BUPT

第5章　树与二叉树

杨震　计算机科学与技术学院

5.1 树的结构及基本操作

树是一类重要的**非线性**数据结构，是以**分支关系**定义的层次结构。

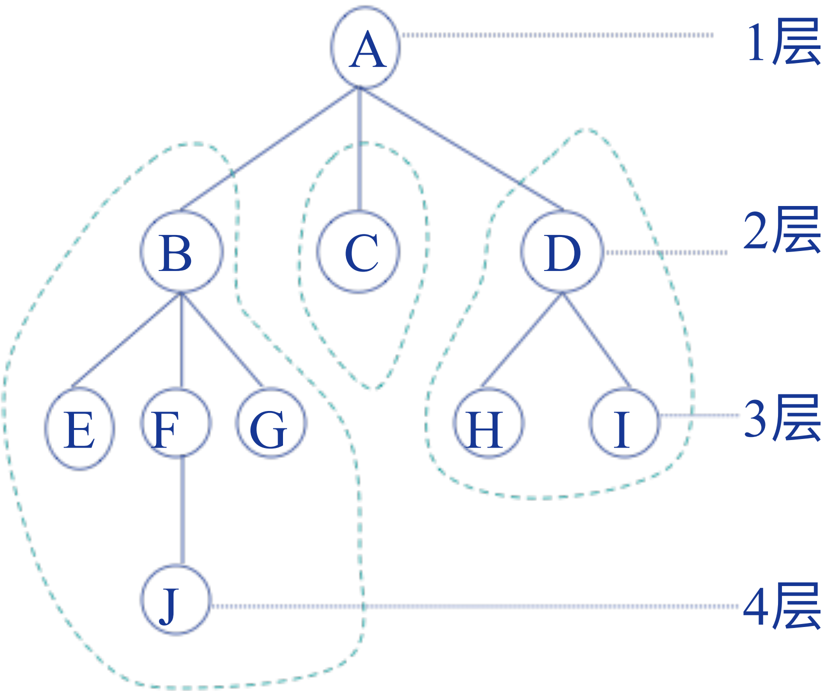
[树的定义]

树是由n(n□0)个结点组成的有限集合T，非空树满足：

1)有一个称之为**根(root)**的结点。

2)除根以外的其余结点被分成m(0□m<n)个互不相交的集合T1， T2，□， Tm，其中每一个集合本身又是一棵树，且称为根的**子树**。

[树的特点]



特点 除根结点外，每个结点都仅有一个前趋(父)结点。

[基本术语]

结点的度 结点拥有的子树数目。

叶子(终端)结点 度为0的结点。

分支(非终端)结点 度不为0的结点。

树的度 树的各结点度的最大值。

内部结点 除根结点之外的分支结点。

双亲与孩子(父与子)结点 结点的子树的根称为该结点的孩子；该结点称为孩子的双亲。

兄弟 属于同一双亲的孩子。

结点的祖先 从根到该结点所经分支上的所有结点。

结点的子孙 该结点为根的子树中的任一结点。

结点的层次 表示该结点在树中的相对位置。根为第一层，其它的结点依次下推；若某结点在第L层上，则其孩子在第L+1层上。

堂兄弟 双亲在同一层的结点互为堂兄弟。

树的深(高)度 树中结点的最大层次。

有序树 树中各结点的子树从左至右是有次序的，不能互换。否则，称为**无序树**。

路径长度 从树中某结点Ni出发，能够“自上而下地”通过树中结点到达结点Nj，则称Ni到Nj存在一条路径，路径长度等于这两个结点之间的分支数。

树的路径长度 从根到每个结点的路径长度之和。

森林 是m(m□0)棵互不相交的树的集合。

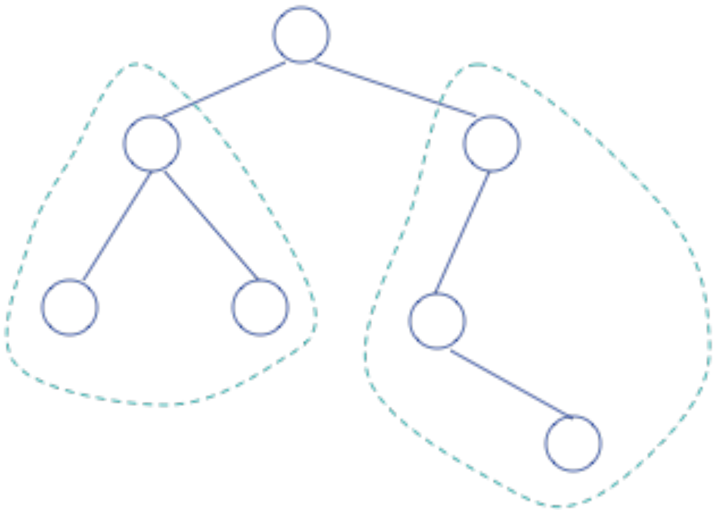
[树的基本操作]

1)初始化操作	initiate(T)
2)求根函数	root(T) / root(x)
3)求双亲函数	parent(T,x)
4)求孩子结点函数	child(T,x,i)
5)求右兄弟函数	right_sibling(T,x)
6)建树函数	crt_tree(x,F)
7)插入子树操作	ins_child(y,i,x)
8)删除子树操作	del_child(x,i)
9)遍历操作	traverse(T)

5.2 二叉树的定义与性质

[二叉树的概念]

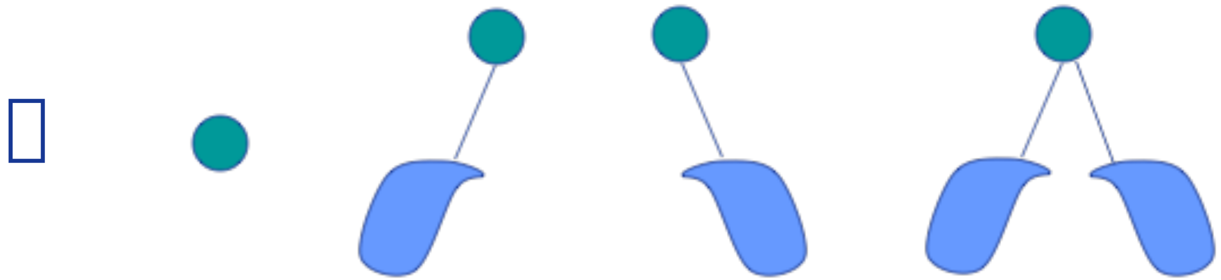
二叉树的定义 二叉树是n(n□0)个结点的有限集合，它或为**空树**(n=0)，或由一个**根**结点和两棵互不相交的**左子树**和**右子树**的二叉树组成。



二叉树的特点

- 定义是递归的
- 0□结点的度□2
- 是有序树

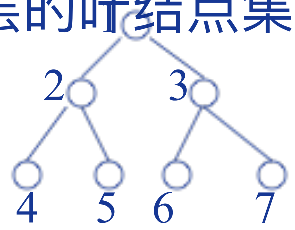
二叉树的五种基本形态

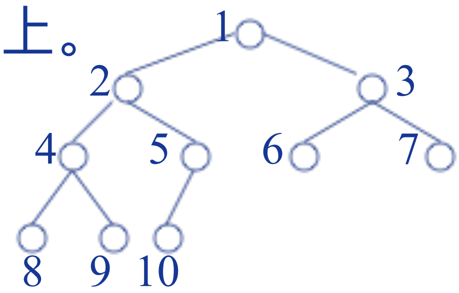


两种特殊的二叉树

满二叉树 每一层上的结点数都是最大结点数。

完全二叉树 只有最下面两层结点的度可小于2，而最下一层的叶结点集中在左边若干位置上。





[二叉树的性质]

