数据结构与算法

Data Structures and Algorithms

第三部分 树

回顾：树--3

1. 二叉树的线索化过程（中序）
2. 堆的概念及ADT操作（插入和删除）
3. 树
   （1）概念
   （2）三种遍历方式
   （3）存储方式
      -- 双亲表示法
      -- 孩子表示法
      -- 孩子兄弟表示法
   （4）树、二叉树、森林的转换（左孩子右兄弟）

3.7.2 哈夫曼树及其应用

- 应用：在电报通信中，电文是以二进制按照一定的编码反射传送。
- 发送方：按照预先规定的方法将要传送的字符换成0和1组成的序列----编码；
- 接收方：由0和1组成的序列换成对应的字符----解码

如何编码能获得较高的传送效率？

Huffman成功解决了该问题！

3.7.2 哈夫曼树及其应用

- **Huffman教授简介**
  - David Huffman教授，美国人，1999年逝世。
  - 在他的一生中，他对于有限状态自动机、开关电路、异步过程和信号设计有杰出的贡献。
  - 他发明的Huffman编码能够使我们通常的数据传输数量减少到最小。
  - 1950年，Huffman在MIT(麻省理工)的信息理论与编码研究生班学习。Robert Fano教授让学生们自己决定是参加期末考试还是做一个大作业。而Huffman选择了后者，原因很简单，因为解决一个大作业可能比期末考试更容易通过。这个大作业促使了Huffman以后算法的诞生。

3.7.2 哈夫曼树的引入

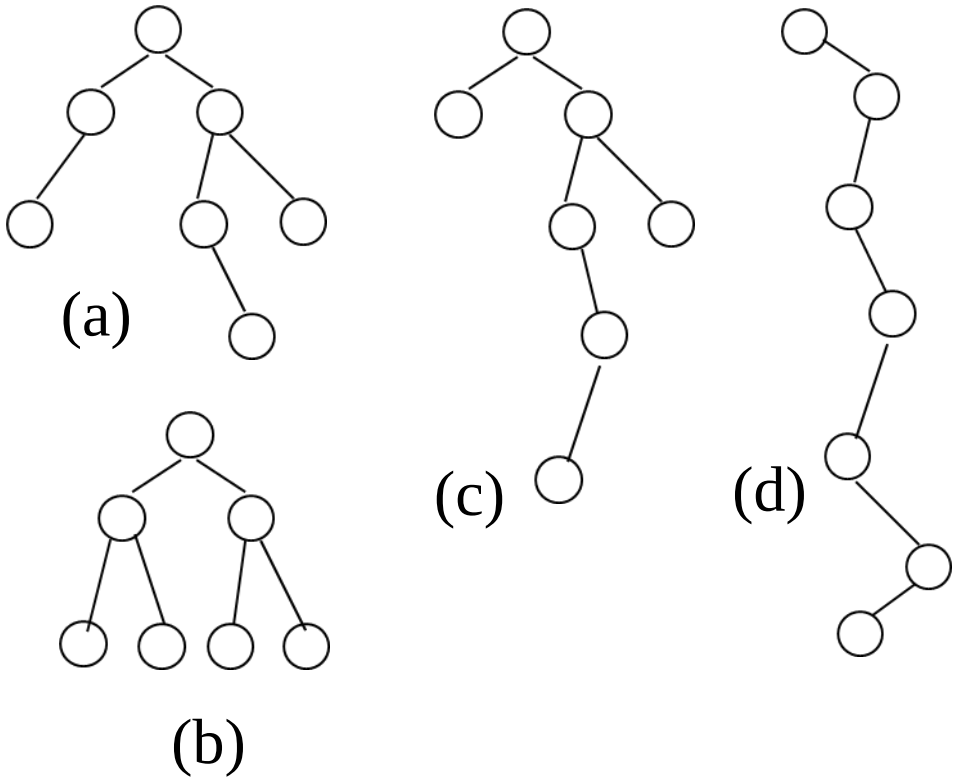
路径长度：结点和树？

La=2*1+3*2+1*3=11

Lb=2*1+4*2=10

Lc=2*1+2*2+1*3+1*4=13

Ld=1+2+3+4+5+6=21

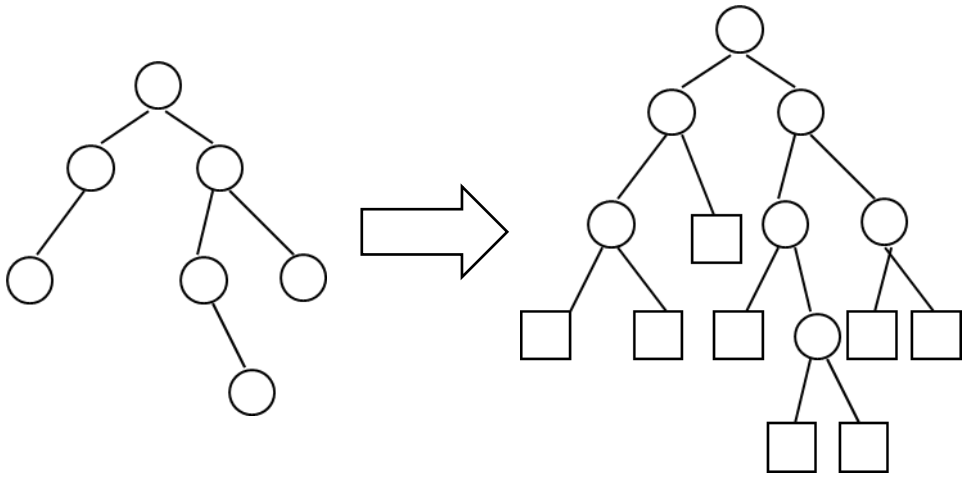


二叉树路径长度以与满或完全二叉树相同的高度形态为最小。

增长树的概念

将非满二叉树中，所有度不满2的结点扩充为2，便得到了扩充二叉树。扩充的节点称为外部结点，其余原来的节点称为内部结点。

增长树 { 内结点 ○
　　外结点 □



如内结点数为 n，则外结点 S = n + 1

内结点路径长度 I = 2×1+3×2+1×3 = 11;

外结点路径长度 E = 1×2+5×3+2×4 = 25;

如内结点路径长度为I，则外结点路径长度 E = I＋2×n。

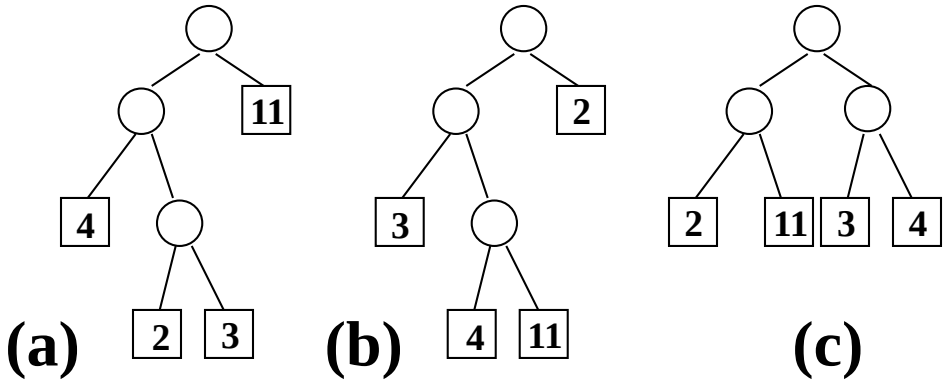
带权路径长度

- **权**：树中结点常常被赋予一种表示某种意义的数值。
- **带权路径长度**：

从树的根到任意结点的路径长度与该结点上权值的乘积。

设：$w_i = \{2, 3, 4, 11\}$

求：$\sum w_j \cdot l_j$（加权路长）



（a）11×1＋4×2＋2×3＋3×3=34

（b）2×1＋3×2＋4×3＋11×3=53

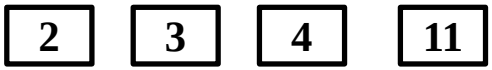
（c）2×2＋11×2＋3×2＋4×2=40

给定实数$w=\{w_1,w_2,\ldots,w_m\}$，构造以 w_i 为权的增长树，其中 $\sum w_i \cdot l_i$ 最小的一棵二叉树称为哈夫曼树。

