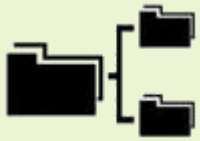


本节主题：

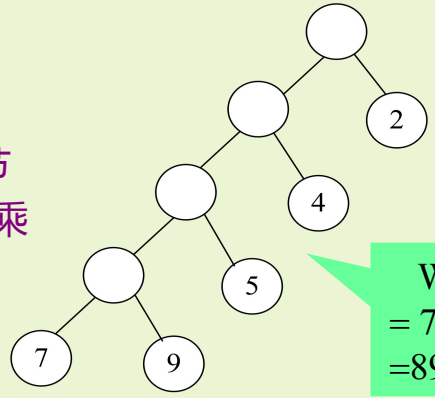
哈夫曼树

哈夫曼树的定义

- 带权路径长度
  - 设二叉树具有n个带**权值**的叶子节点，那么从根节点到各个叶子节点的路径长度与相应节点权值的乘积的和，叫做二叉树的**带权路径长度(WPL)**。

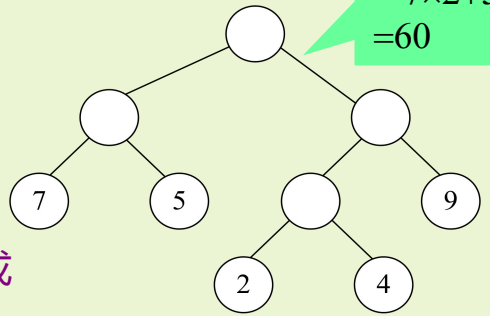
$$WPL=\sum_{i=1}^{n} w_i l_i$$

    - n表示叶子节点的数目
    - w_i表示叶子节点k_i的权值
    - l_i分别表示根到k_i之间的路径长度（即从叶子节点到达根节点的分支数）。
- 哈夫曼树
  - 具有最小带权路径长度的二叉树称为**哈夫曼树**，或称为**最优二叉树**。