性质1 二叉树的第i层上至多有**2i-1**(i□1)个结点。

性质2 深度为k的二叉树至多有**2k-1**个结点(k□1)。

性质3 对任何一棵二叉树T，如果其终端结点数为n0，度为2的结点数为n2，则**n0 = n2 + 1**。

性质4 具有n个结点的完全二叉树的深度为 **□log2n□+1**。

性质5 一棵具有n个结点的完全二叉树(又称顺序二叉树)，对其结点按层从上至下(每层从左至右)进行1至n的编号，则对任一结点i(1□i□n)有：

(1)若i>1，则i的双亲是**□i/2□**；若i=1，则i是根，无双亲。

(2)若2i□n，则i的左孩子是**2i**；否则， i无左孩子。

(3)若2i+1□n，则i的右孩子是**2i+1**；否则， i无右孩子。

顺序存储结构

[**完全二叉树**：按完全二叉树编号存放]

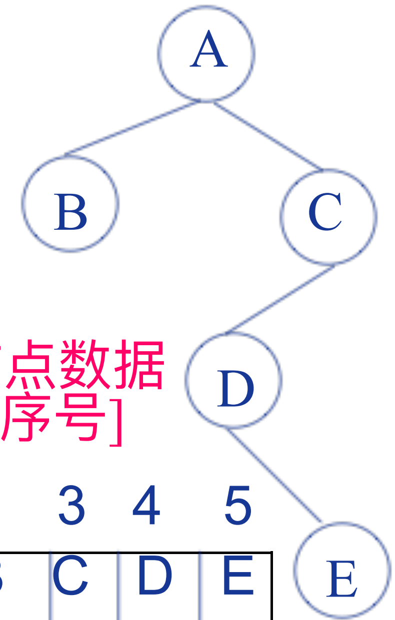
A	B	C	D	E
1	2	3	6	13

[**三元组**：存储结点数据和左、右孩子在向量中的序号]

	0	1	2	3	4
data	A	B	C	D	E
lc	1	-1	3	-1	-1
rc	2	-1	-1	4	-1

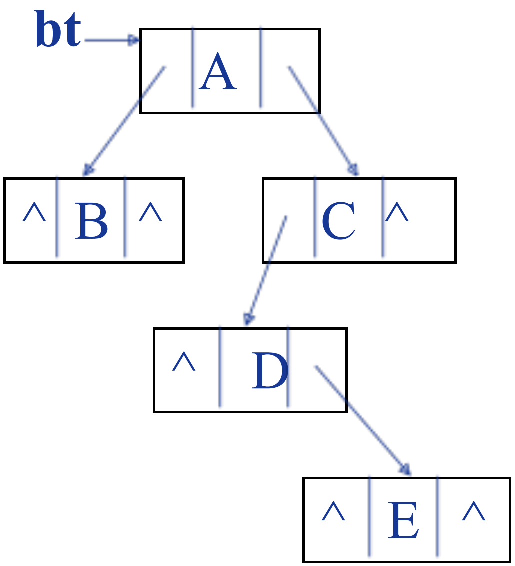
[**双亲**：存储结点数据和 其父结点的序号]

	1	2	3	4	5
data	A	B	C	D	E
parent	0	1	-1	3	-4



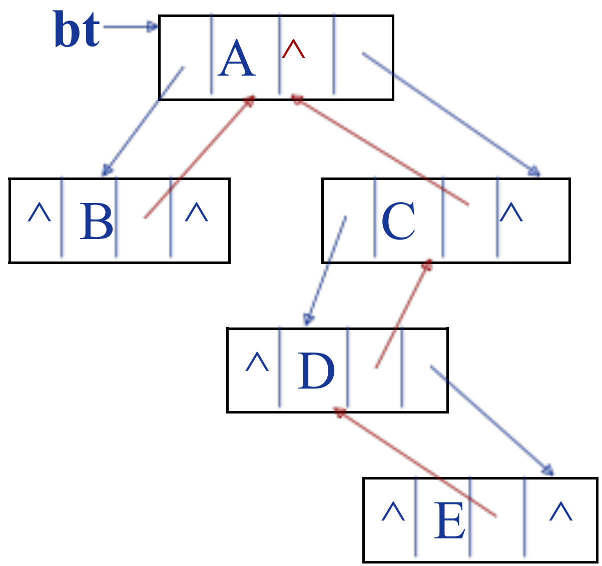
[二叉链表]

lchild	data	rchild



[三叉链表/带双亲的二叉链表]

lchild	data	parent	rchild



结论: 若一个二叉树含有n个结点，则它的二叉链表中必含有2n个

二叉链表的类型定义

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

定义指针变量，用来存放根结点地址，通常用该指针标识一个二叉树

```
BiTree root;
```

三叉链表的类型定义

```
typedef
struct btnode{
btnode *lchild,rchild;
btnode *parent;
elemtp  data;
} *BiTree;
```

若要建立不带头结点的二叉树，可描述如下：
建立一棵空的不带头结点的二叉树

```
 BiTree Initiate ()     /*初始建立一棵空的不带头结点
                          的二叉树*/
   {  BiTNode  *bt;
      bt=NULL;
      return bt;
   }
```

在主函数中，可以通过如下方式调用Initiate函数：

```
main ( )
 {BiTree t ;        /*定义根结点指针变量*/
  t =Initiate ();
   ……
 }
```

2) Create(x，lbt，rbt)

生成一棵以x为根结点的数据域值以lbt和rbt为左右子树的二叉树。

```
BiTree Create(elemtype x, BiTree lbt, BiTree rbt)
 {BiTree  p;
  if((p=(BiTNode*)malloc(sizeof(BiTNode)))==NULL)
      return NULL;
   p->data=x;
   p->lchild=lbt;
   p->rchild=rbt;
   return p;
 }
```

(3) InsertL(bt, x, parent)：

在二叉树bt中的parent所指结点和其左子树之间插入数据元素为x的结点

```
BiTree InsertL(BiTree bt，elemtype x，BiTree parent)
 {BiTree  p;
  if (parent==NULL)
   {printf("\n插入出错")；
    return NULL;
   }
  if ((p=(BiTNode *)malloc(sizeof(BiTNode)))==NULL) return NULL;
  p->data=x;
  p->lchild=NULL;
  p->rchild=NULL;
  if (parent->lchild==NULL) parent->lchild=p;
  else {p->lchild=parent->lchild;
        parent->lchild=p;
       }
  return bt;
}
```

(4) InsertR(bt，x，parent)：功能类同于(3)，算法略。

(5) DeleteL(bt，parent)

在二叉树bt中删除parent的左子树

```
BiTree  DeleteL(BiTree bt，BiTree parent)
  { BiTree  p;
    if (parent==NULL||parent□>lchild==NULL)
      { printf("\n删除出错")；
        return NULL;
      }
    p=parent□>lchild;
    parent□>lchild=NULL;
    free(p);
/*当*p为非叶子结点时，这样删除仅释放了所删子树根结点的空间，*/
/*若要删除子树分支中的结点，需用后面介绍的遍历操作来实现。*/
    return bt;
 }
```

(6) DeleteR(bt，parent)：算法略。

5.4 二叉树的遍历

[遍历的目的] 非线性结构□线性结构

[遍历的概念] 指按某条搜索路线走遍二叉树的每个结点，使得树中每个结点都被访问一次，且仅被访问一次。

[典型的遍历方法]

