

数据结构论文

二叉树及其应用

|  |
| --- |
| *基于哈夫曼编码的文本压缩实现* |
|  |

|  |  |
| --- | --- |
| 学生姓名 | 张赛东 |
| 班级 | 计算机类1706 |
| 学号 | 17408002237 |
| 成绩 |  |
| 指导教师(签字) |  |

**计算机学院**

2018年 5 月 20 日

序 言

在计算机科学中，树是一种重要的非线性[数据结构](http://baike.baidu.com/view/9900.htm)，直观地看，它是[数据元素](http://baike.baidu.com/view/38785.htm)（在树中称为结点）按分支关系组织起来的结构。二叉树是每个[节点](http://baike.baidu.com/view/47398.htm)最多有两个子树的有序树。通常子树被称作“左子树”（left subtree）和“右子树”（right subtree）。二叉树常被用于实现[二叉查找树](http://baike.baidu.com/view/389459.htm)和[二叉堆](http://baike.baidu.com/view/668854.htm)。值得注意的是，二叉树不是树的特殊情形。在图论中，二叉树是一个连通的无环图，并且每一个顶点的度不大于3。有根二叉树还要满足根结点的度不大于2。有了根结点后，每个顶点定义了唯一的根结点，和最多2个子结点。数据结构中二叉树的应用有着很重要的作用，对解决很多程序问题都有很大帮助，因此，我们更应该深入的学习和了解二叉树的应用，才能更方便地解决计算机的相关问题。

而二叉树中相当有名的莫过于哈夫曼树，哈夫曼编码在编码界可谓是人尽皆知，效率实在是高，这里本人将会介绍哈夫曼编码的一个简单实际应用：文本压缩，由其可以升华至文件压缩，大数据压缩等等，下文会有详细介绍，此处不再赘述。

哈夫曼编码的出现还是很让人意外的，1951年，哈夫曼和他在MIT[信息论](https://baike.baidu.com/item/%E4%BF%A1%E6%81%AF%E8%AE%BA" \t "_blank)的同学需要选择是完成学期报告还是期末考试。导师Robert M. Fano给他们的学期报告的题目是，寻找最有效的[二进制编码](https://baike.baidu.com/item/%E4%BA%8C%E8%BF%9B%E5%88%B6%E7%BC%96%E7%A0%81" \t "_blank)。由于无法证明哪个已有编码是最有效的，哈夫曼放弃对已有编码的研究，转向新的探索，最终发现了基于有序频率[二叉树](https://baike.baidu.com/item/%E4%BA%8C%E5%8F%89%E6%A0%91" \t "_blank)编码的想法，并很快证明了这个方法是最有效的。由于这个算法，学生终于青出于蓝，超过了他那曾经和信息论创立者[香农](https://baike.baidu.com/item/%E9%A6%99%E5%86%9C" \t "_blank)共同研究过类似编码的导师。哈夫曼使用自底向上的方法构建二叉树，避免了次优算法Shannon-Fano编码的最大弊端──自顶向下构建树。

次年，David A. Huffman在[麻省理工](https://baike.baidu.com/item/%E9%BA%BB%E7%9C%81%E7%90%86%E5%B7%A5" \t "_blank)攻读博士时发表了《一种构建极小多余编码的方法》（A Method for the Construction of Minimum-Redundancy Codes）一文，它一般就叫做Huffman编码。[1] 

Huffman在1952年根据香农（Shannon）在1948年和范若（Fano）在1949年阐述的这种编码思想提出了一种不定长编码的方法，也称[哈夫曼](https://baike.baidu.com/item/%E9%9C%8D%E5%A4%AB%E6%9B%BC" \t "_blank)（Huffman）编码。哈夫曼编码的基本方法是先对图像数据扫描一遍，计算出各种像素出现的概率，按概率的大小指定不同长度的唯一码字，由此得到一张该图像的哈夫曼码表。编码后的图像数据记录的是每个像素的码字，而码字与实际像素值的对应关系记录在码表中。

哈夫曼编码是可变[字长](https://baike.baidu.com/item/%E5%AD%97%E9%95%BF" \t "_blank)编码(VLC)的一种。 Huffman于1952年提出一种编码方法，该方法完全依据[字符](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6" \t "_blank)出现概率来构造异字头的平均长 度最短的码字，有时称之为最佳编码，一般就称Huffman编码。下面引证一个定理，该定理保证了按字符出现概率分配码长，可使平均码长最短。

下面欢迎进入基础哈夫曼编码的世界！

内 容 摘 要

本论文主要包含以下几个方面：选择此课题的原因，哈夫曼编码的简介、基本术语、构造以及多叉哈夫曼树，对文本压缩的介绍，对压缩文件的原因分析，具体初步地对压缩与解压缩的介绍 ，对本论文所用到的程序的编码核心与解码核心详细介绍 ，对程序的具体功能及操作的说明，对程序运行结果的分析并附上了相应的截图，对程序运行结果的总结，本篇论文完成后的总结，附上所用到的C语言程序源代码，参考文献。

