

数据结构实验报告

课程名称：数据结构		班级：计科 151												
姓名：刘禾子		学号：1500170082				指导教师：刘长云								
实验序号：五						实验成绩：								
一、实验名称														
哈夫曼编码/译码器														
二、实验目的及要求														
1、掌握二叉树的存储方法；														
2、掌握 Huffman 二叉树的构造、编码和译码算法；														
三、实验环境														
Visual C++														
四、实验内容														
1、生成编码二叉树														
输入为字符集和词频，输出为哈夫曼二叉树。														
字符集和词频如下：														
字 符	空 格	A	B	C	D	E	F	G	H	I	J	K	L	M
频 度	186	64	13	22	32	103	21	15	47	57	1	5	32	20
字 符	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
词 频	57	63	15	1	48	51	80	23	8	18	1	16	1	
2、编码器设计														
输入为二叉树和要编码的内容，输出为编码结果														
3、译码器设计														
输入为二叉树和要解码的内容，输出为译码内容														

五、算法描述及实验步骤

(1) **I**: 初始化。从终端读入字符集大小 n ，以及 n 个字符和 n 个权值，建立哈夫曼树，并将它存于文件 `hfmTree` 中。

(2) **E**: 编码。利用已建好的哈夫曼树（如不在内存，则从文件 `hfmTree` 中读入），对文件 `ToBeTran` 中的正文进行编码，然后将结果存入文件 `CodeFile` 中。

(3) **D**: 译码。利用已建好的哈夫曼树将文件 `CodeFile` 中的代码进行译码，结果存入文件 `TextFile` 中。

(4) **P**: 印代码文件。将文件 `CodeFile` 以紧凑格式显示在终端上，每行 50 个代码。同时将此字符形式的编码文件写入文件 `CodePrin` 中。

(5) **T**: 印哈夫曼树。将已在内存中的哈夫曼树以直观的方式（树或凹入表形式）显示在终端上，同时将此字符形式的哈夫曼树写入文件 `TreePrint` 中。

六、调试过程及实验结果

调试过程：

在调试中对文件操作多次出现异常，经过多次调试之后才勉强完成，对于构建哈夫曼树的时候遍历左孩子，递归调用时出现指针访问冲突，设置断点之后可看到正确的实验结果。

程序清单：

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#define MAX 100
#define MAXVALUE 10000
typedef struct{
    char ch;
    int weight, flag;
    int parent, lchild, rchild;
}HTNode;
typedef struct{
    char ch;
    int bit[MAX];
    int start, weight;
}Code;
typedef struct{
    char ch;
    char bit[MAX];
    int weight;
}Coding;
void HaffmanCoding1(int weight[], char ch[], int n, HTNode haffTree[]){
    /*生成哈夫曼树的函数1*/
    int i, j, m1, m2, x1, x2;
    for (i = 0; i<2 * n - 1; i++){
```

```

        if (i < n) {
            haffTree[i].weight = weight[i];
            haffTree[i].ch = ch[i];
        }
        else haffTree[i].weight = 0;
        haffTree[i].parent = -1;
        haffTree[i].flag = 0;
        haffTree[i].lchild = -1;
        haffTree[i].rchild = -1;
    }
    for (i = 0; i < n - 1; i++) {
        m1 = m2 = MAXVALUE;
        x1 = x2 = 0;
        for (j = 0; j < n + i; j++) {
            if (haffTree[j].weight < m1 && haffTree[j].flag == 0) {
                m2 = m1;
                x2 = x1;
                m1 = haffTree[j].weight;
                x1 = j;
            }
            else if (haffTree[j].weight < m2 && haffTree[j].flag == 0) {
                m2 = haffTree[j].weight;
                x2 = j;
            }
        }
        haffTree[x1].parent = n + i;
        haffTree[x2].parent = n + i;
        haffTree[x1].flag = 1;
        haffTree[x2].flag = 1;
        haffTree[n + i].weight = haffTree[x1].weight + haffTree[x2].weight;
        haffTree[n + i].lchild = x1;
        haffTree[n + i].rchild = x2;
    }
    /*把哈弗曼树存储到huffman.txt中。*/
    FILE *fp;
    fp = fopen("huffman.txt", "w+");
    printf("%d\n", n);
    fprintf(fp, "%d\n", n);
    for (i = 0; i < n; i++)
        fprintf(fp, "%c %d %d %d\n", haffTree[i].ch, haffTree[i].parent,
haffTree[i].lchild, haffTree[i].rchild);
    for (i = n; i < 2 * n - 1; i++)
        fprintf(fp, "%d %d %d\n", haffTree[i].parent, haffTree[i].lchild,
haffTree[i].rchild);

```