$w_i = \{2, 3, 4, 11\}$，构造哈夫曼树

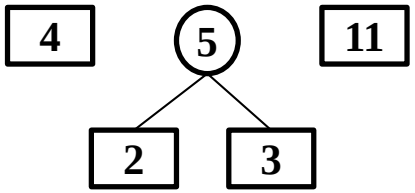
（1）将每一个w_i作为一个外结点，并按从小到大顺序排列（n个森林）；

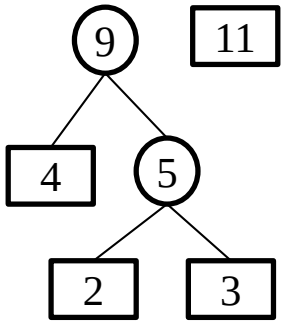
外结点：

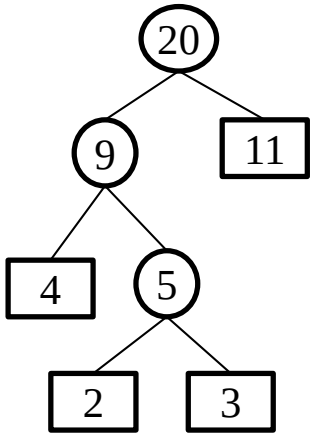


（2）选取最小的两个外结点，增加一个内加点，形成一个增长树。

外结点权之和写入内结点，与其他外结点/增长树一起再次排序。







（3）重复步骤（2）

（4）直到最后形成一棵增长树，为哈夫曼树。

$\sum w_i \cdot l_i = 34$

构造Huffman树的步骤：

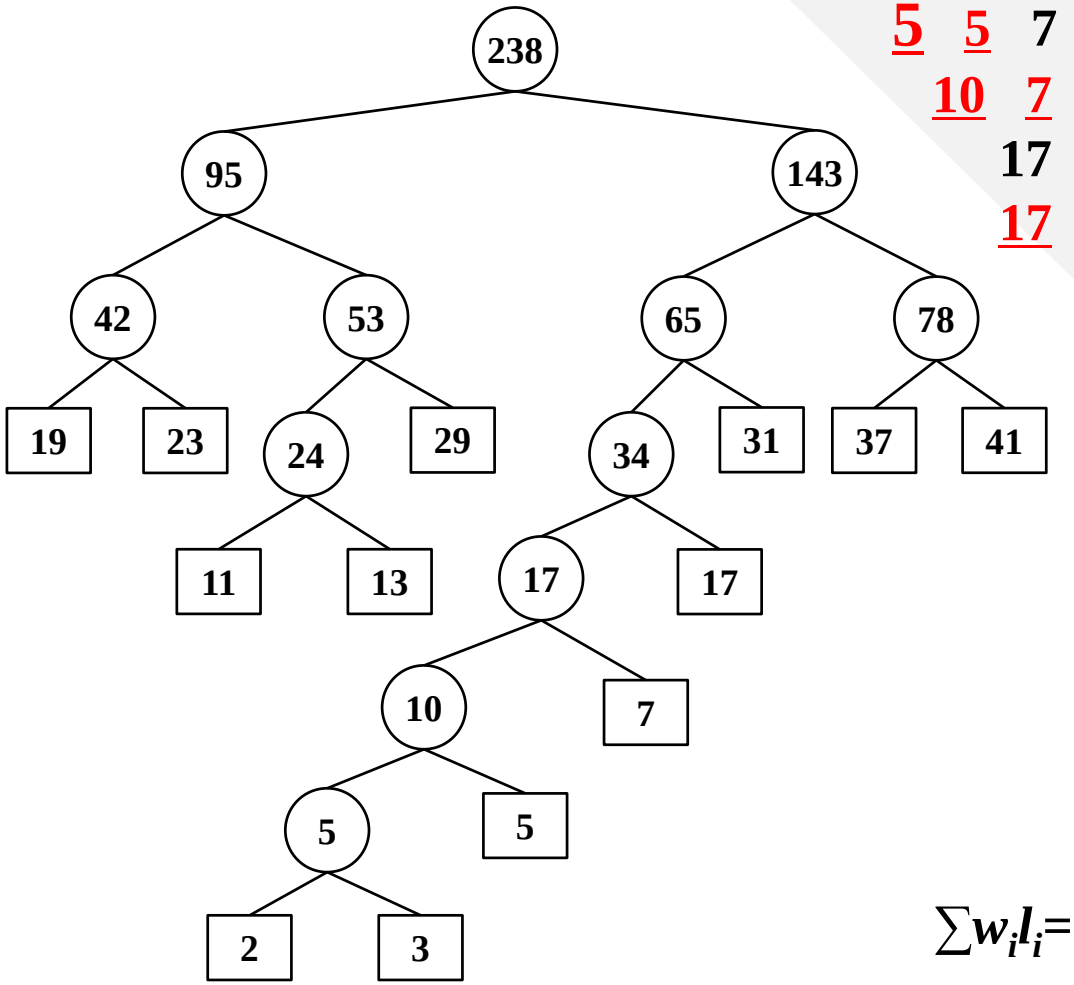
1. 给定的n个权值，分别构造成n个二叉树，并作为各自的根。
2. 设由这n棵二叉树构成的集合为F，在F中选取两棵根结点树值最小的树作为左、右子树，构造一棵新的二叉树，设置新二叉树根的权值 = 左、右子树根结点权值之和；
3. 从F中删除这两棵树，并将新树加入F；
4. 重复 2、3，直到F中只含一颗树为止；

这棵树便是Huffman树。

⑤ ⑭ ㊵ ㉖ ⑩

哈夫曼树形成步骤

W_i = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41}



2	3	5	7	11	13	17	19	23	29	31	37	41
5	5	7	11	13	17	19	23	29	31	37	41	
10	7	11	13	17	19	23	29	31	37	41		
17	11	13	17	19	23	29	31	37	41			
17		24	17	19	23	29	31	37	41			
		24	34	19	23	29	31	37	41			
		24	34		42	29	31	37	41			
			34		42	53	31	37	41			
					42	53	65	37	41			
					42	53	65		78			
						95	65		78			
						95			143			
									238			

$\sum w_i l_i$=804

构造过程的特点：

(1) 每个初始结点最终都成为了叶子结点，且权值越小的结点到根结点的路径长度越大。

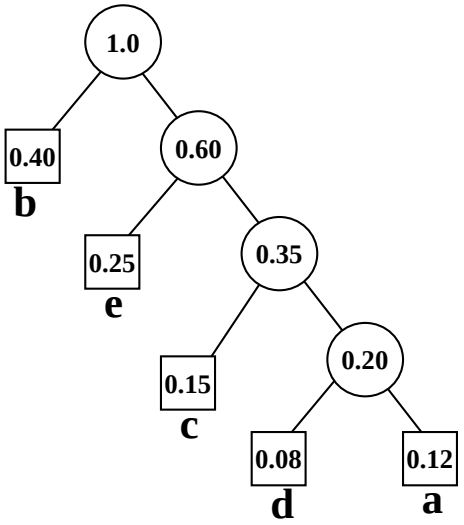
(2) 在哈夫曼树中，权值大的结点离根最近。

(3) 每次构造都选择2棵树作为新结点的孩子，因此哈夫曼树不存在度为1的结点。

(4) n个结点Huffman树构造过程中，共新建了n-1个结点（双分支结点，合并），因此哈夫曼树的结点总数2n-1；

哈夫曼树的构造过程

字符	概率
a	0.12
b	0.40
c	0.15
d	0.08
e	0.25




```
typedef  struct
 {
    float weight;
    int lchild, rchild, parent;
} HTNODE;

typedef HTNODE
HuffmanT[m];
```

n=5

	weight	parent	lchild	rchild
0	0.12	-1	-1	-1
1	0.40	-1	-1	-1
2	0.15	-1	-1	-1
3	0.08	-1	-1	-1
4	0.25	-1	-1	-1
5	-	-1	-1	-1
6	-	-1	-1	-1
7	-	-1	-1	-1
8	-	-1	-1	-1

	weight	parent	lchild	rchild
0	~~0.12~~	5	-1	-1
1	~~0.40~~	8	-1	-1
2	~~0.15~~	6	-1	-1
3	~~0.08~~	5	-1	-1
4	~~0.25~~	7	-1	-1
5	~~0.20~~	6	3	0
6	~~0.35~~	7	2	5
7	~~0.60~~	8	4	6
8	1.00	-1	1	7

