**西 安 邮 电 大 学**

**（计算机学院）**

数据结构课程设计报告

# 题 目： 哈夫曼编/译码器

**专业名称： 数据科学与大数据技术**

**班 级： 1802**

**学生姓名： 肖若雯**

**学号（8位）： 04184064**

**指导教师： 王西龙**

**设计起止时间：** 2019年12月30日—2020年1月3日

## 一. 设计目的

利用Huffman编码进行通信可以大大提高信道利用率．缩短信息传输时间，降低传输成本，这要求在发送端通过一个编码系统对待传数据预先编码，在接收端将传来的数据进行译码（复原）。对于双工信道（即可以双向传输信息的信道），每端都需要一个完整的编／译码系统。此程序就是为这样的信息收发站写一个Huffman码的编／译码系统。

## 二. 设计内容

1. 建立哈夫曼树：读入文件(\*.souce),统计文件中字符出现的频度，并以这些字符频度作为权值，建立哈夫曼树。
2. 编码：利用已建好的哈夫曼树，获得各个字符的哈弗曼编码，并对正文进行编码，然后输出编码结果，并存入文件(\*.code)中。
3. 译码：利用已建好的哈夫曼树将文件(\*.code)中的代码进行译码，并输出译码结果，并存入文件(\*.decode)中。

## 三．概要设计

### 1．功能模块图；

### 各个模块详细的功能描述。

1. void menu() 获取菜单
2. 对源文件进行编码： 由客户端输入需要编码的文件名，然后遍历文件所有字符，获得所有字符数以及字符频度，将字符频度当作对应字符的权值，将所有字符数传入key返回。先建立哈夫曼森林，对每一种字符，分别建立一颗单节点Huffman树，对每一个Huffuman节点的成员进行初始化。然后依次选择根节点权值最小的树作为左右子树，权值之和为父节点的权值。此过程不断进行，每进行一次，森林里就少一颗树，当森林有且仅有一棵树时，这棵树就是Huffman树。在这个过程中，定义一个缓存数组，由后向前保存。最后一位为‘\0’,由叶子向根移动，左‘0’右‘1’，直至遍历完所有字符，得到所有字符的哈夫曼编码，存在每个节点的code成员里。由客户端输入想要保存的文件名，按原文本的字符顺序，将哈夫曼编码存入文件。
3. 查看编码表：输出字符，权值及其最后的哈夫曼编码。
4. 对编码文件进行译码： 由客户端获取需要译码的文件名，然后由哈夫曼树的根结点出发，根据文件内的0或1判断是进入左子树还是进入右子树，直至达到根节点，就算识别出了一个字符。直至文件结束。

## 详细设计

### 1．功能函数的调用关系图

### 2．各功能函数的数据流程图

### 重点设计及编码

* 1. 结构体的设计：
     1. typedef struct {

char ch; //指向字符

char \*code; //指向编码表

int weight; //权值

int parent,LChild,RChild; //父亲节点，左孩子，右孩子

}HTNode,\*HuffmanTree；

* + 1. struct {

char ch; //字符

int num; //出现次数

}binary[60];

* 1. void menu(); //显示菜单
  2. HuffmanTree CreateHuffmanTree(int n);

创建哈弗曼树。

* 1. void selectMin(HuffmanTree HT,int len,int\*s1,int\*s2);

选取权值最小的两个节点。

* 1. void ReadFile(int \*key);

读取文件，统计字符频度。

* 1. void CreateHuffmanCode(HuffmanTree HT,int n);

建立哈弗曼编码。

* 1. void CodeFile(HuffmanTree HT,int n);

将编码后的文件保存。

* 1. void TranslationFile(HuffmanTree HT,int n);

对文件进行译码。

* 1. void PrintForm(HuffmanTree HT,int n);

输出编码表。

1. 哈夫曼树的建立

void selectMin(HuffmanTree HT,int len,int \*s1,int \*s2){

int i ,min=32766; //初始化最小值

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

if(HT[i].parent==0){ //寻找最小权值

if(HT[i].weight < min){

min = HT[i].weight;

\*s1=i;

}

}

}

min=32766;

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

if(i==(\*s1)){

continue;

