



第6章 树和二叉树



6.1 树的定义和基本术语

6.2 二叉树

6.3 遍历二叉树和线索二叉树

6.4 树和森林

6.6 哈夫曼树与哈夫曼编码



6.1 树的基本概念

树型结构是一类重要的非线性数据结构。
其中以树和二叉树最为常用。

树是以分支关系定义的层次结构。

树的应用：各种社会组织机构；语法树；
软件结构图等。

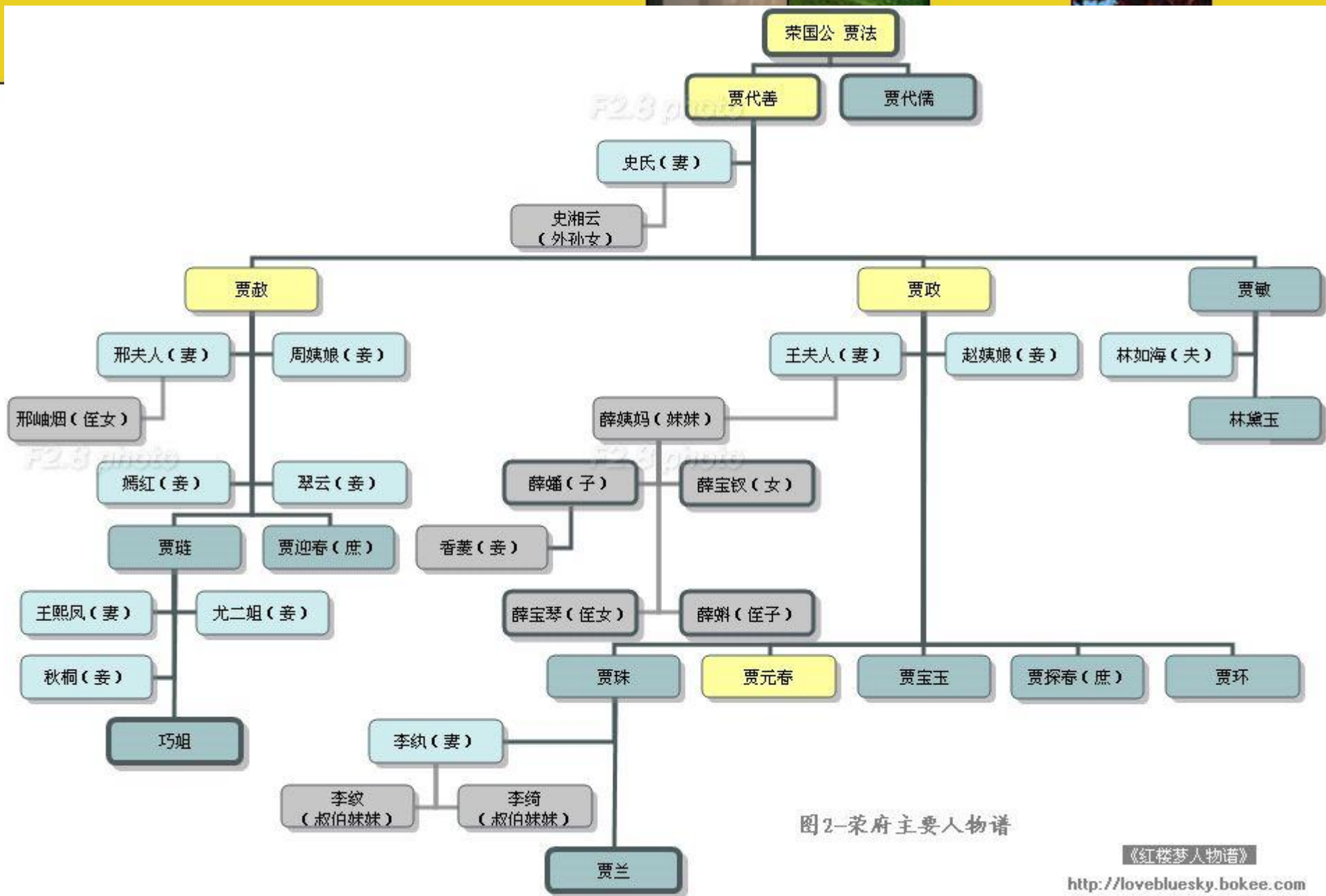


图2-荣府主要人物谱

《红楼梦人物谱》

<http://lovebluesky.bokee.com>



树的定义和基本术语:

树是 n ($n \geq 0$)个结点的有限集。

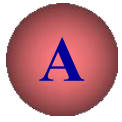
在任意一棵非空树中:

1、有且只有一个根结点;

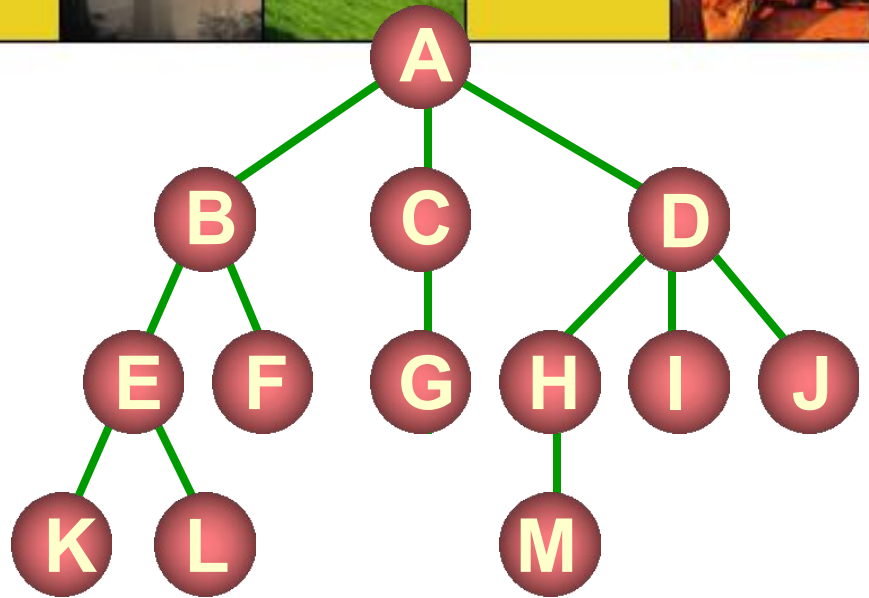
2、当 $n > 1$ 时, 其余结点可分为 m ($m > 0$) 个互不相交的有限

集 T_1, T_2, \dots, T_m , 其中每一个集合本身构成一棵子树。

例如



只有根结点的树



有13个结点的树

其中：A是根；其余结点分成三个互不相交的子集，
 $T1=\{B,E,F,K,L\}$ ； $T2=\{C,G\}$ ； $T3=\{D,H,I,J,M\}$ ，
 $T1,T2,T3$ 都是根A的子树，且本身也是一棵树



树的抽象数据类型定义:

ADT Tree{

数据对象D: D是具有相同特性的数据元素的集合。

数据关系R: 若D为空集, 则称为空树;

若D中仅含一个数据元素, 则R为空集, 否则 $R=\{H\}$, H是如下二元关系:

(1)在D中存在唯一的称为根的数据元素root, 它在关系H下无前驱;

(2)若 $D-\{\text{root}\}$ 不为空, 则存在 $D-\{\text{root}\}$ 的一个划分 D_1, D_2, \dots, D_m , 对任意j不等于k, 有 $D_j \cap D_k = \Phi$, 且对任意的i

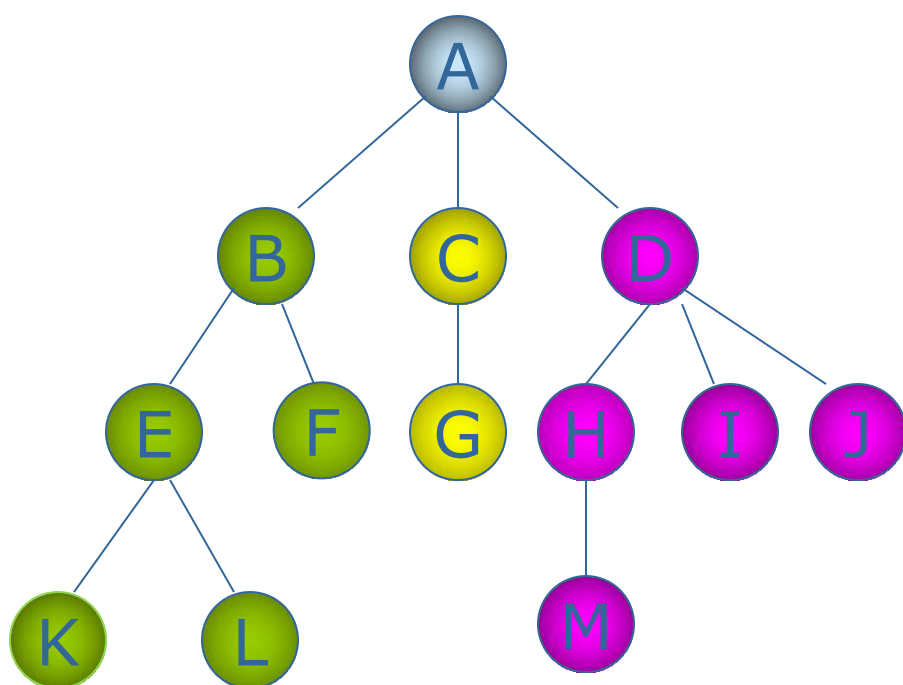
($1 \leq i \leq m$), 唯一存在数据元素 $x_i \in D_i$, 有 $\langle \text{root}, x_i \rangle \in H$;



(3) 对应于 $D - \{\text{root}\}$ 的划分,
 $H - \{\langle \text{root}, x_1 \rangle, \langle \text{root}, x_2 \rangle, \dots, \langle \text{root}, x_n \rangle\}$
有唯一的一个划分 H_1, H_2, \dots, H_m ($m > 0$),
对任意 $j \neq k$, ($1 \leq j, k \leq m$), 有 $H_j \cap H_k = \Phi$,
且对任意 i ($1 \leq i \leq m$), H_i 是 D_i 上的二元关系,

(D_i, H_i) 是一棵符合本定义的树, 称为根 root 的子树。

- 树结构种的一些基本术语:
- 树的结点: 包含一个数据元素及若干指向其子树的分支。
- 结点拥有的子树数称为结点的度 (Degree) 。



A的度为: 3

C的度为: 1

F的度为: 0

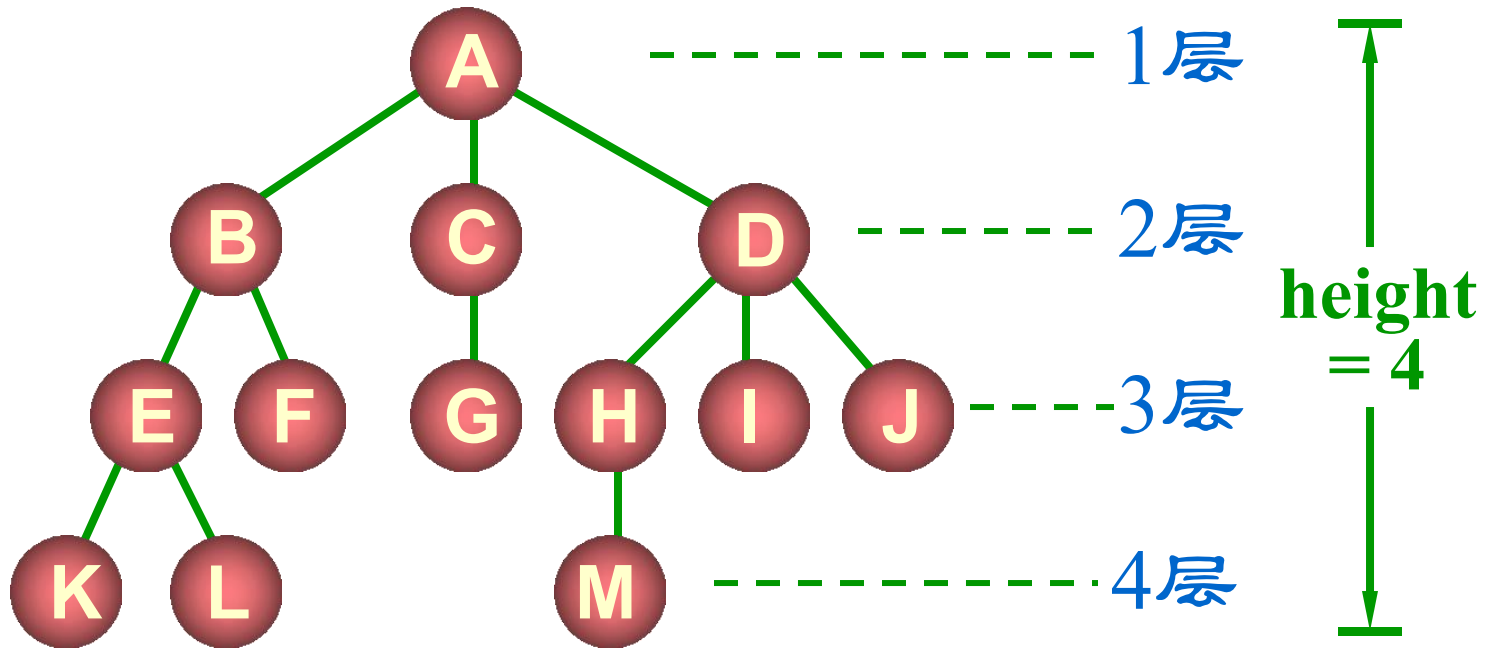


- 度为0的结点称为叶子结点/终端结点。
- 度不为0的结点称为非终端结点或分支结点。
- 除根结点外，分支结点也称为内部结点。
- 树的度是树内各结点度的最大值。
- 结点的子树的根称为该结点的孩子；相应的，该结点称为孩子的双亲。



- 同一个双亲的孩子之间互称**兄弟**。
- 结点的**祖先**是根到该结点所经分支的所有结点。
- 结点的**层次**从根开始定义起，根为第一层，根的孩子为第二层。
- 双亲在同一层的结点互为**堂兄弟**。**堂兄弟的双亲是兄弟关系吗？**
- 树中结点的最大层次称为树的**深度或高度**。

树的基本术语



* 结点

* 结点的度

* 叶结点

* 分支结点

* 子女

* 双亲

* 兄弟

* 祖先

* 子孙

* 结点层次

* 树的深度

* 树的度


* 森林

- 基本操作：
- `InitTree(&T);` //构造空树
- `DestroyTree(&T);` //销毁树
- `CreateTree(&T,definition);` //按definition构造树T
- `ClearTree(&T);` //将树T清为空树

- `TreeEmpty(T)`; //若T为空树，则返回TRUE，否则FALSE
- `TreeDepth(T)`; //返回T的深度
- `Root(T)`; //返回T的根
- `Value(T,cur_e)`; //返回树中结点cur_e的值
- `Assign(T,cur_e,value)`; //给结点cur_e赋值为value
- `Parent(T,cur_e)`; //若cur_e是T的非根结点，则返回它的双亲。

- **LeftChild(T,cur_e);** //若cur_e是T的非叶子结点，则返回它的最左孩子，否则返回“空”。
- **RightSibling(T,cur_e);** // 若cur_e有右兄弟，则返回它的右兄弟，否则函数值为“空”。
- **InsertChild(&T,&p,i,c);**
- 初始条件：树T存在，p指向T中某个结点， $1 \leq i \leq p$ 所指结点的度+1，非空树c与T不相交。
- 操作结果：插入c为T中p指结点的第i棵树。

- **DeleteChild(&T,&p,i);**
- 初始条件：树T存在，p指向T中某个结点， $1 \leq i \leq p$ 所指结点的度
- 操作结果：删除T中p所指结点的第i棵子树。
- **TraverseTree (T, visit ())**
- 操作结果：按某种顺序对T的每个结点调用函数visit () 一次且至多一次。一旦visit () 失败，则操作失败。



线性结构	树结构
存在唯一的没有前驱的"首元素"	存在唯一的没有前驱的"根结点"
存在唯一的没有后继的"尾元素"	存在多个没有后继的"叶子"
其余元素均存在唯一的"前驱元素"和唯一的"后继元素"	其余结点均存在唯一的"前驱(双亲)结点"和多个"后继(孩子)结点"

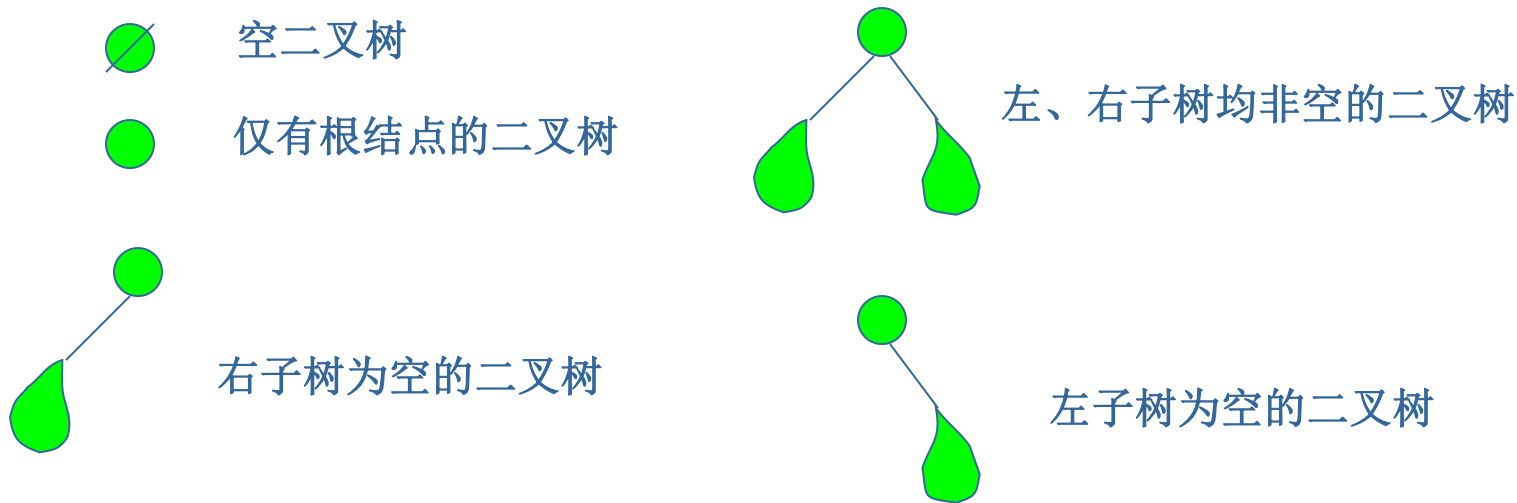
由于线性结构是一个"序列", 元素之间存在的是"一对一"的关系, 而树是一个层次结构, 元素之间存在的是"一对多"的关系。