关 键 词：

哈夫曼树 哈夫曼编码 文本压缩 文件压缩 压缩与解压缩

**目 录**

[一、课题选择 ……………………………………………………………………………………….-5 -](#_Toc366094586)

[（一）原因 - ………………………………………………………………………………………5 -](#_Toc366094587)

[二、哈夫曼编码 -…………………………………………………………………………………5 -](#_Toc366094591)

[（一）哈夫曼编码简介………………………………………………………………………… ..5-](#_Toc366094592)

[（二）哈夫曼树基本术语 - ……………………………………………………………………..6 -](#_Toc366094593)

[（三）哈夫曼树的构造 -………………………………………………………….. …………7-](#_Toc366094594)

[（四）多叉哈夫曼树 -…………………………………………………………………………...7 -](#_Toc366094595)

[三、文本压缩 - ……………………………………………………………………………………...7 -](#_Toc366094600)

[（一）关于文本压缩 - …………………………………………………………………………..7 –](#_Toc366094601)

[（二）文本压缩原因 - ……………………………………………………………………..........8 –](#_Toc366094593)

[（三）压缩与解压缩 -………………………………………………………………………….. 8 -](#_Toc366094594)

[（四）初步了解 -……………………………………………………………………………….. 9 -](#_Toc366094595)

[（五）编码核心 - …………………………………………………………………………..........9 -](#_Toc366094596)

[（六）解码核心 -………………………………………………………………………………...10 -](#_Toc366094597)

[（七）具体功能及操作 -……………………………………………………………………. 11 –](#_Toc366094598)

[（八）程序的运行 -…………………………………………………………………………. 11 -](#_Toc366094599)

[（九）对程序运行结果的总结 ………………………………………………………………..20 –](#_Toc366094599)

[四、结论 - …………………………………………………………………………………………..20 –](#_Toc366094600)

[所用的C语言程序源代码………………………………………………………………………….22 -](#_Toc366094602)

[参考文献 -…………………………………………………………………………………………...26 -](#_Toc366094602)

课题选择

**原因**

  本人在网上查阅了不少资料，发现二叉树有较强的功能，在C++STL中的set、map，Linux虚拟内存的管理，红黑树，B-Tree，B+-Tree在文件系统中的应用等方面都有非常广泛的实际应用。对于二叉树的应用这个课题，我觉得有必要挑选一个自己感兴趣的实际应用，故选择了文本压缩。

哈夫曼编码

**哈夫曼编码简介**

先简单介绍一下哈夫曼编码，其本质上是统计文件中各字符出现的频率，生成最优的编码，使频率高的字符获得对应的尽可能短的0和1组成的序列串，用以替代原来的八位序列串。

在数据通信中，需要将传送的文字转换成二进制的字符串，用0，1码的不同排列来表示字符。例如，需传送的报文为“AFTER DATA EAR ARE ART AREA”，这里用到的字符集为“A，E，R，T，F，D”，各字母出现的次数为{8，4，5，3，1，1}。现要求为这些字母设计编码。要区别6个字母，最简单的二进制编码方式是等长编码，固定采用3位二进制，可分别用000、001、010、011、100、101对“A，E，R，T，F，D”进行编码发送，当对方接收报文时再按照三位一分进行译码。显然编码的长度取决报文中不同字符的个数。若报文中可能出现26个不同字符，则固定编码长度为5。然而，传送报文时总是希望总长度尽可能短。在实际应用中，各个字符的出现频度或使用次数是不相同的，如A、B、C的使用频率远远高于X、Y、Z，自然会想到设计编码时，让使用频率高的用短码，使用频率低的用长码，以优化整个报文编码

为使不等长编码为前缀编码(即要求一个字符的编码不能是另一个字符编码的前缀)，可用字符集中的每个字符作为叶子结点生成一棵编码二叉树，为了获得传送报文的最短长度，可将每个字符的出现频率作为字符结点的权值赋予该结点上，显然字使用频率越小权值越小，权值越小叶子就越靠下，于是频率小编码长，频率高编码短，这样就保证了此树的最小带权路径长度效果上就是传送报文的最短长度。因此，求传送报文的最短长度问题转化为求由字符集中的所有字符作为叶子结点，由字符出现频率作为其权值所产生的哈夫曼树的问题。利用哈夫曼树来设计二进制的前缀编码，既满足前缀编码的条件，又保证报文编码总长最短。

**哈夫曼静态编码：**它对需要编码的数据进行两遍扫描：第一遍统计原数据中各字符出现的频率，利用得到的频率值创建哈夫曼树，并必须把树的信息保存起来，即把字符0-255(2^8=256)的频率值以2-4BYTES的长度顺序存储起来，（用4Bytes的长度存储频率值，频率值的表示范围为0--2^32-1，这已足够表示大文件中字符出现的频率了）以便解压时创建同样的哈夫曼树进行解压；第二遍则根据第一遍扫描得到的哈夫曼树进行编码，并把编码后得到的码字存储起来。

