树

▼ 基本术语

分支：根和子树根之间的连线

结点的度：分支个数

树的度：结点度最大值

叶子结点：度为0的

分支结点：度不为0的（包括根）

层次：根的层次为1。若某结点的层次为L,则其子树的根的层次为L+1

树的深度(高度)：结点的最大层次

结点层次：从上往下数,根结点为1

结点高度：从下往上数,叶子结点为1

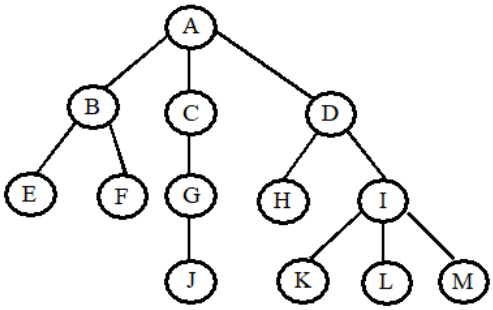
祖先：从根结点到该结点所经分支上的所有结点

森林：m(m>=0)棵互不相交的树的集合 可看成二元组Tree=(root,F）root为根，F为子树森林

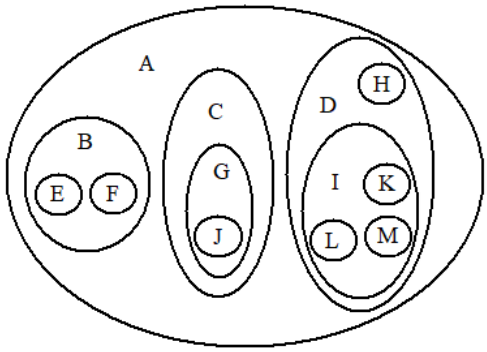
常考性质

- 结点数=总度数+1
- m叉树：每个结点最多有m个孩子（**可以小于**）
- 度为m的树第i层至多有 $m^{(i-1)}$ （i≥1）
- 高度为h的m叉树至多有 $\frac{m^h-1}{m-1}$ 个结点（等比数列求和），至少有h个结点。
- 高度为h、度为m的树至少有 $h+m-1$ 个结点
- 具有n个结点的m叉树的最小高度为 $\lceil \log_m(n(m-1)+1) \rceil$

树的表示



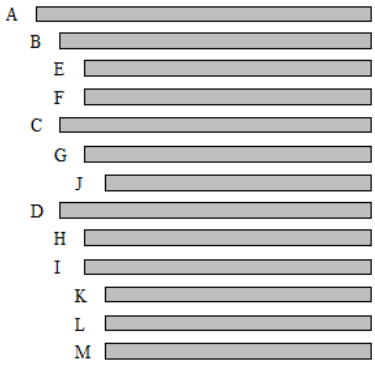
树形表示法



文氏图表示法

A(B(E,F),C(G(J)),D(H,I(K,L,M)))

广义表表示法



凹入表示法

结点计算公式：

$N = n_0 + n_1 + n_2 + n_3 \ldots$

结点总数 $=$ 总度数 $+ 1$

例题

度为3的树，$n_3=100$，$n_2=200$，求叶子个数

总度数+1=节点数：

$n_0 \times 0 + n_1 \times 1 + n_2 \times 2 + n_3 \times 3 + 1 = n_0 + n_1 + n_2 + n_3$

$3 \times 100 + 2 \times 200 + n_1 + 1 = n_0 + n_1 + 100 + 200$

$n_0 = 401$

3. （4 分）已知一棵度为 m 的树中有 N_1 个度为 1 的结点，N_2 个度为 2 的结点，…，N_m 个度为 m 的结点。试问该树中有多少个叶子结点？

总结点数 $= N_1 + 2N_2 + \dots + mN_m + 1 = \sum_{i=1}^{m} iN_i + 1$

$N_0 = \sum_{i=1}^{m} iN_i + 1 - \sum_{i=1}^{m} N_i = 1 + \sum_{i=1}^{m} (i-1)N_i$

▼ 二叉树

二叉树≠度为2的树（作为独生子时，二叉树左右子树有区别）

▼ 常考性质

二叉树

- 第 i 层至多有 $2^{(i-1)}$个结点 （i≥1） #证明：数归
- 深度为 k 的二叉树至多有 2^k-1个结点 （k≥1） # $\sum_{i=1}^{k} 2^{i-1}$
- $n_0 = n_2 + 1$ # $n_0 + n_1 + n_2 = n_1 + 2n_2 + 1$

完全二叉树

- 只有最后两层可能有叶子结点
- 最多只有一个度为1
- 编号方式与满二叉树相同
- $i \le \lfloor n/2 \rfloor$为分支结点，$i > \lfloor n/2 \rfloor$为叶子结点

  $2i \le n$有左孩子， $2i+1 \le n$有右孩子
- 具有n个结点的完全二叉树的深度为$\lfloor \log_2 n \rfloor + 1$
  - $2^{k-1} < n < 2^k - 1$
- 若有 $2k$ 个结点，则 $n_0 = k, n_2 = k-1, n_1 = 1$

  若有 $2k-1$个结点，则 $n_0 = k, n_2 = k-1, n_0 = 0$

  （因为 $n_0 = n_2 + 1$和 $n_1 \le 1$）

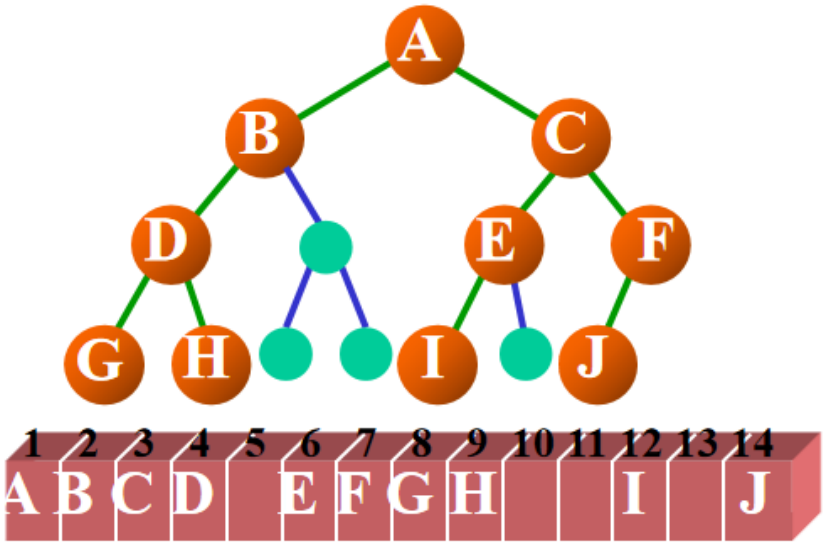
满二叉树（特殊的完全二叉树）

- 第 i 层有 $2^{(i-1)}$个结点

- 深度为 k 的有 2^k-1个结点的二叉树
- 只有最后一层有叶子结点
- 不存在度为1的结点
- 从1开始编号，结点为 i 的左孩子为 $2i$，右孩子为 $2i+1$，父结点为 $\lfloor i/2 \rfloor$