- 题一：在一棵树中，A节点有3个兄弟节点，B节点是A节点的双亲节点，则B节点的度是多少_____。
- 题二：若一棵度为4的树中度为1、2、3、4的节点的个数分别为4、3、2、2，则该树叶子节点的个数是多少？总节点个数是多少？

6.2 二叉树

- 二叉树
- 二叉树 (Binary Tree) 是另一种树型结构，它的特点是每个结点至多只有两棵子树，并且，二叉树的子树有左右之分，其次序不能任意颠倒。
- 二叉树的五种基本形态。



性质

性质1 在二叉树的第 i 层上至多有 2^{i-1} 个结点。 ($i \geq 1$) [证明用归纳法]

证明：当 $i=1$ 时，只有根结点， $2^{i-1}=2^0=1$ 。

假设对所有 j , $i > j \geq 1$, 命题成立，即第 j 层上至多有 2^{j-1} 个结点。

由归纳假设第 $i-1$ 层上至多有 2^{i-2} 个结点。

由于二叉树的每个结点的度至多为 2，故在第 i 层上的最大结点数为第 $i-1$ 层上的最大结点数的 2 倍，即 $2 * 2^{i-2} = 2^{i-1}$ 。

性质2 深度为 k 的二叉树至多有 2^{k-1} 个结点 ($k \geq 1$)。

证明: 由性质1可见, 深度为 k 的二叉树的
最大结点数为

$$\sum_{i=1}^k (\text{第 } i \text{ 层上的最大结点数})$$
$$= \sum_{i=1}^k 2^{i-1} = 2^0 + 2^1 + \dots + 2^{k-1} = 2^{k-1}$$



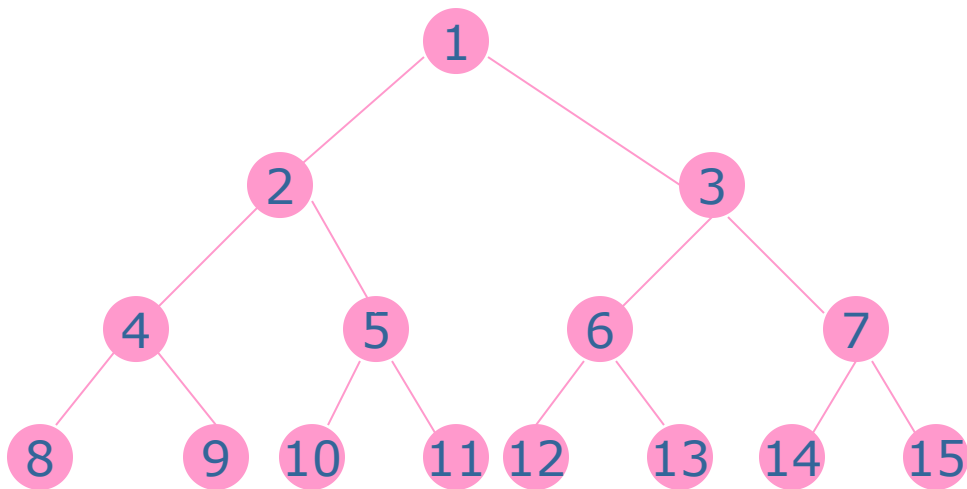
• **性质3** 对任何一棵二叉树T，如果其终端结点数为 n_0 ，度为2的结点数为 n_2 ，则 $n_0=n_2+1$

证明：设 n_1 为二叉树T中度为1的结点数。因为二叉树中所有结点的度均小于或等于2，所以其结点总数为 $n=n_0+n_1+n_2$

再看二叉树中的分支数，除了根结点外，其余结点都有一个分支进入，设B为分支总数，则 $n=B+1$ 。由于这些分支是由度为活的结点引出的，所有又有 $B=n_1+2n_2$ ，于是

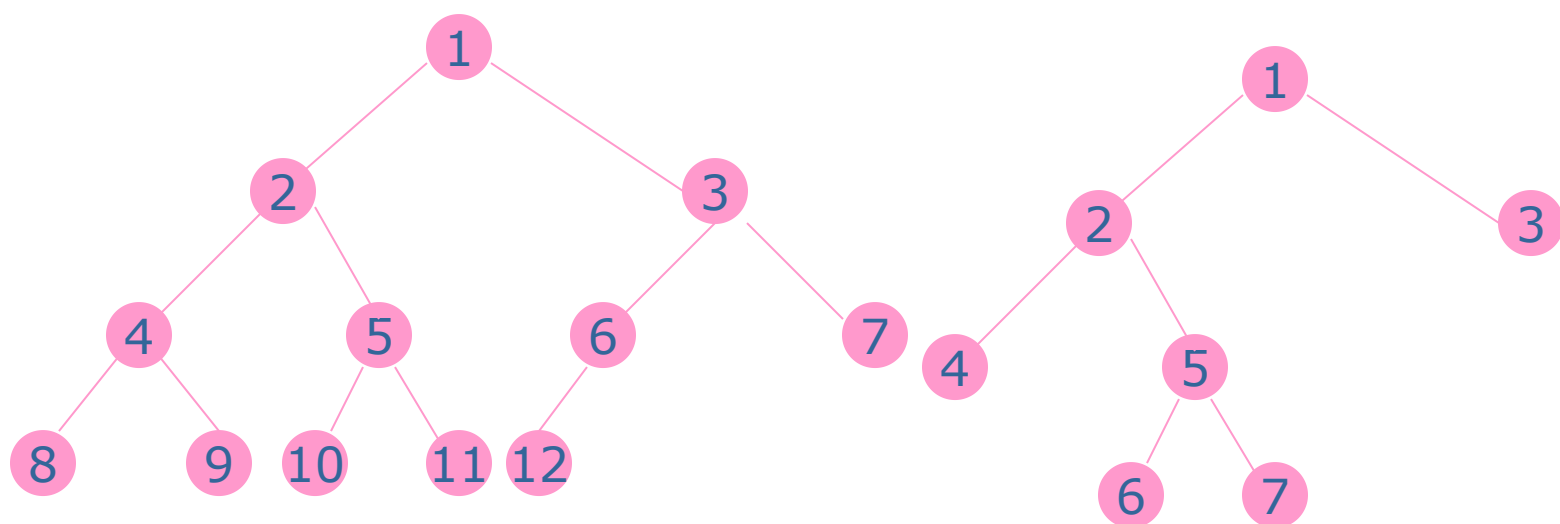
$n=n_1+2n_2+1$ ，所以 $n_0=n_2+1$

- 满二叉树和完全二叉树的定义
- 一棵深度为 k 且有 2^k-1 个结点的二叉树称为满二叉树。



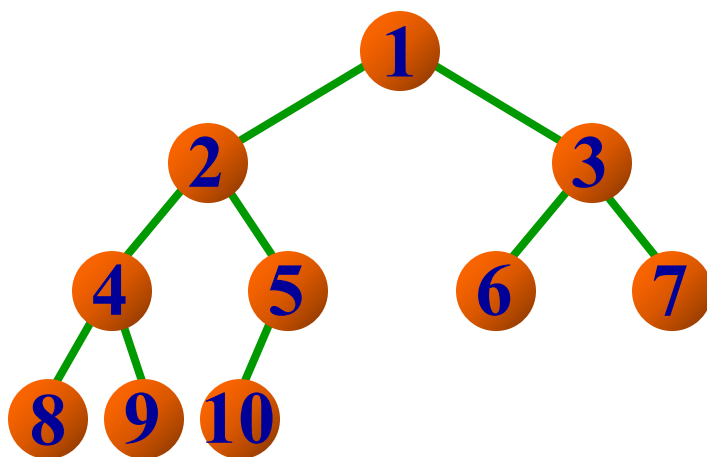


- 对满二叉树的结点进行连续编号，约定编号从根结点起，自上而下，自左而右，由此引出完全二叉树的定义。
- 深度为k的，有n个结点的二叉树，当且仅当其每一个结点都与深度为k的满二叉树中编号从1到n的结点一一对应时，称之为完全二叉树。



- 完全二叉树的两个重要特性：
- **性质4** 具有n个结点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor + 1$
- 证明：假设深度为k，则有：
- $2^{k-1} - 1 < n \leq 2^k - 1$
- 于是 $k-1 \leq \log_2 n < k$
- 因为k是整数，所以
- $k = \lfloor \log_2 n \rfloor + 1$

- **性质5** 如果对一棵有 n 个结点的完全二叉树的结点按层序编号（从第1层到 $\lfloor \log_2 n \rfloor + 1$ ，则对任一结点 i （ $1 \leq i \leq n$ ），有
- (1)如果 $i=1$ ，则结点 i 是二叉树的根，无双亲；如果 $i > 1$ ，则其双亲 $\text{PARENT}(i)$ 是结点 $\lfloor i/2 \rfloor$ 。
- (2)如果 $2i > n$ ，则结点 i 无左孩子**有没有可能有右孩子？**（结点 i 叶子结点）；否则其左孩子 $\text{LCHILD}(i)$ 是结点 $2i$ 。
- (3)如果 $2i+1 > n$ ，则结点 i 无右孩子；否则其右孩子 $\text{RCHILD}(i)$ 是结点 $2i+1$ 。





• 题三：若一棵二叉树具有10个度为2的节点，5个度为1的节点，则度为0的节点个数为_____。

A. 9 B. 11 C. 15 D. 不确定

• 题四：一棵二叉树中有7个叶子节点和5个单分支节点，其总共有_____节点。

A. 16 B. 18 C. 12 D. 31

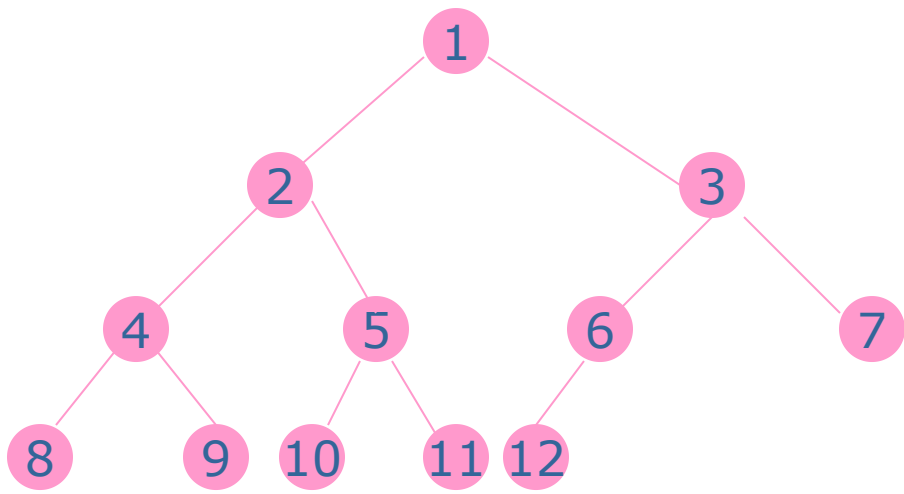


• 题六：一棵完全二叉树中有1001个节点，其中叶子节点的个数是_____。

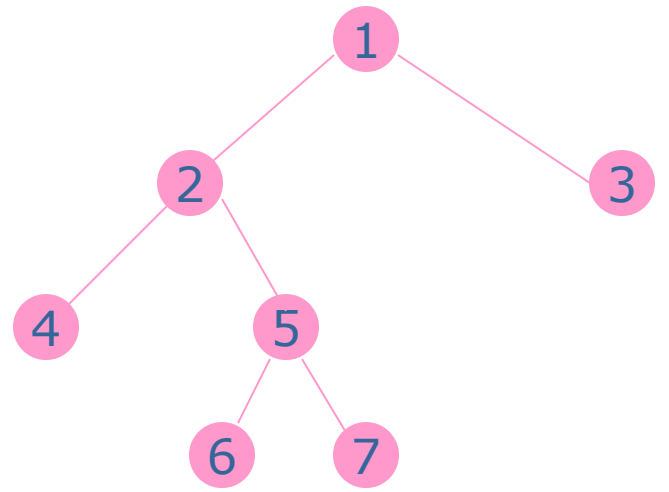
A. 250 B. 500 C. 505 D. 501



- 二叉树的存储结构
- 一、顺序存储结构
- `#define MAX_TREE_SIZE 100`
- `typedef TElemType SqBiTree[MAX_TREE_SIZE];`
- `SqBiTree bt;`
- 按照顺序存储结构的定义，在此约定，用一组地址连续的存储单元依次自上而下，自左至右存储完全二叉树上的结点元素，即将完全二叉树上标号为 i 的结点元素存储在如上定义的一维数组中下标为 $i-1$ 的分量中。



1 2 3 4 5 6 7 8 9 10 11

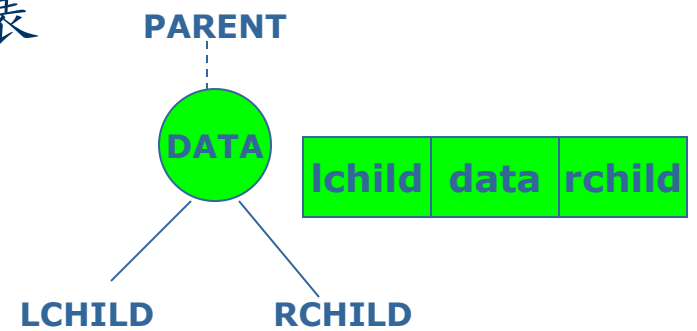


1 2 3 4 5 0 0 0 0 6 7

- 二、链式存储结构

设计不同的结点结构可构成不同形式的链式存储结构。

由二叉树的定义得知，二叉树的结点由一个数据元素和分别指向其左、右子树的两个分支构成，则表示二叉树的链表中的结点至少包含三个域：数据域、左指针域、右指针域。

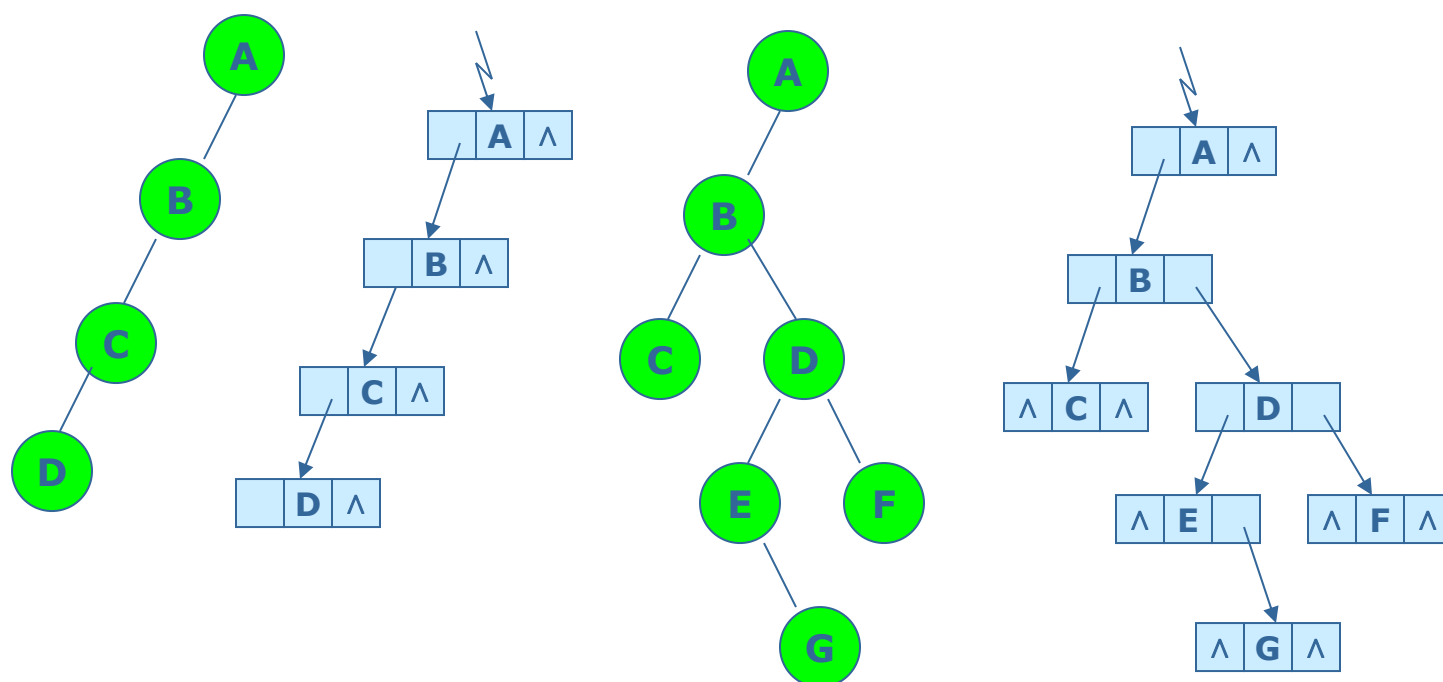


- 有时，为了便于找到结点的双亲，则可在结点结构中增加一个指向其双亲结点的指针域。

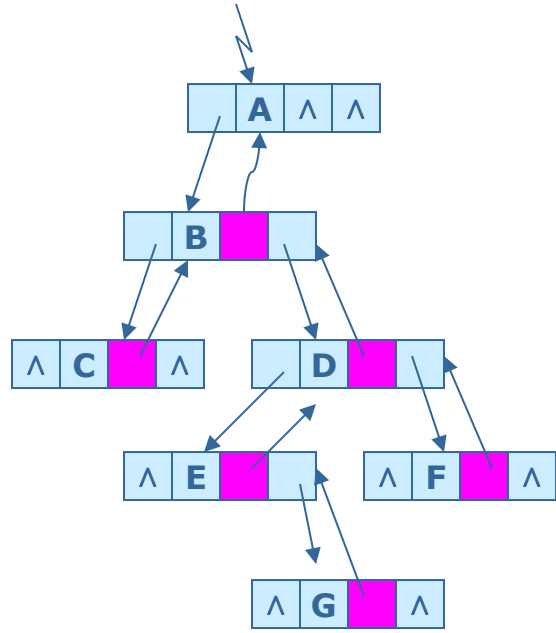
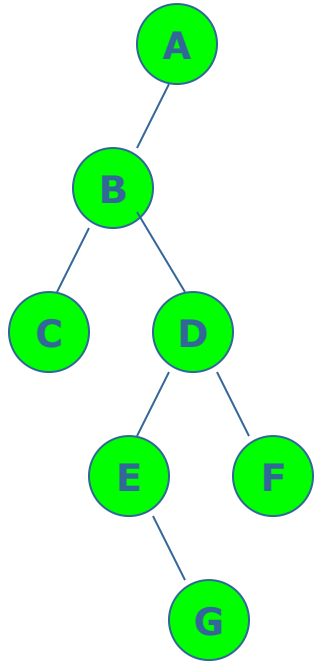