}

if(HT[i].parent==0){ //找次小权值

if(HT[i].weight < min){

min = HT[i].weight;

\*s2=i;

}

}

}

}

HuffmanTree CreateHuffmanTree(int num){

HuffmanTree HT;

int n = num;

int m,i;

int s1=0,s2=0;

if(n<=1){

return NULL;

}

m = 2\*n-1;

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

for(i=1;i<=n;i++){ //初始化

HT[i].parent=0;

HT[i].LChild=0;

HT[i].RChild=0;

HT[i].ch=binary[i-1].ch;

HT[i].weight=binary[i-1].num;

HT[i].code='\0';

}

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

HT[i].parent=0;

HT[i].LChild=0;

HT[i].RChild=0;

HT[i].ch='\0';

}

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

selectMin(HT,i-1,&s1,&s2); //选择权值最小的两个节点

HT[i].weight=HT[s1].weight+HT[s2].weight;

//新添加节点，权值为之和

HT[i].LChild=s1; //更改左孩子

HT[i].RChild=s2; //更改右孩子

HT[s1].parent=i; //更改父节点

HT[s2].parent=i; //更改父节点

}

printf("\n\n\t\t\t哈弗曼树的建立已完成\n");

return HT;

}

1. 根据哈夫曼树求得哈夫曼编码

void CreateHuffmanCode(HuffmanTree HT,int n){

int i,start,c,p;

char \*cd = (char\*)malloc(sizeof(char)\*codeMax); //建立缓冲数组

cd[codeMax-1]='\0'; //因为逆序，所以在最后一位先填入结束符\0

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

start=codeMax-1;

c=i;

p=HT[i].parent;

while(p!=0){

--start;

if(HT[p].LChild==c){

cd[start]='0';

//若当前节点为父节点的左孩子，则对应加一位0

}else{

cd[start]='1';

//若当前节点为父节点的右孩子，则对应加一位1

}

c=p;

p=HT[p].parent; //向上迭代

}

HT[i].code=(char \*)malloc(sizeof(char)\*(codeMax-start));

strcpy(HT[i].code,&cd[start]);

}

free(cd);

printf("\n\t\t\t哈弗曼编码已完成\n\n\t\t\t");

getch();

}

1. 译码

void TranslationFile(HuffmanTree HT,int n){

if(n==0){

printf("\n\t\t\t哈夫曼树未建立，请先执行步骤1,2\n\t\t\t");

getch();

return;

}

int i=0,f;

char fileName[20];

char code ,ch;

printf("\n\t\t\t请输入需要译码的文件名(格式：\*.txt)：");

scanf("%s",fileName);

FILE \*fp=fopen(fileName,"r");

FILE \*fp1=fopen("encode.txt","w");

printf("\n\t\t\t");

if(fp==NULL){

printf("文件打开失败");

exit(1);

}

while(!feof(fp)){

code = fgetc(fp); //从文件中一次读出一个字符

if(feof(fp))

break;

f=2\*n-1;

while(1){

if(code=='0'){

f=HT[f].LChild; //若为0，则进入左子树

}else{

f=HT[f].RChild; //若为1，则进入右子树

}

if(HT[f].LChild==0 && HT[f].RChild==0){

printf("%c",HT[f].ch);

//判断当前节点是否为叶子节点，若为叶子节点，则输出当前节点对应的ch

fprintf(fp1,"%c",HT[f].ch);

break;

}else{

code=fgetc(fp);

}

}

}

fflush(stdin);

printf("\n\n\t\t\t译码完成，如上。");

printf("\n\n\t\t\t文件已保存至encode.txt");

fclose(fp);

