**《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/译码器 |
| 姓 名： | 杨新莹 |
| 院 系： | 计算机学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 22052316 |
| 学 号： | 22050605 |
| 指导教师： | 张灵均 |

2023年12月8日

##### 需求分析

功能需求：

设计一个哈夫曼的编/译码系统，实现下述功能：

1. 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. 编码，利用已经建立好的哈夫曼树（如不在内存里，则从文件hfmTree中的正文进行编码，然后将结果存入文件CodeFile中；
3. 译码，利用已经建好的哈夫曼将文件CodeFile中的代码进行译码，结果存入文件TextFile中；
4. 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. 印哈夫曼树，将已在内存中的哈夫曼树以值观的方式（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TreePrin中
6. 实现各个转换操作的源/目文件，均由用户在选择此操作时指定

界面需求：

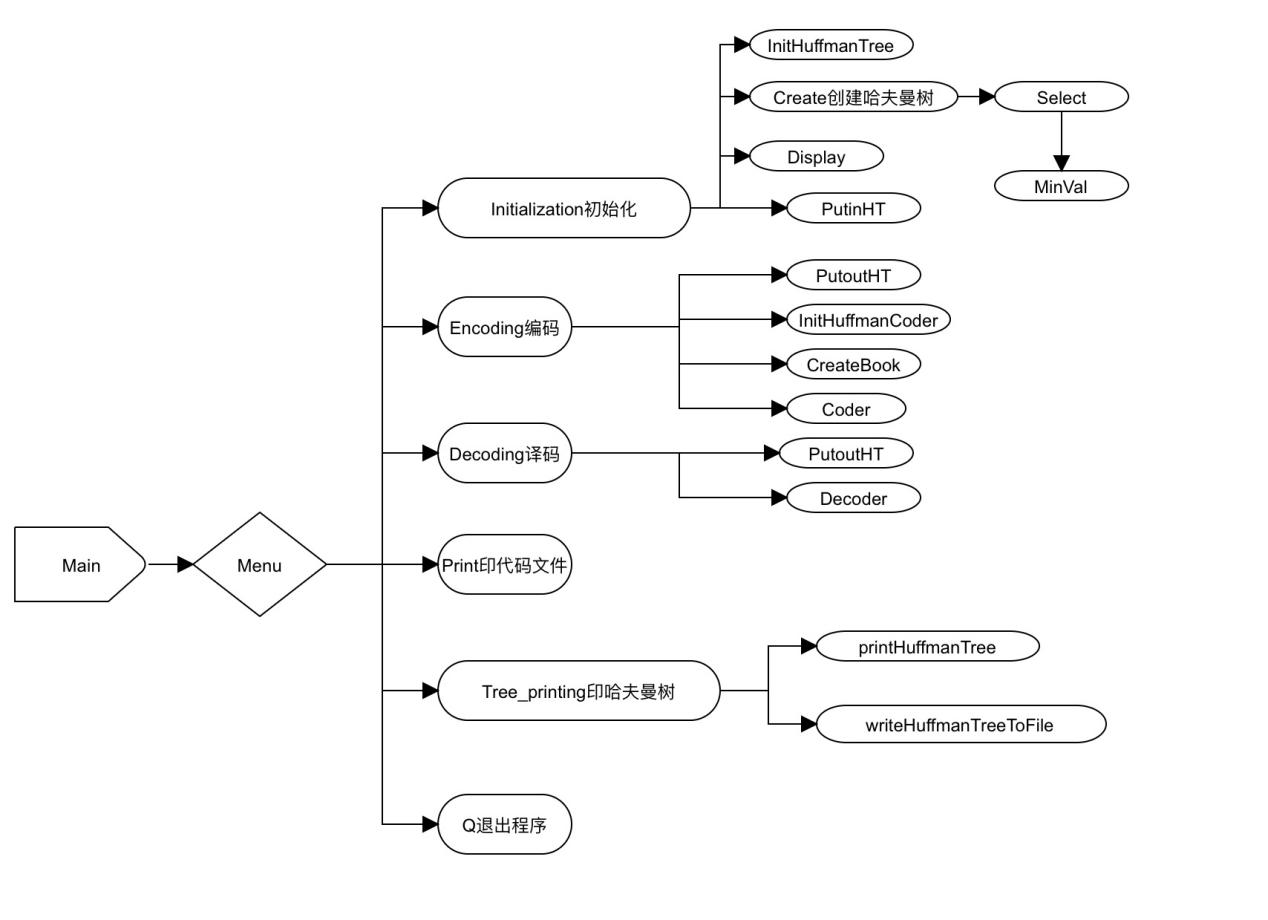
打印出菜单界面供用户选择，打印出各个字符的编码、需要编码的字符内容、编码结果、需要译码的内容、译码的结果、打印代码文件、以凹入表的形式打印哈夫曼树

接口需求：

初始化哈夫曼、编码、译码、打印代码文件、打印哈夫曼树

##### 概要设计

模块层次结构设计



接口设计

void InitHuffmanTree(HuffmanTree& HT, int n);//初始化哈夫曼树

void DEstroyHuffmanTree(HuffmanTree& HT);//销毁哈夫曼树

int MinVal(HuffmanTree& HT, int i);//在前i个结点中选择parant为-1且weight最小的结点并获得其序号

void Select(HuffmanTree& HT, int i, int& s1, int& s2);//s1为最小的两个值中序号小的那个

void Create(HuffmanTree& HT, int n, char ch[], int weight[]);//建立哈夫曼树

void Display(HuffmanTree HT);//显示哈夫曼树

void PutinHT(HuffmanTree& HT);//哈夫曼树保存到文件夹中

int Getn(const string& filename);//获得文件夹中的叶子结点的个数

void PutoutHT(HuffmanTree& HT, const string& filename, int n);//从文件中读入哈夫曼树

void InitHuffmanCoder(HuffmanCoder& HC, int n);//初始化编码表

void DestroyHuffmanCoder(HuffmanCoder& HC);//销毁编码表

void CreateBook(HuffmanCoder& HC, HuffmanTree& HT);//建立编码表

void Coder(HuffmanCoder& HC, char ch[], string filename);//编码

void Decoder(HuffmanTree& HT, char ch[], string filename);//译码

void GotoXY(int x, int y);//光标位置

char menu();//菜单函数，返回操作

void Initialization(HuffmanTree& HT, int& n);//初始化

void Encoding(HuffmanTree& HT, int n);//编码

void Decoding(HuffmanTree& HT, int n);//译码

void Print();//打印代码文件

void printHuffmanTree(const HuffmanTree& huffmanTree, int root, int level = 0);//打印哈夫曼树

void writeHuffmanTreeToFile(const HuffmanTree& huffmanTree, int root, ofstream& outFile, int level = 0);//将哈夫曼树存入文件夹中

void Tree\_printing(HuffmanTree& HT, int n);//印哈夫曼树

界面设计

·菜单页面：

system("cls");

GotoXY(38, 8);

printf("欢迎使用哈夫曼编/译码器");

GotoXY(43, 10);

printf("I. 初始化");

GotoXY(43, 12);

printf("E. 编码");

GotoXY(43, 14);

printf("D. 译码");

GotoXY(43, 16);

printf("P. 印代码文件");

GotoXY(43, 18);

printf("T. 印哈夫曼树");

GotoXY(43, 20);

printf("Q. 退出");

GotoXY(38, 22);

printf("请输入您需要进行的操作[ ]\b\b");

数据结构设计

//哈夫曼树结点

struct HTnode {

char ch;

int weight, parent, lchild, rchild;

};

//哈夫曼树

struct HuffmanTree {

HTnode\* ht;

int htsize;

bool flag = false;

};

//字符编码结构

struct HCode {

char ch;

char\* pstring;

};

//编码表结构

struct HuffmanCoder {

HCode\* hc;

int hcsize;

};

##### 详细设计

“Huffman.h”

#ifndef \_HUFFMAN\_H\_

#define \_HUFFMAN\_H\_

#include<fstream>

#include <iomanip>

#include<iostream>

#include <ios>

using namespace std;

#define max 10000

struct HTnode {

char ch;

int weight, parent, lchild, rchild;

};

struct HuffmanTree {

HTnode\* ht;

int htsize;

bool flag = false;

};

void InitHuffmanTree(HuffmanTree& HT, int n)

{

HT.ht = new HTnode[2 \* n - 1];

HT.htsize = 2 \* n - 1;

HT.flag = true;

}

void DEstroyHuffmanTree(HuffmanTree& HT)

{

delete[]HT.ht;

HT.htsize = 0;

}

int MinVal(HuffmanTree& HT, int i)

{

int j, k=max, min = max;

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

if (HT.ht[j].parent == -1 && HT.ht[j].weight < min)

{

min = HT.ht[j].weight;

k = j;

}

HT.ht[k].parent = max;

return k;

}

void Select(HuffmanTree& HT, int i, int& s1, int& s2)

{

int s;

s1 = MinVal(HT, i);

s2 = MinVal(HT, i);

if (s1 > s2)

{

s = s1;

s1 = s2;

s2 = s;

}

}

//创建哈夫曼树

void Create(HuffmanTree& HT, int n, char ch[], int weight[])

{

int i, s1, s2;

if (n > 1)

{

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

{

HT.ht[i].ch = ch[i];

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

HT.ht[i].parent = -1;

HT.ht[i].lchild = -1;

HT.ht[i].rchild = -1;

}

for (; i < HT.htsize; ++i)

{

Select(HT, i, s1, s2);

HT.ht[s1].parent = HT.ht[s2].parent = i;

HT.ht[i].lchild = s1;

HT.ht[i].rchild = s2;

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

HT.ht[i].parent = -1;

HT.ht[i].ch = ' ';

}

}

cout <<endl<< "哈夫曼树建毕！" << endl<<endl;

}

//显示哈夫曼树

void Display(HuffmanTree HT)

{

int i;

cout << "所建哈夫曼树的静态链表表示如下：" << endl << endl;

cout << "下标位置" << " 字符 " << " 权值 " << " 左孩子 " << " 右孩子 " << " 双亲 " << endl;

for (i = 0; i < HT.htsize; i++)

{

cout << setw(6) << i << setw(6) << HT.ht[i].ch << setw(8) << HT.ht[i].weight << setw(9) << HT.ht[i].lchild << setw(8) << HT.ht[i].rchild << setw(6) << HT.ht[i].parent << endl;

}

cout << endl;

system("pause");

}

//哈夫曼树保存到文件夹中

void PutinHT(HuffmanTree& HT)

{

ofstream outfile("hfmTree.txt", ios::in);

for (int i = 0; i < HT.htsize; i++)

{

outfile<< setw(6) << i << setw(6) << HT.ht[i].ch << setw(8) << HT.ht[i].weight << setw(9) << HT.ht[i].lchild << setw(8) << HT.ht[i].rchild << setw(6) << HT.ht[i].parent << endl;

}

outfile.close();

}

int Getn(const string& filename)

{

ifstream infile(filename, ios::in);

if (!infile)

{

cerr << "Error opening file: " << filename << endl;

return -1;

}

int a, b, c, d, e;

char ch;

int n = 0;

while (infile >> a >> ch >> b >> c >> d >> e)

{

if (c==-1) n++;

}

return n;

}

void PutoutHT(HuffmanTree& HT, const string& filename,int n)

{

ifstream infile(filename, ios::in);

if (!infile)

{

cerr << "Error opening file: " << filename << endl;

return;

}

// 根据确定的 n 值初始化 HuffmanTree

InitHuffmanTree(HT, n);

int i = 0;

int x;

// 遍历文件以读取数据

for (int i = 0; i < HT.htsize ; i++)

{

//fstream obj(filename, ios::in | ios::out);

//obj >> noskipws;

if (i < n&&i!=0) {

infile >> setw(6) >> x >> setw(6) >> HT.ht[i].ch >> setw(8) >> HT.ht[i].weight >> setw(9) >> HT.ht[i].lchild >> setw(8) >> HT.ht[i].rchild >> setw(6) >> HT.ht[i].parent;

}

else

infile >> setw(6) >> x >> HT.ht[i].weight >> setw(9) >> HT.ht[i].lchild >> setw(8) >> HT.ht[i].rchild >> setw(6) >> HT.ht[i].parent;