WPL(T)
= 7×4+9×4+5×3+4×2+2×1
=89

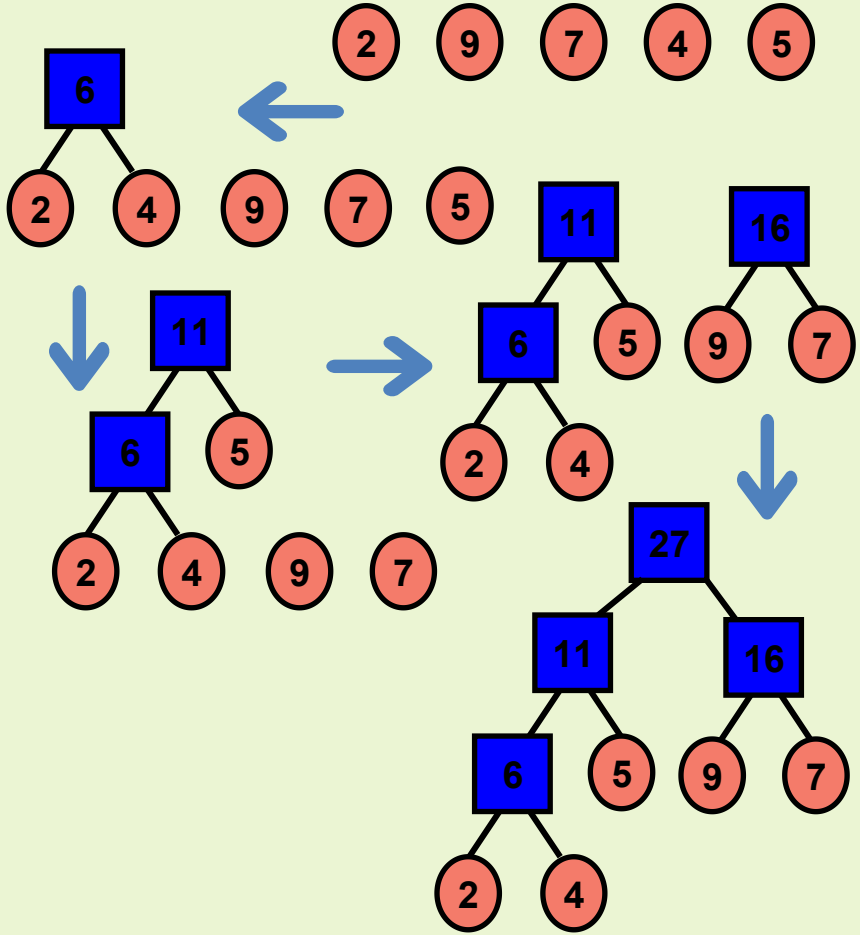
WPL(T)
= 7×2+5×2+2×3+4×3+9×2
=60



n个叶节点的二叉树，共有2n-1个节点。

构造哈夫曼树 $\mathrm{WPL}=\sum_{i=1}^{n} w_i l_i$

- 策略
  - 要使WPL值最小，须使权值越大的叶子节点越靠近根节点，而权值越小的叶子节点越远离根节点。
- 方法
  - （1）给定的n个权值$\{W_1,W_2,\ldots,W_n\}$构造n棵只有一个叶子节点的二叉树，从而得到一个二叉树的集合$F=\{T_1,T_2,\ldots,T_n\}$；
  - （2）在F中选取根节点的权值最小和次小的两棵二叉树作为左、右子树，构造一棵新的二叉树，新的二叉树根节点的权值为其左、右子树根节点权值之和；
  - （3）在集合F中删除作为左、右子树的两棵二叉树，并将新建立的二叉树加入到集合F中；
  - （4）重复（2）、（3）两步，当F中只剩下一棵二叉树时，这棵二叉树便是所要建立的哈夫曼树。



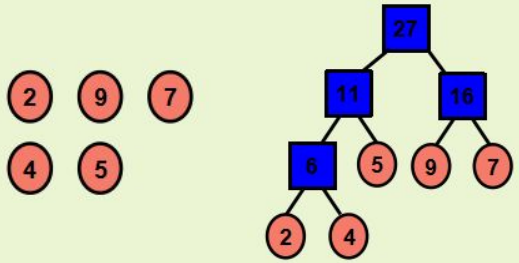
构造哈夫曼树存储及生成算法

```
#define N  5  //叶子节点数
typedef struct
{
   char data;     //节点值
   float weight; //权重
   int parent;    //双亲节点
   int lchild;     //左孩子节点
   int rchild;     //右孩子节点
} HTNode;
HTNode ht[2*N-1];
```

step 1. n个叶子节点只有data和weight域值，所有2n-1个节点的parent、lchild和rchild域置为初值-1。

step 2. 重复处理每个非叶子节点ht[i]（ht[n]～ht[2n-2]，i: n ～2n-2）：

- 从ht[0] ～ht[i-1]中找出根节点（即其parent域为-1）最小的两个节点ht[lnode]和ht[rnode]
- ht[lnode]和ht[rnode]作为左右子树，增加它们的双亲节点ht[i]，有：ht[i].weight= ht[lnode].weight +ht[rnode].weight



ht	dat	wei	par	lch	rch
[0]		2	5	-1	-1
[1]		9	7	-1	-1
[2]		7	7	-1	-1
[3]		4	5	-1	-1
[4]		5	6	-1	-1
[5]		6	6	0	3
[6]		11	8	4	5
[7]		16	8	1	2
[8]		27	-1	6	7