**哈夫曼动态编码：**动态哈夫曼编码使用一棵动态变化的哈夫曼树，对第t+1个字符的编码是根据原始数据中前t个字符得到的哈夫曼树来进行的，编码和解码使用相同的初始哈夫曼树，每处理完一个字符，编码和解码使用相同的方法修改哈夫曼树，所以没有必要为解码而保存哈夫曼树的信息。编码和解码一个字符所需的时间与该字符的编码长度成正比，所以动态哈夫曼编码可实时进行。

**2、哈夫曼译码**

在通信中，若将字符用哈夫曼编码形式发送出去，对方接收到编码后，将编码还原成字符的过程，称为哈夫曼译码。

**哈夫曼基本术语**

**1、路径和路径长度**

在一棵树中，从一个结点往下可以达到的孩子或孙子结点之间的通路，称为路径。通路中分支的数目称为路径长度。若规定根结点的层数为1，则从根结点到第L层结点的路径长度为L-1。

**2、结点的权及带权路径长度**

若将树中结点赋给一个有着某种含义的数值，则这个数值称为该结点的权。结点的带权路径长度为：从根结点到该结点之间的路径长度与该结点的权的乘积。

**3、树的带权路径长度**

树的带权路径长度规定为所有叶子结点的带权路径长度之和，记为WPL

**哈夫曼树构造**

假设有n个权值，则构造出的哈夫曼树有n个叶子结点。 n个权值分别设为 w1、w2、…、wn，则哈夫曼树的构造规则为：

(1) 将w1、w2、…，wn看成是有n 棵树的森林(每棵树仅有一个结点)；

(2) 在森林中选出两个根结点的权值最小的树合并，作为一棵新树的左、右子树，且新树的根结点权值为其左、右子树根结点权值之和；

(3)从森林中删除选取的两棵树，并将新树加入森林；

(4)重复(2)、(3)步，直到森林中只剩一棵树为止，该树即为所求得的哈夫曼树

**多叉哈夫曼树**

哈夫曼树也可以是k叉的，只是在构造k叉哈夫曼树时需要先进行一些调整。构造哈夫曼树的思想是每次选k个权重最小的元素来合成一个新的元素，该元素权重为k个元素权重之和。但是当k大于2时，按照这个步骤做下去可能到最后剩下的元素少于k个。解决这个问题的办法是假设已经有了一棵哈夫曼树(且为一棵满k叉树)，则可以计算出其叶节点数目为(k-1)nk+1,式子中的nk表示子节点数目为k的节点数目。于是对给定的n个权值构造k叉哈夫曼树时,可以先考虑增加一些权值为0的叶子节点，使得叶子节点总数为(k-1)nk+1这种形式,然后再按照哈夫曼树的方法进行构造即可。

文本压缩

**关于文本压缩**

事实上，二叉树被极为频繁地应用在文件压缩上面，哈夫曼树是人们常说的最优二叉树，能够将文件的原字符序列串最大限度地压缩。如果能有精力去完善代码，其实可以利用哈夫曼树压缩几乎所有文件，因为目前绝大部分计算机上所存储的信息就只有0和1；一般来说，一个字符对应八个0和1的组合，ASCII码将大量千奇百怪的字符用0和1按一定的顺序编码，故使用起来十分方便，可以利用它输入输出各种符号文

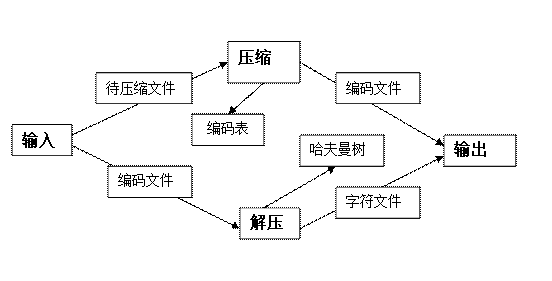
**文本压缩原因**

  压缩文件的原因是什么呢？其实啊，在实际应用中我们一般用不到那么多的字符啊，看键盘上也就不到100种，大部分人实际上常用字符不过30个，然而却要用八个0和1组成的序列串来存储各个字符，这是不是很浪费呢？所以针对使用少量字符的文件们，我们可以利用哈夫曼树来进行压缩，这样就会使常用字符的编码尽可能地短，从而由八个变成三或四个，什么五个六个的基本是不常用的，在总编码里自然较少。

**压缩与解压缩**

压缩完之后，原文件中的每个字符都会按照哈夫曼编码的形式按字节保存在压缩文件中，所占用的存储空间就少了很多，人们就可以腾出许多空间给其他文件，另外一个特别方便的就是传输起来效率太高了，比压缩之前快了很多，节约不少时间。

既然有了压缩，那是不是也要有解压缩呢？没错！解压缩实际上是获取了哈夫曼编码规则后，将其按照规则来进行翻译，也就是将那少于八个0和1的组合序列串恢复成八个；当然啦，这个过程是需要一些配置文件帮忙的

