105&& 106 Construct Binary Tree from Inorder and Preorder / Postorder Traversal

Category	Tree Traversal
Difficulty	Medium
Note	
AC & Time	
Property	
related	

105. Construct Binary Tree from Inorder and Preorder Traversal

```
O(N^2) time; search cost so many time. need to do time and space trade-off
O(1)
using hashmap for searching.


TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        //preorder can give root information
        //inorder can give left and right children information
        if (preorder.size() == 0) return NULL;
        int rootId = 0;
        return bulidTreewithPI(preorder, inorder, rootId, 0, inorder.size()-1);

    }

private:
    TreeNode* bulidTreewithPI(vector<int>& preorder, vector<int>& inorder, int& rootId, int left, int right) {
             if (left > right) {
                 return NULL;
             }
            TreeNode* root = new TreeNode(preorder[rootId]);
            int pivot = left;
            while(inorder[pivot] != preorder[rootId])
                    pivot++;
            rootId++;
            root->left = bulidTreewithPI(preorder, inorder, rootId, left, pivot - 1);
            root->right = bulidTreewithPI(preorder, inorder, rootId, pivot + 1, right);
            return root;

    }

};

// using hashmap vison
unordered_map<int, int> hash;
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        if (preorder.size() == 0) return nullptr;
        for (int i = 0 ; i < inorder.size(); i++)
            hash[inorder[i]] = i;
        int rootId = 0;
        return buildTreeM(preorder, inorder, rootId, 0, inorder.size() - 1);
```

```
    }

private:
       TreeNode* buildTreeM(vector<int>& preorder, vector<int>& inorder, int& rootId, int in_left, int in_right) {
           if (in_left > in_right) return nullptr;
           TreeNode* root = new TreeNode(preorder[rootId++]);
           int pivot = hash[preorder[rootId]];
           root->left = buildTreeM(preorder, inorder, rootId, in_left, pivot -1);
           root->right = buildTreeM(preorder, inorder, rootId, pivot + 1, in_right);
           return root;
       }

// the time cost search change to O(1)
// the time complexity O(n), space O(n);
```

106 . **Construct Binary Tree from Inorder and Postorder Traversal**

```
class Solution {
public:
    unordered_map<int, int> hash;
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        if (postorder.size() == 0) return nullptr;
        for (int i = 0; i < inorder.size(); i++) {
            hash[inorder[i]] = i;
        }
        int rootId = postorder.size() - 1;
        return buildTreeHelper(inorder, postorder, 0, inorder.size() - 1, rootId);
    }

private:
    TreeNode* buildTreeHelper(vector<int>& inorder, vector<int>& postorder, int left_in, int right_in, int& rootId) {
        if (left_in > right_in) return nullptr;
        int pivot = hash[postorder[rootId]];
        TreeNode* root = new TreeNode(postorder[rootId--]);
        root->right = buildTreeHelper(inorder, postorder, pivot + 1, right_in, rootId);
        root->left = buildTreeHelper(inorder, postorder, left_in, pivot - 1, rootId);
        return root;

    }
};

using hash map record the inorder list. time O(n), space O(n)
```