- 利用这两种结点结构所得二叉树的存储结构分别称为二叉链表和三叉链表。

- 链表存储结构示意图



在含有n个结点的二叉链表中有n+1个空链域。



- 二叉树的二叉链表存储表示

```
typedef struct BiTNode{  
    TElemType data;  
    struct BiTNode *lchild,*rchild;  
}BiTNode,*BiTree;
```

- 基本操作的函数原型说明:
- **Status CreateBiTree(BiTree &T);**
- 按先序次序输入二叉树中结点的值（一个字符），空格字符表示空树，构造二叉链表表示的二叉树T。
- **Status PreOrderTraverse(BiTree T,status(*visit)(TElemType e))**
- 采用二叉链表存储结构，visit是对结点操作的应用函数。
- 先序遍历二叉树T，对每个结点调用函数visit一次且仅一次。一旦visit失败，则操作失败。

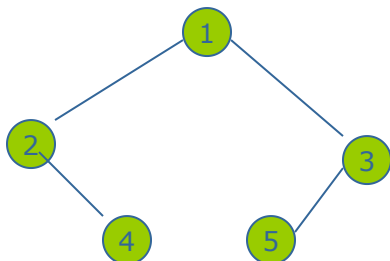


- **Status InOrderTraverse(BiTree T,status(*visit)(TElemType e))**
- 采用二叉链表存储结构，visit是对结点操作的应用函数。
- 中序遍历二叉树T，对每个结点调用函数visit一次且仅一次。一旦visit失败，则操作失败。

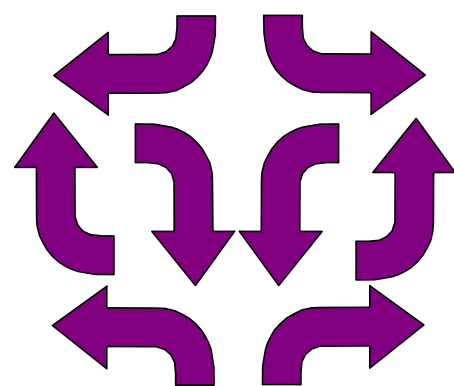
- **Status PostOrderTraverse(BiTree T,status(*visit)(TElemType e))**
- 采用二叉链表存储结构，visit是对结点操作的应用函数。
- 后序遍历二叉树T，对每个结点调用函数visit一次且仅一次。一旦visit失败，则操作失败。

- **Status LevelOrderTraverse(BiTree T,status(*visit)(TElemType e))**
- 采用二叉链表存储结构，visit是对结点操作的应用函数。
- 层序遍历二叉树T，对每个结点调用函数visit一次且仅一次。一旦visit失败，则操作失败。

- 题七：具有n个节点的二叉树采用二叉链存储结构，共有_____个空指针域。
- 题八：已知二叉树如右图所示，此二叉树的顺序存储的结点序列是（ ）
A.123□45 B.12345 C.12□435 D.□ 24153



6.3 遍历二叉树 和线索二叉树





一、问题的提出

二、先左后右的遍历算法

三、算法的递归描述

四、中序遍历算法的非递归描述

五、遍历算法的应用举例


一、问题的提出



遍历:


顺着某一条搜索路径**巡访**二叉树中的结点, 使得每个结点**均被访问一次**, 而且**仅被访问一次**。

“**访问**”的含义可以很广, 如: 输出结点的信息等。



“**遍历**” 是任何类型均有的操作，对线性结构而言，只有一条搜索路径(因为每个结点均只有一个后继)，故不需要另加讨论。

而**二叉树**是非线性结构，每个结点有两个后继，则存在如何**遍历**即按什么样的**搜索路径**遍历的问题。



对“二叉树”而言，可以有
三条搜索路径：

- 1. **先上后下**的按层次遍历；
- 2. **先左**(子树) **后右**(子树)的遍历；
- 3. **先右**(子树) **后左**(子树)的遍历。

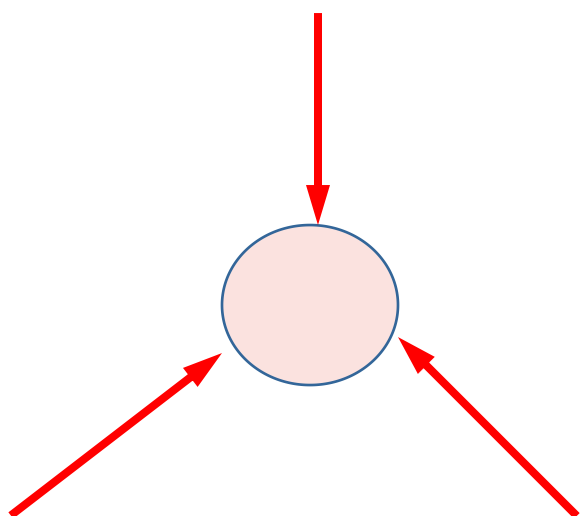


二、先左后右的遍历算法

先(根) 序的遍历算法

中(根) 序的遍历算法

后(根) 序的遍历算法





先(根) 序的遍历算法:

若二叉树为空树, 则空操作; 否则,

- (1) 访问根结点;
- (2) 先序遍历左子树;
- (3) 先序遍历右子树。

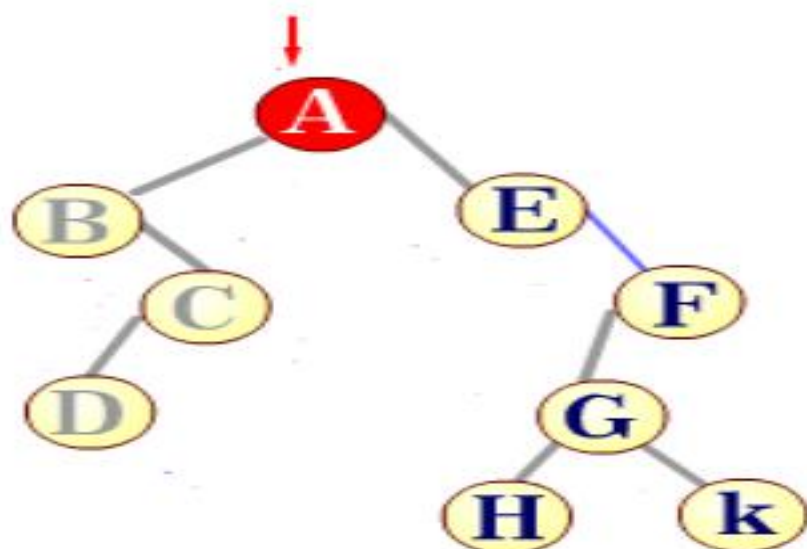
先(根)序遍历

若二叉树为空，则空操作，否则

(1)访问根结点

(2)先序遍历左子树

(3)先序遍历右子树





● 中(根)序的遍历算法:

若二叉树为空树, 则空操作; 否则,

- (1) 中序遍历左子树;
- (2) 访问根结点;
- (3) 中序遍历右子树。

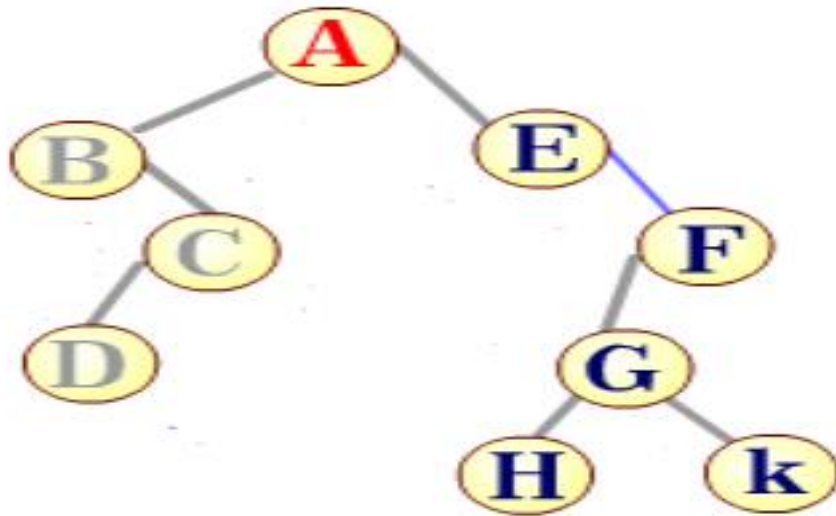
中序遍历

若二叉树为空，则空操作，否则

(1)中序遍历左子树

(2)访问根结点

(3)中序遍历右子树





● 后(根) 序的遍历算法:

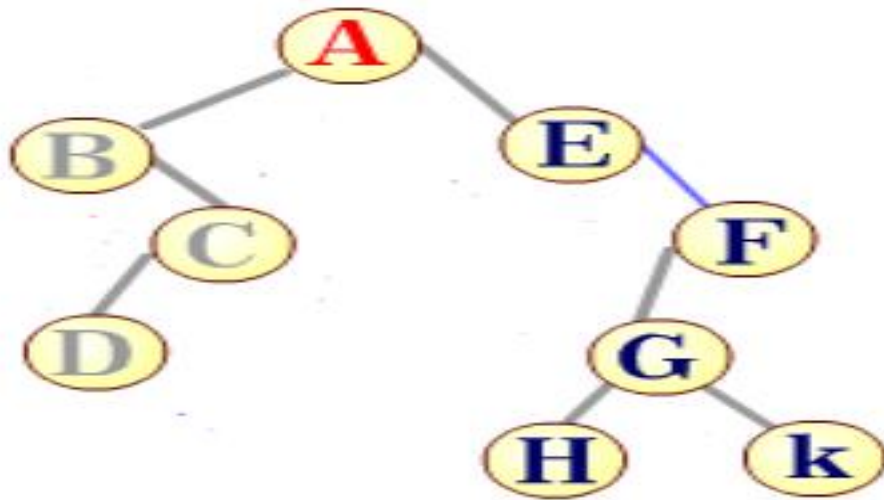
若二叉树为空树, 则空操作; 否则,

- (1) 后序遍历左子树;
- (2) 后序遍历右子树;
- (3) 访问根结点。

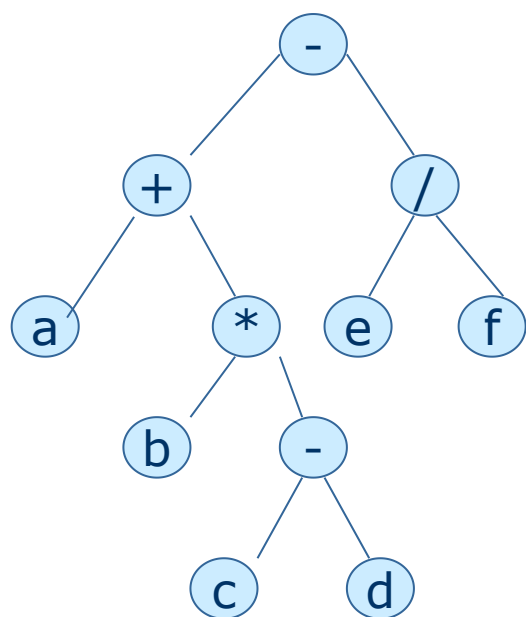
后序遍历

若二叉树为空，则空操作，否则

- (1)后序遍历左子树
- (2)后序遍历右子树
- (3)访问根结点



练习



■ 先序遍历此二叉树：
 $- + a * b - c d / e f$

■ 中序遍历此二叉树：
 $a + b * c - d - e / f$

■ 后序遍历此二叉树：
 $a b c d - * + e f / -$

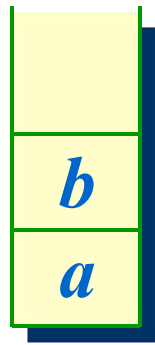
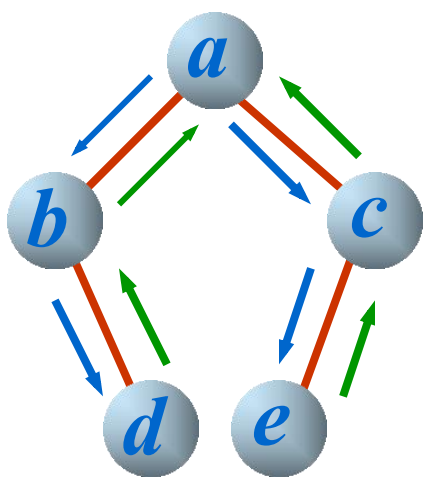


三、算法的递归描述

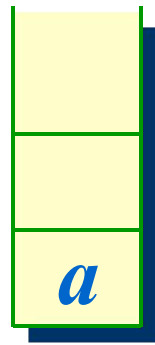
```
void Preorder (BiTree T,  
               void( *visit)(TElemType& e))  
{ // 先序遍历二叉树  
  if (T) {  
    visit(T->data); // 访问结点  
    Preorder(T->lchild, visit); // 遍历左子树  
    Preorder(T->rchild, visit); // 遍历右子树  
  }  
}
```



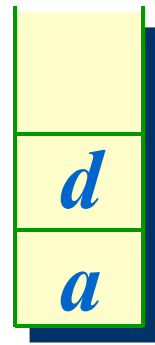
四、中序遍历算法的非递归描述



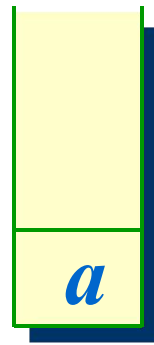
a b入栈



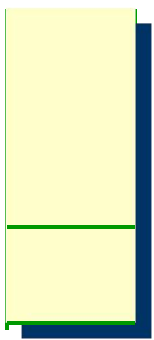
b退栈
访问



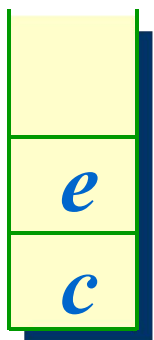
d入栈



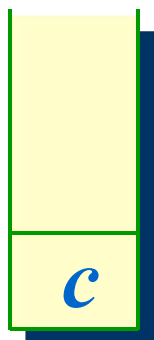
d退栈
访问



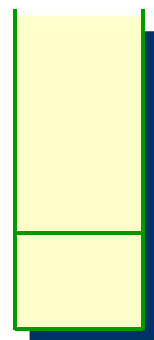
a退栈
访问



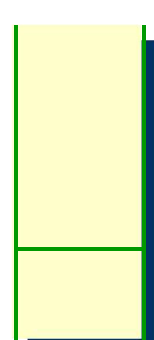
c e入栈



e退栈
访问



c退栈
访问

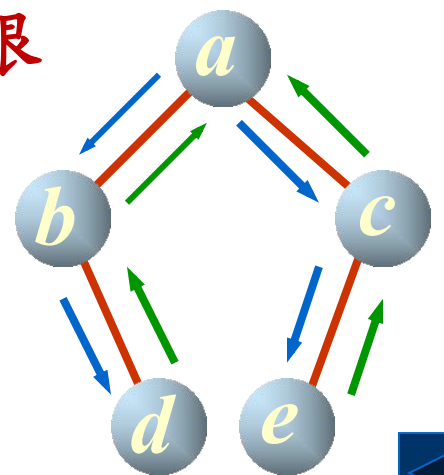



栈空

```

void InOrder ( BinTree T ) {
    stack S;  InitStack( &S ); //递归工作栈
    BinTreeNode *p = T; //初始化
    while ( p != NULL || !StackEmpty(&S) ) {
        if( p != NULL )
            { Push(&S, p); p = p->leftChild; }
        else if ( !StackEmpty(&S) ) { //栈非空
            Pop(&S, p); //退栈
            cout << p->data << endl; //访问根
            p = p->rightChild;
        } //if
    } //while
    return ok;
}

```





五、遍历算法的应用举例

- 1、**建立二叉树的存储结构**
- 2、**统计二叉树中叶子结点的个数
(先序遍历)**
- 3、**求二叉树的深度(后序遍历)**

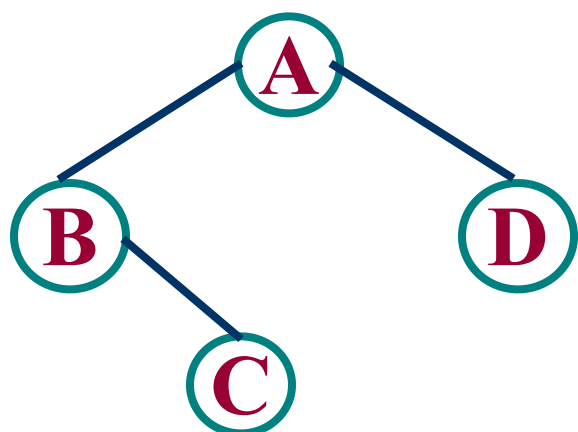
1、以字符串的形式定义一棵二叉树

空树

以空白字符“#”表示



以字符串“ A##”表示



以字符串“ AB#C##D##”表示



```
Status CreateBiTree(BiTree &T) {
```

```
scanf(&ch);
```

```
if (ch=='#') T = NULL;
```

```
else{ T = new BinTree;
```

```
    T->data = ch;           // 生成根结点
```

```
    CreateBiTree(T->lchild); // 构造左子树
```

