



数据结构课程设计报告

**2020~2021 学年第 1 学期**

**题目: 三叉树文件加密**

# 院系: 智能制造学部

# 班级:

**学号:**

**姓名: 陈思绵**

**教师： 金旺春**

完成日期：2020 年 11 月 19 日

**题目：文件加密系统**

**完成功能：**

1. 对密码本进行字符的统计，以及计算其权值，存储在线性表中。（梁沃辉）
2. 对二叉树结构修改为三叉树结构（森林），并利用1中的线性表建立三叉树结构形成012串，与文件中字符进行比对，生成文件编码Code.txt。（林凯）
3. 对需要解密的文件以及密码本进行比对，解密并形成解密文件decode.txt（甘倩茵）
4. 前端界面以及选择功能。（甘倩茵）

## 摘要

在计算机技术突飞猛进的今天，加密程序的开发越来越受到开发者的青睐。本次数据结构课程设计选择文件加密系统，系统主要使用了哈夫曼编码技术，开发了一个对英文文本文件进行加密和解密的程序。在技术上对哈弗曼编码中的最优二叉树进行改进，由二叉树变为三叉树（森林），减少了编码文件的空间，并且在编码过程中我们采用动态分配叶子的方法，一旦密码本中的字符计数出现增加或者减少，或者说密码本中字符的顺序发生改变，生成的012串也会相应的做出改变，而不会把每一个字符的编码给写死。同时，支持用户自定义选择密码本以及加密解密文件。

## 目录

目录

[摘要 2](#_Toc57672170)

[目录 3](#_Toc57672171)

[第1章 课程设计内容 4](#_Toc57672172)

[1.1 设计目的 4](#_Toc57672173)

[1.2 设计要求 4](#_Toc57672174)

[1.3 功能要求 4](#_Toc57672175)

[第2章 系统分析 5](#_Toc57672176)

[2.1 系统结构图 5](#_Toc57672177)

[2.2 函数结构图 5](#_Toc57672178)

[第3章 详细设计 6](#_Toc57672179)

[3.1 HuffmanTreeLib.h文件（三叉树结构主要实现层） 6](#_Toc57672180)

[3.2 Count.h（密码本字符计数功能） 13](#_Toc57672181)

[第4章 运行与测试 17](#_Toc57672182)

[4.1 用户界面 17](#_Toc57672183)

[4.2 选择密码本（用户自定义密码本，密码本选择不同时） 17](#_Toc57672184)

[4.3 文件加密（需在选择密码本后进行） 19](#_Toc57672185)

[4.4 文件解密（需在选择密码本后进行） 22](#_Toc57672186)

[总结 25](#_Toc57672187)

[系统优点 25](#_Toc57672188)

[系统缺点 25](#_Toc57672189)

[设计体会 25](#_Toc57672190)

[参考文献 27](#_Toc57672191)

[附录 28](#_Toc57672192)

# 第1章 课程设计内容

## 1.1 设计目的

目前大多数用户常用的对文件进行加密的方法就是密码保护, 而这种加密方法简单适用, 但它同时也留下了巨大的安全隐患。对于一些破译高手而言, 往往很容易攻破这种密码保护。要想让非法用户很难破译甚至是几乎不可能破译, 他才会望而却步, 而这样做的惟一方法就是使密码尽可能复杂同时对用户而言又满足某种规律。

由于计算机文件中的字符具有统计特性, 能够统计出每种字符在文件中出现的概率。当文件中所有字符的概率统计出来后, 就可以用多元哈夫曼编码的方法给文件生成对应的哈夫曼编码表, 然后用这个哈夫曼编码表对文件进行加密。由于这个哈夫曼编码表足够复杂同时又满足编码的规律, 非法用户想要破解密文几乎是不可能的。[1]

因为在上学期已经实现了二叉树的基本结构以及基本算法，所以本次实际应用我们选择该课题，再次深入了解树或者森林的魅力。

本次课程设计的目标：

1. 回顾《数据结构》中线性表以及二叉树的基本知识，加深对数据结构理解。
2. 掌握C++编程方法以及程序调试的基本技能。
3. 尝试对数据结构进行创新以及改进，改变其空间复杂度以及时间复杂度。

## 1.2 设计要求

（1）以某文件为样本进行哈夫曼编码或其它编码

（2）输入待加密文件进行加密

（3）输入待解密文件解密

## 1.3 功能要求

（1）能够对一个文件其中的字符进行统计，统计其出现的字母（中文）个数。

（2）能够对一个文件进行加密，并输出加密文件以及其密码本

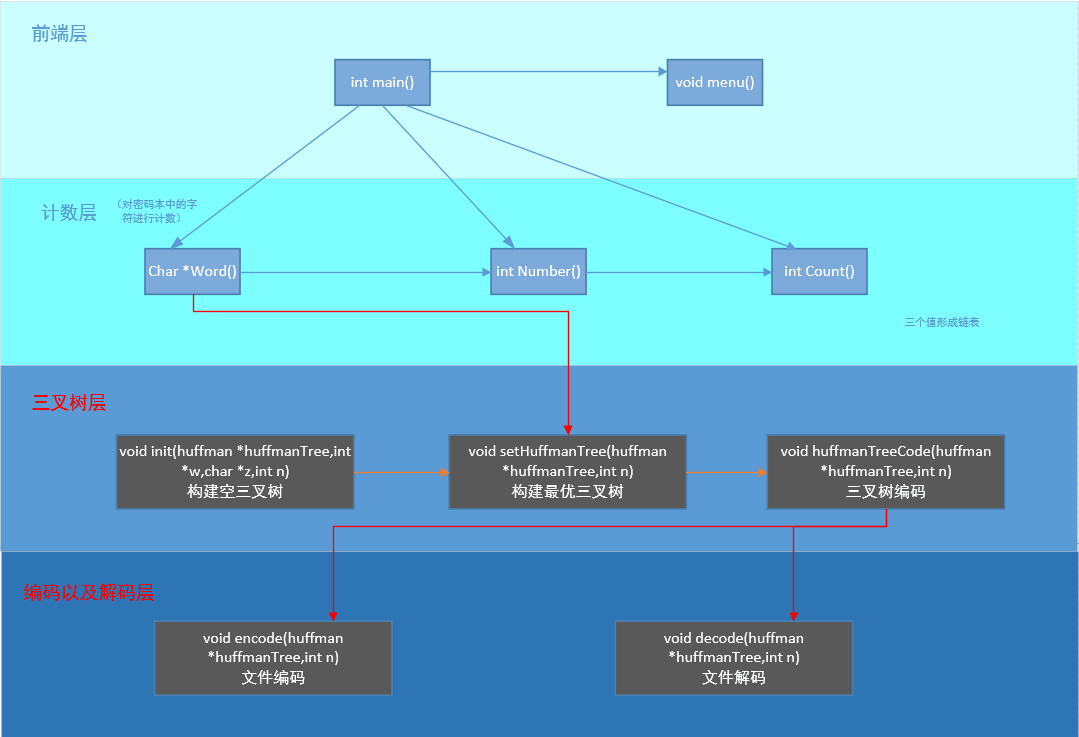
（3）能够将加密生成的文件还原成源文件

# 第2章 系统分析

## 2.1 系统结构图

系统结构图

## 2.2 函数结构图



函数结构图

## 第3章 详细设计

算法设计（功能描述和核心算法设计与实现）

### 3.1 HuffmanTreeLib.h文件（三叉树结构主要实现层）

#### 3.1.1 void init(huffman \*huffmanTree,int \*w,char \*z,int n)

//构建三叉树算法

void init**(**huffman **\***huffmanTree**,**int **\***w**,**char **\***z**,**int n**)**

