

哈夫曼树

版权声明：本文为博主原创文章，未经博主允许不得转载。

一、哈夫曼树的概念和定义

什么是哈夫曼树？

让我们先举一个例子。

判定树：

在很多问题的处理过程中，需要进行大量的条件判断，这些判断结构的设计直接影响着程序的执行效率。例如，编制一个程序，将百分制转换成五个等级输出。大家可能认为这个程序很简单，并且很快就可以用下列形式编写出来：

```
[cpp]
01.  if(score<60)
02.      cout<<"Bad"<<endl;
03.  else if(score<70)
04.      cout<<"Pass"<<endl;
05.  else if(score<80)
06.      cout<<"General"<<endl;
07.  else if(score<90)
08.      cout<<"Good"<<endl;
09.  else
10.      cout<<"Very good!"<<endl;
```

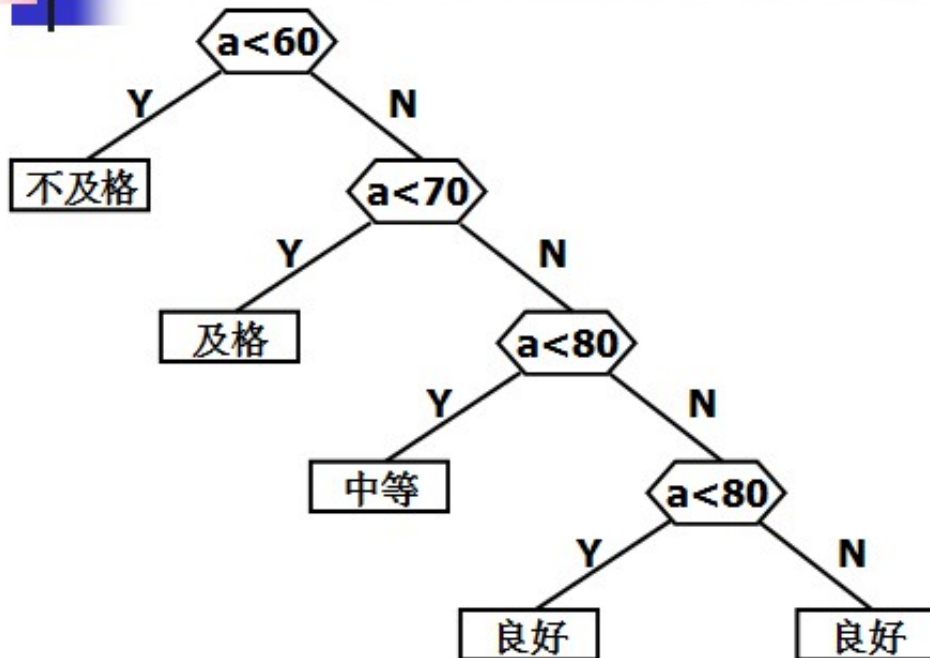
若考虑上述程序所耗费的时间，就会发现该程序的缺陷。在实际中，学生成绩在五个等级上的分布是不均匀的。当学生百分制成绩的录入量很大时，上述判定过程需要反复调用，此时程序的执行效率将成为一个严重问题。