}

HT.ht[0].ch = ' ';

infile.close();

}

struct HCode {

char ch;

char\* pstring;

};

struct HuffmanCoder {

HCode\* hc;

int hcsize;

};

void InitHuffmanCoder(HuffmanCoder& HC, int n)

{

HC.hc = new HCode[n];

HC.hcsize = n;

}

void DestroyHuffmanCoder(HuffmanCoder& HC)

{

for (int i = 0; i < HC.hcsize; i++)

delete[]HC.hc[i].pstring;

delete[]HC.hc;

}

void CreateBook(HuffmanCoder& HC, HuffmanTree& HT)

{

int i, j, c, f, start;

char\* cd = new char[HC.hcsize];

cd[HC.hcsize - 1] = '\0';

cout<<"以下是各字符的哈夫曼编码：" << endl << endl;

for (i = 0; i < HC.hcsize; i++)

{

start = HC.hcsize - 1;

HC.hc[i].ch = HT.ht[i].ch;

for (c = i, f = HT.ht[i].parent; f != -1; c = f, f = HT.ht[f].parent)

if (f >= 0 && f < HT.htsize) {

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

cd[--start] = '0';

}

else {

cd[--start] = '1';

}

}

else {

// 处理异常情况，例如输出错误信息或者中断循环

cerr << "Error: Invalid index in Huffman tree." << endl;

break;

}

HC.hc[i].pstring = new char[HC.hcsize - start];

cout << "第" << setw(2)<<i + 1 << "个字符" << HT.ht[i].ch << "的编码是：";

for (j = start; j < HC.hcsize; j++)

{

cout << cd[j];

HC.hc[i].pstring[j - start] = cd[j];

}

cout << endl;

}

cout << endl;

delete[]cd;

}

void Coder(HuffmanCoder& HC, char ch[],string filename)

{

ofstream outfile(filename, ios::out);

for(int i=0;i<strlen(ch);i++)

for(int j=0;j<HC.hcsize;j++)

if (ch[i] == HC.hc[j].ch)

{

for (int k = 0; k<strlen(HC.hc[j].pstring); k++)

{

cout<<HC.hc[j].pstring[k];

outfile.put(HC.hc[j].pstring[k]);

}

}

outfile.put('\0');

cout << endl;

outfile.close();

}

void Decoder(HuffmanTree& HT,char ch[],string filename)

{

int j(0), i(0), p, pre, root = HT.htsize - 1;

ofstream outfile(filename, ios::out);

cout << "需译码的二进制电文是：" << endl;

j = 0;

while (ch[j])

{

cout << ch[j];

j++;

}

cout << endl;

cout << "译码结果：" << endl;

pre = -1;

p = root;

while (i < strlen(ch))

{

while (p != -1)

{

if (ch[i] == '0')

{

pre = p;

p = HT.ht[p].lchild;

}

else

{

pre = p;

p = HT.ht[p].rchild;

}

i++;

}

cout << HT.ht[pre].ch;

outfile.put(HT.ht[pre].ch);

i--;

pre = -1;

p = root;

}

cout << endl;

system("pause");

}

#endif

“main.cpp”

#include<iostream>

#include<Windows.h>

#include<string>

#include<cstring>

#include <cctype>

#include <limits>

#include <sstream>

#include"Huffman.h"

using namespace std;

char ch[256];

int weight[256];

bool initflag = false;

void GotoXY(int x, int y) {

HANDLE hout;

COORD cor;

hout = GetStdHandle(STD\_OUTPUT\_HANDLE);

cor.X = x;

cor.Y = y;

SetConsoleCursorPosition(hout, cor);

}

char menu()

{

system("cls");

GotoXY(38, 8);

printf("欢迎使用哈夫曼编/译码器");

GotoXY(43, 10);

printf("I. 初始化");

GotoXY(43, 12);

printf("E. 编码");

GotoXY(43, 14);

printf("D. 译码");

GotoXY(43, 16);

printf("P. 印代码文件");

GotoXY(43, 18);

printf("T. 印哈夫曼树");

GotoXY(43, 20);

printf("Q. 退出");

GotoXY(38, 22);

printf("请输入您需要进行的操作[ ]\b\b");

char ch;

char result='q';

ch = getchar();

switch (ch) {

case 'i': result = 'i'; break;

case 'I': result = 'i'; break;

case 'e': result = 'e'; break;

case 'E': result = 'e'; break;

case 'd': result = 'd'; break;

case 'D': result = 'd'; break;

case 'p': result = 'p'; break;

case 'P': result = 'p'; break;

case 't': result = 't'; break;

case 'T': result = 't'; break;

case 'q': result = 'q'; break;

case 'Q': result = 'q'; break;

default:

result = '@';

GotoXY(38, 24);

cout << "无效的选择，请重新输入。\n";

GotoXY(38, 26);

system("pause");

}

getchar();

system("cls");

return result;

}

void Initialization(HuffmanTree &HT,int &n)

{

cout << "请输入树叶结点的个数：" << endl;

cin >> n;

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

{

cout << "请输入第" << i + 1 << "个字符及权值:";

if (i == 0) cin >> weight[i];

else

cin >>ch[i]>> weight[i];

}

ch[0] = ' ';

InitHuffmanTree(HT, n);

Create(HT, n, ch, weight);

Display(HT);

PutinHT(HT);

initflag = true;

}

void Encoding(HuffmanTree &HT,int n)

{

if (!HT.flag)

PutoutHT(HT, "hfmTree.txt", n);

HuffmanCoder HC;

InitHuffmanCoder(HC, n);

CreateBook(HC, HT);

cout << endl << "请选择你想要进行的操作：" << endl << "1.默认编码ToBeTran文件" <<endl<< "2.选择文件进行编码" << endl << "3.输入字符进行编码"<<endl;

char c = getchar();

char ch[256];

string s;

string filename;

if (c == '1')

{

ifstream infile("ToBeTran.txt", ios::in);

if (!infile)

{

cerr << "Error opening file: " << filename << endl;

return;

}

getline(infile, s);

cout << "从文件ToBeTran中读取的待编码的字符串为：" << s << endl << endl;

//将大写字母转换为小写字母，进行下一步编码

for (char& c : s) {

c = tolower(c);

}

// 使用strcpy函数将string复制到char[]数组中

strcpy\_s(ch, sizeof(ch), s.c\_str());

cout << "编码结果:" << endl;

Coder(HC, ch, "CodeFile");

cout << "编码的结果同时放入数据文件CodeFile中" << endl << endl;

}

else if (c == '2') {

cout << endl << "请输入你想要进行译码的文件名：";

getchar();

cin >> filename;

ifstream infile(filename, ios::in);

if (!infile)

{

cerr << "Error opening file: " << filename << endl;

system("pause");

getchar();

return;

}

getline(infile, s);

cout << "从文件" << filename << "中读取的待编码的字符串为：" << s << endl << endl;

for (char& c : s) {

c = tolower(c);

}

strcpy\_s(ch, sizeof(ch), s.c\_str());

cout << "编码结果:" << endl;

string putfilename = filename + "\_Encoder.txt";

Coder(HC, ch, putfilename);

cout << "编码的结果同时放入数据文件" << putfilename << "中" << endl << endl;

}

else {

cout << "请输入待编码的字符：";

getchar();

getline(cin, s);

cout <<"\n\n输入的待编码的字符串为：" << s << endl << endl;

for (char& c : s) {

c = tolower(c);

}

strcpy\_s(ch, sizeof(ch), s.c\_str());

cout << "编码结果:" << endl;

string putfilename = s + "\_Encoder.txt";

Coder(HC, ch, putfilename);

cout << "\n编码的结果同时放入数据文件" << putfilename << "中" << endl << endl;

}

getchar();

system("pause");

}

void Decoding(HuffmanTree& HT, int n)

{

if (!HT.flag)

PutoutHT(HT, "hfmTree.txt", n);

cout << endl << "请选择你想要进行的操作：" << endl << "1.默认译码CodeFile文件" << endl << "2.选择文件进行译码" << endl << "3.输入字符进行译码" << endl;

char c = getchar();

char ch[256];

if (c == '1')

{

ifstream infile("CodeFile.txt", ios::in);

if (!infile)

{

cerr << "Error opening file: " << "CodeFile.txt" << endl;

return;

}

int j = 0;

while (infile.get(ch[j]))

j++;

ch[j] = '\0';

Decoder(HT,ch, "TextFile.txt");

}

else if (c == '2')

{

cout << endl << "请输入你想要进行译码的文件名：";

string filename;

cin >> filename;

ifstream infile(filename, ios::in);

if (!infile)

{

cerr << "Error opening file: " << filename << endl;

return;

}

int j = 0;

while (infile.get(ch[j]))

j++;

ch[j] = '\0';

string putfilename = filename + "\_Decoder.txt";

Decoder(HT, ch, putfilename);

}

else if (c == '3')

{

cout << endl << "请输入你想要进行译码的字符：";

cin >> ch;

string putfilename = string(ch) + "\_Decoder.txt";

Decoder(HT, ch, putfilename);

}

else {

cout << "输入错误";

}

getchar();

system("pause");

}

void Print()

{

cout << "请选择你想要进行的操作：" << endl << "1.打印默认的代码文件CodeFile" << endl << "2.选择需要打印的代码文件"<<endl;

char ch;

ch = getchar();

string filename;

if (ch == '1')

{

filename = "CodeFile.txt";

}

else if (ch == '2')

{

cout << endl << "请输入你想要进行译码的文件名：";

cin >> filename;

}

ifstream inputFile(filename);

// 检查文件是否成功打开

if (!inputFile.is\_open()) {

std::cerr << "无法打开"<<filename<<" 文件！" << endl;

return;

}

// 打开 CodePrin 文件进行写入

string putfilename = filename + "CodePrin.txt";

ofstream outputFile(putfilename);

// 检查文件是否成功创建

if (!outputFile.is\_open()) {

std::cerr << "无法创建"<<putfilename<<" 文件！" << endl;

return ; // 退出程序

}

string line;

while (getline(inputFile, line)) {

// 在终端上以紧凑格式显示每行50个代码字符

for (size\_t i = 0; i < line.length(); i += 50) {

cout << line.substr(i, 50) << endl;

}

// 将紧凑格式的编码写入 CodePrin 文件

for (size\_t i = 0; i < line.length(); i += 50) {

outputFile << line.substr(i, 50) << "\n";

}

}

// 关闭文件

inputFile.close();

outputFile.close();

getchar();

system("pause");

}

void printHuffmanTree(const HuffmanTree& huffmanTree, int root, int level = 0) {

if (root != -1) {

// 打印右子树

printHuffmanTree(huffmanTree, huffmanTree.ht[root].rchild, level + 1);

// 打印当前节点

for (int i = 0; i < level; i++) {

cout << "--";

}

if(huffmanTree.ht[root].rchild==-1&&huffmanTree.ht[root].lchild==-1)

cout << huffmanTree.ht[root].ch << "(" <<huffmanTree.ht[root].weight << ")" << endl;

else

cout << "(" << huffmanTree.ht[root].weight << ")" << endl;

// 打印左子树

printHuffmanTree(huffmanTree, huffmanTree.ht[root].lchild, level + 1);

}

}

// 递归方式写入哈夫曼树到文件

void writeHuffmanTreeToFile(const HuffmanTree& huffmanTree, int root, ofstream& outFile,int level=0) {

if (root != -1) {

// 写入右子树

writeHuffmanTreeToFile(huffmanTree, huffmanTree.ht[root].rchild, outFile,level+1);

for (int i = 0; i < level; i++) {

outFile << "--";

}

// 写入当前节点

if (huffmanTree.ht[root].rchild == -1 && huffmanTree.ht[root].lchild == -1)

outFile << huffmanTree.ht[root].ch << "(" << huffmanTree.ht[root].weight << ")" << endl;

else

outFile << "(" << huffmanTree.ht[root].weight << ")" << endl;

// 写入左子树

writeHuffmanTreeToFile(huffmanTree, huffmanTree.ht[root].lchild, outFile,level+1);

}

}

void Tree\_printing(HuffmanTree& HT,int n)

{

if (!HT.flag)

PutoutHT(HT, "hfmTree.txt", n);

cout << "Huffman Tree:" << endl;

printHuffmanTree(HT, 2\*(n-1));