算法实现

```
void CreateHT(HTNode ht[], int n)
{
   int i,j,k,lnode,rnode;
   float min1,min2;
   for (i=0; i<2*n-1; i++)
      ht[i].parent=ht[i].lchild=ht[i].rchild=-1;
   for (i=n; i<2*n-1; i++)
   {
      //找出权重最小的两个节点
      //生成一个双亲节点
   }
}
```

```
ht[lnode].parent=i;
ht[rnode].parent=i;
ht[i].weight=ht[lnode].weight+ht[rnode].weight;
ht[i].lchild=lnode;
ht[i].rchild=rnode;
```

```
min1=min2=32767;
lnode=rnode=-1;
for (k=0; k<=i-1; k++)
   if (ht[k].parent==-1)
   {
      if (ht[k].weight<min1)
      {
         min2=min1;
         rnode=lnode;
         min1=ht[k].weight;
         lnode=k;
      }
      else if (ht[k].weight<min2)
      {
         min2=ht[k].weight;
         rnode=k;
      }
   } //if
```

基本算法：在给定的n个数据中，找出其最小值和次小值。

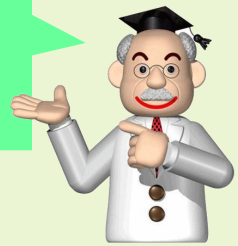
ht	dat	wei	par	lch	rch
[0]		2	5	-1	-1
[1]		9	7	-1	-1
[2]		7	7	-1	-1
[3]		4	5	-1	-1
[4]		5	6	-1	-1
[5]		6	6	0	3
[6]		11	8	4	5
[7]		16	8	1	2
[8]		27	-1	6	7

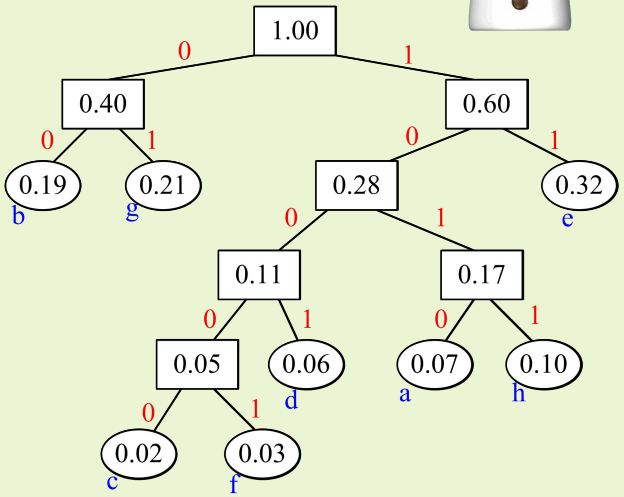
应用：哈夫曼编码

- 问题：符号编码问题
- 举例
  - 用于通信的电文（或一幅图像），有8个符号(设为'a'~'h')
  - 各符号出现的概率分别为：
    0.07、0.19、0.02、0.06、0.32、0.03、0.21、0.10
  - 如何确定各符号的二进制编码，使编码后的代码长度最短？
- 解决方法1
  - 等长编码：a:000 b:001 c:010 d:011 e:100 f:101 g:110 h:111
  - 平均码长：3
- 解决方法2
  - 哈夫曼非等长编码：
    a:1010 b:00 c:10000 d:1001 e:11 f:10001 g:01 h:1011
  - 平均码长 $\sum_{i=1}^{8} w_i l_i$

$$= 0.07 \times 4 + 0.19 \times 2 + 0.02 \times 5 + 0.06 \times 4 + 0.32 \times 2 + 0.03 \times 5 + 0.21 \times 2 + 0.1 \times 4$$
$$= 2.61$$

