二叉树先序、中序、后序遍历的非递归实现

二叉树先序、中序、后序遍历的非递归实现

Filed in: C++学习, 数据结构与算法在网上看了一些用非递归实现先序中序后序 Add comments 遍历二叉树的代码，都很混乱，while、if各种组合嵌套使用，逻辑十分不清晰，把我也搞懵了。想了大半天，写了大半天，突然开了窍，实际上二叉树的这三种遍历在逻辑上是十分清晰的，所以才可以用递归实现的那么简洁。既然逻辑如此清晰，那么用非递归实现也应该是清晰的。

自认为自己的代码比网上搜到的那些都清晰得多，好理解得多。

稍微解释一下：

先序遍历。将根节点入栈，考察当前节点（即栈顶节点），先访问当前节点，然后将其出栈（已经访问过，不再需要保留），然后先将其右孩子入栈，再将其左孩子入栈（这个顺序是为了让左孩子位于右孩子上面，以便左孩子的访问先于右孩子；当然如果某个孩子为空，就不用入栈了）。如果栈非空就重复上述过程直到栈空为止，结束算法。

中序遍历。将根节点入栈，考察当前节点（即栈顶节点），如果其左孩子未被访问过（有标记），则将其左孩子入栈，否则访问当前节点并将其出栈，再将右孩子入栈。如果栈非空就重复上述过程直到栈空为止，结束算法。

后序遍历。将根节点入栈，考察当前节点（即栈顶节点），如果其左孩子未被访问过，则将其左孩子入栈，否则如果其右孩子未被访问过，则将其右孩子入栈，如果都已经访问过，则访问其自身，并将其出栈。如果栈非空就重复上述过程直到栈空为止，结束算法。

其实，这只不过是保证了先序中序后序三种遍历的定义。对于先序，保证任意一个节点先于其左右孩子被访问，还要保证其左孩子先于右孩子被访问。对于中序，保证任意一个节点，其左孩子先于它被访问，右孩子晚于它被访问。对于后序，保证任意一个节点的左孩子右孩子都先于它被访问，其中左孩子先于右孩子被访问。如是而已。

代码里应该体现得比较清楚。这里不光给出了非递归版本，也给出了递归版本。

```
#include <iostream>
#include <stack>

using namespace std;

struct TreeNode
{
        int data;
```

```
        TreeNode* left;
        TreeNode* right;
        int flag;
};

typedef TreeNode *TreePtr;

TreePtr CreateTree()
{
        TreePtr root = new TreeNode;
        cout<<"input the data :\n";
        int n;
        cin>>n;
        if (n == -1)
        {
                return NULL;
        }
        else
        {
                root->data = n;
                root->flag = 0;
                root->left = CreateTree();
                root->right = CreateTree();
        }
        return root;
}

void PreOrderRecursion(TreePtr p)
{
        if (p == NULL)
        {
                return;
        }
        cout<<p->data<<" ";
        PreOrderRecursion(p->left);
        PreOrderRecursion(p->right);
}

void InOrderRecursion(TreePtr p)
{
        if (p == NULL)
        {
                return;
        }
        InOrderRecursion(p->left);
        cout<<p->data<<" ";
        InOrderRecursion(p->right);
}

void PostOrderRecursion(TreePtr p)
{
        if (p == NULL)
        {
```

```
                return;
        }
        PostOrderRecursion(p->left);
        PostOrderRecursion(p->right);
        cout<<p->data<<" ";
}

void PreOrderNoRecursion(TreePtr p)
{
        cout<<"PreOrderNoRecursion\n";

        stack<TreeNode> stk;
        TreeNode t = *p;
        stk.push(t);

        while (!stk.empty())
        {
                t = stk.top();
                stk.pop();
                cout<<t.data<<" ";

                if (t.right != NULL)
                {
                        stk.push((*t.right));
                }

                if (t.left != NULL)
                {
                        stk.push((*t.left));
                }
        }
}

void InOrderNoRecursion(TreePtr p)
{
        cout<<"InOrderNoRecursion\n";

        stack<TreeNode> stk;
        TreeNode t = *p;
        stk.push(t);

        while (!stk.empty())
        {
                if (stk.top().flag == 0)
                {
                        stk.top().flag++;
                        if (stk.top().left != NULL)
                        {
                                stk.push(*(stk.top().left));
                        }
                }
                else
                {
                        t = stk.top();
```

```
                        stk.pop();
                        cout<<t.data<<" ";
                        if (t.right != NULL)
                        {
                                stk.push(*(t.right));
                        }
                }
        }
}

void PostOrderNoRecursion(TreePtr p)
{
        cout<<"PostOrderNoRecursion\n";

        stack<TreeNode> stk;
        TreeNode t = *p;
        stk.push(t);

        while (!stk.empty())
        {
                if (stk.top().flag == 0)
                {
                        stk.top().flag++;
                        if (stk.top().left != NULL)
                        {
                                stk.push(*(stk.top().left));
                        }
                }
                else if (stk.top().flag == 1)
                {
                        stk.top().flag++;
                        if (stk.top().right != NULL)
                        {
                                stk.push(*(stk.top().right));
                        }
                }
                else
                {
                        cout<<stk.top().data<<" ";
                        stk.pop();
                }
        }
}

int main()
{
        TreePtr t = CreateTree();

        cout<<"PreOrderRecursion\n";
        PreOrderRecursion(t);
        cout<<"\n";

        cout<<"InOrderRecursion\n";
```

```
        InOrderRecursion(t);
        cout<<"\n";

        cout<<"PostOrderRecursion\n";
        PostOrderRecursion(t);
        cout<<"\n";

        PreOrderNoRecursion(t);
        cout<<"\n";

        InOrderNoRecursion(t);
        cout<<"\n";

        PostOrderNoRecursion(t);
        cout<<"\n";
}
```

来自：勤奋不止 > 《数据结构和算法》 +关注 与TA对话

上一篇：静态库和动态库的总结

下一篇：连通图的割点、割边(桥)、块、缩点，有向图的强连通分量



发表评论：

您好，请 登录 或者 注册 后再进行评论

其它帐号登录：