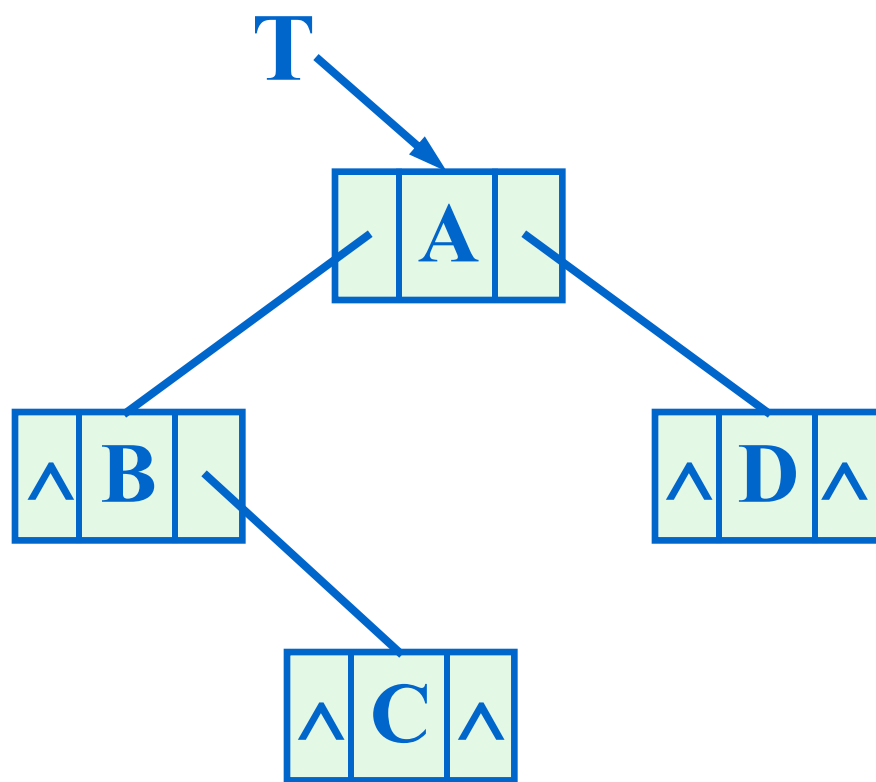
```
    CreateBiTree(T->rchild); // 构造右子树
```

```
}
```

```
return OK; } // CreateBiTree
```

上页算法执行过程举例如下：

$AB\#C\#\#D\#\#$



●由二叉树的先序和中序序列建树

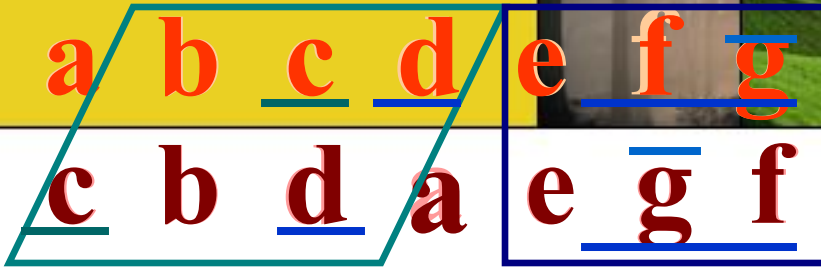
仅知二叉树的先序序列“**abcdefg**”
不能唯一确定一棵二叉树，

如果同时已知二叉树的中序序列
“**cbdaegf**”，则会如何？

二叉树的先序序列 **根** **左子树** **右子树**

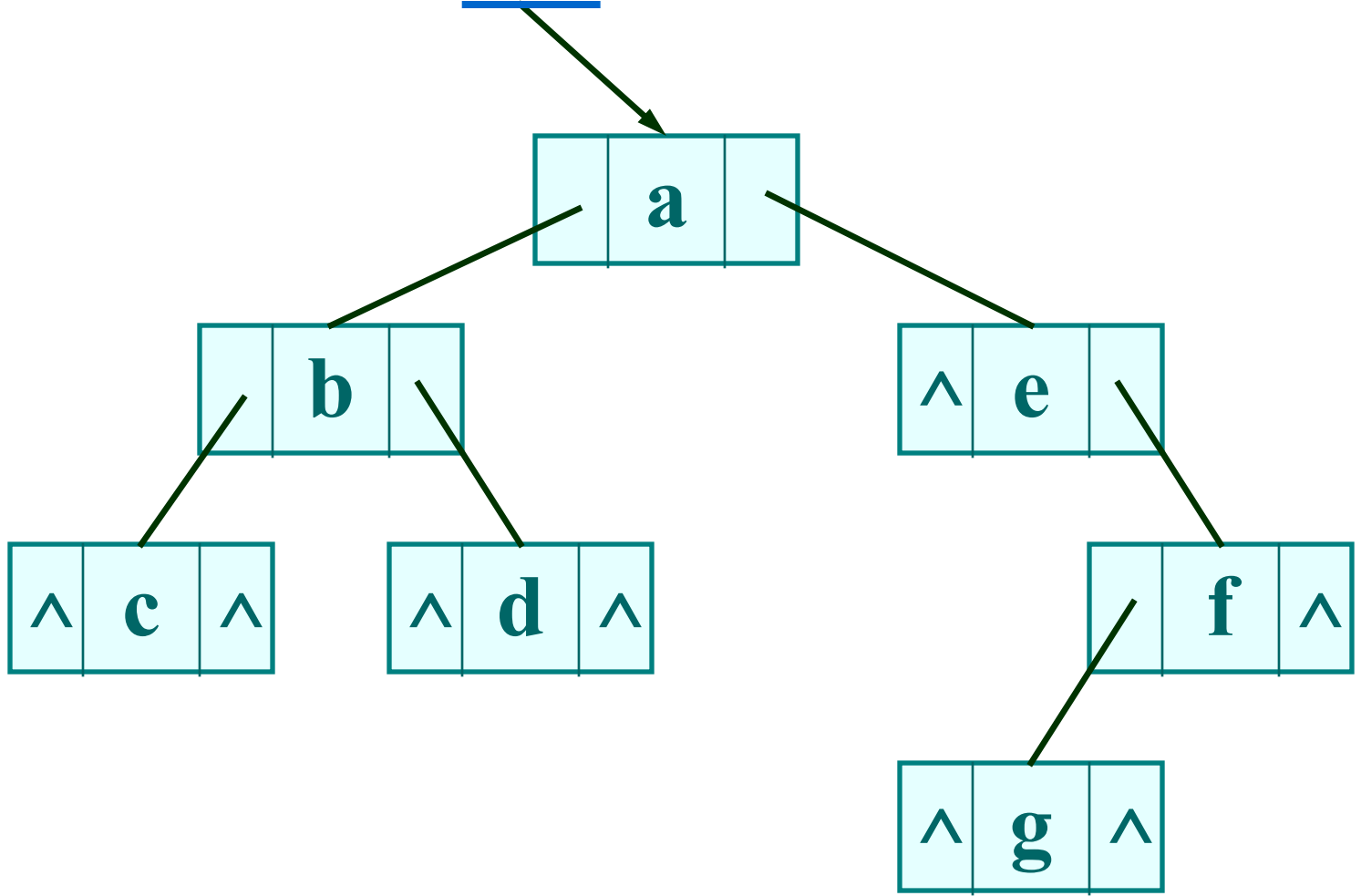
二叉树的中序序列 **左子树** **根** **右子树**

例如:



先序序列

中序序列



2、统计二叉树中叶子结点的个数

算法基本思想:

先序(或中序或后序)遍历二叉树, 在遍历过程中查找叶子结点, 并计数。

由此, 需在遍历算法中增添一个“计数”的参数, 并将算法中“访问结点”的操作改为: 若是叶子, 则计数器增1。

```
void CountLeaf (BiTree T, int& count){
    if ( T ) {
        if ((!T->lchild)&& (!T->rchild))
            count++; // 对叶子结点计数
        CountLeaf( T->lchild, count);
        CountLeaf( T->rchild, count);
    } // if
} // CountLeaf
```



3、求二叉树的深度(后序遍历)



算法基本思想:

首先分析二叉树的深度和它的左、右子树深度之间的关系。

从二叉树深度的定义可知，二叉树的深度应为其左、右子树深度的最大值加1。由此，需先分别求得左、右子树的深度，算法中“访问结点”的操作为：求得左、右子树深度的最大值，然后加 1 。

```
int Depth (BiTree T){ // 返回二叉树的深度
    if ( !T )    depthval = 0;
    else {
        depthLeft = Depth( T->lchild );
        depthRight= Depth( T->rchild );
        depthval = 1 + (depthLeft > depthRight ?
                        depthLeft : depthRight);
    }
    return depthval;
}
```





• 题九：设 n 、 m 为一棵二叉树上的两个节点，在中序序列中 n 在 m 前的条件是_____。

A. n 在 m 右边

B. n 是 m 祖先

C. m 是 n 祖先

D. n 在 m 左边

• 题十：设 n 、 m 为一棵二叉树上的两个节点，在先序序列中 n 在 m 之前，在后序序列中 n 在 m 之后，则 n 和 m 的关系是_____。

A. n 是 m 的兄弟

B. n 是 m 的有兄弟

C. n 是 m 的祖先

D. n 是 m 子孙




- 题十一：一棵二叉树的先序、中序和后序序列分别如下，其中有一部分未显示出来。试求出空格处的内容，并画出该二叉树。

先序序列：_B_F_ICEH_G

中序序列：D_KFIA_EJC_

后序序列：_K_FBHJ_G_A

线索二叉树

- 何谓线索二叉树? 
- 线索链表的遍历算法 
- 如何建立线索链表? 

一、何谓线索二叉树?



遍历二叉树的结果是,

求得结点的一个线性序列。

例如:

先序序列:

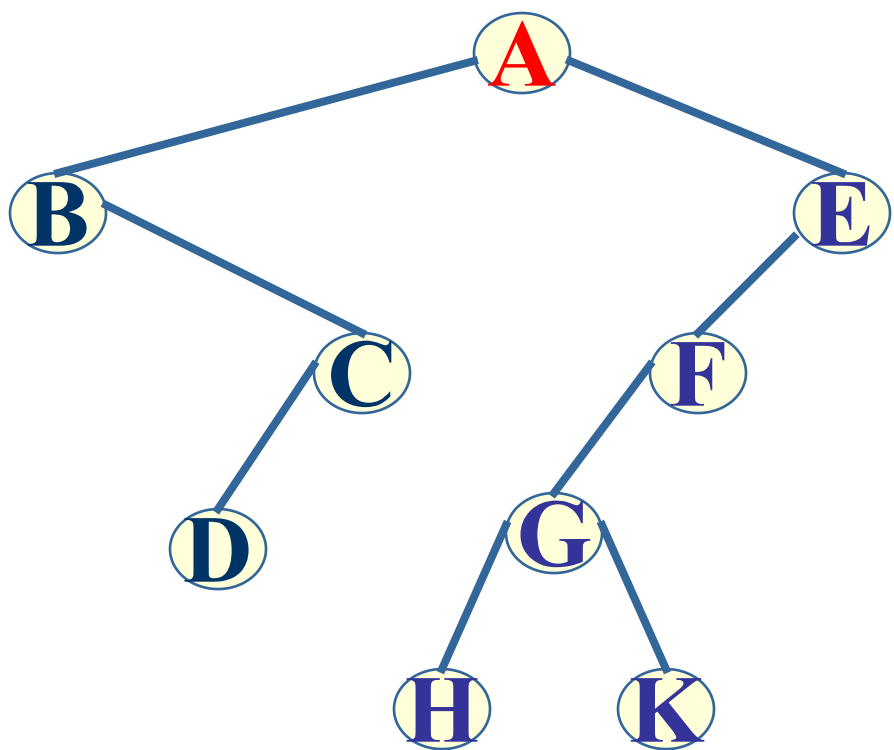
A B C D E F G H K

中序序列:

B D C A H G K F E

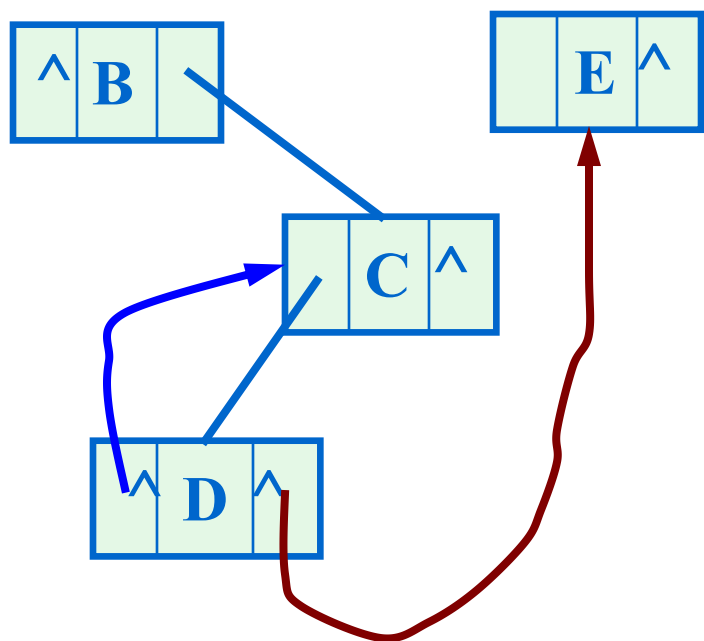
后序序列:

D C B H K G F E A



指向该线性序列中的“前驱”和“后继”的指针，称作“线索”

A B C D E F G H K



包含“线索”的存储结构，称作“线索链表”

与其相应的二叉树，称作“线索二叉树”

对线索链表中结点的约定:



在二叉链表的结点中增加两个标志域,并作如下规定:

- 若该结点的左子树不空,则Lchild域的指针指向其左子树,且左标志域的值为“指针 Link”;否则, Lchild域的指针指向其“前驱”且左标志的值为“**线索 Thread**”。



●若该结点的右子树不空，

则 rchild 域的指针指向其右子树，

且右标志域的值为“指针 Link”；

否则， rchild 域的指针指向其“后继”，

且右标志的值为“**线索 Thread**”。

如此定义的二叉树的存储结构称作

“**线索链表**”。

线索链表的类型描述:

```
typedef enum { Link, Thread } PointerThr;  
    // Link==0:指针, Thread==1:线索  
  
typedef struct BiThrNod {  
    TElemType      data;  
    struct BiThrNode *lchild, *rchild; // 左右指针  
    PointerThr     LTag, RTag;    // 左右标志  
} BiThrNode, *BiThrTree;
```



二、线索链表的遍历算法：

由于在线索链表中添加了遍历中得到的“前驱”和“后继”的信息，从而简化了遍历的算法。

```
for ( p = firstNode(T); p; p = Succ(p) )  
    Visit (p);
```

例如：



对中序线索化链表的遍历算法

※ 中序遍历的第一个结点 ？

左子树上处于“最左下”（没有左子树）的结点。

※ 在中序线索化链表中结点的后继 ？

若无右子树，则为后继线索所指结点；
否则为对其右子树进行中序遍历时访问的第一个结点。



```

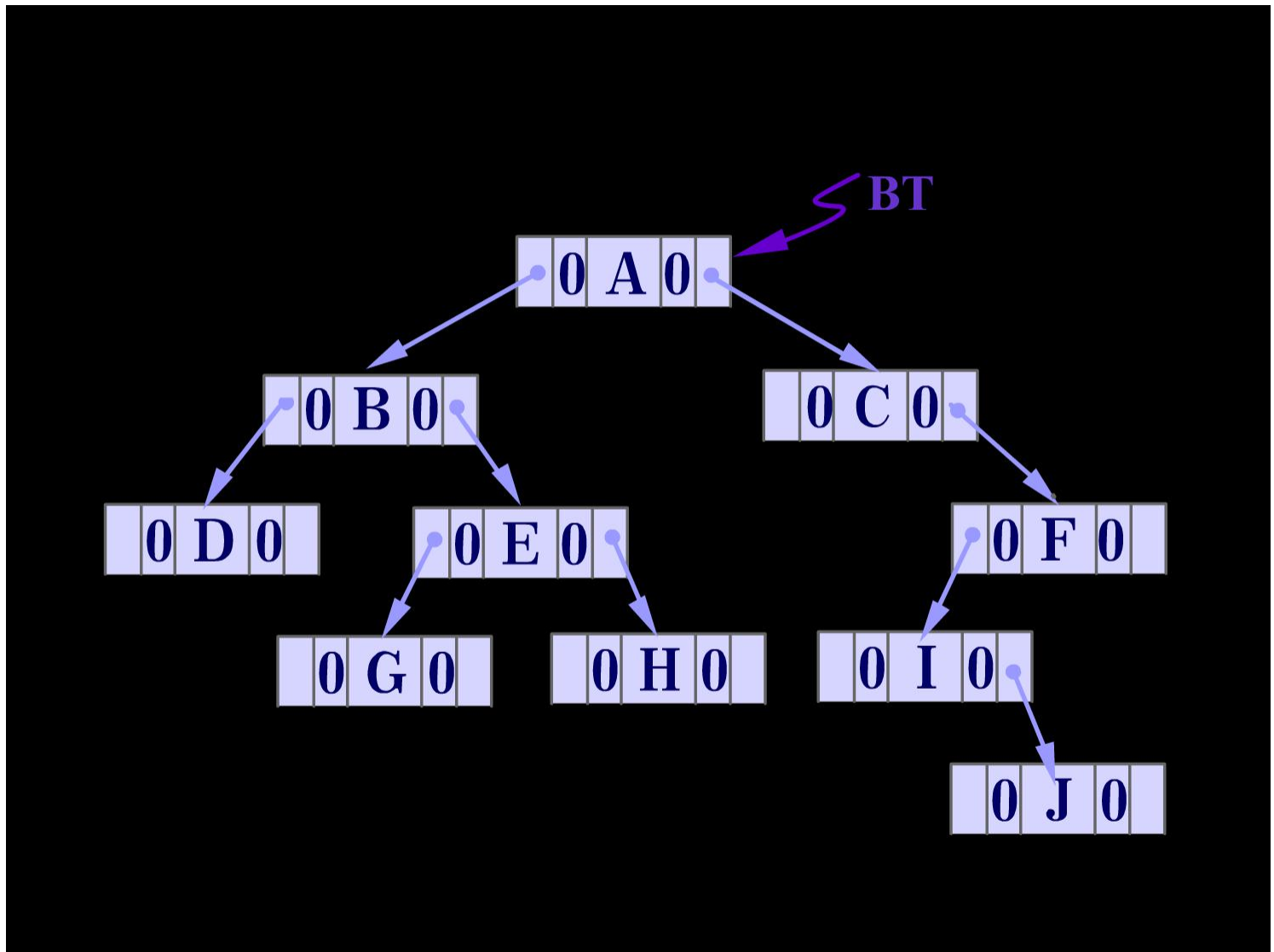
void InOrderTraverse_Thr(BiThrTree T,
                        void (*Visit)(TElemType e)) {
    p = T->lchild;    // p指向根结点
    while (p != T) { // 空树或遍历结束时, p==T
        while (p->LTag==Link) p = p->lchild; // 第一个结点
        Visit(p->data);
        while (p->RTag==Thread && p->rchild!=T) {
            p = p->rchild; Visit(p->data); // 访问后继结点
        }
        p = p->rchild; // p进至其右子树根
    }
}
// InOrderTraverse_Thr