但在实际应用中，往往各个分数段的分布并不是均匀的。下面就是在一次考试中某门课程的各分数段的分布情况：

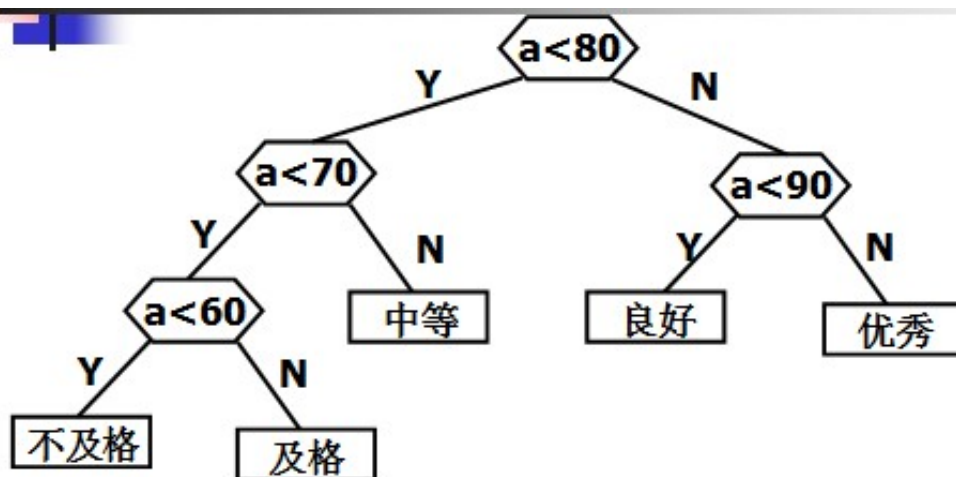
分数	0~59	60~69	70~79	80~89	90~100
比例	0.05	0.15	0.40	0.30	0.10

下面我们就利用哈夫曼树寻找一棵最佳判定树，即总的比较次数最少的判定树。

第一种构造方式：



第二种构造方式：



这两种方式，显然后者的判定过程的效率要比前者高。在也没有别地判定过程比第二种方式的效率更高。

我们称判定过程最优的二叉树为哈夫曼树，又称最优二叉树

=====

定义哈夫曼树之前先说明几个与哈夫曼树有关的概念：

路径： 树中一个结点到另一个结点之间的分支构成这两个结点之间的路径。

路径长度： 路径上的分枝数目称作路径长度。

树的路径长度： 从树根到每一个结点的路径长度之和。

结点的带权路径长度： 在一棵树中，如果其结点上附带有一个权值，通常把该结点的路径长度与该结点上的权值

之积称为该结点的带权路径长度

(weighted path length)

什么是权值?(From 百度百科)

计算机领域中 (数据结构)

权值就是定义的路径上面的值。可以这样理解为节点间的距离。通常指字符对应的二进制编码出现的概率。

至于霍夫曼树中的权值可以理解为：权值大表明出现概率大！

一个结点的权值实际上就是这个结点子树在整个树中所占的比例。

abcd四个叶子结点的权值为7,5,2,4. 这个7,5,2,4是根据实际情况得到的,比如说从一段文本中统计出abcd四个字母出现的次数分别为7,5,2,4. 说a结点的权值为7,意思是说a结点在系统中占有7这个份量.实际上也可以化为百分比来表示,但反而麻烦,实际上是一样的.

树的带权路径长度： 如果树中每个叶子上都带有一个权值，则把树中所有叶子的带权路径长度之和称为树的带权路径长度。

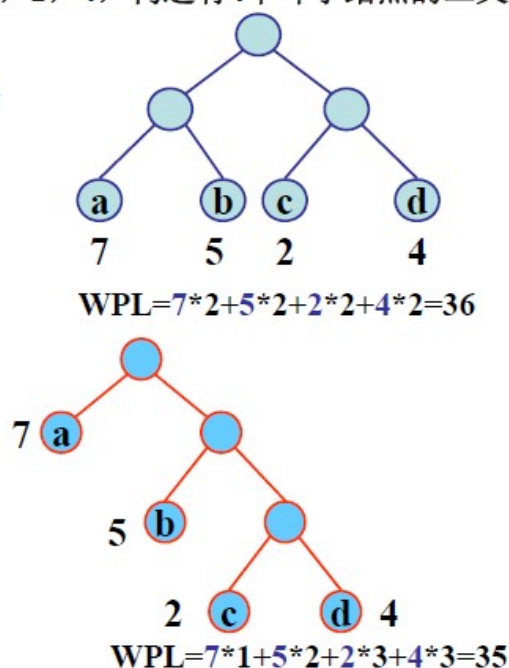
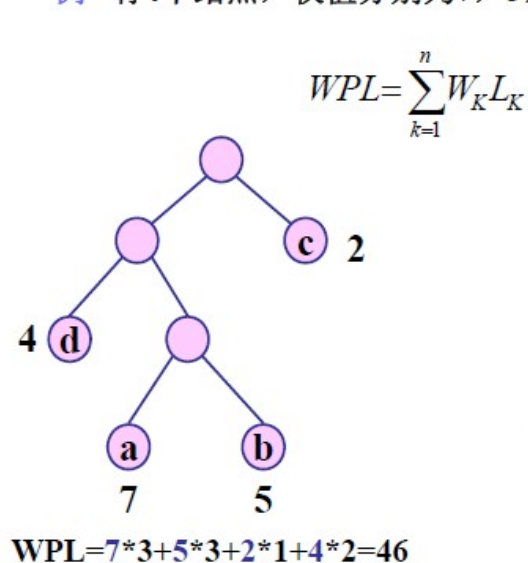
设某二叉树有n个带权值的叶子结点，则该二叉树的带权路径长度记为：