可以用下方的图理解一下

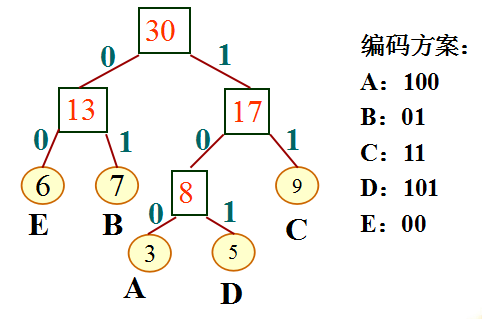
**初步了解**

马上我将介绍基于哈夫曼编码的文本压缩实现，之所以介绍这个，是考虑大多数人喜欢文本，在聊天软件上面用的简短语言、写日记、做各种记录等等，文本无疑是很常用的，也是最为基础的，接下来我要介绍的文本压缩还是较为详细的，应该能让人对二叉树的应用有所了解。

**编码核心**

我个人很喜欢底层的东西，下面将进入正题。  
 文章下方将会给出具体的代码，而且代码有详细注释，整个程序编码的方法较简单，从头开始逐个读取文本字符串中的每个字符，查询编码表从而得到对应编码并且输出。重复进行上述操作直到文本被处理完毕

都提到了编码，不给出点核心代码怎么行呢？为了更好地理解哈夫曼编码，以下给出实现哈夫曼树的编码序列操作：  
void EnCoding(char str[], char hfmstr[])  
{//根据codeset编码表,逐个将str字符串中的字符转化为它的huffman编码,结果编码串放在hfmstr字符串中  
int i, j;  
hfmstr[0]='\0';//把hfmstr串赋空  
i=0;  
while(str[i]!='\0')//从第头开始扫描str的每个字符,一直到该字符的结束     
    {  
j=str[i]-32;//执行字符到huffman的转换  
strcat(hfmstr, codeset[j].code);//把codest编码串添加到hfmstr结尾处  
i++;//每次循环完i的值加1  
}  
}

可以用下方的图理解一下哈夫曼编码：

**解码核心**

解码的方式也较容易让人接受，将指针指向哈夫曼树树根，从总的序列床头部开始一个一个地读取编码序列中的每一个0或者1；若为1则向右子树走，若为0则向左子树走。就像顺藤摸瓜一般，早晚都会走到叶子结点的，此时就取出该节点所对应的字符并输出。然后重新将指针放置于树根，重复上述过程，也是一直到编码序列被全部处理。  
  
   由于压缩前后的文件有较大的区别，人们也常分析它们之间的关系，也会考虑我们买东西，投资等所想到的性价比，于是乎，出现了一个压缩比例的名词，编码后的文本长度为编码序列中0和1的总个数，而原文本的长度为其中所有的字符数量再乘以8。这两个长度的比例即是压缩比例。  
 

也是一样，这又提到了解码，不来点解码核心代码简直对不住上文所给出的编码代码呀，下面给出解码核心代码：  
void DeCoding(huffnode tree[], int n, char hfmstr[], char decodestr[])  
//根据tree数组中的huffman树,逐个对hfmstr字符串中的字符进行译码,结果放在decodestr字符串中  
{  
int i=0,j=0;  
huffnode p;  
p=tree[2\*n-2];//序号2\*n-2节点就是树根节点  
while(hfmstr[i]!='\0')//从头开始扫描每个字符,直到结束  
{while(p.lchild!=-1&&p.rchild!=-1)//指针为空，儿子的值取完了  
{  
if(hfmstr[i]=='0')//为0则向左子树走  
{  
p=tree[p.lchild];//取出叶子节点中的字符  
}  
else if(hfmstr[i]=='1')//为1则向右子树走  
{  
p=tree[p.rchild];//取出叶子节点中的字符  
}  
i++;  
}  
decodestr[j]=p.data;j++;//对字符进行译码,结果放在decodestr字符串中  
p=tree[2\*n-2];//返回根节点  
}  
}

**具体功能及操作**

接下来让我们了解一下文章下方所给代码的具体功能，首先要求我们输入一段文字，那为什么不直接压缩文件呢？那样的话并不能让人很好地理解文本压缩，只是让人看到它有压缩的本事了，还不如直接下载个软件，所以给出这样一种朴素的展现方式，这个时候，我们就可以复制文本，并粘贴到光标处，回车有惊喜！

我看过之后，是有一种耳目一新的感觉,下面我就来实际操作一下吧，我将下面一段浪漫的文字粘贴一下，（现在的我好想来一段music）：

Every night in my dreams,I see you,I feel you,that is how I know you go on.Far across the distance and spaces between us ,you have come to show you go on.Near,far, wherever you are,I believe that the heart does go on.Once more, you opened the door,and you're here in my heart, and my heart will go on and on .

