

# 105. Construct Binary Tree from Preorder and Inorder Traversal

## 105 Construct Binary Tree from Preorder and Inorder Traversal

### Description

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

**Note:**

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

Preorder: 根左右

Inorder: 左根右

### Solution

- Tree
- Depth-first Search
- Array
- Stack

### Depth-first Search

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
```

### Stack

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;
    }
```

```

int i = 0; // index for perorder
int j = 0; // index for inorder
int f = 0; // insert right child flag

TreeNode* root = new TreeNode(preorder[i]);
TreeNode* cur = root;

stack<TreeNode*> st; //
st.push(root);
i++;

while(i<preorder.size())
{
    if(!st.empty() && st.top()->val==inorder[j])
    {
        cur = st.top();
        st.pop();
        f = 1;
        j++;
    }
    else
    {
        if(f==0)
        {
            cur -> left = new TreeNode(preorder[i]);
            cur = cur -> left;
            st.push(cur);
            i++;
        }
        else // whenever the flag is set, insert a node to the right and reset the flag.
        {
            f = 0;
            cur -> right = new TreeNode(preorder[i]);
            cur = cur -> right;
            st.push(cur);
            i++;
        }
    }
}
return root;
}
};

```