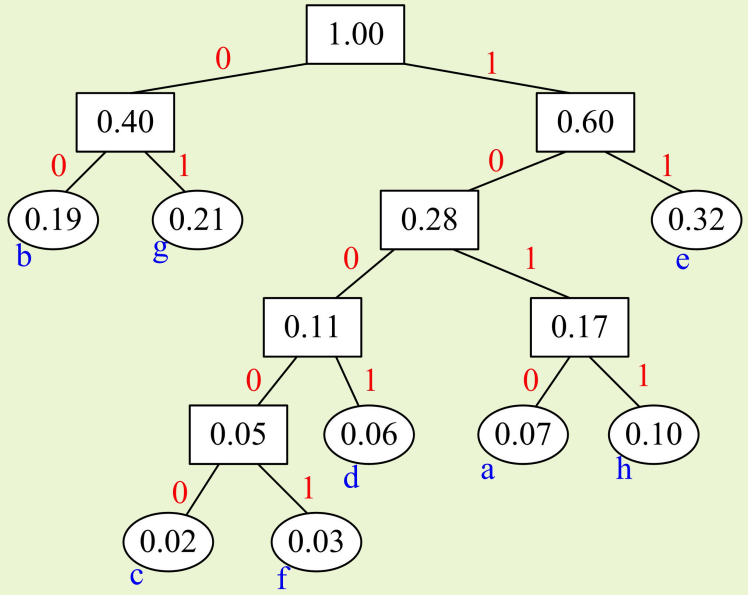
原理：使用频率最高的符号，使其编码长度越短。





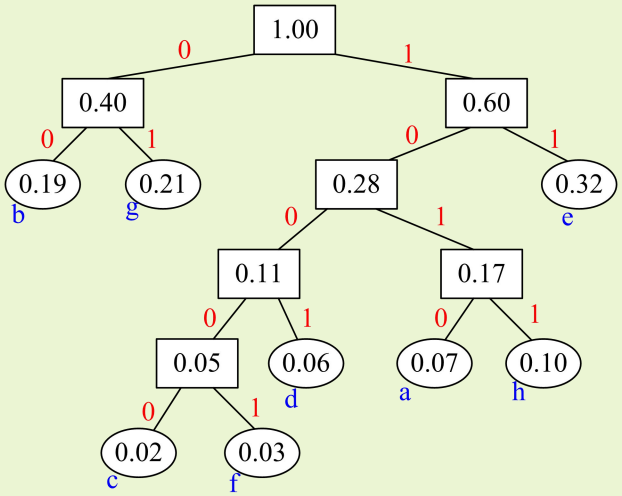
哈夫曼编码构造方法

- 设需要编码的字符集合为{$d_1,d_2,...,d_n$}，各个字符在电文中出现的次数集合为{$w_1,w_2,...,w_n$}。
- 以$d_1,d_2,...,d_n$作为叶节点，以$w_1,w_2,...,w_n$作为权值构造一棵二叉树。
- 规定哈夫曼树中的左分支为0，右分支为1，则从根节点到每个叶节点所经过的分支对应的0和1组成的序列便为该节点对应字符的编码。
- 特点
  - 一种有效的编码方法
  - 形成的编码不是惟一的
  - 当信源概率相等时，效率最低
  - 编码后，形成一个Huffman编码表，解码时需要参照该表。



哈夫曼编码的实现

```
//存放每个节点哈夫曼编码的类型
typedef struct
{
   char cd[N];
   int start;   //哈夫曼码在cd中的起始位置
} HCode;
```




```
void CreateHCode(HTNode ht[],HCode hcd[],int n)
{
   int i,f,c;
   HCode hc;
   for (i=0; i<n; i++)
   {
      hc.start=n;
      c=i;
      f=ht[i].parent;
      while (f!=-1)
      {
         if (ht[f].lchild==c)
            hc.cd[hc.start--]='0';
         else
            hc.cd[hc.start--]='1';
         c=f;
         f=ht[f].parent;
      }
      hc.start++;
      hcd[i]=hc;
   }
}
```

ht	dat	wei	par	lch	rch
[0]		0.07	-1	-1	-1
[1]		0.19	-1	-1	-1
[2]		0.02	-1	-1	-1
[3]		0.06	-1	-1	-1
[4]		0.32	-1	-1	-1
[5]		0.03	-1	-1	-1
[6]		0.21	-1	-1	-1
[7]		0.10	-1	-1	-1
[8]			-1	-1	-1
[9]			-1	-1	-1
[...]			-1	-1	-1

思考题

- 哈夫曼编码的本质是什么？