回车之后很精彩呀！

**程序的运行**

请输入要转换的字符串  
Every night in my dreams,I see you,I feel you,that is how I know you go on.Far across the distance and spaces between us ,you have come to show you go on.Near,far, wherever you are,I believe that the heart does go on.Once more, you opened the door,and you're here in my heart, and my heart will go on and on .   
字符频率统计:  
 :59, ':1, ,:10, .:4, E:1, F:1, I:4, N:1, O:1, a:18, b:2, c:5, d:9, e:35, f:2, g:5, h:14, i:7, k:1, l:4, m:6, n:17, o:28, p:2, r:16, s:11, t:14, u:9, v:4, w:6, y:12,   
Huffman tree :  
i Value Lchild Rchild Weight  
60   58 59 309   
59   56 57 -182   
58   0 55 -127   
57   53 54 -108   
56   51 52 -74   
55   50 13 -68   
54   22 49 -58   
53   47 48 -50   
52   45 46 -39   
51   44 9 -35   
50   43 21 -33   
49   26 24 -30   
48   42 16 -27   
47   41 30 -23   
46   2 25 -21   
45   27 40 -18   
44   39 12 -17   
43   37 38 -16   
42   29 17 -13   
41   15 20 -11   
40   36 11 -9   
39   34 35 -8   
38   19 28 -8   
37   3 6 -8   
36   32 33 -4   
35   23 31 -4   
34   10 14 -4   
33   8 18 -2   
32   5 7 -2   
31   1 4 -2   
30 y -1 -1 -12   
29 w -1 -1 -6   
28 v -1 -1 -4   
27 u -1 -1 -9   
26 t -1 -1 -14   
25 s -1 -1 -11   
24 r -1 -1 -16   
23 p -1 -1 -2   
22 o -1 -1 -28   
21 n -1 -1 -17   
20 m -1 -1 -6   
19 l -1 -1 -4   
18 k -1 -1 -1   
17 i -1 -1 -7   
16 h -1 -1 -14   
15 g -1 -1 -5   
14 f -1 -1 -2   
13 e -1 -1 -35   
12 d -1 -1 -9   
11 c -1 -1 -5   
10 b -1 -1 -2   
9 a -1 -1 -18   
8 O -1 -1 -1   
7 N -1 -1 -1   
6 I -1 -1 -4   
5 F -1 -1 -1   
4 E -1 -1 -1   
3 . -1 -1 -4   
2 , -1 -1 -10   
1 ' -1 -1 -1   
0   -1 -1 -59   
haffman编码为:  
 :00  
':10000110  
,:10110  
.:010000  
E:10000111  
F:10101000  
I:010001  
N:10101001  
O:10101010  
a:1001  
b:1000000  
c:101011  
d:10001  
e:011  
f:1000001  
g:110000  
h:11011  
i:110101  
k:10101011  
l:010010  
m:110001  
n:0101  
o:1110  
p:1000010  
r:11111  
s:10111  
t:11110  
u:10100  
v:010011  
w:110100  
y:11001  
编码序列: 10000111010011011111111100100010111010111000011011111100011010101010011000111001001000111111011100111000110111101100100010010111011011001100111101010010110010001001000001011011010010001100111101010010110111101101110011111000110101101110011011111011010000010001001010101101011110110100001100111101010000110000111000111001010100001010100010011111100100110101111111111010111101110011110110110110010001110101101111111010010101101011011001001010110001001011110000101001101011011101110010000000111111011010001101101010010100101110010110110011110101000011011100101001101100101011111011000101100111101110001011111011111011010000110011110101000011000011100011100101010000101010010111001111111011010000011001111111011000110100110110111111101101001101111111001100111101010000100111111011101100100010010000000110100101101010110100110110011110110111001111100011110110110110011011011100111111111100010001111001110111001100001110001110010101000010101010010110101101100110001111011111011101100011001111010100001110100001001101010111000100111101101101100100011110111011111101101001010110001001100111101010010000110111110110011011011111110110011010101010011000111001001101101110011111111110101100010010101100010011000111001001101101110011111111110001101001101010100100100100011000011100011100101001001010110001001110010101000000  
解码后的字符串: Every night in my dreams,I see you,I feel you,that is how I know you go on.Far across the distance and spaces between us ,you have come to show you go on.Near,far, wherever you are,I believe that the heart does go on.Once more, you opened the door,and you're here in my heart, and my heart will go on and on.   
Program ended with exit code: 0