m=9

n ? m

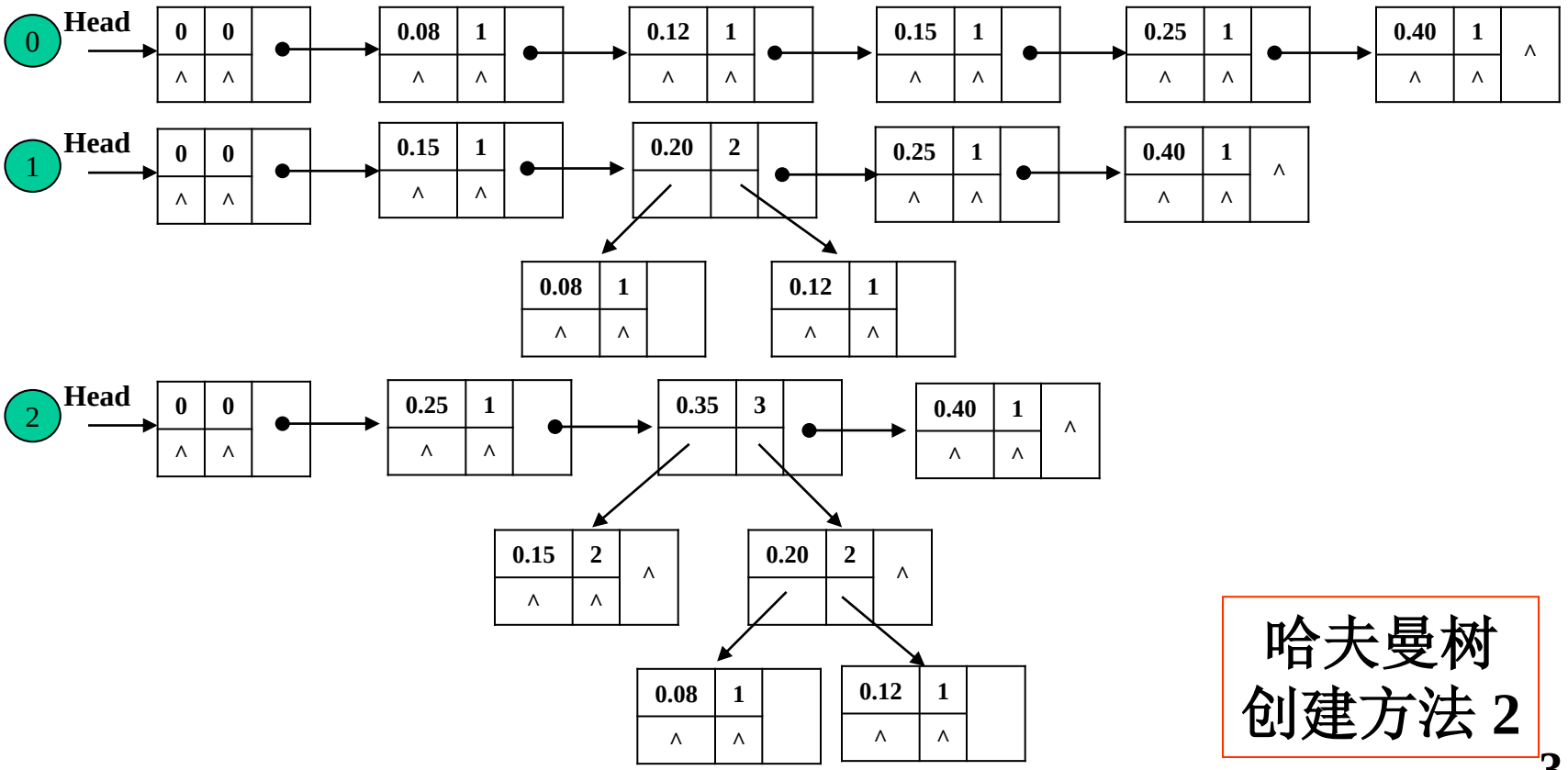
```
void  SelectMin(HuffmanT T, int n1, int *p1, int *p2)
{  int i,j;
   for(i=0;i<=n1;i++)      if(T[i].parent==-1) { *p1=i; break;}
   for(j=i+1;j<=n1;j++)    if(T[j].parent==-1) { *p2=j;break;}
   for(i=0;i<=n1;i++)
     if((T[*p1].weight>T[i].weight)&&(T[i].parent==-1)&&(*p2!=i))  *p1=i;
   for(j=0;j<=n1;j++)
     if((T[*p2].weight>T[j].weight)&&(T[j].parent==-1)&&(*p1!=j))  *p2=j;
}

   void  CreatHT(HuffmanT T)    //创建哈夫曼树
   {
     int i,p1,p2;        InitHT(T);
     for(i=n; i<m; i++)
     {     SelectMin(T, i-1, &p1, &p2);      //选择两个最小的权
           T[p1].parent=T[p2].parent=i;
           T[i].lchild=p1;
           T[i].rchild=p2;
           T[i].weight=T[p1].weight+T[p2].weight;      }
   }
```

哈夫曼树
创建方法 1

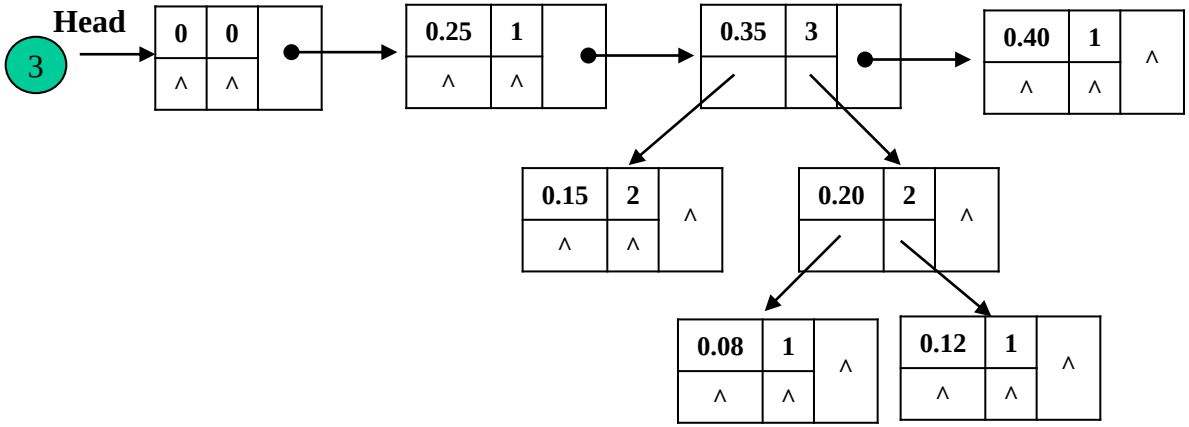
```
typedef struct HNODE
 {
    int data,lev;
        struct HNODE *next,*lchild,*rchild;
} HTREE;
```

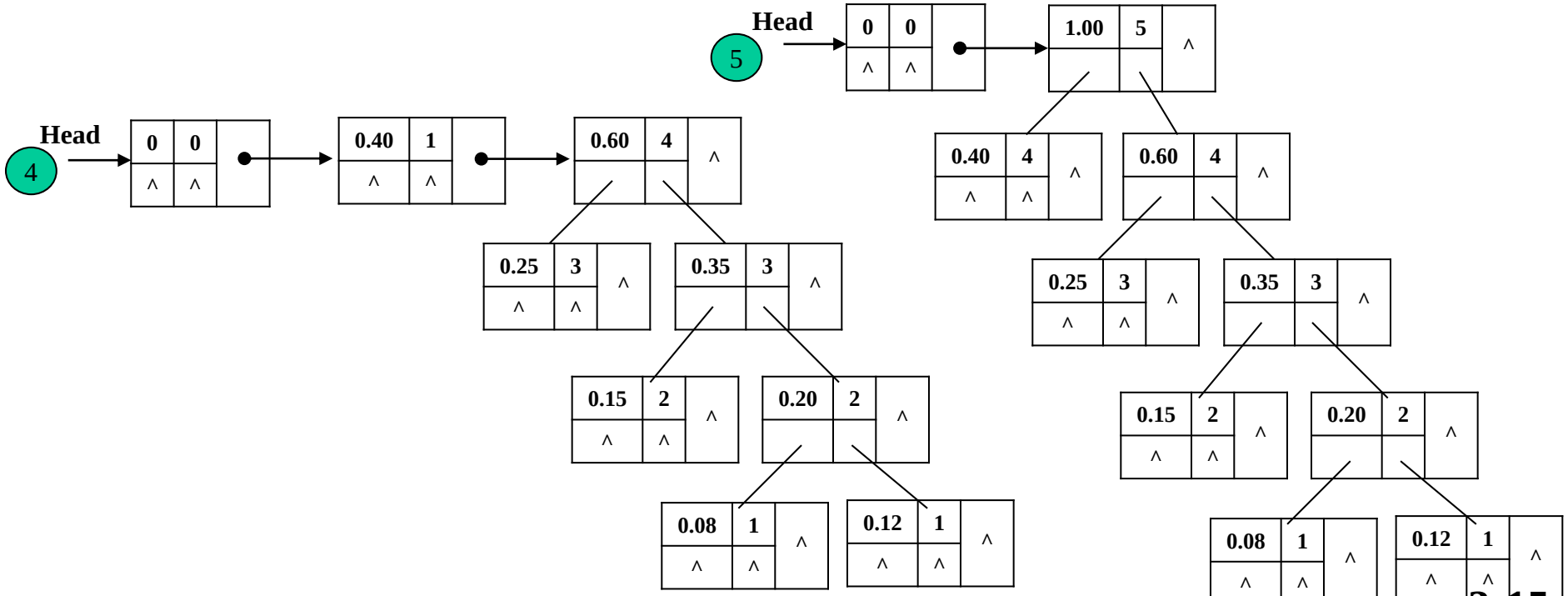
data	lev	next
lchild	rchild	

字符	a	b	c	d	e
概率	0.12	0.40	0.15	0.08	0.25



哈夫曼树
创建方法 2


3
Head
0 0
0.25 1
0.35 3
0.40 1
0.15 2
0.20 2
0.08 1
0.12 1


5
Head
0 0
1.00 5
0.40 4
0.60 4
0.25 3
0.35 3
0.15 2
0.20 2
0.08 1
0.12 1
4
Head
0 0
0.40 1
0.60 4
0.25 3
0.35 3
0.15 2
0.20 2
0.08 1
0.12 1

