**实验五**

题目：树及其应用——哈夫曼编码

班级：17级计科一班 姓名：元雨暄 学号：1725111037 完成日期：2018.11.28

**一、需求分析**

(1) **输入形式**：输入大写字母表示执行的操作名，I表示Initialization操作，输入元素的个数，输入值为正整数，输入每个元素，以空格隔开，在输入每个元素的权重，为正整数，以空格隔开；其他操作只需输入大写字母不需要输入别的内容即可完成。

**输入值的范围**：正整数，字符

(2)**输出的形式**：输出在txt类型文件中，打印哈夫曼树则输出在屏幕上。

(3)**功能**：用户界面设计为“菜单”方式：显示上述功能符号。请用户键入一个选择功能符。此功能执行完毕后再显示此菜单，直至某次用户选择了“Ｑ”为止。

（ 1 ） Ｉ：初始化（Initialization）。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（Encoding）。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（Decoding）。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（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。

（ 5 ） Ｔ：印哈夫曼树（Tree printing）。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。

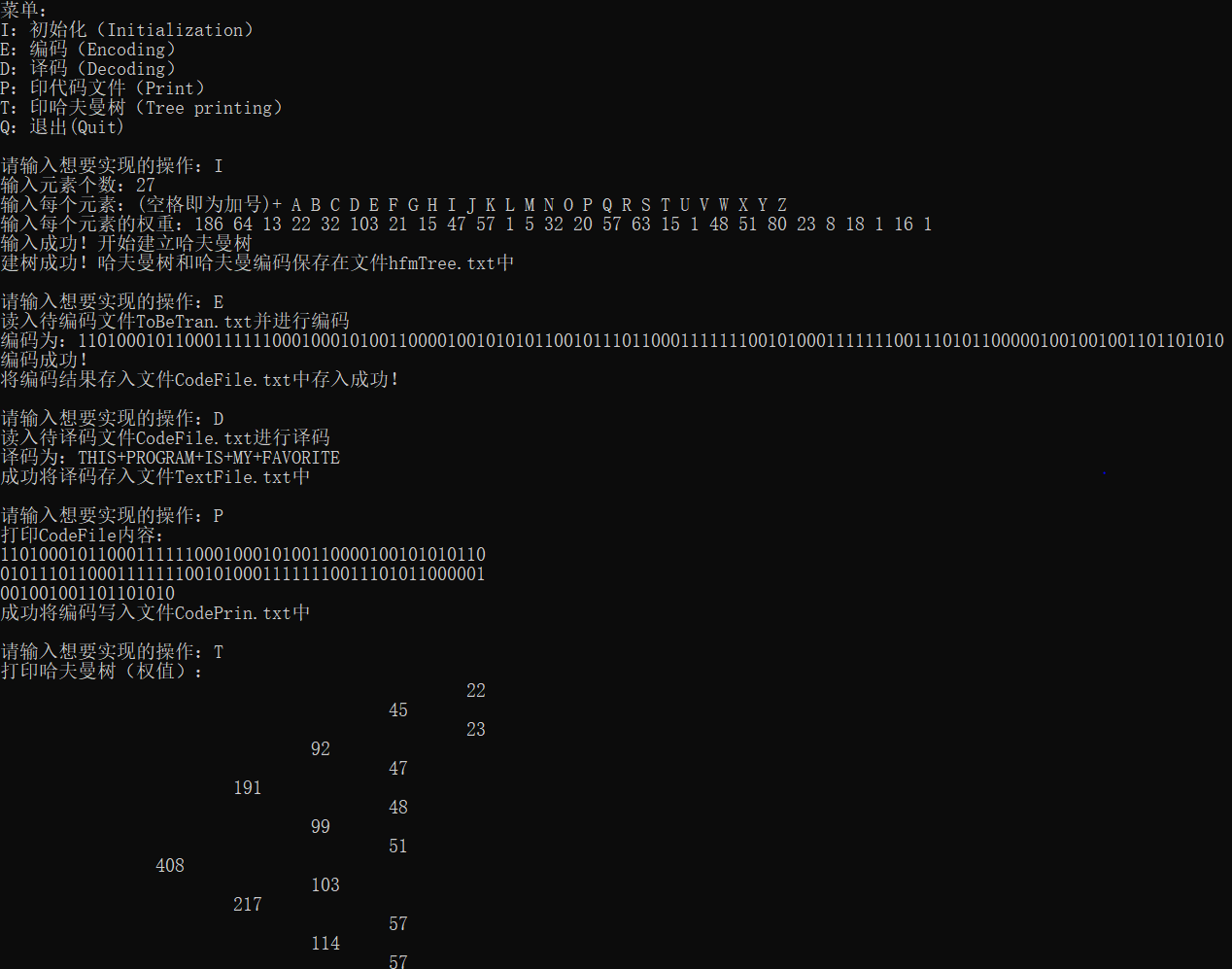
（ 6 ） Q：退出程序。

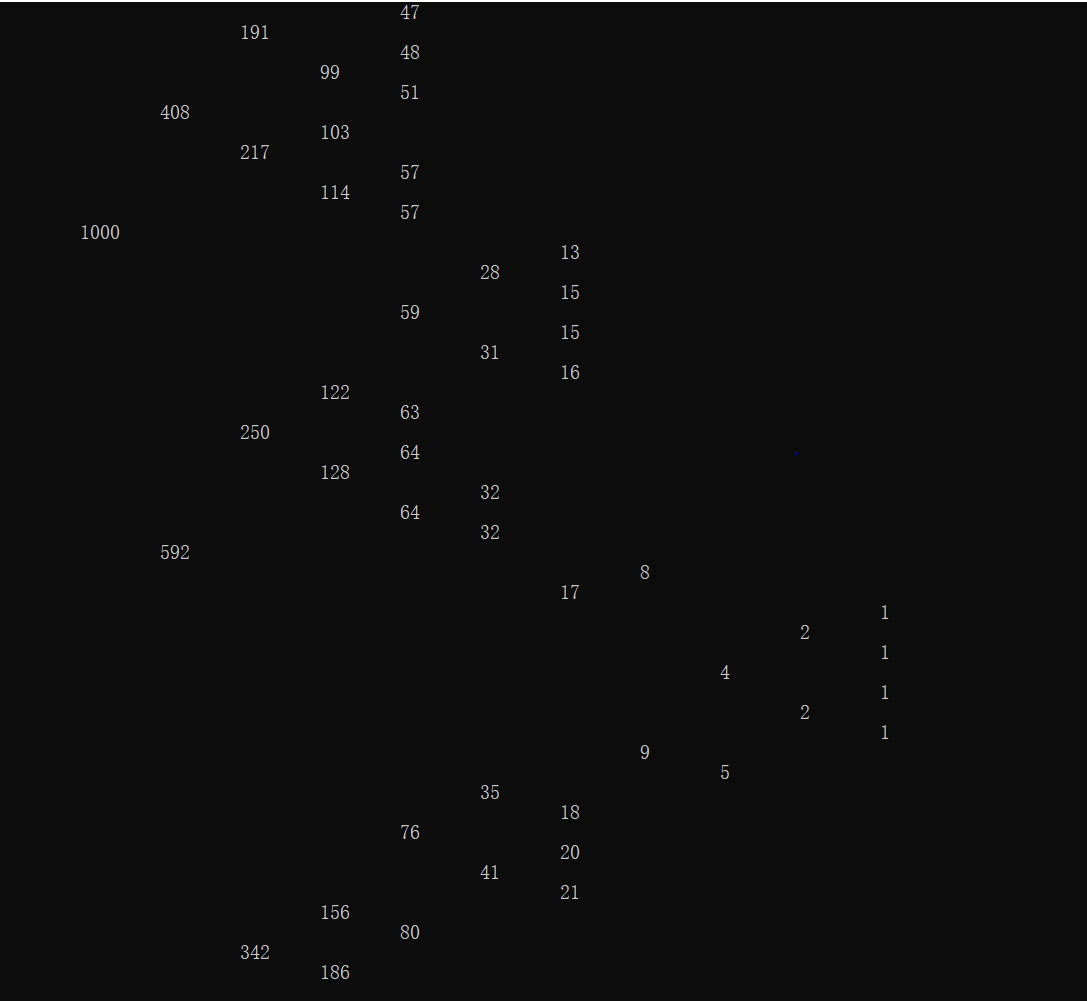
(4)**测试数据**:

