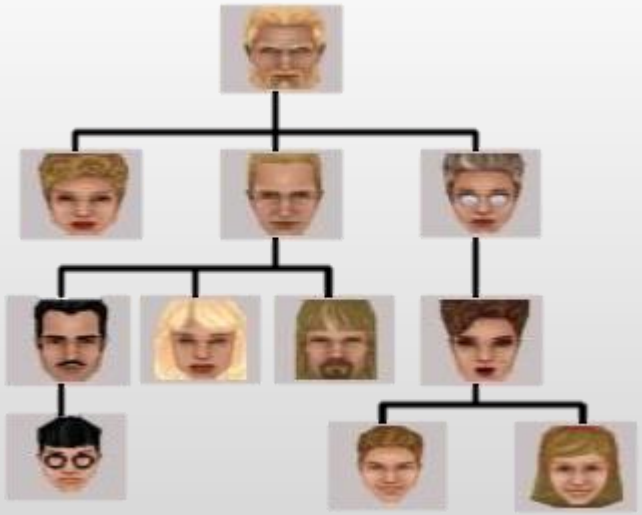
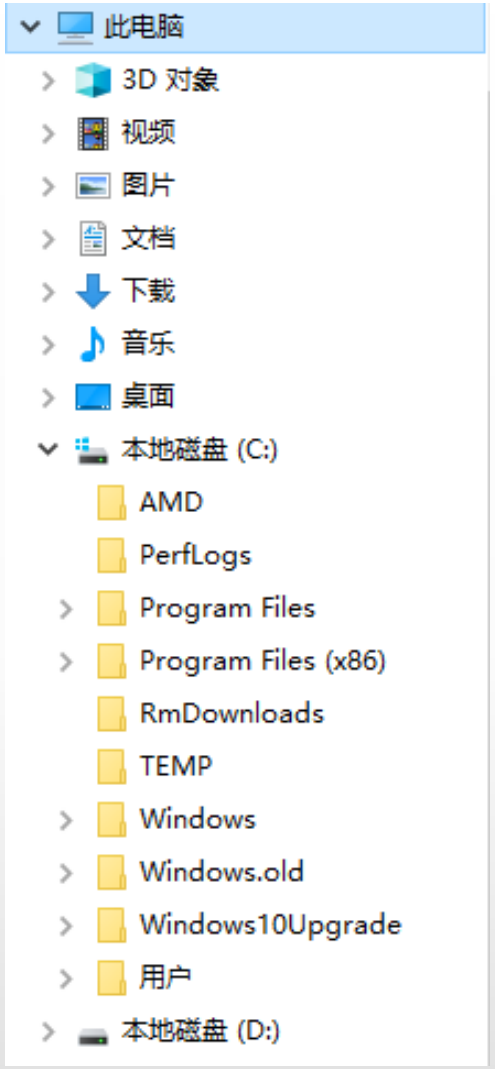
第4章　树和二叉树

什么是树?

客观世界中许多事物存在层次关系

- 人类社会家谱
- 社会组织结构
- 图书信息管理





为什么要使用“树”？

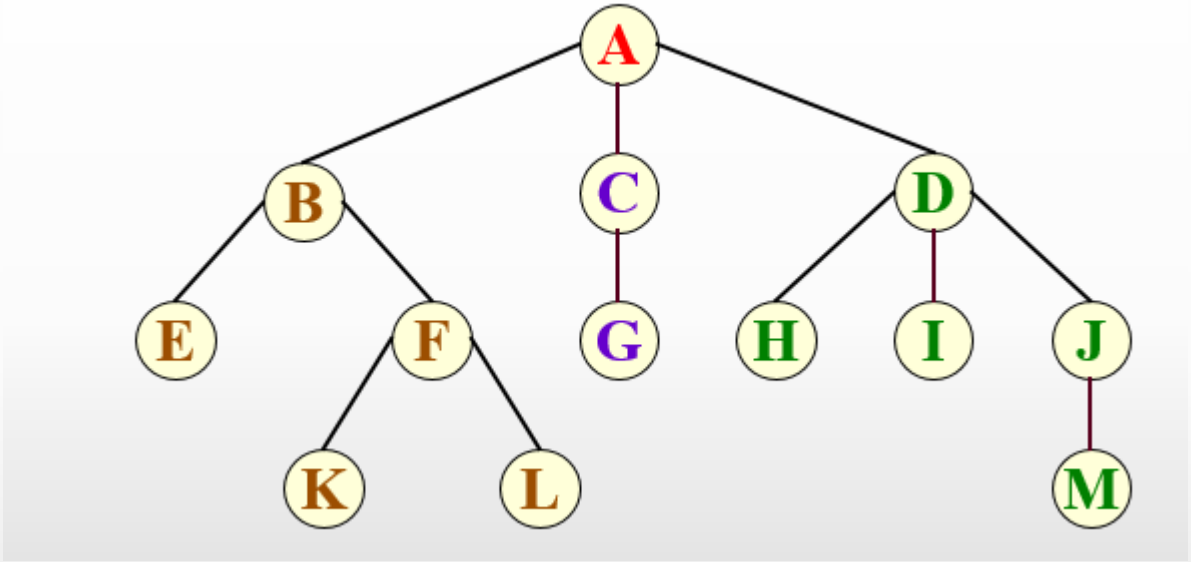
分层次组织在管理上具有更高的效率!

以查找为例，无序的线性结构查找时间复杂度是O（n），

而树可以达到O（logn），二分查找实际上就是在构造一棵查找树。

一、树的定义和基本术语

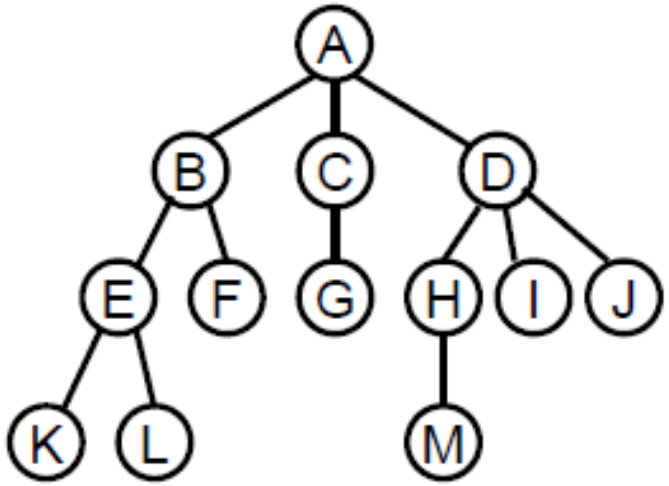
>>观看视频，回答问题



(1) 树的度是?
(2) 树的深度是?
(3) 这棵树的叶子分别是?
(4) 结点F所在的层次及它的度分别是?
(5) 请描述结点A到结点M的路径，并说明它们的路径长度。

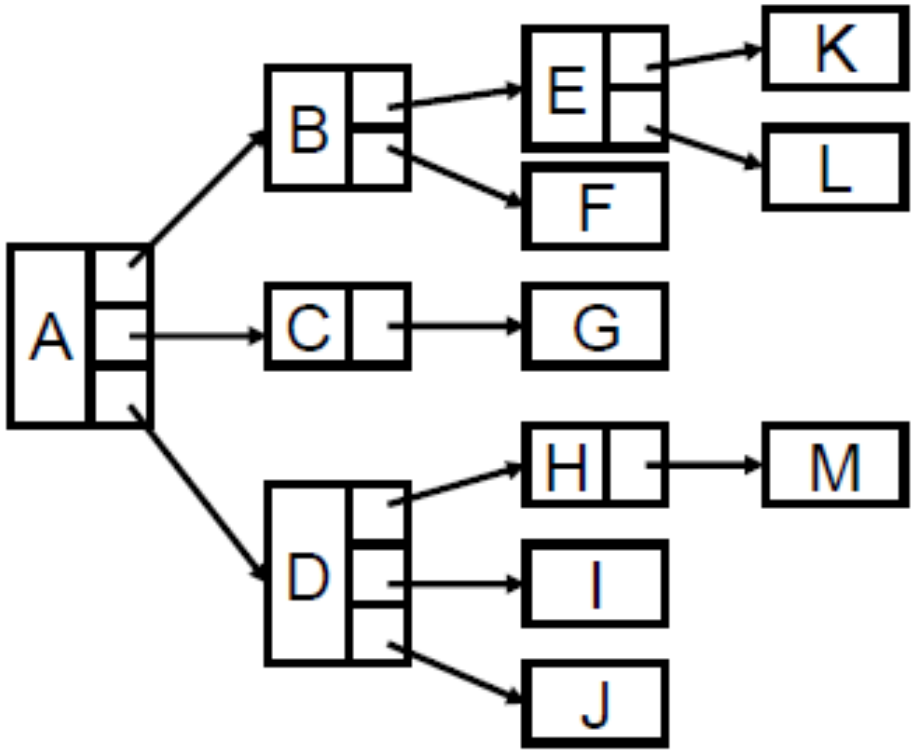
线性结构	树型结构
第一个数据元素（无前驱）	根结点（无前驱）
最后一个数据元素(无后继)	多个叶子结点(无后继)
其它数据元素 (一个前驱、一个后继)	其它数据元素 (一个前驱、多个后继)

