**《数据结构》**

**课程设计报告**

|  |  |
| --- | --- |
| 课程名称： | 《数据结构》课程设计 |
| 课程设计题目： | 哈夫曼编码 |
| 姓 名： | 应宇杰 |
| 院 系： | 计算机 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 19052321 |
| 学 号： | 19151633 |
| 指导教师： | 葛瑞泉 |

2020年12月08日

1. **需求分析**

#### 功能需求：

1、模拟哈夫曼编码的形成，编码哈夫曼数

2、输入数字解码哈夫曼数

3、把输出保存在文件中

#### 界面需求：

分别打印编码好的哈夫曼编码并保存进各txt文件中

1. **概要设计**

输入abcd 4个字母的权重分别为1，5，7，4

哈夫曼编码是先找两个最小组成一个新的结点

#### 取过的两个就不能再取了，依次取一个最小和次小组合到一起，得到哈夫曼图

c

B

d

a

#### 最后得到各个字母的编码

A：110

B: 10

C: 0

D: 111

#### 接口设计

void tips(); //打印操作选择界面

void HuffmanCoding(HuffmanTree&, char\*, int\*, int); //建立赫夫曼树的算法

void select(HuffmanTree HT, int j, int\* x, int\* y); //从已建好的赫夫曼树中选择parent为0，weight最小的两个结点

void Init();

void Coding(); //编码

void Decoding(); //译码

void Print\_code(); //打印译码好的代码

void Print\_tree(); //打印哈夫曼树

int Read\_tree(HuffmanTree&); //从文件中读入赫夫曼树

void find(HuffmanTree& HT, char\* code, char\* text, int i, int m); //译码时根据01字符串寻找相应叶子节点的递归算法

void Convert\_tree(unsigned char T[100][100], int s, int\* i, int j); //将存中的赫夫曼树转换成凹凸表形式的赫夫曼树

#### 数据结构设计

typedef struct {

char ch; //增加一个域，存放该节点的字符

int parent, lchild, rchild;

int weight;

}HTNode, \* HuffmanTree;

1. **详细设计**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct {

char ch; //增加一个域，存放该节点的字符

int parent, lchild, rchild;

int weight;

}HTNode, \* HuffmanTree;

typedef char\*\* HuffmanCode;

void tips(); //打印操作选择界面

void HuffmanCoding(HuffmanTree&, char\*, int\*, int); //建立赫夫曼树的算法

void select(HuffmanTree HT, int j, int\* x, int\* y); //从已建好的赫夫曼树中选择parent为0，weight最小的两个结点

void Init();

void Coding(); //编码

void Decoding(); //译码

void Print\_code(); //打印译码好的代码

void Print\_tree(); //打印哈夫曼树

int Read\_tree(HuffmanTree&); //从文件中读入赫夫曼树

void find(HuffmanTree& HT, char\* code, char\* text, int i, int m); //译码时根据01字符串寻找相应叶子节点的递归算法

void Convert\_tree(unsigned char T[100][100], int s, int\* i, int j); //将存中的赫夫曼树转换成凹凸表形式的赫夫曼树

HuffmanTree HT; //全局变量

int n = 0; //全局变量，存放赫夫曼树叶子结点的数目

int main() {

char select; while (1) {

tips();

scanf("%c", &select);

switch (select) //选择操作，根据不同的序号选择不同的操作

{

case '1':Init(); break;

case '2':Coding(); break;

case '3':Decoding(); break;

case '4':Print\_tree(); break;

case '0':exit(1);

default:printf(" 输入选项错误!\n");

}

getchar();

}

return 0;

}

void tips() // 操作选择界面

{

printf(" 请输入1来初始化\n");

printf(" 请输入2来编码\n");

printf(" 请输入3来译码\n");

printf(" 请输入4来打印赫夫曼树\n");

printf(" 请输入0来退出\n");

}

//初始化函数，输入n个字符及其对应的权值，根据权值建立哈夫曼树，并将其存于文件Herffman中

void Init()

{

FILE\* fp;

int i, n, w[52]; //w数组存放字符的权值

char character[52];

printf("请输入字符个数 :");

scanf("%d", &n); //输入字符集大小

printf("请输入%d个字符及其对应的权值:\n", n);

for (i = 0; i < n; i++) {

char b = getchar();

scanf("%c", &character[i]);

scanf("%d", &w[i]); //输入n个字符和对应的权值

}

HuffmanCoding(HT, character, w, n); //建立赫夫曼树

if ((fp = fopen("Herffmantree.txt", "w")) == NULL)

printf("打开文件错误!\n");