```

三、如何建立线索链表？

在中序遍历过程中修改结点的左、右指针域，以保存当前访问结点的“前驱”和“后继”信息。遍历过程中，附设指针pre，并始终保持指针pre指向当前访问的、指针p所指结点的前驱。

线索链表的生成

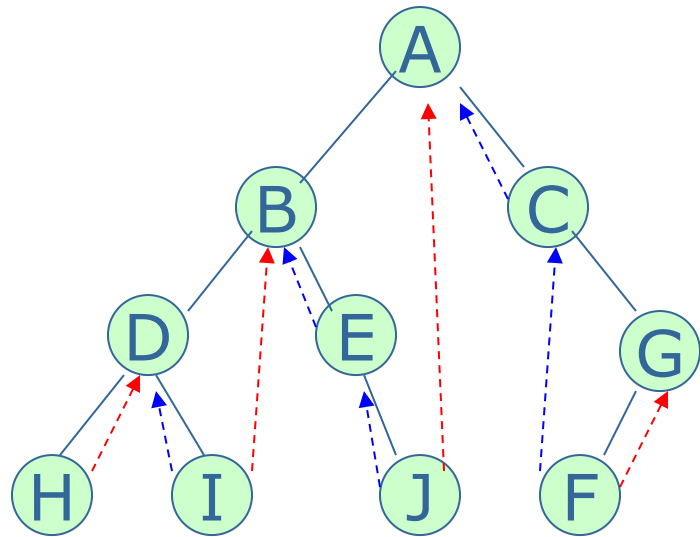
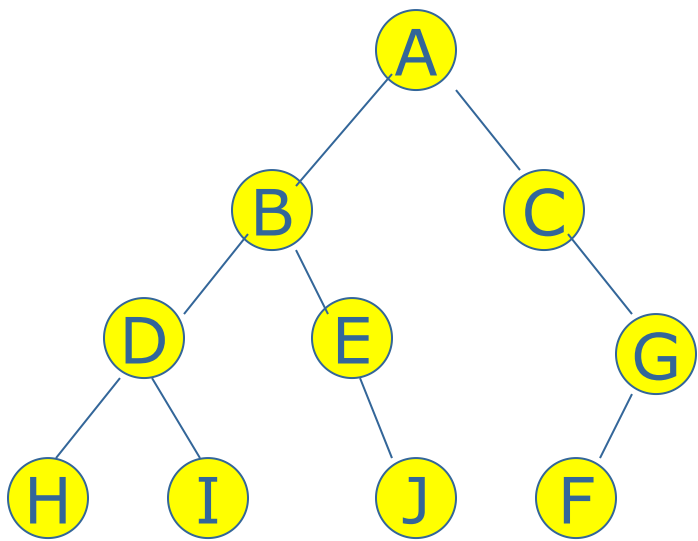


题十二：

假设一棵二叉树，其按后根顺序的结点排列为：
H, I, D, J, E, B, F, G, C, A，而按
中根顺序的结点排列为：H, D, I, B, E,
J, A, C, F, G。

要求画出这棵二叉树和中序线索二叉树。

- 分析：由后根序列的最后一个结点A可推出该二叉树的树根为A，由中序序列中A的位置可知A的左子树由HDIBEJ组成，右子树由CFG组成。又由HDIBEJ在后序序列中顺序可推出该子树的根结点为B，按B在中序序列中的位置知B的左子树由HDI组成，右子树由EJ组成。B的左子树HDI在后序序列中最后一个为D，则该子树的根为D，E的右子树为J，左子树为空，依次类推。



6.4 树和森林

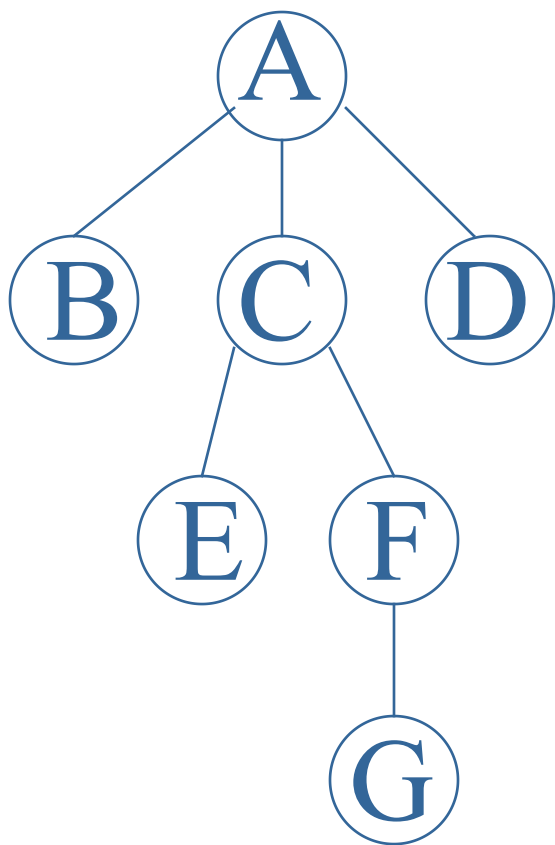




树的三种存储结构

- 一、双亲表示法
- 二、孩子链表表示法
- 三、树的二叉链表(孩子-兄弟)存储表示法

一、双亲表示法:



data parent

0	A	-1
1	B	0
2	C	0
3	D	0
4	E	2
5	F	2
6	G	5

r=0

n=6

C语言的类型描述:

```
#define MAX_TREE_SIZE 100
```

结点结构:

data	parent
-------------	---------------

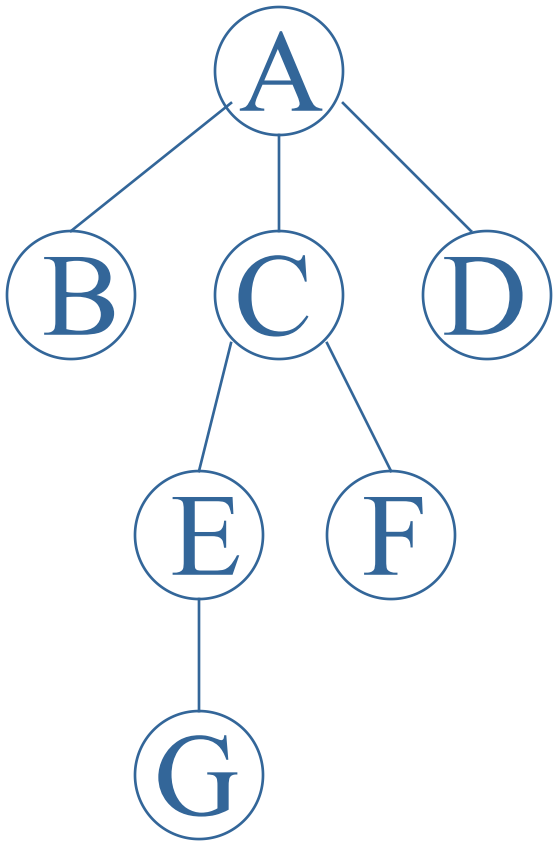
```
typedef struct PTNode {  
    Elem data;  
    int parent; // 双亲位置域  
} PTNode;
```

树结构:

```
typedef struct {  
    PTNode nodes [MAX_TREE_SIZE];  
    int r, n;  
    // 根结点的位置和结点个数  
} PTree;
```



带双亲的孩子链表表示法:



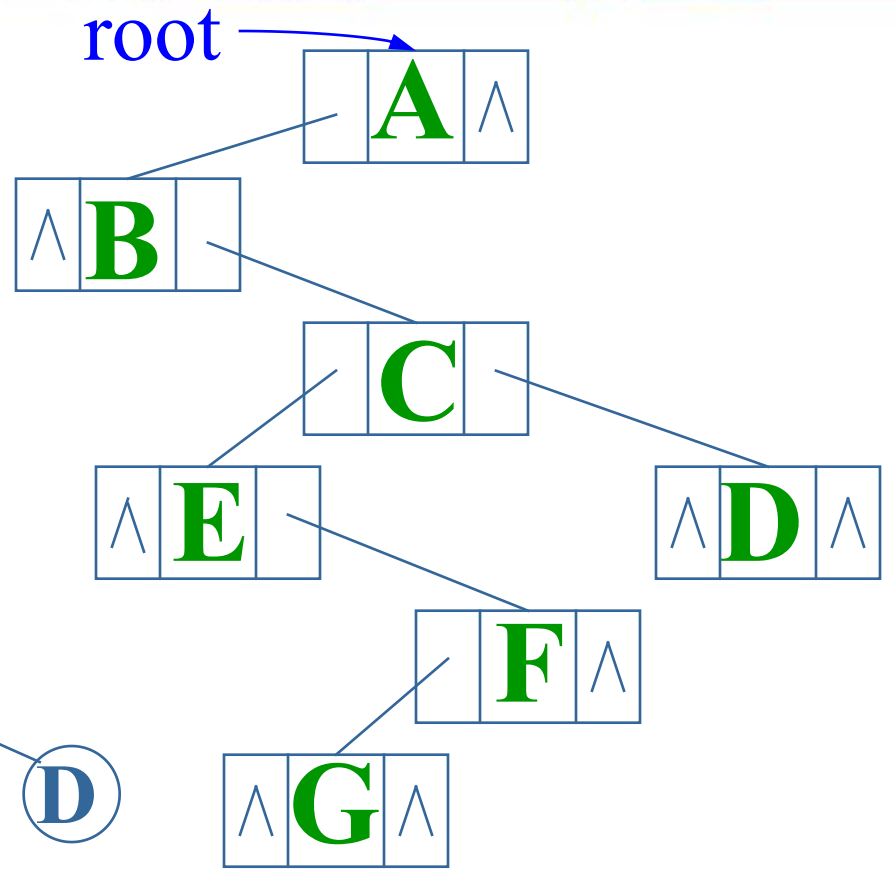
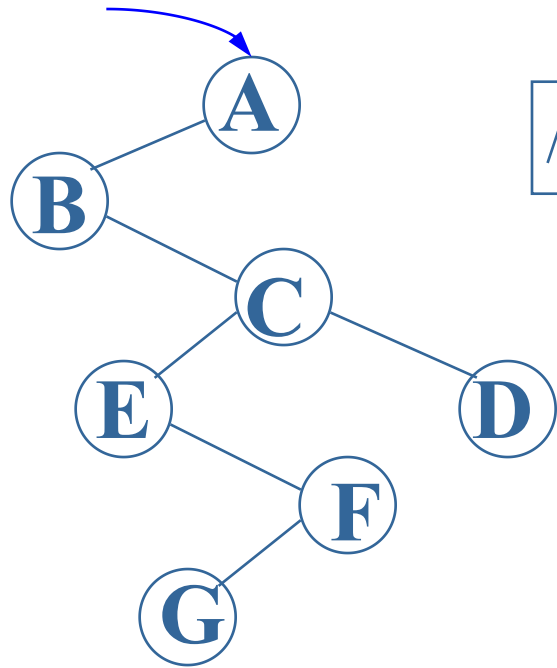
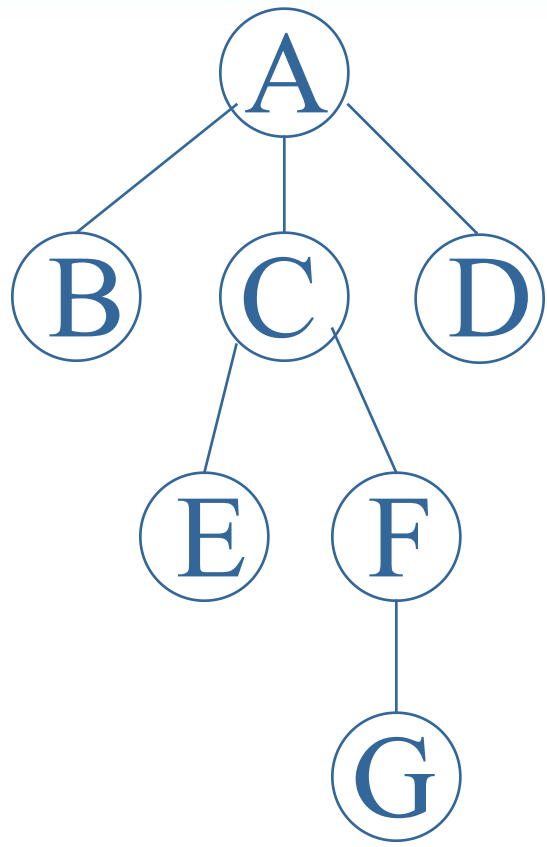
data firstchild

0	A	-1	→	1	→	2	→	3	Λ
1	B	0							Λ
2	C	0	→	4	→	5			Λ
3	D	0							Λ
4	E	2	→	6					Λ
5	F	2							Λ
6	G	4							Λ

r=0
n=6



三、树的二叉链表(孩子-兄弟)存储表示法



结点结构

data	firstChild	nextSibling
------	------------	-------------



森林和二叉树的对应关系

设森林

$$F = (T_1, T_2, \dots, T_n);$$

$$T_1 = (\text{root}, t_{11}, t_{12}, \dots, t_{1m});$$

二叉树

$$B = (\text{LBT}, \text{Node}(\text{root}), \text{RBT});$$



由森林转换成二叉树的转换规则为:


若 $F = \Phi$, 则 $B = \Phi$;

否则,

由 $\text{ROOT}(T_1)$ 对应得到 $\text{Node}(\text{root})$;

由 $(t_{11}, t_{12}, \dots, t_{1m})$ 对应得到 **LBT**;

由 (T_2, T_3, \dots, T_n) 对应得到 **RBT**。



由二叉树转换为森林的转换规则为：

若 $B = \Phi$ ， 则 $F = \Phi$;

否则，

由 $\text{Node}(\text{root})$ 对应得到 $\text{ROOT}(T_1)$;

由 LBT 对应得到 $(t_{11}, t_{12}, \dots, t_{1m})$;