// 写入文件

ofstream outFile("TreePrint.txt");

if (outFile.is\_open()) {

outFile << "Huffman Tree:" << endl;

writeHuffmanTreeToFile(HT, 2 \* (n - 1), outFile);

outFile.close();

cout << "Huffman Tree has been written to TreePrint.txt" << endl;

}

else {

cerr << "Unable to open the file." << endl;

}

system("pause");

}

int main()

{

HuffmanTree HT;

int end = 1;

int n;

//int n= Getn("hfmTree.txt");//已正确读取n

char result='q';

while (end) {

result = menu();

switch (result)

{

case 'i':

Initialization(HT, n);

break;

case 'e':

if(!initflag)

n = Getn("hfmTree.txt");

Encoding(HT, n);

break;

case 'd':

if (!initflag)

n = Getn("hfmTree.txt");

Decoding(HT, n);

break;

case 'p':

if (!initflag)

n = Getn("hfmTree.txt");

Print();

break;

case 't':

if (!initflag)

n = Getn("hfmTree.txt");

Tree\_printing(HT,n);

break;

case 'q':

cout << "正在退出系统......";

end = 0;

break;

default:

break;

}

}

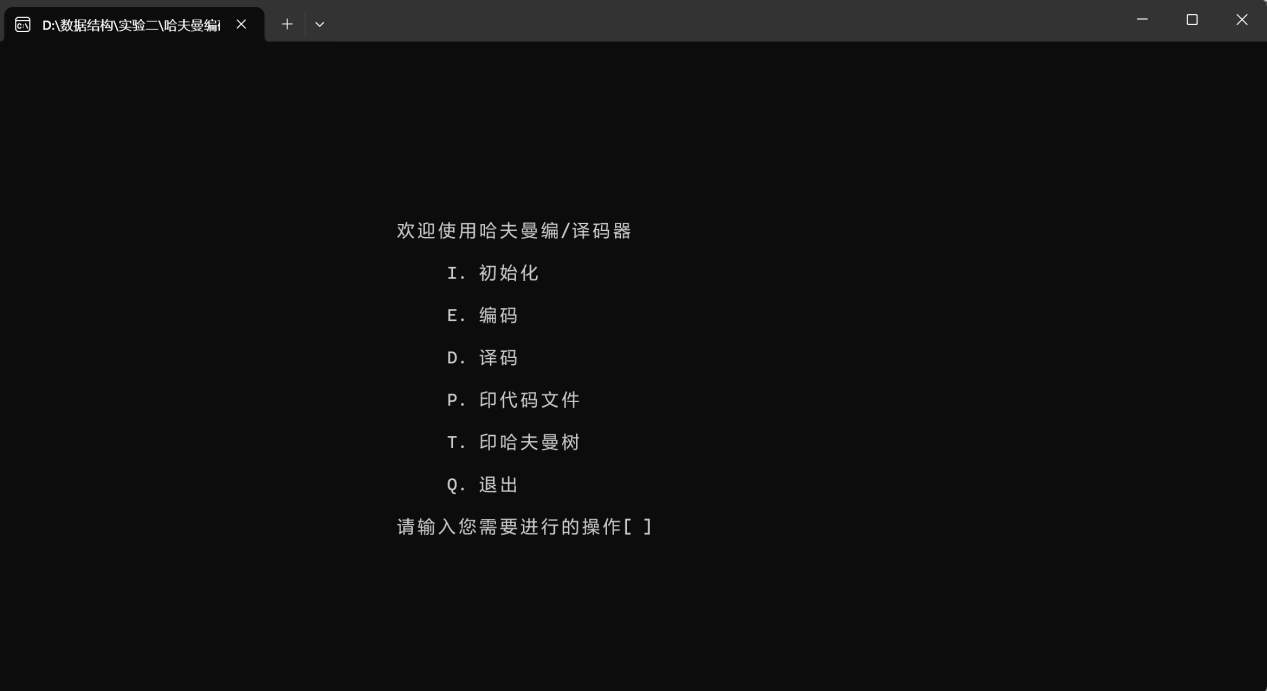
}

##### 调试分析

1. 建立哈夫曼树的第i个结点需要在前i-1个结点中挑选出两棵无双亲结点而且根结点权值最小的子树。筛选过程采用简单选择，时间复杂度为O(n),这样的选择需要重复n-1次，所以建立哈夫曼树的时间复杂度为O(n\*n)
2. 对每个字符的编码，是从树叶结点出发自下而上搜索至根结点的过程，这一搜索过程的时间复杂度为O(lg n)，对有n个树叶的哈夫曼树而言，建立编码表的时间复杂度为O(n\*lg n)
3. 编码的过程是对所有需要编码的每一个字符到编码表中查找，这一查找过程的时间复杂度为O(n),若需要编码的字符有m个，则编码过程的时间复杂度为O（n\*m）
4. 译码的过程是对二进制电文从根节点出发做自上而下搜搜，当搜搜到树叶结点时则确定了一段二进制电文对应的一个字符，这一搜索过程的时间复杂度是O(lg n),若二进制电文中可译出的字符数为m，则译码的过程的时间复杂度为O(m\*lg n)