```

    fclose(fp);
}

void HaffmanCoding2(HTNode haffTree[], int n, Code haffCode[]) {
    /*生成哈夫曼树的函数2*/
    Code *cd = (Code *)malloc(sizeof (Code));
    int i, j, child, parent;
    for (i = 0; i < n; i++) {
        cd->start = n - 1;
        cd->weight = haffTree[i].weight;
        cd->ch = haffTree[i].ch;
        child = i;
        parent = haffTree[child].parent;
        while (parent != -1) {
            if (haffTree[parent].lchild == child)
                cd->bit[cd->start] = 0;
            else
                cd->bit[cd->start] = 1;
            cd->start--;
            child = parent;
            parent = haffTree[child].parent;
        }
        for (j = cd->start + 1; j < n; j++)
            haffCode[i].bit[j] = cd->bit[j];
        haffCode[i].start = cd->start + 1;
        haffCode[i].weight = cd->weight;
        haffCode[i].ch = cd->ch;
    }
}

void Initialization(int weight[], char ch[]) {
    /*构造n个字符的赫夫曼编码HC*/
    FILE *fp;
    int i, j, n;
    char ch1, wj[15];
    printf("系统正在初始化。。。 \n请输入字符集大小n:\n");
    scanf("%d", &n);
    HTNode *myHaffTree = (HTNode *)malloc(sizeof (HTNode)*(2 * n + 1));
    Code *myHaffCode = (Code *)malloc(sizeof (Code)*n);
    for (i = 0; i < n; i++) {
        printf("请输入字符和权值:\n");
        getchar();
        ch[i] = getchar();
        scanf("%d", &weight[i]);
    }
    HaffmanCoding1(weight, ch, n, myHaffTree);
}

```

```

HaffmanCoding2(myHaffTree, n, myHaffCode);
fp = fopen("hfmTree.txt", "w+");
for (i = 0; i < n; i++) {
    printf("%c %d ", myHaffCode[i].ch, myHaffCode[i].weight);
    fprintf(fp, "%c %d ", myHaffCode[i].ch, myHaffCode[i].weight);
    for (j = myHaffCode[i].start; j < n; j++) {
        printf("%d", myHaffCode[i].bit[j]);
        fprintf(fp, "%d", myHaffCode[i].bit[j]);
    }
    fprintf(fp, "\n");
    printf("\n");
}
fclose(fp);
printf("初始化成功!\n");
}

void Encoding() {
    /*利用以建好的哈弗曼树(如不存在, 从文件hfmTree中读)输入正文进行编码, 然后将结果存入
    文件CodeFile中*/
    FILE *fp, *fp1, *fp2;
    char zf[500];
    fp = fopen("hfmTree.txt", "r");
    Coding ch[100];
    char c;
    int i = 0;
    while (feof(fp) == 0) {
        fscanf(fp, "%s %d %s", &ch[i].ch, &ch[i].weight, &ch[i].bit);
        i++;
    }
    fclose(fp);
    printf("现在进行编码操作。。。\n请输入字符串:\n");
    scanf("%s", zf);
    char ch1[20], ch2[20];
    int j;
    fp2 = fopen("CodeFile.txt", "w+");
    int len, k;
    len = strlen(zf);
    for (k = 0; k < len; k++)
        for (j = 0; j < i; j++)
            if (ch[j].ch == zf[k]) {
                fprintf(fp2, "%s", ch[j].bit);
                printf("%s", ch[j].bit);
            }
    printf("\n");
    fclose(fp2);
}

```

