第八届“认证杯”数学中国

数学建模网络挑战赛

承　诺　书

我们仔细阅读了第八届“认证杯”数学中国数学建模网络挑战赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们接受相应处理结果。

我们允许数学中国网站(www.madio.net)公布论文，以供网友之间学习交流，数学中国网站以非商业目的的论文交流不需要提前取得我们的同意。

我们的参赛队号为：4510

参赛队员　(签名)　：

队员 1：范仁义

队员 2：李娅娅

队员 3：陈雪梅

参赛队教练员 (签名)：

参赛队伍组别：本科组

第八届“认证杯”数学中国

数学建模网络挑战赛
编 号 专 用 页

参赛队伍的参赛队号：4510

竞赛统一编号（由竞赛组委会送至评委团前编号）：

竞赛评阅编号（由竞赛评委团评阅前进行编号）：

2015 年第八届“认证杯”数学中国
数学建模网络挑战赛第一阶段论文

题　　目　破解带干扰密码

关 键 词　排除干扰 结构化处理 隐含马尔可夫模型 DES 算法

摘　　要：

密码学起源于数千年以前，长期以来出现了许多密码的编制方法，较为简单的是替换式密码。对拼音文字而言，最简单的形式是单字母替换加密，也就是以每个字母为一个单位，将每个字母替换成另外的字母或者另外的符号。基于对文献的查找和分析，我们最后要得到的明文是英文语料库中的词语，而第二阶段的题目中明确提出加密后，每个单词之间的空格，以及标点符号都会被删除，并且我们获取的密文是通过一个带有噪声干扰的信道传输的，在通信过程中存在增删改的情况，而我们需要改进第一阶段中破解密文的方法，使其能在带有干扰的条件下完成破译工作。那我们就要针对不同的情况，建立不同的模型，以便在不同情况下得到密文对应的明文。

在第二阶段，我们将分以下几种情况进行讨论：

1. 我们获取了一段通过噪声干扰信道传输的密文，通过定性和定量分析得到干扰不会影响密文中字母出现频率，在改进第一阶段算法的基础上我们对没有标点和空格的密文进行了结构化处理，依次采用字母频率分析、高频字母结构化处理、任意文章同步推演和结构化单词间隔划分，尽可能多的找到密文-明文对应关系，以便于实现被干扰密码的破解。
2. 我们获取了一段有干扰的密文，欲找到使密文被读成明文最大的条件概率。可利用贝叶斯公式并且省掉一个常数项，等价变换为破译的密文即为明文的可能性乘以明文为有意义信息的可能性，得到概率。做出两个假设：（1）明文字母为马尔科夫链，也就是说，后一个字母只由前一个字母决定。（2）不同时刻的接收信号只由发送信号决定。根据独立输出假设，可以得到 $(A_4^4)^{\text{密文长度}}$ 种组合，那么我们就可以很容易利用算法 Viterbi 找出上面式子的最大值，进而找出要识别的明文。
3. 我们获取了一段密文，在假设可以最大限度地排除干扰破解密文的情况下，首先初步判断它能不能被直接破译，如果能直接破译，说明在通信过程中没有受到干扰，此时我们将采用 DES 算法破译由古典密码衍生出的分组密码。

我们的每一种方法都配有相关实例，以便于推广对破译能力的评价指标。

参赛队号：4510

所选题目：B 题

参赛密码 _____

（由组委会填写）

Abstract

Cryptology originated thousands of years ago, for a long time the method for the preparation of many password, relatively simple is substitution cipher. For alphabetic writing. The simplest form is single letter substitution cipher, is with each letter as a unit, replacing each letter of the alphabet, with the addition of letters or other symbols. Based on the literature search and analysis, we finally get the plaintext is in a corpus of English words, and the title of the second stage clearly put forward the encryption and the spaces between every word and punctuation will be removed, and we obtain the ciphertext is through a with noise in the transmission channel, in the communication process of additions and deletions to the, and we need to improvement of the first stage of decryption, the decoding can be completed under the conditions of interference with the. Then we will be in accordance with different conditions, different models are established, in order to obtain thecorresponding plaintext ciphertext in different situations.

In the second stage, we will be divided into the following severalconditions are discussed:

1. We get a pass by noise interference channel transmission of the ciphertext, through qualitative and quantitative analysis of noise does not affect the ciphertext letter frequencies, in the improvement on the basis of the first phase of the algorithm we no punctuation and spaces of the ciphertext to structured, according to the times by letter frequency analysis, high frequency letters structured processing, arbitrary the synchronization wargame and structured word interval division, as much as possible find ciphertext and plaintext corresponding relationship, in order to interference of password cracking
2. we get a disturbance of the ciphertext, I want to find the ciphertextplaintext read as the condition of the maximum probability. But by the Bayesian formula and dispense with a constant term, equivalent transformation to decipher the ciphertext is multiplied by the possibility of plaintext expressly for the possibility of the significance of the information, to get the probability. Make two assumptions: (1) plaintext characters as a Markov chain. That is to say, after a letter only by a letter before the decision. (2) received signals of different time is only determined by thetransmitted signal. According to independent output hypothesis, we can get many combinations. Then we can easily use Viterbi algorithm to find the maximum value of the above formula, and then find out to identify the plaintext.
3. We obtain a ciphertext, on the assumption that can maximize to exclude the interference of decryption under, first determine the initial it cannot be directly to decipher. If we can decipher direct, indicating that in the process of communication without interference, then we will use the DES algorithm to decipher derived by classical cipher and block cipher.

Each method we are equipped with the relevant examples, in order to facilitate the promotion of decoding ability evaluation index.

Key Words: Eliminate the interference Structured treatment
Hidden Markov model DES algorithm

目录

一、问题提出

随着社会的发展，信息传播起着越来越重要的作用。现代通信朝着宽带化、智能化、综合化、个人化的全面发展，传播手段不断更新，但面临着一个不可避免的问题，即如何处理不完整，或者在通信传输的过程中，密文存在被丢失、添加和篡改字符的密码。在密码的实际使用中，我们获取的密文往往是经过各种干扰的。假设密文是通过一个带有噪声干扰的信道传输的，在加密后，单词之间的间隔和标点符号全部被删去。在通信过程中，每个字符经过信道传输的结果都属于如下四种情形之一：

1. 该字符在传输过程中被丢失，其概率为 p1；
2. 该字符本身正常传输，在其之后添加了一个随机字符，其概率为 p2；
3. 该字符在传输过程中被篡改为一个随机字符，其概率为 p3；
4. 该字符正常传输，其概率为 1-p1 -p2 -p3。

欲推广对破译能力的评价指标，通过完善之前的破译算法，使其能够在这种带有干扰的条件下完成破译工作。

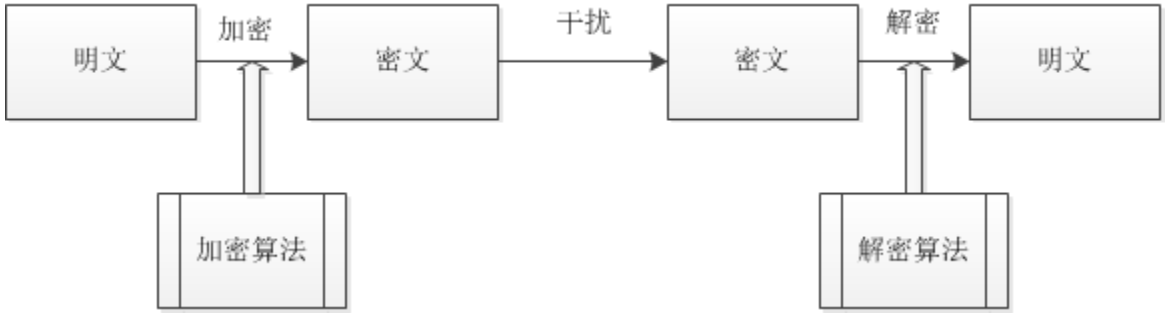

图 1

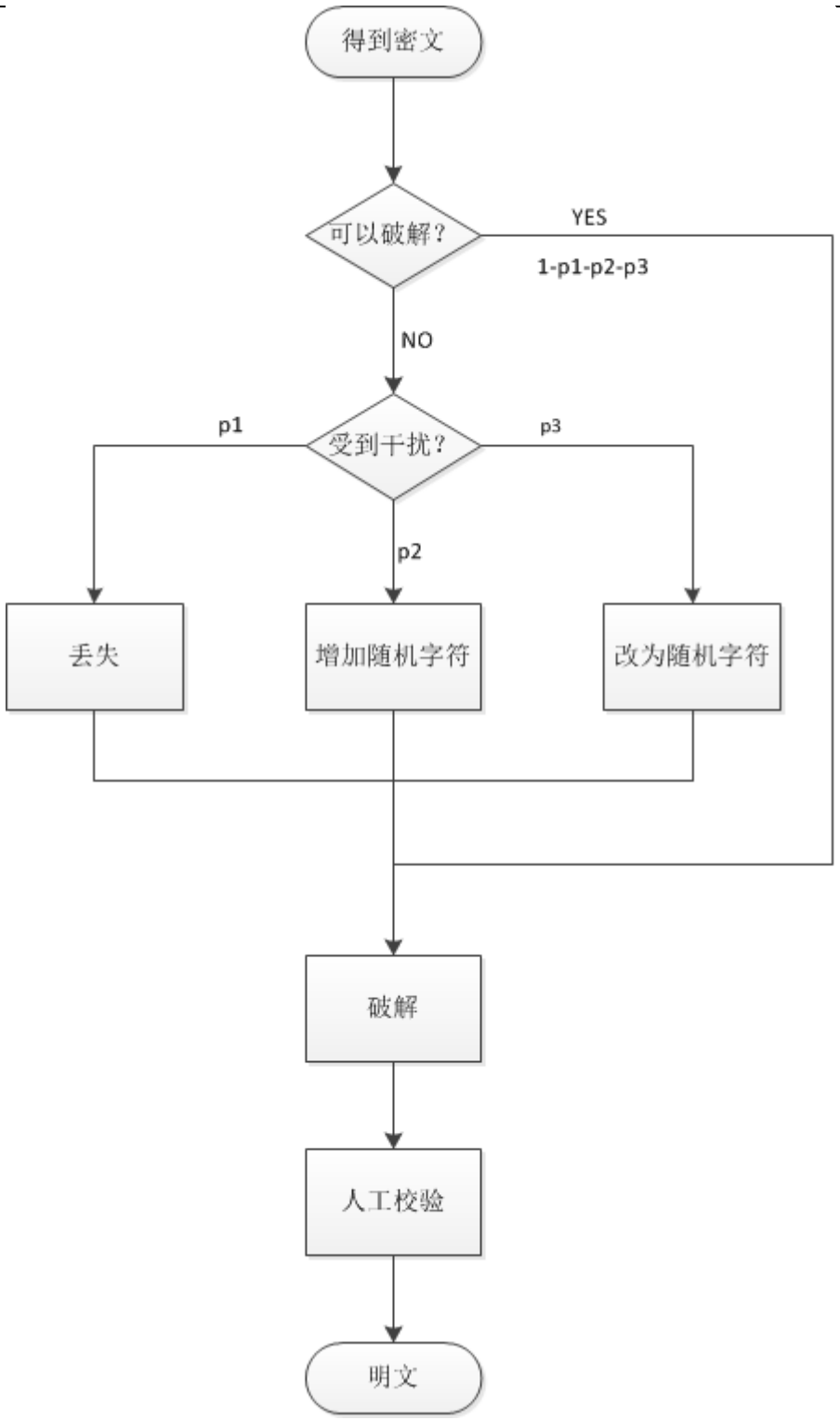
得到密文
可以破解？
YES
1-p1-p2-p3
NO
受到干扰？
p1
p2
p3
丢失
增加随机字符
改为随机字符
破解
人工校验
明文