**{**

int i**;**

//(3n-1)/2) n为奇数时计算结点个数

**if(**n**%**2 **==** 1**)**

**{**

**for(**i**=**0**;**i**<(**3**\***n**-**1**)/**2**;**i**++)**

**{**

huffmanTree**[**i**].**ch**=**z**[**i**];**

huffmanTree**[**i**].**weight**=**w**[**i**];**

huffmanTree**[**i**].**parent**=-**1**;**

huffmanTree**[**i**].**lChild**=-**1**;**

huffmanTree**[**i**].**mChild**=-**1**;**

huffmanTree**[**i**].**rChild**=-**1**;**

**}**

**}**

//3n/2 n为偶数时计算结点个数

**else**

**{**

**for(**i**=**0**;**i**<(**3**\***n**)/**2**;**i**++)**

**{**

huffmanTree**[**i**].**ch**=**z**[**i**];**

huffmanTree**[**i**].**weight**=**w**[**i**];**

huffmanTree**[**i**].**parent**=-**1**;**

huffmanTree**[**i**].**lChild**=-**1**;**

huffmanTree**[**i**].**mChild**=-**1**;**

huffmanTree**[**i**].**rChild**=-**1**;**

**}**

**}**

**}**

因为本次课程设计我们使用的是三叉树，所以与二叉树在结构上有根本性不同，主要体现在结点数为奇数和偶数时的树结构不同：

1. 当叶子结点为偶数时：（设权值weight：2、4、5、8、9、10、12、13、18、20、24、25、30、32），构成最优三叉树的结构如图1所示。

设叶子结点个数为n，度为1的结点个数为m，度为2的结点为p（根节点），度为3的结点个数为q。

即：n+p+m+q=eq+2p+m+1（三叉树的结点数等于这个树的度+1）

得：n=2q+1（其中m=0，p=1）

∴总结点个数N=n+m+p+q=3n/2

当叶子结点个数为14时，总的叶子结点个数是21个

∴当叶子结点个数为 n时，最优三叉树的结点总数为3n/2

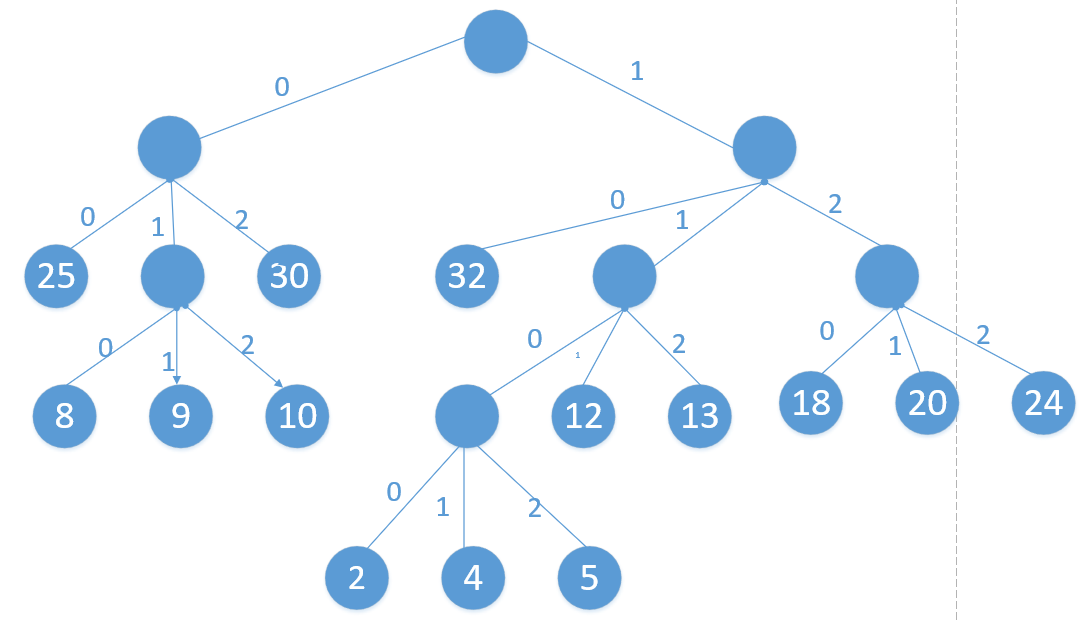


图1

1. 当叶子结点数为奇数时：(设权值weight：2、4、5、8、9、10、12、13、14、18、20、24、25、30、32），构成最优三叉树的结构如图2所示

设叶子结点个数为n，度为2的结点为p（根节点），度为1的结点个数为m，度为3的结点个数为q。

即： n+p+m+q=3q+2p+m+1（一颗三叉树的结点数=这棵树的度加1）

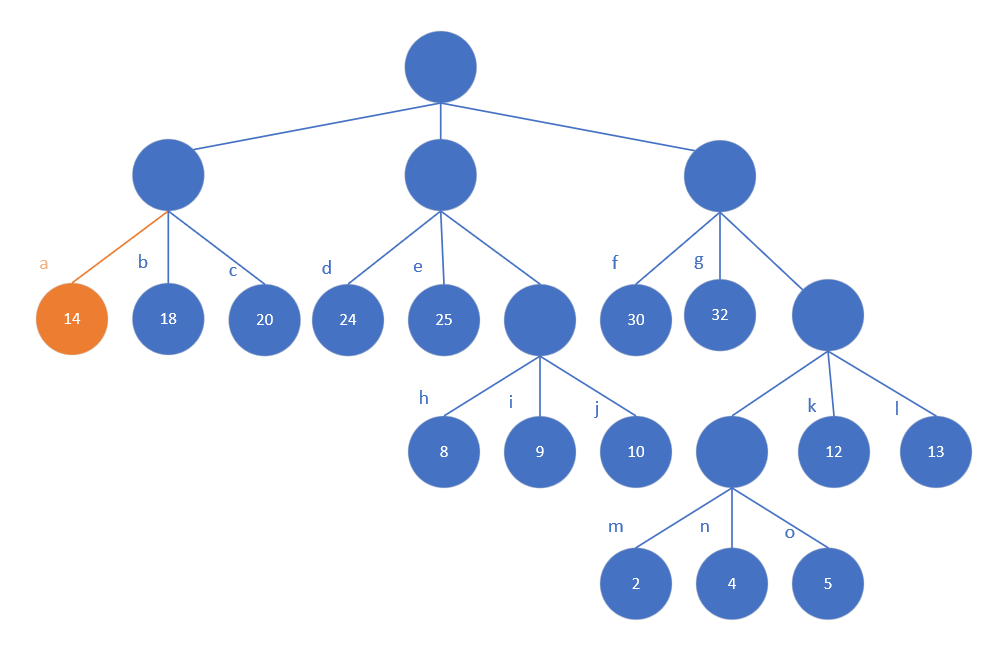
得：n=2q+1（其中m=0,p=0）

总结点个数N=n+m+p+q=(3n-1)/2

当叶子结点个数为15时，总的叶子结点个数是22个（正确）

所以当叶子结点个数为n时，最优三叉树的结点总数是(3n-1)/2

1. 这个函数的主要功能是新建空结点，并将孩子结点和双亲结点置为-1。



图二

#### void setHuffmanTree(huffman \*huffmanTree,int n) （构建最优三叉树）

//构建最优三叉树

void setHuffmanTree(huffman \*huffmanTree,int n)

{

int m,m1,m2,m3,x1,x2,x3,i,j;

if(n%2 == 1)

{

m=(3\*n-1)/2;

}

else

m=3\*n/2;

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

{

//定义m1，m2，m3,的值为1000，作为比较大小交换的载体

m1=m2=m3=MaxValue;

//初始化x1，x2，x3

x1=x2=x3=0;

//比较最小权值，然后相加

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

{

//交换过程

if(huffmanTree[j].parent==-1&&huffmanTree[j].weight<m1)

{

m2=m1;

x2=x1;

m1=huffmanTree[j].weight;

x1=j;

}

else if (huffmanTree[j].parent==-1&&huffmanTree[j].weight<m2)

{

m2=huffmanTree[j].weight;

x2=j;

}

else if (huffmanTree[j].parent==-1&&huffmanTree[j].weight<m3)

{

m3=huffmanTree[j].weight;

x3=j;

}

}

//偶数个节点时生成双亲节点

if(n%2 == 1)

{

huffmanTree[x1].parent=i;

huffmanTree[x2].parent=i;

huffmanTree[x3].parent=i;

huffmanTree[i].lChild=x1;

huffmanTree[i].mChild=x2;

huffmanTree[i].rChild=x3;

huffmanTree[i].weight=m1+m2+m3;

}

//奇数节点时生成双亲节点

else

{

huffmanTree[x1].parent=i;

huffmanTree[x2].parent=i;

huffmanTree[x3].parent=i;

huffmanTree[i].lChild=x1;

huffmanTree[i].mChild=x2;

huffmanTree[i].rChild=x3;

if(n!=m-1)

{

//计算双亲节点权值，方便用于循环中进行构建三叉树

huffmanTree[i].weight=m1+m2+m3;

}

else

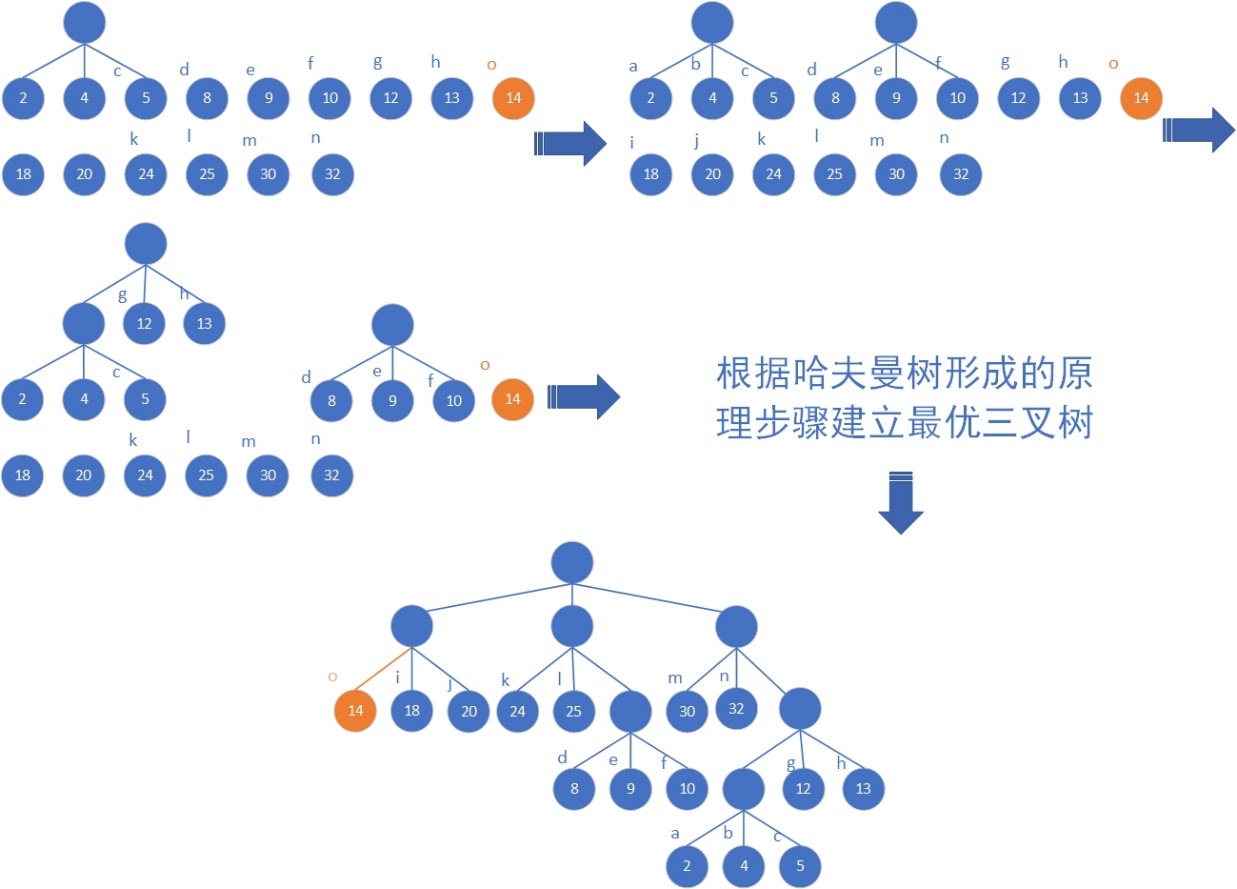
huffmanTree[i].weight=m1+m2;

