莫里斯遍历

有关二叉树的遍历算法

非递归，无额外空间，时间复杂度O(n) 空间复杂度O(1)

很巧妙的遍历算法

核心思想就是利用树节点中的空指针

考虑非递归算法，如果我们不用栈的话，最主要的问题就是遍历完一个节点的左子树后怎么回到这个节点并遍历他的右子树

在遍历左子树的时候，最后一个遍历的节点一定是二叉树中序遍历中，当前节点的前一个节点

也就是当前节点左子树的最右边的节点

我们可以把这个前驱节点的右子树设为当前节点，这样遍历完左子树的时候，也就是遍历完这个前驱节点的时候，我们可以通过先前设置的那个指针回到当前节点，这样我们就可以继续进行右子树的遍历

那么思路也就来了，对于每一个节点，找到他在中序遍历中的前驱节点，即左子树的最右节点，将该节点的右子树设置为当前节点，然后遍历左子树，那么我们一定会通过我们设置的那个指针回到当前节点，我们第二次再去找前驱节点，如果发现前驱节点的右子树已经是当前节点，说明这是第二次来到当前节点，于是我们就可以确定左子树已经遍历完毕，便可以进行右子树的遍历了。

于是我们的算法就来了

1. 新建临时节点，令该节点为`root`；
2. 如果当前节点的左子节点为空，遍历当前节点的右子节点；
3. 如果当前节点的左子节点不为空，在当前节点的左子树中找到当前节点在中序遍历下的前驱节点：
    - 如果前驱节点的右子节点为空，将前驱节点的右子节点设置为当前节点。然后将当前节点加入答案，并将前驱节点的右子节点更新为当前节点。
    - 如果前驱节点的右子节点为当前节点将它的右子节点重新设置为空。当前节点更新为当前节点的右子节点。
4. 重复步骤2和步骤3，直到遍历结束。

这里我给出前序遍历的代码，结合代码可以对算法有深一步的理解

```cpp
class Solution {
public:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> ans;
        TreeNode* now = root;
        while (now != nullptr) {
            if (now->left != nullptr) {
                TreeNode* temp = now->left;
                while (temp->right != nullptr && temp->right != now) {
                    temp = temp->right;
                }
                if (temp->right == nullptr) {
                    temp->right = now;
                    ans.push_back(now->val);
                    now = now->left;
                } else {
                    temp->right = nullptr;
                    now = now->right;
```

```
                }
            } else {
                ans.push_back(now->val);
                now = now->right;
            }
        }
        return ans;
    }
};
```

那么前序遍历的代码看懂的话，相比中序也一定就会了（如果不会的话就再仔细理解一下算法的原理）

那么对于后序遍历呢？难道我们要在当前节点的最右边节点上也设置一个指针指向当前节点？然后等第三次遍历吗

其实不需要的，我们仔细考虑后序遍历的特性，可以发现，对于一颗二叉树而言，他的后序遍历一定先是所有的左子树，再是右子树，那么什么算是左子树呢？

当前节点的左子树，以及当前节点到其最右边节点的路径上的节点的所有左子树，都算是。

因为这些左子树次序，都一定在当前节点到其最右边节点的路径上的节点之前

再仔细想，后序遍历的最后几个节点一定是根节点到其最右边节点的路径上的节点的逆序

所以我们可以将一颗二叉树*斜着*划分，对于每一个节点，他到他最右边节点的这条路径上的点的集合，划分到一起。

那么这颗二叉树的后序遍历就是由这些斜线组成的若干段所组成的，从左到右，从下到上（逆序）

最后给出后序遍历的代码结合理解

```
void Add(TreeNode* now, vector<int> &ans) {
        vector<int>temp;
        while(now) {
            temp.push_back(now->val);
            now = now->right;
        }
        reverse(temp.begin(), temp.end());
        for (auto &x : temp) {
            ans.push_back(x);
        }
    }
    vector<int> postorderTraversal(TreeNode* root) {
        vector<int>ans;
        if (!root) return ans;
        TreeNode *rt = root;
        while(root) {
            if (root->left == nullptr) {
                root = root->right;
            } else {
                TreeNode* now = root->left;
                while(now->right != nullptr && now->right != root) now = now->right;
                if (now->right == root) {
                    now->right = nullptr;
                    Add(root->left, ans);
                    root = root->right;
                } else {
                    now->right = root;
                    root = root->left;
```

```
                }
            }
        }
        Add(rt, ans);
        return ans;
    }
```