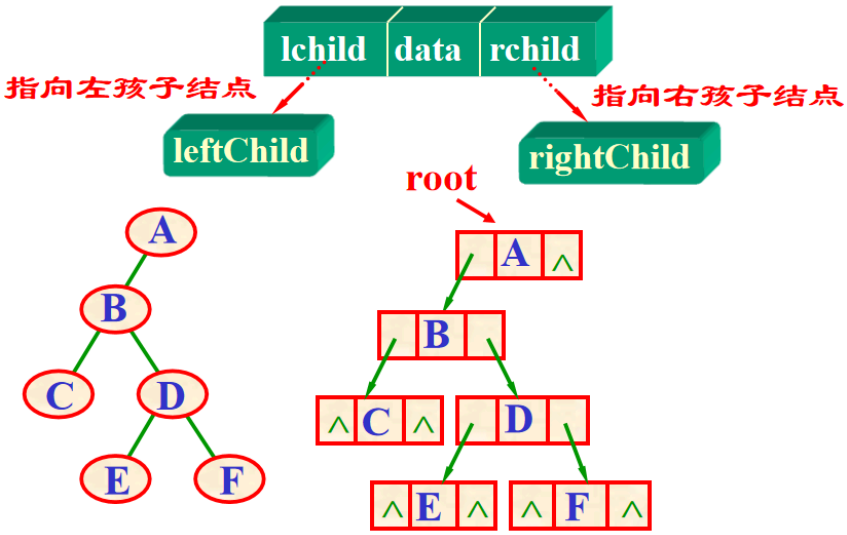
▼ 存储结构

▼ 顺序存储



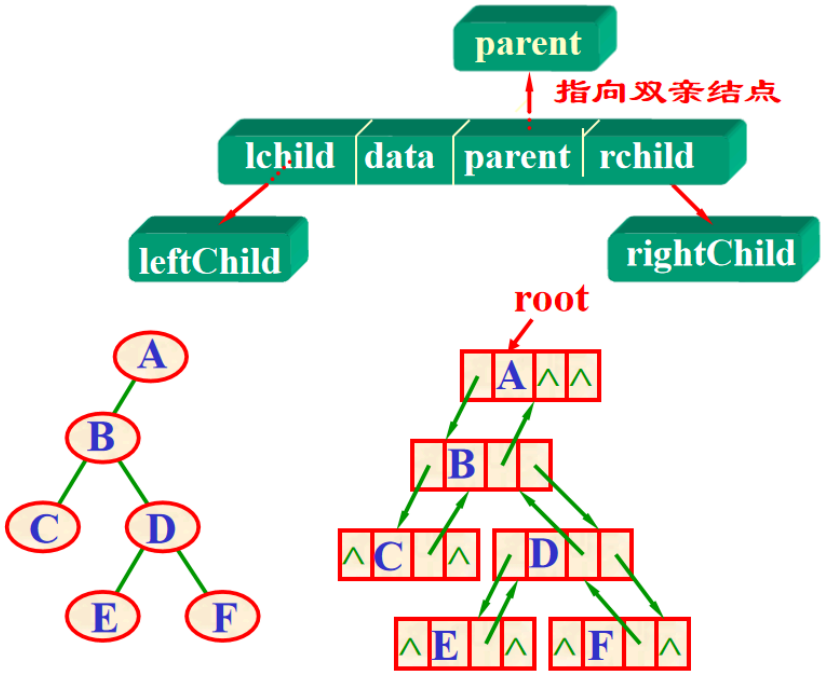
▼ 二叉链表

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

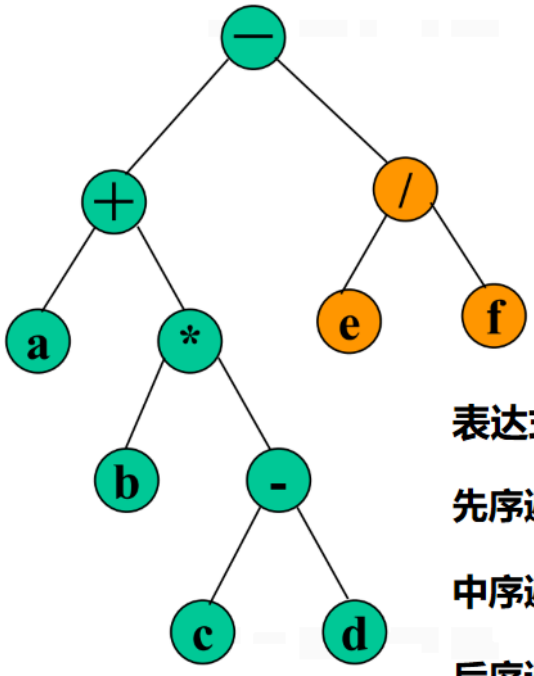


▼ 三叉链表

```
typedef struct TriTNode{
    char data;
    struct TriTNode *lchild,*rchild;
    struct TriTNode *parent;
}*TriTree;
```



▼ 遍历



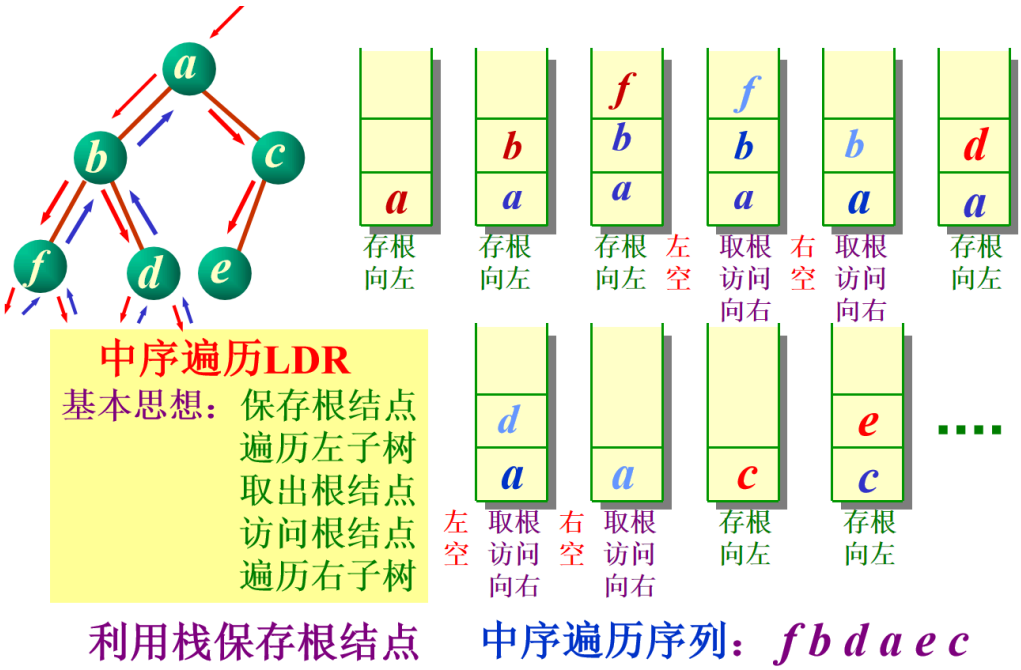
表达式	(a+b*(c-d))-e/f
先序遍历	-+a*b-cd/ef
中序遍历	a+b*c-d-e/f
后序遍历	abcd-*+ef/-