由 RBT 对应得到 (T_2, T_3, \dots, T_n) 。

T1

T2

A

D

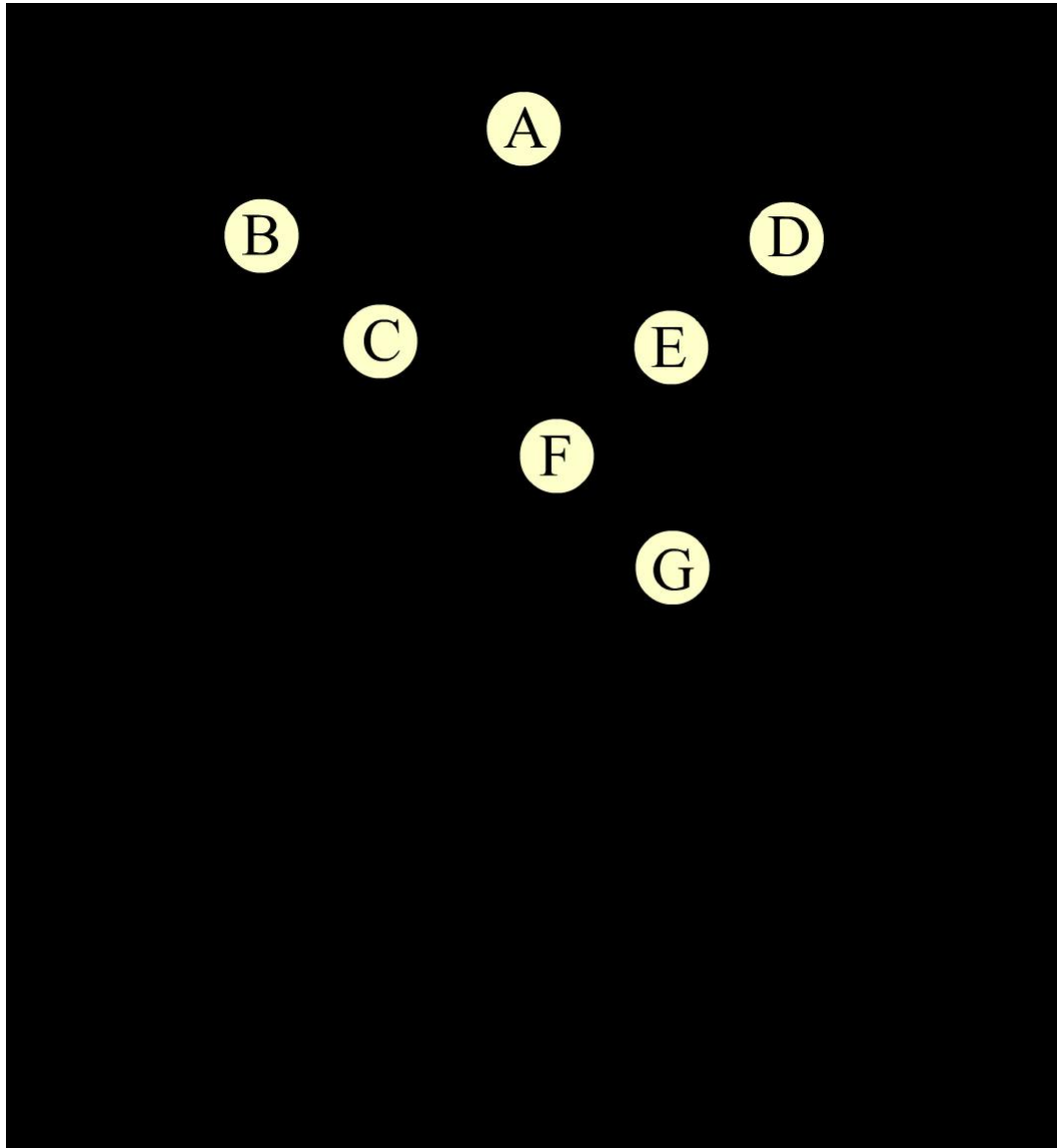
B

C

E

F

G





由此，树的各种操作均可对应二叉树的操作来完成。

应当注意的是，和树对应的二叉树，其左、右子树的概念已改变为：**左是孩子，右是兄弟。**

题十三：

已知一棵树的结点表示如下，其中各兄弟结点是依次出现的，画出对应的二叉树。

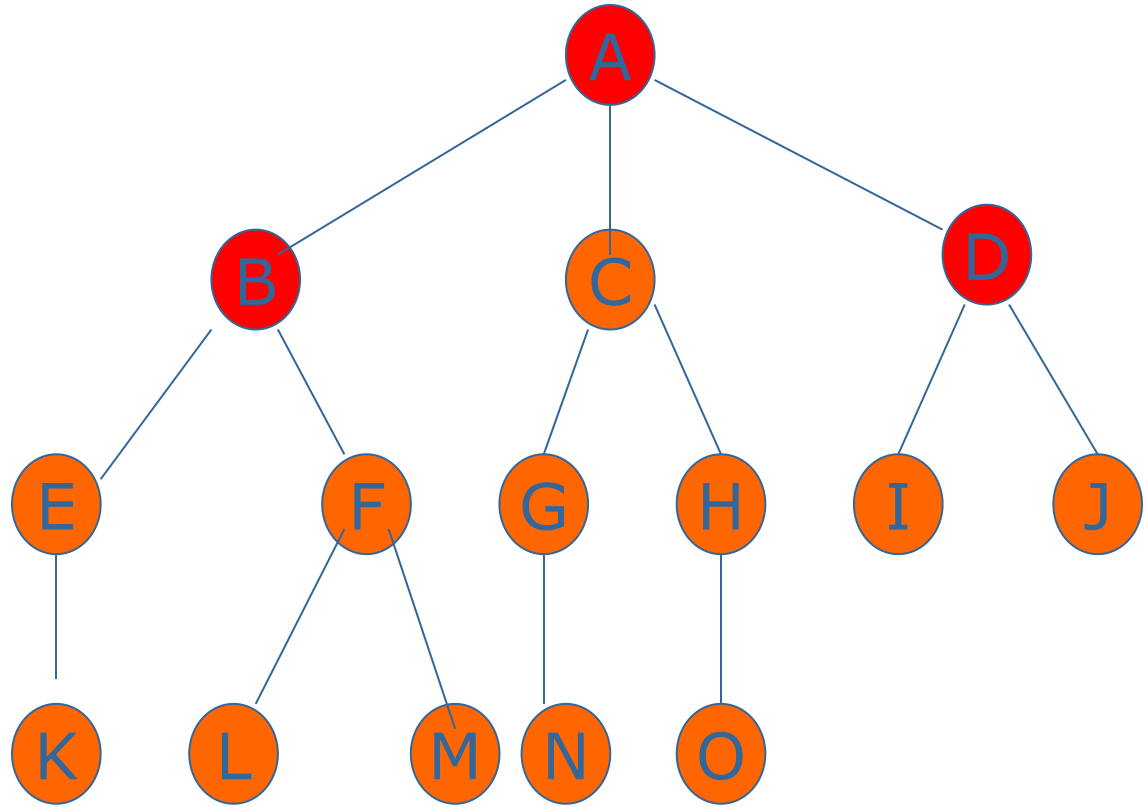
结点下标 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

结点标记 A B C D E F G H I J K L M N O

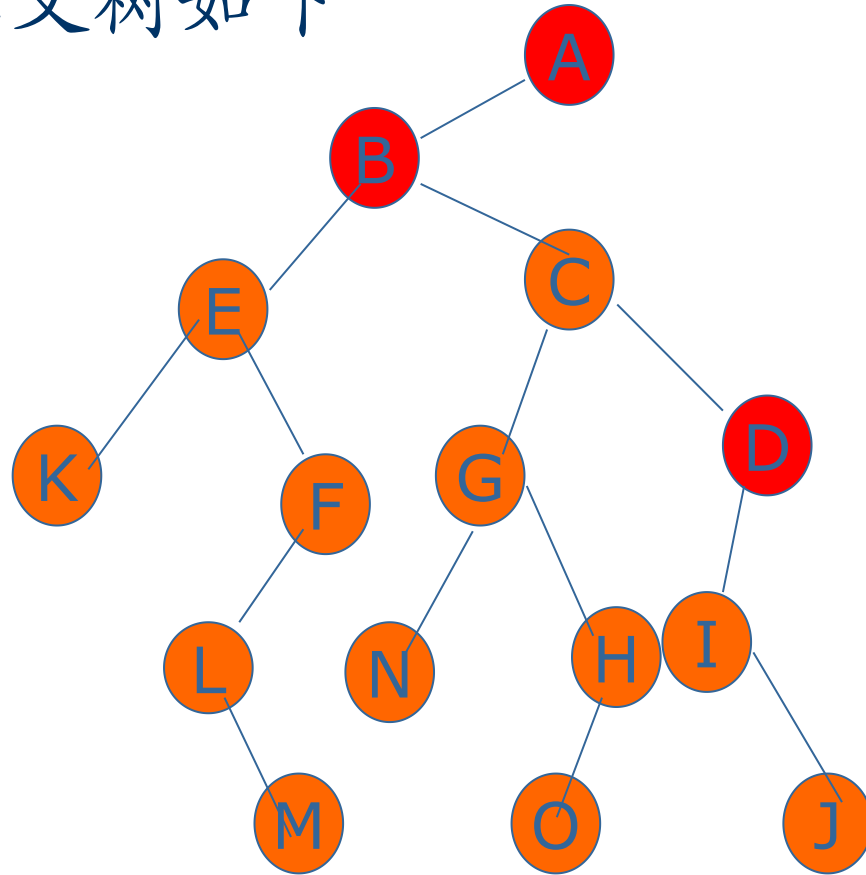
父结点下标

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	/	0	0	0	1	1	2	2	3	3	4	5	5	6	7

- 相应的树如下:



- 相应的二叉树如下





树和森林的遍历

一、树的遍历

二、森林的遍历

树的遍历可有三条搜索路径:

先根(次序)遍历:

若树不空, 则先访问根结点, 然后依次先根遍历各棵子树。

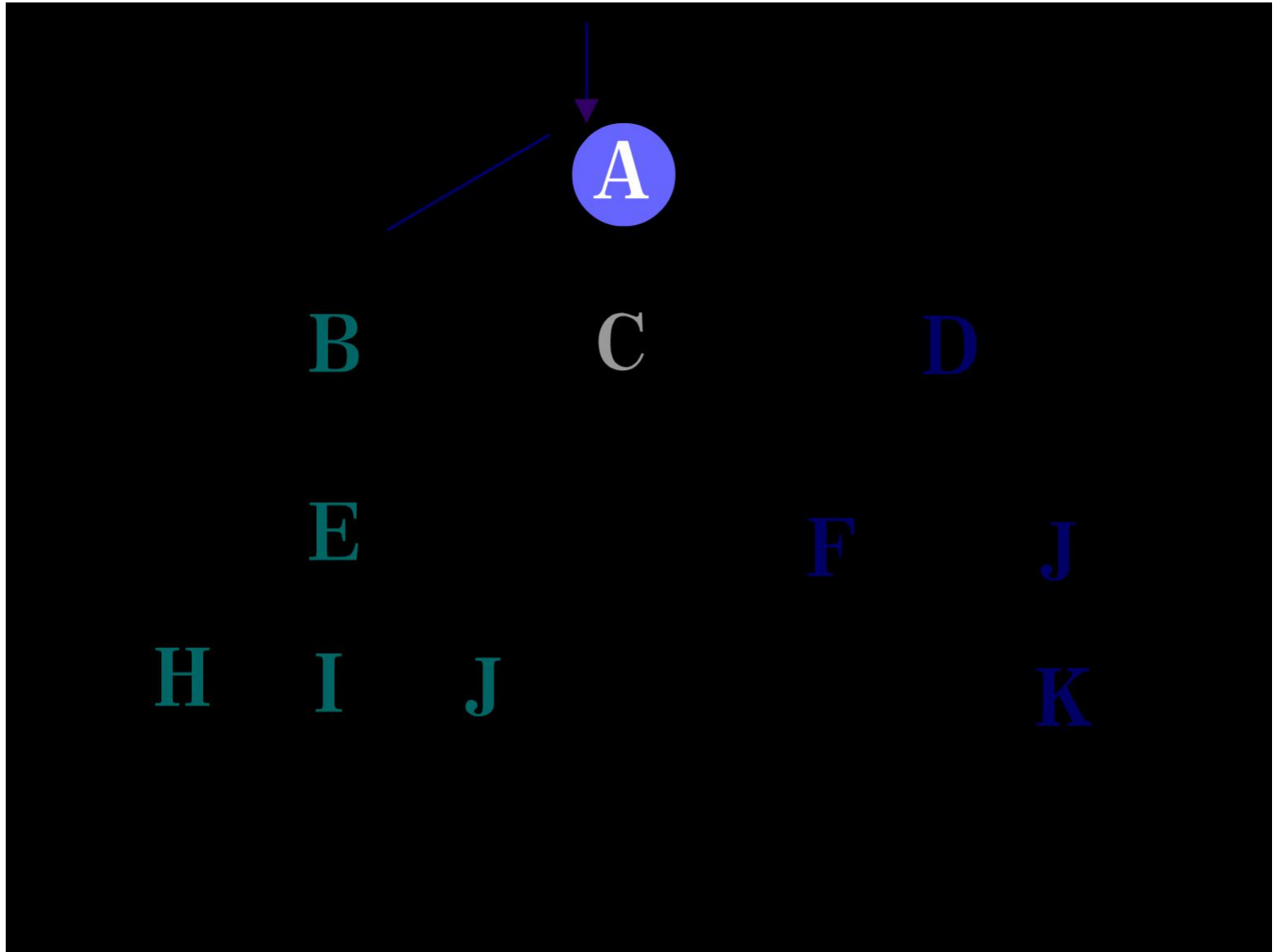
后根(次序)遍历:

若树不空, 则先依次后根遍历各棵子树, 然后访问根结点。

按层次遍历:

若树不空, 则自上而下自左至右访问树中每个结点。

先根遍历



后根遍历

A

B

C

D

E

F

J

H

I

J

K



先根遍历时顶点

的访问次序:

A B E F C D G H I J K

后根遍历时顶点

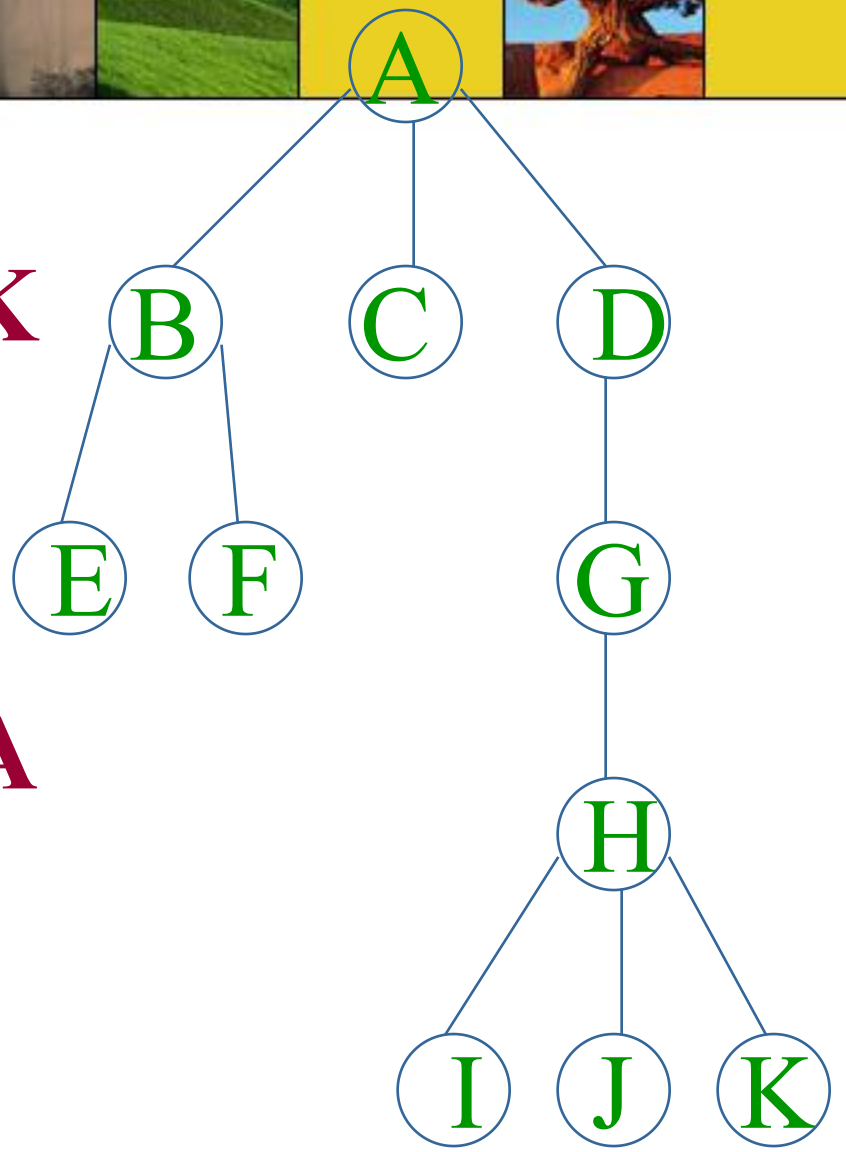
的访问次序:

E F B C I J K H G D A

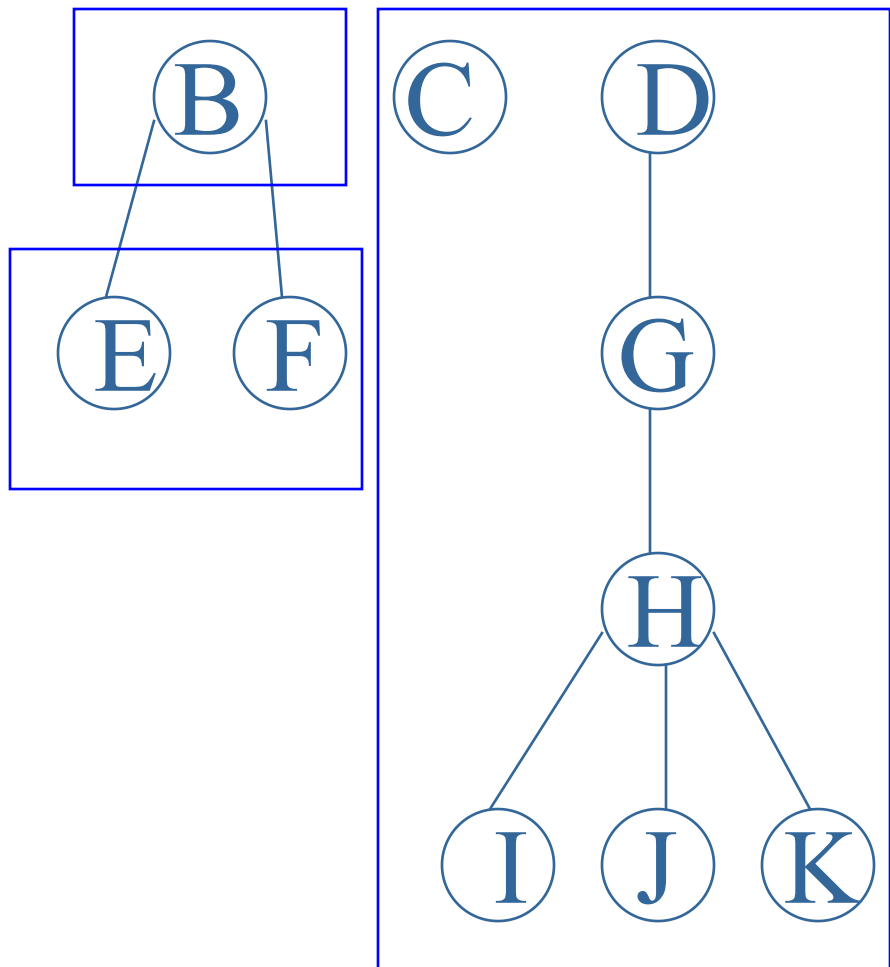
层次遍历时顶点

的访问次序:

A B C D E F G H I J K



森林由三部分构成:



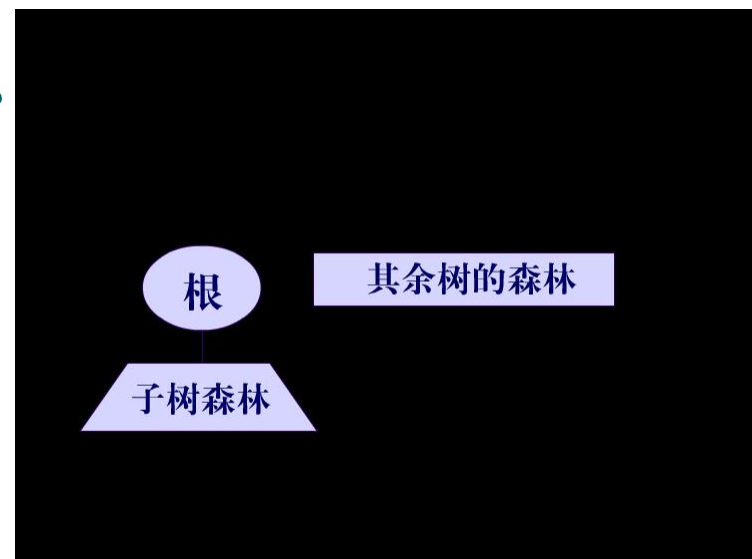
1. 森林中第一棵树的根结点;
2. 森林中第一棵树的子树森林;
3. 森林中其它树构成的森林。

森林的遍历



1. **先序遍历** 若森林不空，则
访问森林中第一棵树的根结点；
先序遍历森林中第一棵树的子树森林；
**先序遍历森林中 (除第一棵树之外) 其
余树构成的森林。**