太精彩了，我再配几张图，了解一下；滚动图片比较长所以额外截了几个分开的方便观看，当然啦，是可以用放大镜去看滚动图片的。



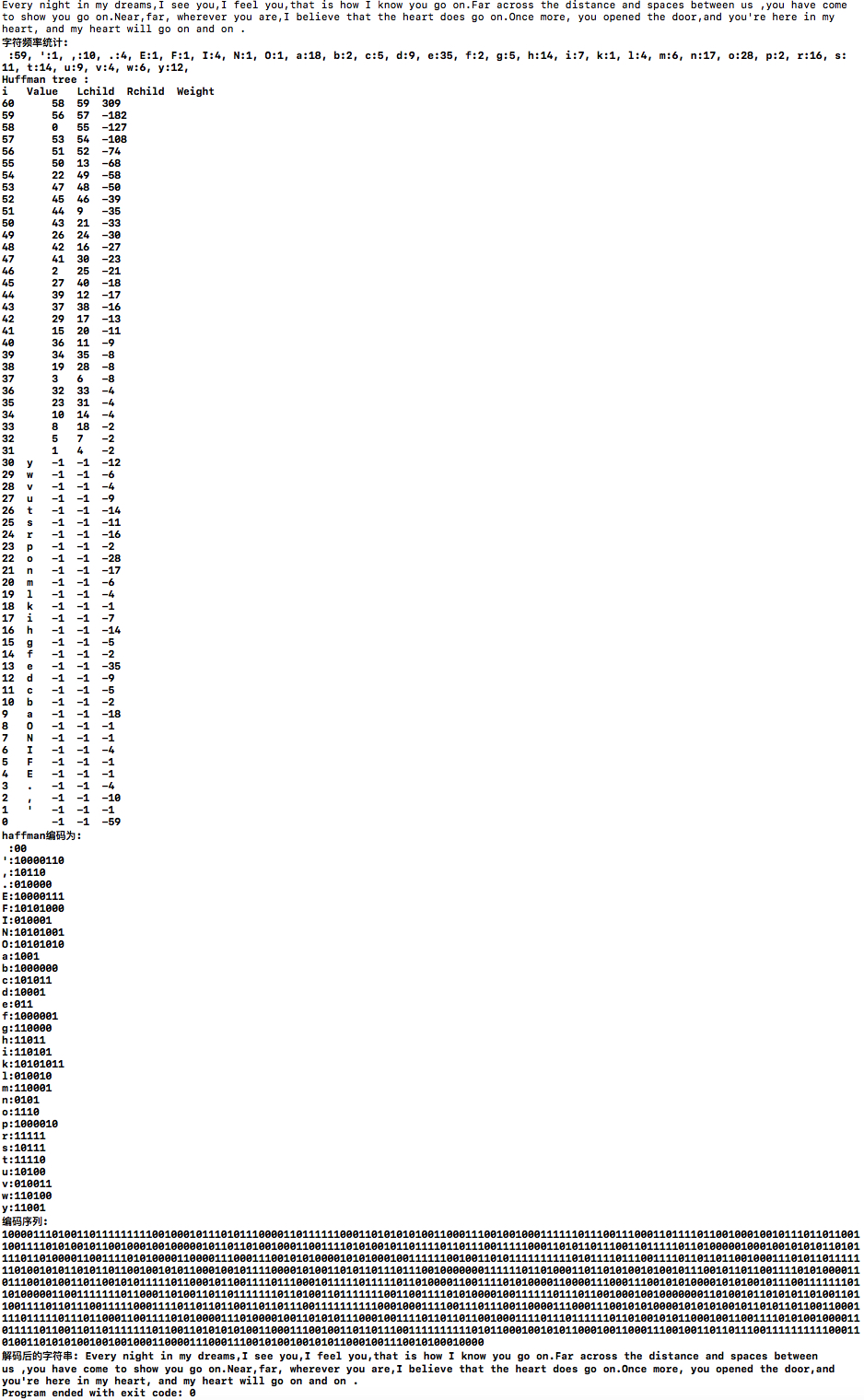










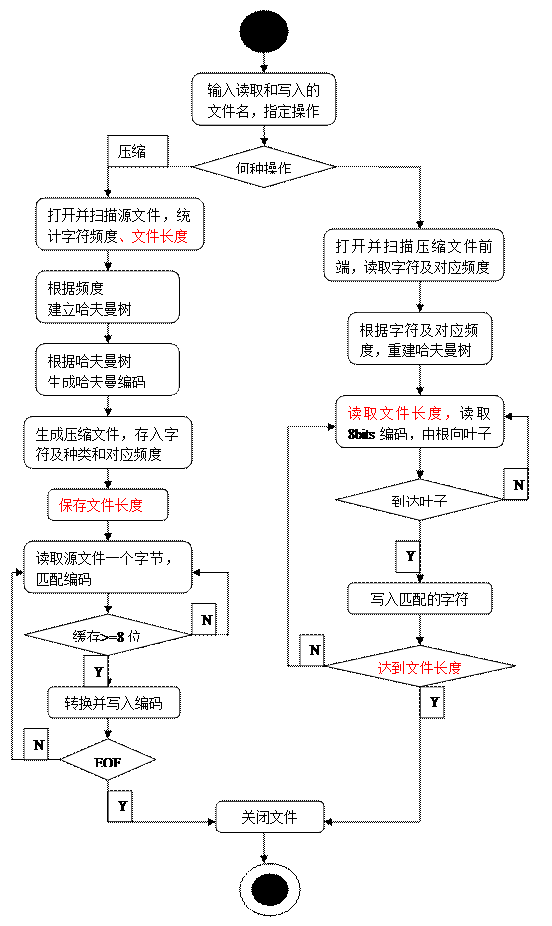


**对程序运行结果的总结**

上面的运行结果非常明了，没想到吧，常常被人忽视的空格符号居然是使用频率最高的，这里按符号、大写、小写的顺序将频率展示了出来，哈夫曼树的信息包括左子树，右子树，权重等信息可谓是一目了然。再然后就是列出了每个字符对应的哈夫曼编码，这里空格符号的编码是00，只有使用频率最高的才有这待遇，文本编码届最短编码挺令我兴奋的。而且，经我仔细对比，确实是频率高的字符对应编码不可能比频率低的长。  
  