总共27个字符，第一行为字符，第二行为权重：

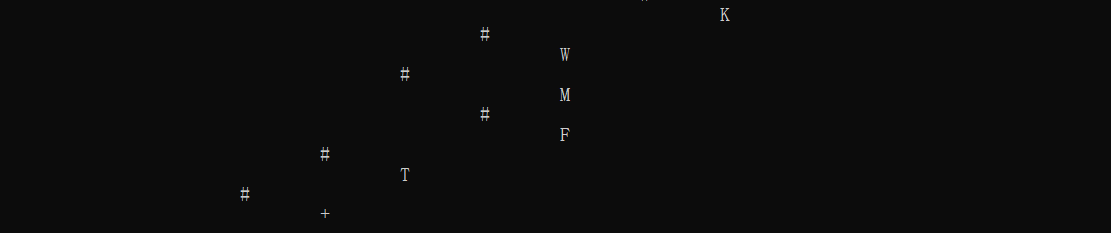
字符: + A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

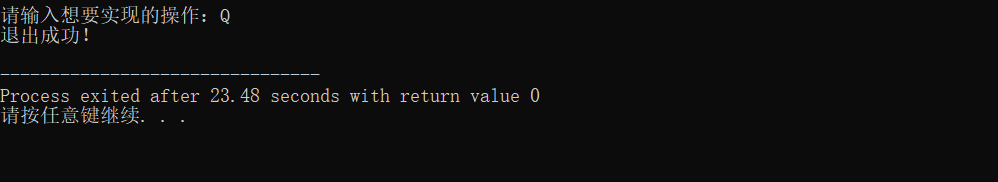
权重: 186 64 13 22 32 103 21 15 47 57 1 5 32 20 57 63 15 1 48 51 80 23 8 18 1 16 1