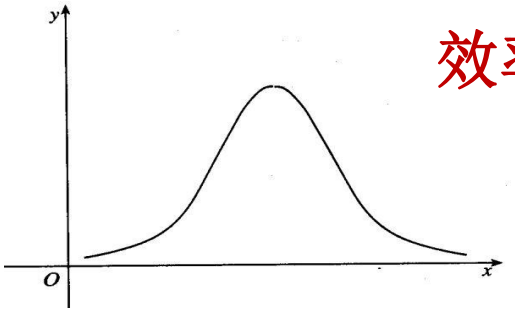
```
void Huffman(HTREE *H)     //创建哈夫曼树
{   HTREE *p,*q,*r;
    while(H->next->next!=Null)
   {
        p=H->next;
        q=H->next->next;
        H->next=q->next;
        r=(HTREE *)malloc(sizeof(struct HNODE));
        if(!r) {   printf("***内存错误!\n");   exit(0); }
        r->data=p->data+q->data;
        r->lev=(p->lev>q->lev?p->lev+1:q->lev+1);
        p->lev=q->lev=r->lev-1;
        r->lchild=p;
        r->rchild=q;
       Insert(H,r);
   }
}
```


```
void Insert(HTREE  *&H, HTREE*q)
 //向哈夫曼树中插入一个结点，保持权的和有序
{     HTREE *p;
      p=H;
      while((p->next!=Null)&&(p->next->data<=q->data))
            p=p->next;
      q->next=p->next;
      p->next=q;
}
```

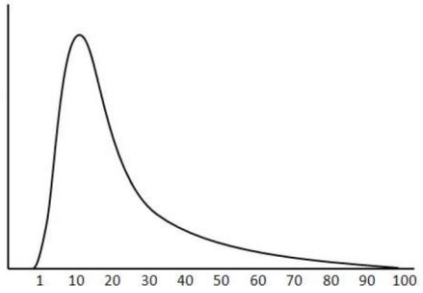
例：输入一批学生成绩，将百分制转换成五级分制。

分数	0~59	60~69	70~79	80~89	90~100
等级	Fail	Pass	General	Good	Excellent

```
scanf("%d",&a);
while(a!=999)
{
    if (a<60) b="Fail"
     else if (a<70)  b="Pass"
     else if (a<80) b="General"
     else if(a<90) b="Good"
     else b="Excellent";
     printf("%s", b);
     scanf("%d",&a);
}
```



效率如何？



若同时已知

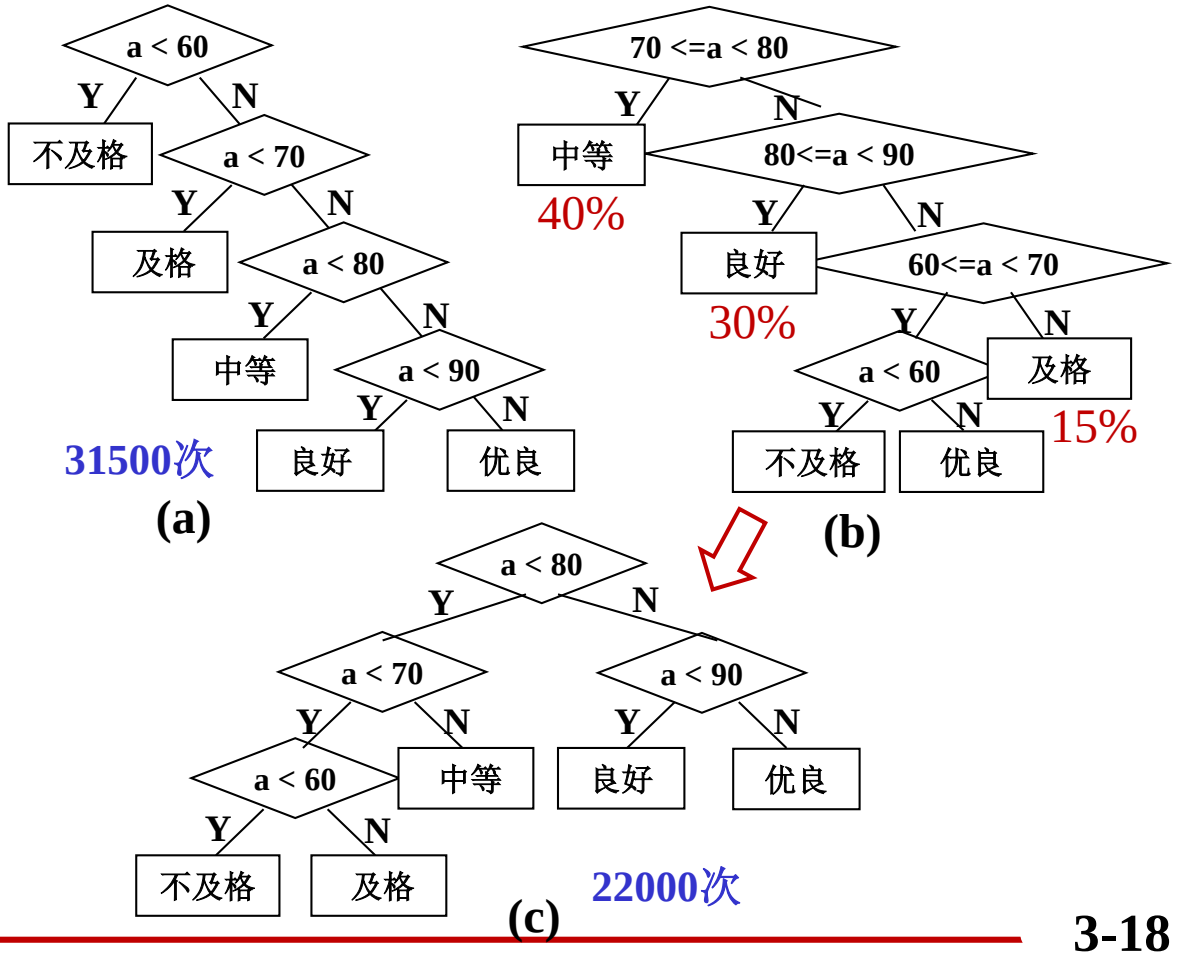
分数	0~59	60~69	70~79	80~89	90~100
等级	Fail	Pass	General	Good	Excellent
概率	0.05	0.15	0.40	0.30	0.10

则： 10000个分数

W_i={5,15,40,30,10}为权；

构造哈夫曼树如图(b)所示；

将判定框中的条件分开，可得到(c)。



哈夫曼树应用—最优编码（Huffman编码）

问题的提出：

哈 2594
尔 2291
滨 1785
工 2504
业 5024
大 2083
学 4907

啊1601阿1602吖6325嗄6436腌7571锕7925埃1603挨1604哎1605唉1606哀1607皑1608癌1609蔼1610矮1611
6441嫒7040瑗7208暖7451碍7733嶝7945嗌8616鞍1616氨1617安1618俺1619按1620暗1621岸1622胺1623案
7281铵7907鹌8038黯8786肮1625昂1626盎1627凹1628敖1629熬1630翱1631袄1632傲1633奥1634懊1635澳
6959媪7033鳌7081嫠7365聱8190螯8292鳌8643鳌8701鏖8773芭1637捌1638扒1639叭1640吧1641笆1642八
1649耙1650坝1651霸1652罢1653爸1654茇6056菝6135岜6517灞6917钯7857粑8446鲅8649魃8741白1655柏
1662摆6267呗6334掰7494斑1663班1664搬1665扳1666般1667颁1668板1669版1670扮1671拌1672伴1673瓣
7851瘢8103癍8113舨8418邦1678帮1679梆1680榜1681膀1682绑1683棒1684磅1685蚌1686镑1687傍1688谤
1693剥1694薄1701雹1702保1703堡1704饱1705宝1706抱1707报1708暴1709豹1710鲍1711爆1712葆6165孢
1713碑1714悲1715卑1716北1717辈1718背1719贝1720钡1721倍1722狈1723备1724惫1725焙1726被1727孛
6703碚7753鹌8039褙8156鐾8645鞴8725奔1728苯1729本1730笨1731崩5946绷5948甭7458镚7928崩1732绷
7420逼1738鼻1739比1740鄙1741笔1742彼1743碧1744蓖1745蔽1746毕1747毙1748毖1749币1750庇1751痹
1758臂1759避1760陛1761匕5616俾5734萆6074荸6109薛6221吡6333哔6357狴6589庳6656愎6725滗6868潟
7815铋7873秕7985裨8152鼻8357箅8375篦8387舭8416襞8437跸8547髀8734鞭1762边1763编1764贬1765扁
1772遍1773匾5650弁5945苄6048忭6677汴6774缏7134飚7614煸7652砭7730碥7760窆8125褊8159蝙8289笾
7027骠7084柫7228飘7609螵7613镖7958镳7980瘭8106裱8149鳔8707彬8752濒1778傧1779别1780瘪1781瘪
1787傧5747豳6557缤7145玢7167槟7336殡7375膑7587镔7957髌8738鬓8762兵1788冰1789柄1790丙1791秉