前、中、后序

```
//前序
void preorder(BTNode *t){
   if(t!=NULL){
      visit(t→data);
      preorder(t→lchild);
      preorder(t→rchild);
   }
}
//中序
void inorder(BTNode *t){
   if(t!=NULL){
      inorder(t→lchild);
      visit(t→data);
      inorder(t→rchild);
   }
}
//后序
```

```
void postorder(BTNode *t){
    if(t!=NULL){
        postorder(t→lchild);
        postorder(t→rchild);
        visit(t→data);
    }
}
```

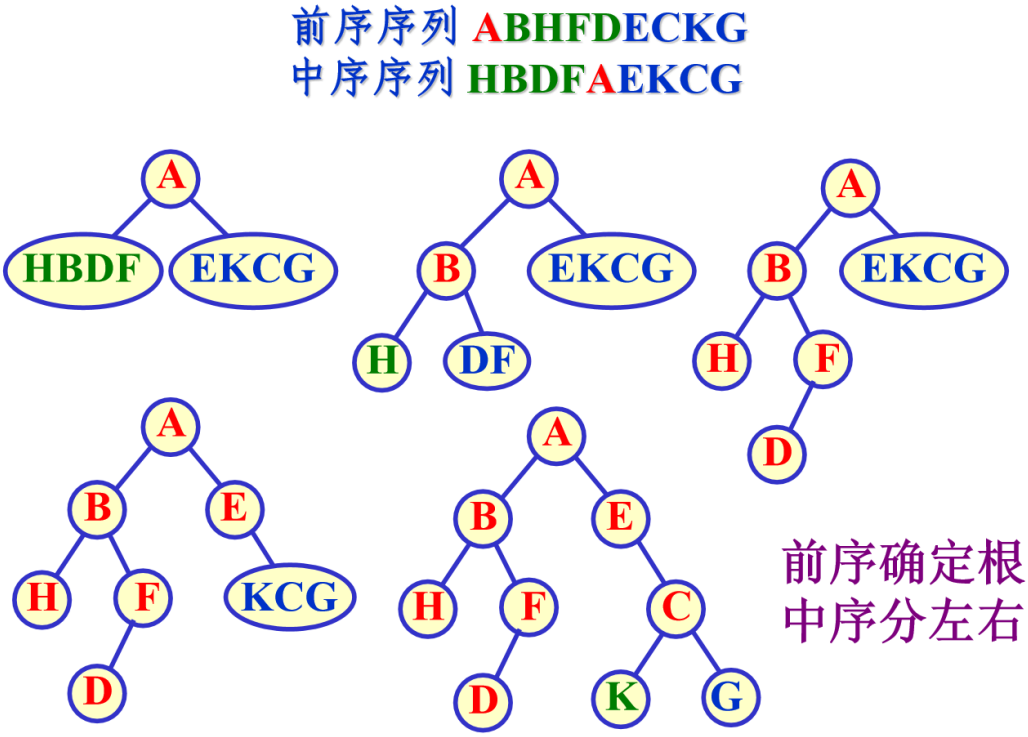
中序非递归



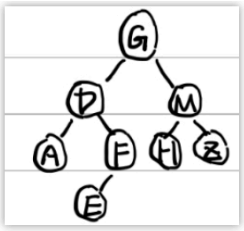

```
//中序非递归
bool InOrderTraverse_Stack(BiTree T,bool(*Visit)(char)){
    SqStack S;
    InitStack(S);
    BiTNode *p=T;
    while(p||!StackEmpty(S)){
        if(p){
            Push(S,p);      //根进栈,遍历左子树
            p=p→lchild;
        }
```

```
        else{
            Pop(S,p);    //根出栈
            if(!Visit(p→data)) return false;  //遍历右子树
            p=p→rchild;
        }
    }
    return true;
}
```

根据前序和中序建树



1. 画出和下列已知遍历序列对应的二叉树 T:



二叉树的先序遍历序列：GDAFEMHZ

二叉树的中序遍历序列：ADEFGHMZ

根据后序和中序建树

```
BiTree BuildTree(int post[],int in[],int n){
    BiTree T;     // 创建二叉树的根节点指针
    int p;
    if(n==0) return NULL;
    T=(BiTree)malloc(sizeof(BiTNode));
    T→data=post[n-1];      // 后序遍历的最后一个元素是根节点
    T→lchild=T→rchild=NULL;
    for(p=0;p<n;p++){   // 在中序遍历中找到根节点的位置
        if(in[p]==post[n-1]) break;
    }
    T→lchild=BuildTree(post,in,p);
    T→rchild=BuildTree(post+p,in+p+1,n-p-1);
    return T;
}
```

层序遍历

```
'''
队头：当前访问的结点
队尾：保存当前访问的孩子结点
操作过程：
    取出队头作为当前结点
    访问当前结点
    若有左孩子，则入队列
    若有右孩子，则入队列
'''
void LevelOrder(BiTree T){
    SqQueue Q;
    InitQueue(Q);
    BiTree p;
    EnQueue(Q,T);
    while(!QueueEmpty(Q)){
        DeQueue(Q,p);
        PrintElem(p→data);
```

```
        if(p→lchild!=NULL) EnQueue(Q,p→lchild);
        if(p→rchild!=NULL) EnQueue(Q,p→rchild);
    }
}
```

应用

查找指定结点（前序遍历）

```
BTNode *find(BTNode *b,elemtype x){
    if(b==NULL) return NULL;
    if(b→data==x) return b;
    p=find(b→lchild,x);
    if(p==NULL) p=find(b→rchild,x);
    return p;
}
```

计算叶子结点个数（前序遍历）

```
void CountLeaf(BiTree T,int &count){
    if(T){
        if((!T→lchild)&&(!T→rchild)) count++;
        CountLeaf(T→lchild,count);
        CountLeaf(T→rchild,count);
    }
}
```