二、树的表示



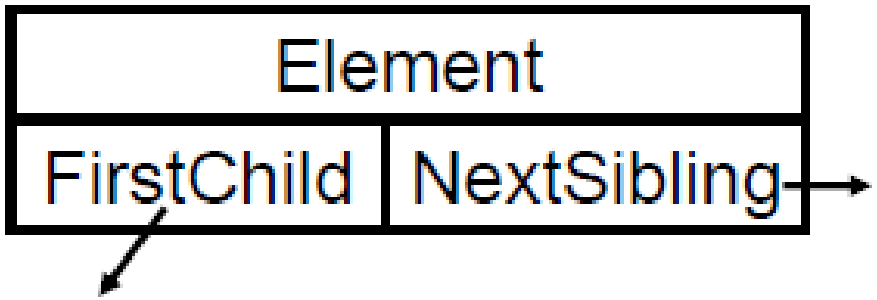
使用顺序结构（数组）表示合适么？

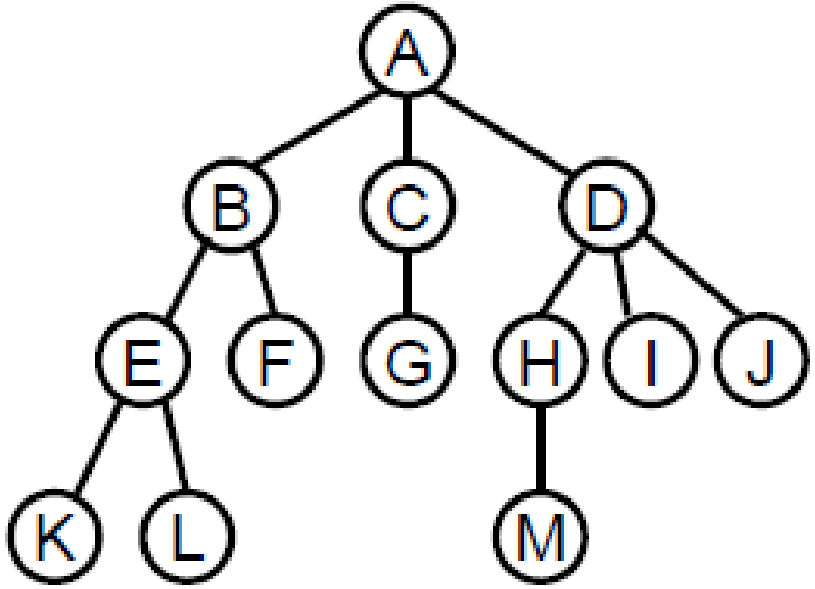
链式存储（链表）：每个结点以一个结构表示

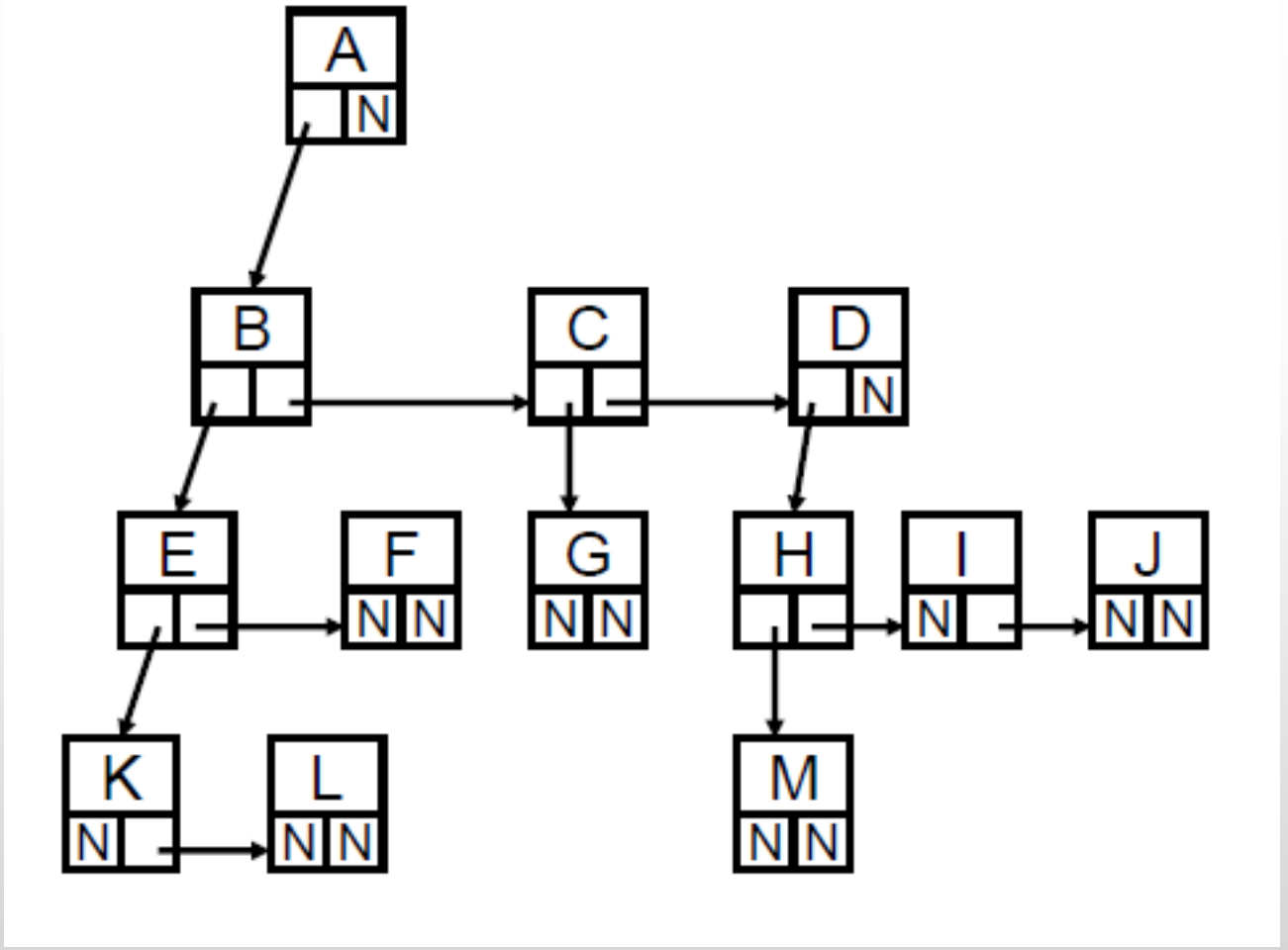


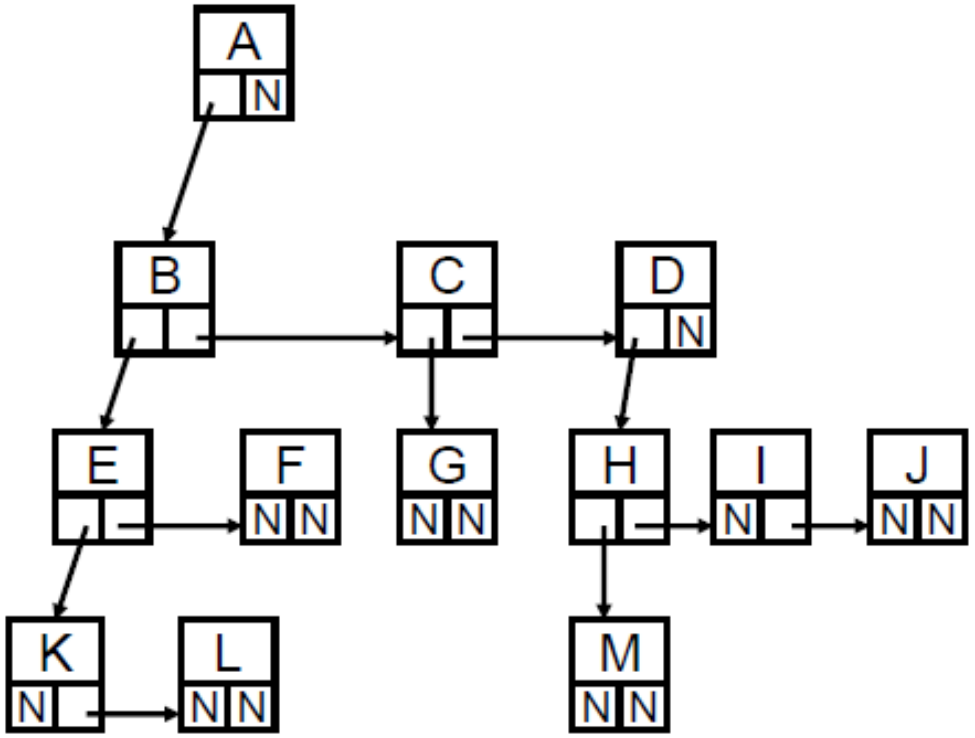
这种表示法有什么问题么？

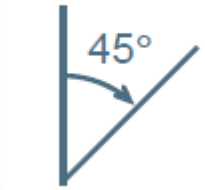
❖ 儿子-兄弟表示法


Element
FirstChild
NextSibling


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

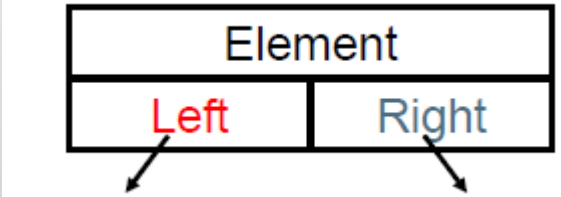

A
N
B
C
D
N
E
F
N N
G
N N
H
I
N
J
N N
K
N
L
N N
M
N N


A
N
B
C
D
N
E
F
N N
G
N N
H
I
N
J
N N
K
N
L
N N
M
N N


45°

A
N
B
E
C
K
N
F
N N
L
N N
G
N N
D
N
H
I
N
J
N N
M
N N

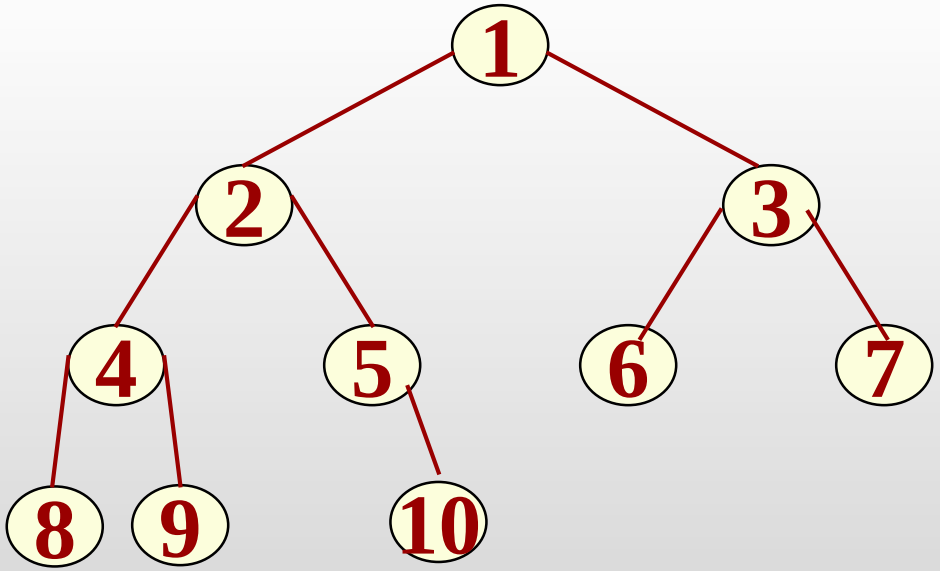
二叉树


Element
Left
Right

三、二叉树

1、二叉树的定义
2、二叉树的性质

>>观看视频，回答问题



请问这棵树是完全二叉树么？

高度为4的二叉树，第4层的最大结点数为_____，总结点数最多为_____ 。

1 如果一个完全二叉树最底下一层为第六层（根为第一层）且该层共有8个叶结点，那么该完全二叉树共有多少个结点？

- A. 31
- B. 39
- C. 63
- D. 71

2 若有一二叉树的总结点数为98，只有一个儿子的结点数为48，则该树的叶结点数是多少？

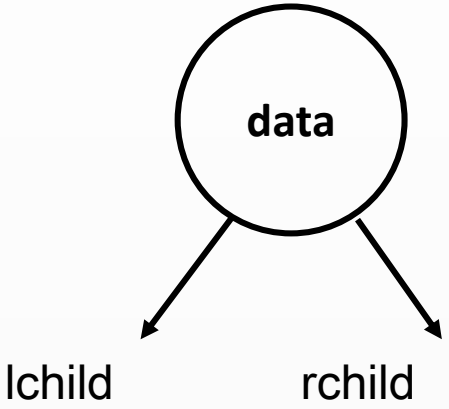
- A. 25
- B. 50
- C. 不确定
- D. 这样的树不存在

3、二叉树的存储结构

（1） 顺序存储　　>>>观看视频

（2） 链式存储

lchild	data	rchild



C 语言的类型描述如下:（教材P109）

```
typedef struct BiTNode {   // 结点结构
    TElemType      data;
    struct BiTNode  *lchild, *rchild;
                          // 左右孩子指针
} BiTNode, *BiTree;
```