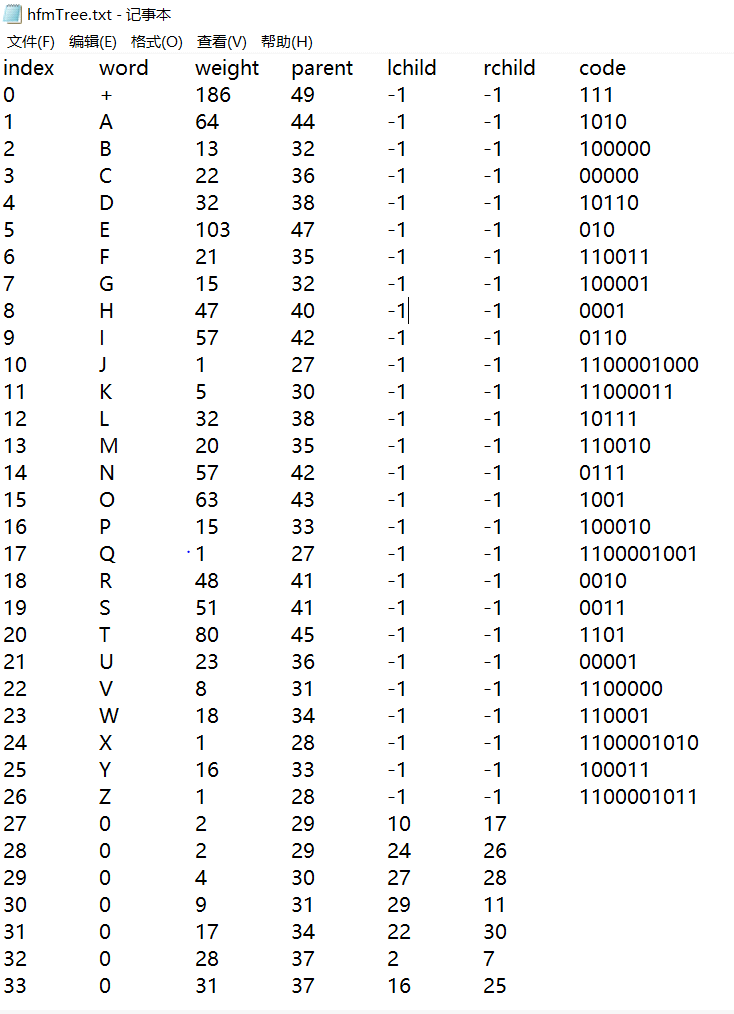


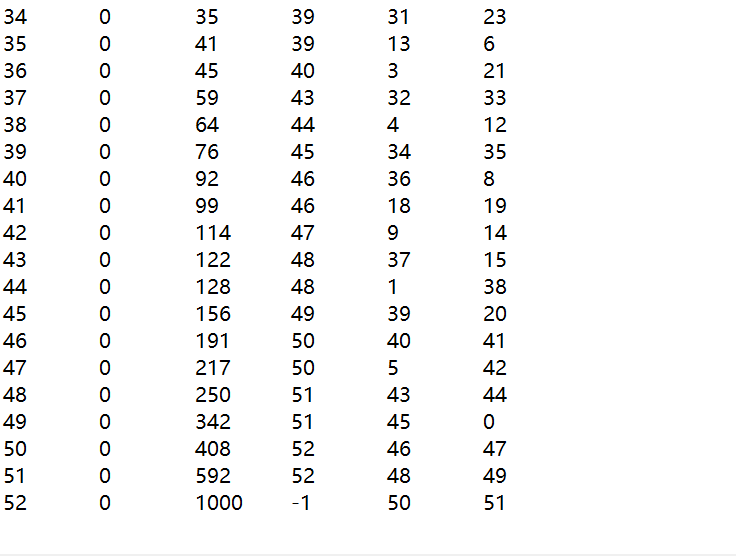


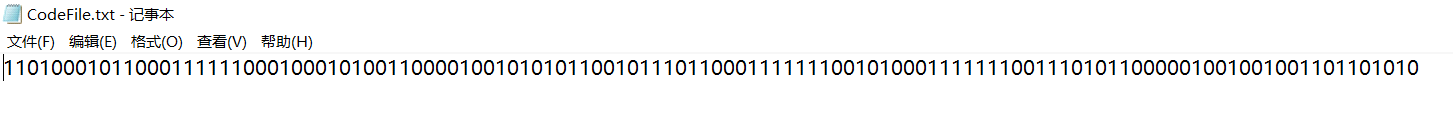


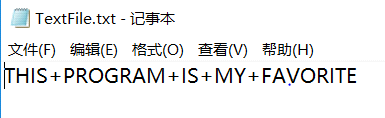


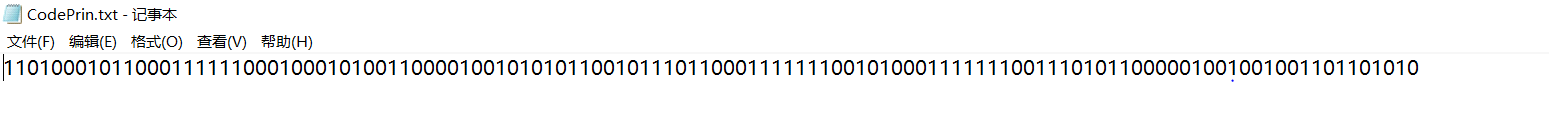


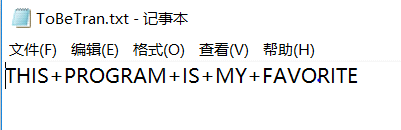












**二、概要设计**

(1)抽象数据定义

struct HTNode

{

int weight,parent,lchild,rchild;

};

内容分别是权重、双亲、左孩子的下标、右孩子的下标。

(2)主程序流程

int main()

{

void encoding(HTNode HT[],HuffmanCode &HC,int n);

void menu();

void encoding(HTNode HT[],HuffmanCode &HC,int n);

void Initialization();

menu();

return 0;

