阅读：10,544　　作者：解学武

二叉树前序遍历、中序遍历和后序遍历及C语言递归实现

链式存储结构存储的二叉树，对树中结点进行逐个遍历时，由于是非线性结构，需要找到一种合适的方式遍历树中的每个结点。

递归思想遍历二叉树

之前讲过，树是由根结点和子树部分构建的，对于每一棵树来说，都可以分为 3 部分：左子树、根结点和右子树。所以，可以采用递归的思想依次遍历每个结点。

根据访问结点时机的不同，分为三种遍历方式：

- 先访问根结点，再遍历左右子树，称为“先序遍历”；
- 遍历左子树，之后访问根结点，然后遍历右子树，称为“中序遍历”；
- 遍历完左右子树，再访问根结点，称为“后序遍历”。

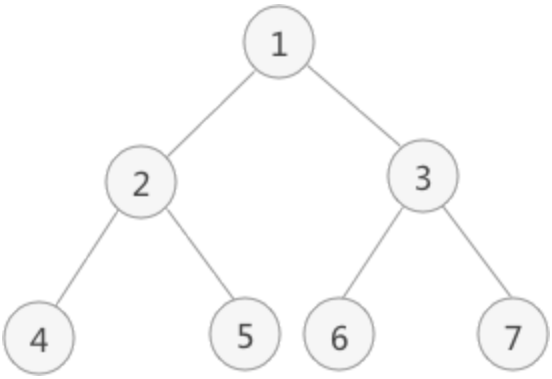

图1 二叉树

三种方式唯一的不同就是访问结点时机的不同，给出一个二叉树，首先需要搞清楚三种遍历方式下访问结点的顺序。

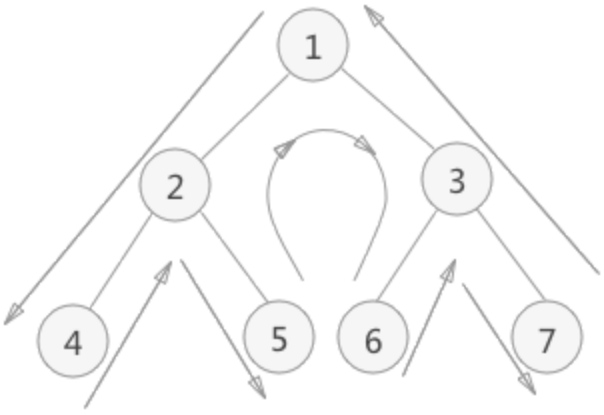

图2 二叉树遍历示意图

图2 中，箭头线条的走势为遍历结点的过程：

先序遍历是只要线条走到该结点的左方位置时，就操作该结点。所以操作结点的顺序为：

```
1 2 4 5 3 6 7
```

中序遍历是当线条越过结点的左子树，到达该结点的正下方时，才操作该结点。所以操作结点的顺序为：

```
4 2 5 1 6 3 7
```

后序遍历是线条完全走过结点的左右子树，到达该结点的右方范围时，就开始操作该结点。所以操作结点的顺序为：

```
4 5 2 6 7 3 1
```

三种遍历方式的完整代码实现

```c
#include <stdio.h>
#include <string.h>
#define TElemType int
//构造结点的结构体
typedef struct BiTNode{
    TElemType data;//数据域
    struct BiTNode *lchild,*rchild;//左右孩子指针
}BiTNode,*BiTree;
//初始化树的函数
void CreateBiTree(BiTree *T){
    *T=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->data=1;
    (*T)->lchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->rchild=(BiTNode*)malloc(sizeof(BiTNode));

    (*T)->lchild->data=2;
    (*T)->lchild->lchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->lchild->rchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->lchild->rchild->data=5;
    (*T)->lchild->rchild->lchild=NULL;
    (*T)->lchild->rchild->rchild=NULL;
    (*T)->rchild->data=3;
    (*T)->rchild->lchild=(BiTNode*)malloc(sizeof(BiTNode));
    (*T)->rchild->lchild->data=6;
    (*T)->rchild->lchild->lchild=NULL;
```

```
26.      (*T)->rchild->lchild->rchild=NULL;
27.      (*T)->rchild->rchild=(BiTNode*)malloc(sizeof(BiTNode));
28.      (*T)->rchild->rchild->data=7;
29.      (*T)->rchild->rchild->lchild=NULL;
30.      (*T)->rchild->rchild->rchild=NULL;
31.      (*T)->lchild->lchild->data=4;
32.      (*T)->lchild->lchild->lchild=NULL;
33.      (*T)->lchild->lchild->rchild=NULL;
34.  }
35.
36.  //模拟操作结点元素的函数，输出结点本身的数值
37.  void displayElem(BiTNode* elem){
38.      printf("%d ",elem->data);
39.  }
40.  //先序遍历
41.  void PreOrderTraverse(BiTree T){
42.      if (T) {
43.          displayElem(T);//调用操作结点数据的函数方法
44.          PreOrderTraverse(T->lchild);//访问该结点的左孩子
45.          PreOrderTraverse(T->rchild);//访问该结点的右孩子
46.      }
47.      //如果结点为空，返回上一层
48.      return;
49.  }
50.  //中序遍历
51.  void INOrderTraverse(BiTree T){
52.      if (T) {
53.          INOrderTraverse(T->lchild);//遍历左孩子
54.          displayElem(T);//调用操作结点数据的函数方法
55.          INOrderTraverse(T->rchild);//遍历右孩子
56.      }
57.      //如果结点为空，返回上一层
58.      return;
59.  }
60.  //后序遍历
61.  void PostOrderTraverse(BiTree T){
62.      if (T) {
63.          PostOrderTraverse(T->lchild);//遍历左孩子
64.          PostOrderTraverse(T->rchild);//遍历右孩子
65.          displayElem(T);//调用操作结点数据的函数方法
66.      }
67.      //如果结点为空，返回上一层
68.      return;
69.  }
70.  int main() {
71.      BiTree Tree;
72.      CreateBiTree(&Tree);
```

```
73.       printf("前序遍历: \n");
74.       PreOrderTraverse(Tree);
75.       printf("\n中序遍历算法: \n");
76.       INOrderTraverse(Tree);
77.       printf("\n后序遍历: \n");
78.       PostOrderTraverse(Tree);
79.   }
```

运行结果:

```
前序遍历:
1 2 4 5 3 6 7
中序遍历算法:
4 2 5 1 6 3 7
后序遍历:
4 5 2 6 7 3 1
```

总结

由于二叉树就是由根结点和左右子树构成的，所以很容易想到使用递归的思想。而递归算法的低层实现实际上使用的是栈的数据结构，所以二叉树的先序、中序和后序遍历同样可以使用非递归的算法实现。

> 非递归算法的具体实现可以查看下一节的内容。