4、二叉树的遍历（教材P110）

所谓遍历二叉树就是按某种顺序访问二叉树中的每个结点一次且仅一次的过程。这里的访问可以是输出、比较、更新、查看元素内容等等各种操作。

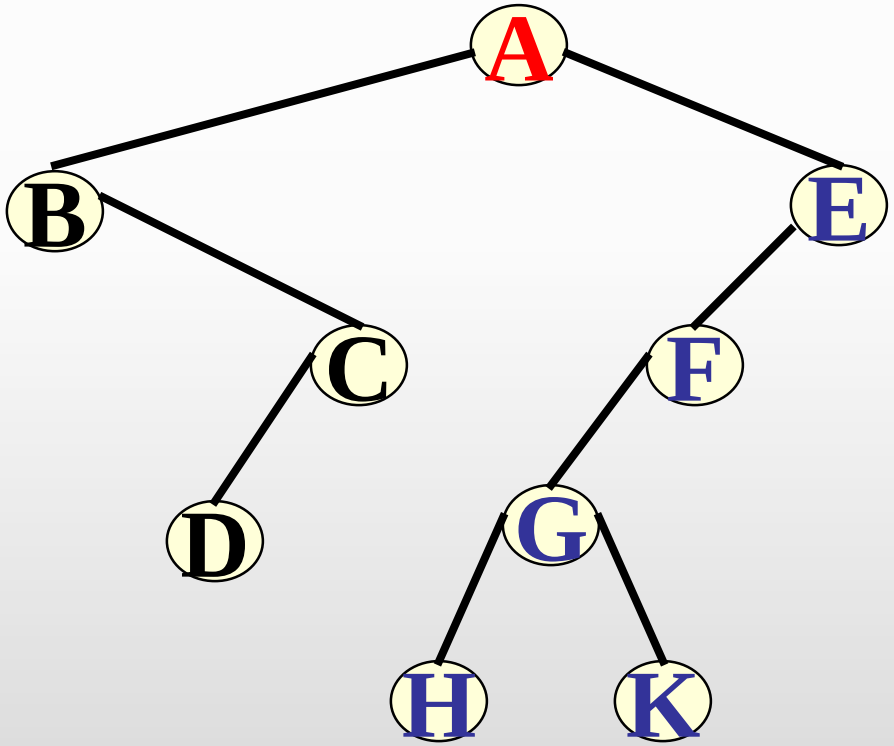
【Tips】“遍历”是任何数据类型都有的操作

线性结构：只有一条搜索路径

树：？

1、按层次遍历二叉树

实现方法为从上层到下层，每层中从左侧到右侧依次访问每个结点。



按层次遍历该二叉树的序列为：

A B E C F D G H K

2、按根、左子树和右子树三个部分进行遍历

二叉树的遍历方式存在六种可能：

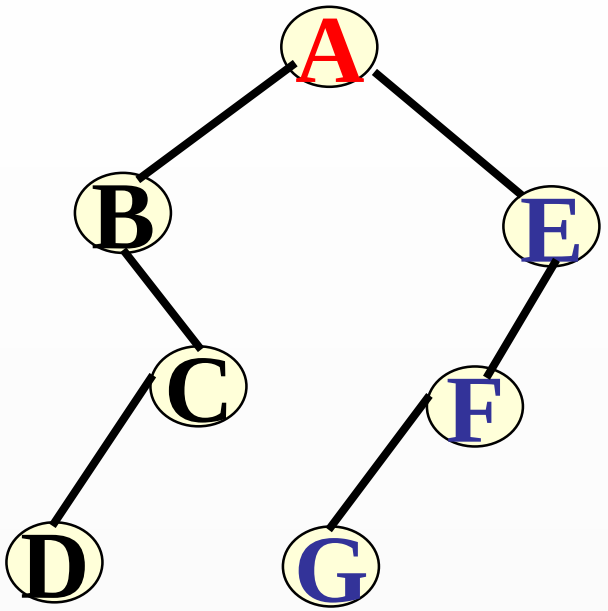
根左右、左根右、左右根、根右左、右根左、右左根

如果限定先左后右，则只有前三种方式，
即先序遍历、中序遍历、后序遍历

（1）先序遍历（教材P？）

遍历过程为：

- 访问根结点；
- 先序遍历其左子树；
- 先序遍历其右子树。




```
void PreOrderTraversal(BiTree BT)
{
  if (BT){                                  //树不空
  printf( “%c” ,BT->data);          // 访问根结点
  PreOrderTraversal(BT->lchild); // 遍历左子树
  PreOrderTraversal(BT->rchild);// 遍历右子树
  }
}
```

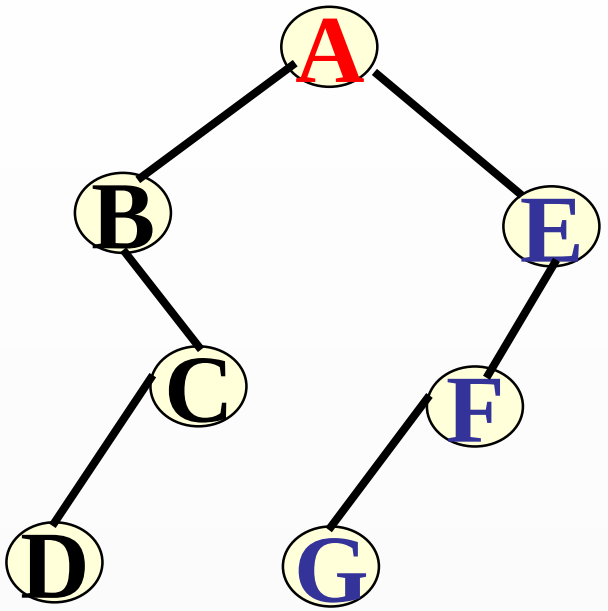
先序遍历结果:

A B C D E F G

（2）中序遍历（教材P？）

遍历过程为：

- 中序遍历其左子树；
- 访问根结点；
- 中序遍历其右子树。




```
void InOrderTraversal(BiTree BT)
{
   if (BT){                                  //树不空
   InOrderTraversal(BT->lchild); // 遍历左子树
   printf( “%c” ,BT->data);          // 访问根结点
   InOrderTraversal(BT->rchild);// 遍历右子树
   }
}
```

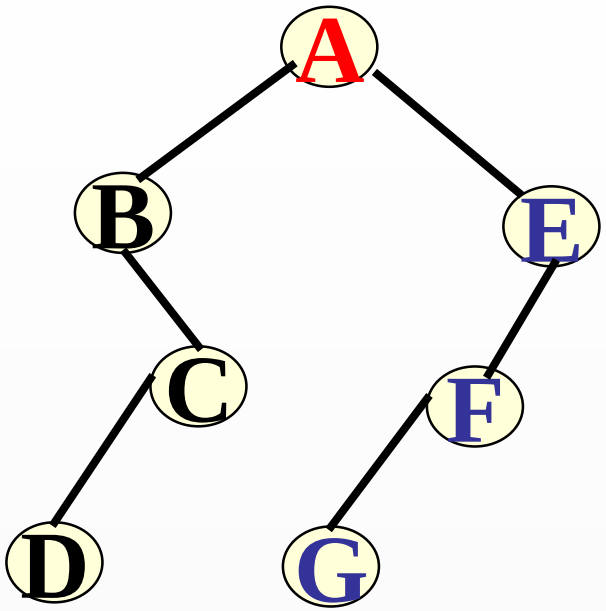
中序遍历结果:

B D C A G F E

（3）后序遍历（教材P？）

遍历过程为：

- 先序遍历其左子树；
- 先序遍历其右子树；
- 访问根结点。

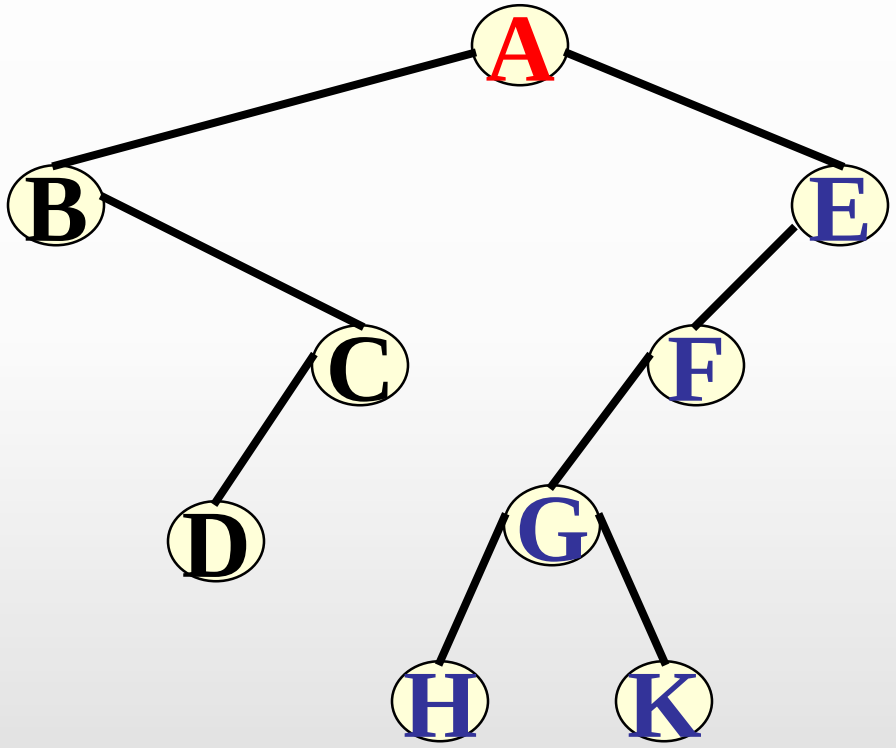



```
void PostOrderTraversal(BiTree BT)
{
  if (BT){                                  //树不空
  PostOrderTraversal(BT->lchild); // 遍历左子树
  PostOrderTraversal(BT->rchild);// 遍历右子树
  printf( “%c” ,BT->data);          // 访问根结点
  }
}
```