}

(3)各程序模块之间的调用

main（）函数调用menu（）菜单函数，menu函数根据用户输入的不同操作调用initialization（）函数、encoding()函数、decoding()函数、print()函数、treeprinting()函数，其中initialization（）函数调用了HuffmanTree（）函数和CreateHuffmanCode（）函数。

主函数模块

Menu函数

其他函数

**三、详细设计**

(1)数据类型的定义

struct HTNode

{

int weight,parent,lchild,rchild;

}; //哈夫曼树的一个结点

typedef char \*\*HuffmanCode; //哈夫曼树编码表

(2)操作的伪码算法

void Initialization(HTNode HT[],HuffmanCode &HC,char \*&ch,int &n)

{ //哈夫曼树和密码的初始化函数，菜单的I操作

void HuffmanTree(HTNode HT[],int w[],int n);//哈夫曼树的初始化函数

void CreateHuffmanCode(HTNode HT[],HuffmanCode &HC,int n);

int \*p;

int opt; //输入的操作序号

cout<<"1：建立新的哈夫曼树\n2：读取文件建立哈夫曼树\n输入操作：";

cin>>opt;

if(opt==1)

{

ofstream myout("hfmTree.txt");

cout<<"输入元素个数：";

cin>>n;

ch=new char[n];

p=new int[n];

cout<<"输入每个元素：(空格即为加号)";

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

cin>>ch[i];

cout<<"输入每个元素的权重：";

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

cin>>p[i];

cout<<"输入成功！开始建立哈夫曼树\n";

HuffmanTree(HT,p,n); //构造哈夫曼树

CreateHuffmanCode(HT,HC,n); //构造哈夫曼编码

myout<<left; //将哈夫曼树及编码读入文件

myout<<"index\tword\tweight\tparent\tlchild\trchild\tcode\n";

for(int i=0;i<2\*n-1;++i)

{

if(i<n)

myout<<i<<"\t"<<ch[i]<<"\t"<<HT[i].weight<<"\t"<<HT[i].parent<<"\t"<<HT[i].lchild<<"\t"<<HT[i].rchild<<"\t"<<HC[i]<<endl;

else

{

myout<<i<<"\t"<<"0\t"<<HT[i].weight<<"\t";

myout<<HT[i].parent<<"\t";

myout<<HT[i].lchild<<"\t";

myout<<HT[i].rchild<<endl;

}

}

cout<<"建树成功！哈夫曼树和哈夫曼编码保存在文件hfmTree.txt中\n";

myout.close();

}

if(opt==2) //操作2 读取文件进行初始化

{

fstream myin;

myin.open("hfm.txt");

n=27;

ch=new char[n];

p=new int[n];

int m1=0,m2=0;

while(!myin.eof())

{

if(m1<n)

{

myin>>ch[m1];

m1++;

}

else

{

myin>>p[m2];

m2++;

}

}

myin.close();

ofstream myout("hfmTree.txt");

HuffmanTree(HT,p,n); //构造哈夫曼树

CreateHuffmanCode(HT,HC,n); //构造哈夫曼编码

myout<<left;

myout<<"index\tword\tweight\tparent\tlchild\trchild\tcode\n";

for(int i=0;i<2\*n-1;++i)

{

if(i<n)

myout<<i<<"\t"<<ch[i]<<"\t"<<HT[i].weight<<"\t"<<HT[i].parent<<"\t"<<HT[i].lchild<<"\t"<<HT[i].rchild<<"\t"<<HC[i]<<endl;

else

{

myout<<i<<"\t"<<"0\t"<<HT[i].weight<<"\t";

myout<<HT[i].parent<<"\t";

myout<<HT[i].lchild<<"\t";

myout<<HT[i].rchild<<endl;

}

}

cout<<"建树成功！哈夫曼树和哈夫曼编码保存在文件hfmTree.txt中\n";

myout.close();

}

}

void CreateHuffmanCode(HTNode HT[],HuffmanCode &HC,int n) //哈夫曼编码实现

{

int start,p,c;

char \*cd = new char[n];