   再然后就是列出了整个使用哈夫曼编码的编码序列，比原来文本字符的8倍长度可要短多了吧，这还只是文本所体现出来的优势，当文本非常多质变为一个大文件后，那优势会有多么明显可想而知！现在的绝大多数文件都能被压缩，这些海量文件被压缩了，是不是感觉有种非常轻松的感觉呢。  
  
   最后的一个步骤就是解码了，这里为了方便展示，将其代码与压缩代码放在一起了，它是完全可以被剥离的，剥离后稍微改进下程序就可以将输入或粘贴的0和1序列串按生成的哈夫曼编码规则翻译成文本，看着一堆0101经翻译后变成了如此动人的旋律，我不由得想起了丑小鸭蜕变为白天鹅的故事。

结论

就像大家所见证过的，一个原理或许可以创造很多奇迹，上述程序虽然不能做到压缩诸如照片、大量文本、文档、电子书、视频等文件，但是这些工作也基本上是建立在其基础之上的，哈夫曼树的出现为文件压缩带来了福音，为人们带来了许多便利，本人很高兴在此分享个人对哈夫曼树的理解和想法，由于本人水平所限，文中疏漏与不足之处在所难免，在此还请见谅。

具体的大型文件压缩是比较复杂的，需要克服诸多问题，最后分享一个文件压缩的步骤图：

所用的C语言程序源代码

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <limits.h>//专门用于检测整型数据数据类型的表达值范围  
#define N 96 //ASCII字符集包含至多N个可见字符  
typedef struct    //Huffman树节点定义  
{ char data; //字符值  
    int weight; //权重  
    int lchild; //左子结点  
    int rchild; //右子结点  
} huffnode; //huffman节点类型  
struct charcode  
{ int count; //字符出现的次数(频率)  
    char code[N]; //字符的Huffman编码  
} codeset[N]; //编码表,长为N,每项对应一个ascii码字符,下标i的项对应ascii编码为i+32的字符  
huffnode \* CreateHufftree(char data[], int weight[], int n)   //建立Huffman树的算法  
{  
    int i,k;  
    int min1,min2,min\_i1,min\_i2;  
    huffnode \*tree;  
    tree=(huffnode \*)malloc((2\*n-1)\*sizeof(huffnode));  //为Huffman树分配2n-1个节点空间  
    for (i=0;i<2\*n-1;i++)  //初始化,将各字符和其频率填入Huffman树,作为叶子结点  
    {  
        tree[i].lchild=tree[i].rchild=-1;  
        if (i<n) {  
            tree[i].data=data[i];  
            tree[i].weight=weight[i];  
        }  
        else tree[i].data=' ';  
    }  
    for (i=n;i<2\*n-1;i++)   ////合并两棵树,作n-1遍  
    {  
        min1=min2=INT\_MAX; //INT\_MAX为最大值  
        min\_i1=min\_i2=-1;  
        for (k=0;k<i;k++)  ////查找定位两个最小权重节点  
            if (tree[k].weight>=0)  //仅在根节点中找  
            {  
                if (tree[k].weight<min1)  
                {  
                    min2=min1;  
                    min\_i2=min\_i1;  
                    min1=tree[k].weight;  
                    min\_i1=k;  
                }  
                else  
                    if (tree[k].weight<min2) {  
                        min2=tree[k].weight;

                        min\_i2=k;  
                    }  
            }  
        tree[i].weight=min1+min2;  // 合并  
        tree[min\_i1].weight \*= -1;  
        tree[min\_i2].weight \*= -1;  
        tree[i].lchild=min\_i1;  
        tree[i].rchild=min\_i2;  
    }  
    return tree;  
}  
void CreateHuffcode(huffnode tree[], int i, char s[])//已知tree[i]节点的编码序列为s,求该节点下所有叶子节点的编码序列。  
{ char s1[N],c;  
    if(i!=-1)

    {  
        if (tree[i].lchild==-1 && tree[i].rchild==-1) {

            c=tree[i].data;  
            strcpy(codeset[c-32].code, s);  
        }  
        else {  
            strcpy(s1, s);  strcat(s1, "0");  
            CreateHuffcode(tree, tree[i].lchild, s1);  
            strcpy(s1, s);  strcat(s1, "1");  
            CreateHuffcode(tree, tree[i].rchild, s1);  
        }  
    }  
    return;