$$WPL = \sum_{k=1}^n w_k l_k$$

公式中， w_k 为第 k 个叶子结点的权值； l_k 为该结点的路径长度。

示例：

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



一般来说，用 n ($n > 0$) 个带权值的叶子来构造二叉树，限定二叉树中除了这 n 个叶子外只能出现度为2的结点。

那么符合这样条件的二叉树往往可构造出许多颗，

其中带权路径长度最小的二叉树就称为**哈夫曼树**或**最优二叉树**

二、哈夫曼树的构造

根据哈夫曼树的定义，一棵二叉树要使其WPL值最小，必须使权值越大的叶子结点越靠近根结点，而权值越小的叶子结点越远离根结点。

哈弗曼依据这一特点提出了一种构造最优二叉树的方法，其基本思想如下：

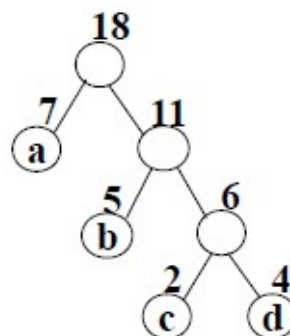
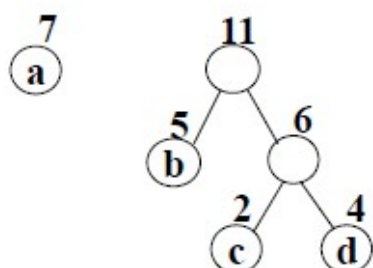
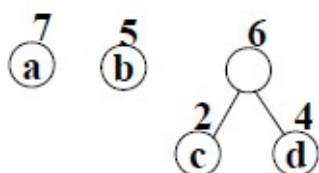
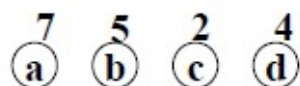
✦ 构造Huffman树的方法——Huffman算法

✦ 构造Huffman树步骤

- ✦ 根据给定的 n 个权值 $\{w_1, w_2, \dots, w_n\}$ ，构造 n 棵只有根结点的二叉树，令其权值为 w_j
- ✦ 在森林中选取两棵根结点权值最小的树作左右子树，构造一棵新的二叉树，置新二叉树根结点权值为其左右子树根结点权值之和
- ✦ 在森林中删除这两棵树，同时将新得到的二叉树加入森林中
- ✦ 重复上述两步，直到只含一棵树为止，这棵树即哈夫曼树

下面演示了用Huffman算法构造一棵Huffman树的过程：

例



三、哈夫曼树的在编码中的应用

在电文传输中，需要将电文中出现的每个字符进行二进制编码。在设计编码时需要遵守两个原则：

- (1) 发送方传输的二进制编码，到接收方解码后必须具有唯一性，即解码结果与发送方发送的电文完全一样；
- (2) 发送的二进制编码尽可能地短。下面我们介绍两种编码的方式。