后序遍历结果:

DCBGFEA

【练习】分别写出先序、中序、后序遍历的结果。



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

5、遍历算法的应用

（1）输入结点值，构造二叉树

算法基本思想:

先序(或中序或后序)遍历二叉树，读入一个字符，若读入字符为空，则二叉树为空,若读入字符非空，则生成一个结点。

将算法中“访问结点”的操作改为：生成一个结点，输入结点的值。

```
BiTree CreateBiTree (BiTree BT)
{
   char ch;
   scanf( “%c” ,&ch);
   if(ch== ‘#’ )  BT=NULL;     //以#表示虚结点的值
   else
      {
       BT=(BiTree)malloc(sizeof(BiTNode));
       BT->data=ch;                    //生成根结点
       BT=CreateBiTree( BT->lchild);      //构造左子树
       BT=CreateBiTree( BT->rchild);      //构造右子树
      }
   return BT;
}
```

(2) 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。

需先分别求得左、右子树的深度，算法中“访问结点”的操作改为：求得左、右子树深度的最大值，然后加1。

```
int Depth (BiTree BT )
{
   int depthleft,depthright;
   if (BT==NULL ) return 0;
    else
    {
     depthleft = Depth( BT->lchild );
     depthright= Depth( BT->rchild );
     if(depthleft>=depthright)
           return  depthleft+1;
     else
           return  depthright+1;
     }
}
```

（3）统计二叉树中叶子结点的个数

算法基本思想:类似求二叉树深度的算法。
二叉树的叶子总数等于它的左、右子树叶子数之和。需先判断结点是否为叶子。

```
int CountLeaf(BiTree BT)
{
   int lnum,rnum;
   if(BT==NULL) return 0;
   else if(BT->lchild==NULL)&&(BT->rchild==NULL)) return 1;
   else
      {
         lnum=CountLeaf( BT->lchild);
         rnum=CountLeaf( BT->rchild);
         return (lnum+rmum);
      }
}
```

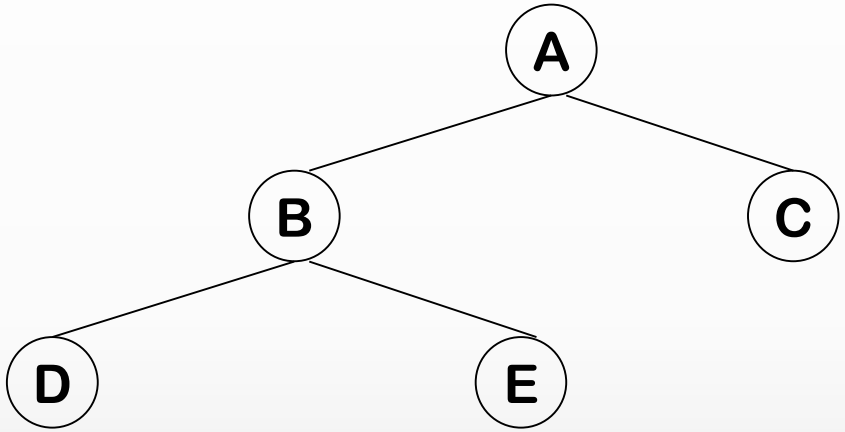
任意一棵二叉树结点的先序序列、中序序列和后序序列都是唯一的。

【问题1】若已知二叉树结点的先序序列和中序序列，能否确定这棵二叉树呢？这样确定的二叉树是否是唯一的呢？

【问题2】若已知二叉树结点的先序序列和后序序列，能否确定这棵二叉树？

由遍历序列恢复二叉树

【例】一棵二叉树的前序序列为：ABDEC，中序序列为：DBEAC。请画出这棵树。



【练习】已知二叉树的中序和后序序列分别为：DCBEA和DCEBA；请画出这棵树，并写出先序序列。

1 假定只有四个结点A、B、C、D的二叉树，其前序遍历序列为ABCD，则下面哪个序列是不可能的中序遍历序列？

- A. ABCD
- B. ACDB
- C. DCBA
- D. DABC

2 对于二叉树，如果其中序遍历结果与前序遍历结果一样，那么可以断定该二叉树_____

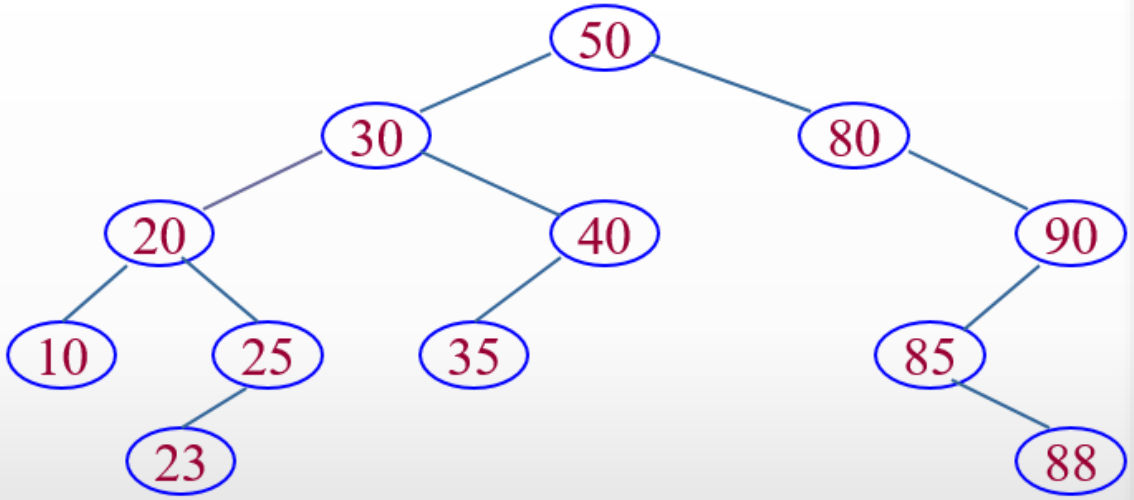
- A. 是完全二叉树
- B. 所有结点都没有左儿子
- C. 所有结点都没有右儿子
- D. 这样的树不存在

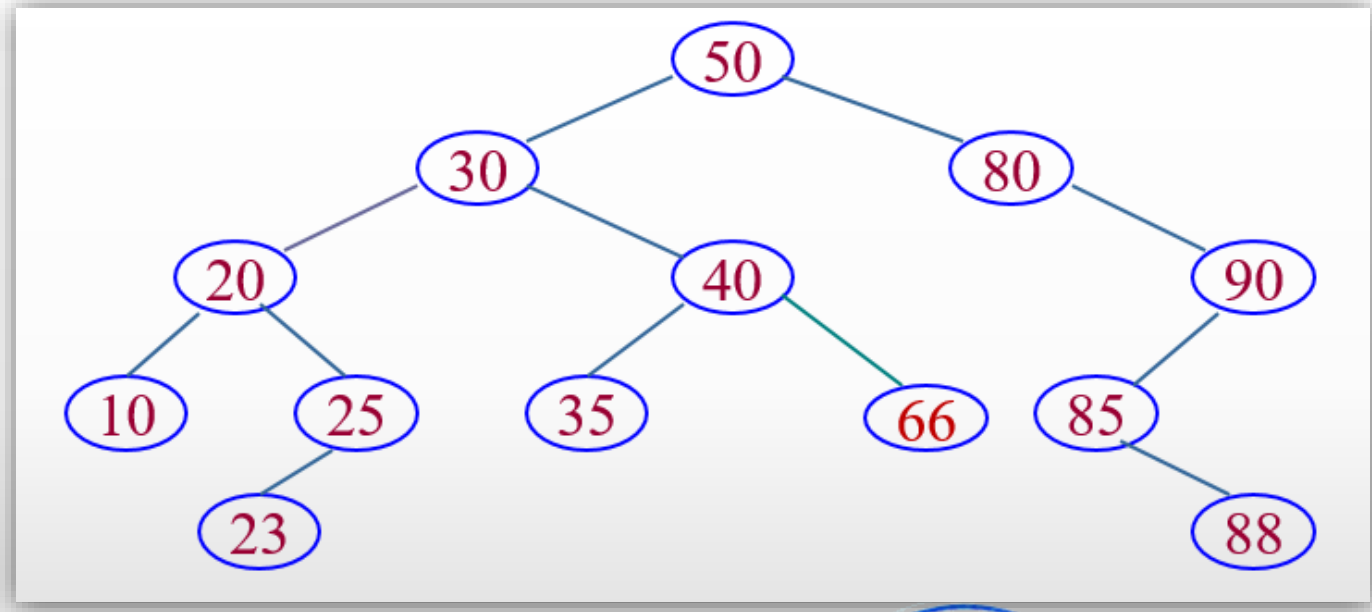
3 已知一二叉树的后序和中序遍历的结果分别是FDEBGCA 和FDBEACG,那么该二叉树的前序遍历结果是什么？