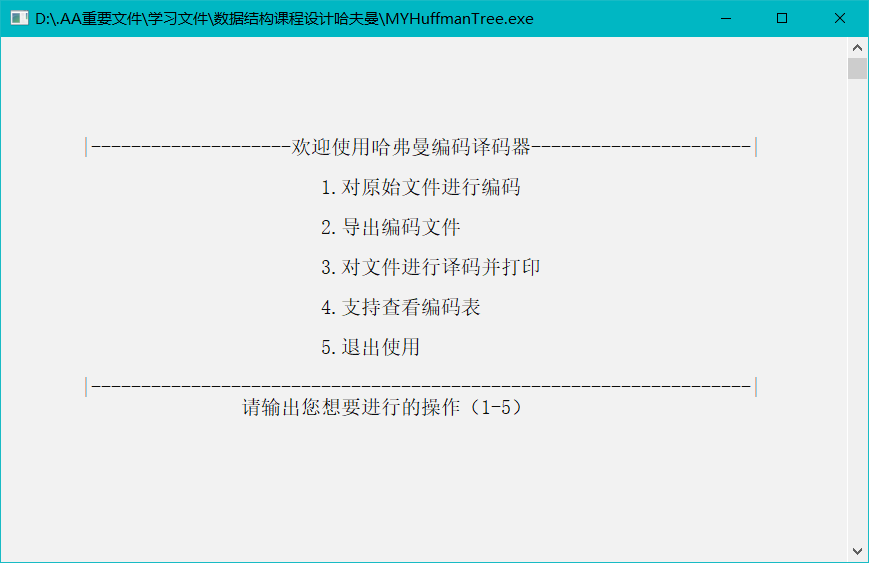
fclose(fp1);

getch();