```

    printf("编码完成!已将结果存入CodeFile.txt中.\n\n");
}

void Decoding() {
    /*利用已建好的哈弗曼树将文件CodeFile中的代码进行译码, 结果存入文件TextFile中。*/
    FILE *fp, *fp1;
    fp = fopen("huffman.txt", "r");
    int i, n;
    fscanf(fp, "%d", &n);
    HTNode *myHaffTree = (HTNode *)malloc(sizeof (HTNode)*(2 * n + 1));
    for (i = 0; i < n; i++)
        fscanf(fp, "%s %d %d %d\n", &myHaffTree[i].ch, &myHaffTree[i].parent,
&myHaffTree[i].lchild, &myHaffTree[i].rchild);
    for (i = n; i < 2 * n - 1; i++)
        fscanf(fp, "%d %d %d\n", &myHaffTree[i].parent, &myHaffTree[i].lchild,
&myHaffTree[i].rchild);
    fclose(fp);
    fp = fopen("CodeFile.txt", "r");
    fp1 = fopen("TextFile.txt", "w+");
    char ch;
    i = 2 * n - 2;
    while (!feof(fp)) {
        fscanf(fp, "%c", &ch);
        if (ch == '0') i = myHaffTree[i].lchild;
        if (ch == '1') i = myHaffTree[i].rchild;
        if (i < n) {
            printf("%c", myHaffTree[i].ch);
            fprintf(fp1, "%c", myHaffTree[i].ch);
            i = 2 * n - 2;
        }
    }
    printf("\n");
    fprintf(fp1, "\n");
    fclose(fp);
    fclose(fp1);
    printf("译码过程完成!已将结果存入TextFile.txt中.\n\n");
}

void Print() {
    /*将文件CodeFile以紧凑格式显示在终端上, 每行50个代码同时将此字符形式的编码文件写入
文件CodePrin中。*/
    FILE *fp1, *fp2;
    fp1 = fopen("CodeFile.txt", "r");
    fp2 = fopen("CodePrin.txt", "w+");
    int count = 0;
    char ch;

```

```

while (!feof(fp1)) {
    fscanf(fp1, "%c", &ch);
    printf("%c", ch);
    fprintf(fp2, "%c", ch);
    count++;
    if (count == 50) {
        printf("\n");
        fprintf(fp2, "\n");
        count = 0;
    }
}
printf("\n");
fprintf(fp2, "\n");
fclose(fp1);
fclose(fp2);
printf("打印代码过程完成!已将结果存入CodePrint.txt中.\n\n");
}

void PrintTree(HTNode *huf, int n, int p, FILE *fp)
{
    /*打印哈弗曼树函数树是横向显示的, 叶子节点显示字符, 非叶子节点显示'@'*/
    int i;
    if (n == -1) return;
    PrintTree(huf, huf[n].rchild, p + 1, fp);
    for (i = 0; i < p; i++) {
        printf(" ");
        fprintf(fp, " ");
    }
    if (p >= 0 && huf[n].rchild == -1) {
        printf("——");
        printf("%c\n", huf[n].ch);
        fprintf(fp, "——%c\n", huf[n].ch);
    }
    else {
        printf("@\n");
        fprintf(fp, "@\n");
    }
    PrintTree(huf, huf[n].lchild, p + 1, fp);
}

void Treeprinting() {
    /*将已在内存中的哈弗曼树以直观的方式(树或凹入表形式)显示在终端上同时将次字符形式的
    哈弗曼树写入文件TreePrint中。*/
    FILE *fp;
    fp = fopen("huffman.txt", "r");

```