##### 用户手册

1、本程序的执行文件为：哈夫曼编码.exe

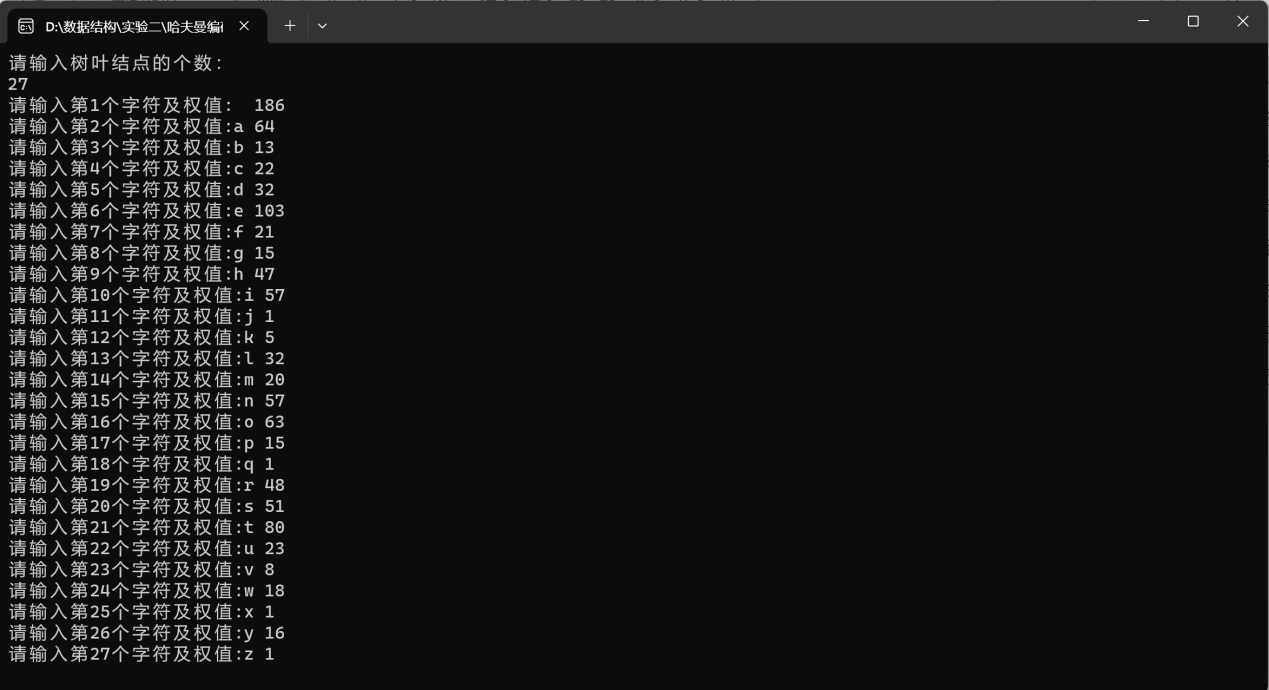
2、进入演示程序后，将显示如下的界面，根据提示进行选择想进行的操作

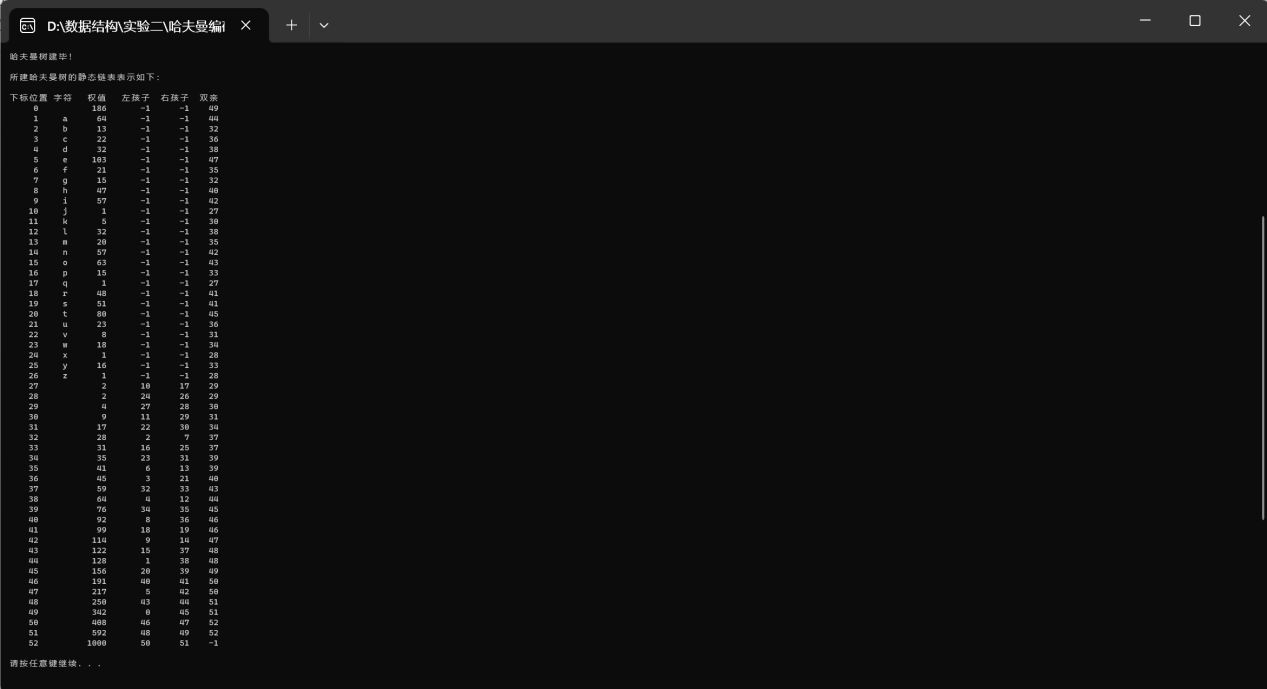
##### 测试结果

下面是各个功能的实现情况

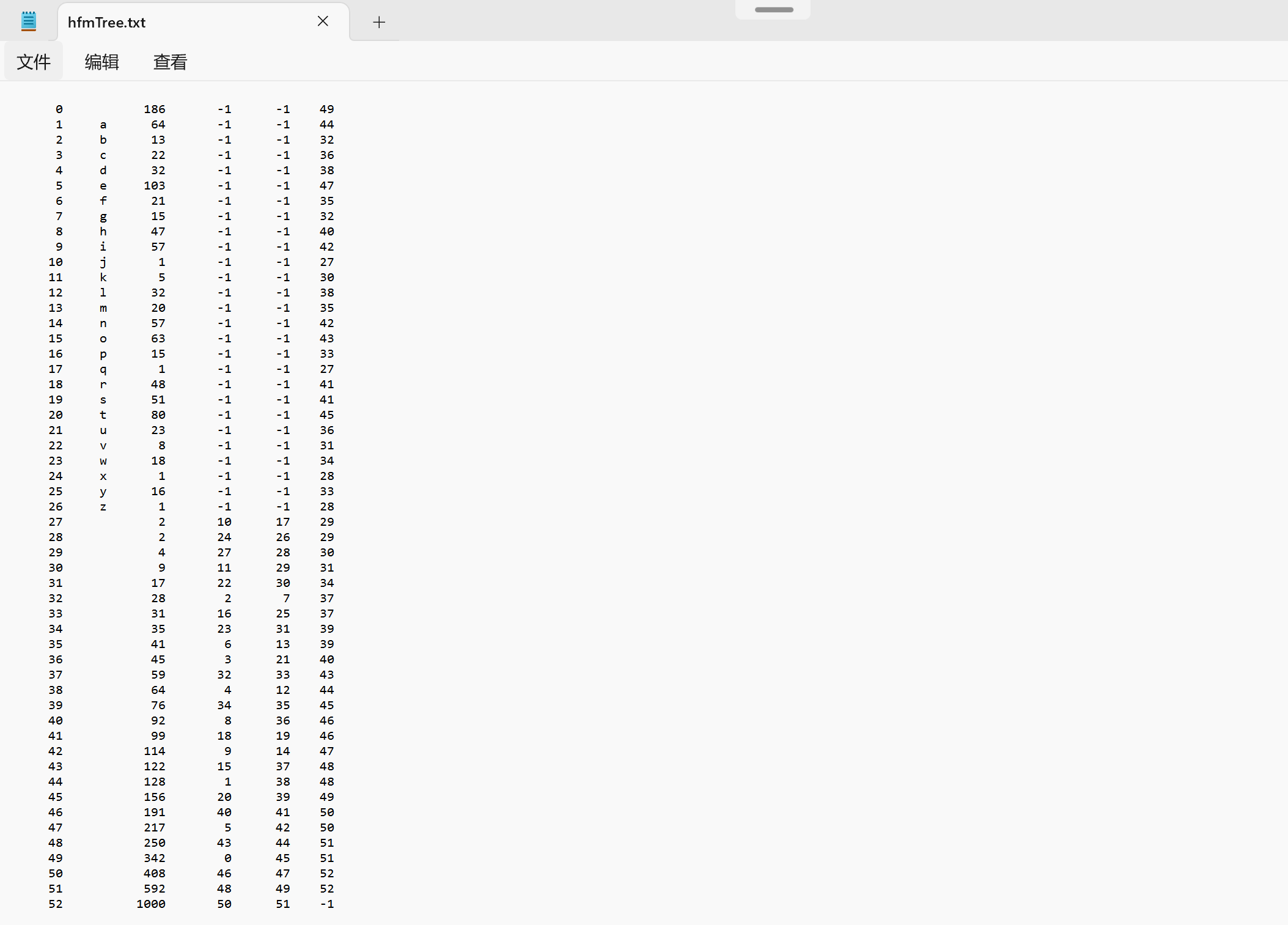
1. 输入i或I，进行初始化

输入树的叶子结点个数，再依次输入字符以及权值



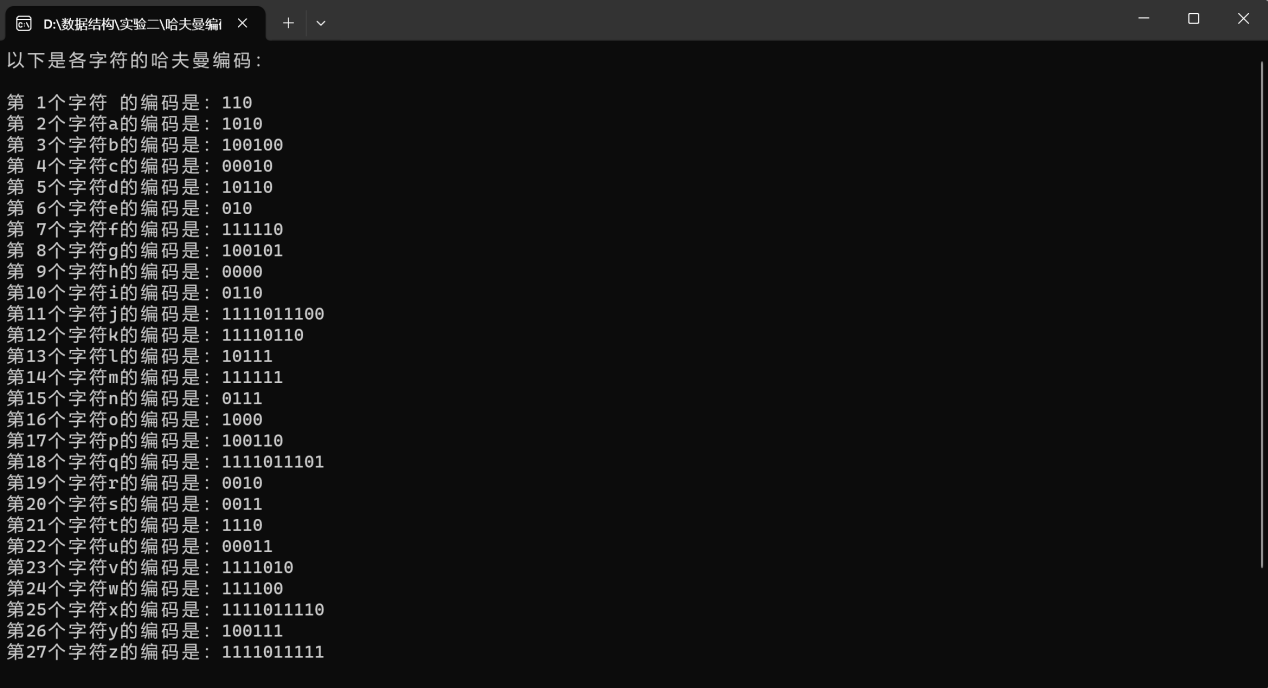