编码（如电报码）{ 等长编码 / 不等长编码

特点：{ 编码长度 / 译码速度 / 传输速度

前缀编码

任何一个字符的编码都不是同一字符集中另一个字符的编码的前缀，即没有一个编码是另一个编码的前缀。

举例 A：0 B：110 C：10 D：111

发送方：将ABACCDA 转换成 0110010101110 发出

接收方： 0110010101110；破译：所得的译码是唯一的。

等长编码

这类编码二进制串的长度取决于电文中不同的字符个数。

举例 要传输的原文为ABACCDA

等长编码 A：00 B：01 C：10 D：11

发送方：将ABACCDA 转换成 00010010101100

接收方：将 00010010101100 还原为 ABACCDA

不等长编码

各个字符的编码长度不等。

举例 A：0 B：00 C：1 D：01 非前缀码

发送方：将ABACCDA 转换成 000011010

接收方：000011010 如何破译？

不等长编码的好处

可以使传送电文的字符串总长度尽可能地短。对出现频率高的字符采用尽可能短的编码，则传送电文的总长便可减少。

方法 利用哈夫曼树构造一种不等长二进制编码，并且构造所得的哈夫曼编码是一种最优前缀编码，使所传电文总长度最短。

举例 消息由(a、b、c、d、e)5个字符组成，已知各字符出现的概率分别为(0.12、0.40、0.15、0.08、0.25)。

现用一个二进制数字串对每个字符进行编码，使任意一个字符的编码不会是任何其他字符编码的前缀，满足“前缀性”。

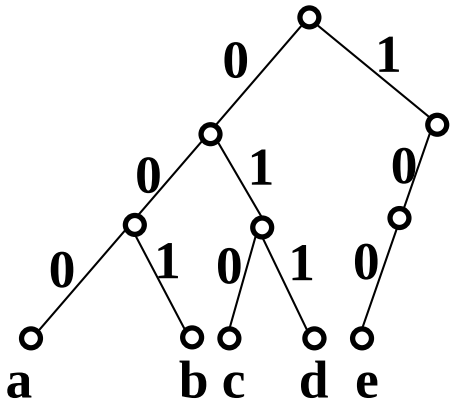
表3-3 两种编码

字符	概率	code1	code2
a	0.12	000	000
b	0.40	001	11
c	0.15	010	01
d	0.08	011	001
e	0.25	100	10

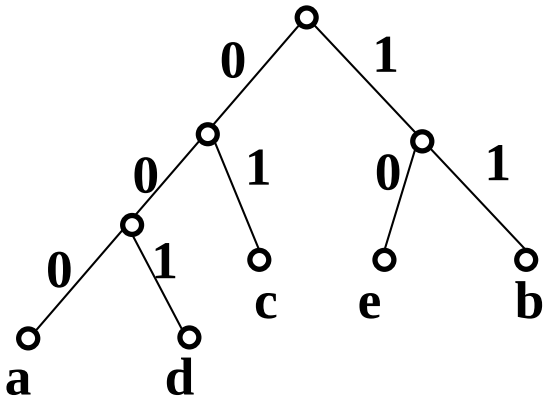
例如：
二进制串**001010011**
按**code1**编码：**bcd**

二进制串**1101001**
按**code2**编码：**bcd**

将每个结点的左分支赋以0，右分支赋以1，将字符作为叶结点的标号。



code1的二叉树



code2的二叉树

code1的平均长度=3×(0.12+0.40+0.15+0.08+0.25)=3.0

code2的平均长度=3×(0.12+0.08)+2×(0.40+0.15+0.25)=2.2

问题：对于给定的字符集以及这些字符出现的频率，如何求一种有前缀性的编码，使字符编码的平均长度最小。

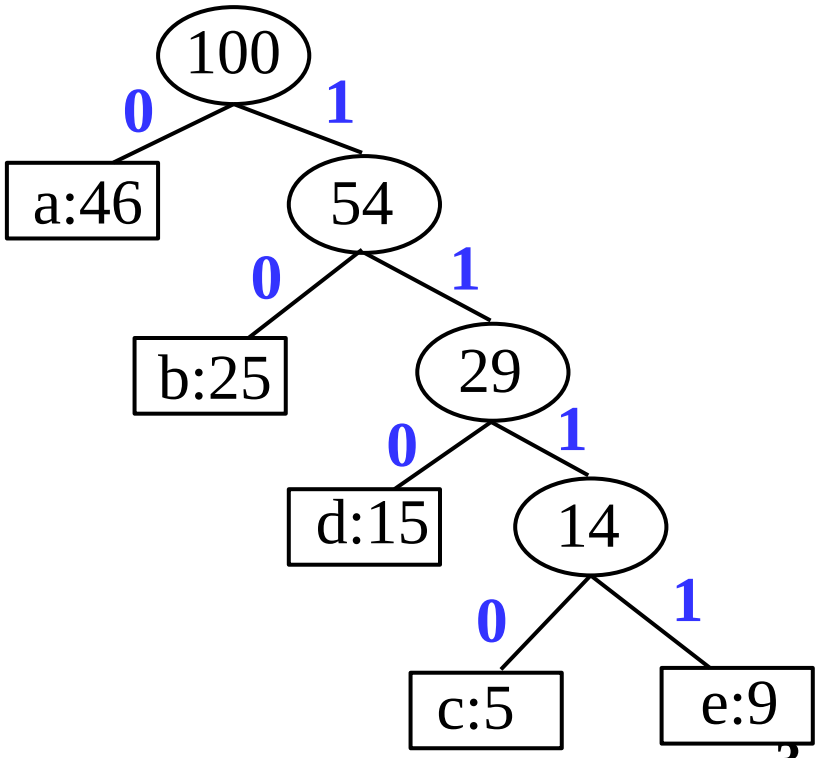
哈夫曼编码的构造过程

（1）将每个出现的字符作为一个独立的结点，权值为出现的次数（频度）；

（2）编码：构造哈夫曼树，并获取从根到当前字符结点路径上边标记的序列；

（3）标记：边标记为**0**表示“转向左孩子”，边标记为**1**表示“转向右孩子”。

a:46 b:25 c:5 d:15 e:9



各字符编码为

a: 0

b: 10

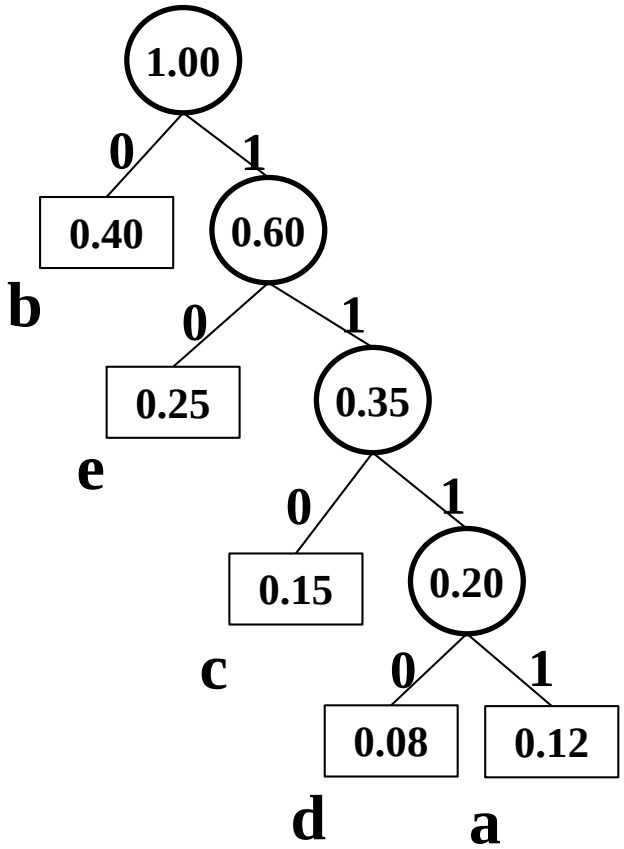
c: 1110

d: 110

e: 1111

采用哈夫曼树来进行编码？ **Huffman编码从上到下编码**

以(a、b、c、d、e)5个字符出现的频率为权，构造哈夫曼树，即： w_i=(0.12、0.40、0.15、0.08、0.25)



