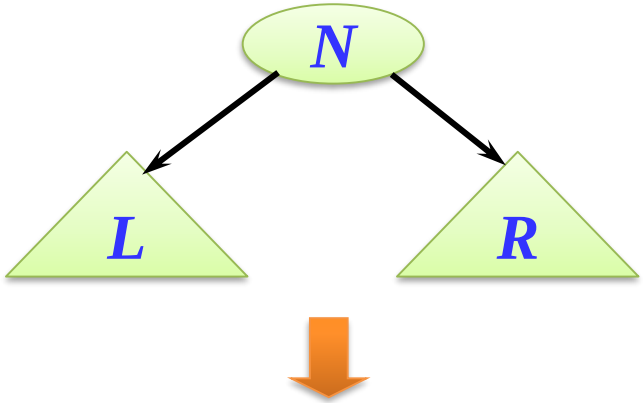
7.5 二叉树的遍历

7.5.1 二叉树遍历的概念

二叉树的遍历是指按照一定次序访问树中所有节点，并且每个节点仅被访问一次的过程。

遍历是二叉树最基本的运算，是二叉树中其他运算的基础。

二叉树的组成：



3种遍历方式

- NLR
- LNR
- LRN
- ~~NRL~~
- ~~RNL~~
- ~~RLN~~

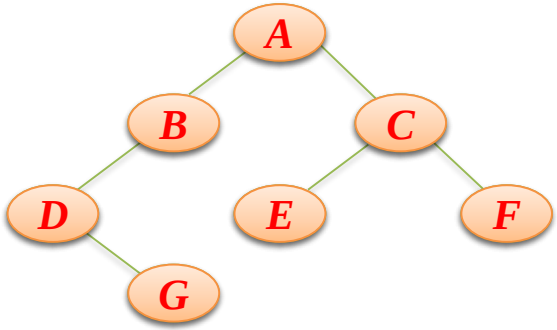
只考虑L → R的情况

1、先序遍历过程

先序遍历*NLR*二叉树的过程是：

- 访问根节点；
- 先序遍历左子树；
- 先序遍历右子树。

二叉树先序遍历演示



先序遍历序列：

A *B* *D* *G* *C* *E* *F*

遍历完毕

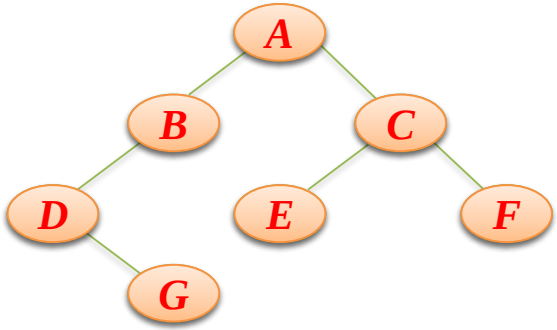
先序序列的第一个节点是根节点

2、中序遍历过程

中序遍历*LNR*二叉树的过程是：

- 中序遍历左子树；
- 访问根节点；
- 中序遍历右子树。

二叉树中序遍历演示



中序遍历序列：

D　　G　　B　　A　　E　　C　　F

遍历完毕

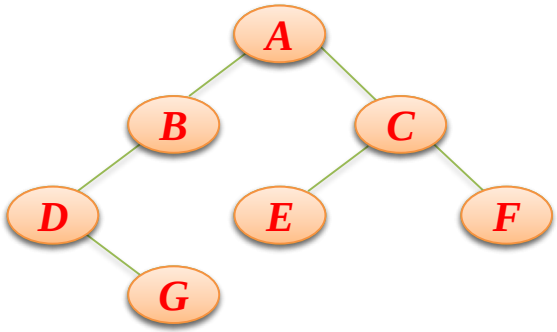
中序序列的根节点左边是左子树的节点，
右边是右子树的节点。

3、后序遍历过程

后序遍历*LRN*二叉树的过程是：

- 后序遍历左子树；
- 后序遍历右子树；
- 访问根节点。

二叉树后序遍历演示



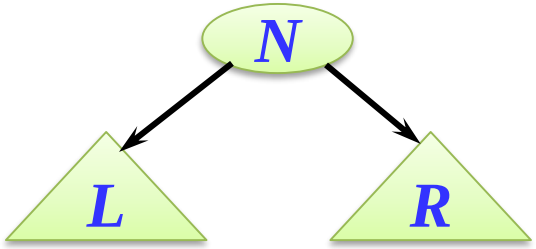
后序遍历序列：

G *D* *B* *E* *F* *C* *A*

遍历完毕

后序序列的最后一个节点是根节点

【例7-3】 以若一颗二叉树的先序序列和后序序列正好相反。该二叉树的形态是什么？



先序序列

$N\ L\ R$

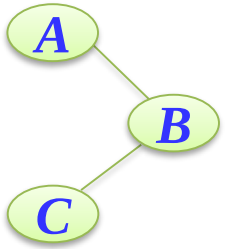
后序序列：$L\ R\ N$
后序序列的反序

$N\ R\ L$

L为空或者R为空时成立

这样的二叉树每层只有一个节点，即二叉树的形态是其高度等于节点个数。

例如



7.5.2 二叉树遍历递归算法

由二叉树的三种遍历过程直接得到3种递归算法。

先序遍历的递归算法：

```
void PreOrder(BTNode *b)
{   if (b!=NULL)
    {   printf("%c ",b->data);          //访问根节点
        PreOrder(b->lchild);
        PreOrder(b->rchild);
    }
}
```

上述访问是直接输出节点值。实际上，访问节点可以对该节点进行各种操作，如计数、删除节点等。

中序遍历的递归算法：

```
void InOrder(BTNode *b)
{       if (b!=NULL)
        {       InOrder(b->lchild);
                printf("%c ",b->data);  //访问根节点
                InOrder(b->rchild);
        }
}
```

后序遍历的递归算法：

```
void PostOrder(BTNode *b)
{    if (b!=NULL)
     {    PostOrder(b->lchild);
          PostOrder(b->rchild);
          printf("%c ",b->data);  //访问根节点
     }
}
```

思考题：

每种遍历序列提供了什么信息？

为什么前面3种遍历都采用递归求解？

7.5.3 层次遍历算法

层次遍历过程：

对于一颗二叉树，从根节点开始，按从上到下、从左到右的顺序访问每一个节点。

每一个节点仅仅访问一次。

算法设计思路：

使用一个队列。

I. 将根节点进队；

II. 队不空时循环：从队列中出列一个节点*p，访问它；

❶ 若它有左孩子节点，将左孩子节点进队；

❷ 若它有右孩子节点，将右孩子节点进队。

对应算法如下：

```
void LevelOrder(BTNode *b)
{    BTNode *p;
     BTNode *qu[MaxSize];          //定义环形队列,存放节点指针
     int front, rear;              //定义队头和队尾指针
     front=rear=0;                 //置队列为空队列
     rear++;
     qu[rear]=b;                   //根节点指针进入队列
```

```
    while (front!=rear)                 //队列不为空循环
    {   front=(front+1)%MaxSize;
        p=qu[front];                    //队头出队列
        printf("%c ",p->data);          //访问节点
        if (p->lchild!=NULL)            //有左孩子时将其进队
        {   rear=(rear+1)%MaxSize;
            qu[rear]=p->lchild;
        }
        if (p->rchild!=NULL)            //有右孩子时将其进队
        {   rear=(rear+1)%MaxSize;
            qu[rear]=p->rchild;
        }
    }
}
```

算法的时间复杂度为O(*n*)。

——本讲完——