即：依次从左至右对森林中的每一棵树进行**先根遍历**。



2. 中序遍历

若森林不空，则

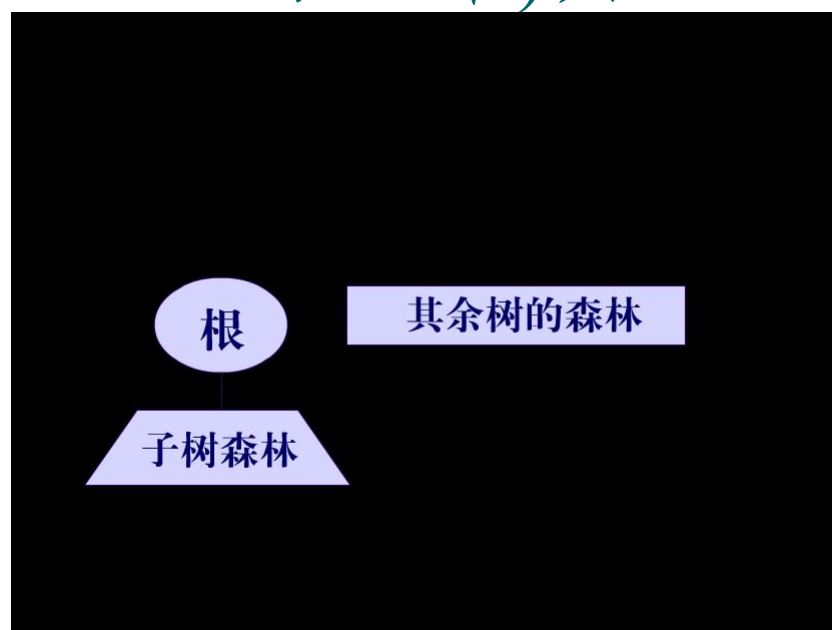
中序遍历森林中第一棵树的子树森林；

访问森林中第一棵树的根结点；

中序遍历森林中 (除第一棵树之外)其

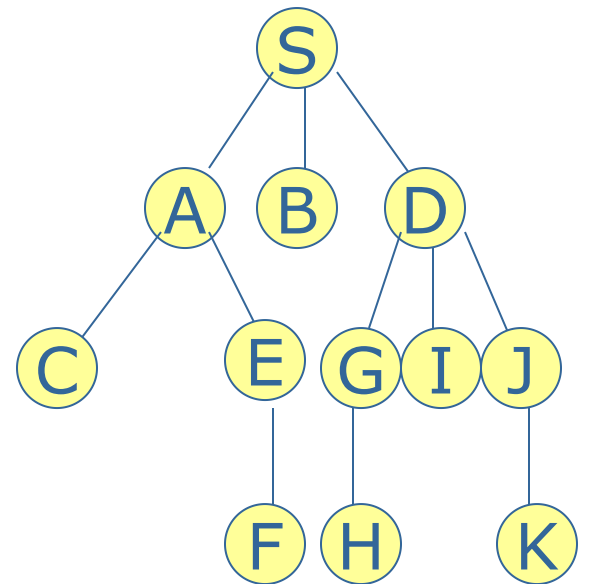
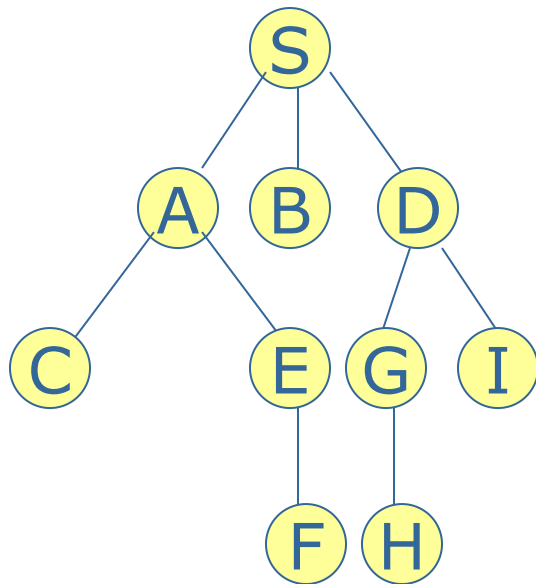
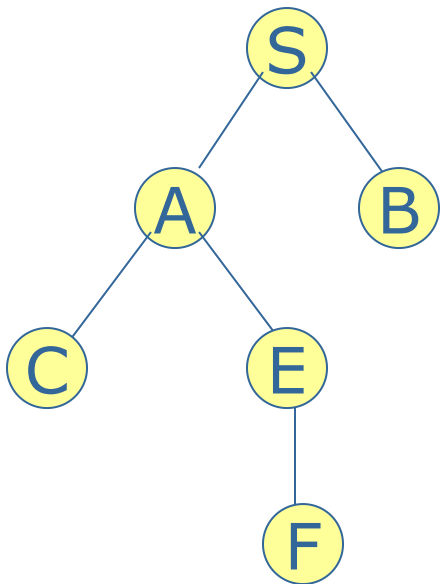
余树构成的森林。

即：依次从左至右对森林中的每一棵树进行后根遍历。



题十四：

假设先根次序遍历某棵树的结点次序为
SACEFBDGHIJK，后根次序遍历该树的结点次序
为CFEABHGIKJDS，要求画出这棵树。



树的遍历和二叉树遍历 的对应关系？

树

森林

二叉树

先根遍历

先序遍历

先序遍历

后根遍历

中序遍历

中序遍历



6.6 哈夫曼树及其应用

哈夫曼树又称最优树，是一类带权路径长度最短的树，有着广泛的应用。

基本概念：

从树中一个结点到另一个结点之间的分支构成这两个结点之间的**路径**，路径上的分支数目称做**路径长度**。

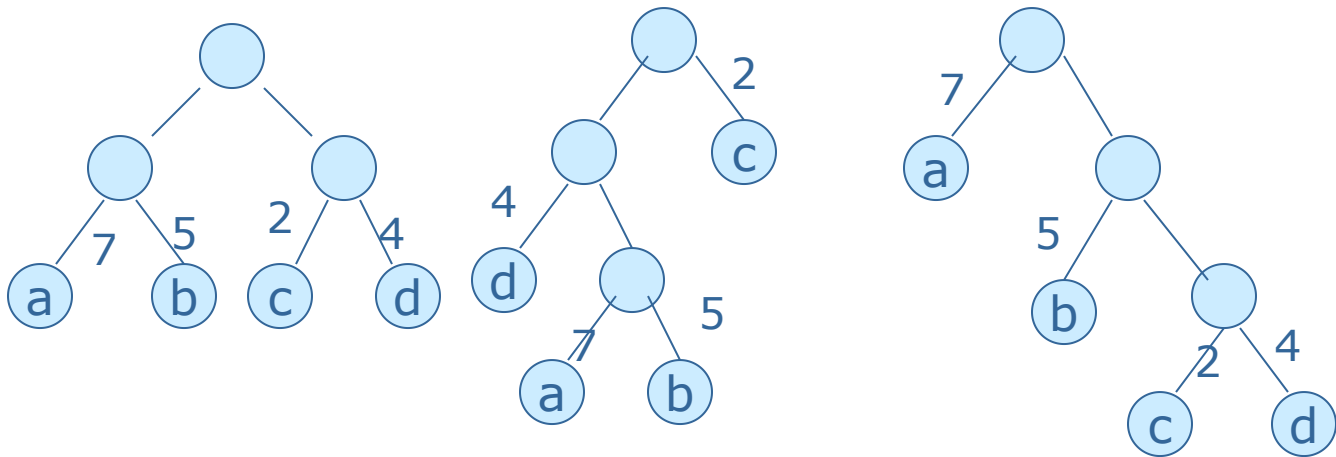
树的路径长度是从树根到每个结点的路径长度之和。

结点的带权路径长度为从该结点到树根之间的路径长度与结点上权的乘积。

树的带权路径长度为树中所有叶子结点的带权路径长度之和，通常记作：

$$WPL = \sum_{k=1}^n w_k l_k$$

最优二叉树（哈夫曼树）：设有n个权值 $\{w_1, w_2, w_3, \dots, w_n\}$ ，构造一棵有n个叶子结点的二叉树，每个叶子结点带权为 w_i ，其中WPL最小的二叉树即为哈夫曼树。



$$WPL_1 = 7 \cdot 2 + 5 \cdot 2 + 2 \cdot 2 + 4 \cdot 2 = 36$$

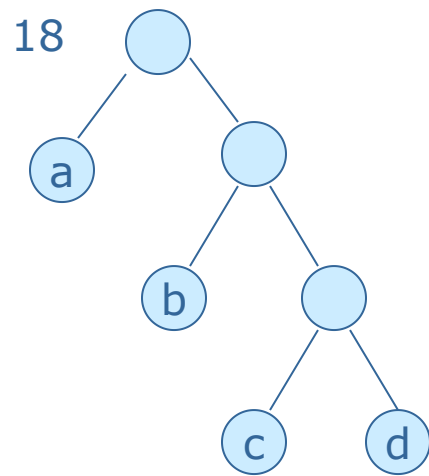
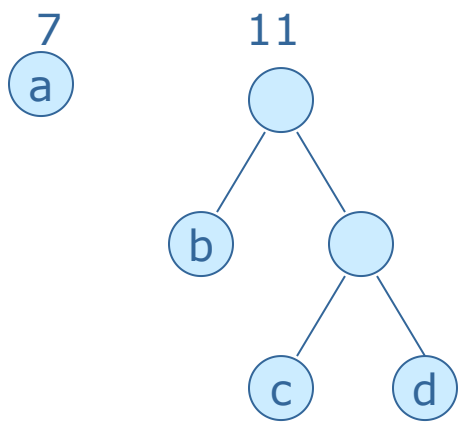
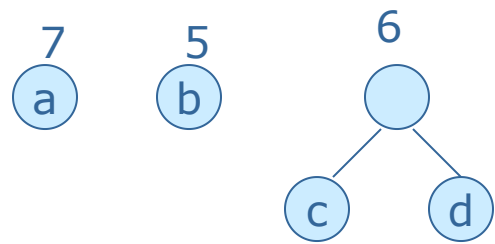
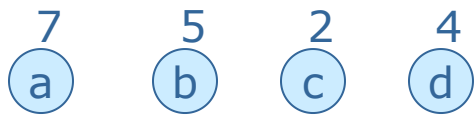
$$WPL_2 = 7 \cdot 3 + 5 \cdot 3 + 2 \cdot 1 + 4 \cdot 2 = 46$$

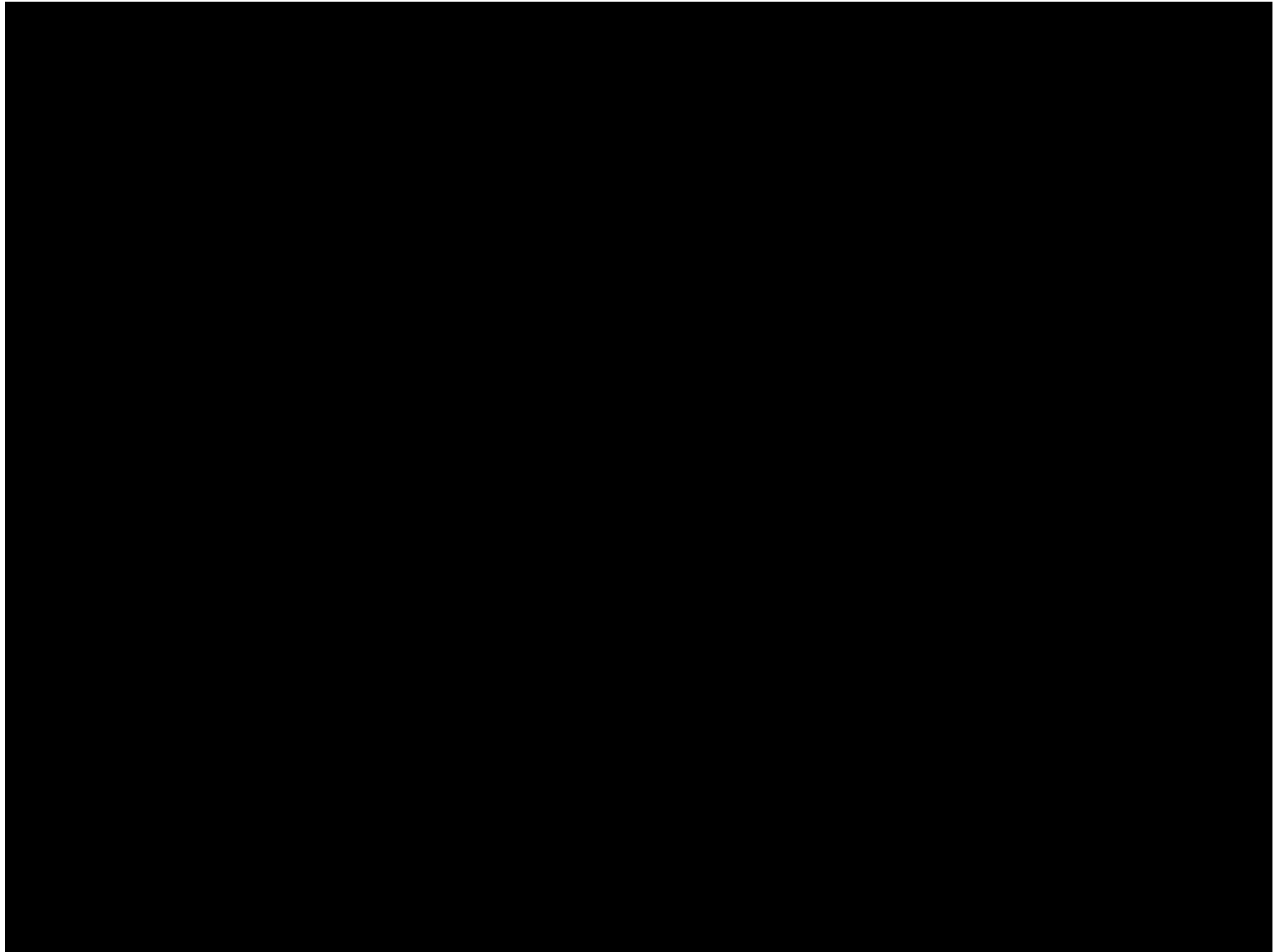
$$WPL_3 = 7 \cdot 1 + 5 \cdot 2 + 2 \cdot 3 + 4 \cdot 3 = 35$$



哈夫曼树的构造算法：

- 1、根据给定的 n 个权值 $\{w_1, w_2, \dots, w_n\}$ 构成 n 棵二叉树的集合 $F = \{T_1, T_2, \dots, T_n\}$ ，其中每棵二叉树 T_i 中只有一个带权为 w_i 的根结点，其左右子树均空；
- 2、在 F 中选取两棵根结点的权值最小的树作为左右子树构造一棵新的二叉树，且置新的二叉树的根结点的权值为其左、右子树的根结点的权值之和；
- 3、在 F 中删除这两棵树，同时将新得到的二叉树加入 F 中；
- 4、重复2和3，直到 F 中只含一棵树为止。