}

## 测试数据及运行结果

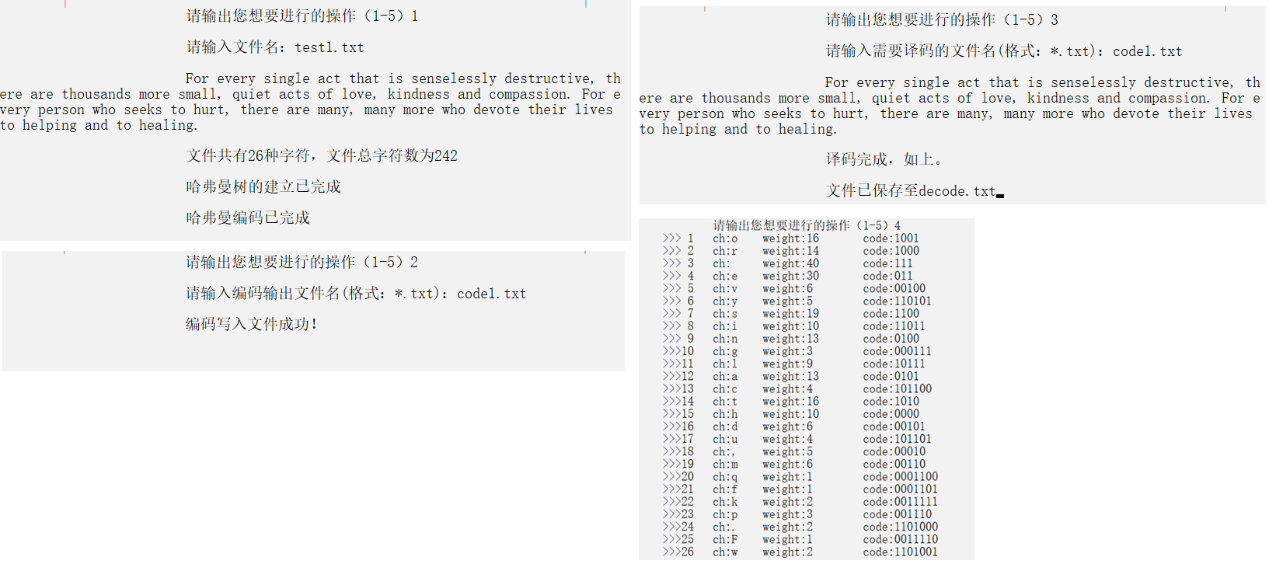
初始化界面

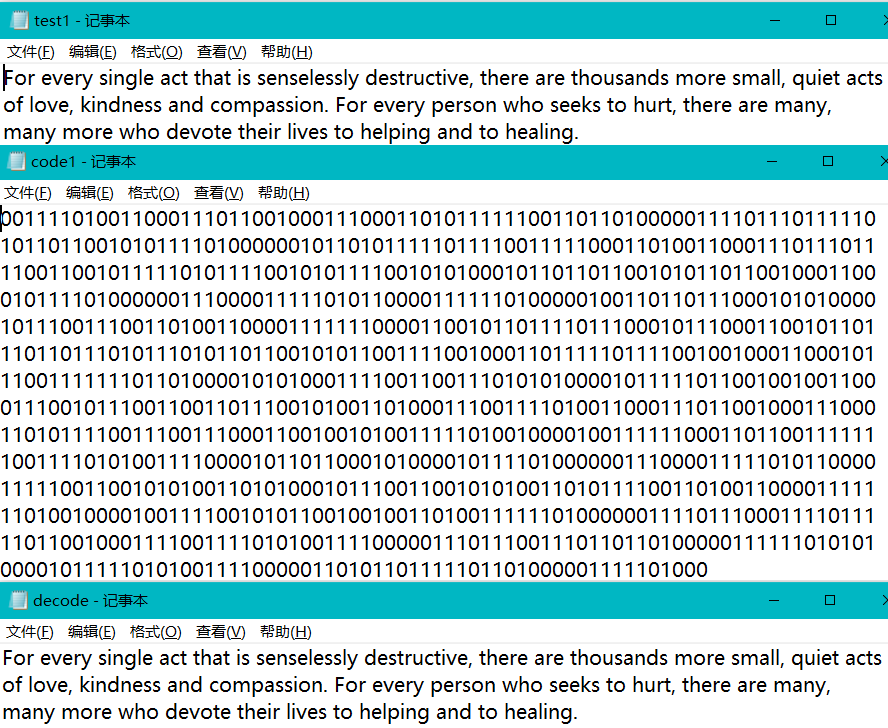


### 1．正常测试数据和运行结果

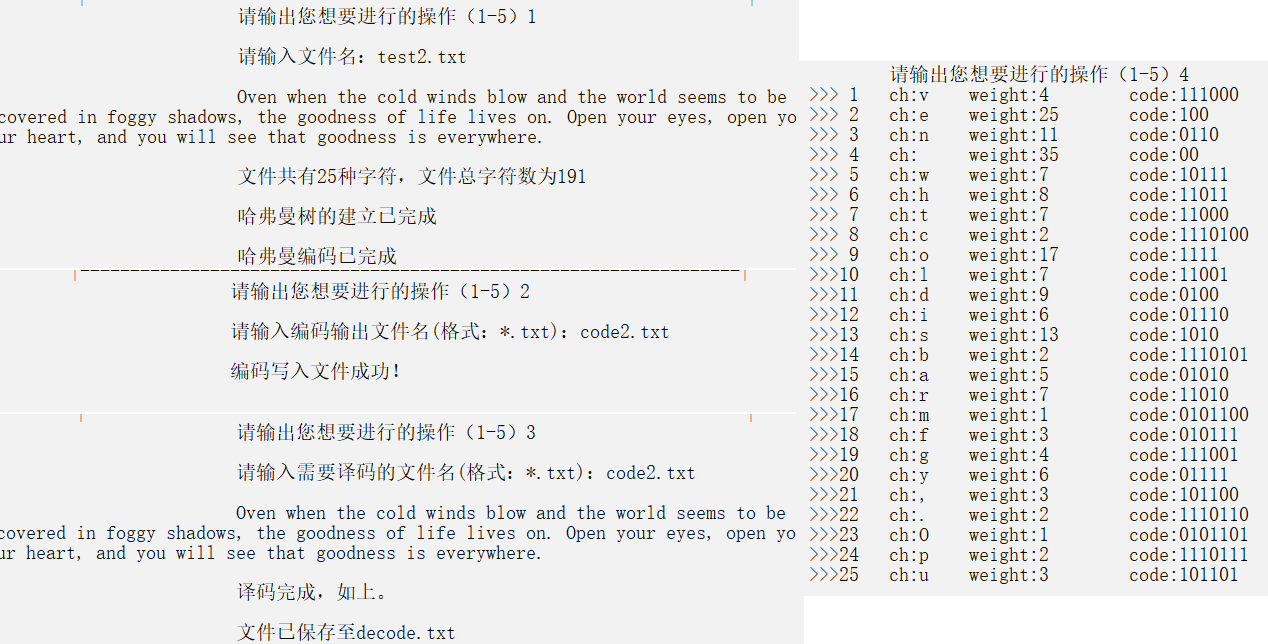
要求提供3组正常测试数据和运行结果

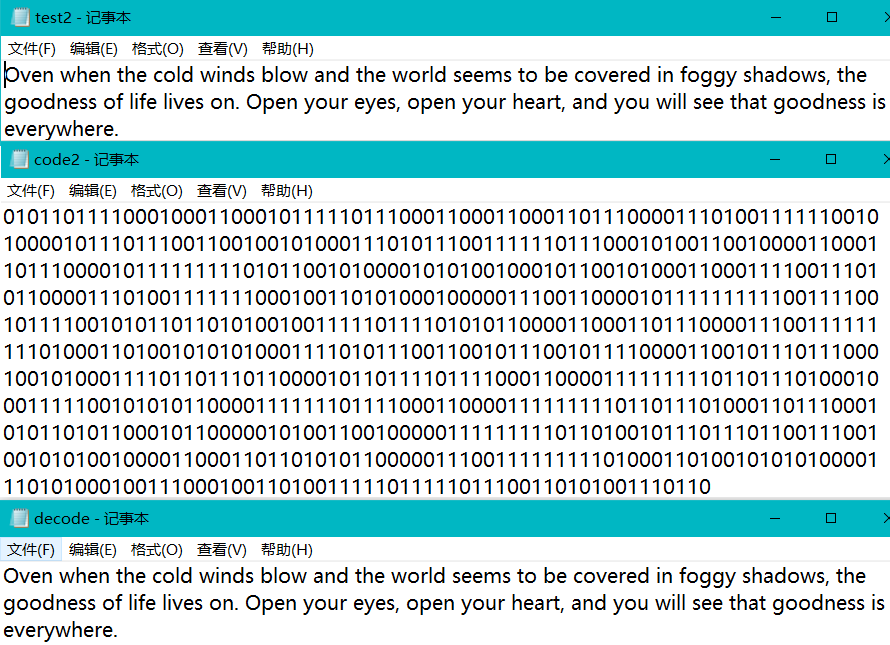