- A. ABDFECG
- B. ABDEFCG
- C. ABDFEGC
- D. ABCDEFG

四、二叉搜索树

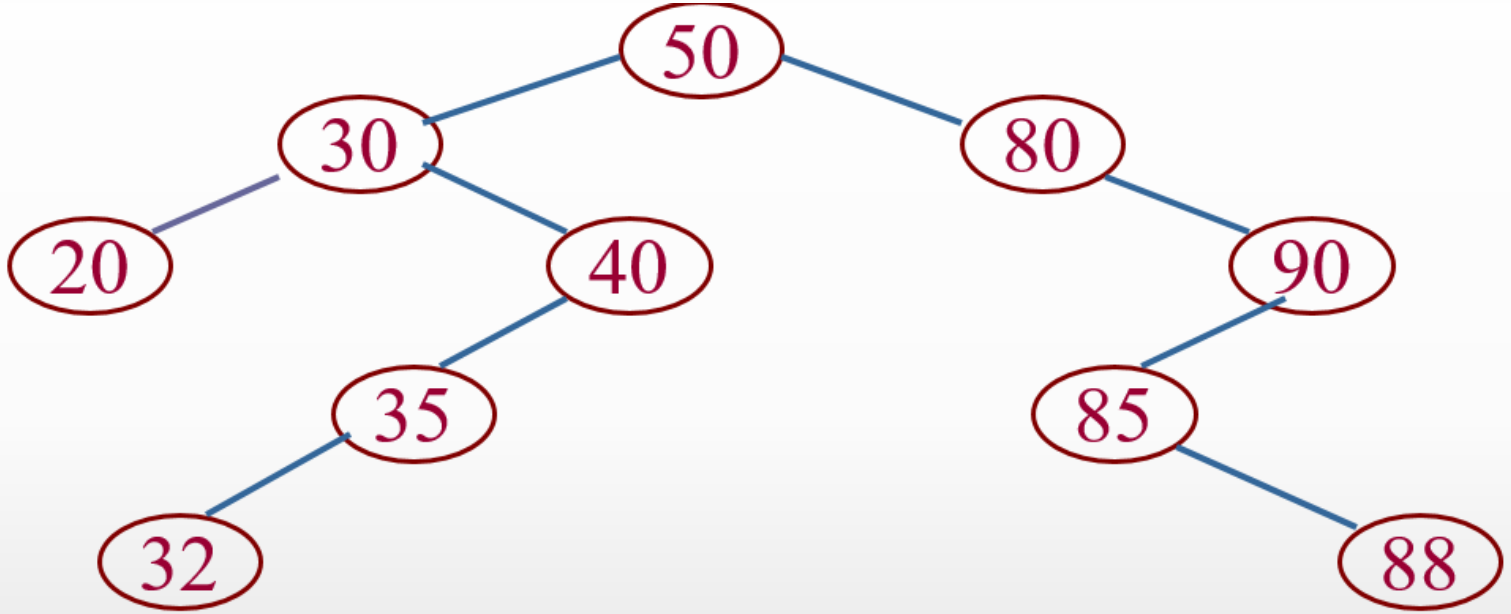
1、二叉搜索树的定义　>>观看视频，回答问题





请问这两棵树是二叉搜索树么？

2、二叉搜索树的查找算法　>>观看视频，回答问题

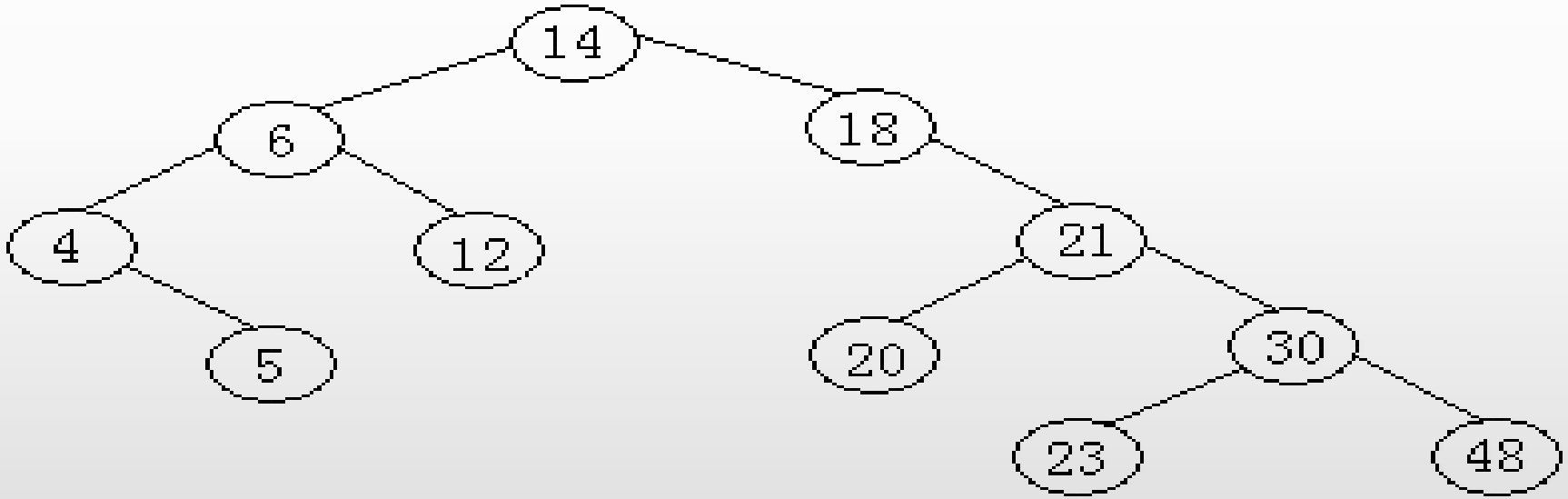


请分别描述 查找关键字 50、35、90、95 的路径

2、二叉搜索树的插入算法 >>观看视频，回答问题

【练习】试按表中顺序构造一棵二叉搜索树。

(14，6，18，21，4，5，30，23，20，12，48)