建立哈夫曼树，并输出哈夫曼树的静态链表

存于文件“hfmTree”中

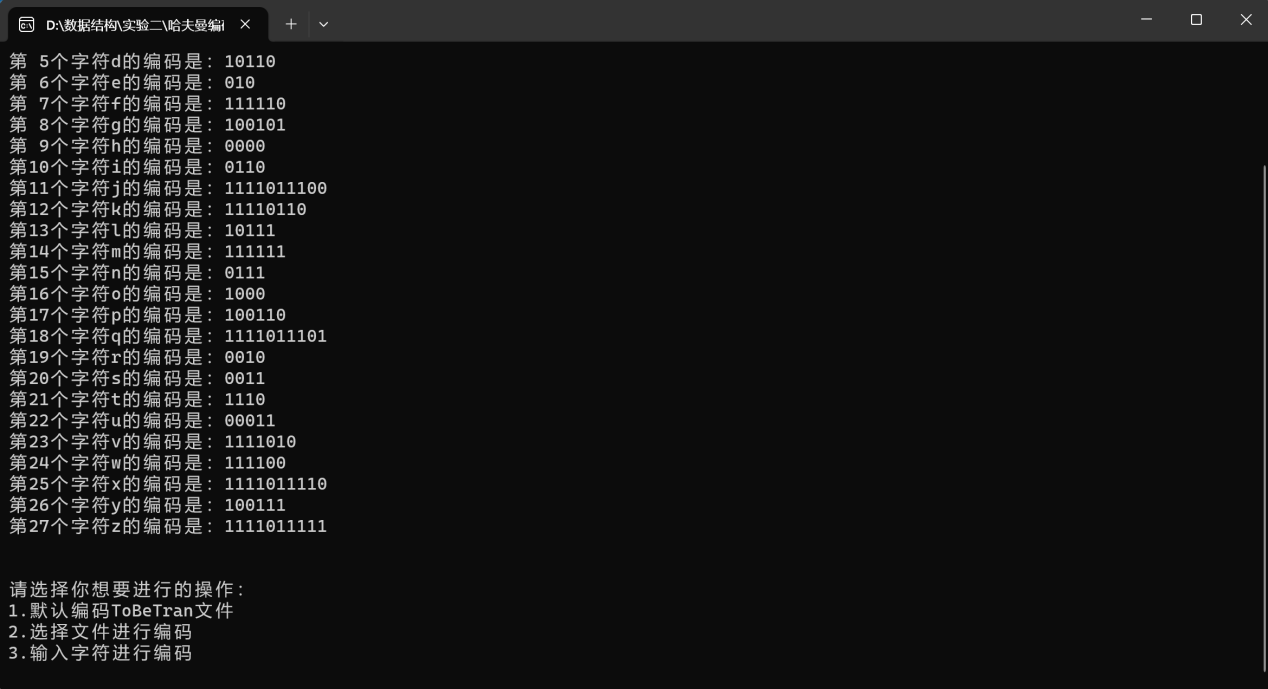


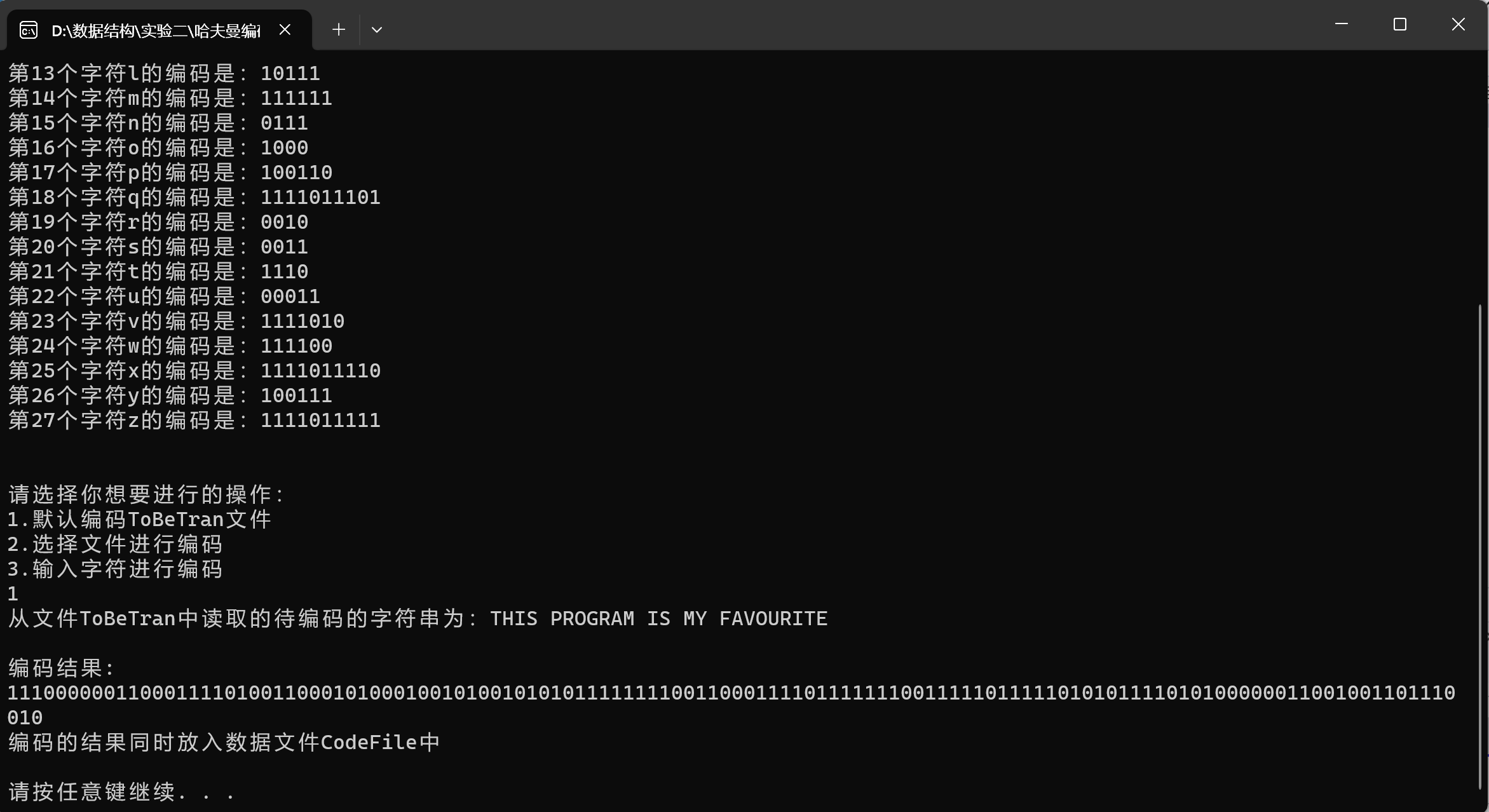
1. 输入e或E，进行编码

显示根据在内存中的或者从文件“hfmTree.txt”中读入的哈夫曼树建立的各个字符的编码

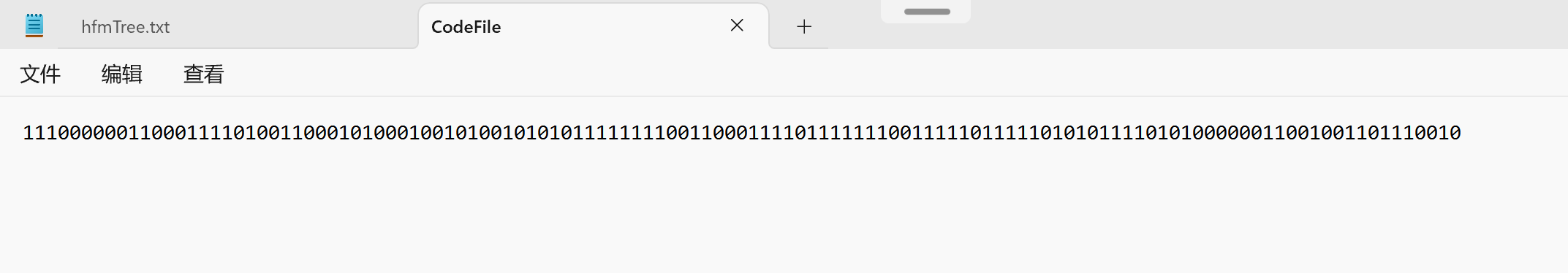


用户进行选择进行编码的内容

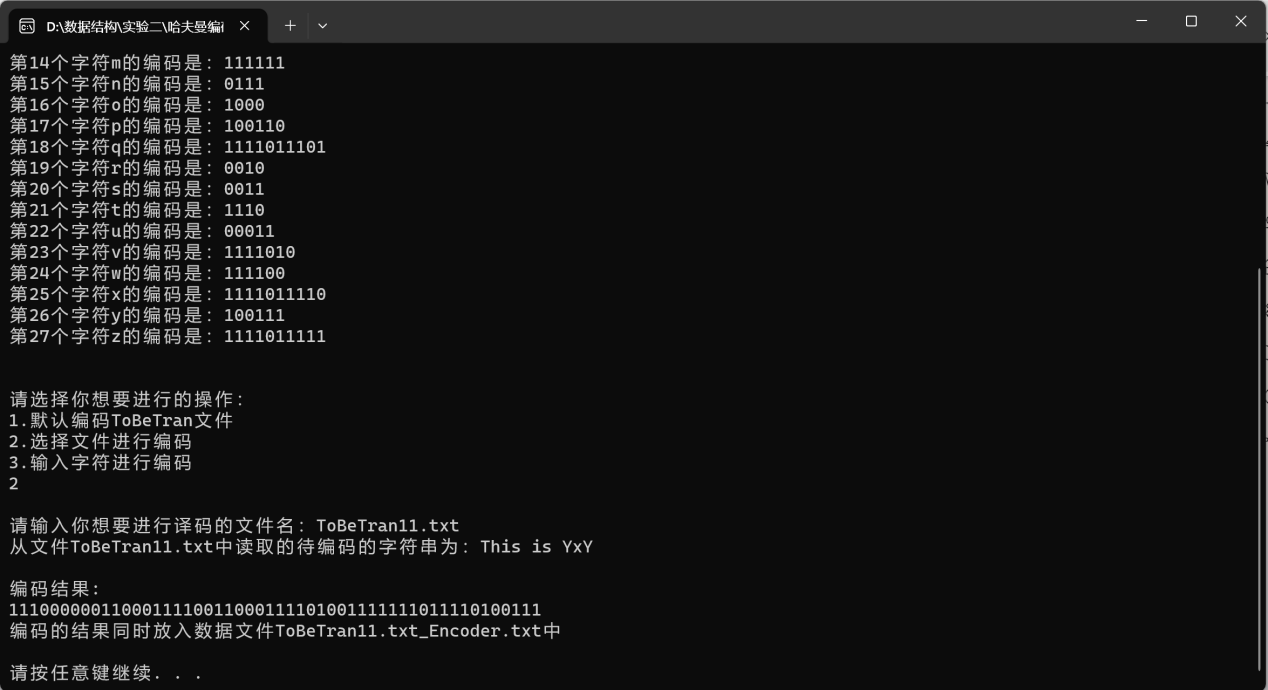


输入1选择默认文档“ToBeTran.txt”