}

}

}

最优三叉树奇数如图三所示，偶数同理：



#### 字符编码函数void huffmanTreeCode(huffman \*huffmanTree,int n)

void huffmanTreeCode(huffman \*huffmanTree,int n)

{

//定义数组存储树节点的字符

char hc[Max];

//叶子节点的012编码长度

int hcLen;

int i,j,k,parent,p;

//先生成叶子节点对应的012编码，然后存储到叶子节点中

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

{

hcLen=0;

//待编码字符的双亲结点下标

parent=huffmanTree[i].parent;

p=i;

while(parent!=-1)//未到达根结点

{

if(huffmanTree[parent].lChild==p)//是左孩子

hc[hcLen]='0',hcLen++;

else if(huffmanTree[parent].mChild==p)//是中孩子

hc[hcLen]='1',hcLen++;

else if(huffmanTree[parent].rChild==p)//是右孩子

hc[hcLen]='2',hcLen++;

p=parent;

parent=huffmanTree[parent].parent;//继续向根结点查找

}

//将编码写入相应字符数组

for(j=0,k=hcLen-1;j<hcLen;j++,k--)

huffmanTree[i].hCode[j]=hc[k];

huffmanTree[i].hCode[j]='\0';//加上字符串结束符

}

return;

}

根据哈夫曼编码规则将012赋予每个字符，图如图一所示。

#### HuffmanTreeLib ADT

**huffmanTree**

存放字符ch;

存放权值weight;

存放左子树lChild;

存放中子数mChild;

存放右子树rChild;

存放该结点的父亲节点parent;

存放该结点的012串hCode[Max];

**Operation**

**init**

前置条件：三叉树不存在

功能：三叉树的初始化

后置条件：构造一个空的三叉树

输入：无

输出：无

调用规则：无

编写者：林凯

最后修改时间：2020年11月10日

**setHuffmanTree**

前置条件：三叉树存在

功能：最优三叉树的构建

后置条件：交换结点使得形成最优三叉树

输入：无

输出：无

调用规则：无

编写者：林凯

最后修改时间：2020年11月10日

**huffmanTreeCode**

前置条件：最优三叉树存在

功能：三叉树编码

后置条件：形成三叉树编码并用链表存储

输入：无

输出：无

调用规则：无

编写者：林凯

最后修改时间：2020年11月10日

**Encode**

前置条件：有Source.txt字符文件

功能：用流方式使得文件内字符形成编码

后置条件：关闭输入输出流

输入：Source.txt

输出：Code.txt

调用规则：无

编写者：林凯

最后修改时间：2020年11月29日

**decode**

前置条件：有Code.txt 012编码文件

功能：使得密码解码

后置条件：关闭输入输出流

输入：Code.txt

输出：Code2.txt

调用规则：无

编写者：甘倩茵

最后修改时间：2020年11月29日

### Count.h（密码本字符计数功能）

#### 3.2.1 链表（线性表）结构函数 LNode

typedef struct LNode

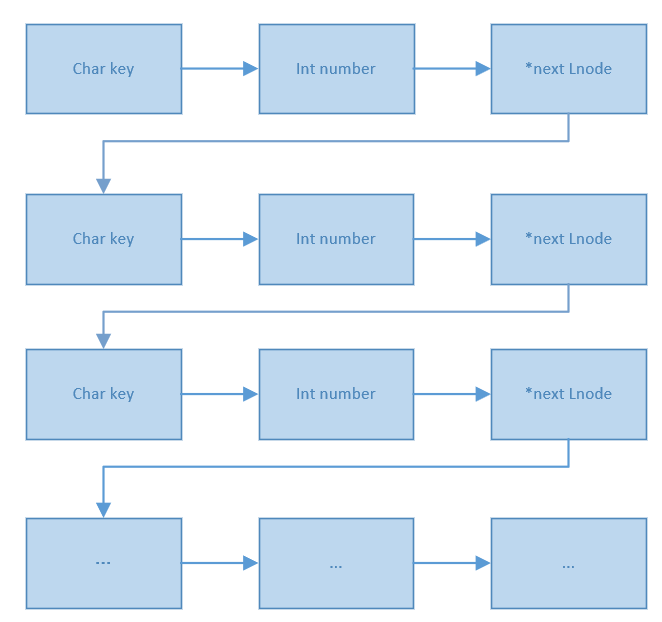
{ char key;

int number;

LNode \*next;

}\*Link;

由于本次线性表结构主要用来统计出现的字符以及其出现的次数，所以线性表采用链表结构，链表结构如图四所示。



图四

#### 3.2.2 初始化链表函数 CreatNode

Link CreateNode(void)

{

//初始化链表节点

Link newnode =(LNode\*)malloc(sizeof(LNode));

if(newnode!=NULL)

{

newnode->number=0;

newnode->next=NULL;

}

return newnode;

}

把number初始化为-1

#### 3.2.3 计数函数 Number()

int \*Number()

{

char c;

bool flag;

int i = 0;

static int W[Max];

Link head=NULL;//头指针

Link current=head;//当前结点

ifstream in\_stream;//源文件

in\_stream>>noskipws;

ofstream out\_stream;//输出文件

in\_stream.open("Source.txt");

//打开源文件错误

if(in\_stream.fail())

{

cout<<"inserting to the file is failing";

exit(1);

}

out\_stream.open("Output.txt");

//打开输出文件错误

if(out\_stream.fail())

{

cout<<"inserting to the file is failing";

exit(1);

}

//遍历文件，依次读取每个字符

while(in\_stream>>c)

{

flag=true;//标志c不存在于链表中

//遍历链表，判断c是否已经存在于链表中

for(current=head;current!=NULL;current=current->next)

{

//c已经存在，c对应的频数加1

if(current->key==c)

{

current->number++;

flag=false;//flag置为false

break;//跳出for循环

}

}

if(flag)

{

//c不存在，创建新结点

Link newnode=CreateNode();

newnode->key=c;

newnode->number++;

newnode->next=head;

head=newnode;

}

//w[i]=current->number;

//z[i]=current->key;

//i++;

}

//cout<<"The result:"<<endl;//控制台输出

//输出至输出文件

out\_stream<<"字母"<<"\t频数"<<endl;

//遍历链表，分别输出至控制台和文件中

for(current=head;current!=NULL;current=current->next)

{

//cout<<current->key<<' '<<current->number<<'\n';

out\_stream<<current->key<<"\t"<<current->number<<endl;

W[i]=current->number;

i++;

}

return W;

//关闭流

in\_stream.close();

out\_stream.close();

//释放链表空间

for(current=head;current!=NULL;current=current->next)

delete current;

}

① 计数函数是将文件中的字符通过文件流ofstream以及ifstream的方式从磁盘流入内存中或者从内存流入磁盘中。

② 由于文件流默认会不读取空格以及回车符等空白符，所以在文件输入流的时候进行一个操作，即in\_stream>>noskipws; 设置in\_stream读取空白符。

③ 函数读取字符的过程中也会检索链表中的其他元素，查看是否有重复元素，如果有则会在number上+1。、

④ 文件流需要在使用后关闭，否则会造成读操作的内存溢出问题以及写操作的缓存转移至硬盘内。

#### 3.2.4 Count.h ADT

**LNode**

存放字符key;

存放权值number;