求二叉树深度（后序遍历）

```
int Depth(BiTree T){
    int depth;
    if(!T) depth=0;
    else{
        int depthleft=Depth(T→lchild);
        int depthright=Depth(T→rchild);
        depth=1+(depthleft>depthright?depthleft:depthright);
```

```
    }
    return depth;
}
```

求结点个数

```
int NodeCount(BiTree T){
    if(!T) return 0;
    else return NodeCount(T→lchild)+NodeCount(T→rchild)+1;
}
```

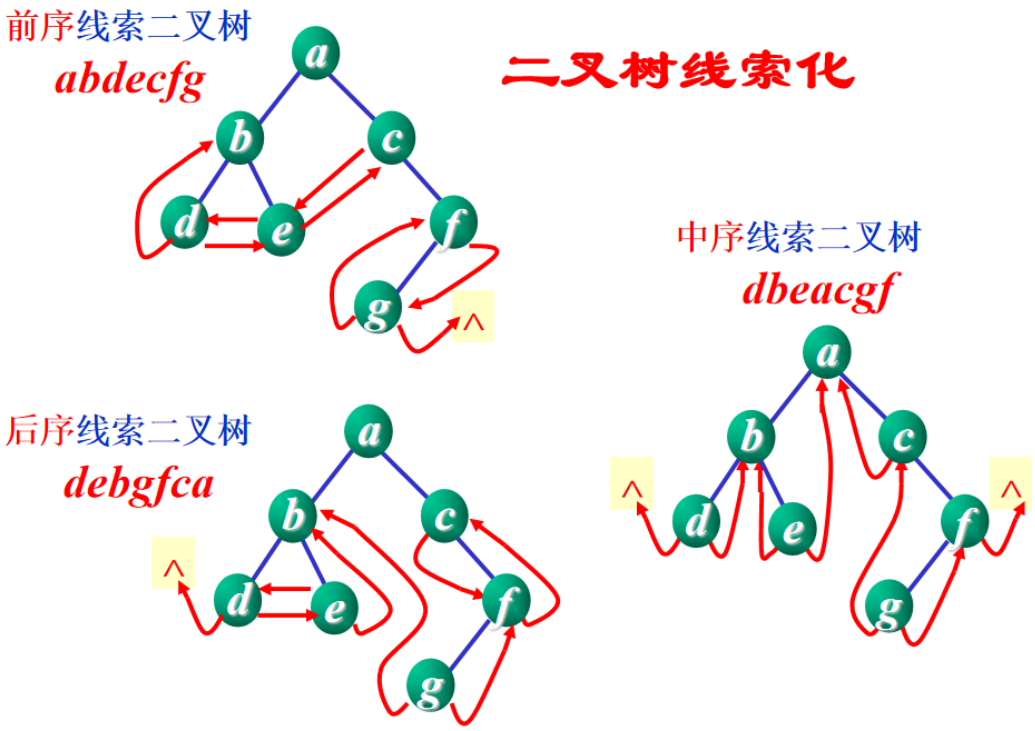
复制二叉树（后序遍历）

```
//生成一个结点
BiTNode *GetTreeNode(char item,BiTNode *lptr,BiTNode *rptr){
    BiTNode *T=(BiTNode*)malloc(sizeof(BiTNode*));
    T→data=item;
    T→lchild=lptr;
    T→rchild=rptr;
    return T;
}
//复制二叉树(后序遍历)
BiTNode *CopyTree(BiTNode *T){
    BiTNode *newlptr,*newrptr;
    if(!T) NULL;
    if(T→lchild) newlptr=CopyTree(T→lchild);
    else newlptr=NULL;
    if(T→rchild) newrptr=CopyTree(T→rchild);
    else newrptr=NULL;
    BiTNode *newnode=GetTreeNode(T→data,newlptr,newrptr);
    return newnode;
}
```

▼ 线索二叉树

```
typedef enum PointerType{
    Link=0,Thread=1
}PointerType;

typedef struct ThreadNode{
    char data;
    struct ThreadNode *lchild,*rchild;
    PointerType ltag,rtag;    //tag=0指孩子,tag=1指线索
}ThreadNode,*ThreadTree;
```



建立线索二叉树

先建立一般的二叉链表再线索化

中序线索化

```
void InThreading(ThreadTree p,ThreadTree &pre){
    if(p){
        InThreading(p→lchild,pre);    //左子树线索化
```

```
        //处理根结点
        if(!p→lchild){
            p→ltag=Thread;
            p→lchild=pre;
        }
        if(pre&&!pre→rchild){    //为前驱结点建立后继线索
            pre→rtag=Thread;
            pre→rchild=p;
        }
        pre=p;
        InThreading(p→rchild,pre);
    }
}
void CreateInThread(ThreadTree T){
    ThreadTree pre=NULL;
    if(T){
        InThreading(T,pre);    //中序线索化二叉树
        if(pre→rchild==NULL){ //处理遍历的最后一个结点
            pre→rtag=Thread;
        }
    }
}
```

先序线索化

```
void PreThreading(ThreadTree p,ThreadTree &pre){
    if(p){
        if(!p→lchild){
            p→lchild=pre;
            p→ltag=Thread;
        }
        if(pre&&!pre→rchild){
            pre→rchild=p;
            pre→rtag=Thread;
        }
        pre=p;
```

```
        //访问左子树时要判断左子树是否为线索  否则会进入死循环
        if(p→ltag==Link){
            PreThreading(p→lchild,pre);
        }
        PreThreading(p→rchild,pre);
    }
}
void CreatePreThread(ThreadTree T){
    ThreadTree pre=NULL;
    if(T){
        PreThreading(T,pre);
        if(!pre→rchild){
            pre→rtag=Thread; //处理最后一个结点
        }
    }
}
```

后序线索化

```
void PostThreading(ThreadTree p,ThreadTree &pre){
    if(p){
        PostThreading(p→lchild,pre);
        PostThreading(p→rchild,pre);
        if(!p→lchild){
            p→lchild=pre;
            p→ltag=Thread;
        }
        if(pre&&!pre→rchild){
            pre→rchild=p;
            pre→rtag=Thread;
        }
        pre=p;
    }
}
void CreatePostThread(ThreadTree T){
    ThreadTree pre=NULL;
```

```
    if(T){
        PostThreading(T,pre);
        if(!pre→rchild){
            pre→rtag=Thread;
        }
    }
}
```

中序遍历

```
//中序找左
ThreadNode *FirstNode(ThreadNode *p){
    //循环找到最左下结点
    while(p→ltag==Link) p=p→lchild;
    return p;
}
//中序后继
ThreadNode *NextNode(ThreadNode *p){
    //右孩子存在，访问右孩子  左根右
    //FirstNode找到右孩子的最左下结点
    if(p→rtag==Link) return FirstNode(p→rchild);
    //右孩子为线索，直接返回线索
    else return p→rchild;
}
//中序遍历线索二叉树
void Inorder(ThreadTree T){
    for(ThreadNode *p=FirstNode(T);p;p=NextNode(p)){
        visitall(p);
    }
    printf("\n");
}
```