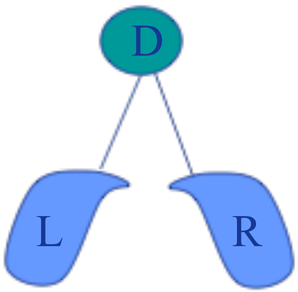
先(根)序遍历 DLR

中(根)序遍历 LDR

后(根)序遍历 LRD

层序遍历

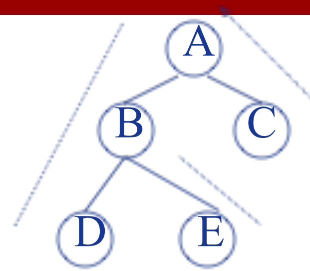
上述逆序方式很少使用



先序遍历：ABDEC

中序遍历：DBEAC

后序遍历：DEBCA



[利用遍历结果确定二叉树]

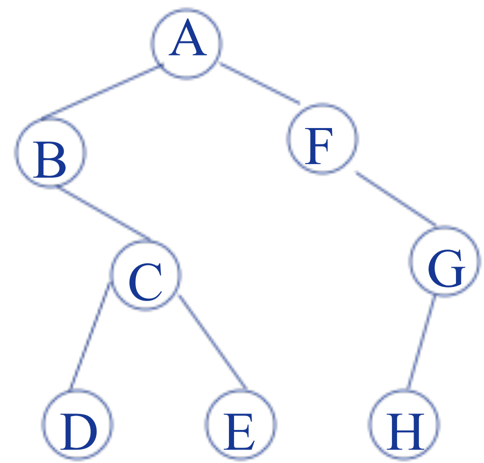
先序序列+中序序列

中序序列+后序序列

先序序列+后序序列

思考：层序+先序/中序/后序

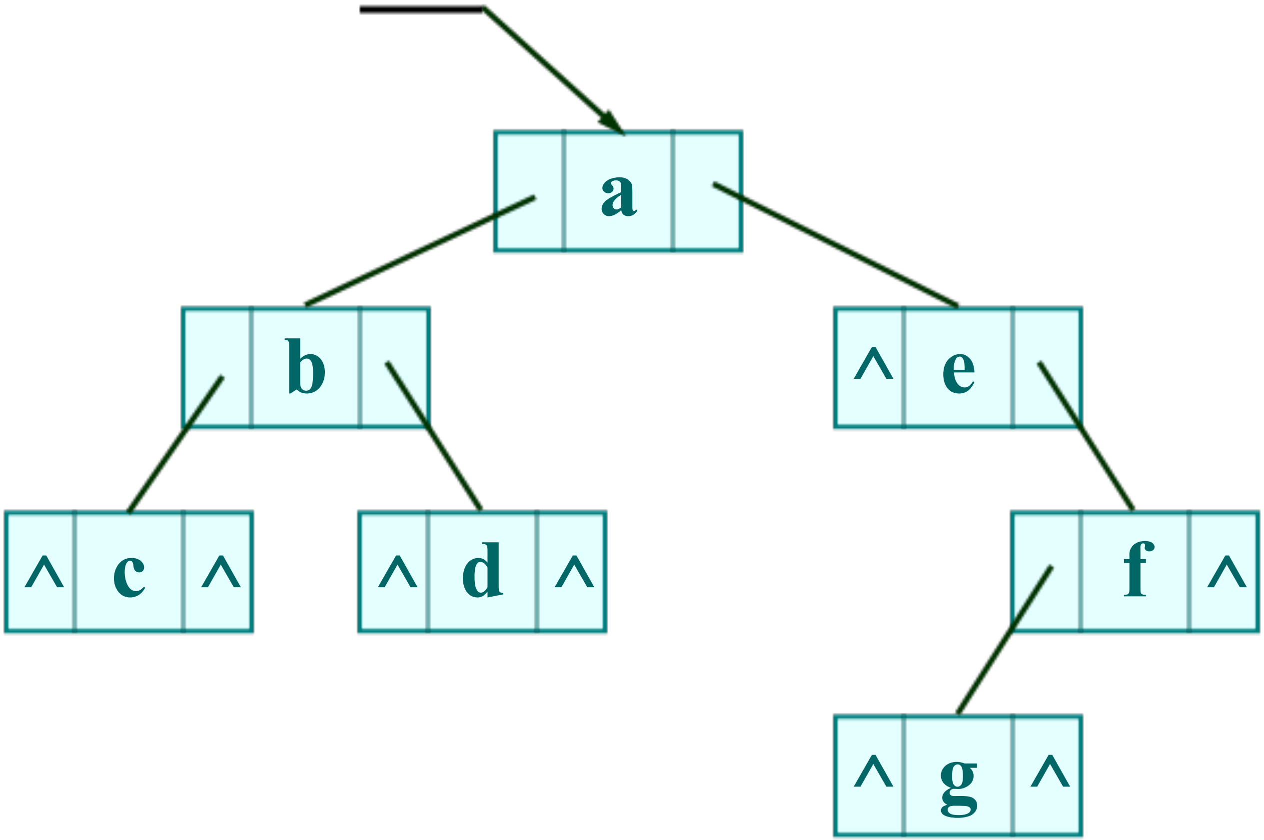
能否确定？如何做?



先序序列: ABCDEFGH

a b c d e f g 先序序列

c b d a e g f 中序序列



[先序遍历算法]

[递归算法]

```
void Preorder (BiTree T
{
    if (T)
    { visit(T);//可以是打印语句
        Preorder(T->lchild);
         Preorder(T->rchild);
     }
}
```

[另一种描述]

```
void Preorder2 (BiTree T)
{
    visit(T);
    if (T->lc)
            Preorder2(T->lchild);
    if (T->rc)
            Preorder2(T->rc);
}
```

[消除尾递归的递归算法]

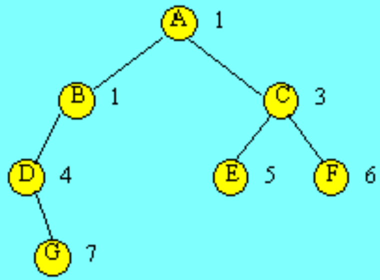
```
void Preorder3 (BiTree T)
{
    while (T)
   {
      visite(T);
      Preorder3(T->lchild);
      T = T->rchild;
}
```

[非递归算法]

```
void Preorder4 (BiTreeT)
{ inistack(s);
   p=bt;
   push(s,NULL);
   while (p)
   {     visit(p);
         if (p->rchild)
             push(s, p->rchild);
         if ( p->lchild)
              p=p->lchild;
         else
             p=pop(s);
      }
}
```

先序遍历的非递归算法

```
void NRPreOrder(BiTree bt)
{/*非递归先序遍历二叉树*/
  BiTNode* stack[MAXNODE],p;          /*设足够大的栈空间*/
  int top=-1;       /*栈初始化为空*/
  if (bt==NULL) return;
  p=bt;   /*p指向根结点*/
  while(!(p==NULL&&top==-1))     /*指针p为空且栈空时结束*/
    { while(p!=NULL)
        {Visit (p);                /*访问当前结点*/
         top++;
stack[top]=p;          /*将当前指针p压栈*/
 p=p->lchild;         /*指针指向p的左孩子结点*/
        }
      if (top<0)  return;      /*栈空时结束*/
      else  { p=stack[top];
           top - -;          /*从栈中弹出栈顶元素*/
           p=p->rchild;        /*到栈顶元素右子树*/
```