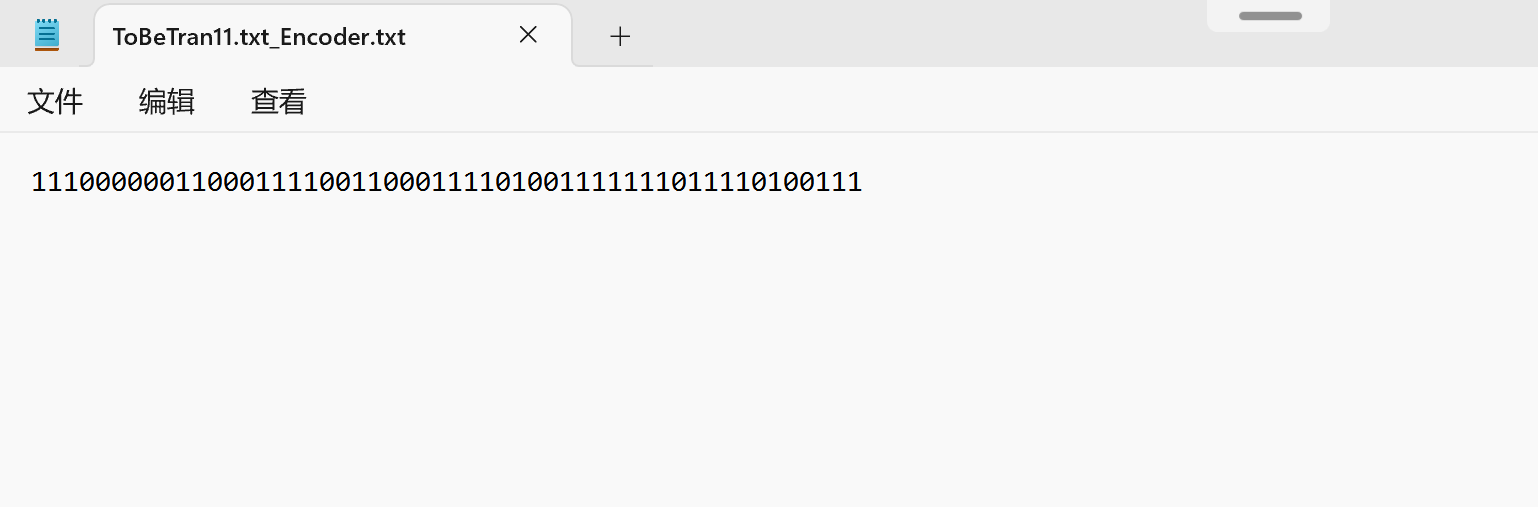
编码结果存入文件“CodeFile”中



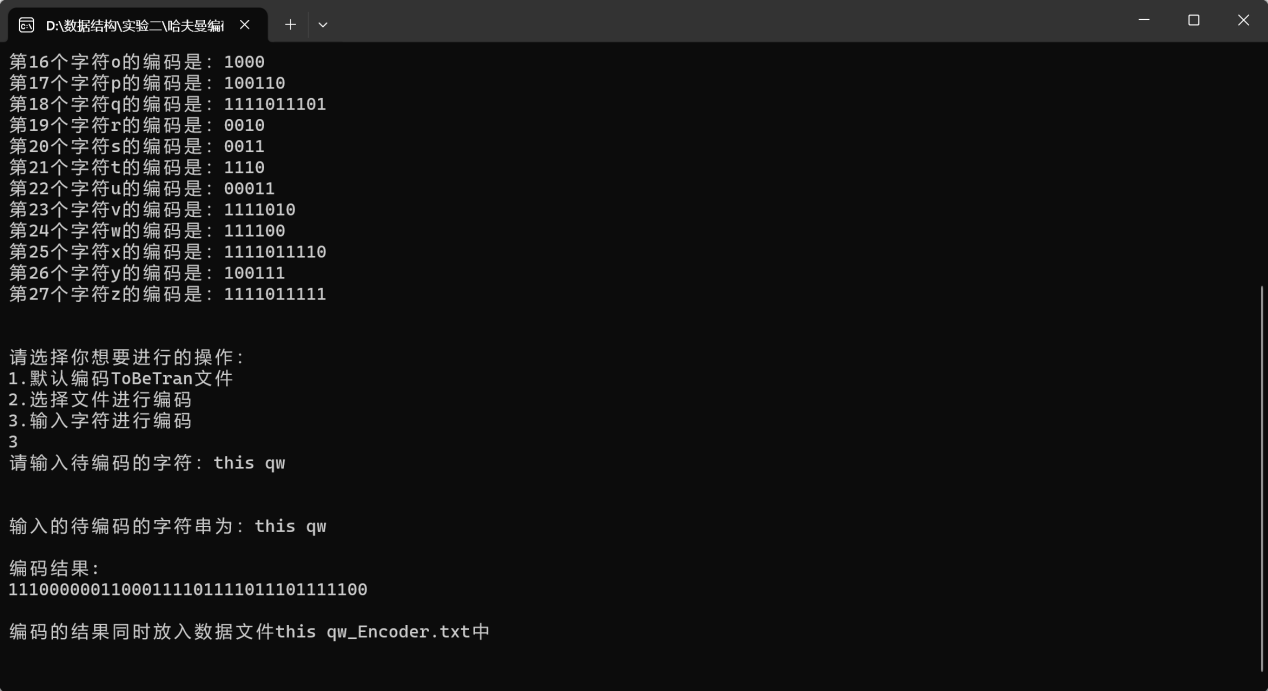
输入2选择文件进行编码



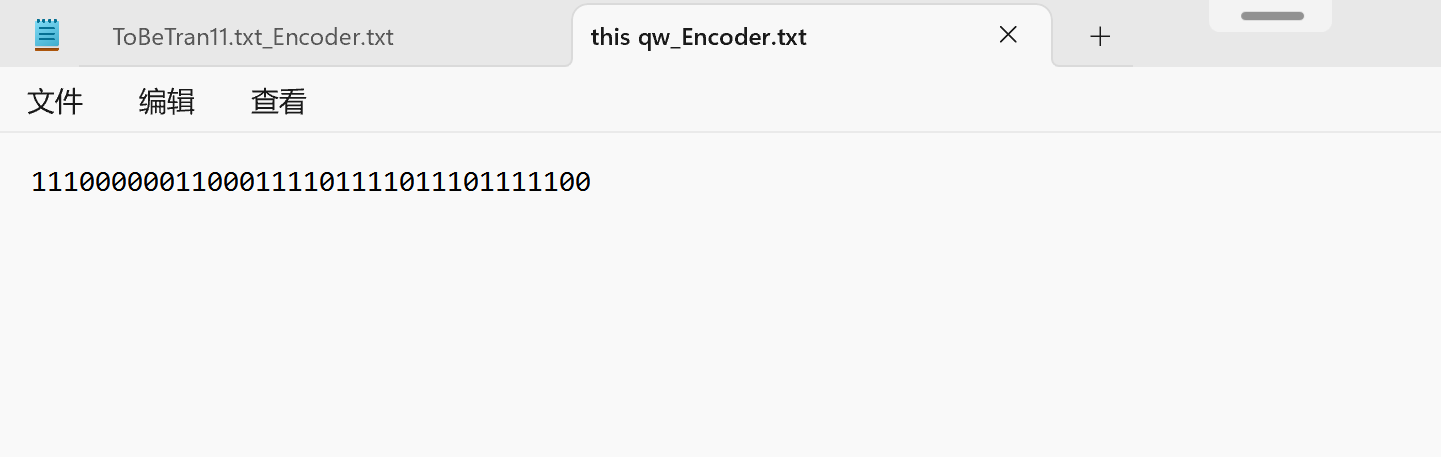
编码结果存入文件中



输入3在终端输入内容进行编码

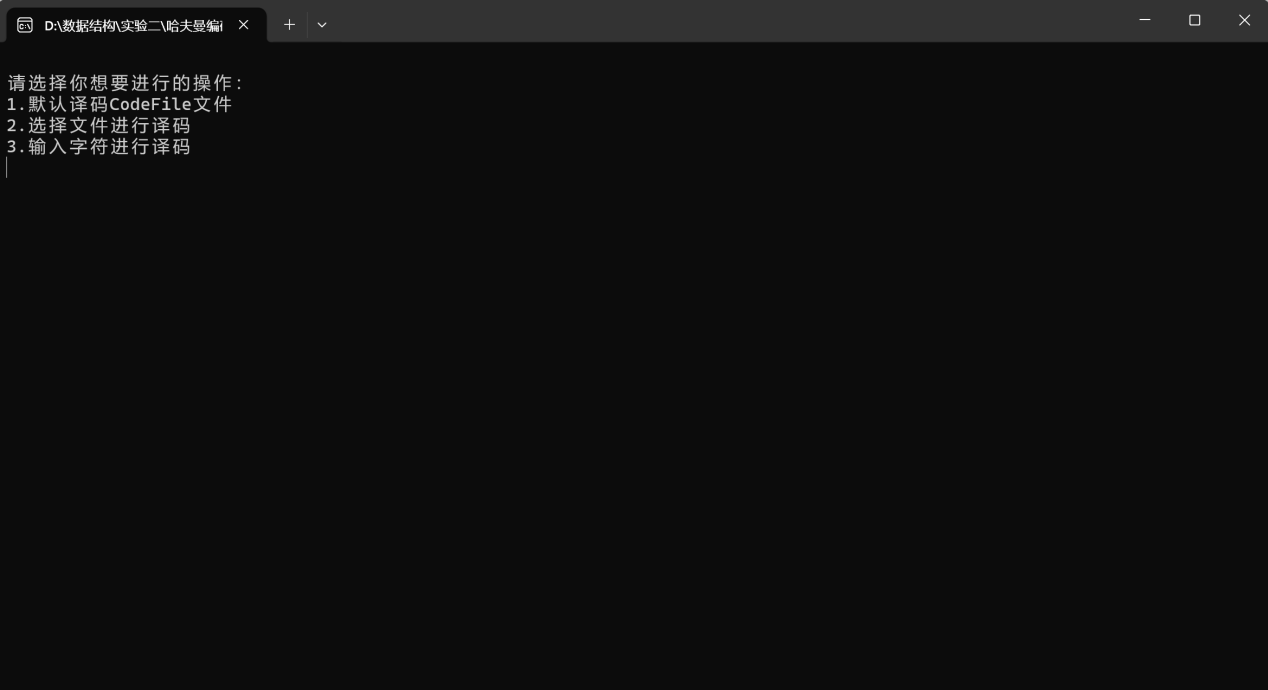


编码结果存入文件

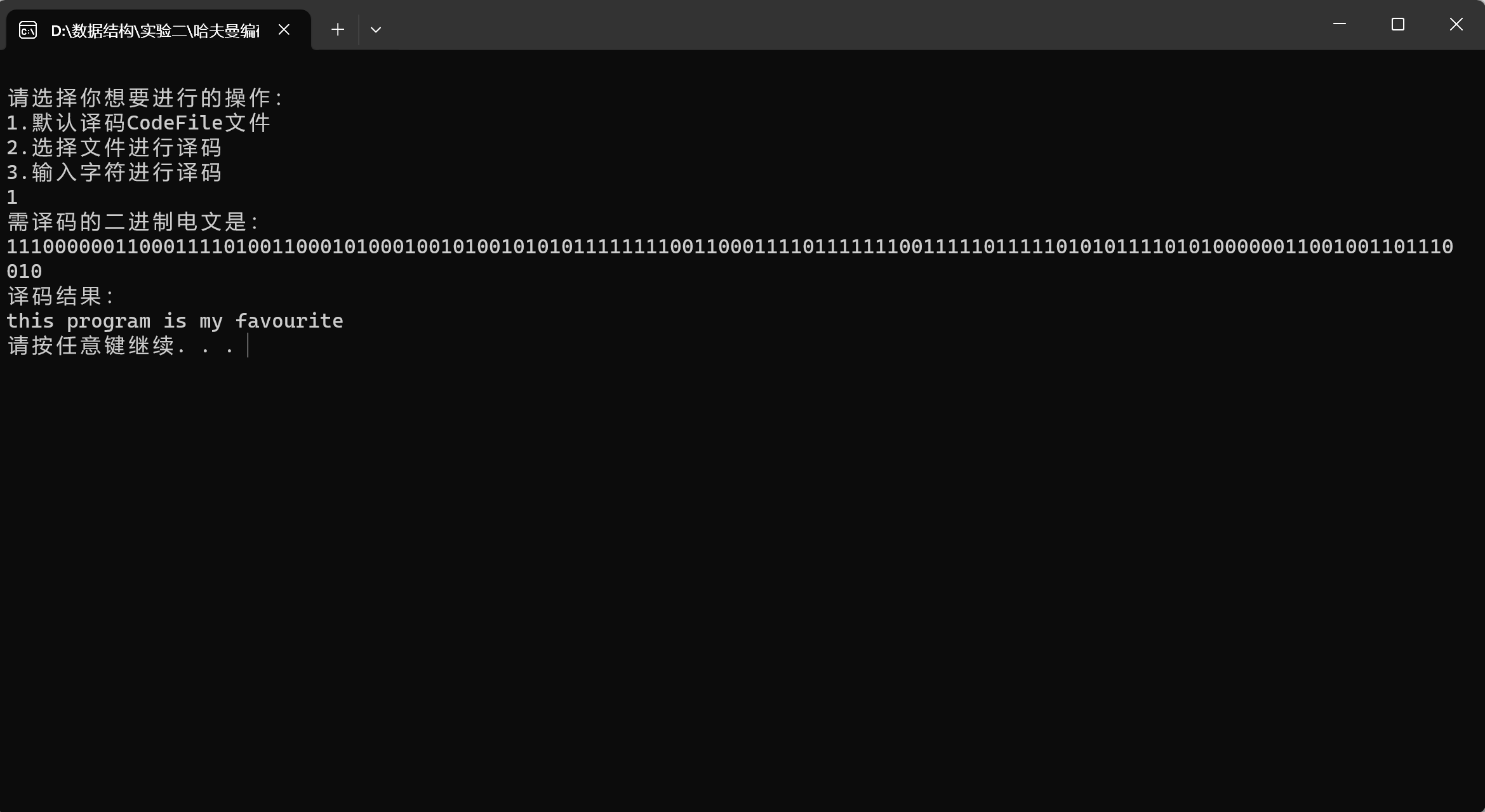


1. 输入d或D，进行译码

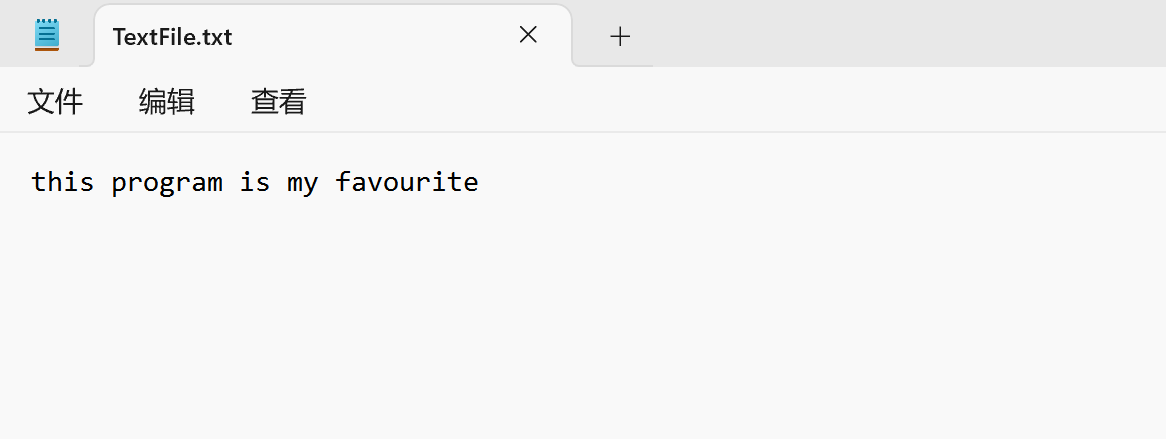
进行选择需要译码的内容



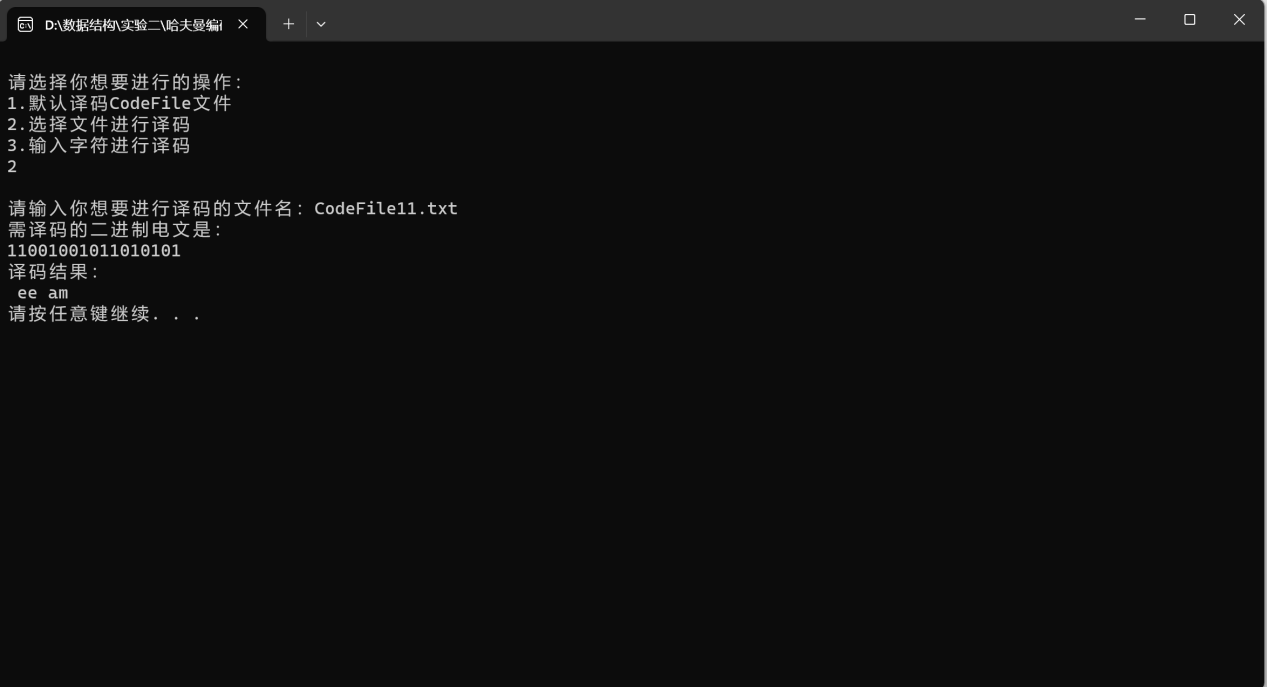
输入1译码默认文档“CodeFile”



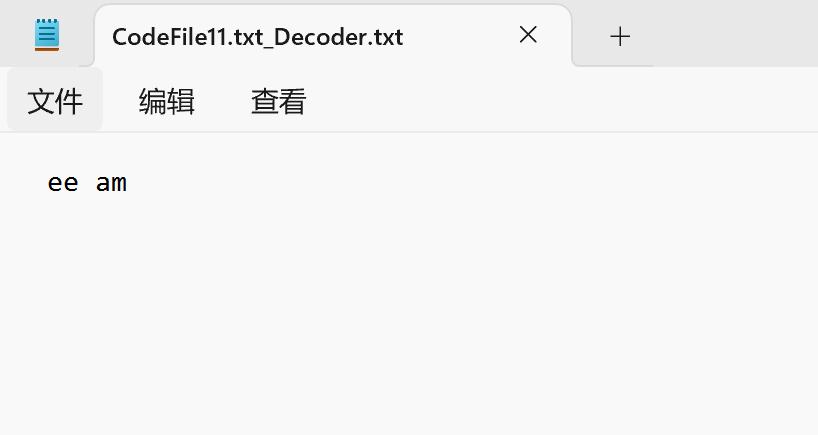
译码结果存入文件“TestFile”