1）



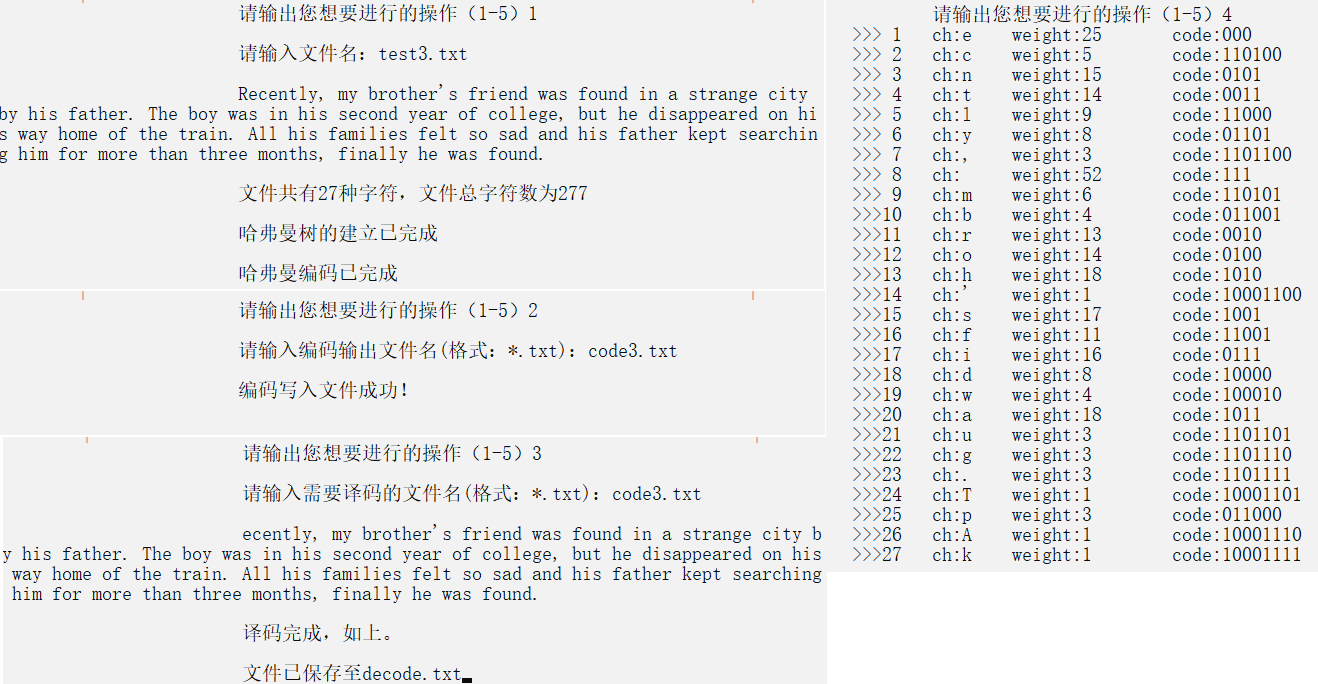


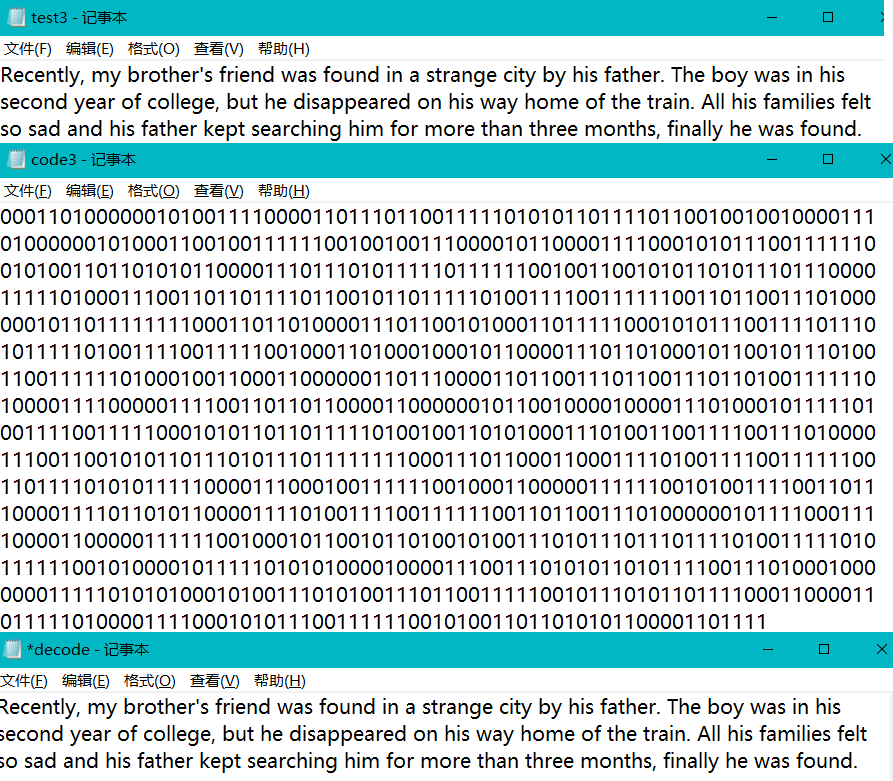
2）





3）

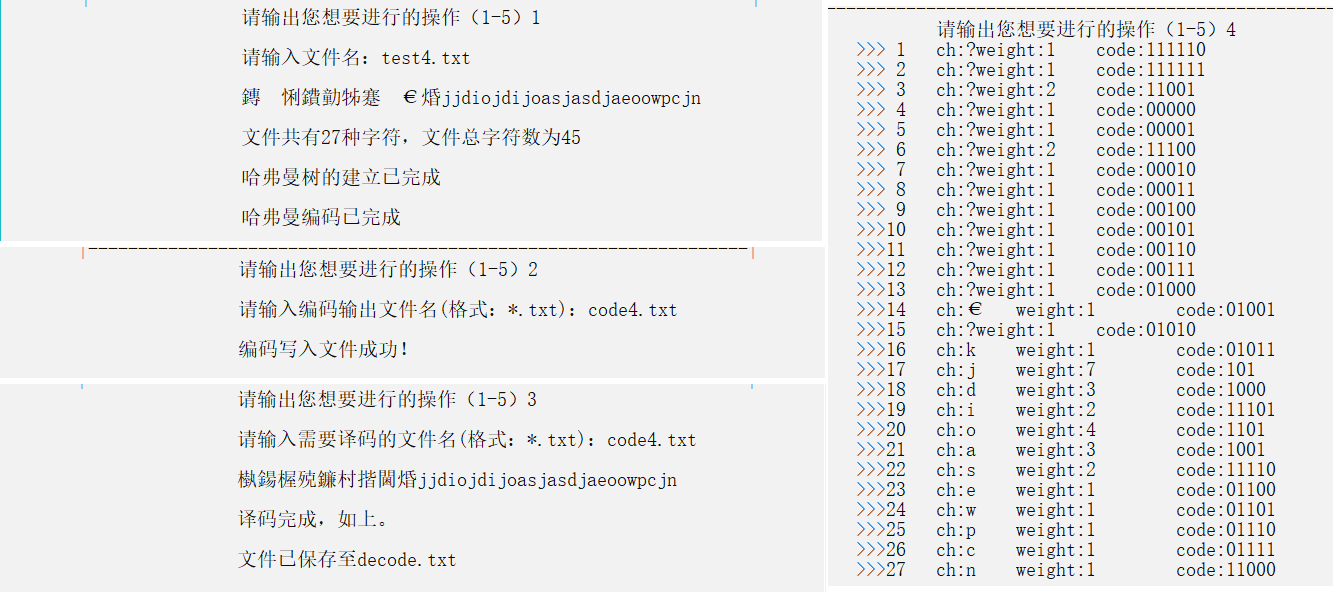


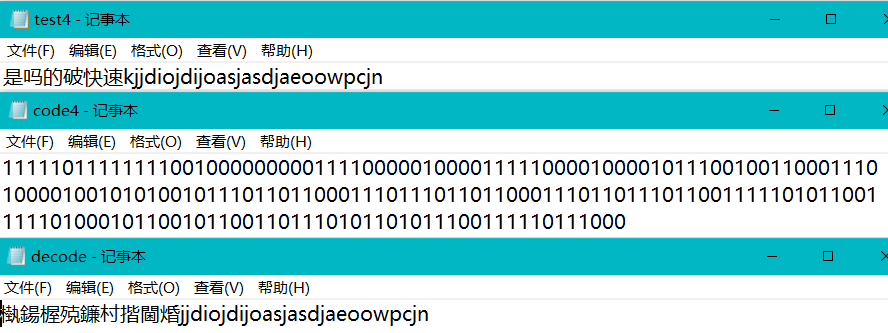


