利用前序、中序（后序)构建二叉树

想起最近做到的两道利用前序、中序以及利用中序、后序构建二叉树的题，其也广泛出现在面试题中的选择题中，现整理总结如下。

1. 利用前序、中序构建二叉树

LeetCode 105. [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.

For example, given

```
preorder = [3,9,20,15,7]
inorder = [9,3,15,20,7]
```

Return the following binary tree:

```
    3
   / \
  9  20
    /  \
   15   7
```

代码如下：

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        if (preorder.empty() || inorder.empty()) {
            return NULL;
        }
        return buildTreeCore(preorder, 0, preorder.size() - 1, inorder, 0,
inorder.size() - 1);
```

```
    }


    TreeNode* buildTreeCore(vector<int>& preorder, int pre_begin, int pre_end,
vector<int>& inorder, int in_begin, int in_end) {
        if (pre_begin > pre_end) return NULL;
        TreeNode* root = new TreeNode(preorder[pre_begin]);
        for (int i = 0; i <= pre_end - pre_begin; ++i) {
            if (preorder[pre_begin] == inorder[in_begin + i]) {
                root->left = buildTreeCore(preorder, pre_begin + 1, pre_begin
+ i, inorder, in_begin, in_begin + i - 1);
                root->right = buildTreeCore(preorder, pre_begin + i + 1,
pre_end, inorder, in_begin + i + 1, in_end);
            }
        }
        return root;
    }
};
```

其中需要注意两个点：

1. 为了保持这类题的一致解题思路，这里最好传入的参数为 `preorder.size() - 1` 和 `inorder.size() - 1`，虽然直接传入 `preorder.size()` 和 `inorder.size()` 也可以，但是这个做法在利用中序、后序数据构建二叉树时相对麻烦，故而最好统一；
2. 对应判定条件 `for (int i = 0; i <= pre_end - pre_begin; ++i)`，这种判定条件其实处理起来也是相对比较方便，当然，也可以利用 `for (int i = in_begin; i <= in_end; ++i)` 这种处理，然后利用 `if (preorder[pre_begin] == inorder[i])` 进行判断， 但是后面利用中序或者后序进行构建组织索引时就会比较麻烦，如本题就需要更新为：`root->left = buildTreeCore(preorder, pre_begin + 1, i - in_begin + pre_begin, inorder, in_begin, i - 1);` 以及 `root->right = buildTreeCore(preorder, i - in_begin + 1 + pre_begin, pre_end, inorder, i + 1, in_end);`

2. 利用中序、后序构建二叉树

LeetCode 106. Construct Binary Tree from Inorder and Postorder Traversal

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

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

For example, given

```
inorder = [9,3,15,20,7]
postorder = [9,15,7,20,3]
```

Return the following binary tree:

```
    3
   / \
  9  20
    /  \
   15   7
```

代码如下：

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
        if (inorder.empty() || postorder.empty()) return NULL;
        return buildTreeCore(inorder, 0, inorder.size() - 1, postorder, 0,
postorder.size() - 1);
    }


    TreeNode* buildTreeCore(vector<int>& inorder, int in_front, int in_back,
vector<int>& postorder, int post_front, int post_back) {
        if (in_front > in_back) return NULL;
        TreeNode* root = new TreeNode(postorder[post_back]);
        for (int i = 0; i <= in_back - in_front; ++i) {
            if (postorder[post_back] == inorder[in_front + i]) {
                root->left = buildTreeCore(inorder, in_front, in_front + i -
1, postorder, post_front, post_front + i - 1);
                root->right =  buildTreeCore(inorder, in_front + i + 1,
in_back, postorder, post_front + i, post_back - 1);
            }
        }
        return root;
    }

};
```

这里就体现出来传参的简约性：`buildTreeCore(inorder, 0, inorder.size() - 1, postorder, 0, postorder.size() - 1);`

以及判定条件：`for (int i = 0; i <= in_back - in_front; ++i)` 和 `if (postorder[post_back] == inorder[in_front + i])`。

3. 总结

上述利用前序、中序构建二叉树与利用中序、后序构建二叉树的解题思路是近似的；

但是要注意传参，如自己在利用中序、后序构建二叉树时 `buildTreeCore(inorder, 0, inorder.size(), postorder, 0, postorder.size());` 然后直接构建二叉树为 `TreeNode* root = new TreeNode(postorder[post_back])`，造成错误，故而以后都统一传参为 `buildTreeCore(inorder, 0, inorder.size() - 1, postorder, 0, postorder.size() - 1);` 比较方便；