```

int i, n;
fscanf(fp, "%d", &n);
HTNode *myHaffTree = (HTNode *)malloc(sizeof (HTNode)*(2 * n + 1));
for (i = 0; i<n; i++)
    fscanf(fp, "%s %d %d %d\n", &myHaffTree[i].ch, &myHaffTree[i].parent,
&myHaffTree[i].lchild, &myHaffTree[i].rchild);
for (i = n; i<2 * n - 1; i++)
    fscanf(fp, "%d %d %d\n", &myHaffTree[i].parent, &myHaffTree[i].lchild,
&myHaffTree[i].rchild);
fclose(fp);
fp = fopen("TreePrint.txt", "w+");
PrintTree(myHaffTree, 2 * n - 2, 0, fp);
fclose(fp);
printf("打印哈夫曼树过程完成!同时已将结果存入TreePrint中.\n\n");
}
void Menuprint() {
    printf("*****\n");
    printf("*****\n");
    printf("*****\n");
    printf("*****      欢迎使用哈夫曼编 / 译码器      \n");
    printf("*****\n");
    printf("*****\n");
    printf("***** E. 编码      D. 译码      P. 印代码文件      T. 印哈夫曼树      Q. 退出\n");
    printf("*****\n");
    printf("*****\n");
}
int main()
{
    int i, j, n = 4;
    int weight[100];
    char ch[100], cha;
    Menuprint();
    Initialization(weight, ch);
    while (1) {
        printf("请输入要执行的操作:\nE. 编码      D. 译码      P. 印代码文件      T. 印哈
夫曼树      Q. 退出\n");
        printf("请输入要执行的操作:\n");
        scanf("%s", &cha);
        if (cha == 'Q') break;
        switch (cha) {
            case 'E': Encoding(); break;
            case 'D': Decoding(); break;
            case 'P': Print(); break;
            case 'T': Treeprinting(); break;
        }
    }
}

```



```

    }
}
return 0;
}

```

实验结果：

只选取了前面 6 个字符做测试：

```

E:\Software\Visual Studio 2010\项目组\哈夫曼编码器\Debug\哈夫曼编码器.exe
*****
*****          欢迎使用哈夫曼编 / 译码器          *****
*****
*****          E. 编码      D. 译码      P. 印代码文件      T. 印哈夫曼树      Q. 退出      *****
*****
*****
*****
系统正在初始化。。。
请输入字符集大小n:
6
请输入字符和权值:
186
请输入字符和权值:
A64
请输入字符和权值:
B13
请输入字符和权值:
C22
请输入字符和权值:
D32
请输入字符和权值:
E103
6
186 0
A 64 110
B 13 11110
C 22 11111
微软拼音 半 :
E 103 10
初始化成功!
微软拼音 半 :的操作:

请输入要执行的操作:
E. 编码      D. 译码      P. 印代码文件      T. 印哈夫曼树      Q. 退出
请输入要执行的操作:
E
现在进行编码操作。。。
请输入字符串:
BCBC AD
编码完成!已将结果存入CodeFile.txt中.

请输入要执行的操作:
E. 编码      D. 译码      P. 印代码文件      T. 印哈夫曼树      Q. 退出
请输入要执行的操作:
请输入要执行的操作:
E. 编码      D. 译码      P. 印代码文件      T. 印哈夫曼树      Q. 退出
请输入要执行的操作:
D
译码过程完成!已将结果存入TextFile.txt中.

```

打印代码过程完成! 已将结果存入CodePrin.txt中.

请输入要执行的操作:

E. 编码

D. 译码

P. 印代码文件

T. 印哈夫曼树

Q. 退出

请输入要执行的操作:

T

```

      ---C
      @
      ---B
      @
      ---D
      @
      ---A
      @
      ---E
      @
```

hfmTree.txt - 记事本

文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
186 0
A 64 110
B 13 11110
C 22 11111
D 32 1110
E 103 10
```

huffman.txt - 记事本

文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
6
10 -1 -1
A 8 -1 -1
B 6 -1 -1
C 6 -1 -1
D 7 -1 -1
E 9 -1 -1
7 2 3
8 4 6
9 1 7
10 5 8
-1 0 9
```

打印出的哈夫曼树是倒置的。

七、总结

- 1.对于 C 语言文件操作有待提高,写文件与读文件操作总会出不了结果这是可以进一步提高的,对于字符串指针的操作经过这次实验后也有了不错的认识。
- 2.构建哈夫曼树的过程是从叶子结点到根逐个建立的,而遍历采用递归,先遍历左子树,再遍历右子树。
- 3.最后的译码操作可后期修改之后显示在终端上,但是文件内写入无结果有待改进。