### 2．异常测试数据及运行结果

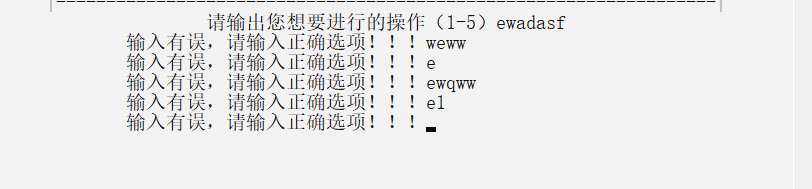
要求提供2组异常测试数据和运行结果

1）中文输入





2）非法输入



## 六．调试情况，设计技巧及体会

### 1．改进方案

对自己的设计进行评价，指出合理和不足之处，提出改进方案；

* + - 1. 对于中文字符的输入，可以考虑字符库扩大的方式来成功读入。
      2. 还可以加入文件压缩和解压缩操作。

### 2．体会

在数据结构与算法的一周课程设计中，我通过对哈夫曼编码译码算法的实现，深入理解了数据结构在代码编写和软件编写的过程中所起的巨大作用。也教会了我在做事之前要提前构建合适的数据结构。而在编写过程中，不可避免地遇到了一些bug，但在查找资料的过程中，这些问题被不断解决。

## 七．参考文献

[1]王曙燕. 数据结构与算法[M]. 2019年8月第一版. 北京市西城区德外大街4号:高等教育出版社, 2019.