图 2

二、问题分析

2.1 问题背景的理解

密码学起源于数千年以前，直到最近的几十年为止，这部分密码学被称为经典密码学，经典密码的加密方法主要是使用笔和纸，或者简单的机械辅助工具。历史上有许多密码的编制方法，较为简单的是替换式密码，也就是将文中出现的字符一对一地替换成其它的符号。单字母替换加密是在古代就使用过的一种加密方法，这种密码和破译方法在小说中也经常提到，例如爱伦•坡的《金甲虫》和柯南•道尔在福尔摩斯系列故事《归来记》中的“跳舞小人”。我们研究密码的破解是基于一些实用的情景，例如探测敌方的军情时需要截取敌方的情报、破解恐怖分子团伙制造袭击时的消息，找回日常生活中加密一些文件之后时隔太久忘记密码等一系列情况，进而将损失降到最小，而不是用破解密码的方法去做一些违法犯罪的活动。

2.2 问题的分析

对拼音文字而言，最简单的形式是单字母替换加密，也就是以每个字母为一个单位，将每个字母替换成另外的字母或者另外的符号。基于对文献的查找和分析，我们的最后要得到的明文是英文语料库中的词语，而我们需要对不完整，或者在通信传输的过程中，被丢失、添加和篡改字符的密码找到破解的方法，打个比方来说，明文就像锁在屋子里面的宝贝，密钥是打开屋子所需要的钥匙，而密文是外面的人，我们就是要从众多相似钥匙中做出判断，找到打开屋子真正的钥匙，拿到屋子里面的宝贝。那我们就要针对可能出现的钥匙，建立不同的模型，以便在不同情况下得到密文对应的明文。

在第二阶段，我们将分以下几种情况进行讨论：

1) 由于通信信道噪声干扰的存在，我们在获取了一段密文之后，首先通过定性和定量分析找出干扰（增删改）对字母统计的影响，分析过后得到干扰并不会影响字母统计，于是我们的模型就是在第一阶段的基础上进行改进。由于题目中明确提出加密后单词之间的空格和标点都会被删去，所以这个模型更注重于结构化，依次对密文进行：字母频率分析、高频字母的结构化处理、任意文章进行同步推演、结构化单词间隔划分、部分密码表试解密并循环推算，尽可能的找出更多的明密文对应关系，以便于最后破解带干扰的密文。
2) 我们获取了一段有干扰的密文，欲找到使密文被读成明文最大的条件概率。可利用贝叶斯公式并且省掉一个常数项，等价变换为破译的密文即为明文的可能性乘以明文为有意义信息的可能性，得到概率。做出两个假设：（1）明文字母为马尔科夫链，也就是说，后一个字母只由前一个字母决定。（2）不同时刻的接收信号只由发送信号决定。根据独立输出假设，可以得到 $(A_4^4)^{\text{密文长度}}$ 种组合，那么我们就可以很容易利用算法 Viterbi 找出上面式子的最大值，进而找出要识别的明文。
3) 我们获取了一段密文，在假设可以最大限度地排除干扰破解密文的情况下，首先初步判断它能不能被直接破译，如果能直接破译，说明在通信过程中没有受到干扰，此时我们将采用 DES 算法破译由古典密码衍生出的分组密码。

三、符号说明

Xi	密文中 a~z 的个数为
M	模型一中明文的长度
T	模型二中明文的长度
$\alpha_t(i)$	t 时刻状态为 i 的所有前面状态路径的最大概率

表 1

四、模型假设

1. 由于英文语料库中包含大量信息，我们建立的模型所定义的明文只是语料库里面的一部分信息；
2. 我们所选取的语料库中的信息有一定的代表性，在模型的推广中能起作用；
3. 能最大限度地排除干扰破解密文得到合理化的明文；
4. 索引所花时间在控制范围和接受范围内；
5. 明文、密文一一对应；
6. 运算足够精准，算法能正确运行；
7. 分析足够仔细；

五、模型的建立与求解

5.1 破解无标点无间隔的密文——结构化处理

5.1.1 模型的建立

此模型是在分析了第二阶段题目的要求之后建立的。由于通信信道噪声干扰的存在，我们在获取了一段密文之后，首先通过定性和定量分析找出干扰（增删改）对字母统计的影响，分析过后得到干扰并不会影响字母统计，于是我们的模型就是在第一阶段的基础上进行改进。由于题目中明确提出加密后单词之间的空格和标点都会被删去，所以这个模型更注重于结构化，依次对密文进行：字母频率分析、高频字母的结构化处理、任意文章进行同步推演、结构化单词间隔划分、部分密码表试解密并循环推算，尽可能的找出更多的明密文对应关系，以便于最后破解带干扰的密文。

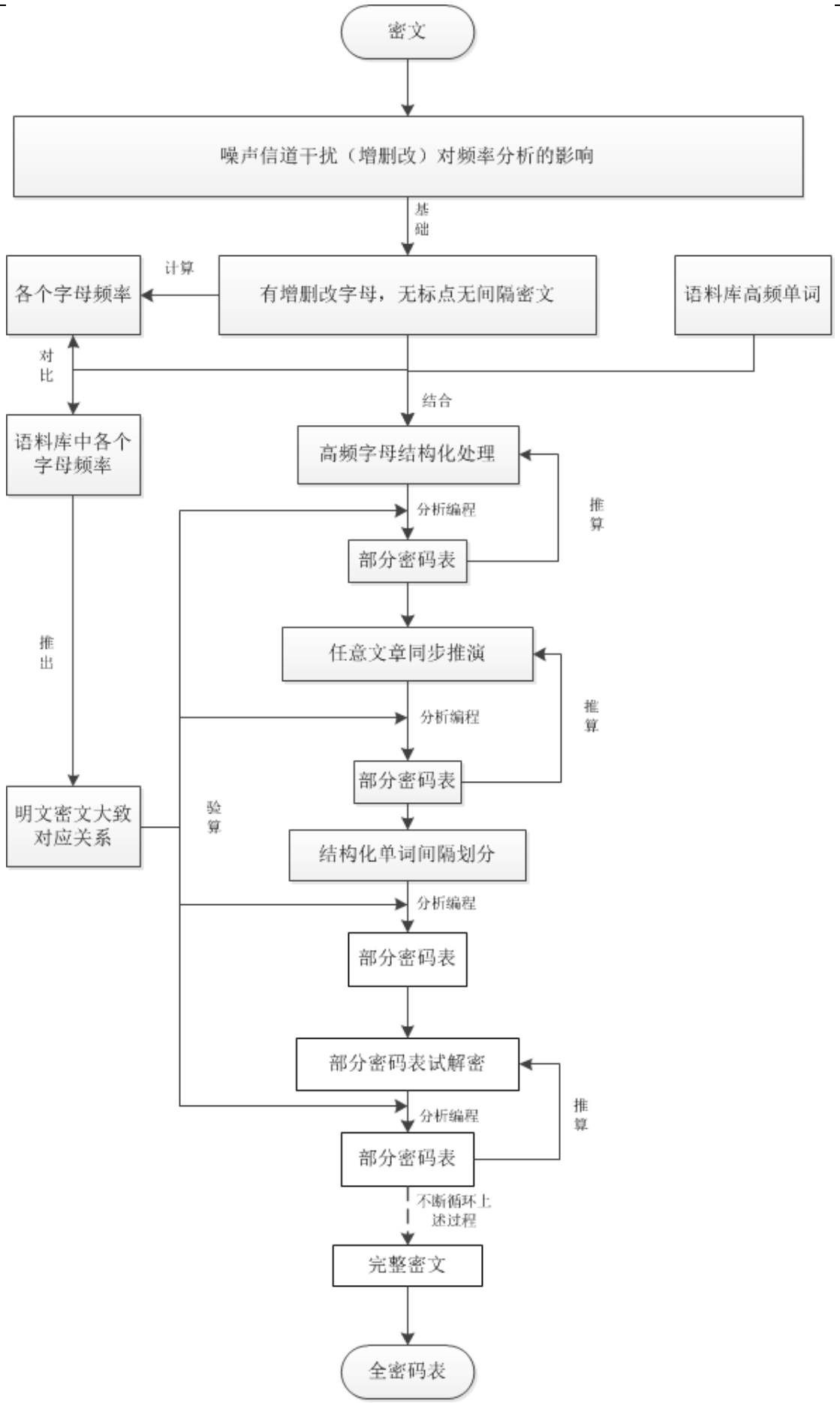
密文
噪声信道干扰（增删改）对频率分析的影响
基础
有增删改字母，无标点无间隔密文
计算
各个字母频率
语料库高频单词
对比
结合
语料库中各个字母频率
高频字母结构化处理
分析编程
推算
部分密码表
任意文章同步推演
推出
分析编程
推算
部分密码表
明文密文大致对应关系
验算
结构化单词间隔划分
分析编程
部分密码表
部分密码表试解密
分析编程
推算
部分密码表
不断循环上述过程
完整密文
全密码表

图 3

5.1.2 模型的实例

我们现在获取了一段是经过带有噪声干扰信道传输的密文（即密文中的字母有被增删改的情况），而且密文本身没有带任何标点和空格，密文如下：

HVTFOFZHLADSRROAERFPAFHIRLVPRFARFKROKVYDLFMMFLEVHHLADSFNSMDJRKARLRMRIEFKROF
PREDLFSOERFPDKERUREHKVFHKEVSXREFSOPVERYEVHYREVNHLASFFSOKARFLLVPYMDHAPRSKVUK
ARTRMMIRDSXUVEKARLADSRHRYRJYMRFHUFEFHDFPLVSLRESROKARYNEHNDKUVELADSRHROERFPL
VPIDSRHIVKAVNESFKDVSHQDHDVSUVEKARUNKNERFSODSSODQDONFMHOERFPTARSDKLVPRHKVSAD
SFTRFERYVDHROKVINDMODKDSKVFPVORESDCROLVNSKEZUEVPFORQRMVYDSXLVNSKEZKAEVNXAVN
KKARPVORESADHKVEZXNEYRVYMRFSOLVNSKEZAFQRHNUURERQUEVPFHREDRHVUPDHREDRHHHVKAF
KTRAFQRUNMERFHVSKVXEVTYVTREUPNMKVERHKVERVNEXMVEZDSKARFSLDRSKKDPRHFHUVEKARDS
ODQDONFMFHYRLKKARLNEERSKKFHJUVEPRDHKVHKNOZAFEOFSOPFHKREFMMHVEKHVUJSVTMROXRT
ADLATDMMIRYMFZRODKHEVMRLSKARPVORESDCOLVSHKENLKDVSDIRMDRQRFHHKEVSXMZFHRQREVN
ELVNSKEZFSOYRVYMRTDMMIREDHDSXVSMZTARSTRFHFDSODQDONFMOUVNENKPVHKKVORODLFKRKA
RRSREXZFHPNLAFHTRLFSHVKAFKDSPZNSOREHKFSIODSXVUKARLADSRHROERFPIZAFEOTVEJFSOE
RMRSKMRHHORKREPDSFKDVSFSOYREHDHKKADHOERFPTDMMIRIVNSOKVLVPRKEN

