第9周小结

1 二叉树遍历

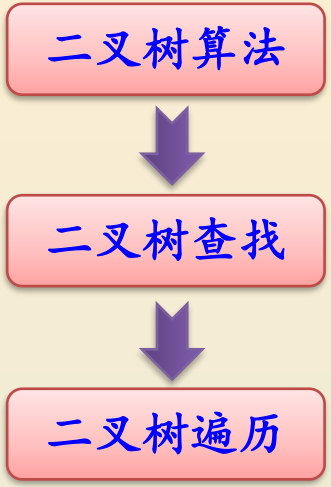
❶ 遍历过程

- 某种次序
- 访问所有节点
- 不重复访问

② 常用遍历方法

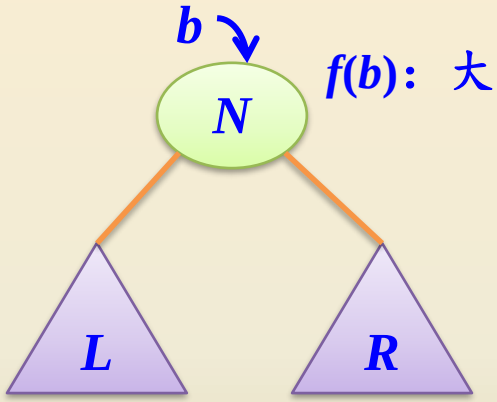
- 先序遍历
- 中序遍历
- 后序遍历
- 层次遍历

具有递归性



❸ 递归遍历算法应用

基于递归遍历 ⇔ 采用递归数据结构的递归算法设计方法



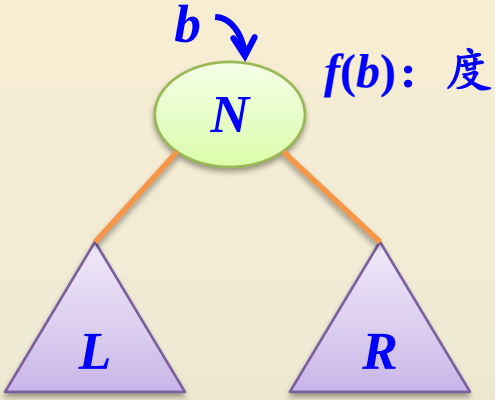
f(*b*)：大问题

f(*L*)：小问题　　*f*(*R*)：小问题

3部分组成，两种类型：节点，子树

- 先节点，再子树 ⇨ 先序
- 先子树，再节点 ⇨ 后序

假设二叉树采用二叉链存储结构，设计一个算法求二叉树b中度为2的节点个数。



f(b)：度为2的节点数

f(b->lchild)：L中度为2的节点数

f(b->rchild)：R中度为2的节点数

$f(b)=0$	当b=NULL
$f(b)=1+f(b->lchild)+f(b->rchild)$	当b节点度为2
$f(b)=f(b->lchild)+f(b->rchild)$	其他情况

算法如下：

```
int dnodes(BTNode *b)
{    if (b==NULL) return 0;
     if (b->lchild!=NULL && b->rchild!=NULL)
        return 1+dnodes(b->lchild)+dnodes(b->rchild);
     else
       return  dnodes(b->lchild)+dnodes(b->rchild);
}
```

假设二叉树采用二叉链存储结构，设计一个算法求二叉树*b*中第*k*层的节点个数。

- 设计算法为knumber(*b*，*h*，*k*，&*n*)，*h*表示*b*所指的节点层次，*n*是引用型参数，用于保存第*k*层的节点个数。
- 初始调用时，*b*为根节点指针，*h*为1，*n*赋值为0，即调用方式是：*n*=0；knumber(*b*，1，*k*，*n*)。

递归算法赋初值方式

算法如下：

```
void knumber(BTNode *b，int h，int k，int &n)
{    if (b==NULL)          //空树直接返回
         return;
     else                  //处理非空树
     {    if (h==k) n++;   //当前访问的节点在第k层时，n增1
          else if (h<k)    //若当前节点层次小于k，递归处理左、右子树
          {    knumber(b->lchild，h+1，k，n);
               knumber(b->rchild，h+1，k，n);
          }
     }
}
```