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

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

{

start=n-1;

c=i;

p=HT[i].parent;

while(p!=-1)

{

--start;

if(HT[p].lchild==c) cd[start]='0';

else cd[start]='1';

c=p;

p=HT[p].parent;

}//从叶子结点到根结点层层上去，最后将获得的一串编码复制给哈夫曼编码表对应位置

HC[i]=new char[n-start];

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

}

delete cd;

}

void Select(HTNode HT[],int sum,int &s1,int &s2)//从已有的元素中选取权重最小的两个元素

{

int i,j;

for(i=0;i<sum;i++)

{

if(HT[i].parent==-1)

{

s1=i; //初始化s1

break;

}

}

for(i=0;i<sum;i++)

{

if(HT[i].parent==-1 && HT[s1].weight>HT[i].weight) s1=i;

}

for(j=0;j<sum;j++)

{

if(HT[j].parent==-1 && j!=s1)

{

s2=j;

break;

}

}

for(j=0;j<sum;j++)

{

if(HT[j].parent==-1 && HT[s2].weight>HT[j].weight&& j!=s1) s2=j;

}

}

void HuffmanTree(HTNode HT[],int w[],int n)//初始化一个哈夫曼树

{

for(int i=0;i<2\*n-1;i++)

{

HT[i].weight=0;

HT[i].parent=-1;

HT[i].lchild=-1;

HT[i].rchild=-1;

}

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

HT[i].weight=w[i];

for(int j=n;j<2\*n-1;j++)

{

int s1,s2;

Select(HT,j,s1,s2);

HT[s1].parent=j;

HT[s2].parent=j;

HT[j].lchild=s1;

HT[j].rchild=s2;

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

}

}

void encoding(HTNode HT[],HuffmanCode &HC,char \*ch,int n)

{

char s;

string ss="";

char \*c=new char[9999];

int i=0;

ifstream ToBeTran;

ofstream CodeFile("CodeFile.txt");

ToBeTran.open("ToBeTran.txt");

cout<<"读入待编码文件ToBeTran.txt并进行编码\n";

while (!ToBeTran.eof())

{

ToBeTran>>s;

c[i]=s; //输出

i++;

}

i=0;

while(c[i+1]!='\0')

{

for(int j=0;j<n;j++)

{

if(c[i]==ch[j])

{

ss=ss+HC[j];

i++;

break;

}

}

}

cout<<"编码为："<<ss;

cout<<"\n编码成功！\n将编码结果存入文件CodeFile.txt中";

CodeFile<<ss;

CodeFile.close();

cout<<"存入成功！\n";

}

void decoding(HTNode HT[],char \*ch,char \*&b,int n) //b[] 用来记录译出的字符

{

cout<<"读入待译码文件CodeFile.txt进行译码\n";

char s;

int i=0;

string ss="";

ifstream CodeFile;

char \*d=new char[999];

CodeFile.open("CodeFile.txt");

while (!CodeFile.eof())

{

CodeFile>>s;

d[i]=s;

i++;

}

i=0;

int p=2\*n-2; //根结点下标

int k=0;

for(i=0;d[i+1]!='\0';i++)

{

if(d[i]=='0')p=HT[p].lchild;

if(d[i]=='1')p=HT[p].rchild;

if(HT[p].lchild==-1 && HT[p].rchild==-1)

{

b[k++]=ch[p];

p=2\*n-2;

}

}

b[k]='\0'; //将结果存入TextFile

k=0;

ofstream TextFile("TextFile.txt");

cout<<"译码为：";

while(b[k]!='\0')

{

cout<<b[k];

TextFile<<b[k];

k++;

}

TextFile.close();

cout<<"\n成功将译码存入文件TextFile.txt中\n";

}

void print() //印代码文件（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。

{

char \*q=new char[MAX\_LEN];

char in;

ifstream CodeFile;

ofstream CodePrin("CodePrin.txt");

CodeFile.open("CodeFile.txt");

int i=0,count=0;

while(!CodeFile.eof())

{

CodeFile>>in;

q[i]=in;

i++;

}

i=0;

cout<<"打印CodeFile内容：\n";

while(q[i+1]!='\0')