图 4

我们现在需要对这段无标点无间隔且有错误的密文进行解密，使它变成我们人能轻易读懂的通俗语言。

我们的算法主要分为 5 个大的步骤：

1. 字母频率分析
2. 高频字母的结构化处理
3. 任意文章进行同步推演
4. 结构化单词间隔划分

5. 部分密码表试解密

其中 1,2,3,4,5 步是不断循环的，以求出完整的密码表，直至最后获取完整的密码表。

在通信过程中，由于传输信道存在噪声干扰，我们获取的密文并不一定就是原来的明文直接对应的密文，而有可能是增加、删除和被篡改过后的不完整的密文。基于第一阶段的研究，我们尝试改进之前的破译算法，以便于在有干扰的情况下也能破解密码。下面我们将通过定性定量分析讨论字符的丢失、增加和被篡改对频率分析的影响。

1. 丢失

我们获取了一段密文，其对应的明文的长度是M，而题目中给出字符在传输过程中被丢失的概率为p1，所以理论上出现字符丢失之后密文的长度为M(1-p1)。

经过编程统计，得到密文中a~z的个数为Xi（1≤i≤26），如下图所示。

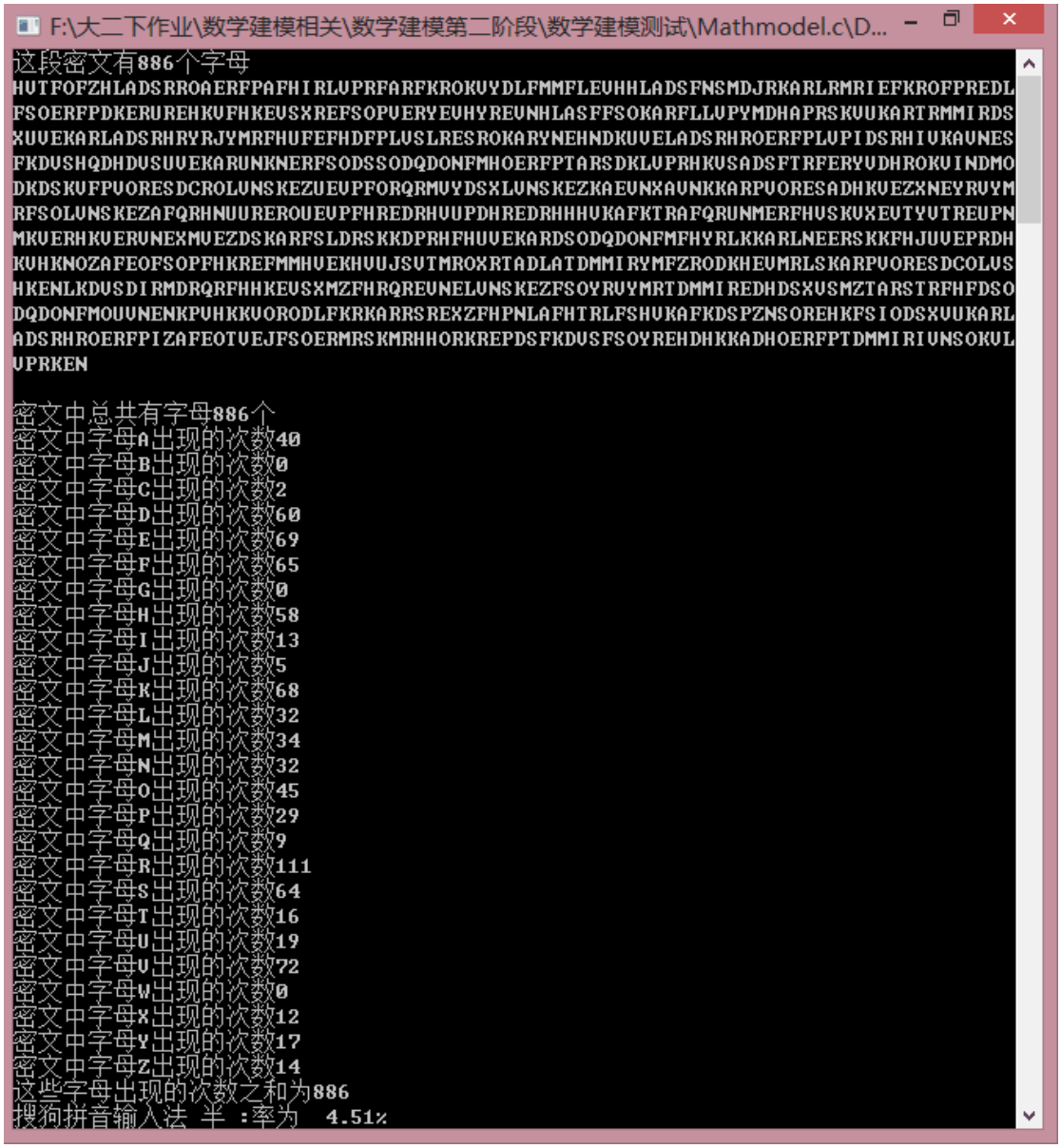

图5

每一个字符被丢失的可能性都为p1，由于X_i不尽相同，所以每个字符被丢失的个数会有所不同,但总体上来说，$\sum_{i=1}^{26} Xi * p1 = M * p1$是成立的。

将统计得到的每个字符的数目放到下面的表中，再结合表中的数据分析字符丢失对

字母统计的影响。

字符丢失情况分析表						
出现字符	a	b	c	……	y	z
字符数统计	40	0	2	……	17	14
被丢失的可能	P1					
被丢失的个数	X_{1*} P1	X_{2*} P1	X_{3*} P1	……	X_{25*} P1	X_{26*} P1

表2

如果明文加密后通过可靠信道传输没有被干扰，每个字符出现的概率为$Pi=\frac{Xi}{M}$，而对于存在丢失字符的情况，每个字符出现的概率为$Pi=\frac{Xi(1-p1)}{M(1-p1)}=\frac{Xi}{M}$，和没有出现丢失字符的时候的概率一样，说明在传输过程中出现字符丢失并不会对字母统计产生特别大的影响，我们可以使用频率分析来破解密码。

2. 增加

和存在丢失的情形类似，我们获取的密文对应的明文的长度也是M，此时题目中给出的在传输过程中存在增加字符的概率为p2，相对于明文，密文中增加的字符的个数理论上为M*p2，而26个字母中每一个都可能是增加到密文中的那一个字符，所以此时密文中a~z的个数为：$Xi+M*p2*\frac{1}{26}$。

增加随机字符情况						
出现字符	a	b	c	……	z	字符总数
增加前字符数	40	0	2	……	14	886
增加字符概率	P2					
增加字符数	$M*p2*\frac{1}{26}$	$M*p2*\frac{1}{26}$	$M*p2*\frac{1}{26}$		$M*p2*\frac{1}{26}$	$M(1+p2)$

表3

如果没有增加的字符，密文中每个字符出现的概率为$Pi=\frac{Xi}{M}$，而存在增加字符的情况后，每个字符出现的概率为$PI=\frac{Xi+M*p2*\frac{1}{26}}{M(1+p2)}\neq\frac{Xi}{M}$，此时存在两种情况：（1）p2比较小时，Pi的分子分母都只有很小的变化，总体上来说，各个字符出现的概率不断趋近，但每个字符出现的频率变化趋势是不变的；（2）p2比较大时，对于某些出现频率比较低的字符，增加的个数$M*p2*\frac{1}{26}$可能比它们增加前的XI占有更大的权重，此时Pi就会明显地增大，整个密文中出现的字符的频率有一种接近的趋势，但不会超过原本出现频率就很高的字符。上面这两种情况对字母统计都没有太大的影响，可以用频率分析来尝试破解密码。

可能有人会说上述的两种情况只是理论条件下成立，在实际进行密码破译的时候，

很难得到某一个字符刚好增加了$M*p2*\frac{1}{26}$次。就像抛硬币时在抛的次数不是特别多的时候，正反面各占一半的情形是很少出现的。在这种情况下，由于偏差的存在，使得某些原本出现频率比较低的字符，在通信干扰后出现的次数超过了原本出现频率比较高的字符，不再符合语料库中各个字符出现的频率分布规律，这个时候就不能再使用频率分析来进行密码破解。但更大的问题是我们为了分析在这一阶段题目的要求下，能不能对字母进行统计进而用频率分析来破译密码。而频率分析正好是适用于密文规模庞大的时候，在密文很长的时候，得到一个字符增加了$M*p2*\frac{1}{26}$次是完全有可能的，进而也验证了存在随机字符增加的时候字母统计仍然可以使用。

3. 修改

对于一般通信过程来说，总是会有数据的传输，而数据传输经过不可信的通道时可能会受到各种干扰，除了上面所说的丢失和增加，还有密文中字符被篡改的可能。而字符被篡改可理解为每个字符丢失后再增加。不妨假设明文为S1S2S3……，我们截取到被篡改的密文为010203……此时，我们可建立隐含马尔可夫模型。这是因为：首先s1,s2,s3,...是一个马尔可夫链，也就是说，si只由si-1决定。第二，第i时刻的接收信号oi只由发送信号si决定（又称为独立输出假设,即
P(o1,o2,o3,...|s1,s2,s3....)=P(o1|s1)*P(o2|s2)*P(o3|s3)），因此我们可以得出对应位置一个接一个篡改的概率与密文篡改位置的字母先丢失再增加的概率是相同的。而我们之前已经得到密文被丢失以及在字符后增加一个随机字符，对字母频率基本上无影响，误差在可行性范围之内。故字符在篡改前后依然适用于频率分析法。