▼ 习题

含有 100 个结点的二叉树，若采用二叉链表存储，则各个结点中所有的空指针域共有 101 个。采用三叉链表存储，则各个结点中所有的空指针域共有 102 个。

n个结点，二叉：n+1个空指针；三叉：n+2个空指针

100个结点共200个指针，除根外每个都有一个指针指向它，200-99=101

300个，除根都指父，除根都有指向它的 300-99-99=102

5. 含有 100 个结点的完全二叉树，度为 0，1，2 的结点数量分别为 50 、1 、49 。树的高度为 7 。

$\begin{cases} n_0 = n_2 + 1 \\ n_1 = 1 \text{ or } 0 \end{cases}$

$\lfloor \log_2 n \rfloor + 1 = \lfloor \log_2 100 \rfloor + 1 = 7$

50 → 2^6 7层

7.前序遍历和中序遍历结果相同的二叉树为[B,F]；前序遍历和后序遍历结果相同的二叉树为[B]。

A. 一般二叉树
B. 只有根结点的二叉树
C. 根结点无左孩子的二叉树
D. 根结点无右孩子的二叉树
E. 所有结点只有左子树的二叉树
F. 所有结点只有右子树的二叉树

前=中：根左右=左根右 ⇒ 无左

前=后：根左右=左右根 ⇒ 只有根

中=后：左根右=左右根 ⇒ 无右

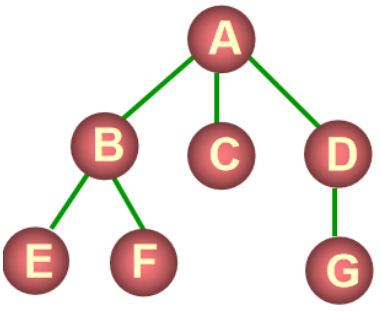
中序=后序

4、某二叉树的前序和后序正好相反，则该二叉树一定是__二叉树。

解析：本题考点是二叉树的特性。由于二叉树的前序序列是先访问根结点再访问左右子树得到的，二叉树的后序序列是先访问左右子树最后访问根结点得到的，因此，高度等于其结点数的二叉树的前序和后序正好相反。

▼ 树的存储结构

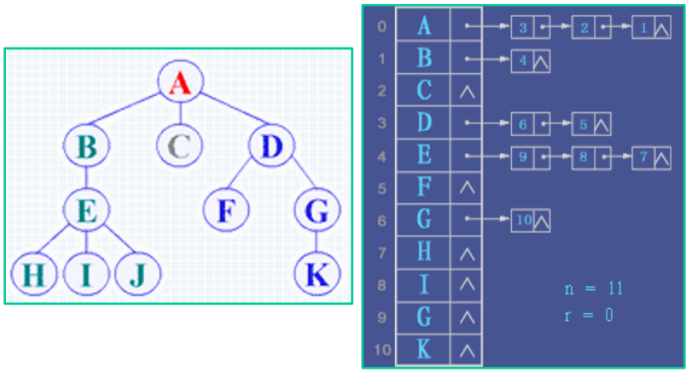
1. 双亲表示法



	0	1	2	3	4	5	6
data	A	B	C	D	E	F	G
parent	−1	0	0	0	1	1	3

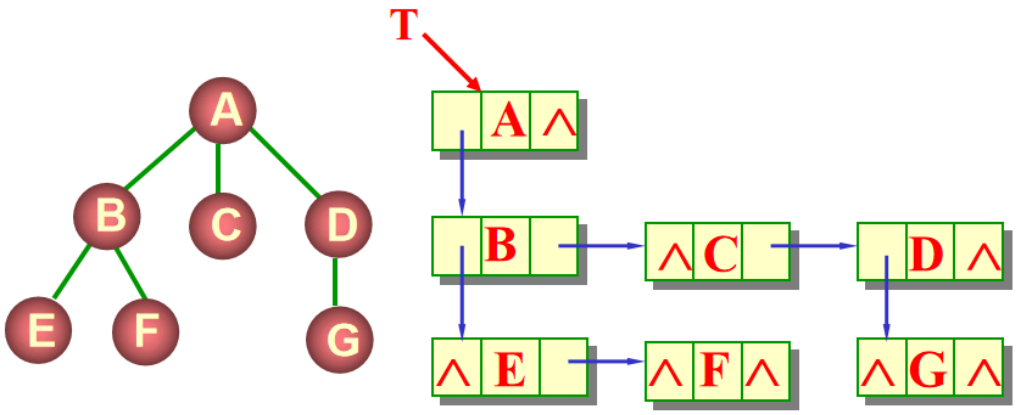
```
typedef struct PTNode{
    char data;
    int parent; //父结点下标
}PTNode;
typedef struct{
    PTNode nodes[MAX_TREE_SIZE];
    int r,n; //根位置和结点数
}PTree;
```

2. 孩子表示法




```
typedef struct CTNode{
    int child;  //孩子结点在数组中的位置
    struct CTNode *next;  //下一个孩子
}*ChildPtr;
typedef struct{
    char data;
    ChildPtr firstchild; //第一个孩子
}CTBox;
typedef struct{
    CTBox nodes[MAX_TREE_SIZE];
    int n,r; //结点数和根位置
}CTree;
```

3. 孩子兄弟表示法




```
typedef struct CSNode{
    char data;
    struct CSNode *firstchild,*nextsibling;
    //左指针指向第一个子结点，右指针指向下一个兄弟结点
}CSNode,*CSTree;
```

