaryTree();
        int pre[] = new int[]{10, 30, 20, 5, 15};
        char preLN[] = new char[]{'N', 'N', 'L', 'L', 'L'};
        int n = pre.length;

        // construct the above tree
        Node mynode = tree.constructTree(pre, preLN, n, root);

        // Test the constructed tree
        System.out.println("Following is Inorder Traversal of the Constructed Binary Tree: ");
        tree.printInorder(mynode);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Output:

```
Following is Inorder Traversal of the Constructed Binary Tree:
20 30 5 10 15
```

Time Complexity: O(n)

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

20 Comments Category: Trees

Related Posts:

- Check if removing an edge can divide a Binary Tree in two halves
- Check sum of Covered and Uncovered nodes of Binary Tree
- 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?

(Login to Rate and Mark)

3 Average Difficulty : **3/5.0**
Based on **2** vote(s)

Add to TODO List
Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.
prashant
below standard question
prakash
"If left subtree is empty, the right subtree must also be empty and preLN[] entry for root must be 'L'." Why? This is not mentioned in the question. Am I missing something
rahul
Do we really need to index_ptr as pointer to maintain the index? Just passing by value should suffice right?
Amit Bgl
wow code 😀

eric wu
No need to use n, the recursion will exit by itself when it reaches all the leaves.
eric wu
No need to use n, the recursion will exit by itself when it reaches all the leaves.

```
/* Paste your code here (You may delete these lines if not writing code) */
```

abhishek08aug
Intelligent 😀
xiaoc10

```
    if (index == n)
        return NULL;
```

Why the above two lines are necessary?
eric wu
No need to use n, the recursion will exit after it reaches all the leaves.
L

```
It shud be
    if (preLN[index] == 'N')
    {
      temp->left  = constructTreeUtil(pre, preLN, index_ptr, n);
      temp->right = constructTreeUtil(pre, preLN, index_ptr, n);
    } else {
      temp->left = NULL;
      temp->right = NULL;
    }
```

Sreenivas Doosa
@L:
You don`t need to add the else condition to set left and right child to NULL. Because it has already been set to NULL when you create a New Node..
Gopika
I dont under stand where is ‘n’ coming from. Can you please explain.
Ankit Gupta
If you are talking about the ‘n’ in the order O(n). It is from the running time of the recursive call constructTreeUtil(). @var index_ptr takes values in the range [0, n). Hence the order.
wakeup123
n is the size of the array pre[], as well as preLN[]. it is being passed to the function constructTree(), while calling the function in the main. As you can see below……
int n = sizeof(pre)/sizeof(pre[0]);
// construct the above tree
root = constructTree (pre, preLN, n);
Priyank
Why is this true: “If left subtree is empty, the right subtree must also be empty” ?
kartik
As per the problem statement, every node has either 0 or 2 children.
Gopika
I am not clear about where is ‘n’ coming from.
Can you please explain.
Thanks.

```
/* Paste your code here (You may delete these lines if not writing code) */
```

kartik
n is size of input arrays and size of tree.
ritesh
Thats because every node has 2 or 0 children as per question.So it must be a complete binary tree.Therefore as per its preorder style of traversal,if the left subtree is empty then its not possible to get a right subtree.There has to have a leftsubtree at first.