**Operation**

**CreateNode**

前置条件：没有形成链表

功能：初始化链表并将number赋0

后置条件：无

输入：无

输出：无

调用规则：无

编写者：梁沃辉

最后修改时间：2020年11月10日

**Number**

前置条件： 无

功能：给Source.txt文件内出现的字符计数并输出至链表以及文件Output.txt中

后置条件：关闭输入输出流

输入：用户输入

输出：Output.txt

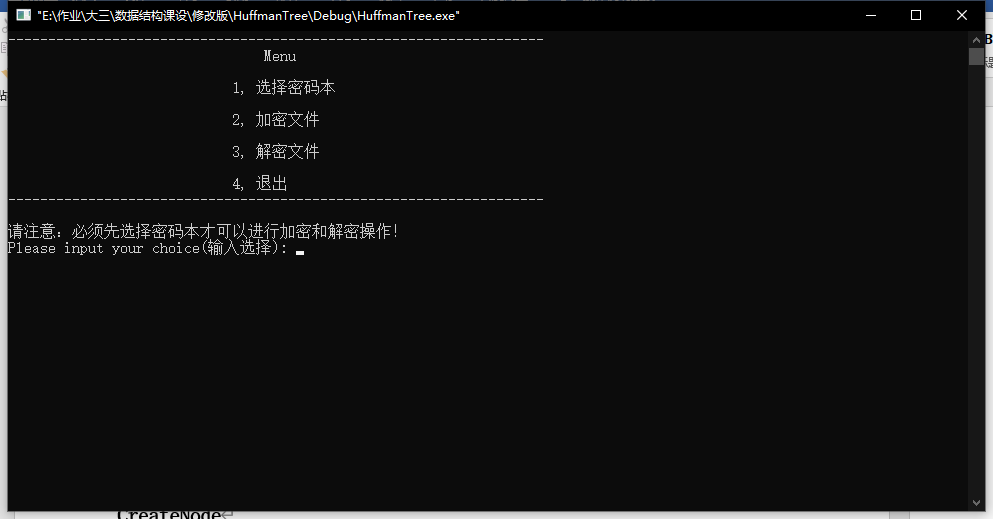
调用规则：无

编写者：梁沃辉

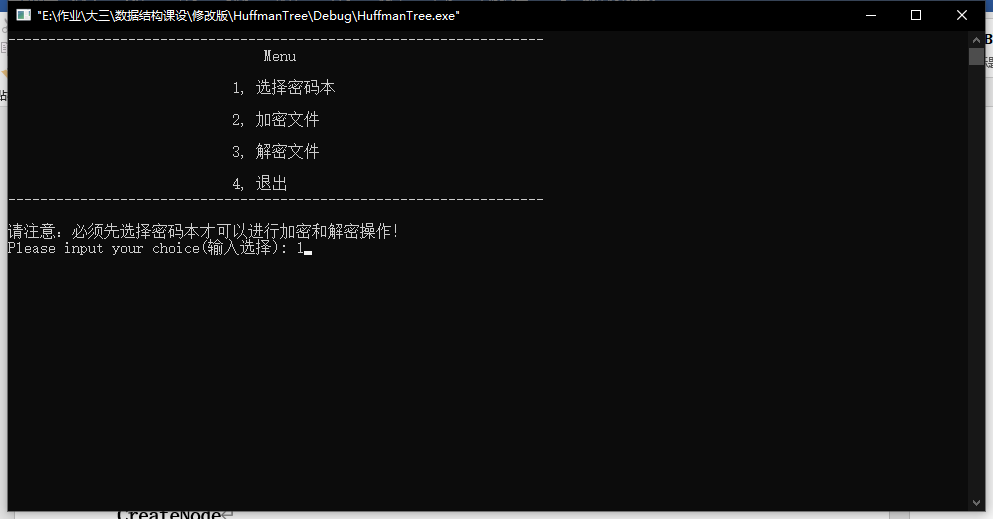
最后修改时间：2020年11月10日

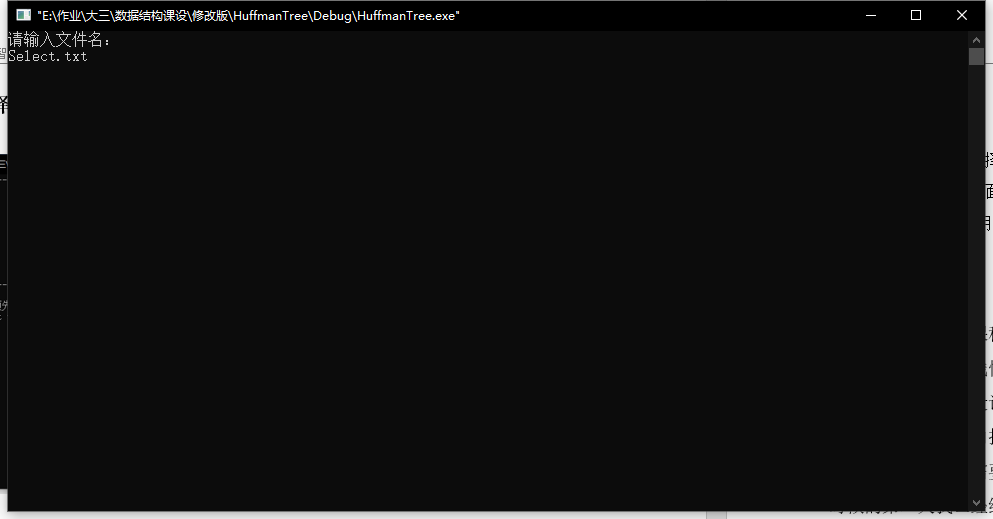
### 第4章 运行与测试

## 4.1 用户界面



## 4.2选择密码本（用户自定义密码本，密码本选择不同时）

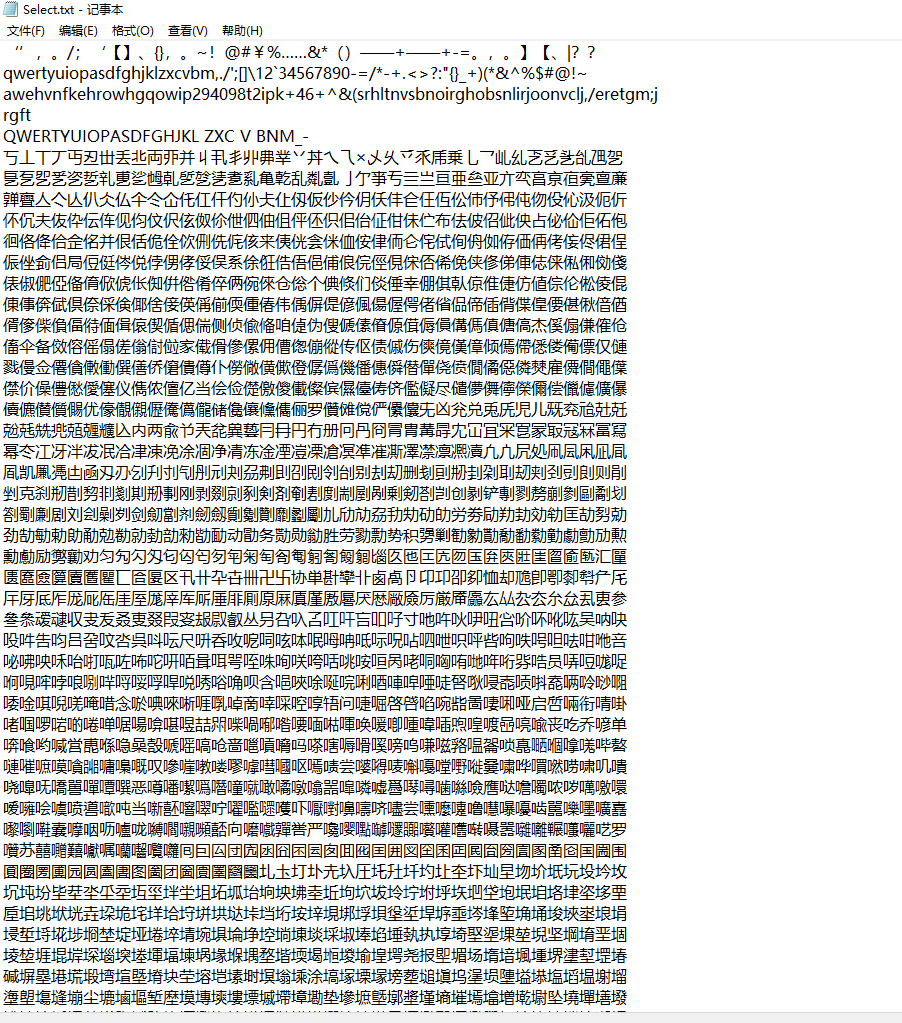




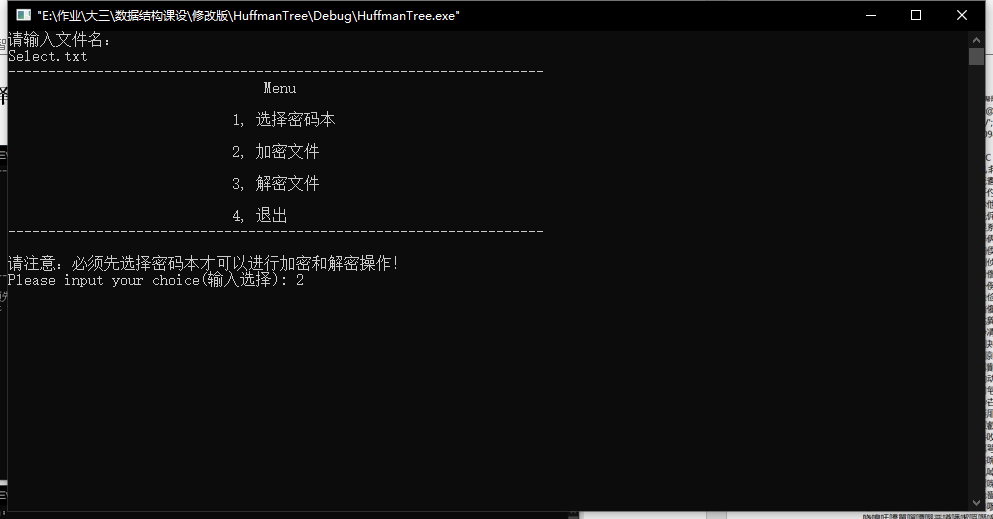
其中Select.txt有所有的英文字符，24000多个中文字符，以及所有标点符号，涵盖了所有常用字符。