{

cout<<q[i];

CodePrin<<q[i];

count++;

i++;

if(count==50)

{

cout<<endl;

count=0;

}

}

cout<<"\n成功将编码写入文件CodePrin.txt中\n";

}

void printhelp(HTNode HT[],int index,string ss,char \*ch,ofstream &TreePrin,int n)

{

ss+=" ";

if(HT[index].lchild==-1 && HT[index].rchild==-1)

{

//cout<<ss<<HT[index].weight<<endl;

if(index>n)

{

cout<<ss<<"#"<<endl;

TreePrin<<ss<<"#"<<endl;

}

else

{

cout<<ss<<ch[index]<<endl;

TreePrin<<ss<<ch[index]<<endl;

}

return;

}

else

{

printhelp(HT,HT[index].lchild,ss,ch,TreePrin,n);

// cout<<ss<<HT[index].weight<<endl;

if(index>n)

{

cout<<ss<<"#"<<endl;

TreePrin<<ss<<"#"<<endl;

}

else

{

cout<<ss<<ch[index]<<endl;

TreePrin<<ss<<ch[index]<<endl;

}

printhelp(HT,HT[index].rchild,ss,ch,TreePrin,n);

}

}

void treeprinting(HTNode HT[],int n,char \*ch) //印哈夫曼树（Tree printing）。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

{

ofstream TreePrin("TreePrin.txt");//

void printhelp(HTNode HT[],int index,string ss,char \*ch,ofstream &TreePrin,int n);

string ss="";

int index=2\*n-2; //index从最后一个结点开始

cout<<"打印哈夫曼树（权值）：\n";

printhelp(HT,index,ss,ch,TreePrin,n);

}

void menu()//菜单函数，用在用户界面上，并且调用各种操作

{

char g; //g是每次输入的操作

int n; //n是元素总个数

char \*ch; //ch是读入的字符 例如+ a b c d e f g等

HuffmanCode HC=new char\*[MAX\_LEN];

HTNode \*HT=new HTNode[MAX\_LEN];

char \*b=new char[MAX\_LEN];

cout<<"菜单：\n";

cout<<"I：初始化（Initialization）\nE：编码（Encoding）\nD：译码（Decoding）\nP：印代码文件（Print）\nT：印哈夫曼树（Tree printing）\nQ：退出(Quit)\n";

while(true)

{

cout<<"\n请输入想要实现的操作：";

cin>>g;

switch(g)

{

case 'I':

Initialization(HT,HC,ch,n);

break;

case 'E':

encoding(HT,HC,ch,n);

break;

case 'D':

decoding(HT,ch,b,n);

break;

case 'P':

print();

break;

case 'T':

treeprinting(HT,n,ch);

break;

case 'Q':

cout<<"退出成功！\n";

exit(0);

}

}

return;

}

(3)函数的调用关系图

Menu函数()

Initialization() encoding() decoding() print() treeprinting()

HuffmanTree()CreateHuffmanCode()

Select()

**四、调试分析**

(1)问题：文件的读取和写入不熟悉，

解决:在查阅相关书籍之后解决

(2)算法的时空分析：

时间复杂度：设n为元素个数

void Initialization() O(n\*n)

void HuffmanTree() O(n)

void CreateHuffmanCode() O(n)

void Select() O(n\*n)

void encoding() O(n)

void decoding() O(n)

void print() O(n)

void printhelp() O(n)

void treeprinting() O(n)

void menu() O(1)

空间复杂度：

void Initialization() O(n)

void HuffmanTree() O(n)

void CreateHuffmanCode() O(n)

void Select() O(1)

void encoding() O(1)

void decoding() O(1)

void print() O(1)

void printhelp() O(1)

void treeprinting() O(1)

void menu() O(1)

**五、用户使用说明**

当弹出界面时，用户可以根据菜单的提示，输入相应操作的大写字母，根据每次操作弹出的提示完成操作，在输入多个值的时候每两个值需要用空格隔开。

**六、测试结果**

1.自己新建哈夫曼树

2.读取文件建立哈夫曼树