求所构造这棵树的ASL（平均查找长度P133）

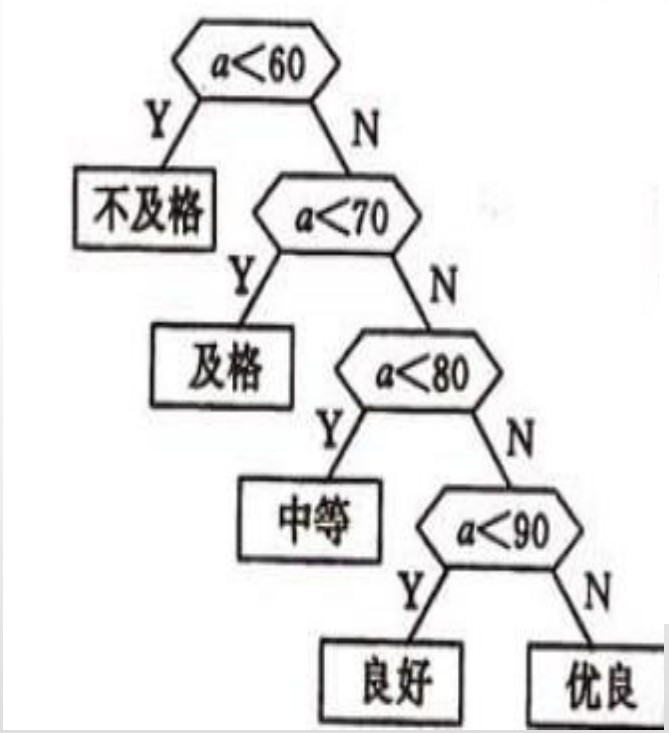
3、二叉搜索树的删除算法　>>观看视频

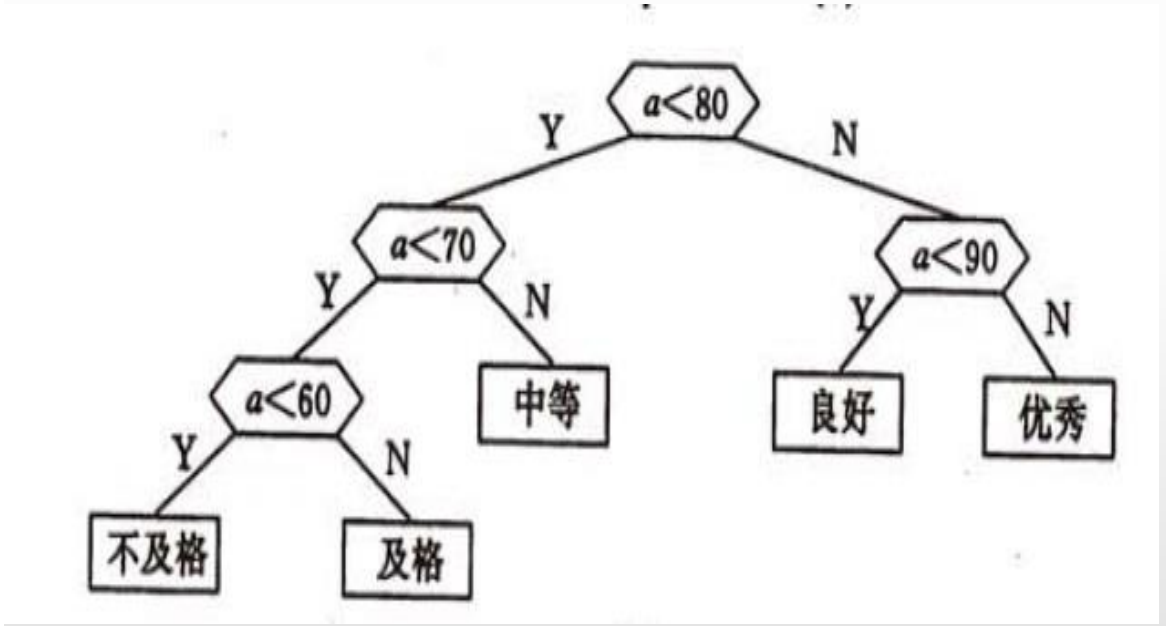
可分三种情况讨论：

- （1）被删除的结点是叶子；
- （2）被删除的结点只有左子树或者只有右子树；
- （3）被删除的结点既有左子树，也有右子树。

五、哈夫曼树与哈夫曼编码

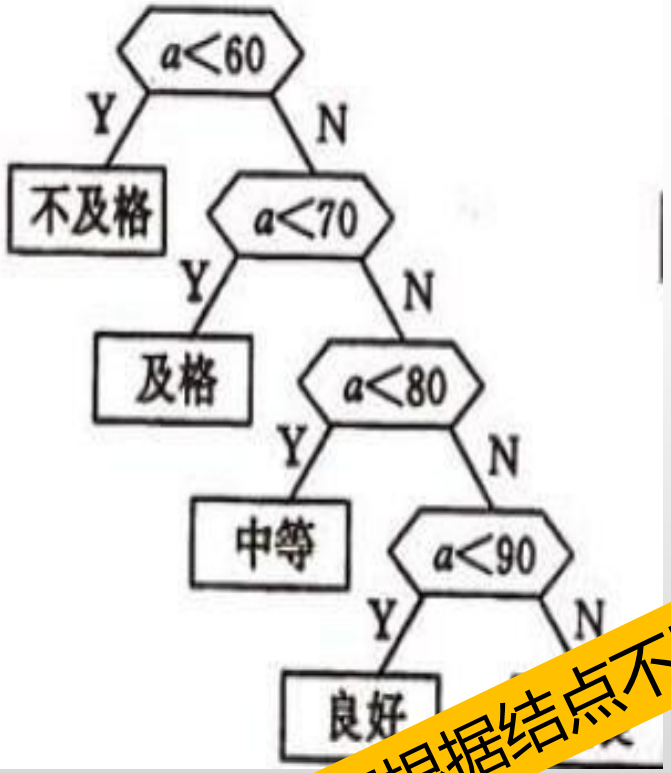
【例】编制一个程序，将百分制转换成五个等级输出。



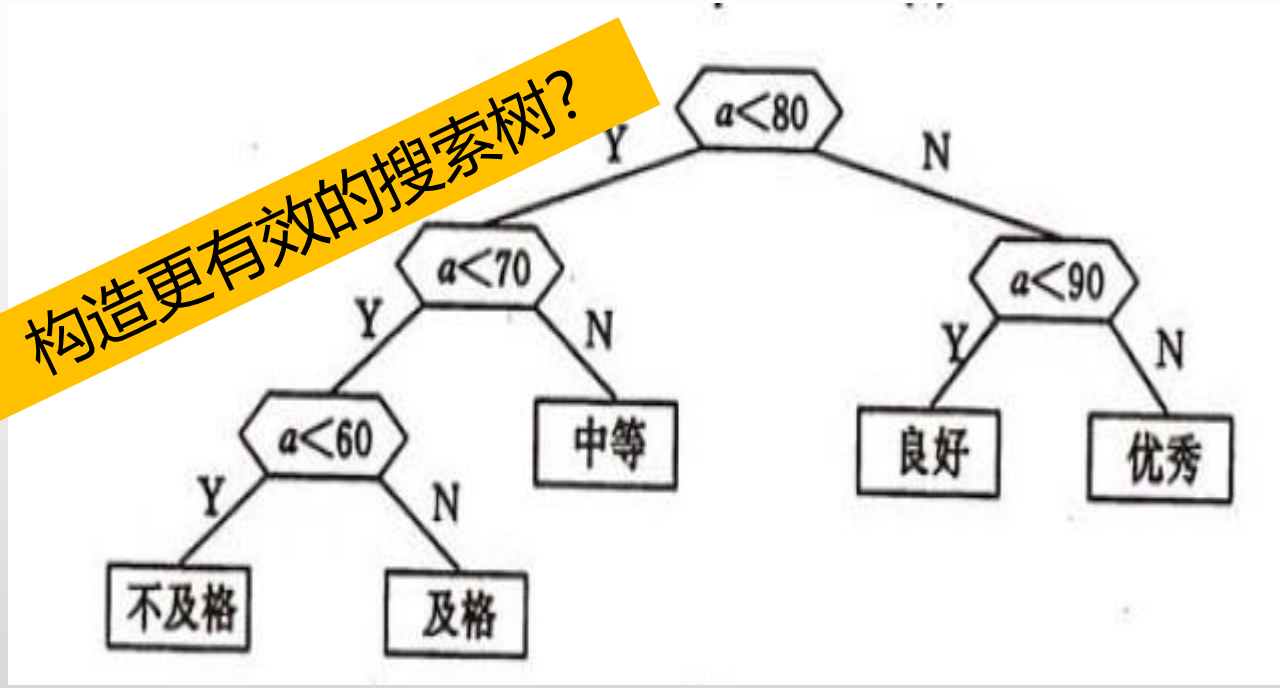


在实际中，学生的成绩在五个等级上的分布是不均匀的，假设其分布规律如下表所示：

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



查找效率：**0.05× 1+0.15 ×2+0.4× 3+0.3 ×4+0.1× 4 = 3.15**

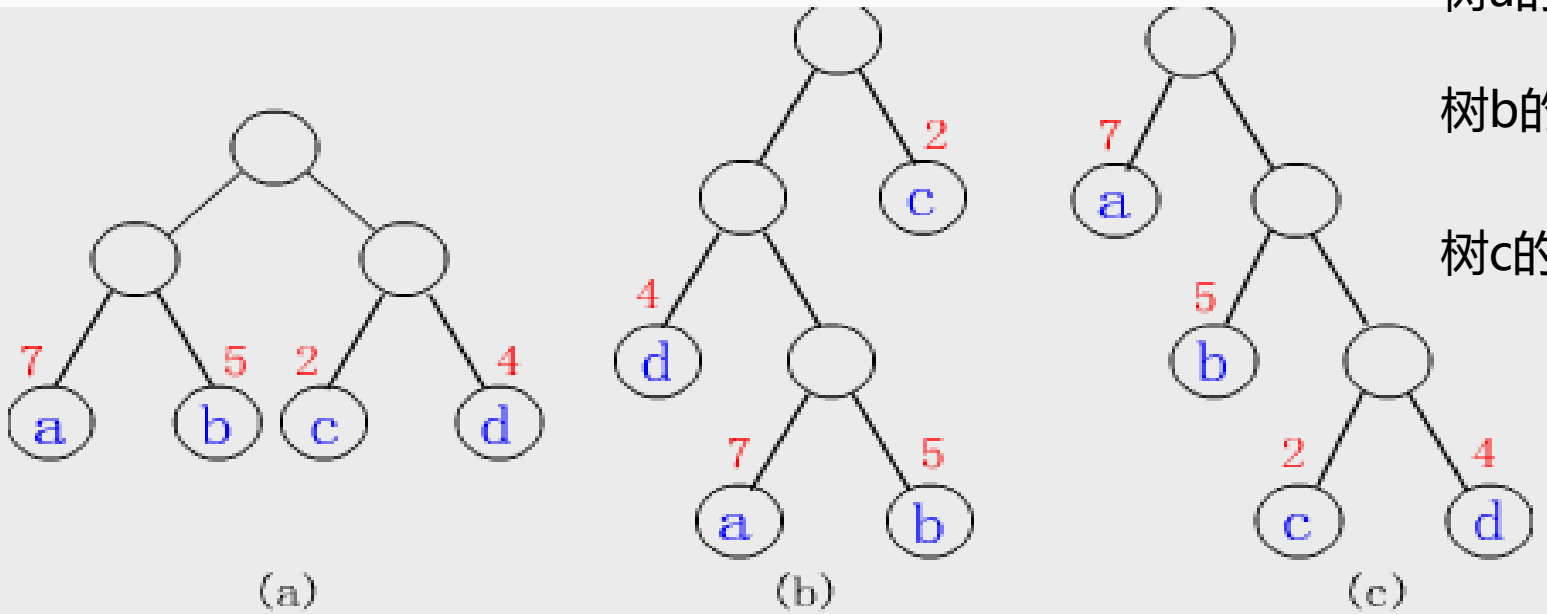


如何根据结点不同的查找频率，构造更有效的搜索树？

查找效率：**0.05× 3+0.15 ×3+0.4× 2+0.3 ×2+0.1× 2=2.2**

哈夫曼（Huffman)树，也叫赫夫曼树，它是一颗最优二叉树：WPL最小的二叉树。

WPL（带权路径长度）：设二叉树有n个叶子结点，每个叶子结点带有权值 w_k，从根结点到每个叶子结点的长度为 l_k，则每个叶子结点的带权路径长度之和就是：$WPL= \Sigma w_k l_k\ (k=1到n)$



具有不同WPL的二叉树(结点旁的数字为权)