▼ 树和森林的遍历

树的遍历

1. 先根遍历（规则同先序遍历）

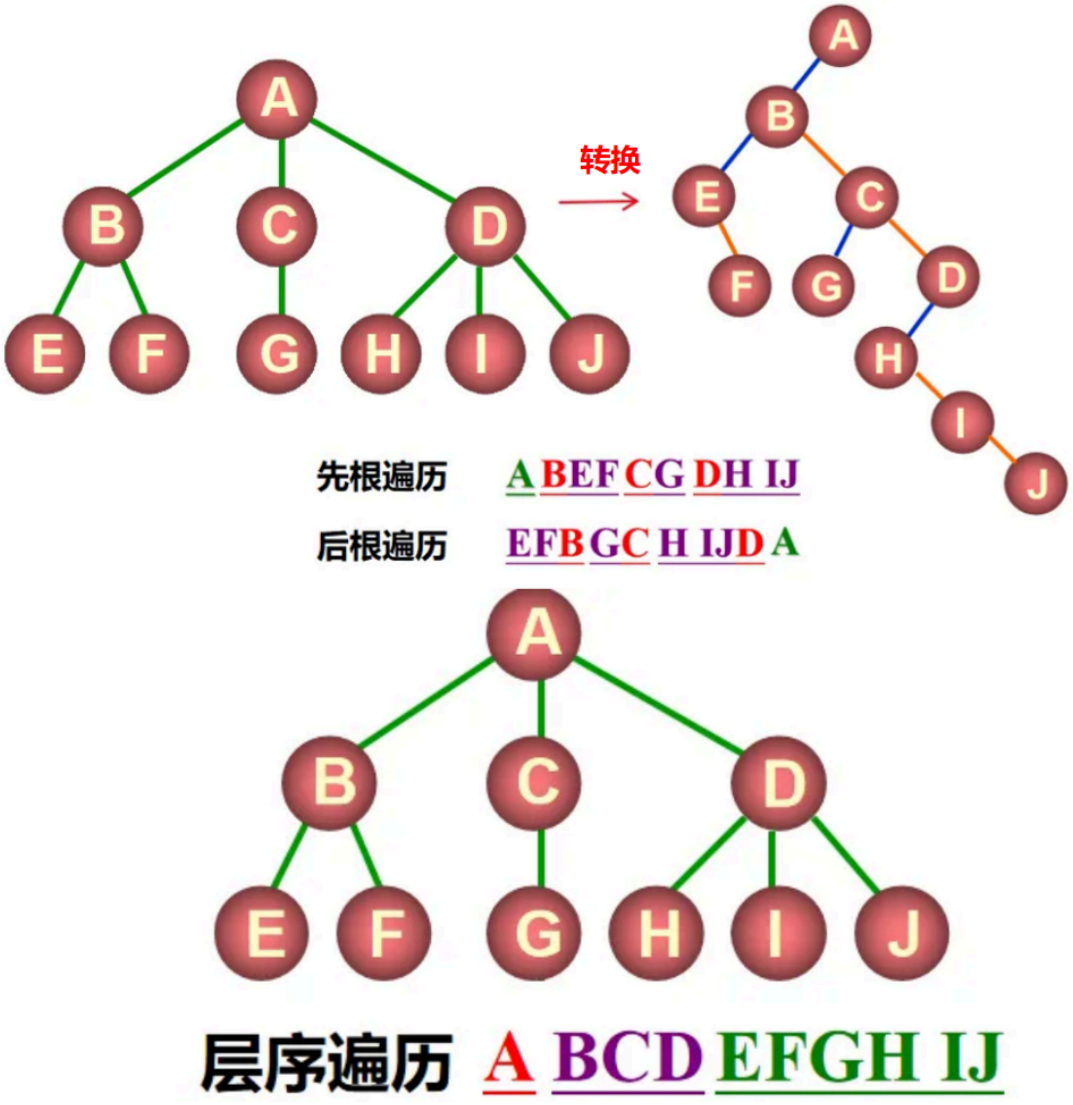
   转化为二叉树后与对其先序遍历序列相同

2. 后根遍历（规则同后序遍历）

   转化为二叉树后与对其中序遍历序列相同

3. 层次遍历（队列）

4. 树和森林与二叉树的转换：用孩子兄弟表示法

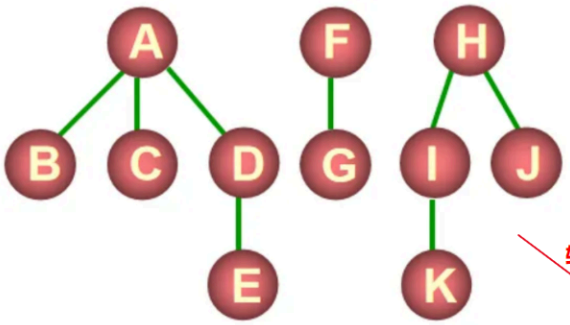


森林的遍历

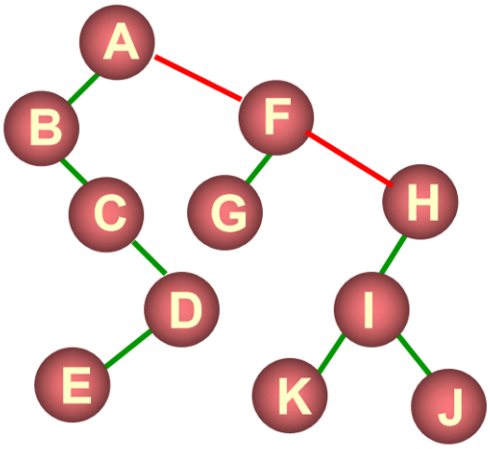
前序遍历（规则同二叉树前序遍历）

中序遍历（规则同二叉树后序遍历）

层序遍历

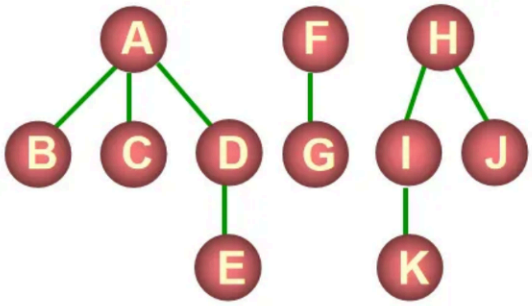


转换



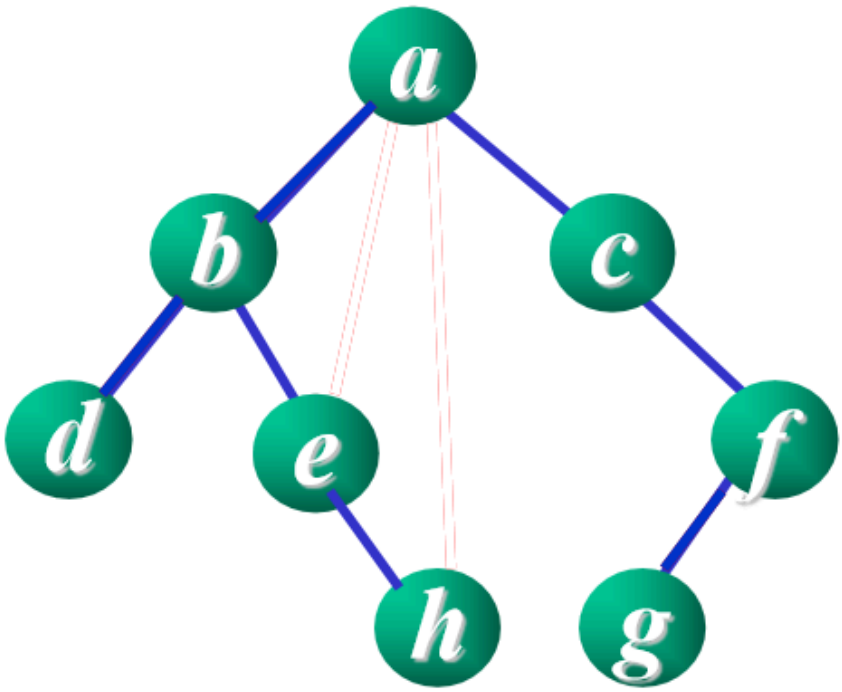
前序遍历 ABCDE FG HIKJ

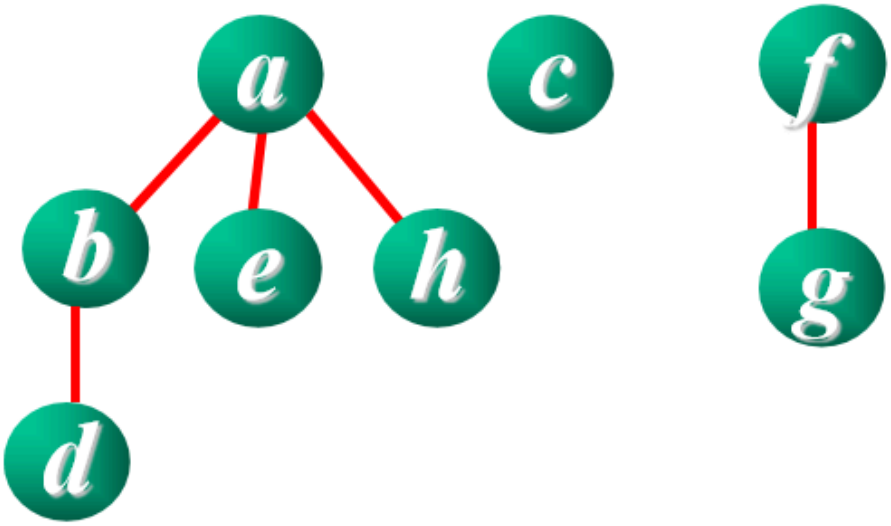
中序遍历 BCEDA GF KIJH



层序遍历 AFH BCDGIJ EK

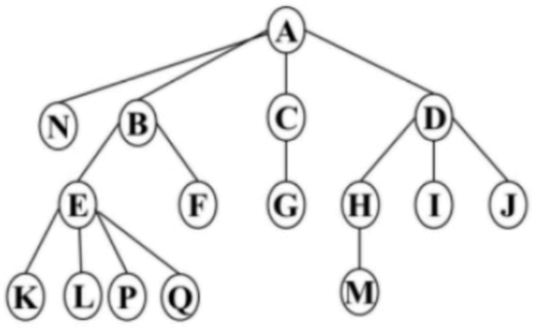
二叉树转换为树或森林



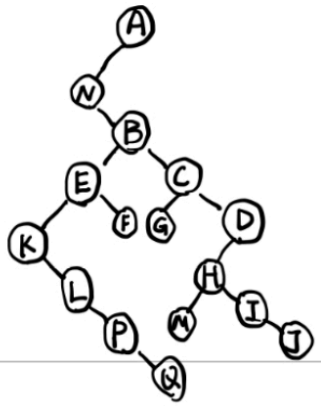


▼ 题

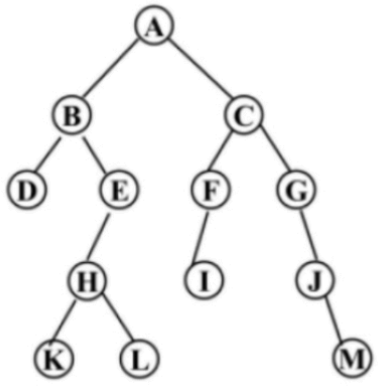
2. 请给出下面的树的后根遍历序列，并画出其对应的二叉树