哈夫曼编码code3

字符	概率	code3
a	0.12	1111
b	0.40	0
c	0.15	110
d	0.08	1110
e	0.25	10

Code3编码的平均长度，即：

$$\sum w_i l_i = 2.15$$

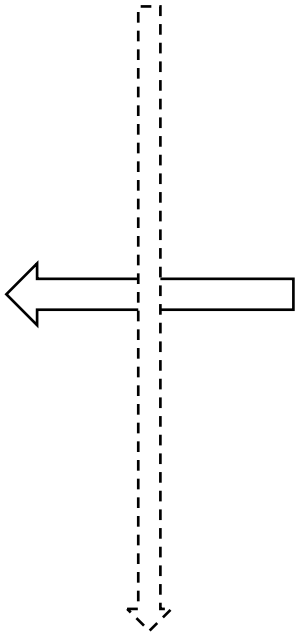
bcd的编码为：01101110

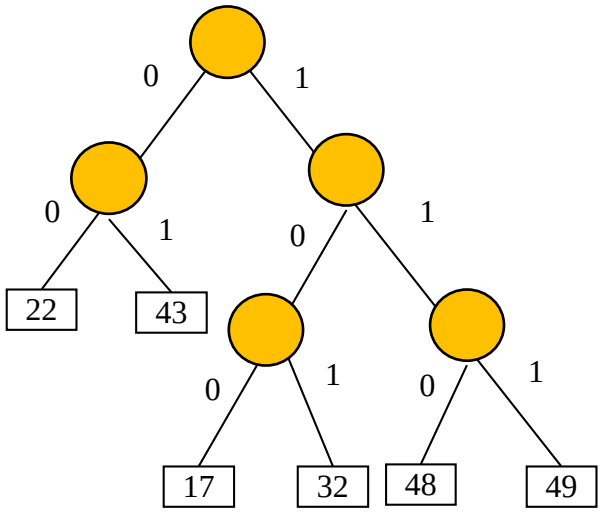
哈弗曼编码是一种广泛应用且非常有效的数据压缩编码。

【例3-23】某文件内一组原始数据：

32	22	22	43	49	22	22	17	48	43

符号	频率	编码
22	4	00
43	2	01
17	1	100
32	1	101
48	1	110
49	1	111





原始数据的代码转换：

101	00	00	01	111	00	00	100	110	01

国际流行两种图像压缩编码标准：

- 静态图像进行压缩JPEG(Joint photographic Experts Group)
- 动态图像进行压缩MPEG(Moving Picture Experts Group)