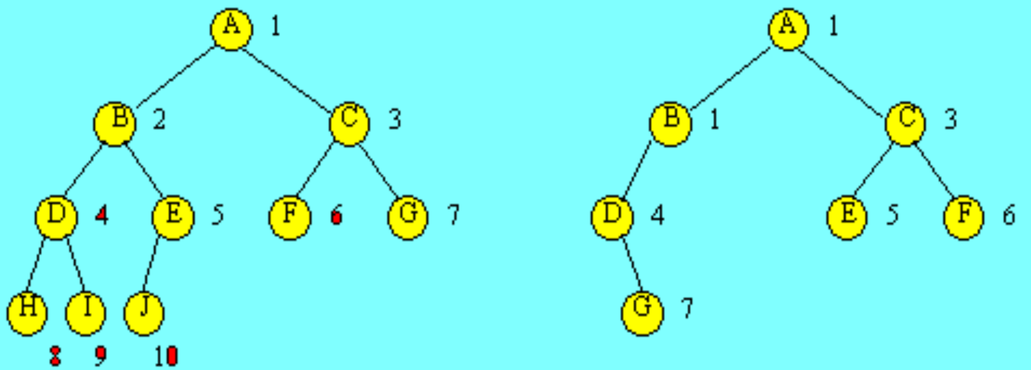
步骤	指针 p	栈 stack 内容	访问结点值
初态	A	空	
1	B	A	A
2	D	A，B	B
3	∧	A，B，D	D
4	G	A，B	
5	∧	A，B，G	G
6	∧	A，B	
7	∧	A	
8	C	空	
9	E	C	C
10	∧	C，E	E
11	∧	C	
12	F	空	
13	∧	F	F
14	∧	空	

[中序遍历算法]

```
void Inorder (BiTree bt)
{
    if (T)
   {
     Inorder(T->lchild);
     visit(T);
     Inorder(T->rchild);
   }
}
```

[后序遍历算法]

```
void  Postorder(BiTree T,
void(*visit)( BiTree ))
{
if (T)
{       Postorder(T->lchild);
        Postorder(T->rchild);
        visit(T);
}
}
```



(a) 一棵完全二叉树

(b) 一棵非完全二叉树

完全二叉树和非完全二叉树示意图

…

InOrder(bt)

…

InOrder(&A)：

1： InOrder(&B)

11： Visit(A);

12： InOrder(&C)

InOrder(&B)：

2： InOrder(&D)

9： Visit(B)

10： InOrder(Φ)

InOrder(&D)：

3： InOrder(Φ)

4： Visit(D)

5： InOrder(&G)

InOrder(&G)：

6： InOrder(Φ)

7： Visit(G)

8： InOrder(Φ)

InOrder(&C)：

13： InOrder(&E)

17： Visit(C);

18： InOrder(&F)

InOrder(&E)：

14： InOrder(Φ);

15： Visit(E);

16： InOrder(Φ)

InOrder(&F)：

19： InOrder(Φ);

20： Visit(F);

21： InOrder(Φ)

[按层次遍历算法]

层次遍历二叉树

```
 void LevelOrder(BiTree bt)
   {BiTNode *Queue[MAXNODE];
    int front,rear;
    if (bt==NULL) return;
    front=-1;
    rear=0;
    queue[rear]=bt;
    while(front!=rear)                          /*队列非空时*/
       { front++;                          /*出队列*/
          Visit (queue[front]);       /*访问队首结点的数据域*/
          if (queue[front]->lchild!=NULL)      /*将队首结点的左孩子结点
入队列*/
           { rear++;
              queue[rear]=queue[front] ->lchild;
           }
          if (queue[front]->rchild!=NULL)      /*将队首结点的右孩子结点
```

对用二叉链表表示的二叉树，设计一个算法，求其后序遍历的第一个结点。

```
typedef   struct   node {
etype   data;
node   *left,  *right;
} node,   *bitptr;

bitptr  firstnode( bitptr  root )
```

```
bitptr  firstnode( bitptr  root )
{
 if  ( ! root )  return  NULL;        // 空树不存在此结点
 while  ( root->left  ||  root->right )   // 尚未找到叶子结点
 {
    if  ( root->left )
        root = root->left;
       // 若有左子树，则沿左分枝向下查找最左下的叶子结点
   else
        root = root->right;
        // 若无左子树，则沿右分枝向下查找最左下的叶子结点
 }
 return  root;                    // 返回后序遍历的第一个结点
}
```

算法思想：

（1）若树根有左子树，则树根的左子树上最左下的叶子结点即为所求；

（2）若树根无左子树，仅有右子树，则树根右子树上最左下的叶子结点即为所求。

1. 设树T的度为4，其中度为1，2，3和4的结点个数分别为4，2，1，1 则T中的叶子数为 D（ ）

A．5 B．6 C．7 D．8

D

2.在下述结论中，正确的是（ ）①只有一个结点的二叉树的度为0; ②二叉树的度为2； ③二叉树的左右子树可任意交换; ④深度为K的完全二叉树的结点个数小于或等于深度相同的满二叉树。

A．①②③ B．②③④ C．②④ D．①④

3. 若一棵二叉树具有10个度为2的结点，5个度为1的结点，则度为0的结点个数是（B）

A. 9　　B. 11　　C. 15　　D. 不确定

4. 在一棵三元树中度为3的结点数为2个，度为2的结点数为1个，度为1的结点数为2个，则度为0的结点数为（ C ）个

A. 4　　B. 5　　C. 6　　D. 7

B

5. 具有10个叶结点的二叉树中有（ ）个度为2的结点

A. 8　　B. 9　　C. 10　　D. ll

6. 一棵完全二叉树上有1001个结点，其中叶子结点的个数是（E）

A. 250 B. 500 C. 254 D. 505 E. 以上答案都不对

7. 一棵二叉树高度为h,所有结点的度或为0，或为2，则这棵二叉树最少有(B)结点

A. 2h B. 2h-1 C. 2h+1 D. h+1

8. 一棵具有 n个结点的完全二叉树的树高度（深度）是（ A ）

A. ⌊log2n⌋+1 B. log2n+1 C. ⌊log2n⌋

9. 深度为h的满m叉树的第k层有（A）个结点。(1=<k=<h)

A． mk-1　　B． mk-1　　C． mh-1　　D． mh-1

10. 一棵二叉树的前序遍历序列为ABCDEFG，它的中序遍历序列可能是（ B ）

A． CABDEFG　　B． ABCDEFG　　C． DACEFBG　　D． ADCFEG

11. 已知一棵二叉树的前序遍历结果为ABCDEF,中序遍历结果为CBAEDF,则后序遍历的结果为（ A ）。

A． CBEFDA　　B． FEDCBA　　C． CBEDFA　　D． 不

具有n个结点，度为m的树，各结点的度之和为 n-1________

在顺序存储(按完全二叉树存储)的二叉树中，存储编号为i和j的两个结点处在同一层的条件是 $\lfloor \log_2 i \rfloor == \lfloor \log_2 j \rfloor$________。

已知二叉树有50个叶子结点，则该二叉树的总结点数至少是 99________。

以下程序是二叉链表树中序遍历的非递归算法，请填空使之完善。二叉树链表的结点类型的定义如下：

```
typedef struct node   /*C语言/
  {char data; struct  node *lchild,*rchild;}*bitree;

void vst(bitree bt)                    /*bt为根结点的指针*/
{
     bitree  p; p=bt;  initstack(s);    /*初始化栈s为空栈*/
     while(p || !empty(s))             /*栈s不为空*/
       if(p)
      { push (s,p); (1)____;}     /*P入栈*/
       else
      { p=pop(s); printf("%c",p->data); (2)_____;}  /*栈顶元素出栈*/
}
```

