电子科技大学信息与软件工程学院

**实 验 报 告**

学 号

姓 名

（实验） 课程名称 数据结构与算法

理论教师 郝宗波

实验教师 李美蓉

**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：**  **学 号：**  **指导教师： 郝宗波**

**实验地点：**清水河科技实验大楼 **实验时间：2017.04.21**

**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 MAX\_SIZE 20

#define MAX\_CHARLEN 10

typedef struct {

    char data;

    int weight;

    int parent;

    int lchild;

    int rchild;

}HFTreeNode, \*HFTree;

typedef struct CodeNode

{

    char ch; //存储字符

    char bits[MAX\_CHARLEN+1]; //存放编码位串

}HuffmanCode;

typedef struct CodeNode

{

    char ch; //存储字符

    char bits[MAX\_CHARLEN+1]; //存放编码位串

}HuffmanCode;

void countTimes(int count[], int\* num) {

    int i = 0, c;

    FILE\* fp;

    char ch;

    for (i = 0; i < 52; i++){

        count[i] = 0;

    }

    fp=fopen("/Users/ryshen/Desktop/test.txt","r");

    if(!fp)

    {

        printf("The file does not exist!\n");

        exit(0);

    }

    for (i = 0; fscanf(fp,"%c",&ch)!=EOF; i++) {

        if (ch >= 97) {

            c = ch - 'a';

        }

        else {

            c = ch - 'A' + 26;

        }

        count[c]++;

    }

    fclose(fp);

    for (i = 0; i < 52; i++){

        if (count[i] != 0) {

            (\*num)++;

        }

    }

}

void SelectMin (HFTree T, int n, int\* p1, int\* p2){

    int i, j;

    for (i = 1; i <= n; i++){

        if (T[i].parent == 0){

            (\*p1) = i;

            break;

        }

    }

    for (j = 1; j <= n; j++){

        if ((T[j].parent == 0) && (T[j].weight < T[i].weight)){

            (\*p1) = j;

            i = j;

        }

    }

    for (i = 1; i <= n; i++){

        if ((T[i].parent == 0) && i != (\*p1)){

            (\*p2) = i;

            break;

        }

    }

    for (j = 1; j <= n; j++){

        if ((T[j].parent == 0) && T[j].weight < T[i].weight && j != (\*p1)){

            (\*p2) = j;

            i = j;

        }

    }

}

int createHFTree(HFTreeNode T[], int sum, int count[]){

    int i, p1 = 0, p2 = 0, c = 1;

    if (sum < 1 || sum > MAX\_SIZE){

        printf("sum is illegal\n");

        return 1;

    }

    for (i = 1; i < 2 \* sum; i++){

        T[i].parent = 0;

        T[i].lchild = 0;

        T[i].rchild = 0;

        T[i].weight = 0;

    }

    for (i = 0; i <= 52 && c<=sum; i++){

        if (count[i] != 0){

            T[c].weight = count[i];

            if (i < 26){

                T[c].data =  i + 'a';

            }

            else {

                T[c].data =  i + 'A' - 26;

            }

            c += 1;

        }

    }

    for (i = sum + 1; i <= 2 \* sum; i++){

        SelectMin (T, 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;

    }

    return 0;

}

void CharSetHuffmanEncoding(HFTreeNode T[], HuffmanCode H[], int sum) {

    int c, p, i;

    char cd[sum + 1];

    int start = 0;

    cd[sum] = '\0';

    for(i = 1; i <= sum; i++){

        H[i].ch = T[i].data;

        start = sum;

        c = i;

        while((T[c].parent > 0)) {

            p = T[c].parent;

            if(T[p].lchild == c)  {

                cd[--start] = '0';

            }

            else {

                cd[--start] = '1';

            }

            c = p;

        }

        strcpy(H[i].bits, &cd[start]);

    }

}

int main() {

    int count[52];

    int num, i;

    countTimes(count, &num);

    HuffmanCode H[MAX\_CHARLEN];

    HFTreeNode T[2\*num];

    createHFTree(T, num, count);

    for (i = 0; i < 52; i++){

        if (count[i] != 0) {

            if (i < 26){

                printf("%c:%d\t", i+'a', count[i]);

            }

            else{

                printf("%c:%d\t", i+'A'-26, count[i]);

            }

        }

    }

    printf("\n");

    CharSetHuffmanEncoding(T, H, num);

    for (i = 1; i <= num; i++) {

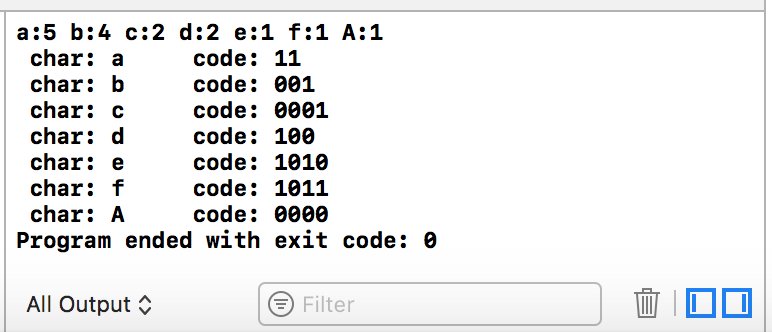
        printf(" char: %c\t code: %s\n", H[i].ch, H[i].bits);

    }

    return 0;

}

**九、程序运行结果：**

****

**十、实验结论：**

测试正确，程序满足项目需求。

**十一、总结及心得体会：**

总结：

这次实验主要考察了对哈夫曼树和哈夫曼编码的理解和掌握，同时也考察了C语言对文件的操作。第一题的关键点在于要想到用数组来存储读入的字符，这样才能方便计数，而对数组下标的选择，进行一些减法操作后就能节省很多浪费的空间，也是一个比较重要的地方。

第二题对哈夫曼树的构建就是考察对老师上课所讲的内容的掌握和应用。创建树的时候，太心急了，没有想清楚逻辑什么的，就直接开始乱搞，导致循环赋值的逻辑有问题，又找了很久的错，惨痛的教训啊。。找权重小的结点时又因为懒直接复制粘贴，结果i，j搞反又翻车，所以说，不要偷懒啊。

第三题，没有认真分析就开始做，传了个没用的字符参数，后来同学告诉我才发现，所以以后要先认真分析自己有什么，要得到的是什么啊。。还有，debug的能力真的很重要啊。。加油吧少年！

心得体会：

这次的实验其实做了挺久，主要是老师讲了之后没有及时复习，都忘光了，所以做的时候不太会，所以说，还是得及时复习才会事半功倍啊。而且原理也懂的，就是会在很多意想不到的地方翻车，就会找错找很久，比如经常把i，j写反之类的辣鸡错误，或者是空指针就开始乱搞了，感觉C语言的知识也不太会了，还是得复习一下了。这次实验最大的收获就是，虽然老师讲的都会，但是还是得自己动手实践，多复习思考才能真正的掌握，不然会费时费力还达不到预期的效果。