所以，在尽量减少偏差的情况下，字母统计对于字符的丢失、增加和修改都是适用的，我们可以继续使用频率分析进行密码破译。

具体过程如下：

1.字母频率分析

1.1 编程求密文中各字母频率

编程求出我们所获取的密文中的各个字母出现的频率，结果如下：

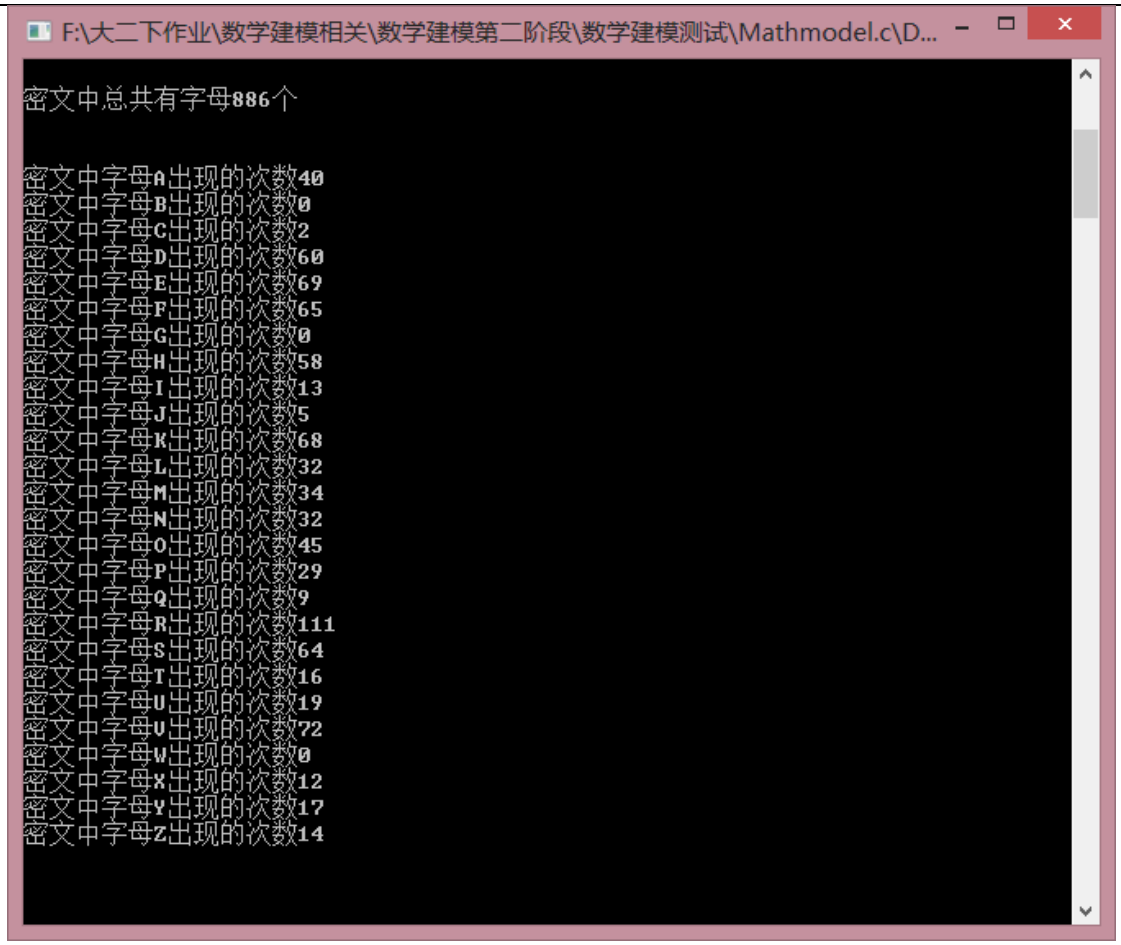

图 6

各个字母在密文中出现的概率如下：

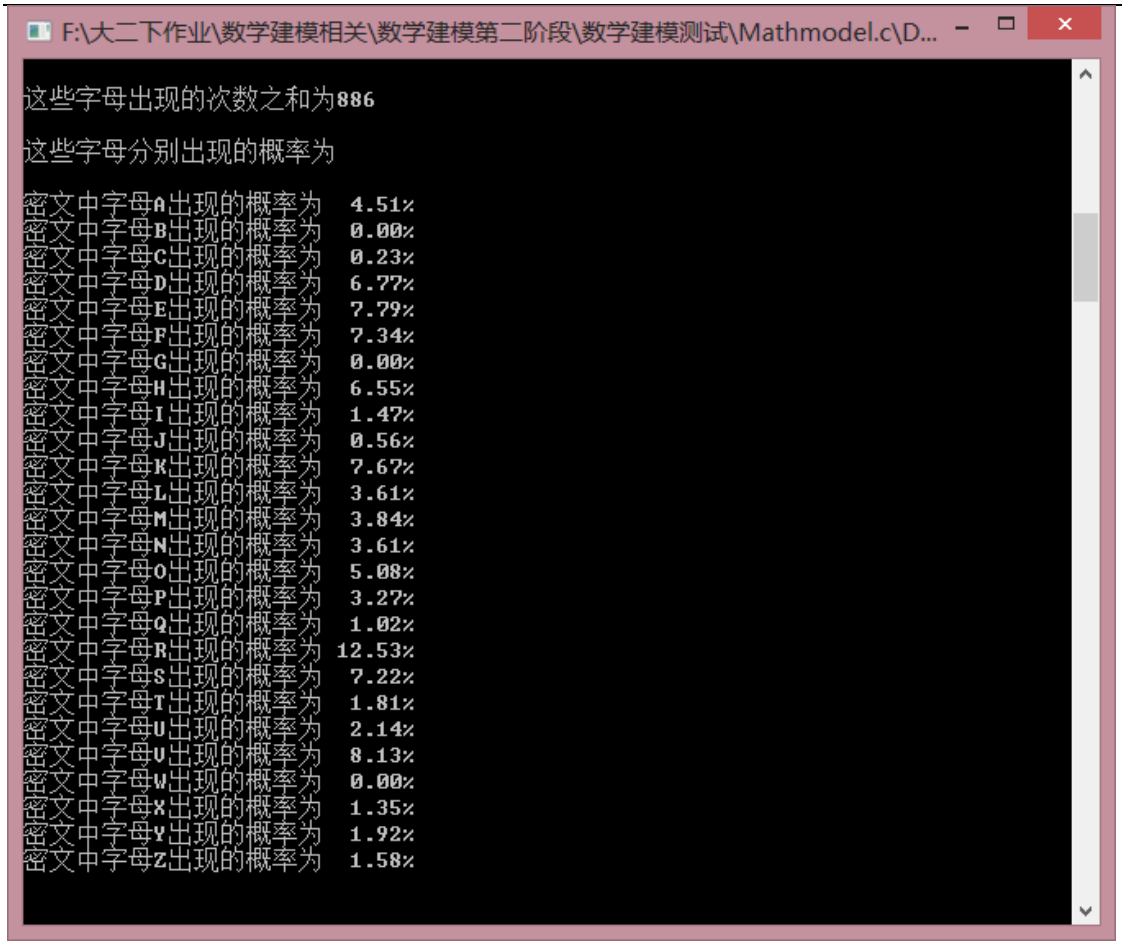

图 7

我们可以发现密文中出现频率最高的字母为 R(12.53%)、V(8.13%)、E(7.79%)、K(7.67%)、F(7.34%)、S（7.22%)、D(6.77%)、H(6.55%)、O(5.08%)、A(4.51%)
密文中出现频率较低的单词为 J(0.56%)、C(0.23%)、G(0.00%)、W(0.00%)、B(0.00%)

1.2 信道干扰传输对密文各字母频率的影响

我们在上面已经证明了字母的增删改对密文字母频率几乎是没有影响的：
丢失字母的情况，无论丢失字母概率大小，都对密文字母频率几乎无影响。
增加字母的情况，各字母概率有接近的趋势，增加字母概率越大，各个字母接近趋势越明显，但是概率高的依然是概率高的，都对密文字母频率几乎无影响，当获取密文比较少的情况，理论分析向具体随机情况转换的时候是有一点误差的。
修改字母的情况，修改字母可以看成是先丢失后增加，具体影响相当去丢失和增加的中间值，所以修改密文字母也是对密文字母频率几乎无影响。

1.3 密文字母出现频率与标准语料库字母出现频率做对比分析

查阅英语语料库资料可发现英语中出现频率最高的 12 个字母为“ETAOIN SHRDLU”。出现频率最低的字母为“JKQXZ”。

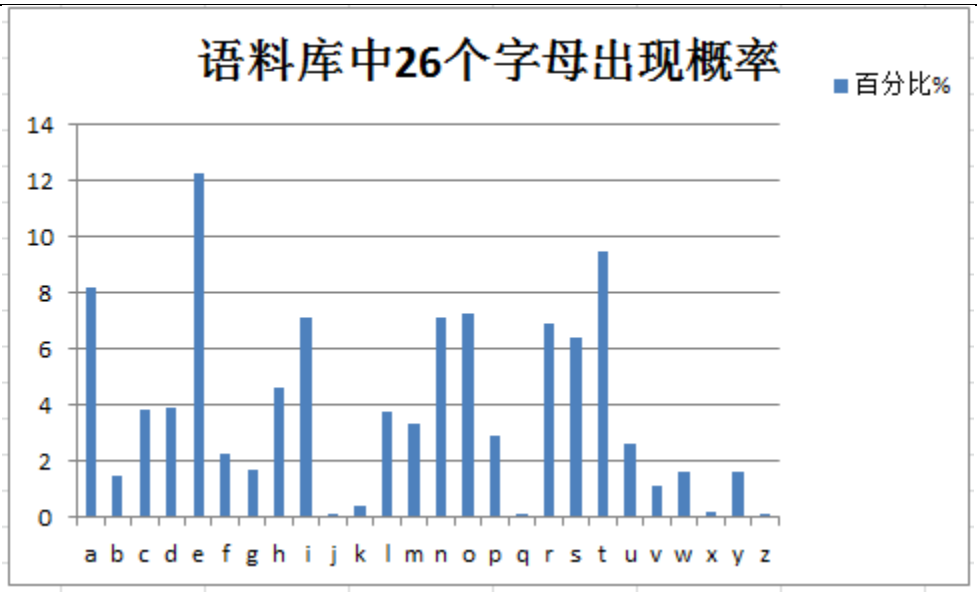

图 8

密文中各字母出现的频率如下：

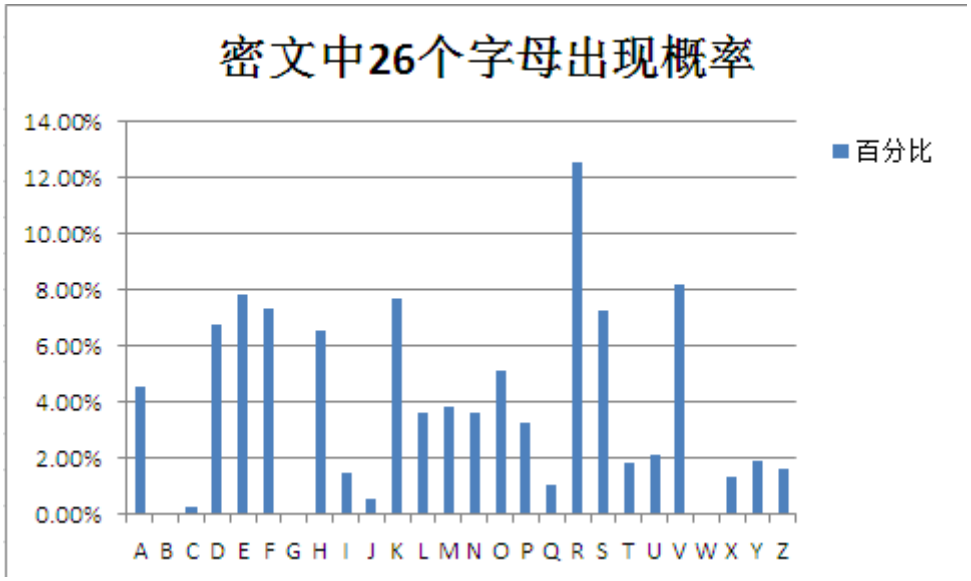

图 9

我们可以发现密文中出现频率最高的字母为 R(12.53%)、V(8.13%)、E(7.79%)、K(7.67%)、F(7.34%)、S（7.22%)、D(6.77%)、H(6.55%)、O(5.08%)、A(4.51%)

密文中出现频率较低的单词为 J(0.56%)、C(0.23%)、G(0.00%)、W(0.00%)、B(0.00%)

故密文和明文的字母间存在某种对应关系：

	出现频率比较高的字母	出现频率比较低的字母
密文字母（密文统计）	R，V，E，K，F，S，D，H，O，A	B，G，W，C，J
明文字母（语料库）	E，T，A，I，O，N，R，S	Z，X，Q，J，K

表 4

分析语料库中各个字母概率，我们可以发现语料库中的 E 字母出现的 12.32%，明显大于

其他字母出现的概率（最接近的 T 的概率是 9.56%，相差了近 3 个百分点）。
分析密文中各个字母出现的概率，我们可以得到 R 出现的概率为 12.53%，也是明显大于其他字母出现的概率的（最接近的 V 的概率是 8.13%，相差了 4 个百分点）。
所以我们姑且认为明文中的 E 是对应密文中的 R 的，其实这只是一种假设，情况不符合再重新对应即可。