(1) p=p->lchild // 沿左子树向下

(2) p=p->rchild

试找出满足下列条件的二叉树

1）先序序列与后序序列相同 2）中序序列与后序序列相同

3）先序序列与中序序列相同 4）中序序列与层次遍历序列相同

解答：

若先序序列与后序序列相同，则或为空树，或为只有根结点的二叉树

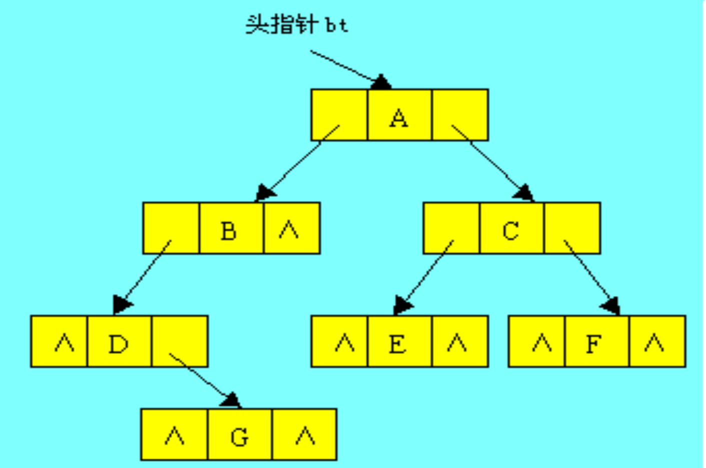
若中序序列与后序序列相同，则或为空树，或为任一结点至多只有左子树的二叉树.

若先序序列与中序序列相同，则或为空树，或为任一结点至多只有右子树的二叉树.

若中序序列与层次遍历序列相同，则或为空树，或为任一结点至多只有右子树的二叉树

构建一棵二叉树的二叉链表也是基于遍历的过程进行的。这里按照先序遍历的过程构建。

首先建立二叉树带空指针的先序次序，依此作为构建时结点的输入顺序，如对于下图所示的二叉树，输入序列为：ABD0G000CE00F00（0表示空，为了简化问题，设数据元素的类型为字符型）。



(a) 不带头结点的二叉链表

建立二叉树的二叉链表

```
void CreateBinTree(BinTree *T)
 {/*以先序遍历序列构造二叉链表存储的二叉树*/
   char ch;
   scanf("%c",&ch);
   if (ch=='0')
     *T=NULL;                           /*读入0时，将相应结点置空*/
   else
        { *T=(BinTNode*)malloc(sizeof(BinTNode));
                                   /*生成结点空间*/
          *T->data=ch;
          CreateBinTree(&(*T->lchild));   /*构造二叉树的左子树*/
          CreateBinTree(&(*T->rchild));   /*构造二叉树的右子树*/
        }
   }
```

[在二叉树中查找结点值为x的结点]

```
void  pre_find(BiTree bt, ElemType x, BiTree &q)//用q返回
 { //F是全局bool型变量，初值为FALSE

    if (bt && F==FALSE)
     {
         if (bt->data==x)
         {   q=bt; F=TRUE;}
          else
          {  pre_find(bt->lchild, x, q);
              pre_find(bt->rchild, x, q)
           }
      }
```

```
{……
 q=NULL;
 bool F=FALSE;
  pre_find(bt,x,q);
  ……
}
```

[求二叉树中每个结点所处的层次]

```
void  pre_level(BiTree p, int level)

 {
      if (p)
      {
          write(p->data, level);//实现时可用printf代替
          pre_level(p->lchild; level+1);
          pre_level(p->rchild; level+1);
       }
 }
```

```
{……
  pre_level(bt, 1);
  ……
}
```

[求二叉树的高度]

```
void  pre_height(BiTree p, int level)
{     if (p)
      {  if (h<level)   h=level;
          pre_height(p->lchild, level+1);
          pre_height(p->rchild,level+1);
       }
}
```

```
{……
 h=0;
 pre_height(bt, 1);
 ……
}
```

```
void  post_height(BiTree bt,int &h)
{     if (bt==NULL) h=0
      else
      { post_height(bt->lchild, h1);
        post_height(bt->rchild,h2);
        h:=1+max(h1,h2);
      }
}
```

```
{……
post_height(bt, h);
  ……
}
```

[复制一颗二叉树]

```
void  pre_copy(BiTree bt; BiTree &q)
{
   if (bt)
    {     new(q);
          q->data= bt->data;
           pre_copy(bt->lchild, q->lchild);
           pre_copy(bt->rchild, q->rchild);
     }
    else
         q=NULL;
}
```

```
{……
 pre_copy(bt, q);
 ……
}
```

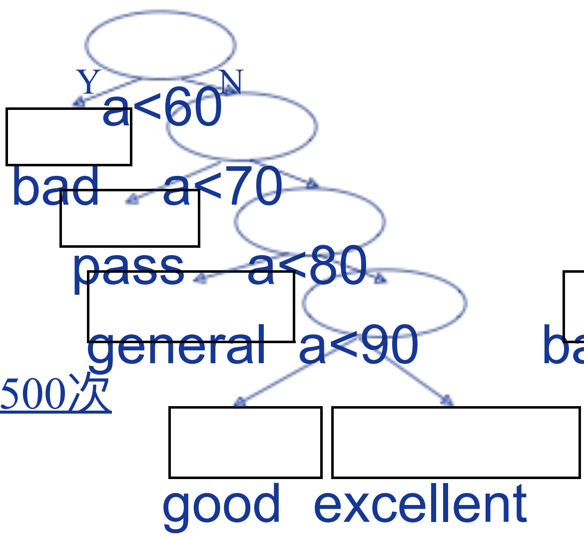
已知一棵二叉树按顺序方式存储在数组A[0..n]中（A[0]不存放结点信息），设计算法求下标分别为i和j的两个结点的最近公共祖先结点。

```
int forefather( eletype A[],  int  i,  int  j, int n )
{
  while  ( i != j )
      if  ( i > j )
          i = i / 2;  // 下标为i的结点的双亲结点的下标
      else
          j = j / 2;      // 下标为j的结点的双亲结点的下标
  return  i;              //返回最近祖先结点的下标
}
```

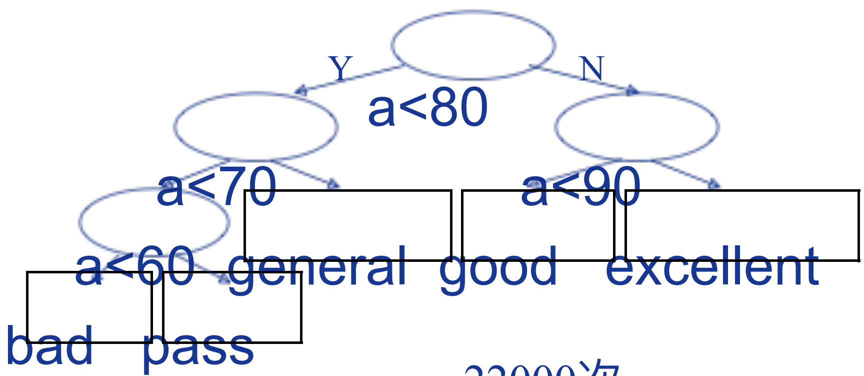
哈夫曼树（最优二叉树）的概念

[例] 编制将百分制转换为五级分制的程序(10000数据)