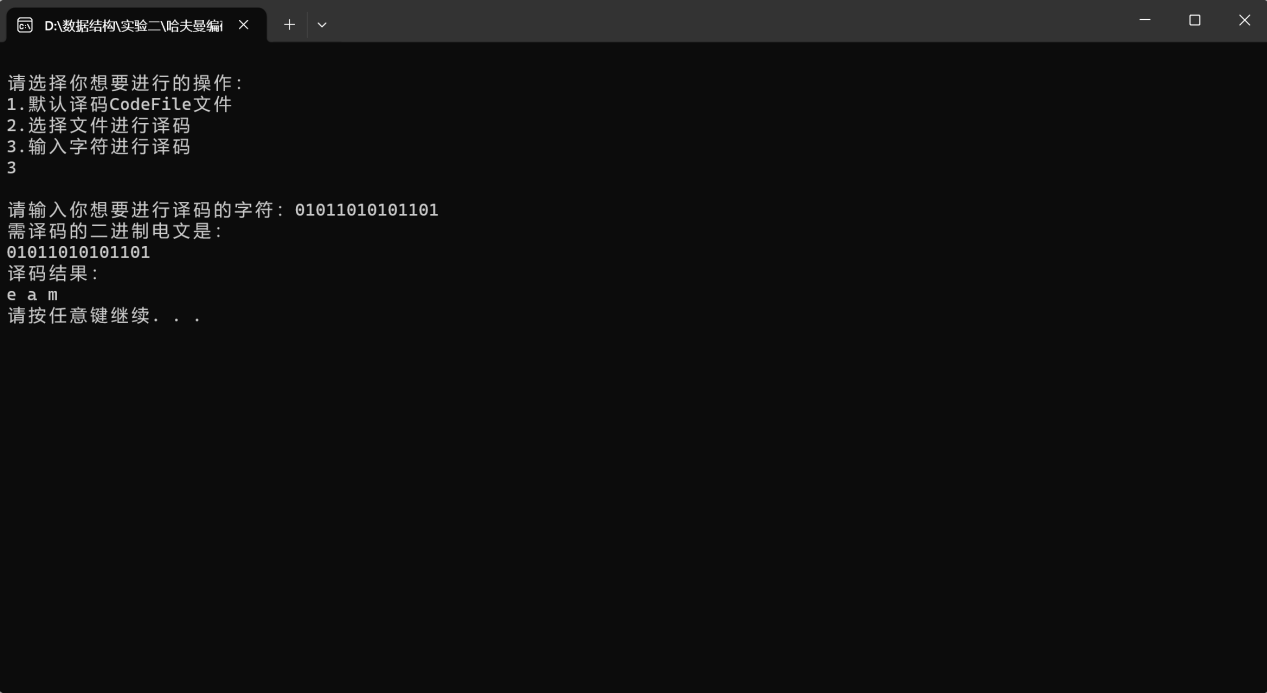
输入2选择文件进行译码



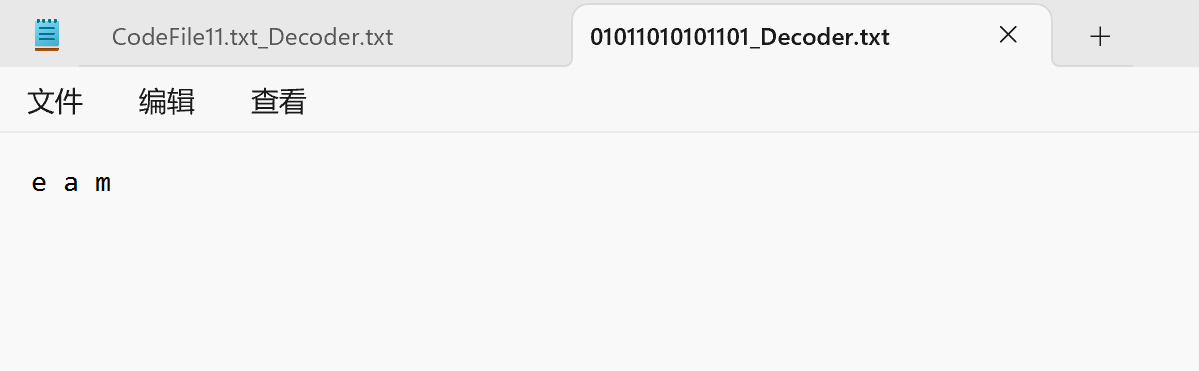
译码结果存入文件



输入3在终端输入内容进行译码

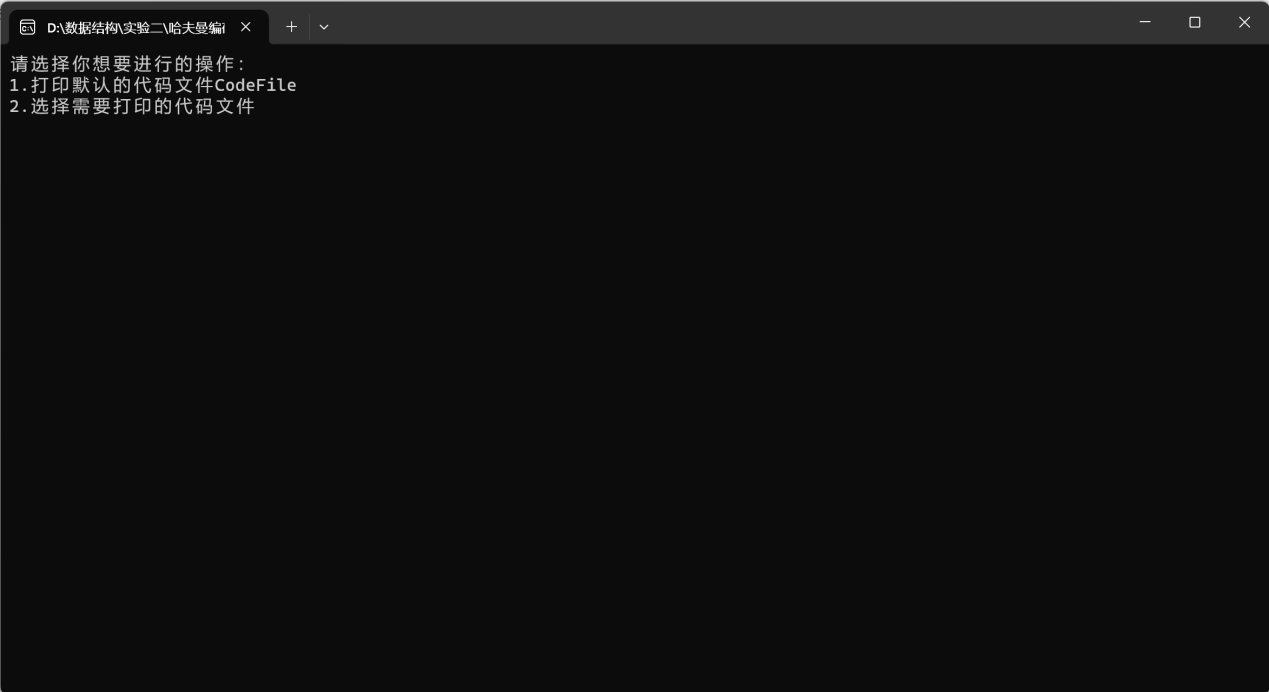


结果存入文件

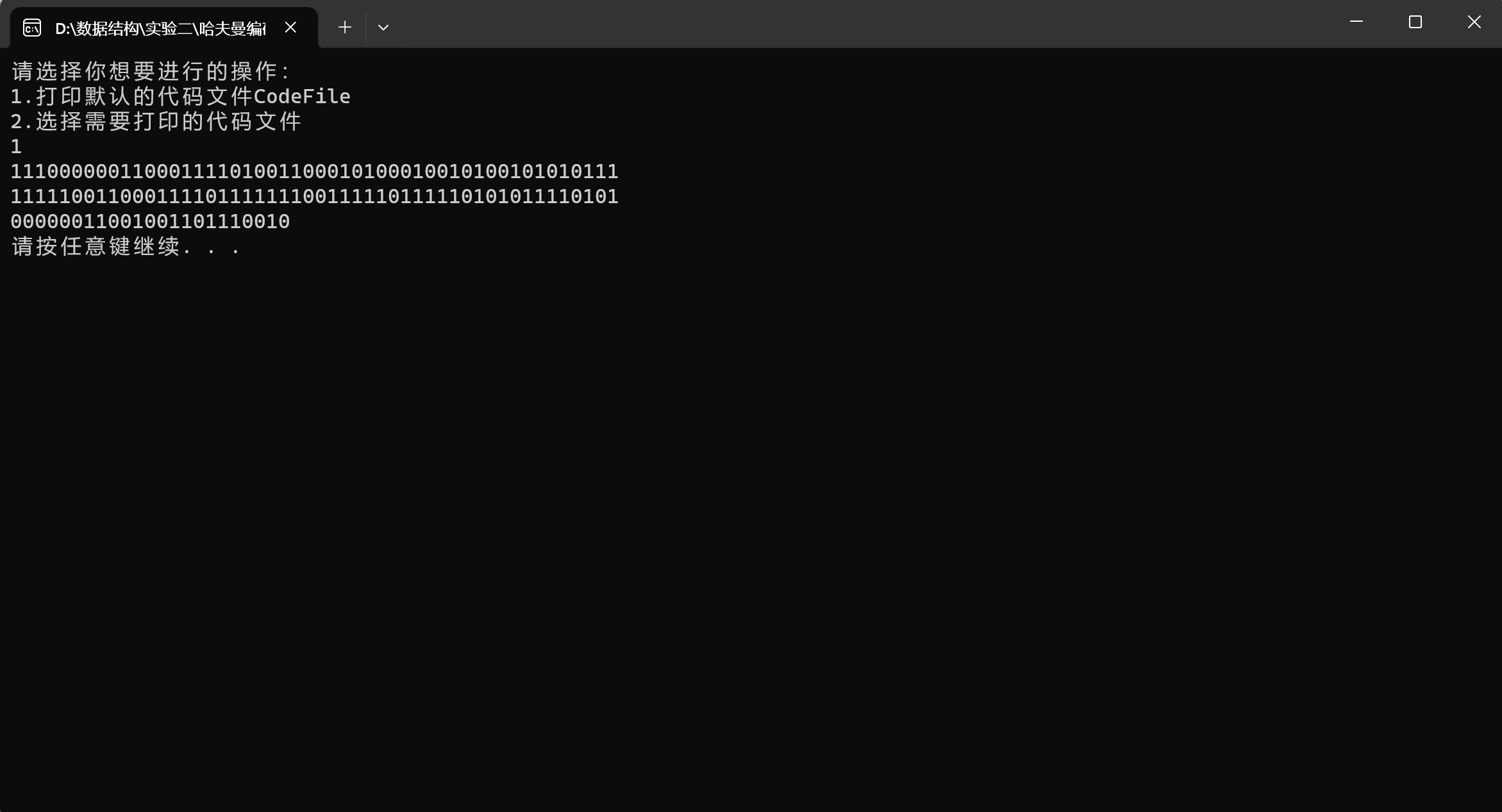


1. 输入p或P，进行印代码文件

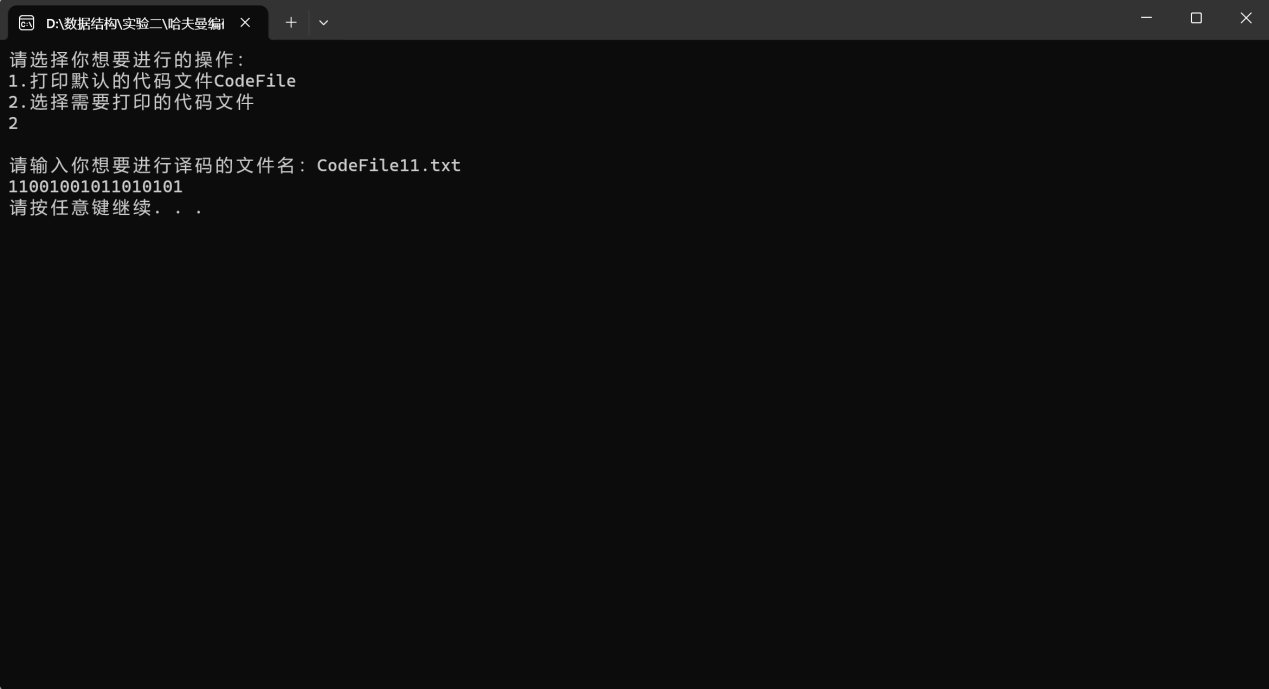
进行选择需要打印的代码文件