当然，用户也可以自定义其中的密码本，只要双方约定使用一个密码本即可双方都加密以及解密成功。

只要输入密码本之后也就进入了三叉树中进行字符编码，其中编码后字符存在程序的链表中，提供加密以及解密依据。

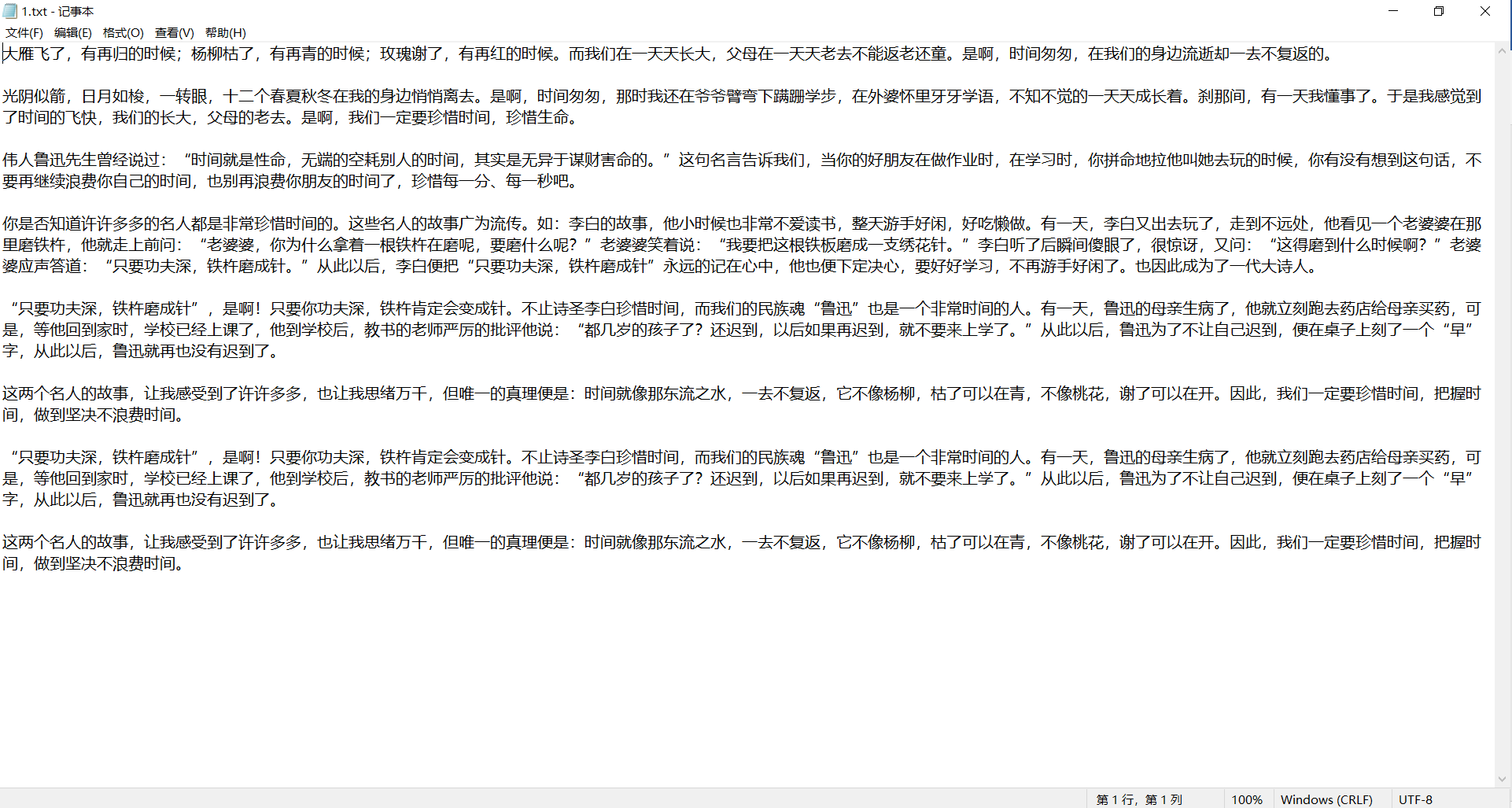


## 4.3 文件加密（需在选择密码本后进行）

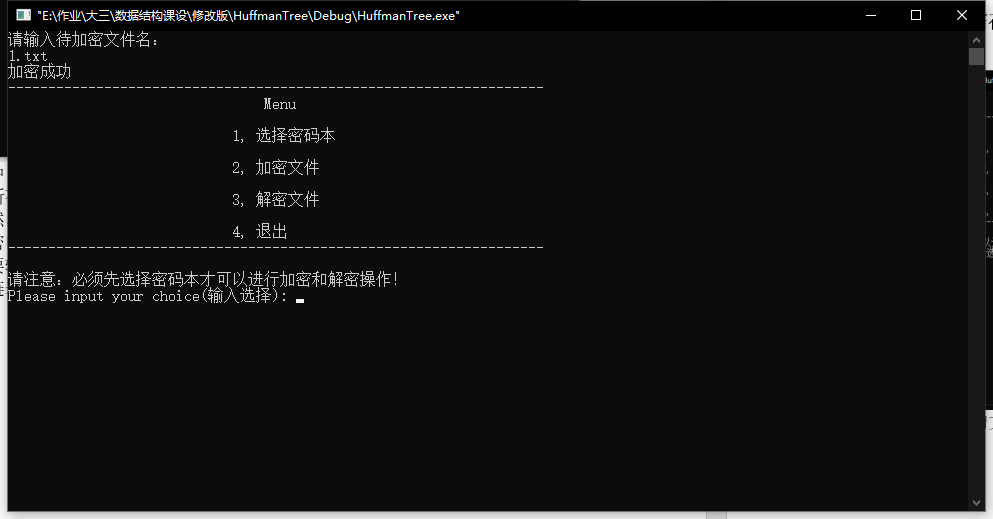


根据用户输入的文件名进行加密（下面是两个示例）

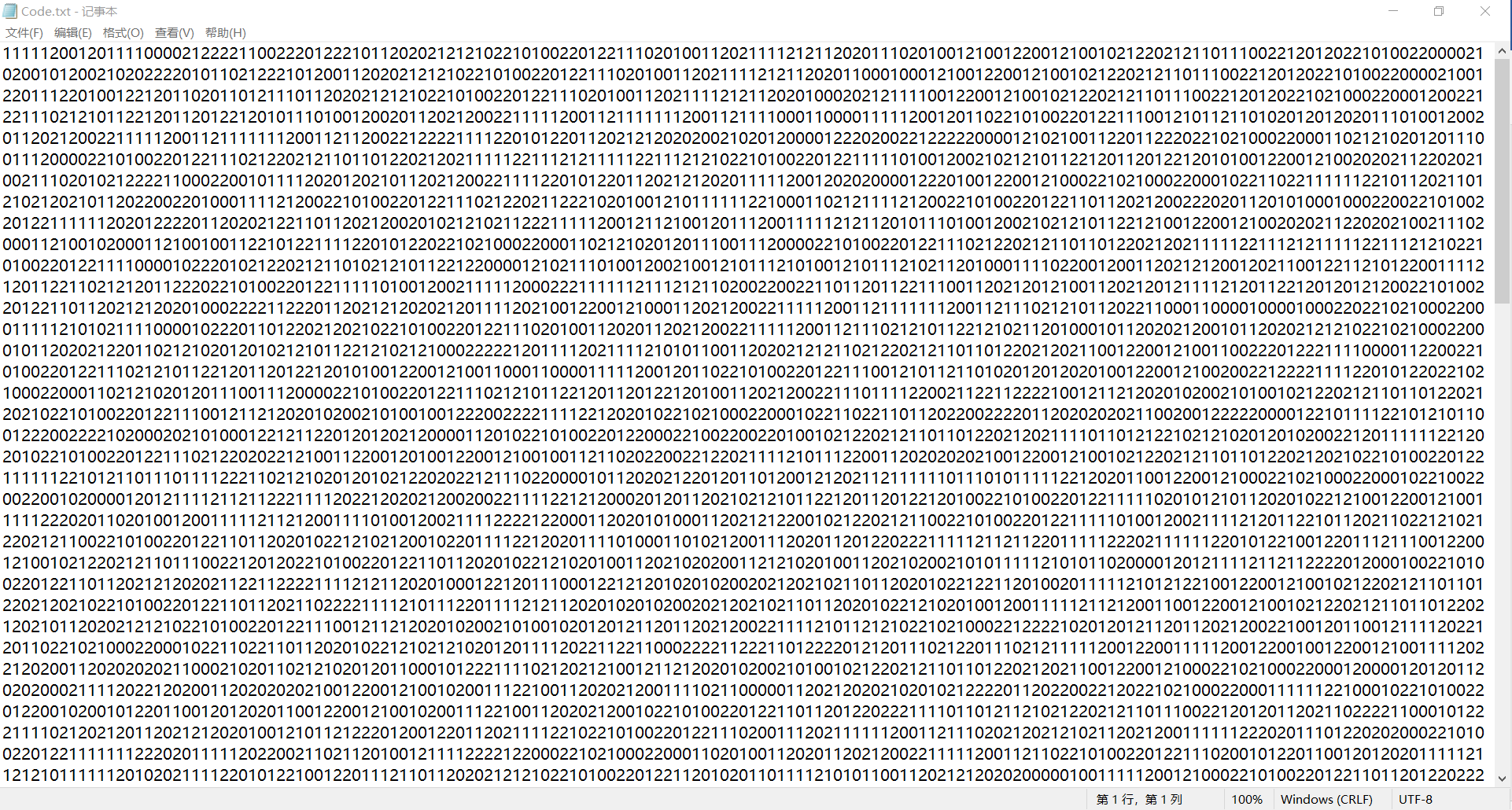
1. 1.txt中信息



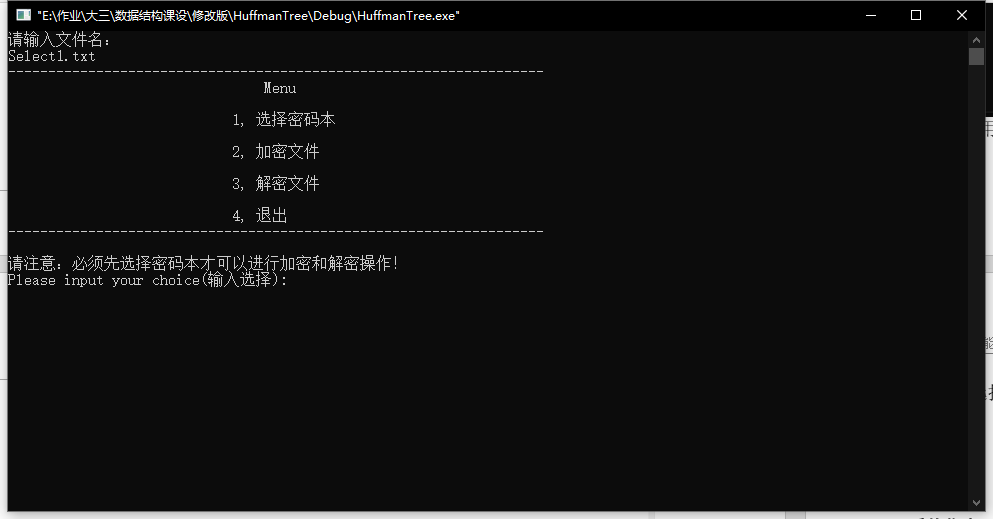
输入1.txt进行加密



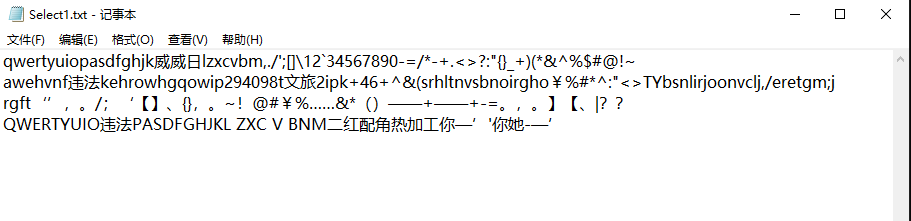
加密后的文件存储在根目录下的Code.txt中



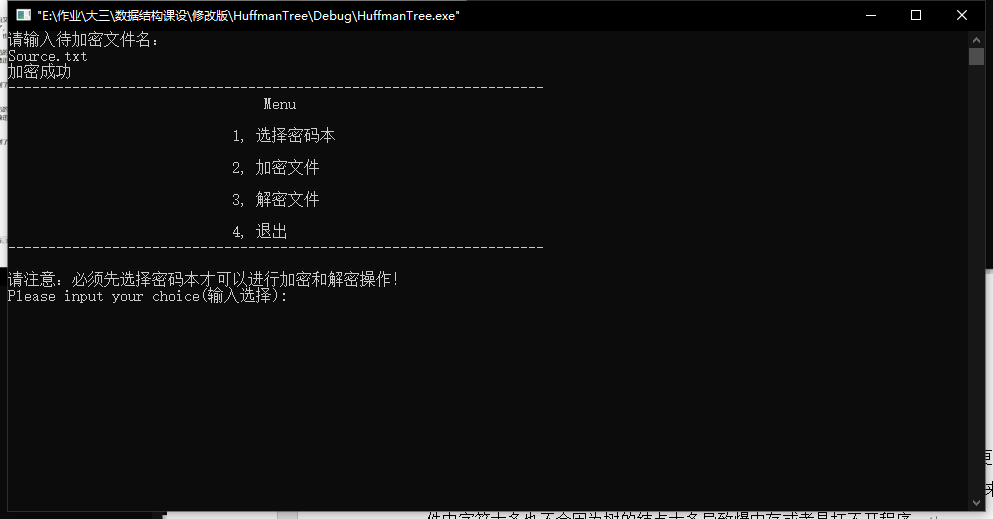
1. 选择另一密码本Select1.txt并进行另一个文件加密



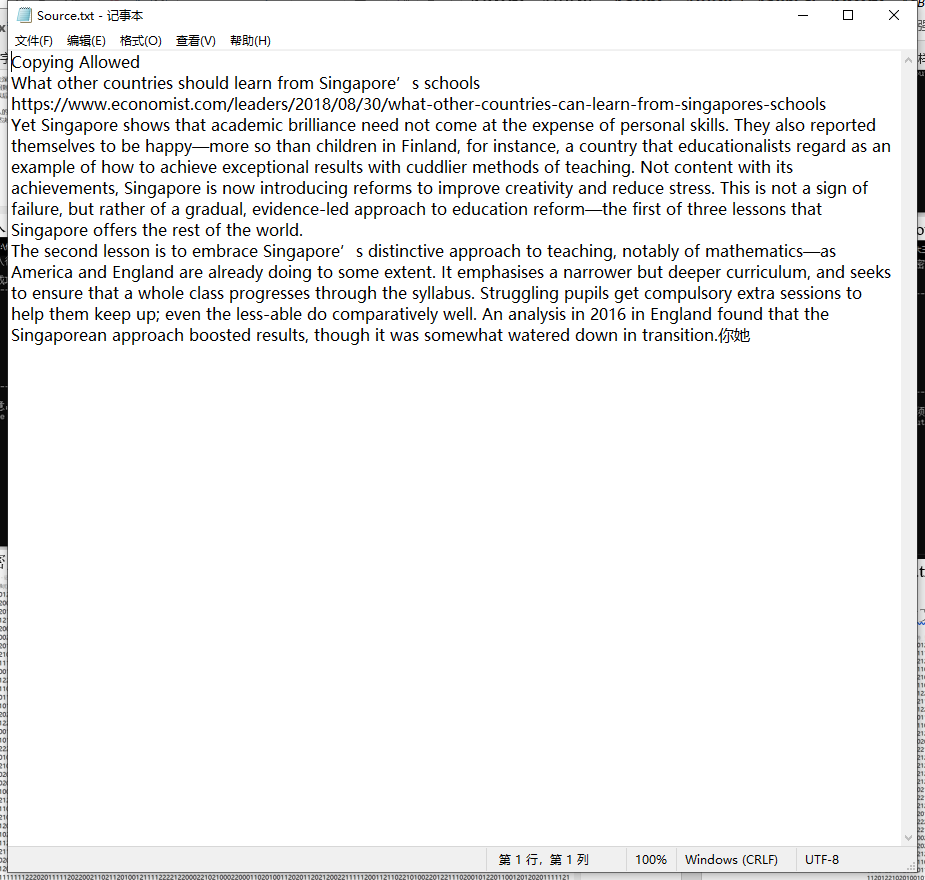
密码本Select1.txt



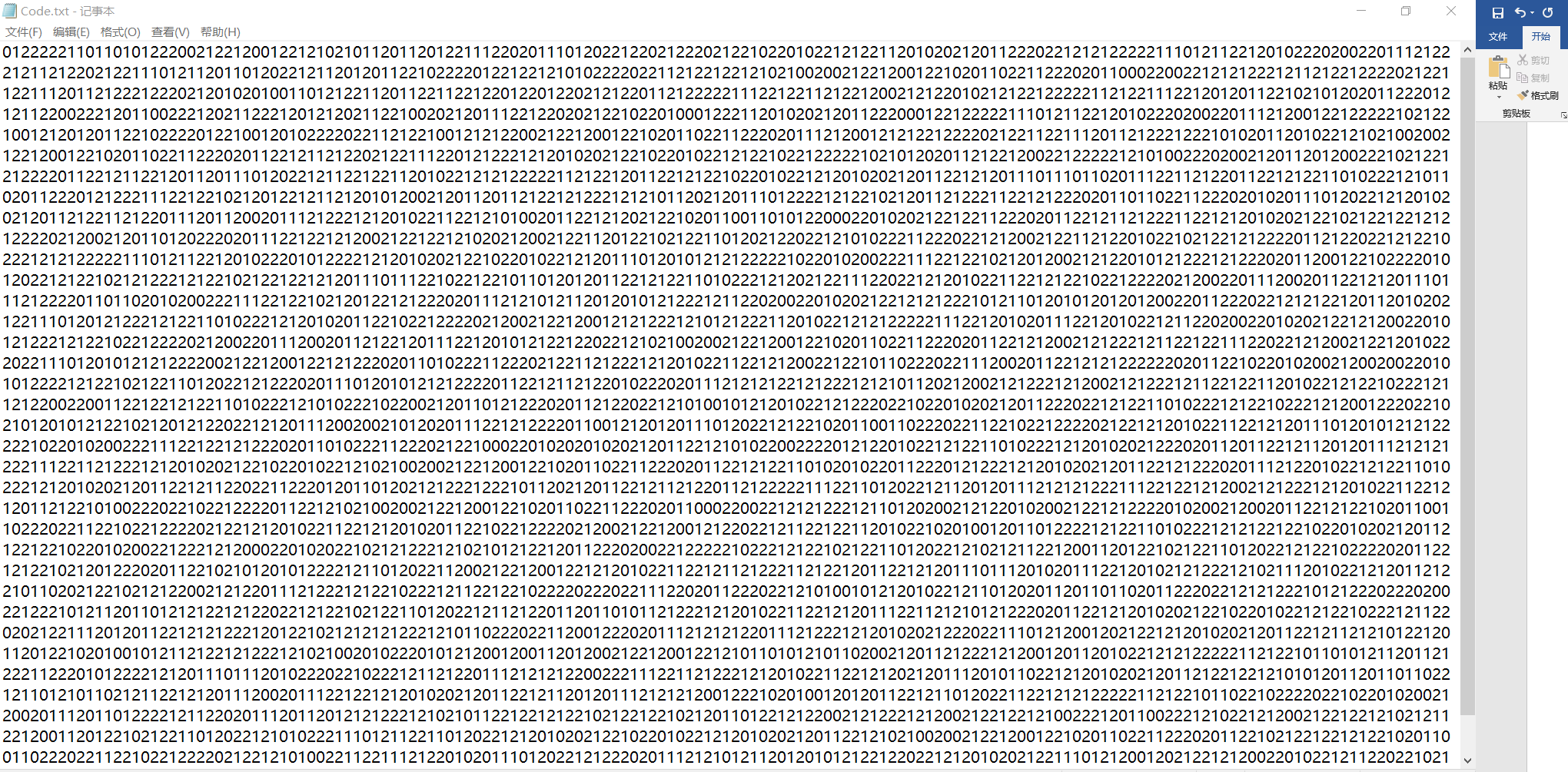
选择Source.txt进行加密



Source.txt内容



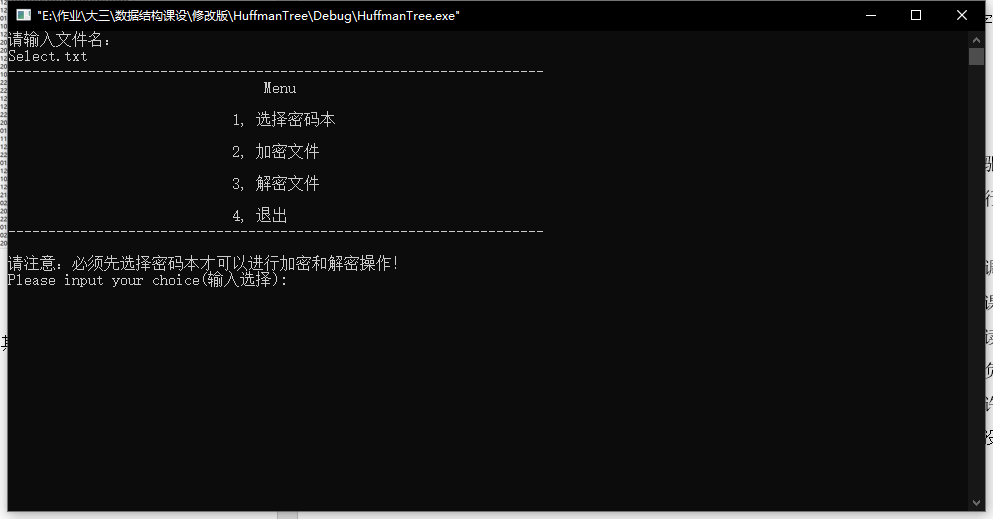
加密后文件同样存在Code.txt



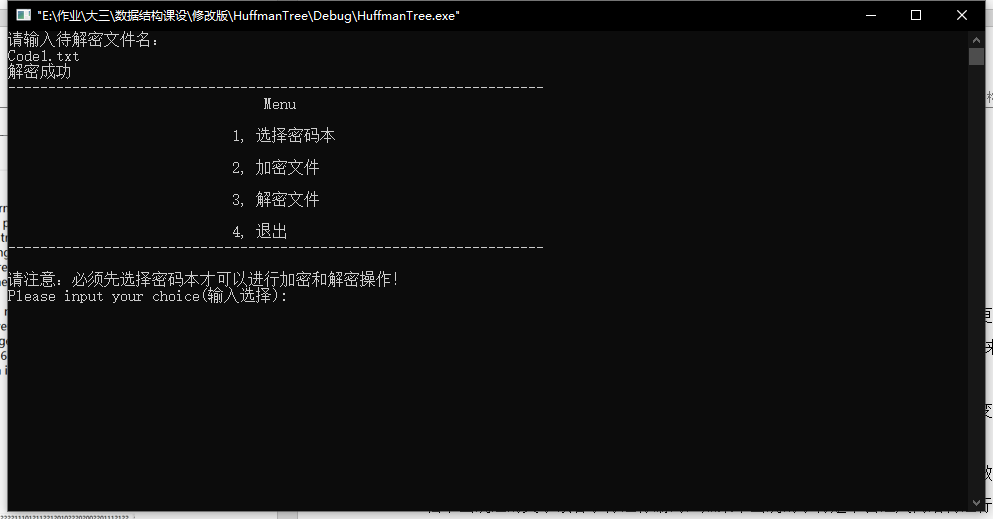