分数	0—59	60—69	70—79	80—89	90—100
比例	0.05	0.15	0.40	0.30	0.10



31500次



22000次

树的路径长度 从树根到每一个结点的路径上的分支数。

带权路径长度 结点的路径长度与该结点的权之积。

树的带权路径长度 树中所有叶子结点的带权路径长度之和。

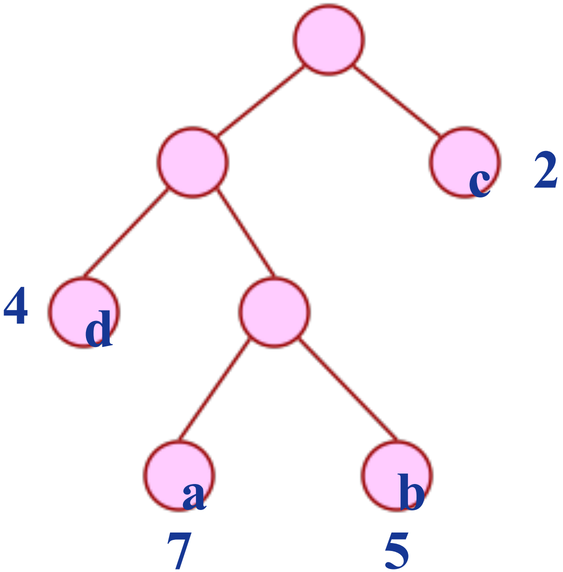
[illegible] $$wpl = \sum_{k=1}^{n} w_k l_k$$

[illegible] w_k — [illegible]

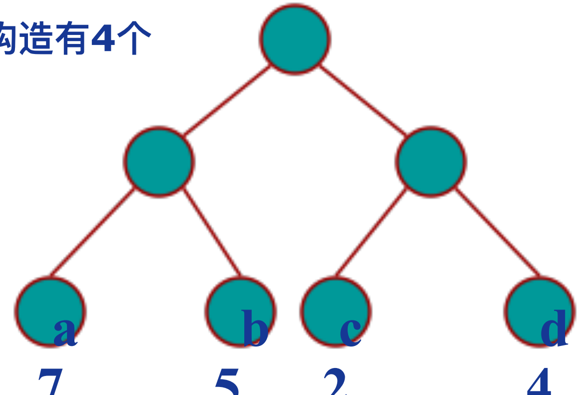
l_k — [illegible]

最优二叉树(哈夫曼树) 带权路径长度WPL最小的二叉树。

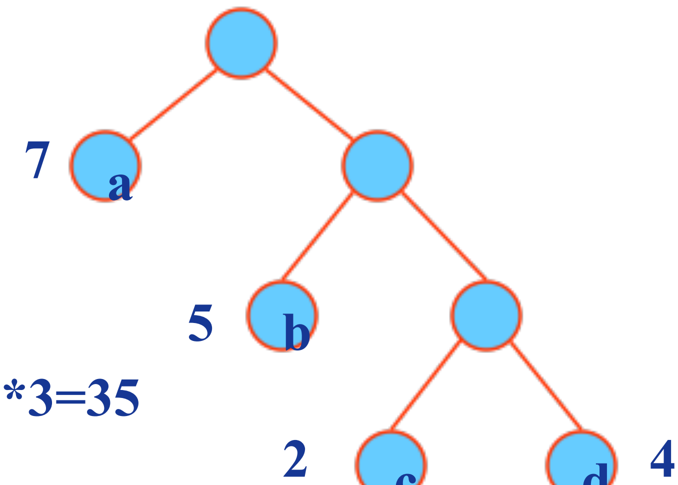
[例] 有4个结点，权值分别为7，5，2，4，构造有4个叶子结点的二叉树



WPL=7*3+5*3+2*1+4*2=46



WPL=7*2+5*2+2*2+4*2=36



WPL=7*1+5*2+2*3+4*3=35

[建立哈夫曼树（最优二叉树）的方法]

[基本思想]

使权大的结点靠近根。

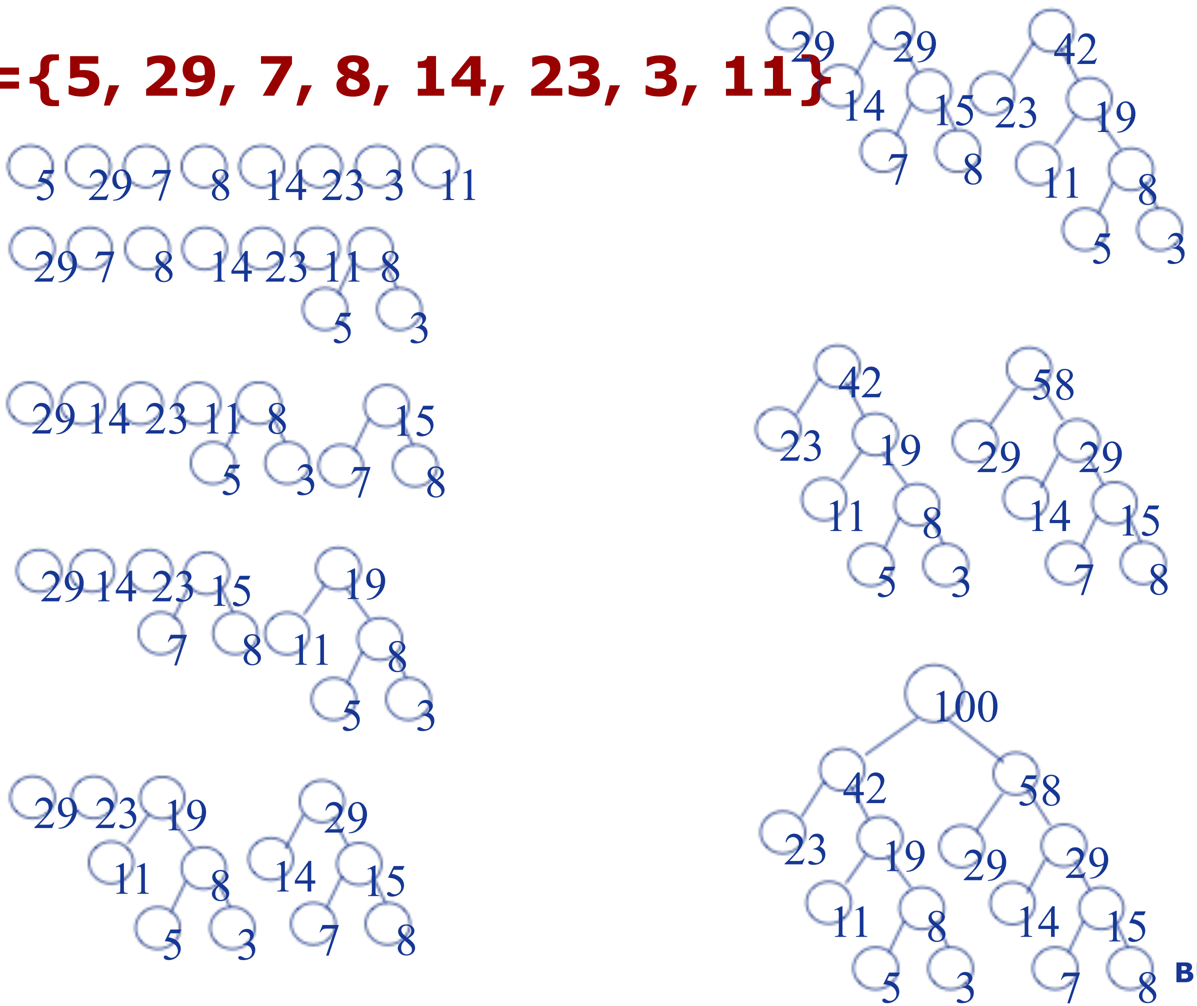
(1)将给定权值从小到大排序成{w1,w2,…,wm}，生成一个森林F={T1,T2,…,Tm}，其中Ti是一个带权Wi的根结点，它的左右子树均空。