1. 等长编码

这种编码方式的特点是每个字符的编码长度相同（编码长度就是每个编码所含的二进制位数）。假设字符集只含有4个字符A，B，C，D，用二进制两位表示的编码分别为00，01，10，11。若现在有一段电文为：ABACCDA，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。这种编码的特点是译码简单且具有唯一性，但编码长度并不是最短的。

2. 不等长编码

在传送电文时，为了使其二进制位数尽可能地少，可以将每个字符的编码设计为不等长的，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。例如，可以为A，B，C，D四个字符分别分配0，00，1，01，并可将上述电文用二进制序列：000011010发送，其长度只有9个二进制位，但随之带来了一个问题，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。

因此，为了设计长短不等的编码，以便减少电文的总长，还必须考虑编码的**唯一性**，即在建立不等长编码时必须使任何一个字符的编码都不是另一个字符的前缀，这宗编码称为前缀编码（prefix code）

- (1) 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；
- (2) 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

例题：

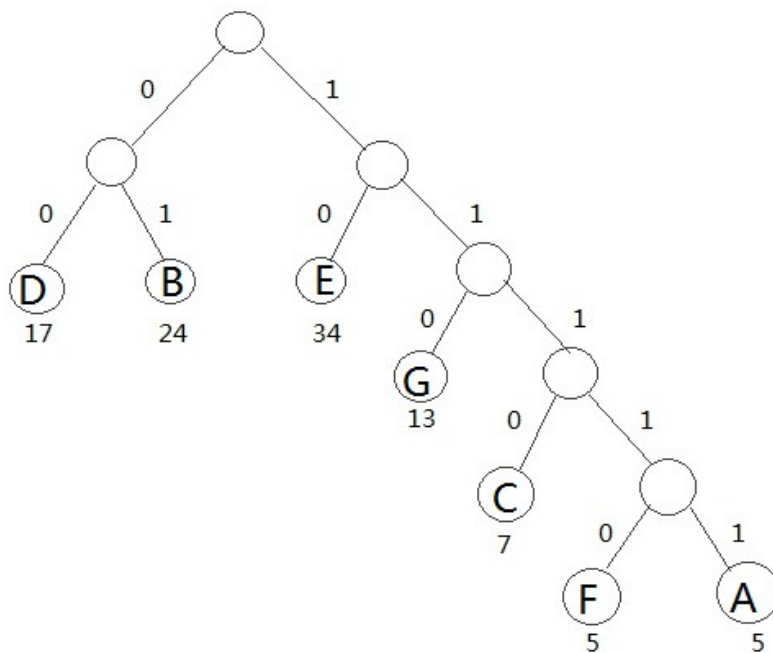
假设一个文本文件TFile中只包含7个字符{A，B，C，D，E，F，G}，这7个字符在文本中出现的次数为{5，24，7，17，34，5，13}

利用哈夫曼树可以为文件TFile构造出符合前缀编码要求的不等长编码

具体做法：

1. 将TFile中7个字符都作为叶子结点，每个字符出现次数作为该叶子结点的权值
2. 规定哈夫曼树中所有左分支表示字符0，所有右分支表示字符1,将依次从根结点到每个叶子结点所经过的分支的二进制位的序列作为该结点对应的字符编码
3. 由于从根结点到任何一个叶子结点都不可能经过其他叶子，这种编码一定是前缀编码，哈夫曼树的带权路径长度正好是文件TFile编码的总长度