## 4.4 文件解密（需在选择密码本后进行）

1. 对Code1.txt进行解密（Code1.txt中内容为加密中①的内容，我将其复制到Code1.txt中作区分）

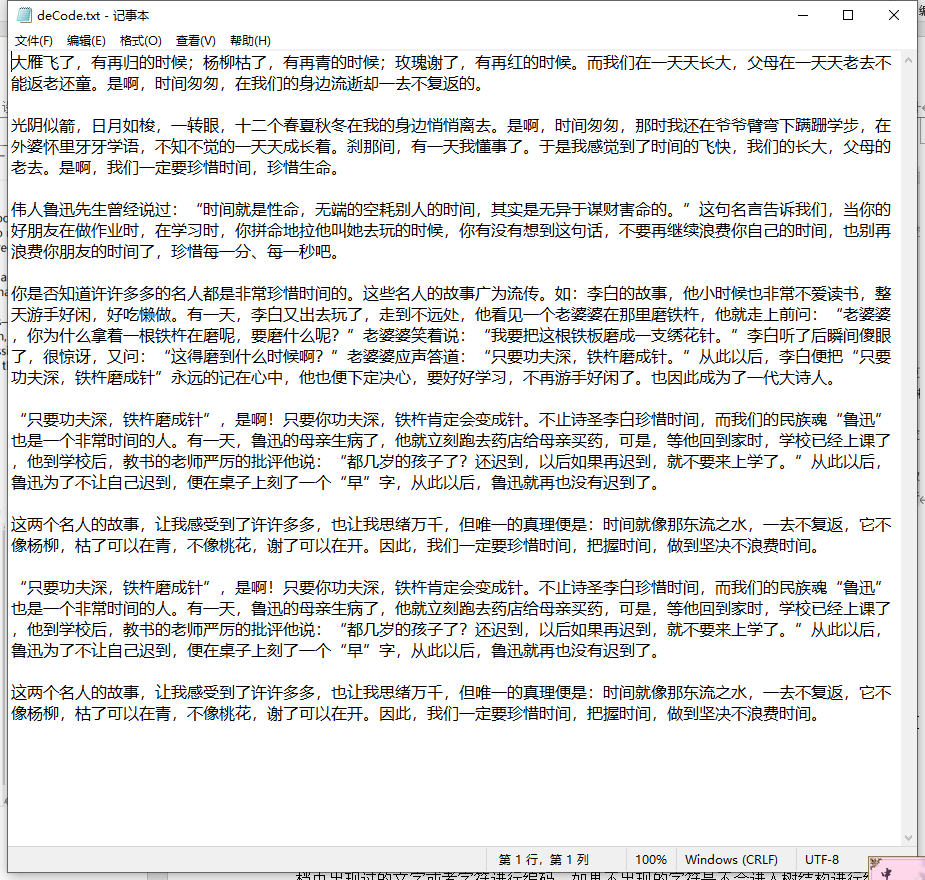
**选择密码本（选择与加密相同的密码本Select.txt）**



Code1.txt解密



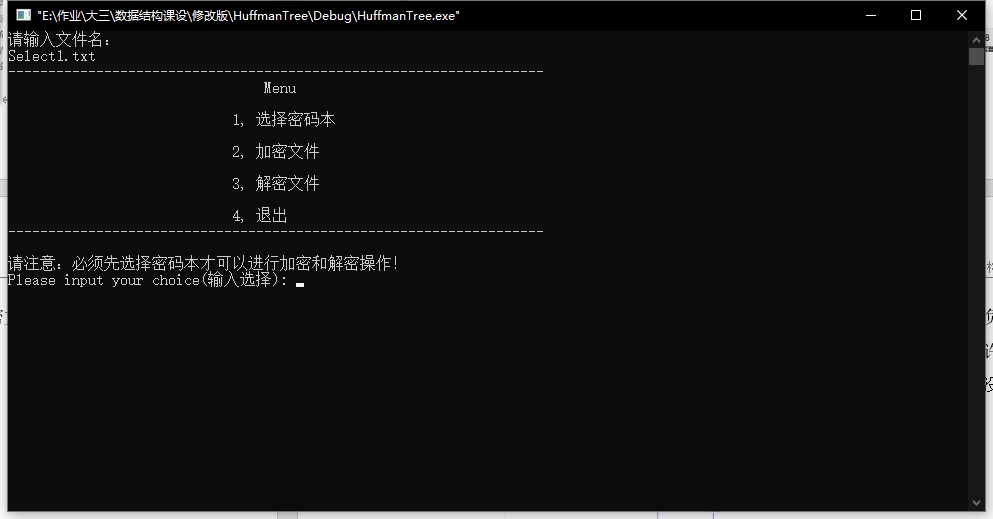
解密后文件放在deCode.txt中



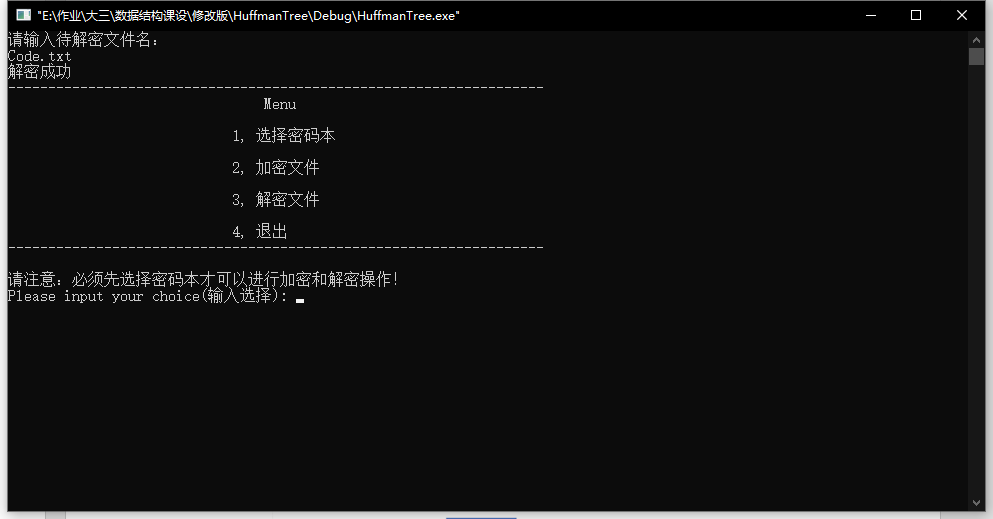
与加密中①的1.txt相同，说明解密成功

1. 对Code.txt进行解密（其中Code.txt中的文件是加密中②的密文）

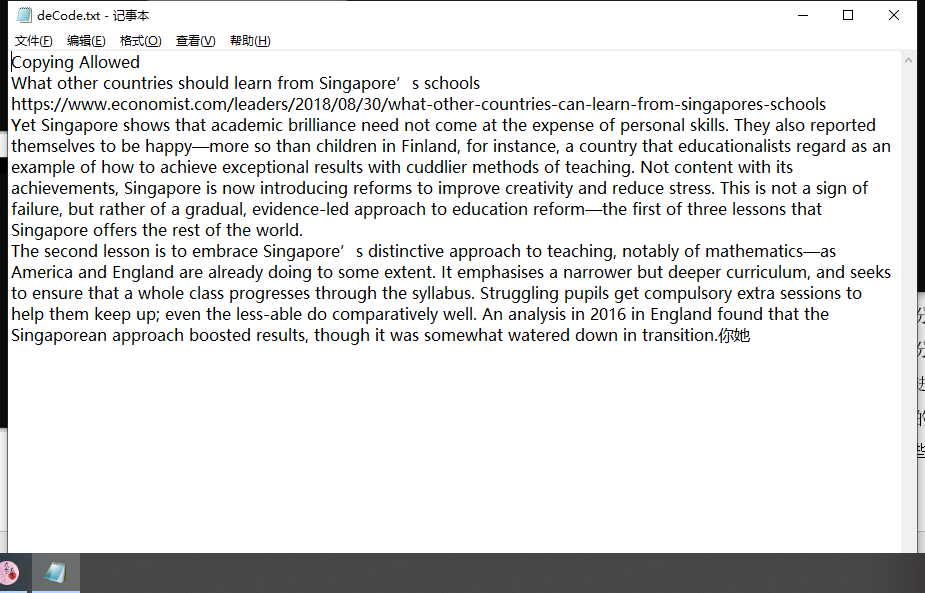
选择密码本Select1.txt



选择Code.txt进行解密



解密结果同样放在deCode.txt中



与加密中的②Sorce.txt相同，说明解密成功

## 总结

### 系统优点

1. 本系统中由于使用了三叉树，相较于二叉树更优的是三叉树在编码的时候可以更加节省空间，不单单是节省012串的空间，更加的是节省了开辟树的内存空间，这样一来即使文件内字符太多也不会因为树的结点太多导致爆内存或者是打不开程序。
2. 使用了三叉树的创新可以使得我们更好的了解树结构的形成以及其原理，即使变成更多分支的树的结构也可以按照原理一步一步形成更多分支的树结构。