那么现在的密码表为：

密文—明文对应表							
密文字母	R						
明文字母	E						

表 5

2.高频字母的结构化处理
2.1 仿照语料库中的高频单词做密文高频字母结构化
英语中出现频率最高的单词有 the，of ，and，to，a，in，that，is，I，it，for，as 我们现在有了密文中的字母 R 对应明文中的 E，而语料库中出现频率高的单词中出现 E 的单词为 the，密文对应的明文中 the 的结构也应该会出现的概率比较高，因为我们现在获取的密文是没有标点和间隔的,我们可以在密文中出现字母 R 的地方来统计所有**R 的结构，找到**R 结构中出现最高的具体结构来和 the 做对照。

由于密文是没有标点和间隔的，所以密文对应的明文中出现 the 的结构有三种情况：
第一种：直接出现单词 the
第二种：一个单词里面包含 the，例如 then，brother
第三种：前一个单词的末尾与后一个单词的开头组成了 the，例如 about head

我们选这样的结构一般选三个字母及以上对应的结构，一个两个字母组成的单词（例如 of，of ，to，a，in，is，I）对应的结构是没有意义的，而结构中字母个数比较少的情况上述第二种、第三种情况出现的比较多，不好判断。

所以明文 the 对应的密文的**R 的结构会出现的比较多，我们找到密文**R 出现概率最大的结构来和 the 做对应，我们就可以明文字母 t 和 h 的对应关系。