(2)把F中根的权值最小的两棵二叉树T1和T2合并成一棵新的二叉树T： T的左子树是T1，右子树是T2，T的根的权值是T1、T2树根结点权值之和。

(3)将T按权值大小加入F中，同时从F中删去T1和T2

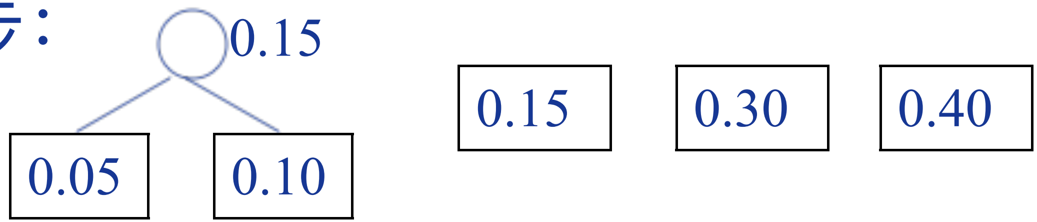
(4)重复(2)和(3)，直到F中只含一棵树为止，该树即为所求。

[例] **w={5, 29, 7, 8, 14, 23, 3, 11}**

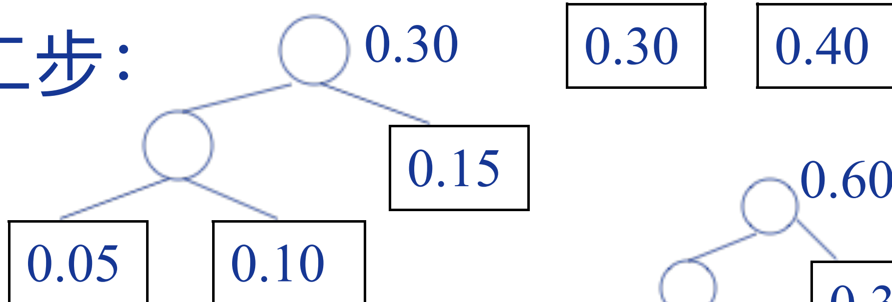


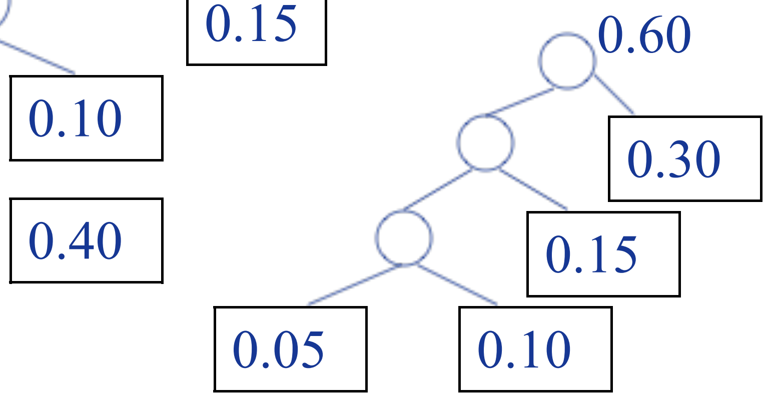

0.05
0.10
0.15
0.30
0.40

第一步：


0.15
0.05
0.10
0.15
0.30
0.40

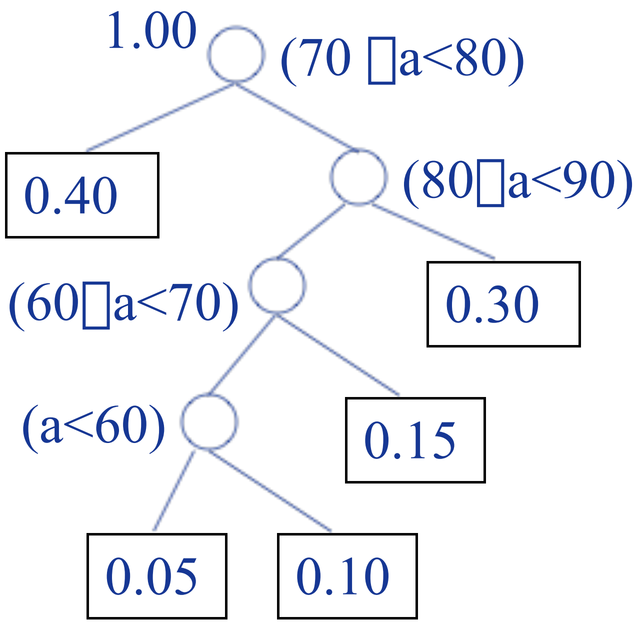
第二步：


0.30
0.30
0.40
0.15
0.05
0.10


0.60
0.30
0.40
0.15
0.05
0.10

第三步：

第四步：



<u>20500次</u> （10000数据）

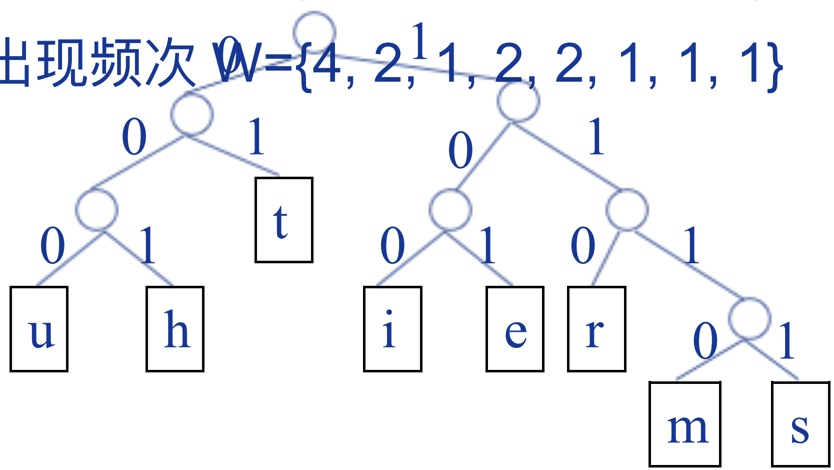
[哈夫曼树的应用]

- 最佳判定树
- 哈夫曼编码：用于通信和数据传送中字符的二进制编码，可以使电文编码总长度最短。

[例] 对time tries truth哈夫曼编码

字符集 D={t, i, m, e, r, s, u, h}

出现频次 W={4, 2, 1, 2, 2, 1, 1, 1}



t	01
i	100
m	1110
e	101
r	110
s	1111
u	000
h	001

- 哈夫曼编码是不等长编码
- 哈夫曼编码是前缀编码，即任一字符的编码都不是另一字符编码的前缀
- 哈夫曼编码树中没有度为1的结点。若叶子结点的个数为n，则哈夫曼编码树的结点总数为 2n-1
- 发送过程：根据由哈夫曼树得到的编码表送出字符数据
- 接收过程：按左0、右1的规定，从根结点走到一个叶结点，完成一个字符的译码。反复此过程，直到接收数据结束