3. 程序中允许用户自行选择密码本，而且对于密码本中的字符进行动态分配，即密码本有一个字符发生改变，加密后所有密文都会发生改变。

### 系统缺点

* 1. 由于中文字符进入程序中需要两个字符，而我们的程序是单字符进入，所以中文字符所需要的空间大大增加。
  2. 比对的过程是顺序查找，在解密的过程中如果密码本的字符过多导致字符编码过长时会，可以使用折半查找或者把012串编码再次进行编码的方法减少查找时间。

### 设计体会

本次数据结构课程设计中，我再次复习了数据结构中的二叉树结构（前驱，后继，根，叶子），线性表结构（链表结构），并且通过对二叉树的了解，进行创新，使得这次课程设计所使用的由二叉树改变为三叉树（森林结构）。

因为这次在组内担任的是组长，这也是第一次参与到这样的软件分工协调工作，还是很多东西需要注意的，首先第一点是工作的分配问题，所以在选好课题的时候的第二天我已经给队友准备好了这次课程设计的分工合作，我负责文件读取并获得字符以及其权值，林凯负责使用字符及其权值并进行编码工作，甘倩茵负责前端的设计以及解码工作（如下图），这是我第一次写的程序流程图，其中有许多不

成熟的地方，以及两个不同数据结构之间的传参问题，这些都是我第一次作为组长没有想到的东西，所以这是我还需要解决并且提高的地方。

第二，我说一下我们小组的想法问题，在一开始选题的时候我们的想法非常的多，首先我们想到的不是稳稳的做完一个编码，我们想到的是做一个图片的编码，因为我们想到每一个像素其实都是由RGB三种不同颜色组成，而这个数据我们也可以进行编码，对不同位置的像素RGB值进行编码得到这个像素的颜色值，但是一个问题是如果使用cmd命令行进行的话并不能做到彩色，所以我们又想到