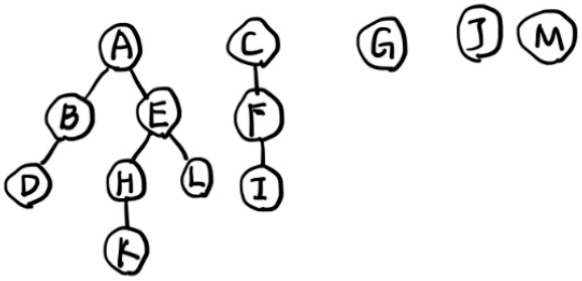
NKLPQEFBGCMHIJDA



3. 请给出下面二叉树的中序遍历序列，并画出对应的森林

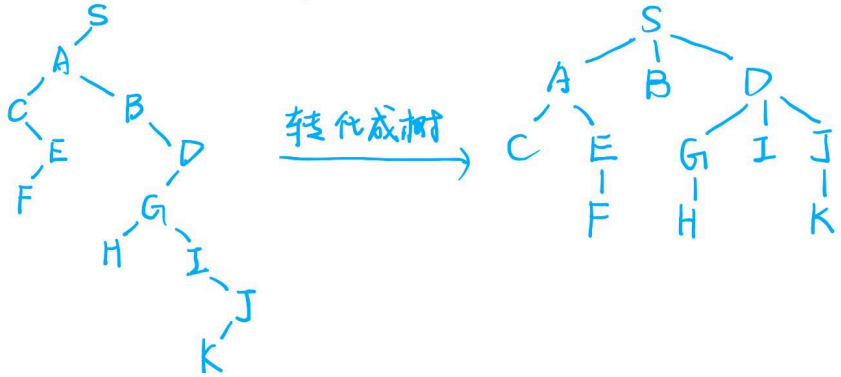


DBKHLEAIFCGJM



1. （4 分）设先序遍历某棵树的结点序列为 SACEFBDGHIJK，后序遍历该树的结点序列为 CFEABHGIKJDS，要求画出这棵树。

将该树转化为二叉树，则该树的后序遍历为二叉树的中序遍历

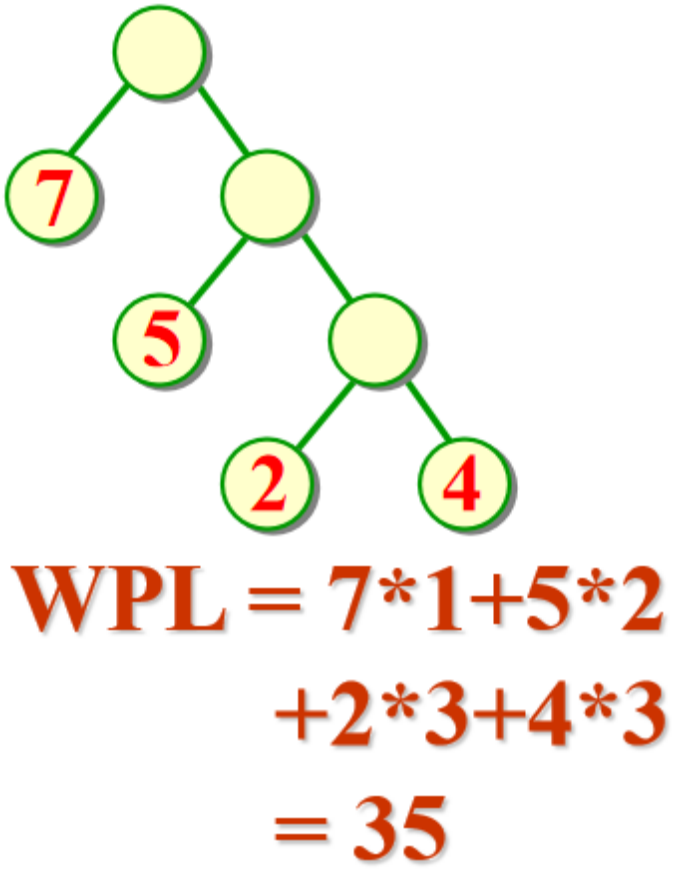


▼ 哈夫曼树

- 带权路径长度:从根到该结点的路径长度与结点权值的乘积
- 树的带权路径长度（WPL）:所有叶子结点的带权路径长度之和

  $WPL = \sum_{k=1}^{n} w_k l_k$
- 不存在度为1的结点
- 哈夫曼树不唯一，但WPL必然相同且最优
- m个叶子，经m-1次合并，产生m-1分支结点，则哈夫曼共有2m-1个结点。

路径长度为所在层数-1

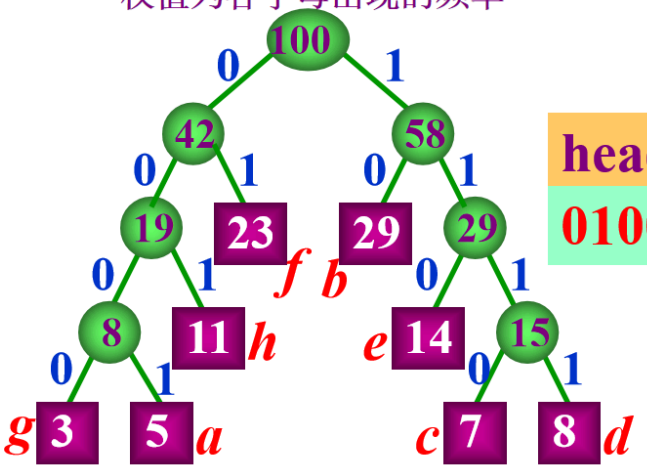


哈夫曼编码

a b c d e f g h

{ 5, 29, 7, 8, 14, 23, 3, 11 }

权值为各字母出现的频率



a :0001　　*e* :110

b :10　　*f* :01

c :1110　　*g* :0000

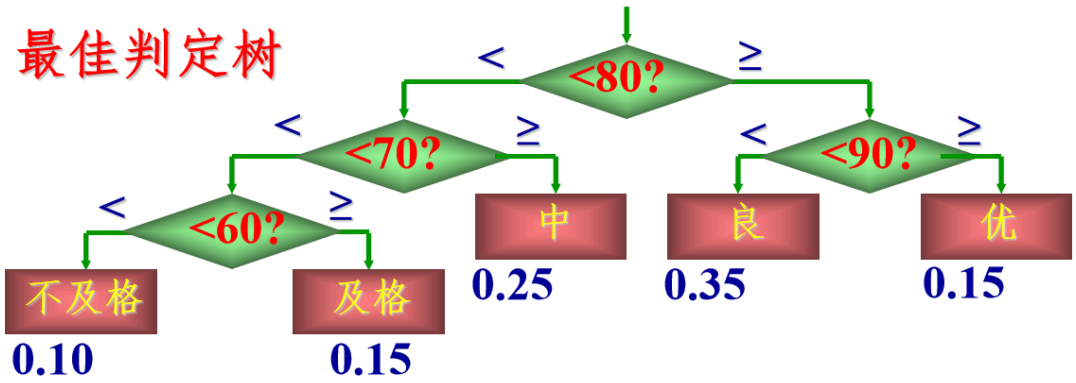