密文中**R 的结构有如下这些：

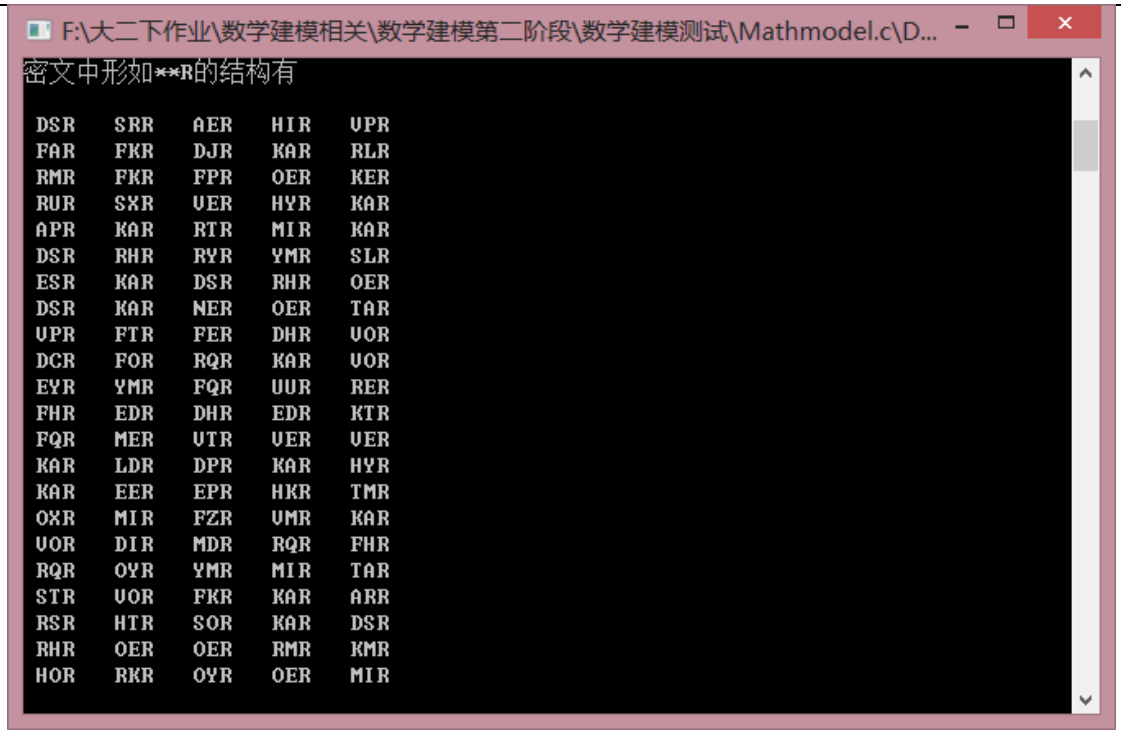

图 10

我们按照字母顺序给这些结构排序，结果如下：

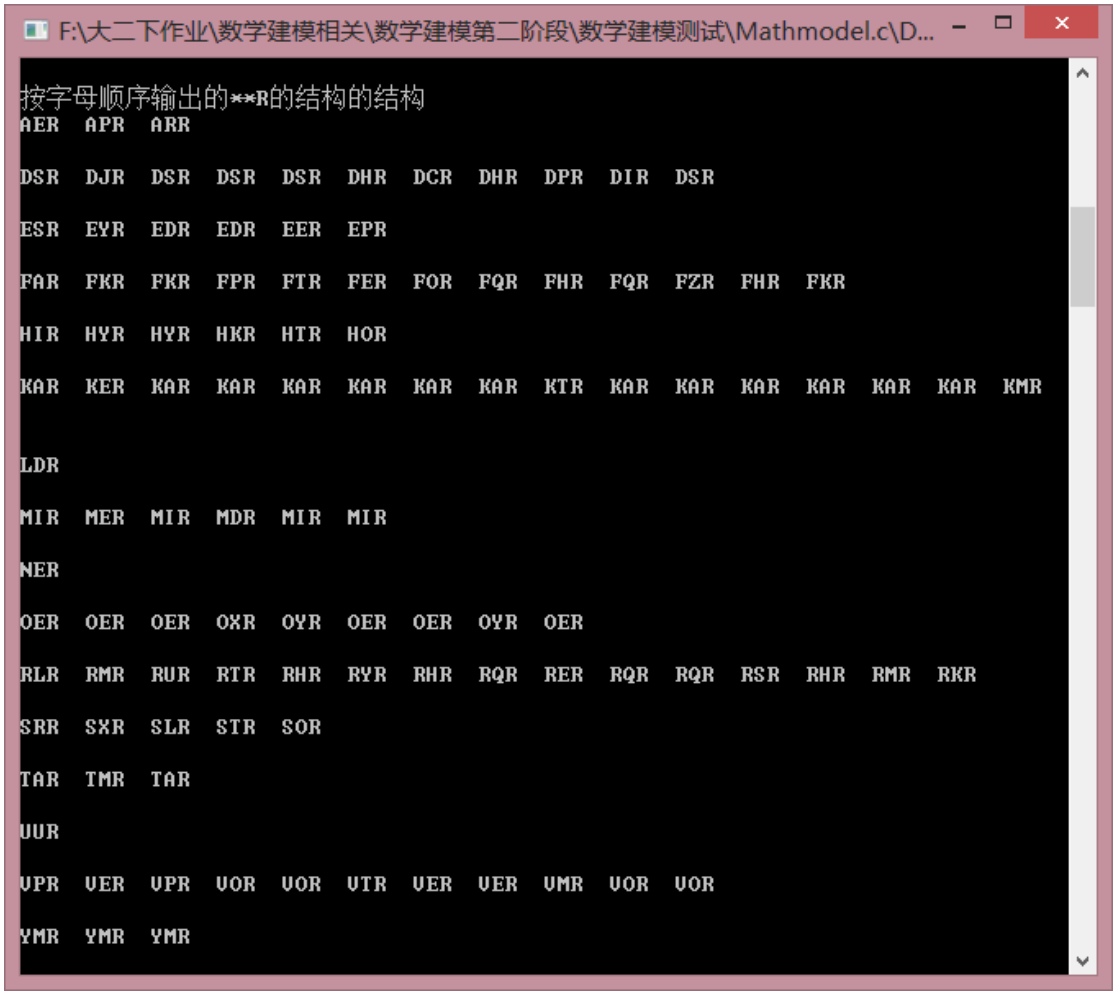

图 11

我们统计**R 各个结构出现的次数：

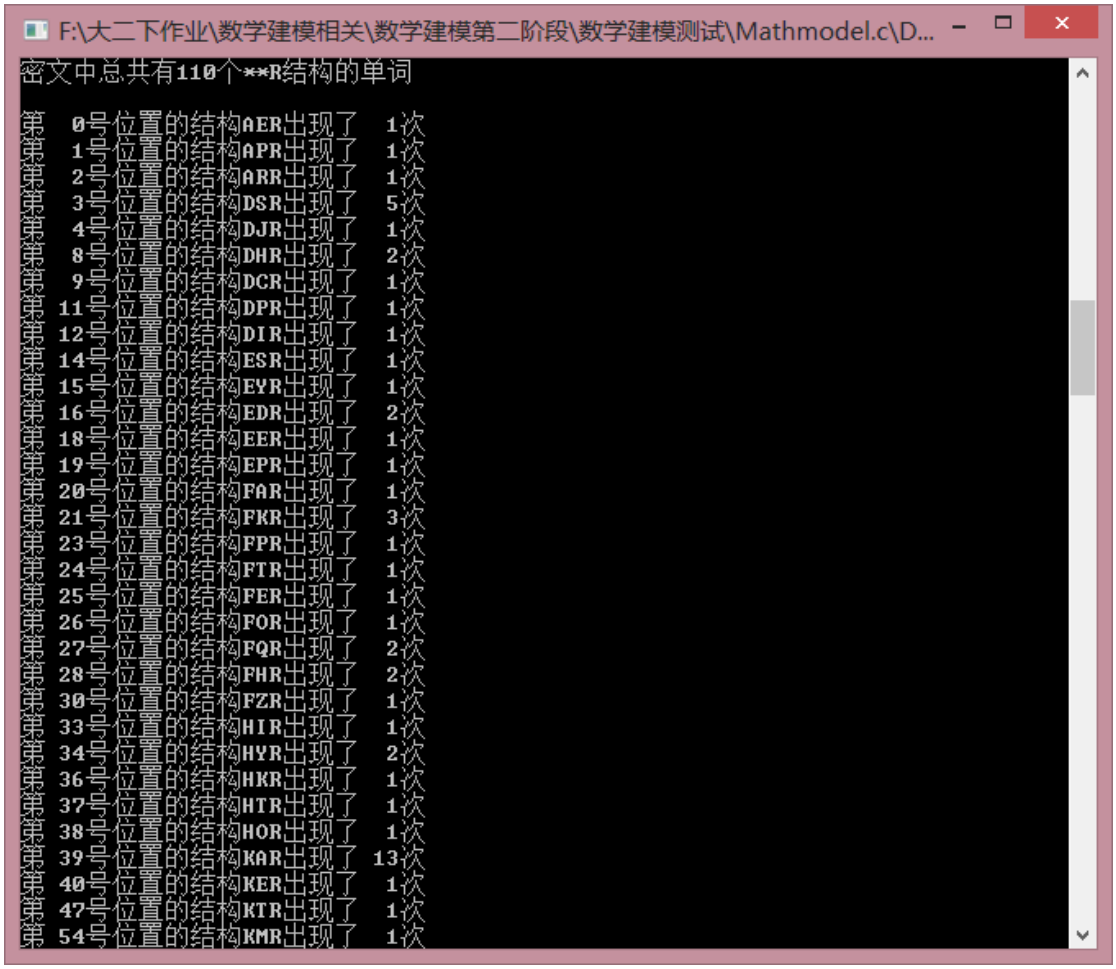

图 12

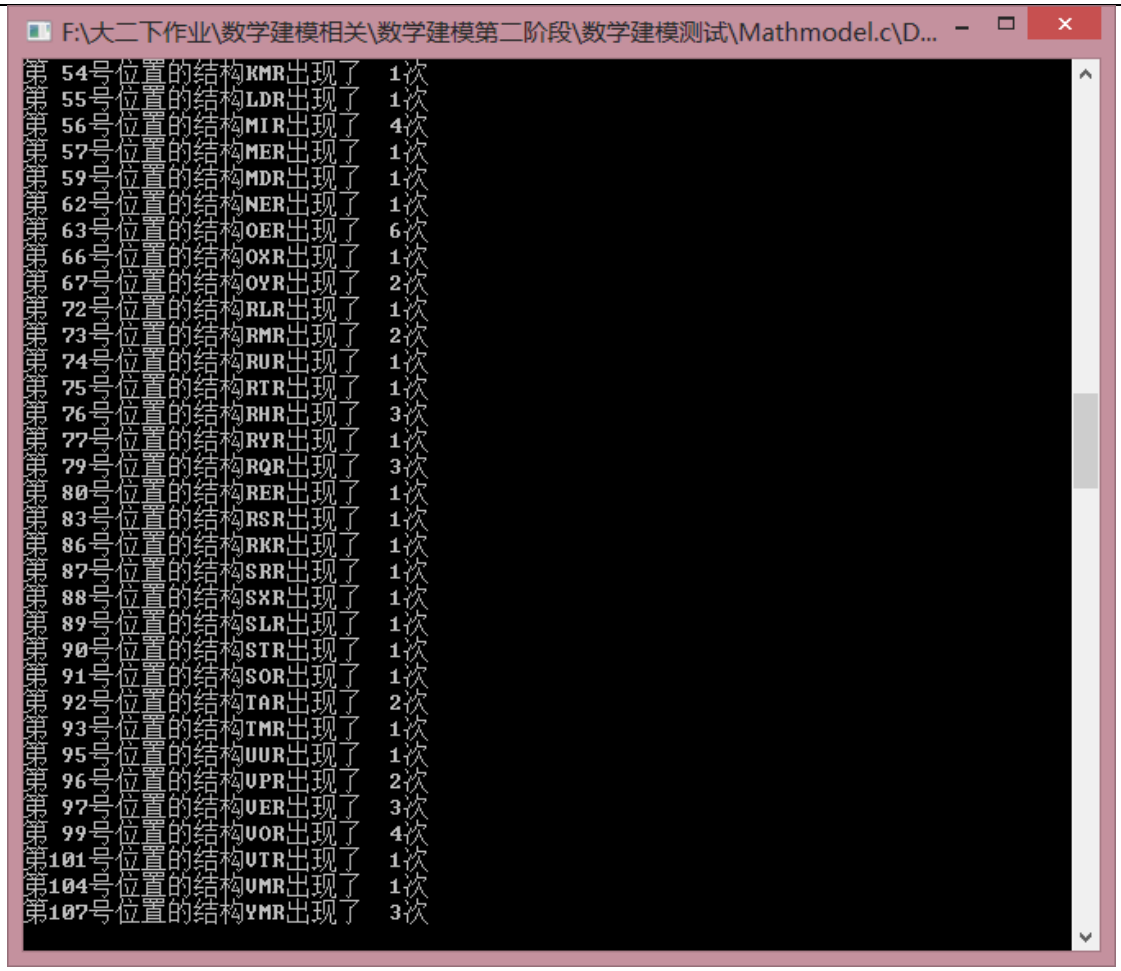

图 13

因为**R 这样的结构组合有太多种可能，出现次数比较低的结构我们可以直接忽略，出现次数比较低的结构对我们的解密是几乎没有帮助的，出现频率高的结构如下：

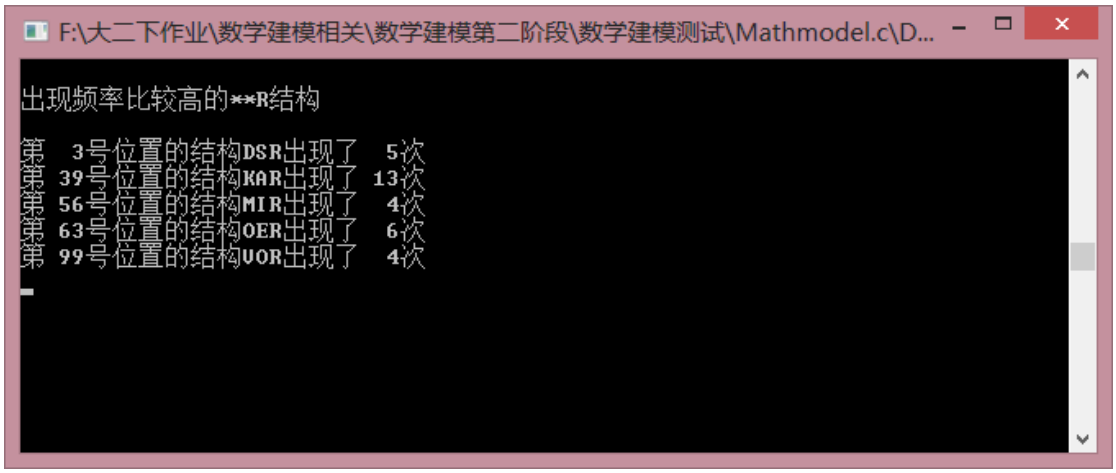

图 14

我们可以发现 KAR 这个结构出现的次数 13 次是远远多于其他**R 结构出现的次数的，所以我们可以认为密文 KAR 是对应明文 the，这样对应的概率是比较高的，很难出错。

即现在我们有了密文字母 K 对应明文字母 t，密文字母 A 对应明文字母 h，故此时密码表为：

密文—明文对应表							
密文字母	R	K	A				

明文字母	E	T	H				

表 6

3.任意文章进行同步推演

我们在网上随便找一篇常用的文章（最好和需要解密密文对应的明文是同一个领域的），对这篇文章进行去标点去空格操作，不对这篇文章进行加密过程，然后对这段文章进行和密文一样的对应字母的结构化处理，例如我们可以在这篇文章中找**E（因为密文的 R 对应我们明文的 E）的结构，然后统计各种**E 出现的次数，然后和密文中的**R 做对比。说明：我们解密的时候这样的文章是随便找的，只要文章不是比较生僻的就行，如果和需要解密密文对应的明文是同一个领域的，那样更会事半功倍。

我们随便找到的常用文章如下：

Today was Sunday. My parents were free,too. I got up at 7'oclock, because my families planed to go to the zoo. After the breakfast, I took the camera and went to the station together with my parents. It was already 9'oclock when we arrived at the zoo. There were so many monkeys, tigers, lions, wolves and other animals in the zoo. We took many photos in the zoo with the animals. I will show these photos in my class after they were printed. The time passed so fast, we left the zoo at 1'oclock PM. I was so happy today. Now I feel very tired and I will go to bed early tonight.

English is one of my best subjects and I started learning English when I was ten years old. But at the very beginning, listening seemed a little difficult to me. So I have been doing a lot of listening practice, such as listening to tapes, watching English TV programs. And I found it really helped a lot. In fact, there are some more helpful ways to learn English well. For example, I enjoy singing English songs and I want to join an English club or find a pen pal from English-speaking countries. I believe that nothing is impossible if you put your heart into it.

对这篇文章进行去标点去空格、小写变大写之后的操作之后的文章如下：

图 15

我们在这篇文章中找**E 的结构，和密文的**R 的结构做对比。

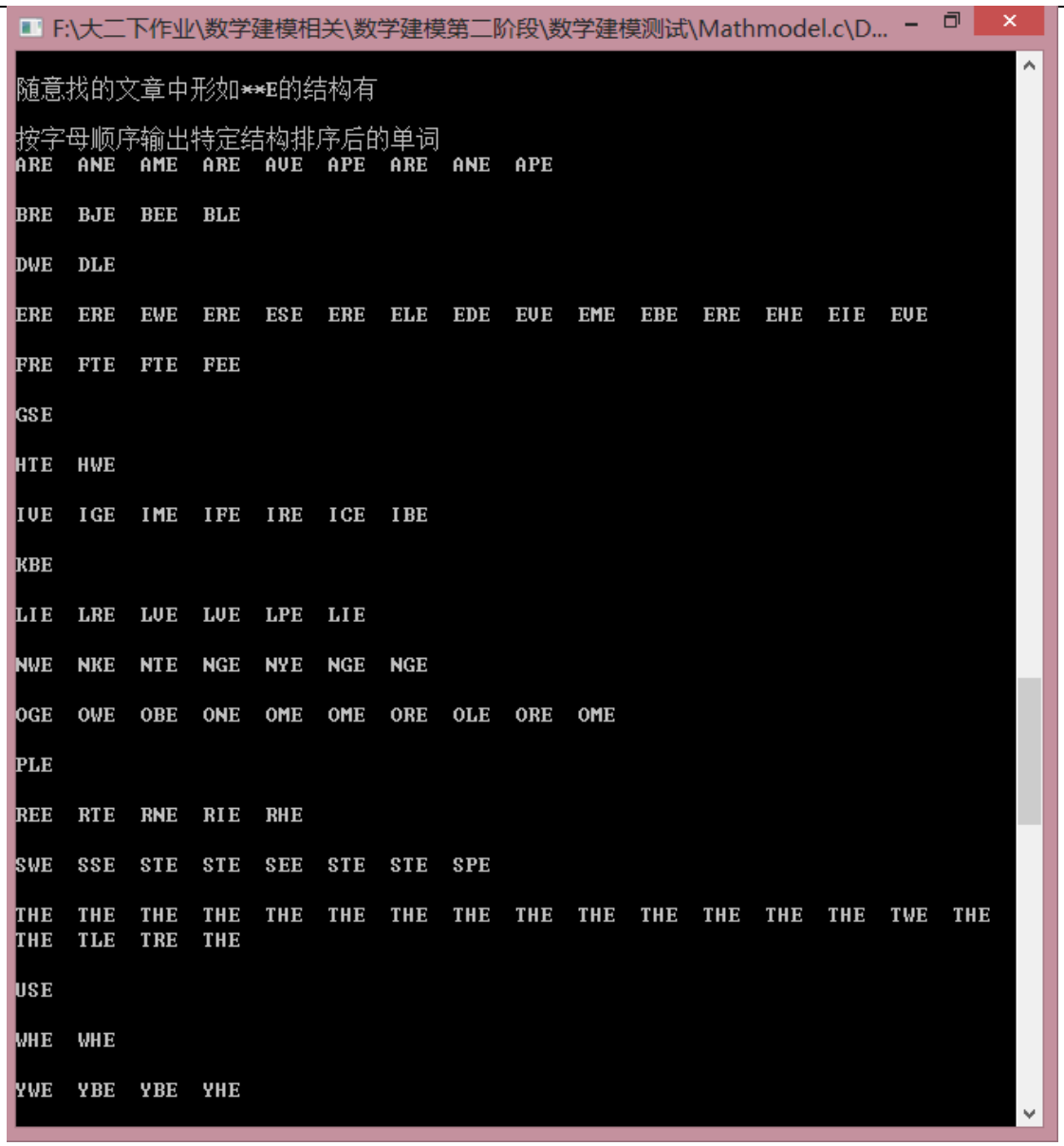

图 16

找到这篇文章中出现频率高的**E 结构，和密文的**R 的结构做对比。

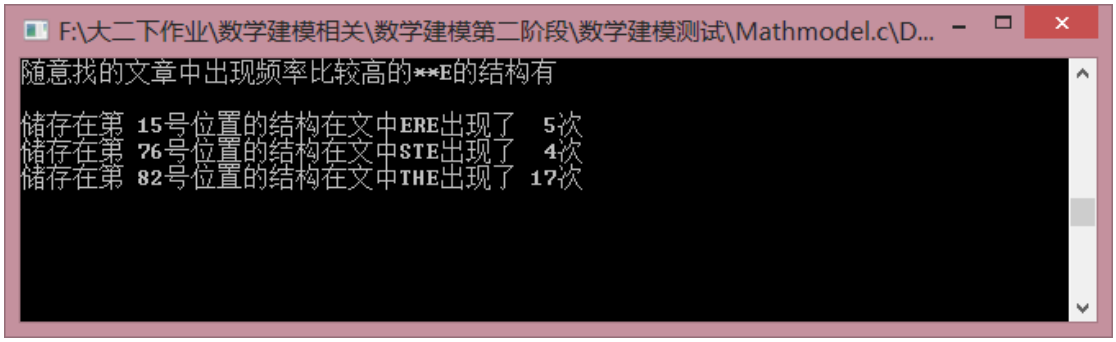

图 17

下面是密文中的高频**R 的结构和随意找的文章中高频**E 结构对比表：

密文中的高频**R 的结构和随意找的文章中高频**E 结构对比表			
密文高频**R 结构		随意文章高频**E 结构	
结构	次数	结构	次数
KAR	13	THE	17
OER DSR MIR VOR	6 5 4 4	ERE STE	5 4

