电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2017221302006

姓 名 周玉川

（实验） 课程名称 数据结构

理论教师 刘勇国

实验教师 肖老师

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告（2）**

**学生姓名：** 周玉川  **学 号：2017221302006**  **指导教师： 刘勇国，肖老师**

**实验地点：**清水河基础实验大楼 **实验时间：2018.4.13**

**一、实验室名称：**学校实验中心软件实验室

**二、实验项目名称：**编程实现哈夫曼树的构造和哈夫曼编码算法

**三、实验学时：**4

**四、实验原理：**

霍夫曼编码（Huffman Coding）是一种编码方式，是一种用于无损数据压缩的熵编码（权编码）算法。1952年，David A. Huffman在麻省理工攻读博士时所发明的。

在计算机数据处理中，霍夫曼编码使用变长编码表对源符号（如文件中的一个字母）进行编码，其中变长编码表是通过一种评估来源符号出现机率的方法得到的，出现机率高的字母使用较短的编码，反之出现机率低的则使用较长的编码，这便使编码之后的字符串的平均长度、期望值降低，从而达到无损压缩数据的目的。

例如，在英文中，e的出现机率最高，而z的出现概率则最低。当利用霍夫曼编码对一篇英文进行压缩时，e极有可能用一个比特来表示，而z则可能花去25个比特（不是26）。用普通的表示方法时，每个英文字母均占用一个字节（byte），即8个比特。二者相比，e使用了一般编码的1/8的长度，z则使用了3倍多。倘若我们能实现对于英文中各个字母出现概率的较准确的估算，就可以大幅度提高无损压缩的比例。

霍夫曼树又称最优二叉树，是一种带权路径长度最短的二叉树。所谓树的带权路径长度，就是树中所有的叶结点的权值乘上其到根结点的路径长度（若根结点为0层，叶结点到根结点的路径长度为叶结点的层数）。树的路径长度是从树根到每一结点的路径长度之和，记为WPL=(W1\*L1+W2\*L2+W3\*L3+...+Wn\*Ln)，N个权值Wi（i=1,2,...n）构成一棵有N个叶结点的二叉树，相应的叶结点的路径长度为Li（i=1,2,...n）。可以证明霍夫曼树的WPL是最小的。

**五、实验目的：**

本实验通过编程实现哈夫曼编码算法，使学生掌握哈夫曼树的构造方法，理解树这种数据结构的应用价值，并能熟练运用C语言的指针实现构建哈夫曼二叉树，培养理论联系实际和自主学习的能力，加强对数据结构的原理理解，提高编程水平。