树a的WPL=7×2+5×2+2×2+4×2=36

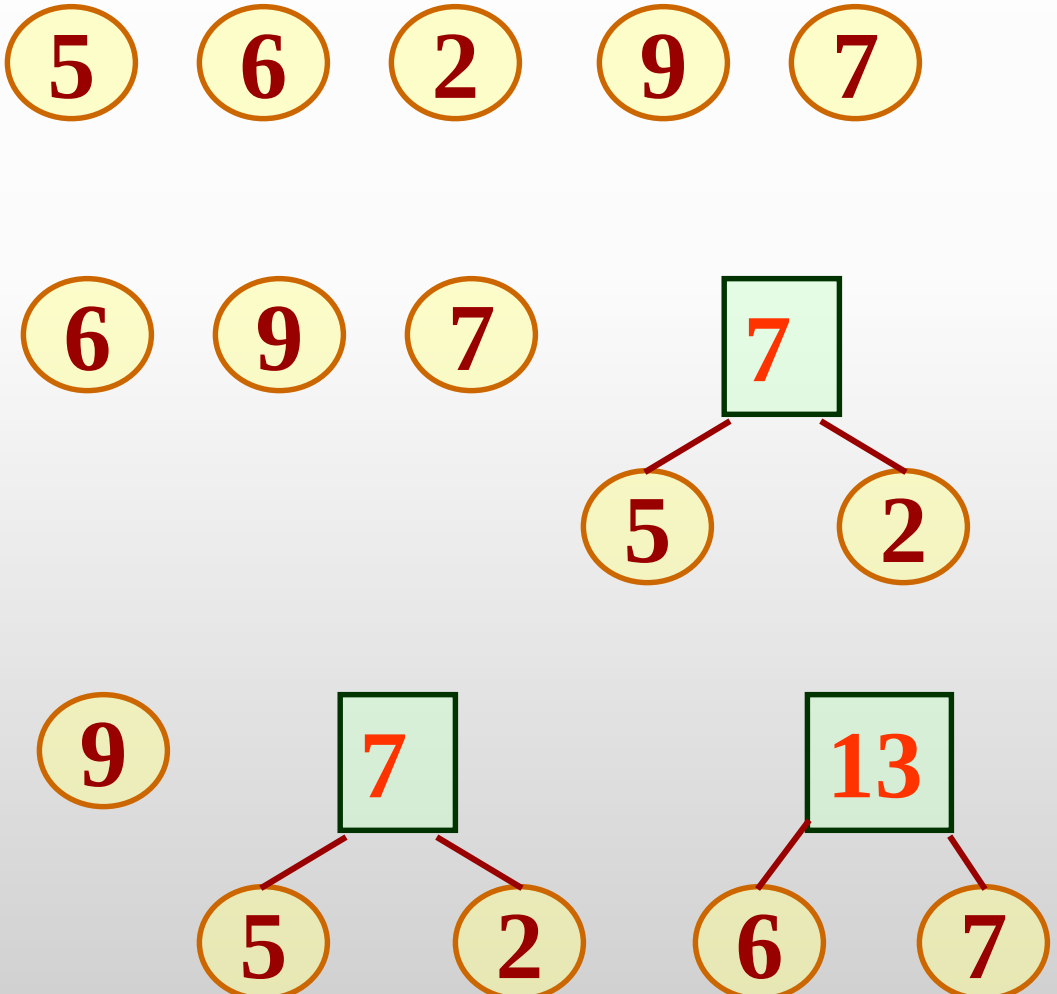
树b的WPL=7×3+5×3+4×2+2×1=46

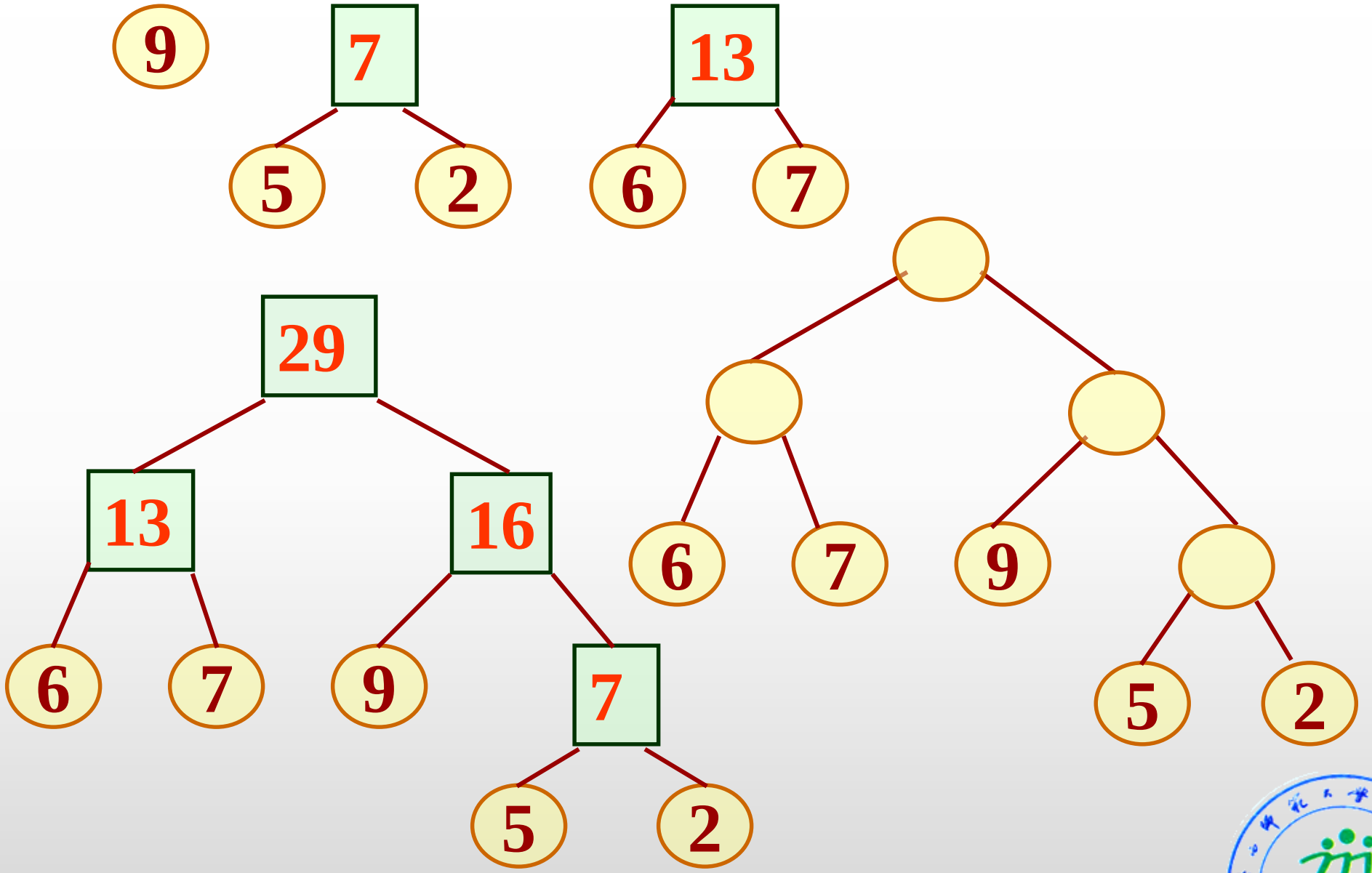
树c的WPL=7×1+5×2+2×3+4×3=35

1、哈夫曼树的构造

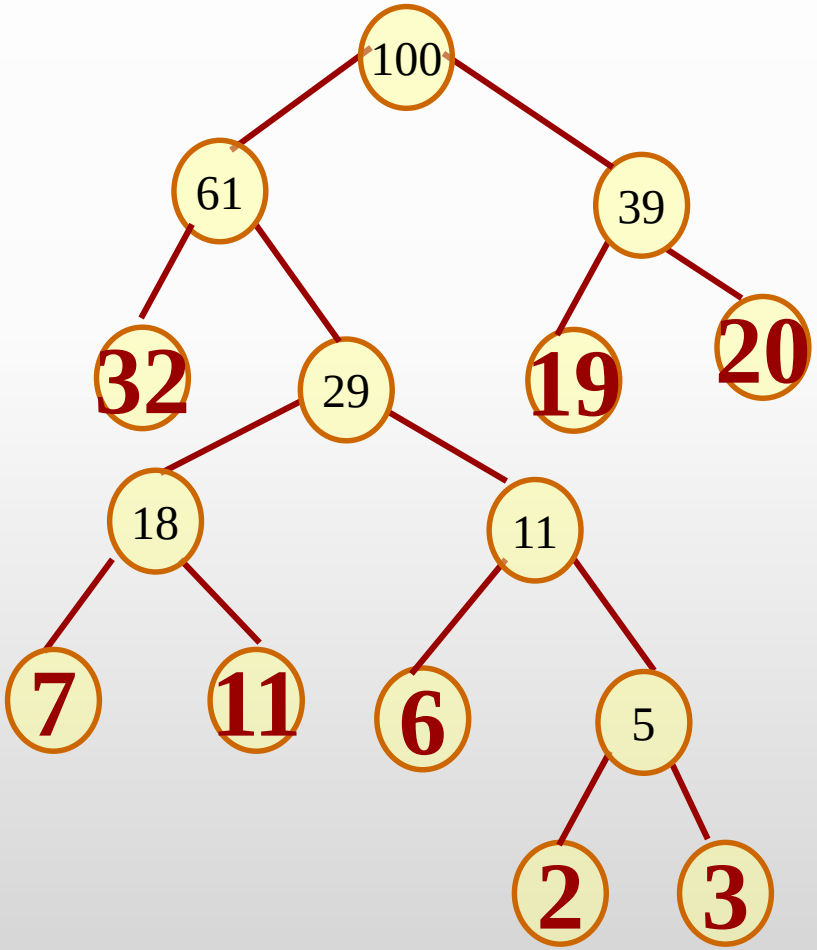
每次把权值最小的两棵二叉树合并

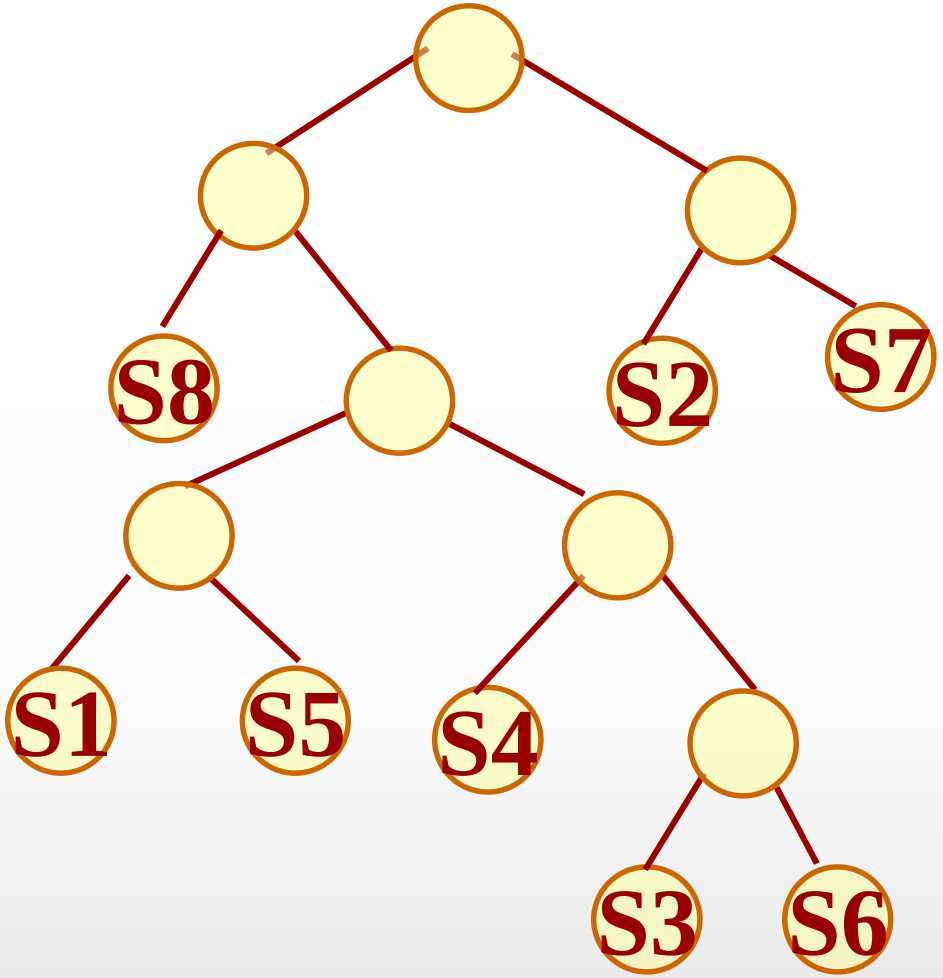
例如: 已知权值 W={ 5, 6, 2, 9, 7 }




9
7
5
2
13
6
7
29
13
6
7
16
9
7
5
2
6
7
9
5
2

【练习】设有权值分别为7，19，2，6，11，3，20，32的八种符号S1-S8，画出这些符号组成的赫夫曼树。


100
61
39
32
29
19
20
18
11
7
11
6
5
2
3


S8
S2
S7
S1
S5
S4
S3
S6

2、哈夫曼树的特点

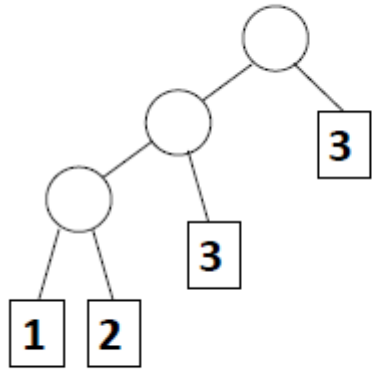
✓没有度为**1**的结点；
✓n个叶子结点的哈夫曼树共有**2n-1**个结点；

n0：叶结点总数
n1：只有一个儿子的结点总数
n2：有2个儿子的结点总数

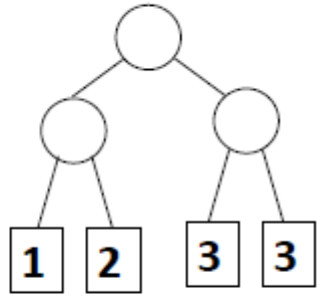
$n2=n0-1$

✓哈夫曼树的任意非叶节点的左右子树交换后仍是哈夫曼树；
✓对同一组权值{***w1*，*w2*，……，*wn***}，是否存在不同构的两棵哈夫曼树呢？

对一组权值{ 1, 2 , 3, 3 }，不同构的两棵哈夫曼树：



WPL = 18



WPL = 18

3、哈夫曼编码

解决早期远距离通信（主要是电报）的数据传输最优化问题。

例如，我们有一段文字内容为“ABACCDA”，现在要通过网络传输给别人。可以用两位二进制表示的编码分别为00，01，10，11。

则应发送二进制序列：00010010101100，总长度为14位。

当接收方接收到这段电文后，将按两位一段进行译码。

问题：编码长度并不是最短的。

解决办法：使用频度较高的字符分配一个相对比较短的编码，使用频度较低的字符分配一个比较长的编码。

“ABACCDA”

可以为A，B，C，D四个字符分别分配0，00，1，01，
则发送的二进制序列是：000011010发送，总长度只有9个二进制位。

问题：接收方接到这段电文后无法进行译码。
因为无法断定前面4个0是4个A，1个B、2个A，还是2个B，即译码不唯一，因此这种编码方法不可使用。

如何避免二义性?