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

if (fwrite(&HT[i], sizeof(HTNode), 1, fp) != 1) //将建立的赫夫曼树存入文件Herffmantree.txt中

printf("文件写入错误!\n");

}

printf("赫夫曼树建立成功，并已存于文件Herffmantree.txt中\n");

fclose(fp);

}

//建立赫夫曼树

void HuffmanCoding(HuffmanTree& HT, char\* character, int\* w, int n)

{

int m, i, x, y;

HuffmanTree p;

if (n <= 1) return;

m = 2 \* n - 1;

HT = (HuffmanTree)malloc((m + 1) \* sizeof(HTNode));

for (p = HT + 1, i = 1; i <= n; ++i, ++p, ++character, ++w)

{

p->ch = \*character; p->weight = \*w; p->parent = 0; p->lchild = 0; p->rchild = 0;

}

for (; i <= m; ++i, ++p)

{

p->ch = 0; p->weight = 0; p->parent = 0; p->lchild = 0; p->rchild = 0;

}

for (i = n + 1; i <= m; ++i) {

select(HT, i - 1, &x, &y); //从树中挑两个最小的

HT[x].parent = i;

HT[y].parent = i;

HT[i].lchild = x;

HT[i].rchild = y;

HT[i].weight = HT[x].weight + HT[y].weight;

}

}

//从HT[1]到HT[j]中选择parent为0，weight最小的两个结点，用x和y返回其序号

void select(HuffmanTree HT, int j, int\* x, int\* y)

{

int i;

//查找weight最小的结点

for (i = 1; i <= j; i++)

if (HT[i].parent == 0) { \*x = i; break; }

for (; i <= j; i++)

if ((HT[i].parent == 0) && (HT[i].weight < HT[\*x].weight))

\*x = i;

HT[\*x].parent = 1; //查找weight次小的结点

for (i = 1; i <= j; i++)

if (HT[i].parent == 0)

{

\*y = i; break;

} for (; i <= j; i++)

if ((HT[i].parent == 0) && (i != \*x) && (HT[i].weight < HT[\*y].weight))

\*y = i;

}

//对文件tobetrans中的正文进行编码，然后将结果存入文件codefile中

void Coding()

{

FILE\* fp, \* fw;

int i, f, c, start;

char\* cd;

HuffmanCode HC;

if (n == 0)

n = Read\_tree(HT);

//从文件hfmtree.txt中读入赫夫曼树, 返回叶子结点数

//求赫夫曼树中各叶子节点的字符对应的的编码，并存于HC指向的空间中

HC = (HuffmanCode)malloc((n + 1) \* sizeof(char\*));

cd = (char\*)malloc(n \* sizeof(char));

cd[n - 1] = '\0';

for (i = 1; i <= n; ++i) {

start = n - 1;

for (c = i, f = HT[i].parent;f!= 0; c = f,f = HT[f].parent)

if (HT[f].lchild == c)

cd[--start] = '0';

else

cd[--start] = '1';

HC[i] = (char\*)malloc((n - start) \* sizeof(char));

strcpy(HC[i], &cd[start]);

} free(cd);

if ((fp = fopen("tobetrans.txt", "rb+")) == NULL)

printf("打开文件 tobetrans.txt 错误!\n");

if ((fw = fopen("codefile.txt", "wb+")) == NULL)

printf("打开文件 codefile.txt 错误!\n");

char temp;

fscanf(fp, "%c", &temp); //从文件读入第一个字符

while (!feof(fp)) {

for (i = 1; i <= n; i++)

if (HT[i].ch == temp)

break; //在赫夫曼树中查找字符所在的位置

for (int r = 0; HC[i][r] != '\0'; r++) //将字符对应的编码存入文件

fputc(HC[i][r], fw);

fscanf(fp, "%c", &temp); //从文件读入下一个字符

}

fclose(fw); fclose(fp);

printf("\n已将文件hfmtree.txt成功编码,并已存入codefile.txt中！\n\n");