基于先序遍历的思路

假设二叉树采用二叉链存储结构，设计一个算法求二叉树*b*的宽度（采用递归方法）。

levelnumber(BTNode **b*，int *h*，int *a*[])：求二叉树*b*中所有层的节点个数，存放在*a*数组中，*a*[*h*]表示第*h*层节点个数

f(*b*，*h*，*a*) ⇔ 不做任何事情　　当*b*=NULL

f(*b*，*h*，*a*) ⇔ *a*[*h*]++;　　其他情况

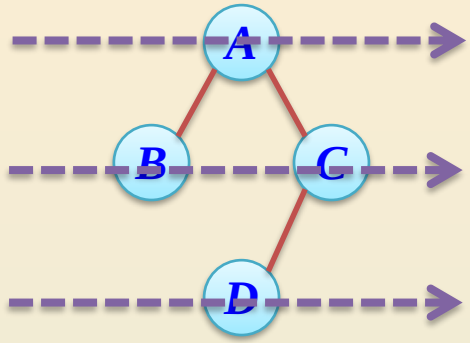
f(*b*->*lchild*，*h*+1，*a*)

f(*b*->*rchild*，*h*+1，*a*)

```
void levelnumber(BTNode *b，int h，int a[])
{
    if (b==NULL)
        return;
    else
    {
        a[h]++;
        levelnumber(b->lchild，h+1，a);
        levelnumber(b->lchild，h+1，a);
    }
}
```

```
int BTWidth1(BTNode *b)
{    int width=0，i;  int a[MaxSize];
     for (i=1;i<MaxSize;i++)
          a[i]=0;                  //a设置所有元素初始化为0
     levelnumber(b，1，a);
     i=1;
     while (a[i]!=0)               //求a中最大元素即宽度
     {     if (a[i]>width)
                width=a[i];
           i++;
     }
     return width;
}
```

❹ 层次遍历算法应用



- 每个节点有唯一的双亲节点
- 节点的层次 = 双亲节点的层次+1

假设二叉树采用二叉链存储结构，设计一个算法求二叉树*b*的宽度（采用层次遍历方法）。

```
int BTWidth2(BTNode *b)
{    struct
     {    int lno;              //节点的层次
          BTNode *p;            //节点指针
     } Qu[MaxSize];             //定义非环形队列
     int front，rear;           //定义队头和队尾指针

     int lnum，width，i，n;
     front=rear=0;              //置队列为空队
```

```
if (b!=NULL)
{   rear++;
    Qu[rear].p=b;                  //根节点进队
    Qu[rear].lno=1;                //根节点的层次为1
    while (rear!=front)            //队不空时循环
    {   front++;
        b=Qu[front].p;             //出队节点p
        lnum=Qu[front].lno;
        if (b->lchild!=NULL)       //有左孩子，将其进队
        {       rear++;
                Qu[rear].p=b->lchild;
                Qu[rear].lno=lnum+1;
        }
        if (b->rchild!=NULL)       //有右孩子，将其进队
        {       rear++;
                Qu[rear].p=b->rchild;
                Qu[rear].lno=lnum+1;
        }
}
```

```
        width=0; lnum=1; i=1;        //width存放宽度
        while (i<=rear)
        {     n=0;
              while (i<=rear && Qu[i].lno==lnum)
              {     n++;                    //n累计一层中的节点个数
                    i++;                    //i扫描队列中所有节点
              }
              lnum=Qu[i].lno;
              if (n>width) width=n;
        }
        return width;
    }
    else return 0;
}
```