通过哈夫曼树来构造的编码称为**哈弗曼编码（huffman code）**



A: 1111
 B: 01
 C: 1110
 D: 00
 E: 0
 F: 11110
 G: 110

```

[cpp]
01. #include<iostream>
02. #include<cstdio>
03. #include<cstring>
04. using namespace std;
05.
06. #define N 10          // 带编码字符的个数，即树中叶结点的最大个数
07. #define M (2*N-1)    // 树中总的结点数目
08.
09. class HTNode{         // 树中结点的结构
10. public:
11.     unsigned int weight;
12.     unsigned int parent,lchild,rchild;
13. };
14.
15. class HTCode{
16. public:
17.     char data;         // 待编码的字符
18.     int weight;        // 字符的权值
19.     char code[N];      // 字符的编码
20. };
21.
22. void Init(HTCode hc[], int *n){
23. // 初始化，读入待编码字符的个数n，从键盘输入n个字符和n个权值
24.     int i;
25.     printf("input n = ");
26.     scanf("%d",&(*n));
  
```



```

27.
28.     printf("\ninput %d character\n",*n);
29.
30.     fflush(stdin);
31.     for(i=1; i<=*n; ++i)
32.         scanf("%c",&hc[i].data);
33.
34.     printf("\ninput %d weight\n",*n);
35.
36.     for(i=1; i<=*n; ++i)
37.         scanf("%d",&(hc[i].weight) );
38.     fflush(stdin);
39. }//
40.
41. void Select(HTNode ht[], int k, int *s1, int *s2){
42.     // ht[1...k]中选择parent为0, 并且weight最小的两个结点, 其序号由指针变量s1, s2指
    示
43.     int i;
44.     for(i=1; i<=k && ht[i].parent != 0; ++i){
45.         ; ;
46.     }
47.     *s1 = i;
48.
49.     for(i=1; i<=k; ++i){
50.         if(ht[i].parent==0 && ht[i].weight<ht[*s1].weight)
51.             *s1 = i;
52.     }
53.
54.     for(i=1; i<=k; ++i){
55.         if(ht[i].parent==0 && i!=*s1)
56.             break;
57.     }
58.     *s2 = i;
59.
60.     for(i=1; i<=k; ++i){
61.         if(ht[i].parent==0 && i!=*s1 && ht[i].weight<ht[*s2].weight)
62.             *s2 = i;
63.     }
64. }
65.
66. void HuffmanCoding(HTNode ht[],HTCode hc[],int n){
67.     // 构造Huffman树ht, 并求出n个字符的编码
68.     char cd[N];
69.     int i,j,m,c,f,s1,s2,start;
70.     m = 2*n-1;
71.
72.     for(i=1; i<=m; ++i){
73.         if(i <= n)
74.             ht[i].weight = hc[i].weight;
75.         else
76.             ht[i].parent = 0;
77.         ht[i].parent = ht[i].lchild = ht[i].rchild = 0;
78.     }
79.
80.     for(i=n+1; i<=m; ++i){
81.         Select(ht, i-1, &s1, &s2);
82.         ht[s1].parent = i;
83.         ht[s2].parent = i;

```

```

84.         ht[i].lchild = s1;
85.         ht[i].rchild = s2;
86.         ht[i].weight = ht[s1].weight+ht[s2].weight;
87.     }
88.
89.     cd[n-1] = '\0';
90.
91.     for(i=1; i<=n; ++i){
92.         start = n-1;
93.         for(c=i,f=ht[i].parent; f; c=f,f=ht[f].parent){
94.             if(ht[f].lchild == c)
95.                 cd[--start] = '0';
96.             else
97.                 cd[--start] = '1';
98.         }
99.         strcpy(hc[i].code, &cd[start]);
100.    }
101. }
102.
103.
104. int main()
105. {
106.     int i,m,n,w[N+1];
107.     HTNode ht[M+1];
108.     HCode hc[N+1];
109.     Init(hc, &n);        // 初始化
110.     HuffmanCoding(ht,hc,n);    // 构造Huffman树, 并形成字符的编码
111.
112.     for(i=1; i<=n; ++i)
113.         printf("\n%c---%s",hc[i].data,hc[i].code);
114.     printf("\n");
115.
116.     return 0;
117. }

```

—— 生命的意义，在于赋予它意义。

原

创 <http://blog.csdn.net/shuangde800> , By
D_Double