}

//将文件codefile中的代码进行译码，结果存入文件textfile中

void Decoding() {

FILE\* fp, \* fw; int m, i;

char\* code, \* text, \* p;

if (n == 0)

n = Read\_tree(HT);//从文件hfmtree.txt中读入赫夫曼树,返回叶子结点数

if ((fp = fopen("codefile.txt", "rb")) == NULL)

printf("打开文件 codefile.txt 错误!\n");

if ((fw = fopen("textfile.txt", "wb+")) == NULL)

printf("打开文件 textfile.txt 错误!\n"); code = (char\*)malloc(sizeof(char));

fscanf(fp, "%c", code); //从文件读入一个字符

for (i = 1; !feof(fp); i++) {

code = (char\*)realloc(code, (i + 1) \* sizeof(char)); //增加空间

fscanf(fp, "%c", &code[i]); //从文件读入下一个字符

}

code[i - 1] = '\0'; // codefile.txt文件中的字符已全部读入，存放在code数组中

text = (char\*)malloc(100 \* sizeof(char));

p = text;

m = 2 \* n - 1;

if (\*code == '0')

find(HT, code, text, HT[m].lchild, m); //从根节点的左子树去找

else

find(HT, code, text, HT[m].rchild, m); //从根节点的右子树去找

for (i = 0; p[i] != '\0'; i++) //把译码好的字符存入文件textfile.txt中

fputc(p[i], fw);

fclose(fp); fclose(fw);

printf("\n已将codefile.txt文件成功译码，兵已存入textfile.txt文件！\n\n");

}

//将文件codefi1e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codeprint中。

//void Print\_code() {

// FILE\* fp, \* fw; char temp; int i;

// if ((fp = fopen("codefile.txt", "rb")) == NULL)

// printf("打开文件 codefile.txt 错误!\n");

// if ((fw = fopen("codeprint.txt", "wb+")) == NULL)

// printf("打开文件 codeprint.txt 错误!\n");

// printf("\n文件codefi1e显示如下:\n");

// fscanf(fp, "%c", &temp); //从文件读入一个字符

// for (i = 1; !feof(fp); i++) {

// printf("%c", temp); if (i % 50 == 0) printf("\n");

// fputc(temp, fw); //将该字符存入文件codeprint.txt中

// fscanf(fp, "%c", &temp); //从文件读入一个字符

// }

// printf("\n\n已将此字符形式的编码写入文件codeprint.txt中！\n\n");

// fclose(fp);

// fclose(fw);

//}

//将已在存中的哈夫曼树显示在屏幕上，并将此字符形式的哈夫曼树写入文件treeprint中。

void Print\_tree() {

unsigned char T[100][100];

int i, j, m = 0;

FILE\* fp;

if (n == 0)

n = Read\_tree(HT); //从文件hfmtree.txt中读入赫夫曼树,返回叶子结点数

Convert\_tree(T, 0, &m, 2 \* n - 1); //将存中的赫夫曼树转换成凹凸表形式的树，存于数组T中

if ((fp = fopen("treeprint.txt", "wb+")) == NULL)

printf("打开文件 treeprint.txt 错误!\n");

printf("\n打印已建好的赫夫曼树：\n");

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

for (j = 0; T[i][j] != 0; j++) {

if (T[i][j] == ' ') { printf(" "); fputc(T[i][j], fp); }

else

{

printf("%d", T[i][j]); fprintf(fp, "%d\n", T[i][j]);

}

}

printf("\n");

}

fclose(fp);

printf("\n已将该字符形式的哈夫曼树写入文件treeprint.txt中！\n\n");

}

//从文件Herffman.txt中读入赫夫曼树，返回叶子节点数

int Read\_tree(HuffmanTree& HT)

{

FILE\* fp;

int i, n;

HT = (HuffmanTree)malloc(sizeof(HTNode));

if ((fp = fopen("Herffmantree.txt", "r")) == NULL)

printf("打开文件 Herffmantree.txt 错误!\n");

for (i = 1; !feof(fp); i++) {

HT = (HuffmanTree)realloc(HT, (i + 1) \* sizeof(HTNode)); //增加空间

fread(&HT[i], sizeof(HTNode), 1, fp); //读入一个节点信息

}

fclose(fp);

n = (i - 1) / 2;

return n;