了二维码的编码问题，我们知道现在的二维码都带着非常多的应用信息或者说是网站信息，我们可以使用01串代表更多的信息，然后0代表一个单位的黑像素，1代表一个单位的白像素，然后用这样的黑白像素代表一个网站的全部信息，并可以超链接过去，但是想一想这好像又有点脱离题目。所以这次实验我们从算法入手，改变二叉树结构为三叉树，这样不仅能更深入的了解哈夫曼树的结构更加加深我们对其他进制编码的了解以及理解。

（11.30更新部分）

通过答辩与老师沟通之后我发现了我在开始设计的时候就已经存在了的问题，在设计的时候我是想着发送的时候像计算机网络一样发送密码本会随着密文一起发送，到对方之后解密会利用密码本进行比对再解码成正常的文章，但是我忽略了一点，如果密码本012串跟着发送的话别人一比对就能解密，所以我这个加密做的是没有效果的。一开始答辩的时候我还懵着，其实就是一开始就是错的设计思路。但是好在我们的功能都是逐个逐个功能去写就不需要去重写功能，只需要将我们的功能函数顺序进行对调，先密码本编码再对明文进行加密。有了这一次的教训之后我会更加懂得设计理念的重要性，一定要理解好题目的要求再进行功能设计，很感激在大学期间有一次这样的经历。

最后，这次课程设计让我懂得了团队协作的魅力，自己作为一个组长需要跟队员进行沟通以及交流，还有在一些时候需要跟队友有舍有得，不要盲目执着。这次课程设计也让我更多的了解到了数据结构的线性表以及哈夫曼树的结构，了解了编码解码的过程。

## 参考文献

[1] 王防修 多元哈夫曼编码在加密技术中的应用[J] 武汉工业学院 2009-03-01

[2] 数据结构实验指导手册（2017版）.pdf

## 附录