二叉树的构造

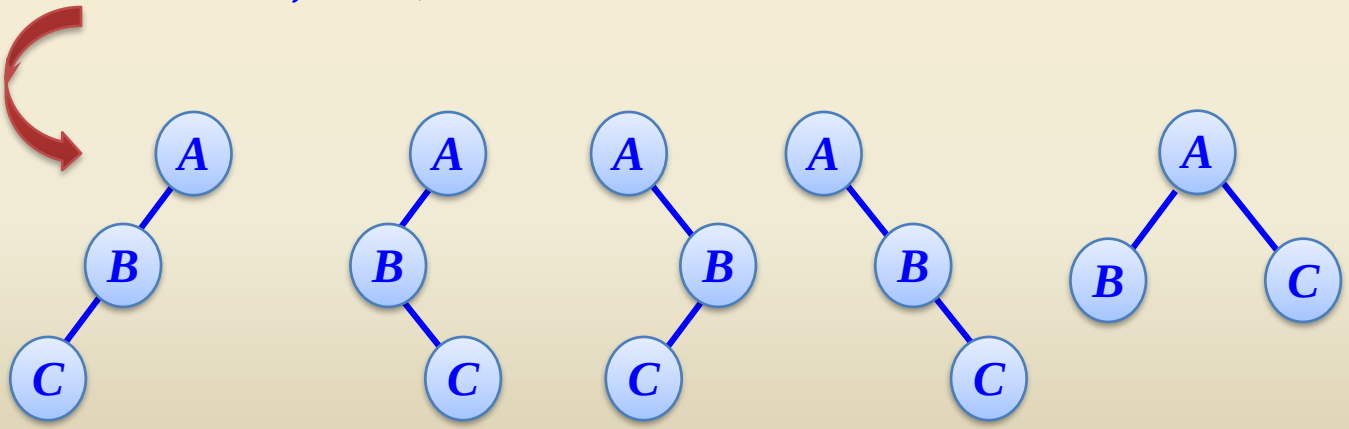
- ☑ 由中序序列和先序序列可以唯一构造一棵二叉树
- ☑ 由中序序列和后序序列可以唯一构造一棵二叉树
- ☑ 由中序序列和层次序列可以唯一构造一棵二叉树

由一个固定的先序序列（含n个不同的节点），构造的二叉树个数？

$$\frac{1}{n+1}C_{2n}^{n}=\frac{1}{n+1}\times\frac{(2n)!}{n!\times n!}$$

第n个Catalan数

当n=3，结果为5。



若某非空二叉树的先序序列和中序序列正好相反，则该二叉树的形态是什么？

先序序列　　中序序列的反序

N L R ═══ *R N L*

↓

*R*为空

↓

所有节点没有右子树的单支树

3 线索二叉树

二叉链 + 空指针改为线索 ➡ 线索二叉树

↓

左、右空指针指向前驱、后继节点

↓

前驱、后继节点与遍历方式有关 ➡

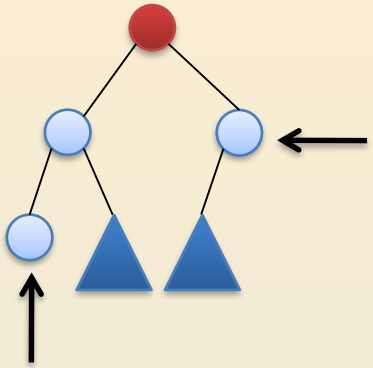
- 先序线索二叉树
- 中序线索二叉树
- 后序线索二叉树

建立线索二叉树的目的？

以中序线索二叉树说明：

- 对二叉树中序遍历，递归算法：时间复杂度均为O(n)，空间复杂度均为O(h)
- 对二叉树中序遍历，非递归算法：时间复杂度均为O(n)，空间复杂度均为O(h)
- 对中序线索二叉树中序遍历，时间复杂度均为O(n)，空间复杂度均为O(1)

二叉树中序序列的开始节点和尾节点？



根节点的最右下节点

根节点的最左下节点

4 哈夫曼树

n_0个叶子节点，含有权值

- 构造哈夫曼树：权值越小距离根节点越远
- 构造哈夫曼编码：权值越小编码越长

哈夫曼树中：

- 哈夫曼树满足二叉树的性质
- $n_1=0$
- 没有两个字符的编码相同
- 没有两个字符编码的前缀相同

如果一棵哈夫曼树T中共有255个节点，那么该树用于对几个字符进行哈夫曼编码？

- n=255，n_1=0
- n_0=n_2+1，总节点数n = n_0+n_2 = 2n_0-1 ⇨ n_0 = (n+1)/2 = 128
- 所以该树用于对128个字符进行哈夫曼编码。