}

//译码时根据01字符串寻找相应叶子节点的递归算法

void find(HuffmanTree& HT, char\* code, char\* text, int i, int m) {

if (\*code != '\0') //若译码未结束

{

code++;

if (HT[i].lchild == 0 && HT[i].rchild == 0) //若找到叶子节点

{

\*text = HT[i].ch; //将叶子节点的字符存入text中

text++;

if ((\*code == '0'))

find(HT, code, text, HT[m].lchild, m); //从根节点的左子树找

else

find(HT, code, text, HT[m].rchild, m); //从根节点的右子树找

}

else //如果不是叶子节点

if (\*code == '0')

find(HT, code, text, HT[i].lchild, m); //从该节点的左子树去找

else

find(HT, code, text, HT[i].rchild, m); //从该节点的右子树去找

}

else

\*text = '\0'; //译码结束

}

//将文件中的赫夫曼树转换成凹凸表形式的赫夫曼树打印出来

void Convert\_tree(unsigned char T[100][100], int s, int\* i, int j) {

int k, l;

l = ++(\*i);

for (k = 0; k < s; k++)

T[l][k] = ' ';

T[l][k] = HT[j].weight;

if (HT[j].lchild)

Convert\_tree(T, s + 1, i, HT[j].lchild);

if (HT[j].rchild)

Convert\_tree(T, s + 1, i, HT[j].rchild);

T[l][++k] = '\0';

}

**四、调试分析**

1、输入编码时一定要先字母在权值

2、在打开文件时遇到打不开文件或者不能自动创建文件的问题，把fp = fopen("treeprint.txt", "wb+")中改成wb+问题解决。

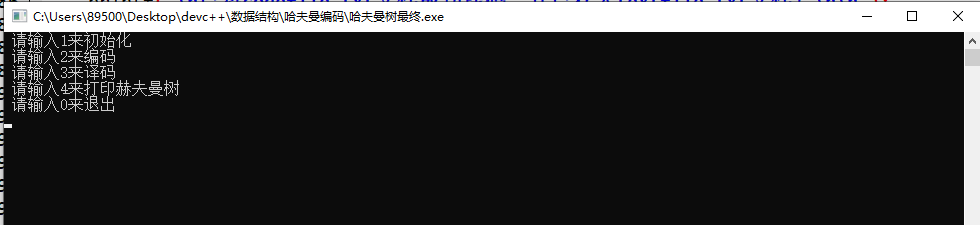
3、在构建哈夫曼树时，要创建2\*n-1个结点数量

4、在打开文件后别忘记关闭文件

5、译码时根据01字符串寻找相应叶子节点的递归算法最为简单方便

**五、用户手册**

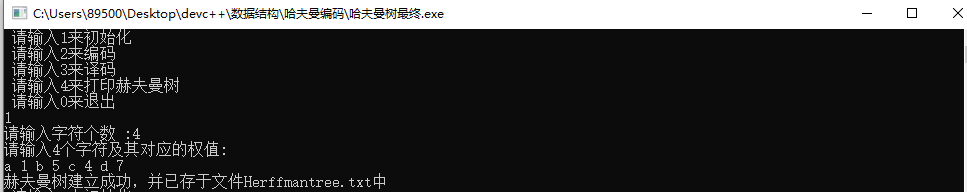
1. 本程序的执行文件为：哈夫曼编码）.exe
2. 进入演示程序后，将显示如下的界面



