二叉树的非递归遍历

2022年8月15日 10:14

先序遍历

```
class Solution {
public:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> result;
        stack<TreeNode*> s;
        TreeNode* node=root;
        while(node||!s.empty()){
            while(node){
                result.push_back(node->val);
                s.push(node);
                node=node->left;
            }
            node=s.top();
            s.pop();
            node=node->right;
        }
        return result;
    }

};
```

迭代法统一版本

```
class Solution {
public:
    vector<int> preorderTraversal(TreeNode* root) {
        vector<int> ans;
        stack<TreeNode* > st;
        if(root) st.push(root);
        TreeNode* node=nullptr;
        while(!st.empty()){
            node=st.top();
            if(node){
                st.pop();
                if(node->right) st.push(node->right);
                if(node->left) st.push(node->left);
                st.push(node);
                st.push(nullptr);
            }
            else{
                st.pop();
                node=st.top();
                st.pop();
                ans.push_back(node->val);
            }
        }
        return ans;
    }
};
```

中序遍历

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> result;
        stack<TreeNode* >s;
        TreeNode* node=root;
        while(node||!s.empty()){
            while(node){
                s.push(node);
                node=node->left;
            }
            node=s.top();
            s.pop();
            result.push_back(node->val);
            node=node->right;
        }
        return result;
    }
};
```

迭代法统一版本

```
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> ans;
        stack<TreeNode* >st;
        if(root) st.push(root);
        while(!st.empty()){
            TreeNode* node=st.top();
            if(node){
                st.pop();
                if(node->right) st.push(node->right);
                st.push(node);
                st.push(nullptr);
                if(node->left) st.push(node->left);
            }
            else{
                st.pop();
                node=st.top();
                st.pop();
                ans.push_back(node->val);
            }
        }
        return ans;
    }
};
```

迭代法统一版本

后序遍历

```
class Solution {
public:
    vector<int> postorderTraversal(TreeNode* root) {
        vector<int> result;
        stack<TreeNode*> s;
        TreeNode* node=root;
        TreeNode* pre=nullptr;
        while(node||!s.empty()){
            while(node){
                s.push(node);
                node=node->left;
            }
            node=s.top();
            if(!node->right||node->right==pre){
                result.push_back(node->val);
                s.pop();
                pre=node;
                node=nullptr;
            }
            else{
                node=node->right;
            }
        }
        return result;
```

由于后续遍历是先访问左子树L 右子树R，最后访问根节点root
因此当遍历到右子树的根节点R_root后，栈中弹出，此时栈顶元素为根节点root
若不加此判断条件，会因为root的右孩子不为空，而将root->right压入栈中，导致死循环。

需要给node节点进行赋值，以防元素重复入栈
注意！此处不可写为：

```
while(node||!s.empty()){
    while(node){
        s.push(node);
        node=node->left;
```

```
            node=node->right;
        }
    }
    return result;
}
};
```

```
    while(node){
        s.push(node);
        node=node->left;
    }
    node=s.top();
    if(!node->right||node->right==pre){
        result.push_back(node->val);
        s.pop();
        pre=node;
    }
    node=node->right;
}
return result;
```

因为会出现上述的问题 导致元素重复入栈！！
（说白了 node->right不一定为空）

迭代法统一版本

```
class Solution {
public:
    vector<int> postorderTraversal(TreeNode* root) {
        vector<int> ans;
        stack<TreeNode*> st;
        if(root) st.push(root);
        while(!st.empty()){
            TreeNode* node=st.top();
            if(node){
                st.pop();
                st.push(node);
                st.push(nullptr);
                if(node->right) st.push(node->right);
                if(node->left) st.push(node->left);
            }
            else{
                st.pop();
                node=st.top();
                st.pop();
                ans.push_back(node->val);
            }
        }
        return ans;
    }
};
```