Construct Binary Tree from Preorder and Inorder Traversal

Given preorder and inorder traversal of a tree, construct the binary tree.

Note:
You may assume that duplicates do not exist in the tree.

Solution 1

I din't find iterative solutions discussed in the old Discuss. So, I thought, I will add my solution in here.

The idea is as follows:

1) Keep pushing the nodes from the preorder into a stack (and keep making the tree by adding nodes to the left of the previous node) until the top of the stack matches the inorder.

2) At this point, pop the top of the stack until the top does not equal inorder (keep a flag to note that you have made a pop).

3) Repeat 1 and 2 until preorder is empty. The key point is that whenever the flag is set, insert a node to the right and reset the flag.

```
class Solution {
public:
    TreeNode *buildTree(vector<int> &preorder, vector<int> &inorder) {

        if(preorder.size()==0)
            return NULL;

        stack<int> s;
        stack<TreeNode *> st;
        TreeNode *t,*r,*root;
        int i,j,f;

        f=i=j=0;
        s.push(preorder[i]);

        root = new TreeNode(preorder[i]);
        st.push(root);
        t = root;
        i++;

        while(i<preorder.size())
        {
            if(!st.empty() && st.top()->val==inorder[j])
            {
                t = st.top();
                st.pop();
                s.pop();
                f = 1;
                j++;
            }
            else
            {
                if(f==0)
                {
                    s.push(preorder[i]);
                    t -> left = new TreeNode(preorder[i]);
                    t = t -> left;
                    st.push(t);
                    i++;
                }
                else
                {
                    f = 0;
                    s.push(preorder[i]);
                    t -> right = new TreeNode(preorder[i]);
                    t = t -> right;
                    st.push(t);
                    i++;
                }
            }
        }

        return root;
    }
};
```

written by gpraveenkumar original link here

Solution 2

Hi guys, this is my Java solution. I read this post, which is very helpful.

The basic idea is here: Say we have 2 arrays, PRE and IN. Preorder traversing implies that PRE[0] is the root node. Then we can find this PRE[0] in IN, say it's IN[5]. Now we know that IN[5] is root, so we know that IN[0] - IN[4] is on the left side, IN[6] to the end is on the right side. Recursively doing this on subarrays, we can build a tree out of it :)

Hope this helps.

```
public TreeNode buildTree(int[] preorder, int[] inorder) {
    return helper(0, 0, inorder.length - 1, preorder, inorder);
}

public TreeNode helper(int preStart, int inStart, int inEnd, int[] preorder, int[]
inorder) {
    if (preStart > preorder.length - 1 || inStart > inEnd) {
        return null;
    }
    TreeNode root = new TreeNode(preorder[preStart]);
    int inIndex = 0; // Index of current root in inorder
    for (int i = inStart; i <= inEnd; i++) {
        if (inorder[i] == root.val) {
            inIndex = i;
        }
    }
    root.left = helper(preStart + 1, inStart, inIndex - 1, preorder, inorder);
    root.right = helper(preStart + inIndex - inStart + 1, inIndex + 1, inEnd, pre
order, inorder);
    return root;
}
```

written by jiaming2 original link here

Solution 3

```
TreeNode *buildTree(vector<int> &preorder, vector<int> &inorder) {
    return create(preorder, inorder, 0, preorder.size() - 1, 0, inorder.size() -
1);
}

TreeNode* create(vector<int>& preorder, vector<int>& inorder, int ps, int pe, int
is, int ie){
    if(ps > pe){
        return nullptr;
    }
    TreeNode* node = new TreeNode(preorder[ps]);
    int pos;
    for(int i = is; i <= ie; i++){
        if(inorder[i] == node->val){
            pos = i;
            break;
        }
    }
    node->left = create(preorder, inorder, ps + 1, ps + pos - is, is, pos - 1);
    node->right = create(preorder, inorder, pe - ie + pos + 1, pe, pos + 1, ie);
    return node;
}
```

The first element in preorder array can divide inorder array into two parts. Then we can divide preorder array into two parts. Make this element a node. And the left sub-tree of this node is the left part, right sub-tree of this node is the right part. This problem can be solved following this logic.

written by zxyperfect original link here