表 7

通过对这个表格的分析，在随意找的文章中 THE 结构出现了 17，远高于其他结构出现的次数，在密文中找到的 KAR 结构的次数为 13 次，远高于其他结构，所以我们进一步的验证了密文 KAR 对应明文 THE，即进一步验证了密文 K 对应明文 T，密文 A 对应明文 H。

随意文中出现次数相对较高的 ERE、STE 和密文中出现相对次数较高的 OER、DSR、MIR、VOR 也可能有着某种对应关系，但是对应关系不明显，我们先暂时不去分析它们，其实这里可以通过编程一条一条尝试对应。

随意文中的 ERE 的第一个字母和第三个字母是相同的，在单字母替换加密中对应的密文也应该是第一个字母和第三个字母相同，OER、DSR、MIR、VOR 都不满足这样的条件。

4.结构化单词间隔划分

我们根据刚刚已经得到的密文 KAR 对应明文 THE，对密文中的 KAR 的前后分别加空格。

由于密文是没有标点和间隔的,所以密文对应的明文中出现 the 的结构有三种情况：
第一种：直接出现单词 the
第二种：一个单词里面包含 the，例如 then，brother
第三种：前一个单词的末尾与后一个单词的开头组成了 the，例如 about head

虽然这样加空格肯定会破坏第二种和第三种结构，但是也会找出第一种结构的单词 the，而且加空格后整个密文我们看起来会轻松明了，当我们不断循环我们的解密前几个步奏，找到的 the 这样的结构越来越多的时候，添加空格会把一个单词拆散和把不同单词头尾拼合在一起，到时候我们只用根据英语语言进行拆分和拼合，这样会比较简单。

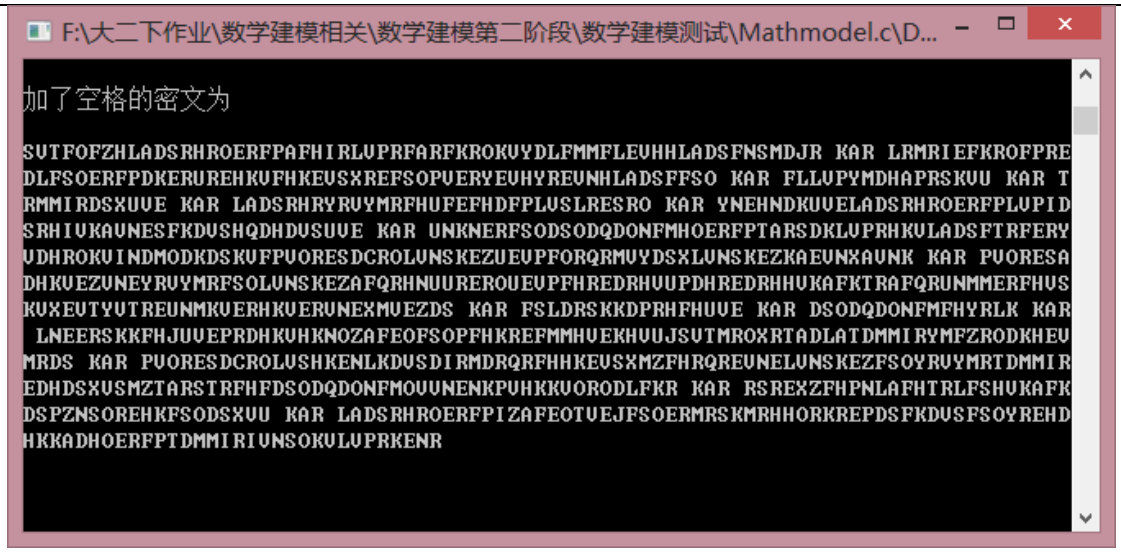

图 18

5. 部分密码表试解密

我们已经得到一个部分明文密文对应关系的密码表为：

密文—明文对应表							
密文字母	R	K	A				
明文字母	E	T	H				

表 8

我们用这个已经假设出来的部分明文密文对应关系的密码表对密文进行解密，得到结果如下图：

图 19

现在得到的已知部分密码表试解密出来的密文作用不大，当密码表中的字母对应关系达到一定的数量后，就会比较有用。

6. 循环步骤 2 到步骤 5

走完上面的五个步骤，我们完成了一个流程，我们继续循环到第 2 步：

6.1 高频字母的结构化处理

我们用已知的密码表中密文出现的高频字母来推测出适合这个高频单词的结构，例如：

密文—明文对应表							
密文字母	R	K	A				
明文字母	E	T	H				

表 9

英语中出现频率最高的单词有 the，of ，and，to，a，in，that，is，I，it，for，as

我们现在有了密文中的字母 K 对应明文中的 T，而语料库中出现频率高的单词中出现 T 的单词为 that，密文对应的明文中 that 的结构也应该会出现的概率比较高，因为我们现在获取的密文是没有标点和间隔的，我们可以在密文中出现字母 K 的地方来统计所有 K**K 的结构，又因为我们现在已经知道密文 H 是对应明文 A，所以我们可以在密文中出现字母 K 的地方来统计所有 KA*K 的结构来对照 that，而且 th*t 的单词只有 that 一个，但是还是有别的单词包含 that 的情况，例如 thatch（茅草），也有可能是前一个单词的尾部和下一个单词的头部组合成 that，例如 death at，但是这两种情况出现的概率较低，远远少于如英语高频单词 that 自身出现的概率，所以如果密文中 KA*K 出现的概率相对较高，那么几乎可以断定密文 KA*K 就是明文 that。

密文中的 KA*K 有可能是密文经过带有噪声干扰信道传输是字母的增删改而来，但是这个的概率也比较低，高频单词本身比较多，增删改字母之后高频单词还是会比较多，所以影响不大，上文有具体详细的分析了字母增删改对字母频率分析的影响。

图 20

我们发现 KA*K 的结构 KAFK 出现了两次，

故密文和语料库的字母出现频率高低对应表：

	出现频率比较高的字母	出现频率比较低的字母
密文字母（密文统计）	R，V，E，K，F，S，D，H，O，A	B，G，W，C，J
明文字母（语料库）	E，T，A，I，O，N，R，S	Z，X，Q，J，K

表 10

又密文中 S 是密文出现概率较高的字母，与 S 对应的 N 是语料库中出现概率较高的字母，密文中的 O 是密文出现概率一般的字母，与 O 对应的 D 是语料库中出现概率一般的的字母，故可以进一步验证密文 FSO 就是对应明文中的 and。

即密文 S 对应明文 N，密文 O 对应明文 D，此时密码表为：

密文—明文对应表

密文字母	R	K	A	F	S	O	
明文字母	E	T	H	A	N	D	

表 11

6.3 第三次高频字母的结构化处理

我们现在有了密文中的字母 F 对应明文中的 A，而语料库中出现频率高的单词中出现 A 的单词有 as，正如我们前面所说，as 这个单词只包含两个字母，所以虽然 as 本身为一个高频单词，但是 as 出现在别的单词里面的几率较大，但是由于我们在密文中找的是找的多种但凡能出现 as 结构的和，所以相对而言 as 还是会最多，下面我举例证明。

at 在语料库中也是出现频率较高的单词，但是我们已经求得的密码表里面已经包含了明文中的 a，t 两个字母，所以 at 和 as 不会产生混淆。对照已经求得的部分密码表可知 at 对应的密文为 FK，我们判断的时候直接判断 FK 为 at 即可。

现在判断 as 对应的密文和别的以 a 开头的结构（例如 ai）对应密文的次数的关系：
as 对应密文出现总次数 = 明文中 as 这个单词出现次数 + 单词中包含 as 结构次数 + 上一个单词末尾和下一个单词开头组成的 as 次数 + 密文传输时候对 as 的增删改情况
ai 对应密文出现总次数 = 明文中 ai 这个单词出现次数 + 单词中包含 ai 结构次数 + 上一个单词末尾和下一个单词开头组成的 ai 次数 + 密文传输时候对 ai 的增删改情况

明文中 as 这个单词出现次数 是远远大于 明文中 ai 这个单词出现次数（因为 ai 不是单词），而后面几项对 as 和 ai 的影响差不多，所以结合起来 as 结构出现的次数还是会明显大于其他以 a 开头的两个字母出现的概率。

所以我们现在在密文中找 F*的结构，具体 F*结构出现多的几乎就可以对应 as。
下面是 F*各个结构出现的次数和概率：

图 21

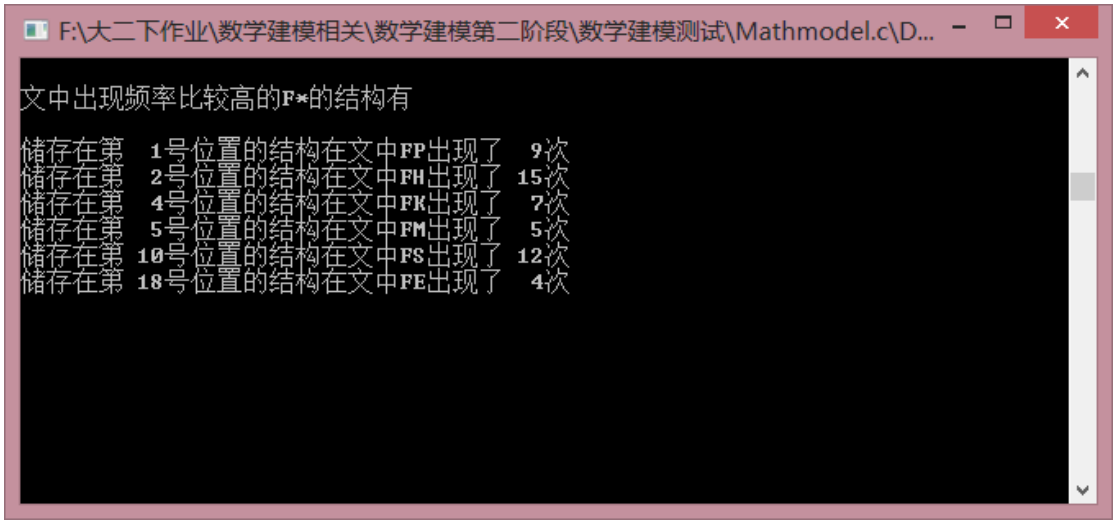

图 22

在这个图中我们可以发现 FH（15 次）和 FS（12 次）出现的概率都相对较高，对比现在已有的密码表：

密文—明文对应表							
密文字母	R	K	A	F	S	O	
明文字母	E	T	H	A	N	D	

表 12

对比现在已有的密码表可以知道 FS 对应的明文字母为 an，故我们可以大致推测出密文 FH 是对应明文的 as 的，即密文的 H 对应明文的 S 字母。

密文和语料库的字母出现频率高低对应表：

	出现频率比较高的字母	出现频率比较低的字母
密文字母（密文统计）	R，V，E，K，F，S，D，	B，G，W，C，J

	H，O，A	
明文字母（语料库）	E，T，A，I，O，N，R，S	Z，X，Q，J，K

表 13

又根据上文的密文中的各个字母概率的统计和语料库中各个字母概率的统计，密文的 H 字母是密文中出现概率较高的字母，语料库中的的 S 字母是语料库中出现概率较高的字母，所以可以认为密文 H 对应明文 S；

