电子科技大学信息与软件工程学院

实 验 报 告

学　　号　2016220304022

姓　　名　罗悦

（实验）　课程名称　数据结构与算法

理论教师　陈安龙

实验教师　陈安龙

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：罗悦　　**学 号：**2016220304022　　**指导教师：**陈安龙

实验地点：清水河科技实验大楼　　**实验时间：**5.1

一、实验室名称：学校实验中心软件实验室 506

二、实验项目名称：编程实现赫夫曼树的构造和赫夫曼编码算法

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赫夫曼编码算法，使学生掌握赫夫曼树的构造方法，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赫夫曼二叉树，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）实现赫夫曼树的构建算法；

（3）遍历赫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：

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define MaxSize 1000

typedef struct {
    int weight;
    int parent;
    int lchild;
    int rchild;
}HFTreeNode;
typedef HFTreeNode HuffmanTree;

struct CodeNode{
     char ch;                       //存储字符
     char bits[MaxSize];            //存放编码位串
};
typedef struct CodeNode CodeNode;
typedef CodeNode HuffmanCode;
```

```
int n;
int cs[26]={0};
char c1[MaxSize];
HuffmanTree T[53];
HuffmanCode H1[27];
HuffmanCode H2[27];
int sum(){                              //读文本 返回出现字母的个数
    char c;
    int num=0,i=0;
    printf("请输入文本:\n");
    gets(c1);
    while(c1[i]!='\0'){
        c=c1[i];
        if(c!=' '){
            if(c<='Z')cs[c-65]++;
            else cs[c-97]++;
        }
        i++;
    }
    for(i=0;i<26;i++)
         if(cs[i]!=0)
             num++;
    return num;
}

void SelectMin(int g,int &s1,int &s2)         //确定权值最小的两个根结点
{
     int j, k, m, i;
     for (k=1; k<=g; k++){                     //找到 parent 为空的子树
          if (T[k].parent==0){
             s1=k;
             break;
          }
     }
     for (j=1; j<=g; j++){                  //找到 parent 为空的权值最小子树
          if((T[j].weight<=T[s1].weight) && (T[j].parent==0))
             s1=j;
     }
     for (m=1; m<=g; m++){                 //找到 parent 为空的不同于 s1 子树
         if((T[m].parent==0) && (m!=s1)){
             s2=m;
            break;
          }
```

```
     }
     for (i=1; i<=g; i++){        //找到 parent 为空的不同于 s1 的权值次小子树
          if((T[i].weight<=T[s2].weight) && (T[i].parent==0) && (i!=s1)){
             s2=i;
          }

     }
}

void CreateHuffmanTree(){
    int i,p1,p2,m;
    int j=0;
    if(n<1){
         printf("没有输入文本！");
        exit(0);
    }
    m=2*n;
    for(i=1;i<m;i++){
        T[i].parent=0;
        T[i].lchild=0;
        T[i].rchild=0;
        T[i].weight=0;
    }
    for(i=1;i<=n;i++){
            while(cs[j]==0)
             j++;
            T[i].weight=cs[j];
            j++;
    }

    for(i=n+1;i<m;i++){
        SelectMin(i-1,p1,p2);
        T[p1].parent=i;
        T[p2].parent=i;
        T[i].lchild=p1;
        T[i].rchild=p2;
        T[i].weight=T[p1].weight+T[p2].weight;
    }
}

void CharSetHuffmanEncoding(){
    int c,p1,i,m;
    char ch;
    int j=0;
```

```
char cd[n+1];
int start;
cd[n]='\0';
for(i=1;i<=26;i++)
    H1[i].ch='a'+i-1;
for(i=1;i<=26;i++)
    H2[i].ch='a'+i-1;
 printf("每个字符所出现的次数为：\n");
for(m=1;m<=26;m++)H1[m].ch='a'+m-1;
for(m=0;m<=25;m++){
    printf("%c:%d\t",H1[m+1].ch,cs[m]);
}
printf("\n");

printf("每个字符所对应的哈夫曼编码为：\n");
for(i=1;i<=n;i++){
    while(cs[j]==0)j++;
    ch=i+96;
    start=n;
    c=i;
    p1=T[c].parent;
    while(p1>0){
        if(T[p1].lchild==c)cd[--start]='0';
        else cd[--start]='1';
        c=p1;
        p1=T[p1].parent;

    }
    strcpy(H1[j+1].bits,&cd[start]);
    printf("%c:%s\n",H1[j+1].ch,H1[j+1].bits);
    j++;
}
printf("文本的赫夫曼编码为：\n");
while(c1[i]!='\0'){
    if(c1[i]<='Z')m=c1[i]-'A';
    else m=c1[i]-'a';
    if(c1[i]!=' ')printf("%s",H1[m+1].bits);
    i++;
 }
 while(c1[i]!='\0'){
    if(c1[i]<='Z')m=c1[i]-'A';
    else m=c1[i]-'a';
    if(c1[i]!=' ')printf("%s",H1[m+1].bits);
    i++;
```

```
    }
}


int main(){
    int i=0;
    n=sum();
    CreateHuffmanTree();
    CharSetHuffmanEncoding();
    return 0;
}
```

九、程序运行结果：

十、实验结论：

赫夫曼编码可以清晰的显示出文本中每个字符所出现的次数和每个字符所对应的编码。并且此实验运用最优二叉树的方法使对文本中的字符进行了精准的遍历。

十一、总结及心得体会：

此次实验通过编程实现赫夫曼编码算法，使我掌握赫夫曼树的构造方法，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赫夫曼二叉树，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。在构建最优二叉树的同时会遇到许多 C 语言上的问题，但我通过学习和实践都一一解决了这些问题。本实验之所以运用最优二叉树，是因为二叉树模型算法思想比较简单易懂，即使是在二叉树步数较大时，仍可以精确地获得理论价格，且对于美式、欧式期权均适用。二叉树模型也有一些不足之处，如在步

数较少时，只能对理论价格求得近似解，精确度不佳，而在步数过大时，计算复杂度较高，且同样不适用其他类型的期权。总之，赫夫曼编码巧妙地运用了最优二叉树，使得所得到的编码简便化并且准确的输出。