使用前缀码（prefix code）：任何字符的编码都不是另一字符编码的前缀

a: 1
e: 0
s: 10

如，a是s的前缀，所以它们不是前缀码

哈夫曼编码是一种最优前缀编码

构造方法如下：

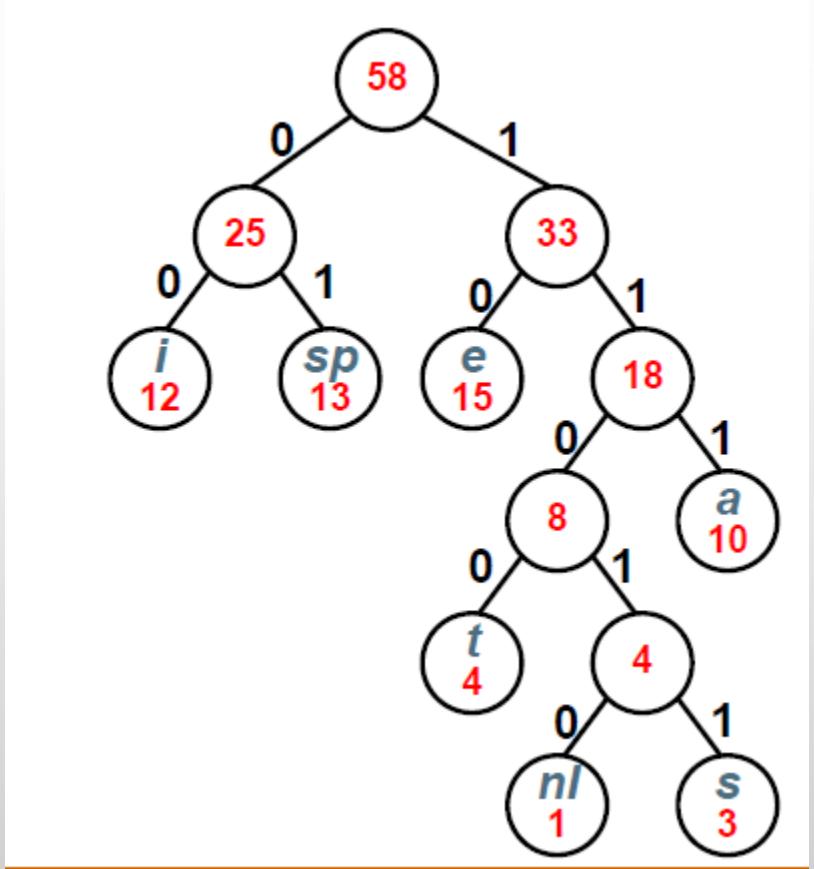
- 利用字符集中每个字符的使用频率作为权值构造一个哈夫曼树；
- 从根结点开始，为到每个叶子结点路径上的左分支赋予0，右分支赋予1，并从根到叶子方向形成该叶子结点的编码。

【例】假设有一个电文字符集中有{a,e,i,s,t,sp,nl}7个字符，每个字符的使用频率分别为{0.1,0.15,0.12,0.03,0.04,0.13,0.01}，试设计哈夫曼编码。

为方便计算，将所有字符的频度乘以100，得到

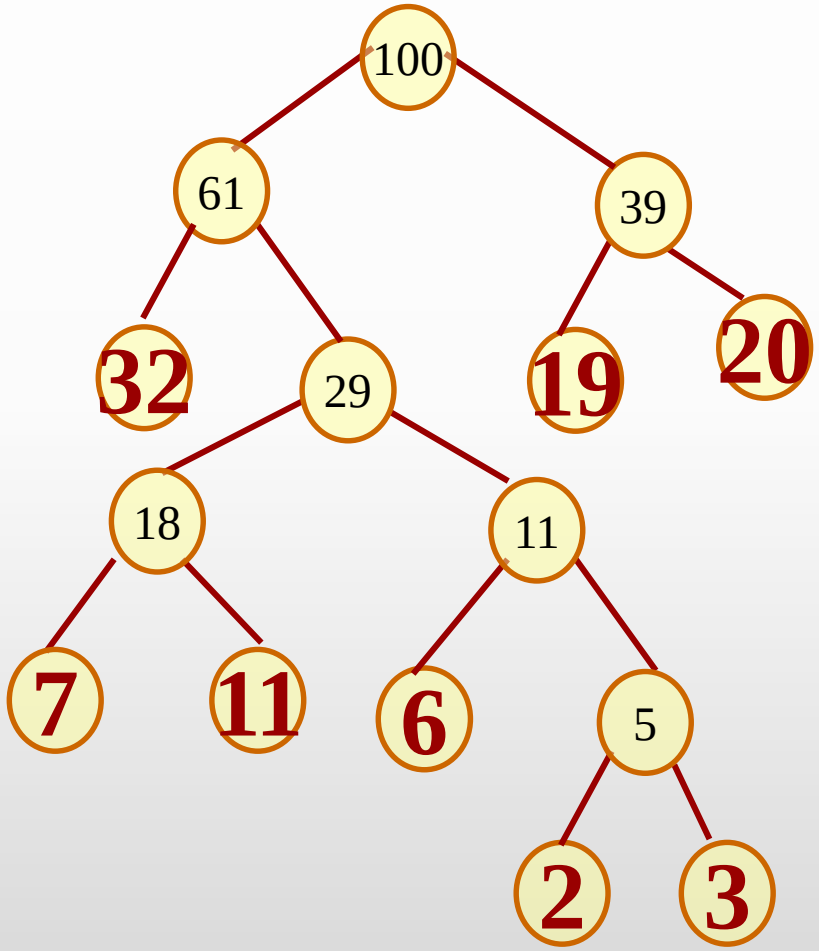
a	e	i	s	t	sp	nl
10	15	12	3	4	13	1

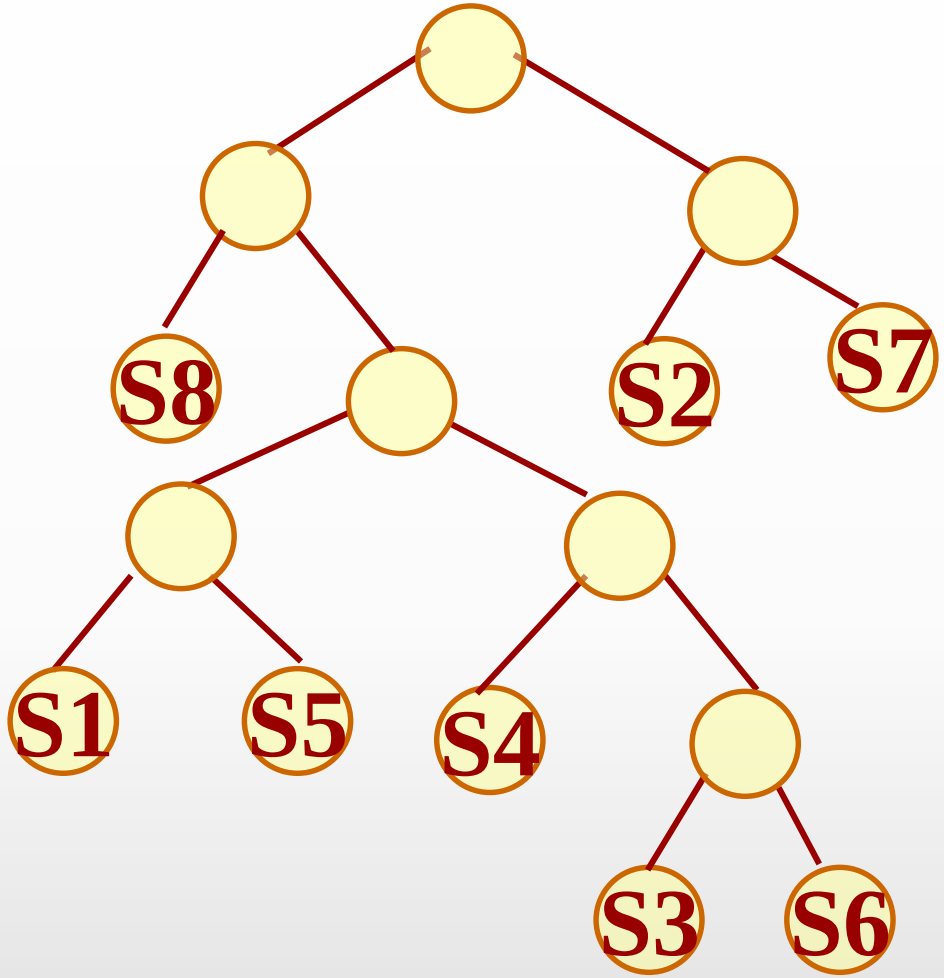
哈夫曼编码设计如图：



a : 111
e : 10
i : 00
s : 11011
t : 1100
sp : 01
nl : 11010

【练习】设某符号系统有8种符号S1-S8，其使用频率依次为0.07，0.19，0.02，0.06，0.11，0.03，0.20，0.32。试为这8种符号设计赫夫曼编码。要求画出赫夫曼树。


100
61
39
32
29
19
20
18
11
7
11
6
5
2
3


S8
S2
S7
S1
S5
S4
S3
S6