在多媒体技术如视频信号的压缩技术中用到了哈夫曼编码。

哈夫曼编码是一种无失真编码，即对源数据压缩后形成的编码，进行恢复时，可完全恢复源数据，但它对静态的数据是可行的。

- 自适应哈夫曼编码。

3.7.3 二叉排序树

【定义】

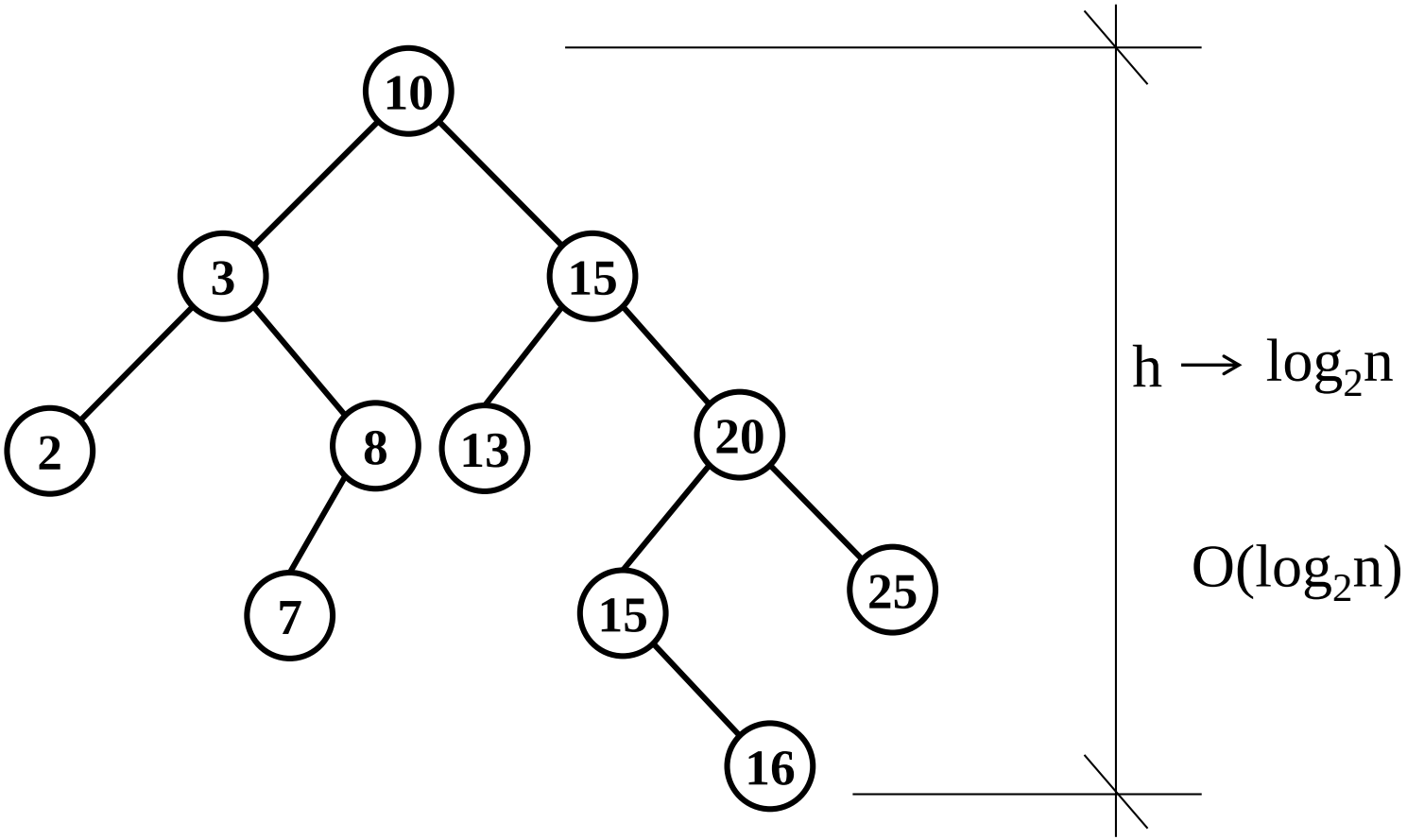
二叉排序树或者是一棵空树，或者是具有下列性质的二叉树：

（1）若它的左子树非空，则左子树上的所有结点的值均小于它的根结点的值；

（2）若它的右子树非空，则右子树上的所有结点的值均大于或等于它的根结点的值；

（3）它的左右子树分别为二叉排序树。

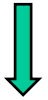
S= { 10，15，3，2，8，13，20，15，16，25，7}



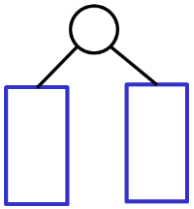
LDR= { 2，3，7，8，10，13，15，15，16，20，25}

查找复杂度？

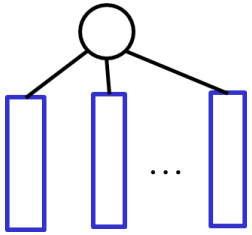
二叉树排序树

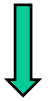


平衡二叉树AVL

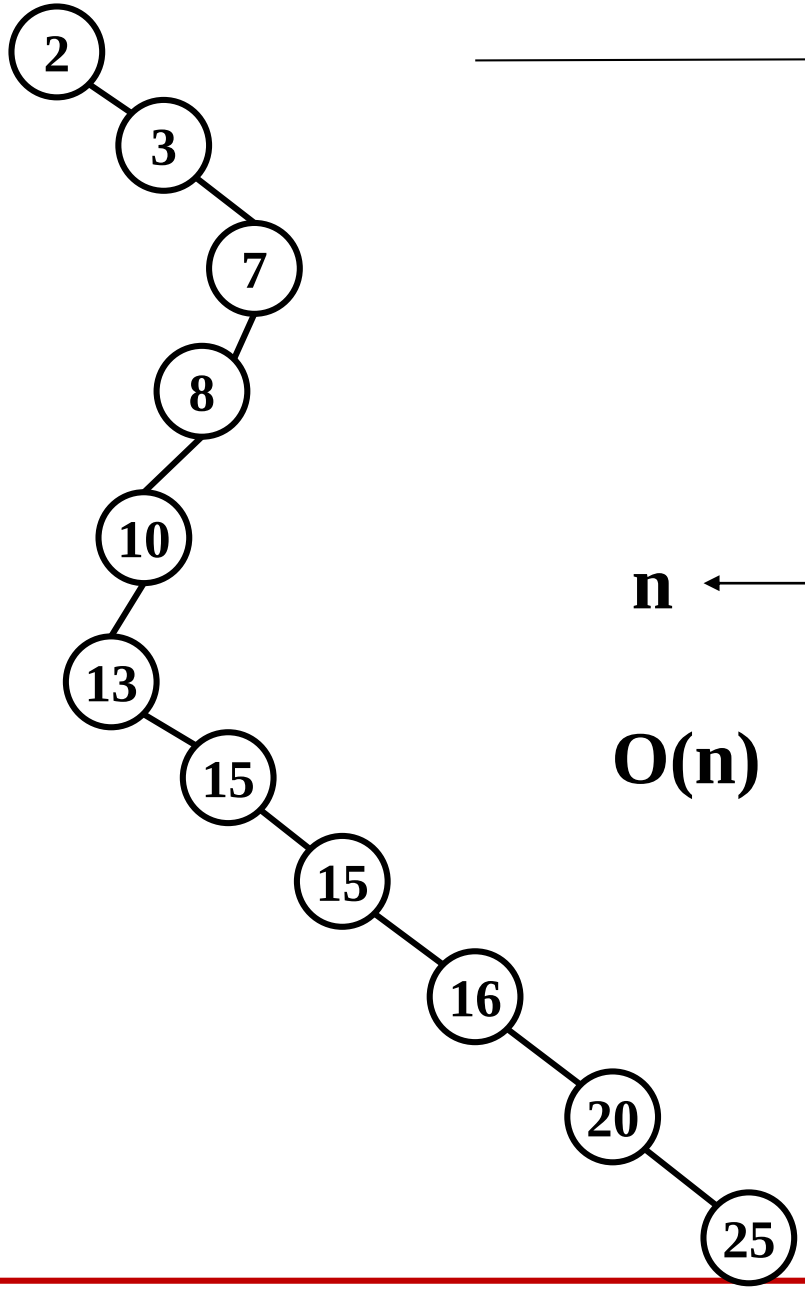


m路查找树





B树

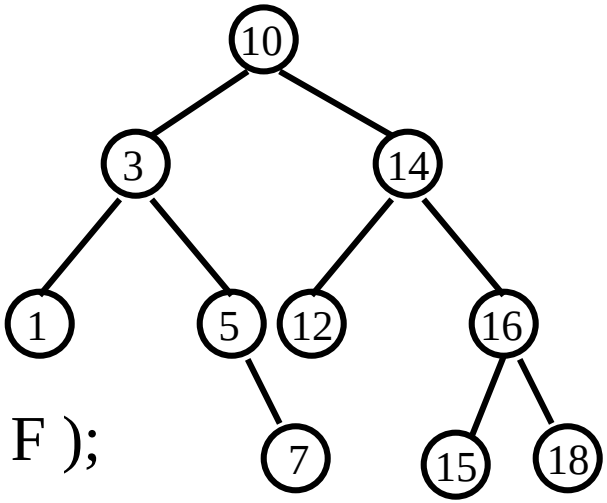


n ← h

O(n)

二叉排序树

```
Struct  CellType  {
        Records   data ;
        CellType  *lchild,*rchild ;}
Typedef   Celltype  *  BST ;
```



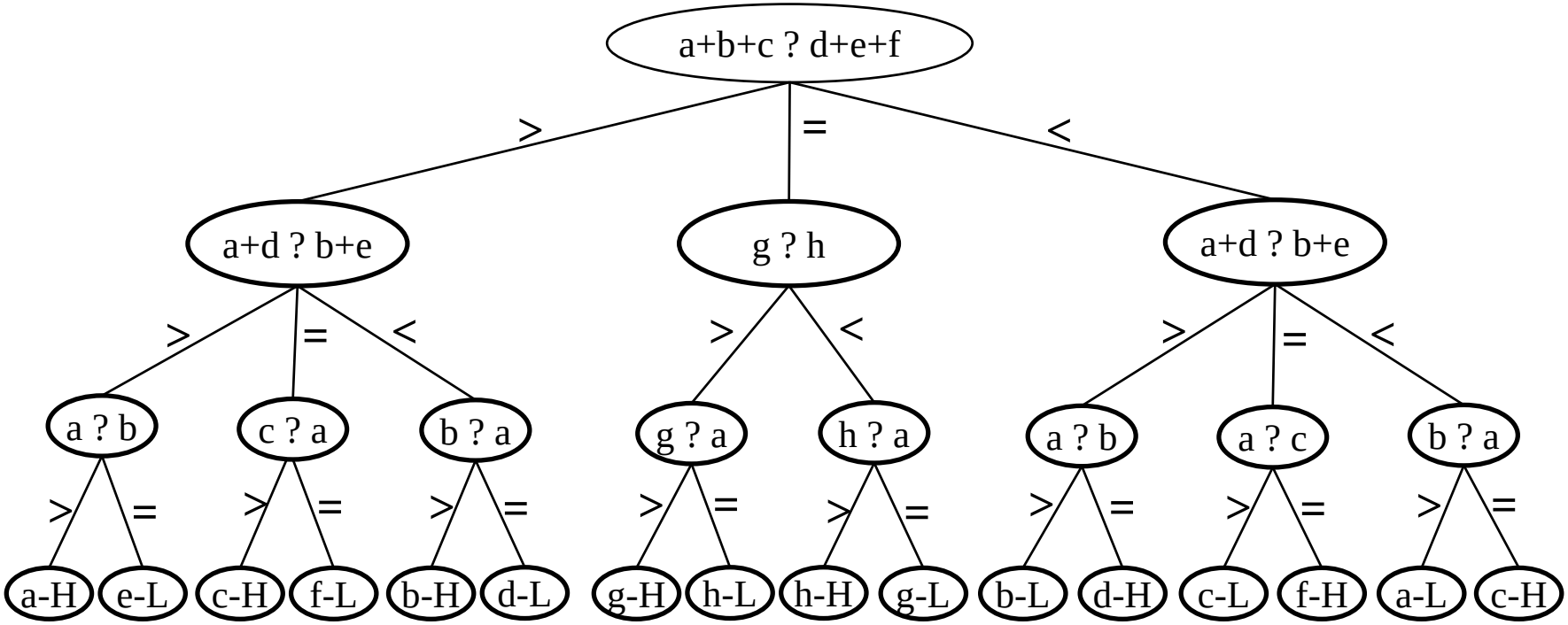

```
BST   Search( keytype  k, BST  F );
{   p = F ;
    if ( p == Null )
          return Null ;
    else if ( k == p->data.key )
          return p ;
    else if ( K < p->data.key )
          return ( search ( k,  p->lchild ) ) ;
    else if ( K > =p->data.key )
          return ( search ( k,  p->rchild ) ) ;
}
```

在二叉查找树中插入新结点：

```
Void  Insert ( Records  R , BST   *&F )
{
   if ( F ==Null )
      {  F = New  CellType ;
         F->data = R ;
         F->lchild = Null ;
         F->rchild = Null ;   }
   else  if ( R.key < F->data.key )
         Insert ( R , F->lchild )
   else  if ( R.key >= F->data.key )
         Insert ( R , F->rchild )
}
```

3.7.4 树的应用—判定树

假定有八枚硬币a、b、c、d、e、f、g、h，已知其中1枚是伪造的假币，假币的重量与真币不同，或重或轻。要求以天平为工具，用最少的比较次数挑出假币。



H—假币重于真币；L—假币轻于真币。

```
void EightCoins( ) //输入8枚硬币的重量，经过三次比较，找出其中的伪币
{
  int a,b,c,d,e,f,g,h;
  cin>>a>>b>>c>>d>>e>>f>>g>>h;
  switch(Compare(a+b+c,d+e+f)){
   case '=':if(g>h)Comp(g,h,a);
            else Comp(h,g,a);
            break;
   case '>':switch(Compare(a+d,b+e)){
              case '=':Comp(c,f,a);break;
              case '>':Comp(a,e,b);break;
              case '<':Comp(b,d,a);break;
            }
           break;
   case '<':switch(Compare(a+d,b+e)){
              case '=':Comp(f,c,a);break;
              case '>':Comp(d,b,a);break;
              case '<':Comp(e,a,b);break;
            }
            break;
   }
}//EightCoins
```

```
char Comppare(int a,int b)
 //比较两组硬币的轻重
{
     if(a<b)
        return('<');
     else if(a==b)
        return('=');
      else
        return('>');
}//Compare
```

```
void Comp(int x,int y,int z)
 //将x与标准硬币z进行比较
{
  if(x>z)
       cout << x<<"Heavy!";
  else
       cout <<y<<"Light!";
}//Comp
```