**六、实验内容：**

（1）实现输入的英文字符串输入，并设计算法分别统计不同字符在该字符串中出现的次数，字符要区分大小写；

（2）实现哈夫曼树的构建算法；

（3）遍历哈夫曼生成每个字符的二进制编码；

（4）显示输出每个字母的编码。

**七、实验器材（设备、元器件）：**

PC机一台，装有C语言集成开发环境。

**八、数据结构与程序：**

|  |
| --- |
| **//finished by zhouyuchuan --2017221302006**  **#include <bits/stdc++.h>**  **using namespace std;**  **#define N 26**  **typedef struct{**  **char data;**  **double weight;**  **int parent;**  **int lchild;**  **int rchild;**  **}HTNode;**  **typedef struct**  **{**  **char cd[N\*2];**  **int start;**  **}HCode;**  **HTNode words[N\*4];**  **HCode output[N\*2];**  **char str[100];**  **int n=0;**  **void ReadInput(void);**  **void BuildHuffman(void);**  **void SecretOutput(void);**  **void TransSecret(void);**  **void Disp(HCode hc);**  **int main(void)**  **{**  **ReadInput();**  **BuildHuffman();**  **SecretOutput();**  **TransSecret();**  **return 0;**  **}**  **void ReadInput(void)**  **{**  **memset(words,0,sizeof(words));**  **memset(output,0,sizeof(output));**  **int i;**  **for (i=0;i<N;i++){**  **words[i].data=i+'a';**  **words[i+N].data=i+'A';**  **}**  **printf("Input: ");**  **gets(str);**  **i=0;**  **while (str[i]!='\0')**  **{**  **if (str[i]<='z'&&str[i]>='a')**  **words[str[i]-'a'].weight++;**  **else if (str[i]>='A'&&str[i]<='Z')**  **words[str[i]-'A'+N].weight++;**  **i++;**  **}**  **}**  **void BuildHuffman(void){**  **int i,k,lnode,rnode;**  **double min1,min2;**  **for (i=0;i<2\*N;i++)**  **if (words[i].weight!=0)n++;**  **for (i=0;i<4\*N-1;i++)**  **words[i].lchild=words[i].parent=words[i].rchild=-1;**  **for (i=N\*2;i<N\*2+n-1;i++)**  **{**  **min1=min2=99999;**  **lnode=rnode=-1;**  **for (k=0;k<=i-1;k++)**  **{**  **if (words[k].weight==0)continue;**  **if (words[k].parent==-1)**  **{**  **if(words[k].weight<min1)**  **{**  **min2=min1;rnode=lnode;**  **min1=words[k].weight;lnode=k;**  **}**  **else if(words[k].weight<min2)**  **{**  **min2=words[k].weight;rnode=k;**  **}**  **}**  **}**  **words[i].weight=min1+min2;**  **words[i].lchild=lnode;words[i].rchild=rnode;**  **words[lnode].parent=i;words[rnode].parent=i;**  **}**  **}**  **void SecretOutput(void)**  **{**  **int i,f,c;**  **HCode hc;**  **for (i=0;i<N\*2;i++)**  **{**  **if (words[i].weight==0)continue;**  **hc.start=n;c=i;**  **f=words[i].parent;**  **while (f!=-1)**  **{**  **if (words[f].lchild==c)**  **hc.cd[hc.start--]='0';**  **else**  **hc.cd[hc.start--]='1';**  **c=f;f=words[f].parent;**  **}**  **hc.start++;**  **output[i]=hc;**  **}**  **printf("Output: \n");**  **for (i=0;i<N\*2;i++)**  **{**  **if (words[i].weight!=0)**  **{**  **printf("%c: ",words[i].data);**  **Disp(output[i]);**  **}**  **}**  **}**  **void Disp(HCode hc)**  **{**  **int i;**  **for(i=hc.start;i<=n;i++)**  **printf("%c",hc.cd[i]);**  **printf("\n");**  **}**  **void TransSecret(void)**  **{**  **int FDot=N\*2+n-2,result;**  **char ch[1000];**  **printf("Please input the secret code:\n");**  **scanf("%s",ch);**  **int i=0;**  **for (i=0;i<strlen(ch);i++)**  **{**  **if (ch[i]=='1')**  **result=words[FDot].rchild;**  **else**  **result=words[FDot].lchild;**  **if (words[result].lchild==-1&&words[result].rchild==-1)**  **{**  **printf("%c",words[result].data);**  **FDot=N\*2+n-2;**  **}**  **else**  **FDot=result;**  **}**  **printf("\n");**    **}** |

**九、程序运行结果：**

|  |
| --- |
| **(1)实验结果1：** |
| **评价：输入一串英文字符，编码正常，输出正确。** |
| **(2):实验结果2：** |
| **评价：输入编码进行解码，输出正确，解码成功。** |
| **(3)实验结果3：** |
| **评价：页面清晰，各字符对应编码清晰明了。** |

|  |
| --- |
| **(4)实验结果4：** |
| **评价：界面一切正常，译码编码成功。** |

**十、实验结论：**

**实现哈夫曼编码，各字符对应的代码不同，且使用频率越高的字符对应的代码长度越小，方便查找。总之，实验成功，结论正确。**

**十一、总结及心得体会：**

1. **实验难点：**
2. **如何收集字符出现频率而且区分大小写：定义一个长度为26\*2的数组，初始化为0，每个字母对应一个位置，读到该字母，数组对应位置加一。**
3. **如何找树的根节点：分析发现，构造的最后一个节点即为哈夫曼树的根节点。**
4. **代码实现有难度：多看书，多理解，难度自然减小。**
5. **实验感悟：**
6. **上课认真听课，课下老老实实写代码，做实验不愁。**
7. **代码虽多，任务虽重，只要愿意去理解，愿意去熬，都会掌握的。**
8. **兴趣是推动自己学习的极大动力。**
9. **由于本人比较喜欢代码，在学ACM,一般的数据结构都是提前学过的，比如最小生成树算法（并查集真是好用），最短路径算法，老师再在讲一遍时不会慌张，学习成功比较明显。**
10. **代码能力是靠自己用双手巧出来的。**

**---auther**

**周玉川**