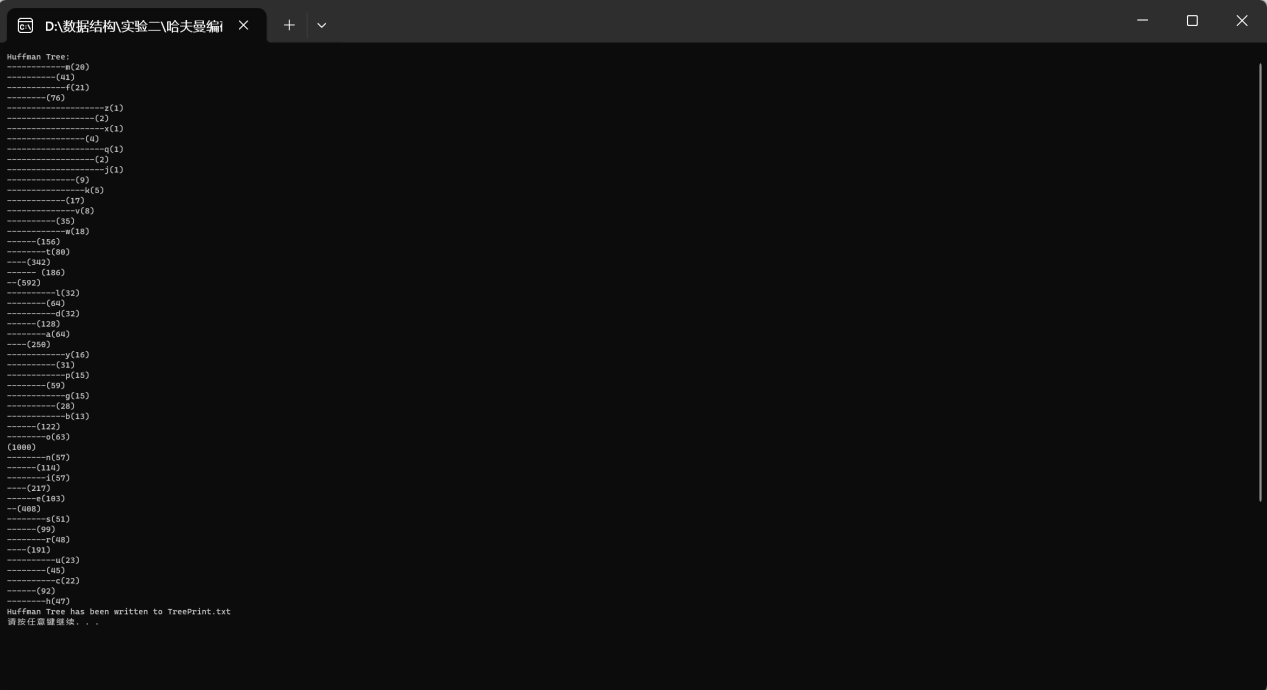
输入1打印默认的代码文件“CodeFile”



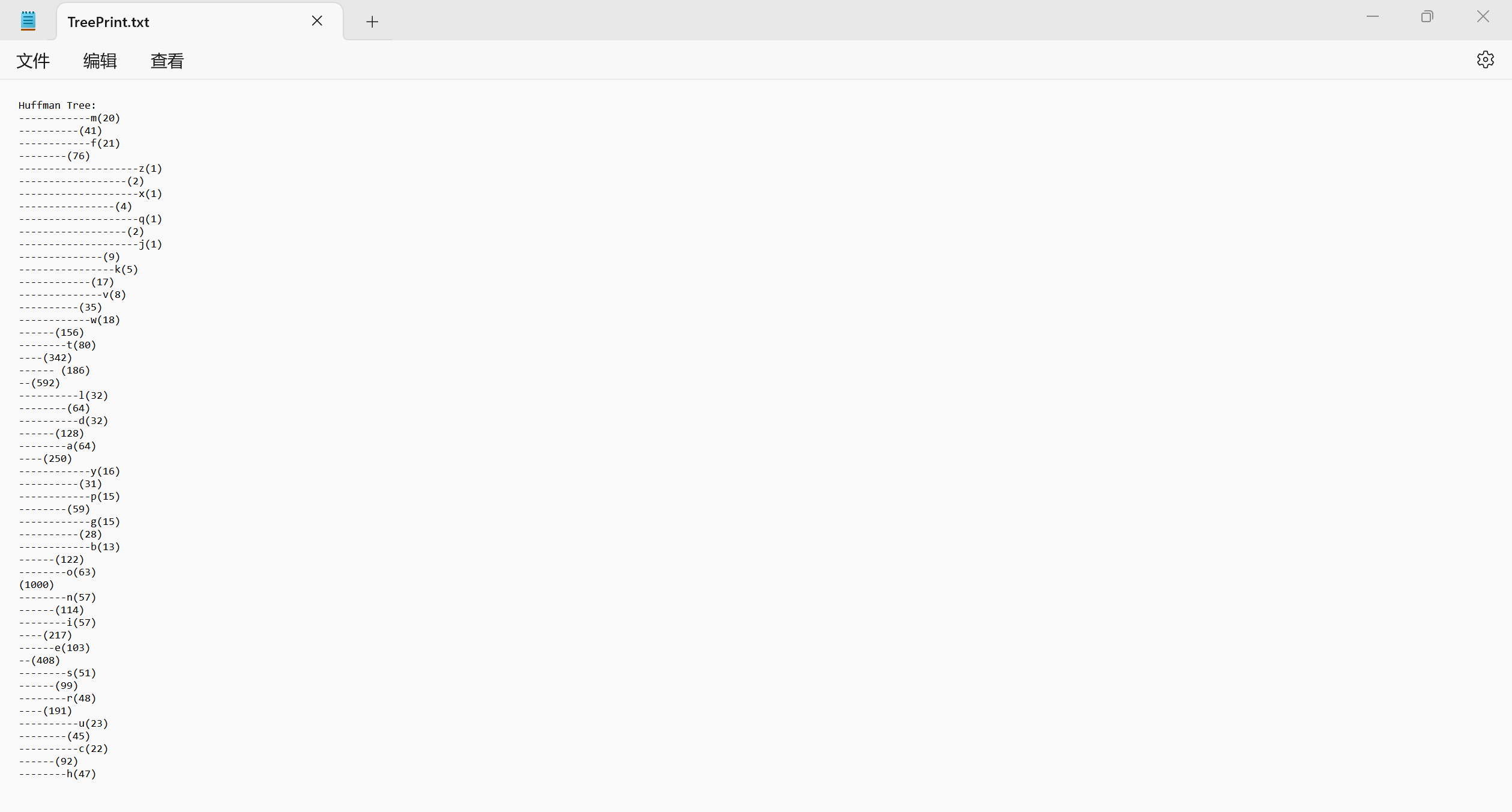
输入2选择文件进行打印代码文件



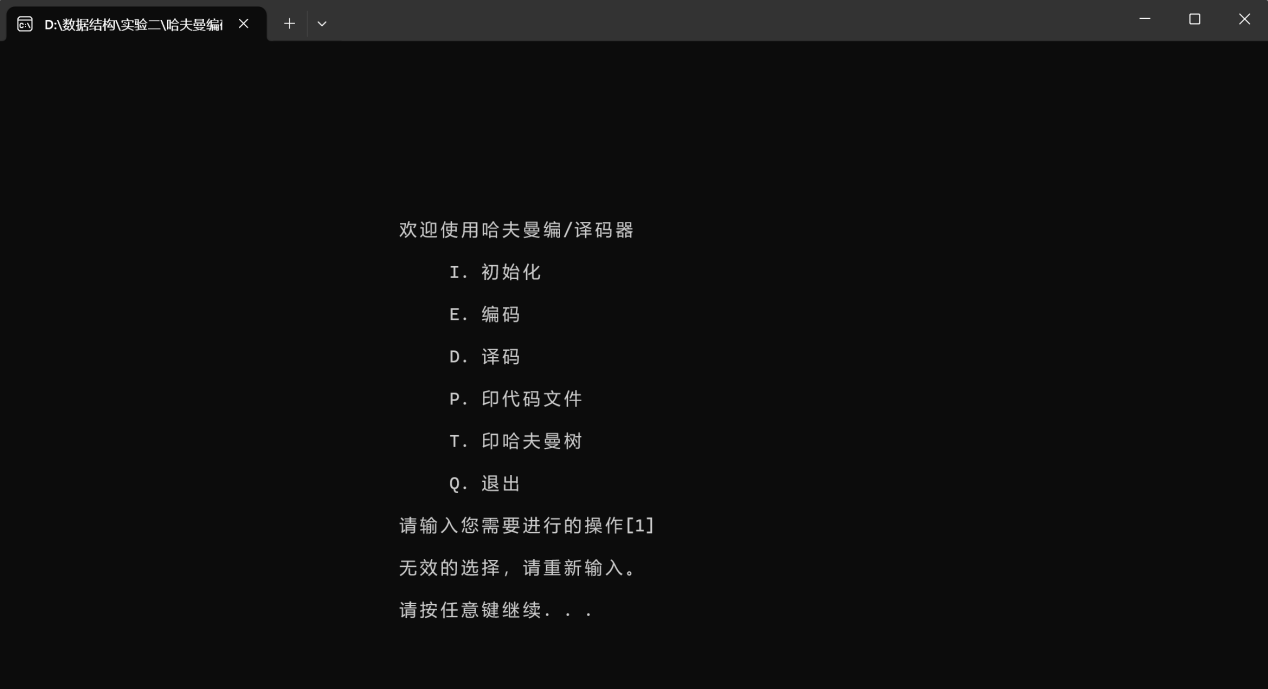
1. 输入t或T，进行印哈夫曼树



存入文件



1. 输入q或Q，退出系统
2. 增加了容错处理



PS：详细的运行结果请打开哈夫曼编码.exe文件

##### 附录

源程序文件名清单：main.cpp，哈夫曼编码.exe, Haffman.h