判定树的特点：

① 一个判定树是一个算法的描述；

② 每个内部结点对应一个部分解；

③ 每个叶子（外部结点）对应一个解；

④ 每个内部结点连接与一个获得新信息的测试；

⑤ 从每个结点出发的分支标记着不同的测试结果；

⑥ 一个求解过程对应于从根到叶的一条路；

⑦ 一个判定树是所有可能的解的集合。

- 说有六个球四个质量一样的普通球和两个一轻一重的球轻球和重球质量和等于两个普通球质量 现在给你一架天平让你称三次找出轻球和重球；

- 12个球中有1个次品，在一个没有砝码的天枰上，只称3次，找出该球，并判断该球比其它球轻还是重；

- 有4堆外表上一样的球，每堆4个。已知其中三堆是正品、一堆是次品，正品球每个重10克，次品球每个重11克，请你用天平只称一次，把是次品的那堆找出来；

- 有27个外表上一样的球，其中只有一个是次品，重量比正品轻，请你用天平只称三次（不用砝码），把次品球找出来；

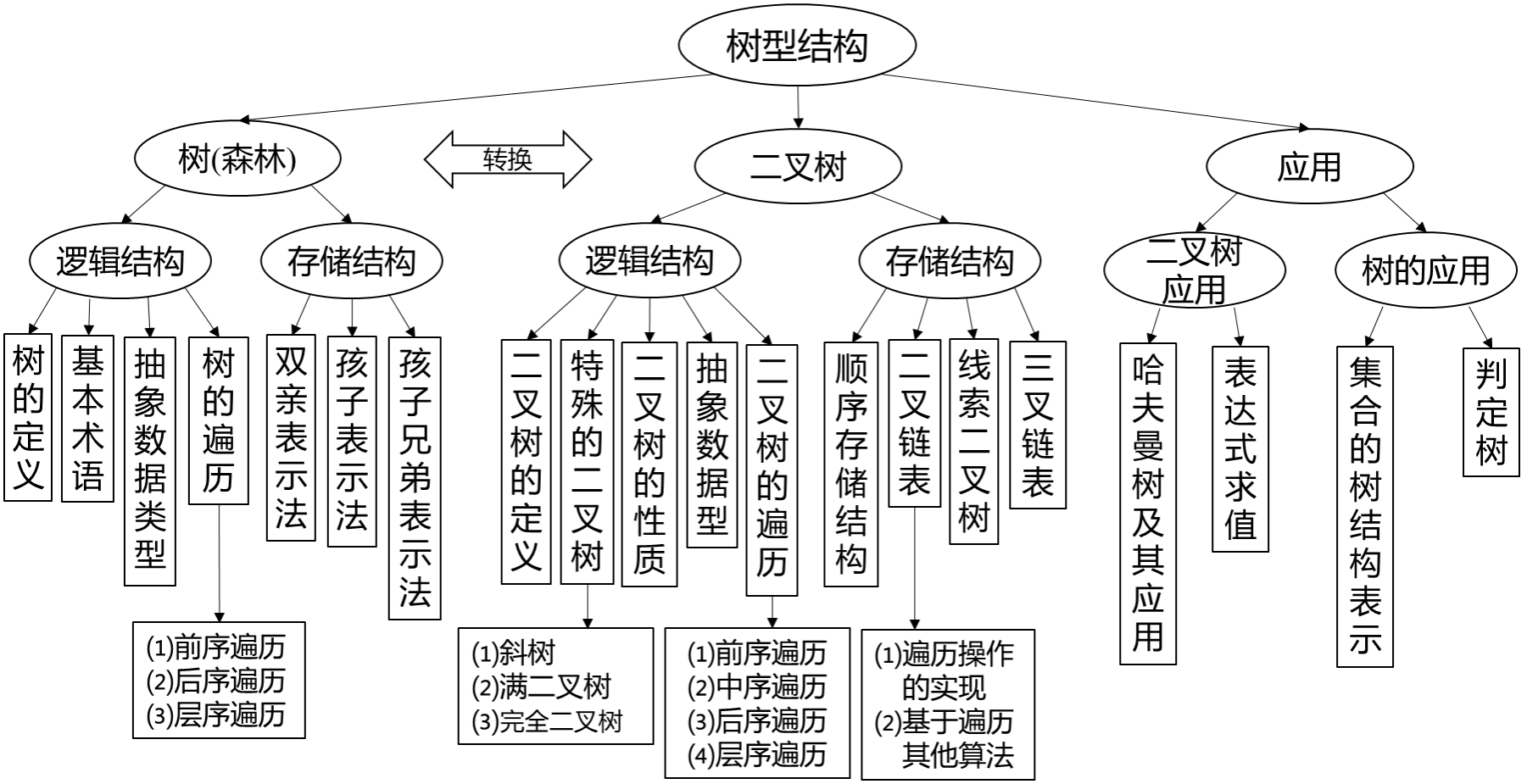
- 现有一架无码天平和m个球，这m个球中有m-1个标准球和一个坏球，坏球或比标准球重，或比标准球轻。规定只准使用n次天平。 求证：（1）m=1/2（3^n-3）时，必可找到坏球且知其轻重。 （2）m=1/2（3^n-1）时，必可找到坏球但未必知其轻重。

树

- 树的ADT
  - 逻辑结构
  - 存储结构
- 二叉树
  - 逻辑结构
  - 存储结构
    - 顺序存储
    - 二/三叉链表
  - 基本性质
  - 遍历二叉树
    - 先根顺序
    - 中根顺序
    - 后根顺序
    - 层序遍历*

    （中根顺序：递归；层序遍历：非递归*）
  - 线索二叉树
  - 二叉排序树
- 树的存储结构
  - 双亲表示法(数组)
  - 孩子表示法(邻接表)
  - 左右链表示(二叉树)
- 树的应用
  - 哈夫曼树
  - 判定过程

知识点结构


树型结构
树(森林)
转换
二叉树
应用
逻辑结构
存储结构
逻辑结构
存储结构
二叉树应用
树的应用
树的定义
基本术语
抽象数据类型
树的遍历
双亲表示法
孩子表示法
孩子兄弟表示法
二叉树的定义
特殊的二叉树
二叉树的性质
抽象数据型
二叉树的遍历
顺序存储结构
二叉链表
线索二叉树
三叉链表
哈夫曼树及其应用
表达式求值
集合的树结构表示
判定树
(1)前序遍历
(2)后序遍历
(3)层序遍历
(1)斜树
(2)满二叉树
(3)完全二叉树
(1)前序遍历
(2)中序遍历
(3)后序遍历
(4)层序遍历
(1)遍历操作的实现
(2)基于遍历其他算法

例题 将森林转换成对应的二叉树，若在二叉树中，结点u是结点v的父结点的父结点，则在原来森林中，u和v具备哪种关系（）

I 父子关系 II 兄弟关系 III u的父结点与v的父结点互为兄弟 I; II

例题 将森林F转换为对应的二叉树T，则F中叶子结点个数等于(C)

A. T中叶子结点个数 B. T中度为1的结点个数

C. T中左孩子指针为空的结点个数 D. T中右孩子指针为空的结点数

例题 若将一棵树T转化成对应的二叉树BT，则下列对BT的遍历中，其遍历序列与T的后根遍历序列相同的是(B)

A. 先根遍历 B. 中根遍历 C. 后根遍历 D. 层次遍历

例题 已知森林F及与之对应的二叉树T，若F的先根遍历序列是a,b,c,d,e,f，中根遍历序列是b,a,d,f,e,c，则T的后根遍历序列是(C)

A. b,a,d,e,f,c B. b,d,e,e,c,a C. b,f,e,d,c,a D. e,f,d,c,b,a F和T 先、中序对应

例题 若森林F有15条边、25个结点，则F包含树的个数是 (C)

A. 8 B. 9 C. 10 D. 11

N个结点的树有n-1条边

例题 5个字符有如下4种编码方案，不是前缀编码的是( )

A. {01, 0000, 0001, 001, 1} B. {001, 000, 001, 010, 1} **D**

C. {000,001,010,011,100} D. {0, 100, 110, 1110, 1100}

例题 已知字符集{a,b,c,d,e,f},若各字符出现的次数分别为6,3,8,2,10,4, 则对应字符集中各字符的哈夫曼编码可能是( ) **A**

A. {00, 1011, 01, 1010, 11, 100} B. {00, 100, 110, 000, 0010, 01}

C. {10, 1011, 11, 0011, 00, 010} D. {0011, 10, 11, 0010, 01, 000}

例题 已知字符集{a,b,c,d,e,f,g,h}，若各字符的哈夫曼编码依次是0100, 10, 0000, 0101, 001, 011, 0001，则编码序列0100110010010111101 01 的译码结果是 ( ) **D**

A. {a c g a b f h} B. {a d b a g b b} C. {a f b e a g d} D. {a f e e f g d}

例题 对n个互不相同的符号进行哈夫曼编码。若生成的哈夫曼树一共有115个结点，那么n的值是( ) **C**

A. 56 B. 57 C. 58 D. 60

思考题

若任意一个字符的编码都不是其他字符编码的前缀，则称这种编码具备前缀特性。现有某字符集（字符个数>=2）的不等长编码，每一个字符的编码均为二进制的0、1序列，最长为L位，且具有前缀特性。请回答下列问题：

1）哪种数据结构适宜保存上述具有前缀特性的不等长编码？

2）基于你所设计的数据结构，简述从0/1串到字符串的译码过程。

3）简述判定某字符集的不等长编码是否具有前缀特性的过程。