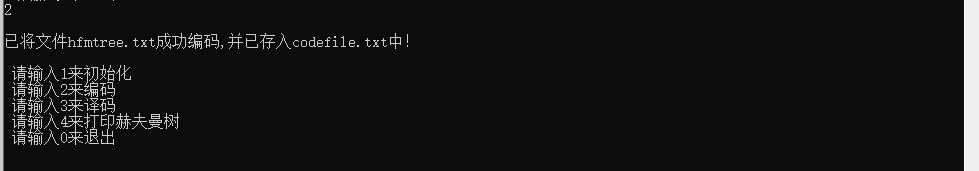
操作结果将写在测试结果中

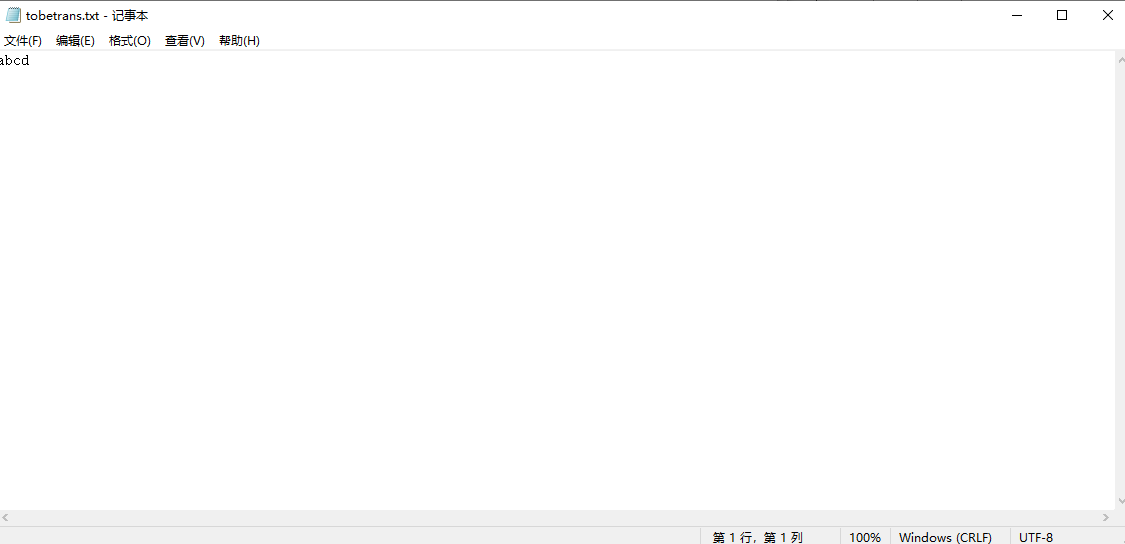
**六、测试结果**

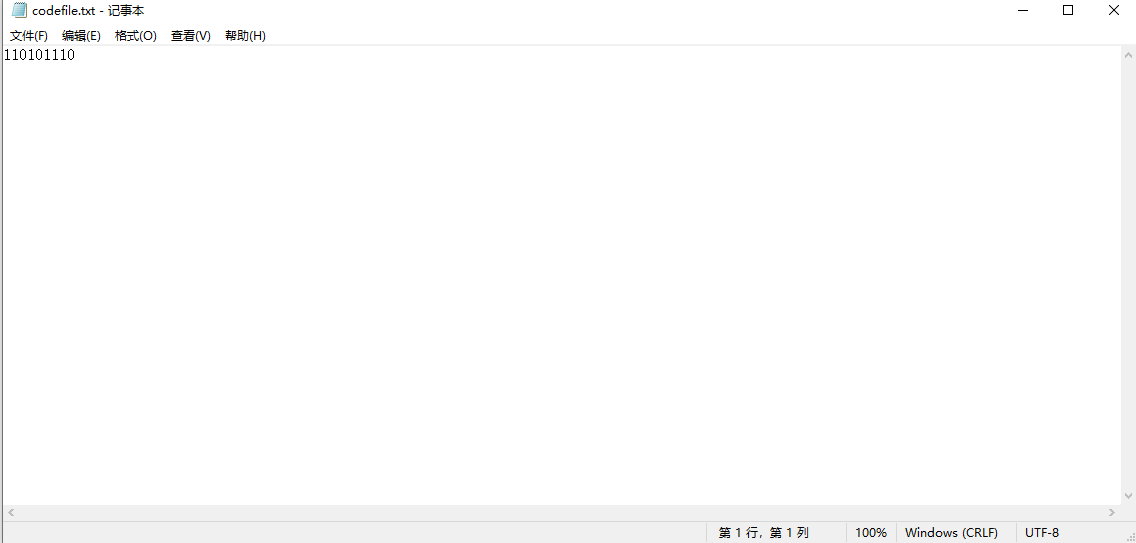
1、输入1初始化，先输入字符个数 4，再依次输入a 1 b 5 c 4 d 7,按下回车键



