```
/*-------------------------------------------------------------+
|leetcode105 <rebuild binary tree>                             |
|给出一颗二叉树的前序与中序序列，重建这颗二叉树，保证没有重复的值 |
+-------------------------------------------------------------*/
思路：对于一棵二叉树的前序与中序序列，可以唯一确定这颗二叉树，只需要确定根节点的位置。
在前序遍历中，根节点在第一个位置，继而在中序遍历中找到这个节点，那么左侧便是左子树的
所有节点，右侧便是右子树的所有节点，再在前序遍历中找出这一段，其第一个节点则是该子树
的根节点，继而可以一步步确定这棵树。
递归每次相当于确定根节点位置，当前节点为root,则左子树的根就是root->left
<边界>:
    1.递归的终止条件：序列的开始index大于结束的index
    2.每次递归的分化：

    preorder:  [ 3][ 9][20][15][ 7]
                ^
                |
                root

    inorder:    [ 9][ 3][15][20][ 7]
                    ^
                    |
                    pos_root

    故左子树的俩个对应序列为 pre[beg_pre+1,beg_pre+num_left]
                             in[beg_in,pos_root-1]
    右子树的两个对应的序列为 pre[beg_pre+num_left+1,end_pre]
                             in[pos_root+1,end_in]


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

    }

    TreeNode *_buildTree(vector<int> &preorder, vector<int> &inorder,
                     int beg_pre, int end_pre, int beg_in, int end_in)
    {
        /*递归终止条件*/
        if (beg_pre > end_pre) { return nullptr; }
        if (beg_in > end_in) { return nullptr; }

        TreeNode *root = new TreeNode(preorder[beg_pre]);
        int pos_root = 0;
        /*找根*/
        for (int i = beg_in; i <= end_in; i++) {
                if (inorder[i] == preorder[beg_pre]) {
                        pos_root = i;
                        break;
                }
        }
        int left_num = pos_root - beg_in;
        /*左右分别递归*/
        root->left = _buildTree(preorder, inorder,
                                beg_pre + 1, beg_pre + left_num, beg_in, pos_root-1);
        root->right = _buildTree(preorder, inorder,
                                 beg_pre + left_num+1 , end_pre, pos_root+1 , end_in);
        return root;
    }
};
```