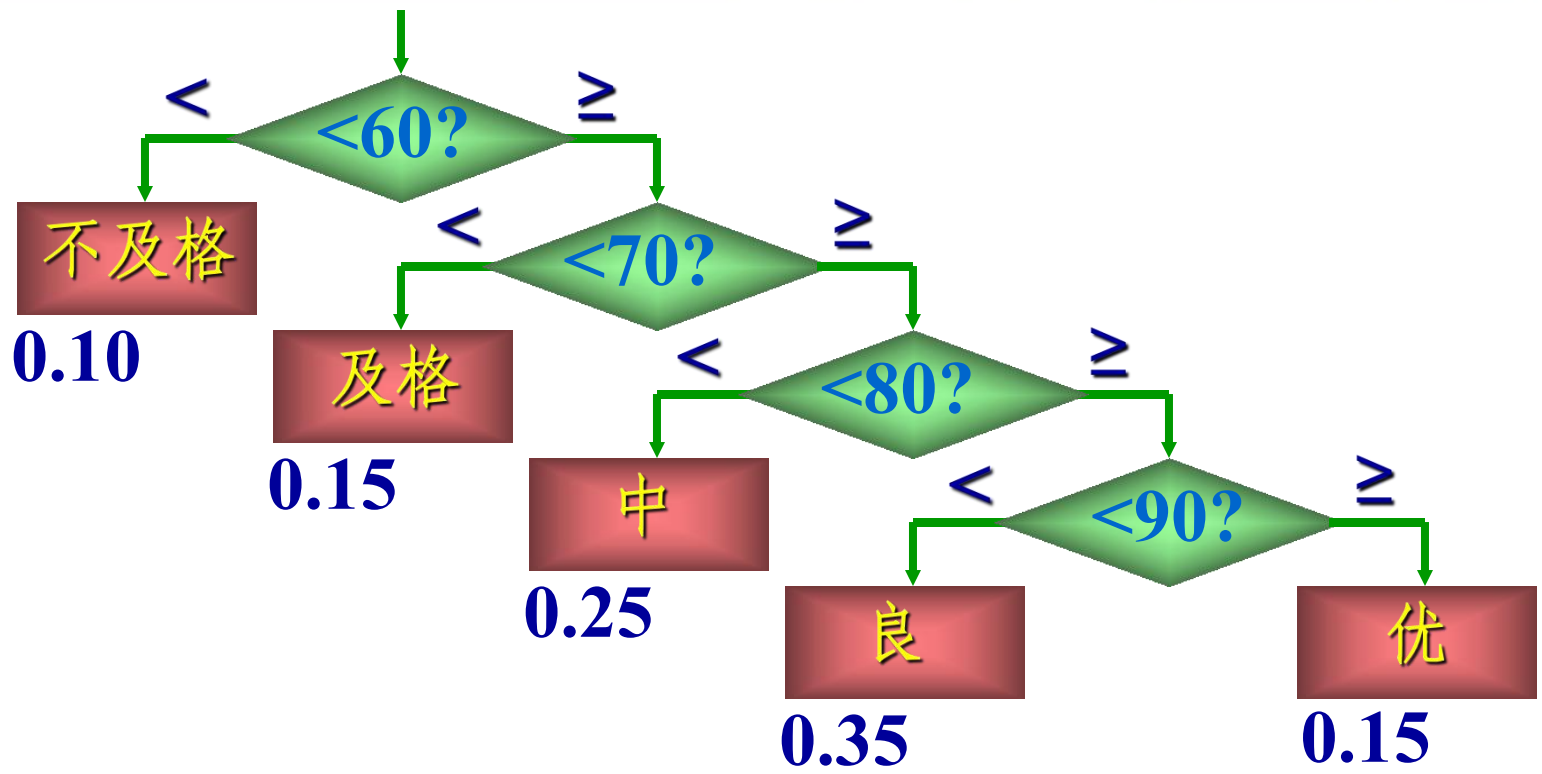


最佳判定树

考试成绩分布表

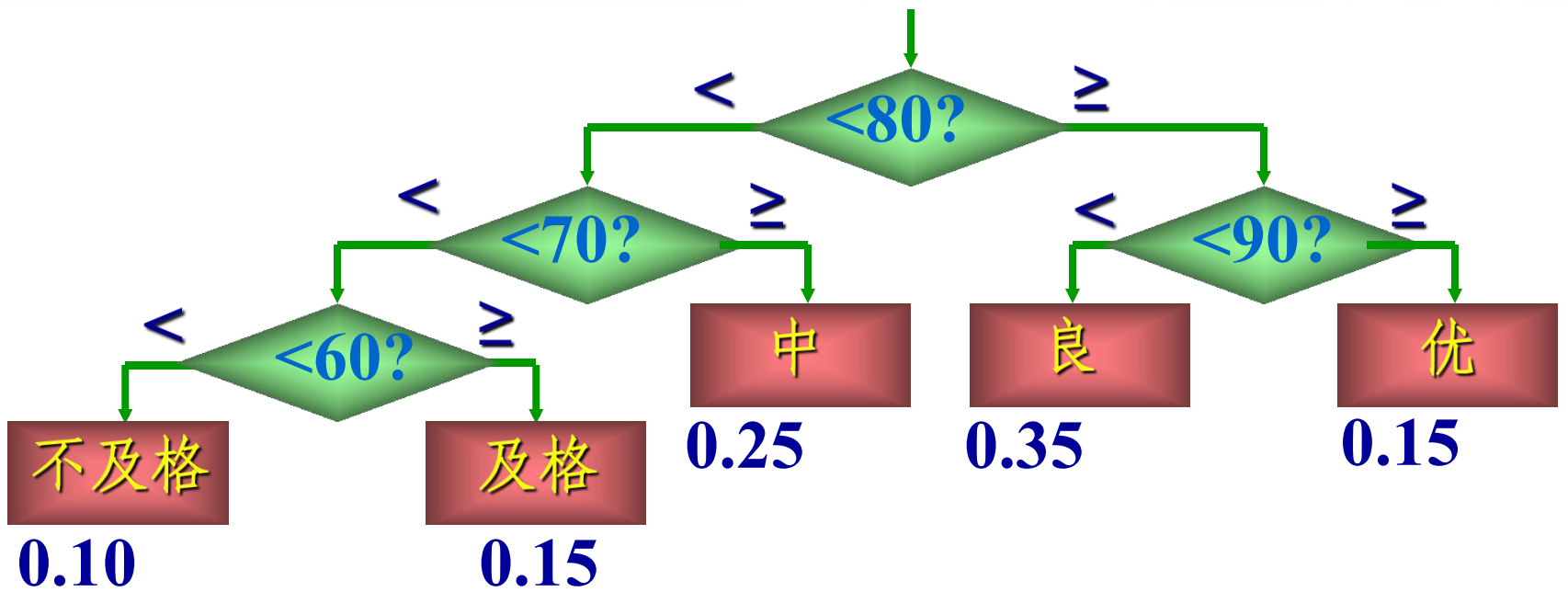
$[0, 60)$	$[60, 70)$	$[70, 80)$	$[80, 90)$	$[90, 100)$
不及格	及格	中	良	优
0.10	0.15	0.25	0.35	0.15

判定树



$$\begin{aligned} \text{WPL} &= 0.10 * 1 + 0.15 * 2 + 0.25 * 3 + 0.35 * 4 + 0.15 * 4 \\ &= 3.15 \end{aligned}$$

最佳判定树



$$\begin{aligned} \text{WPL} &= 0.10 * 3 + 0.15 * 3 + 0.25 * 2 + 0.35 * 2 + 0.15 * 2 \\ &= 0.3 + 0.45 + 0.5 + 0.7 + 0.3 = 2.25 \end{aligned}$$

霍夫曼编码

主要用途是实现数据压缩。

设给出一段报文：

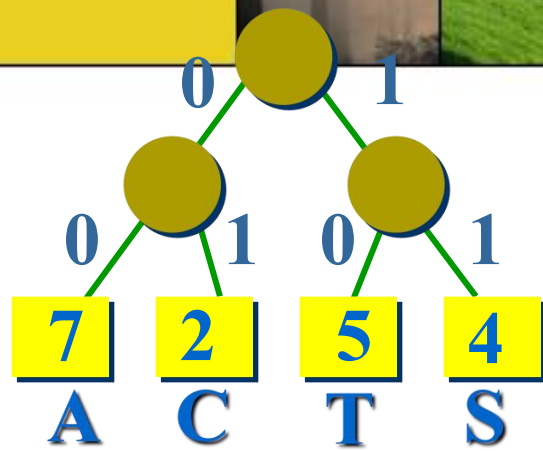
CAST CAST SAT AT A TASA

字符集合是 $\{C, A, S, T\}$ ，各个字符出现的频度(次数)是 $W = \{2, 7, 4, 5\}$ 。

若给每个字符以等长编码

A : 00 T : 10 C : 01 S : 11

则总编码长度为 $(2+7+4+5) * 2 = 36$ 。



若按各个字符出现的概率不同而给予不等长编码, 可望减少总编码长度。

各字符出现概率为 $\{2/18, 7/18, 4/18, 5/18\}$, 化整为 $\{2, 7, 4, 5\}$ 。以它们为各叶结点上的权值, 建立霍夫曼树。左分支赋 0, 右分支赋 1, 得霍夫曼编码(变长编码)。

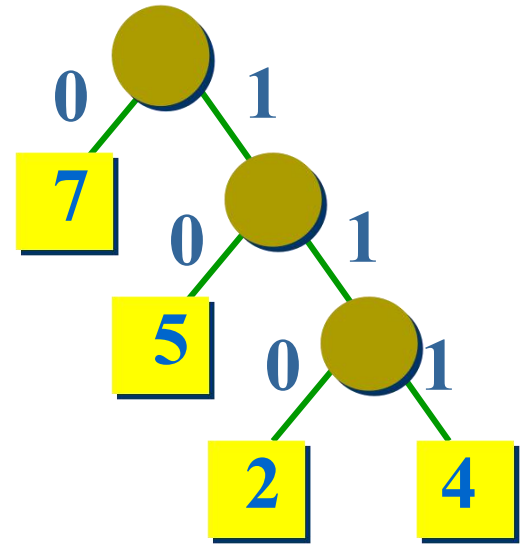
A : 0 T : 10 C : 110 S : 111

它的总编码长度: $7*1+5*2+(2+4)*3 = 35$ 。

比等长编码的情形要短。

总编码长度正好等于霍夫曼树的带权路径长度WPL。

霍夫曼编码是一种前缀编码。解码时不会混淆。



霍夫曼编码树

哈夫曼编码：

- 前缀编码思想的提出：在传送电文时，希望总长尽可能短，如果对每个字符设计长度不等的编码，且让电文中出现次数较多的字符采用尽可能短的编码，则传送电文的总长便可减少。
- 如果设计A, B, C, D的编码分别为0, 00, 1和01，则ABACCCDA可以转换成总长为9的字符串000011010，但是，这样的电文无法翻译，例如传送过去的字符串中前四个字符的子串0000就可能有多种译法，或是AAAA，或ABA，或BB。
- 因此，若要设计长短不等的编码，则必须是任一个字符的编码都不是另一个字符的编码的前缀，这种编码就称为前缀编码。

哈夫曼树和哈夫曼编码的存储表示

一棵有 n 个叶子结点的哈夫曼树共有 $2n-1$ 个结点。由于在构成哈夫曼树之后，为求编码需从叶子结点出发走一条从叶子到根的路径；而为译码须从根出发走一条从根到叶子的路径。则对每个结点而言，既需知双亲的信息，又需知孩子结点的信息。

由此设定下述存储结构：

```
typedef struct{
    unsigned int weight;
    unsigned int parent,lchild,rchild;
}HTNode,*HuffmanTree;
typedef char **HuffmanCode;
```

求哈夫曼编码的算法:

```
void HuffmanCoding(HuffmanTree &HT,HuffmanCode &HC,int *w,int n){
    //w存放n个字符的权值,构造哈夫曼树HT,并求出n个字符的Huffman编码
    if(n<=1) return;
    m=2n-1;
    HT=(HuffmanTree)malloc(m+1)*sizeof(HTNode);
    for(p=HT,i=1;i<=n;++i,++p,++w) *p={*w,0,0,0};
    for(;i<m;++i,++p) *p={0,0,0,0};//i=n+1 to m
    for(i=n+1;i<=m;++i){//建Huffman树
        //在HT[1..i-1]中选择parent为0且weight最小的两个结点,其序号分别为s1和s2
        Select(HT,i-1,s1,s2);
        HT[s1].parent=i; HT[s2].parent=i;
        HT[i].lchild=s1; HT[i].rchild=s2;
        HT[i].weight= HT[s1].weight+HT[s2].weight;
    }
}
```

//从叶子到根逆向求每个字符的Huffman编码

HC= (HuffmanCode)malloc((n+1)*sizeof(char));//分配n个字符编码的头指针变量

Cd=(char *)malloc(n*sizeof(char));

cd [n-1]='\0';

for(i=1;i<=n;i++){

start=n-1;

for(c=i,f=HT[i].parent;f!=0;c=f,f=HT[f].parent)

if(HT[f].lchild==c) cd[--start]='0';

else cd[--start]='1';

HC[i]=(char *)malloc((n-start)*sizeof(char));

strcpy(HC[i],&cd[start]);

}

Free(cd);

}

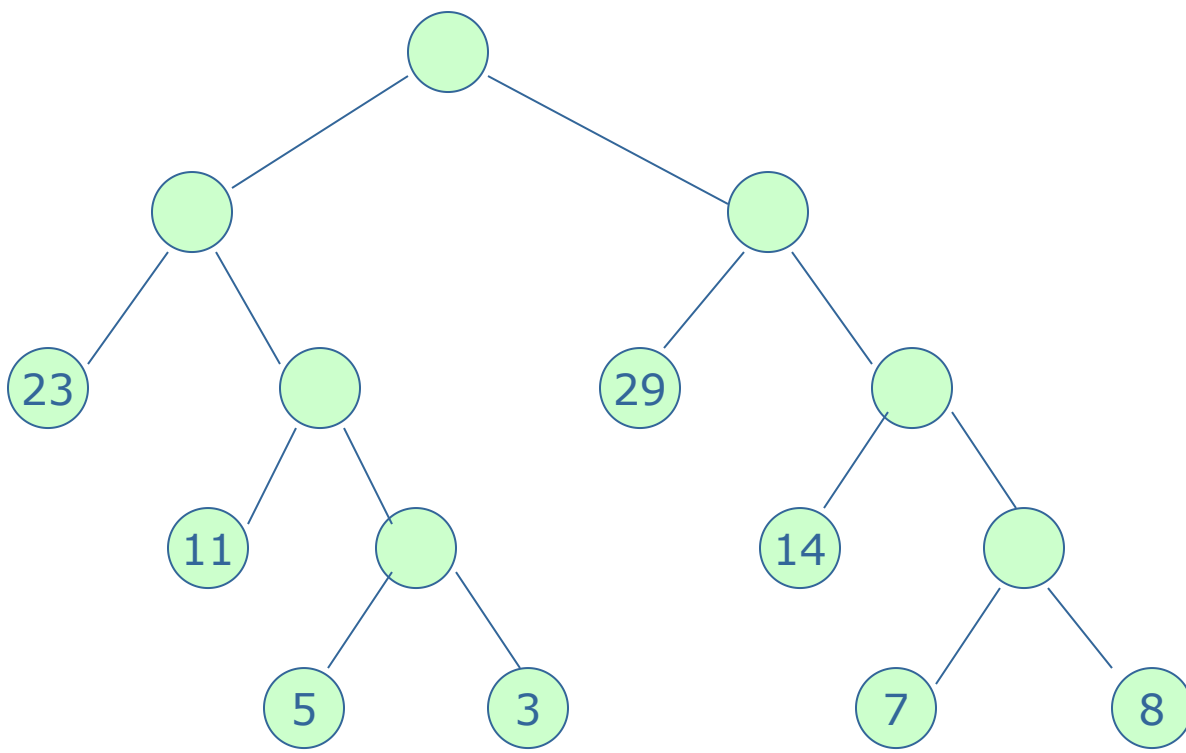
例11 已知某系统在通信联络中只可能出现八种字符，其概率分别为0.05，0.29，0.07，0.08，0.14，0.23，0.03，0.11，试设计Huffman编码。

设权 $w=(5, 29, 7, 8, 14, 23, 3, 11)$ ， $n=8$ ，则 $m=15$


存储结构的初始状态为：

	weight	parent	lchild	rchild
1	5	0		
2	29	0		
3	7	0		
4	8	0		
5	14	0		
6	23	0		
7	3	0		
8	11	0		
9		0		
10		0		
11		0		
12		0		
13		0		
14		0		
15		0		

	weight	parent	lchild	rchild
1	5 ×	0(9)		
2	29	0(14)		
3	7 ×	0(10)		
4	8 ×	0(10)		
5	14 ×	0(12)		
6	23 ×	0(13)		
7	3 ×	0(9)		
8	11 ×	0(11)		
9	8 ×	0(11)	1	7
10	15 ×	0(12)	3	4
11	19 ×	0(13)	8	9
12	29	0(14)	5	10
13	42	0(15)	6	11
14	58	0(15)	2	12
15	100	0	13	14



1	→	0 1 1 0
2	→	1 0
3	→	1 1 1 0
4	→	1 1 1 1
5	→	1 1 0
6	→	0 0
7	→	0 1 1 1
8	→	0 1 0



题十五：设某通信电文由A, B, C, D, E, F六个字符组成，它们出现的频率分别为2, 16, 14, 8, 5, 17。试问字符D的Huffman编码由几个二进位组成？

A、1 B、2 C、3 D、4 E、5

题十六：已知a, b, c, d, e, f, g, h字符出现的百分比分别为45%, 15%, 10%, 8%, 5%, 5%, 4%, 8%, 试给出它们的Huffman编码。



【本章小结】

在这一章讨论了树和二叉树两种数据类型的定义以及它们的实现方法。树是以分支关系定义的层次结构，结构中的数据元素之间存在着“一对多”的关系，因此它为计算机应用中出现的具有层次关系或分支关系的数据，提供了一种自然的表示方法。如用树描述人类社会的族谱和各种社会组织机构。在计算机学科和应用领域中树也得到广泛应用，例如在编译程序中，用树来表示源程序的语法结构等。

二叉树是和树不同的另一种树型结构，它有明确的左子树和右子树，因此当用二叉树来描述层次关系时，其“左孩子”表示“下属关系”，而“右孩子”表示的是“同一层次的关系”。由于二叉树还是以后各章中讨论其它问题时经常用到的工具，因此二叉树的几个重要特性是我们应该熟练掌握的。



【本章小结】

树和二叉树的遍历算法是实现各种操作的基础。对非线性结构的遍历需要选择合适的搜索路径，以确保在这条路径上可以访问到结构中的所有数据元素，并使每一个数据元素只被访问一次。由于树和二叉树是层次分明的结构，因此按层次进行遍历是自然的事，它必能实现既访问到所有元素，又不会重复访问。此外，对树和二叉树还可进行先左后右的遍历。

遍历的实质是按某种规则将二叉树中的数据元素排列成一个线性序列，二叉树的线索链表便可看成是二叉树的一种"线性"存储结构，在线索链表上可对二叉树进行"线性化"的遍历，即不需要"递归"，而是从"第一个元素"起，逐个访问"后继元素"直至"后继"为"空"止。因此线索链表是通过遍历生成的，即在遍历过程中保存结点之间的"前驱"和"后继"的关系，并为方便起见，在线索链表中添加一个"头结点"，并由此构成一个"双向循环链表"。在实际应用时也可简化为"单向循环链表"（即只保存后继或前驱关系）。

在这一章作为应用，介绍了最优树和最优前缀编码的构造方法，最优树是一种"带权路径长度最短"的树，最优前缀编码是最优二叉树的一种应用。

作业

- 1. 已知一棵树边的集合为
 $\{ \langle I, M \rangle, \langle I, N \rangle, \langle E, I \rangle, \langle B, E \rangle, \langle B, D \rangle, \langle A, B \rangle, \langle G, J \rangle, \langle G, K \rangle, \langle C, G \rangle, \langle C, F \rangle, \langle H, L \rangle, \langle C, H \rangle, \langle A, C \rangle \}$, 请画出这棵树, 并回答下列问题:
 - (1) 哪个是根结点?
 - (2) 哪些是叶子结点?
 - (3) 哪个是结点 G 的双亲?
 - (4) 哪些是结点 G 的祖先?
 - (5) 哪些是结点 G 的孩子?
 - (6) 哪些是结点 E 的子孙?
 - (7) 哪些是结点 E 的兄弟? 哪些是结点 F 的兄弟?
 - (8) 结点 B 和 N 的层次号分别是什么?
 - (9) 树的深度是多少?
 - (10) 以结点 C 为根的子树的深度是多少?

2. 假设 n 和 m 为二叉树中两结点，用“1”、

“0”或“ ϕ ”(分别表示肯定、恰恰相反或者不一定) 填写下表

答 已知	问	前序遍历时 n 在 m 前?	中序遍历时 n 在 m 前?	后序遍历时 n 在 m 前?
	n 在 m 左方			
	n 在 m 右方			
	n 是 m 祖先			
	n 是 m 子孙			

不一定，一定，不一定

不一定，一定不，不一定

一定，不一定，一定不

→ 1 1 → →



- 3. 假设用于通讯的电文仅由 8 个字母组成，字母在电文中出现的频率分别为 0.07, 0.19, 0.02, 0.06, 0.32, 0.03, 0.21, 0.10。试为这 8 个字母设计哈夫曼编码。使用 0~7 的二进制表示形式是另一种编码方案。对于上述实例，比较两种方案的优缺点。
- 4. 假设一棵二叉树的前序序列为 EBADCFHGKJ 和中序序列为 ABCDEFGHIJK。请画出该树。