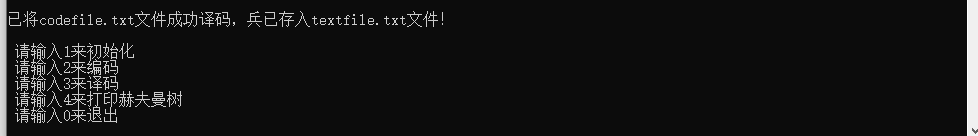
**2、输入2编码**

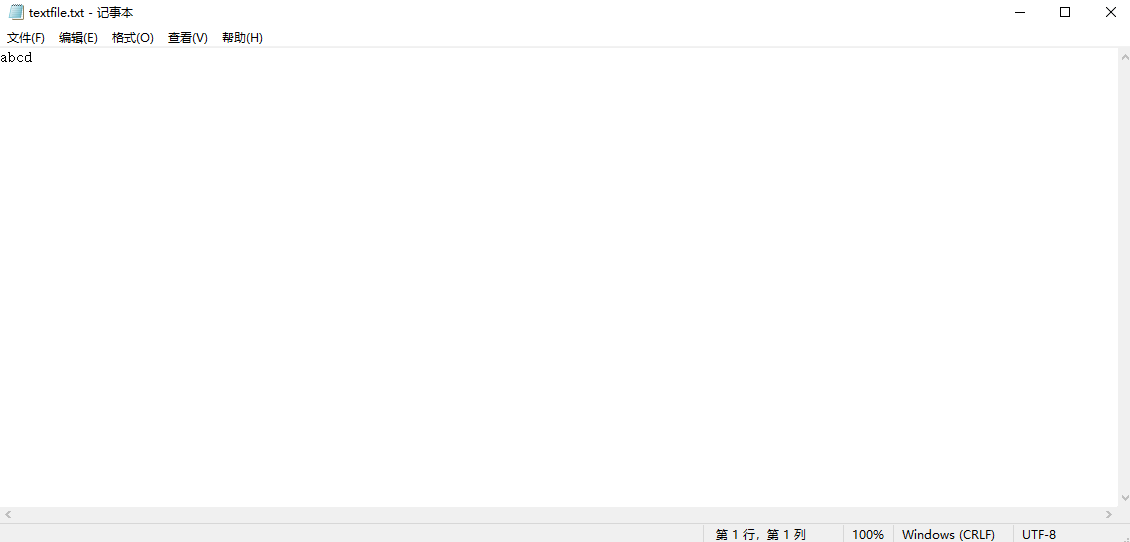






**3、输入3译码**

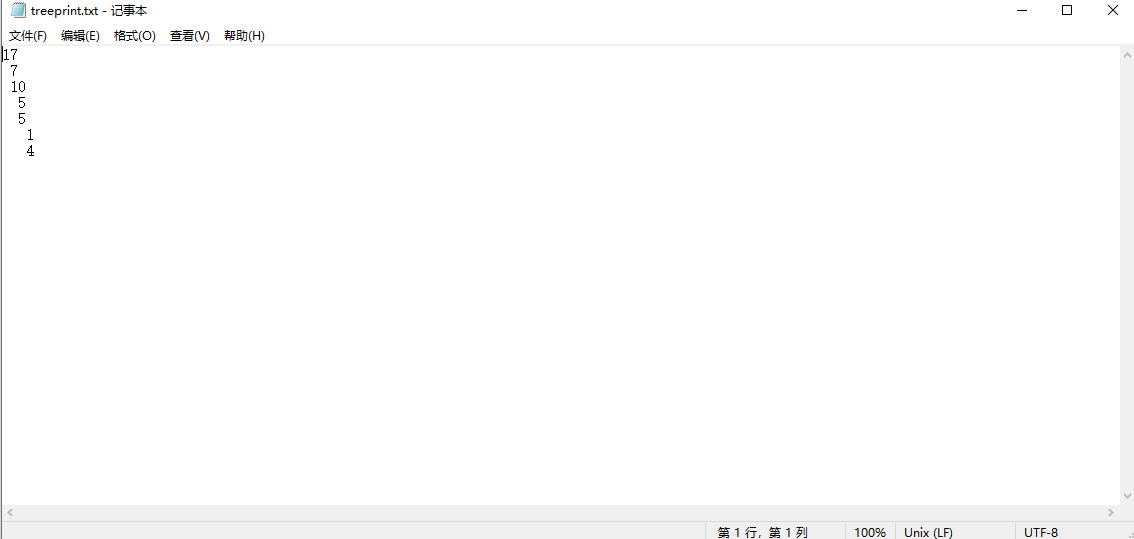




**4、、输入4打印**



17为最上面结点，最开始1和4两个结点，组成5，5和第三小的5组成成10，10和7组成17



**七、附录**

哈夫曼数.cpp

哈夫曼数.exe