d :1111　　*h* :001

head编码: 001 110 0001 1111

0100011110110译码: *face*

[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

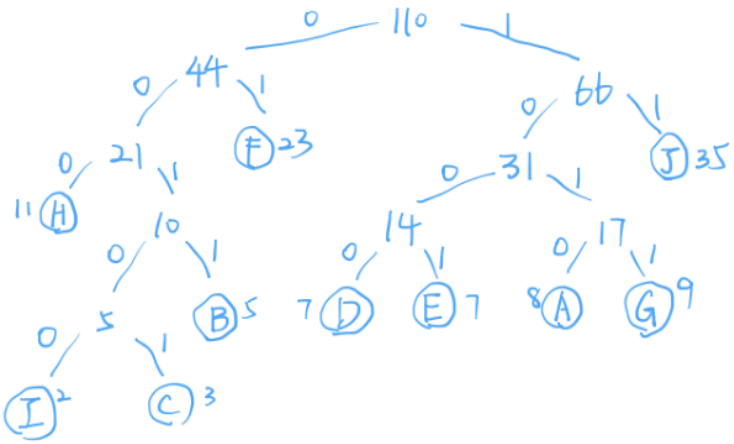
最佳判定树



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

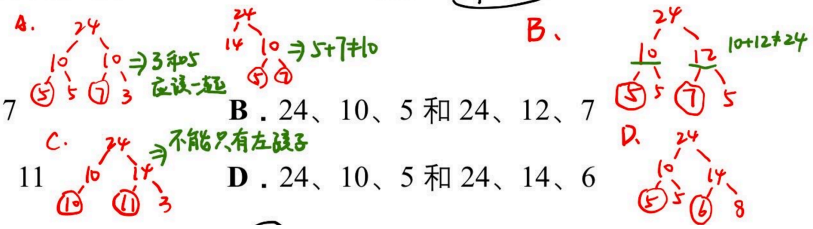
2.（10 分）已知某电文中共出现了 10 种不同的字母，每个字母出现的频率分别为 A:8，B:5，C:3，D:7，E:7，F:23，G:9，H:11，I:2，J:35，现在对这段电文用二进制进行编码（即码字由 0,1 组成），问电文编码总长度**至少**有多少位（5 分）？请画出相应的图（5 分）。



A：1010　　G：1011
B：0011　　H：000
C：00101　　I：00100
D：1000　　J：11
E：1001
F：01

8×4+5×4+3×5+7×4+7×4+23×2+9×4+11×3+2×5+35×2=318

7．下列选项给出的是从根到两个叶子结点路径上的结点权值序列，能属于同一颗哈夫曼树的是（ D ）。



A．24、10、5 和 24、10、7　　**B**．24、10、5 和 24、12、7

C．24、10、10 和 24、14、11　　**D**．24、10、5 和 24、14、6