}

void PrintHufftree(huffnode tree[], int n)      //输出tree中的Huffman树  
{  
    int i;  
    printf("Huffman tree :\n");  
    printf("i\tValue\tLchild\tRchild\tWeight\n");  
    for(i=2\*n-2;i>=0;i--)  
    {  
        printf("%d\t",i);

        printf("%c\t",tree[i].data);  
        printf("%d\t",tree[i].lchild);  
        printf("%d\t",tree[i].rchild);  
        printf("%d\t",tree[i].weight);  
        printf("\n");  
    }  
}  
void EnCoding(char str[], char hfmstr[])  
{//根据codeset编码表,逐个将str字符串中的字符转化为它的huffman编码,结果编码串放在hfmstr字符串中  
    int i, j;  
    hfmstr[0]='\0';//把hfmstr串赋空  
    i=0;  
    while(str[i]!='\0')//从第头开始扫描str的每个字符,一直到该字符的结束  
    {  
        j=str[i]-32;//执行字符到huffman的转换  
        strcat(hfmstr, codeset[j].code);//把codest编码串添加到hfmstr结尾处  
        i++;//每次循环完i的值加1  
    }  
}  
void DeCoding(huffnode tree[], int n, char hfmstr[], char decodestr[])  
//根据tree数组中的huffman树,逐个对hfmstr字符串中的字符进行译码,结果放在decodestr字符串中  
{  
    int i=0,j=0;

    huffnode p;

    p=tree[2\*n-2];//序号2\*n-2节点就是树根节点  
    while(hfmstr[i]!='\0')//从头开始扫描每个字符,直到结束  
    {while(p.lchild!=-1&&p.rchild!=-1)//指针为空，儿子的值取完了  
    {  
        if(hfmstr[i]=='0')//为0则向左子树走  
        {  
            p=tree[p.lchild];//取出叶子节点中的字符  
        }  
        else if(hfmstr[i]=='1')//为1则向右子树走

        {  
            p=tree[p.rchild];//取出叶子节点中的字符  
        }  
        i++;  
    }  
        decodestr[j]=p.data;j++;//对字符进行译码,结果放在decodestr字符串中  
        p=tree[2\*n-2];//返回根节点  
    }  
}  
  
int main()  
{  
    int i,j;

    huffnode \* ht; //Huffman树  
    char data[N]; //要编码的字符集合  
    int  weight[N]; //字符集合中各字符的权重(频率)  
    int n=0; //字符集合中字符的个数  
    char str[1000];         //需输入的原始字符串  
    char hfm\_str[1000]=""; //编码后的字符串  
    char decode\_str[1000]="";//解码后的字符串  
    printf("请输入要转换的字符串\n");  
    gets(str);

    for(i=0;i<N;i++) { //初始化编码表,频率为0,编码串为空串

        codeset[i].count=0;  
        codeset[i].code[0]='\0';

    }  
    i=0;  
    while(str[i]!='\0') { //统计原始字符串中字符出现的频率,存入编码表  
        j=str[i]-32;  
        codeset[j].count++; //codeset[0]~[95]对应ascii码32~127的字符  
        i++;  
    }  
    for(i=0;i<N;i++) //统计原始字符串中出现的字符个数  
        if(codeset[i].count!=0) n++;  
    printf("字符频率统计:\n"); //显示统计结果  
    for(i=0;i<N;i++)  
        if(codeset[i].count!=0) printf("%c:%d, ", i+32, codeset[i].count);  
    printf("\n");  
    j=0;  
    for(i=0;i<N;i++) //生成要编码的字符集合,以及权重  
        if (codeset[i].count!=0) {  
            data[j]=i+32;  
            weight[j]=codeset[i].count;  
            j++;  
        }  
    ht=CreateHufftree(data,weight,n);  //建立Huffman树,根节点是ht[2\*n-2]  
    PrintHufftree(ht,n); //显示Huffman树的存储结果  
    CreateHuffcode(ht, 2\*n-2, "");  //以ht[2\*n-2]为根,以空字符串为起始编码字符串,求出各叶子节点的编码字符串  
    //显示codeset中的Huffman编码,参见"显示频率统计结果"的代码.  
    printf("haffman编码为:\n");  
    for(i=0;i<N;i++){  
        if(codeset[i].count!=0)  
            printf("%c:%s\n",i+32,codeset[i].code );  
    }  
    EnCoding(str, hfm\_str);  //对str字符串进行编码,放在hfm\_str字符串  
    printf("编码序列: %s\n",hfm\_str);  
    DeCoding(ht, n, hfm\_str, decode\_str); //对hfm\_str字符串进行译码,放在decode\_str字符串中  
    printf("解码后的字符串: %s\n",decode\_str);  
    free(ht);  //释放Huffman树  
}

参考文献

1百度百科

2.百度文库

3.百度学术

4.搜狗百科

5.CSDN博客

6.新浪博客

7.博客园

8.知乎