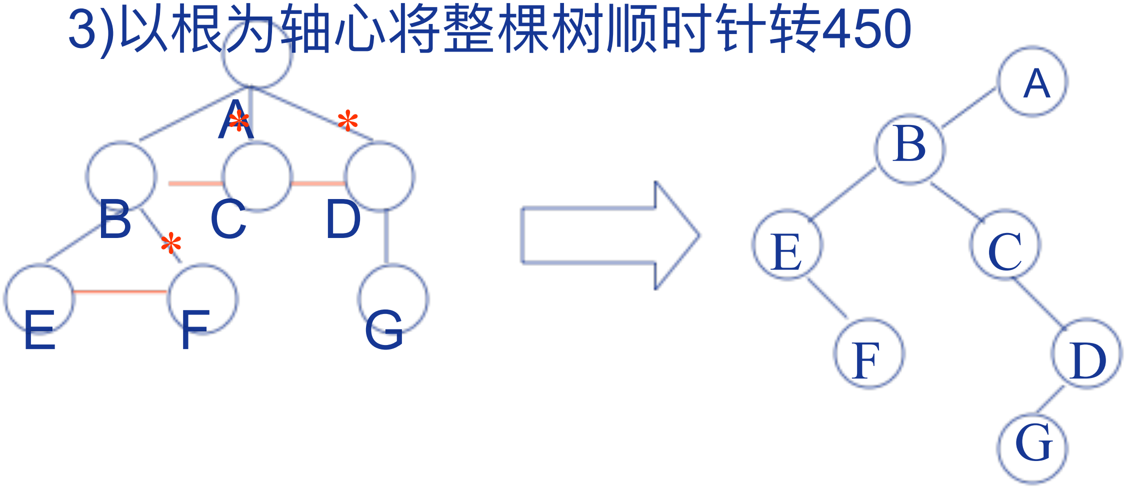
森林与二叉树的转换

[树转换为二叉树]

1)在兄弟间加一连线

2)对每一结点，去掉它与孩子的连线(最左子除外)

3)以根为轴心将整棵树顺时针转450



特点：

无右子树

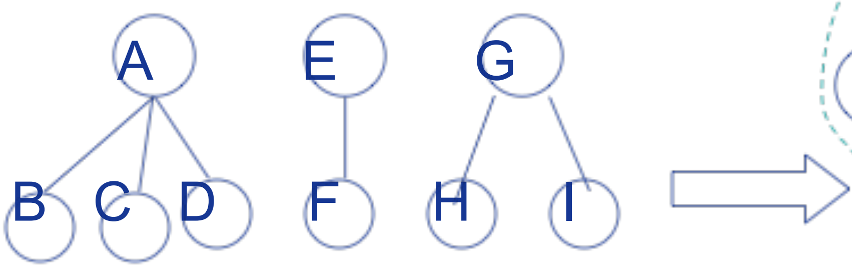
左支是孩子

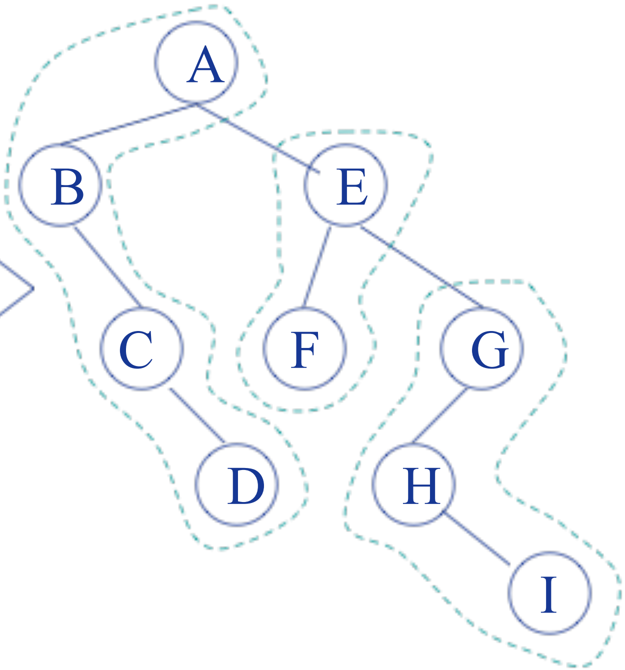
右支是兄弟

[森林转换为二叉树]

1)先将森林里的每一棵树转换成一棵二叉树

2)从最后一棵树开始，把后一棵树的作为前一棵树的根的右子

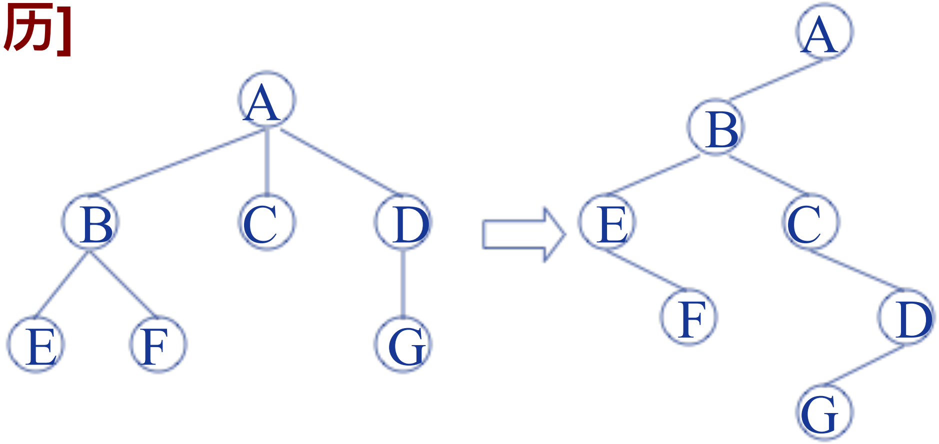




[二叉树转换为树/森林]

逆向过程

[树的遍历]



- **先序遍历** 先访问树的根结点，然后依次先根遍历根的每棵子树 ABEFCDG(**二叉树先序**)
- **后序遍历** 先依次后根遍历根的每棵子树，然后访问树的根结点 EFBCGDA(**二叉树中序**)

[森林的遍历]

- **先序遍历**

  访问第一棵树的根结点；

  先序遍历第一棵树的根的子树森林；

  先序遍历除第一棵树外剩余的树构成的森林

  (逐棵先序遍历每棵子树/对应二叉树的**先序遍历**)

- **中序遍历**

  中序遍历第一棵树的根的子树森林；

  访问第一棵树的根结点；

  中序遍历除第一棵树外剩余的树构成的森林

  (逐棵后序遍历每棵子树/对应二叉树的**中序遍历**)

本 章 作 业

1. 设某二叉树的先序遍历序列为:ABCDEFGGI，中序遍历序列为:BCAEDGHFI：

（1）试画出该二叉树;

（2）将（1）所得的二叉树转化为对应的树或森林;

2. 试编写算法，判断两颗棵以二叉链表表示的二叉树p和q是否相似。

bool Similar(bitptr p, bitptr q)//若相似函数返回TRUE，否则为FLASE

3.试编写算法，统计二叉树bt中叶子结点的个数

int CountLeaf(bitptr bt)

2.3题数据结构如下：

```
typedef struct node
{
   elemtype data ;
   struct node *lchild, *rchild;
}btnode, *bitptr ;
```

4.设T是一棵二叉树，除叶子结点外，其它结点的度数皆为2，若 T中有6个叶结点，试问：

(1) T树的最大深度Kmax=?最小可能深度Kmin=?

(2) T树中共有多少非叶结点?

(3) 若叶结点的权值分别为1,2,3,4,5,6。请构造一棵哈曼夫树，并计算该哈曼夫树的带权路径长度wpl。