此时密码表为：

密文—明文对应表							
密文字母	R	K	A	F	S	O	H
明文字母	E	T	H	A	N	D	S

表 14

6.4 任意文章进行同步推演

上文高频字母的结构化处理了 KA*K（对应明文 that）结构，F**（对应明文 and）结构，F*（对应明文 as）结构，任意文章同步推演就是对明文结构做对密文结构同样的处理，下面是 KA*K，F**，F*的实例。

a.密文结构 KA*K 实例

密文结构 KA*K（that），我们找出此时已经去标点、去空格、小写变大写之后的操作之后的文章的结构 TH*T（that），然后对密文结构 KA*K（that）和对随意文章结构 TH*T（that）做同意的操作，找出各自高频的结构做对照。

密文结构 KA*K（that） 推出 / 对照 随意文章结构 TH*T（that）

密文结构 KA*K（that）结构出现情况：

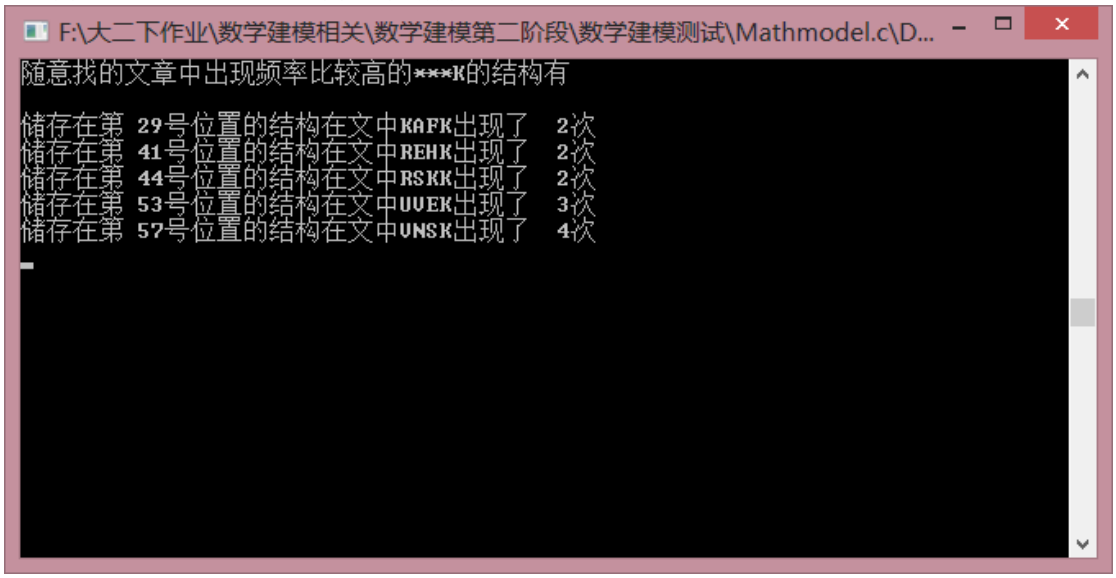

图 23

随意文章结构 TH*T（that）出现情况：

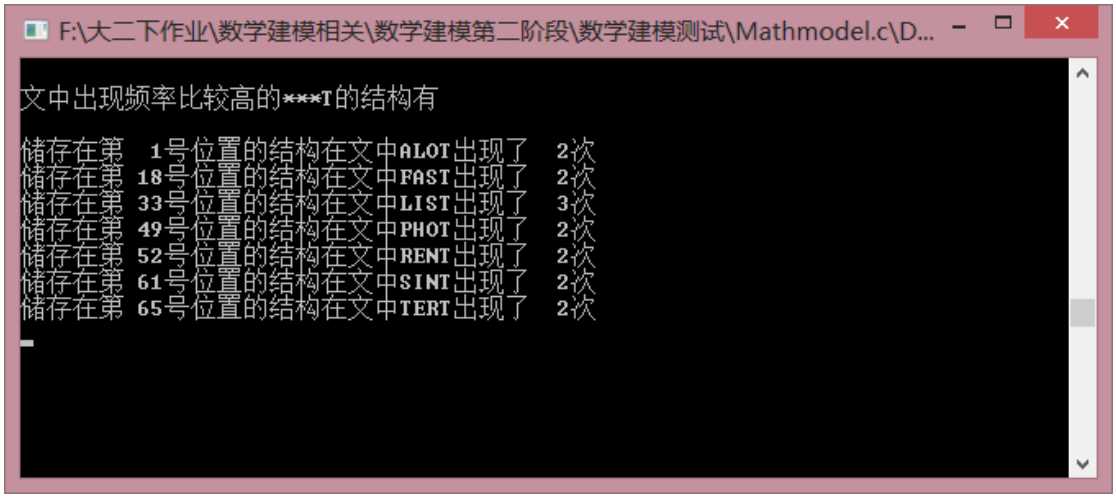

图 24

编程找的时候我是直接把密文结构 KA*K（that）结构简化为了***K 的结构，把随意文章结构 TH*T（that）简化为了***T 的结构。
上面两图中中各个结构出现的次数都很低，参照效果不明显。

b.密文结构 F**实例

现在我们分析密文结构 F**（对应单词 and）和随意文章结构 A**（对应单词 and）之间的对照。

密文结构 F**（and） 推出 / 对照 随意文章结构 A**（and）

密文结构 F**（as）结构出现情况：

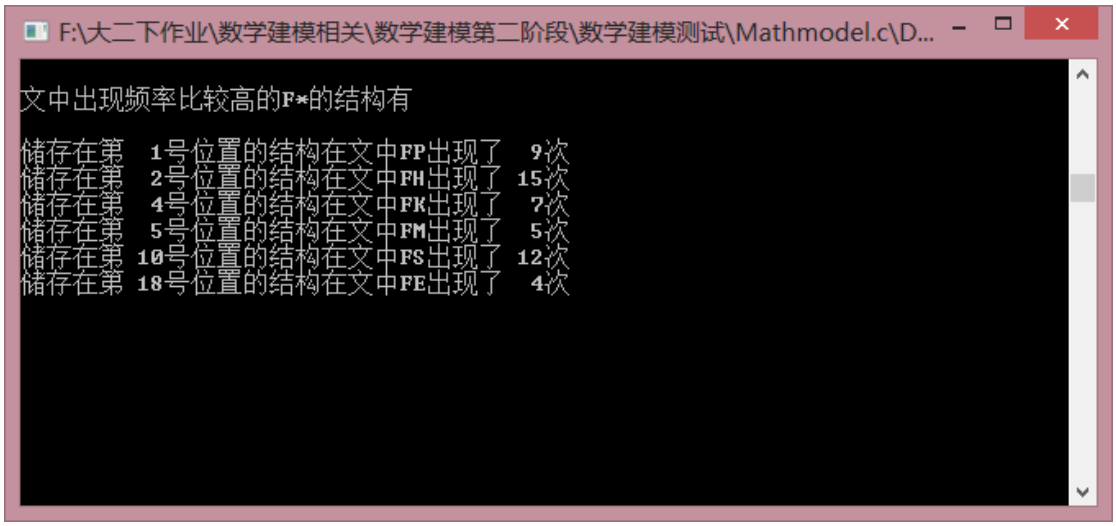

图 25

随意文章结构 A**（and）出现情况：

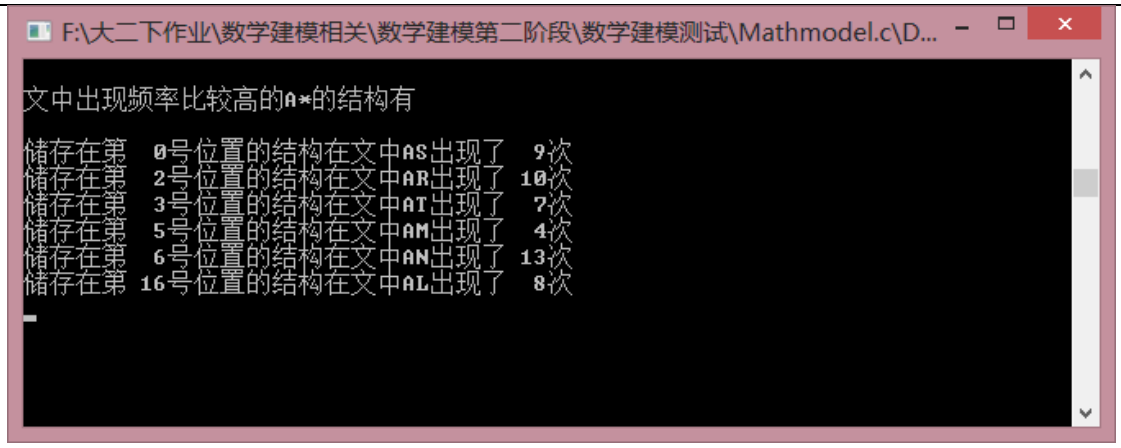

图 26

两个图对比，去掉密码表中已知的结构 at，as，an，可得密文中的 FP，FM，FE 对应随机文中的 AS，AR，AM，AL。

故可得到如下的对应关系：

密文	明文
P M E	S R M L

表 15

密文和语料库的字母出现频率高低对应表：

	出现频率比较高的字母	出现频率比较低的字母
密文字母（密文统计）	R，V，E，K，F，S，D，H，O，A	B，G，W，C，J
明文字母（语料库）	E，T，A，I，O，N，R，S H，R，D，L，U	Z，X，Q，J，K

表 16

结合上面两个表得：

密文	明文
P(密文中的中频词) M（密文中的高频词） E（密文中的高频词）	S（语料库中的高频词） R（语料库中的高频词） M（语料库中的中频词） L（语料库中的高频词）

表 17

上表只存在两个存在对应关系的中频词，故很容易猜想密文中的字母 P(密文中的中频词)对应明文中的字母 M（语料库中的中频词），此时为猜想，但是有很大的可能性，其实我们可以编程验证，穷举上面的 3*4=12 种可能，故此时可知密文 P 对应明文 M，此时的密码表为：

密文-明文对应表																									
密文	F			O	R			A					P	S					H	K						
明文	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

表 18

6.5 结构化单词间隔划分

上面过程中已经找到的单词有 HN(as)，KAFK(that)，FSO（and)，KD(to)
对密文中但凡出现过这些单词的位置都给它划分开，所得的结果如图：

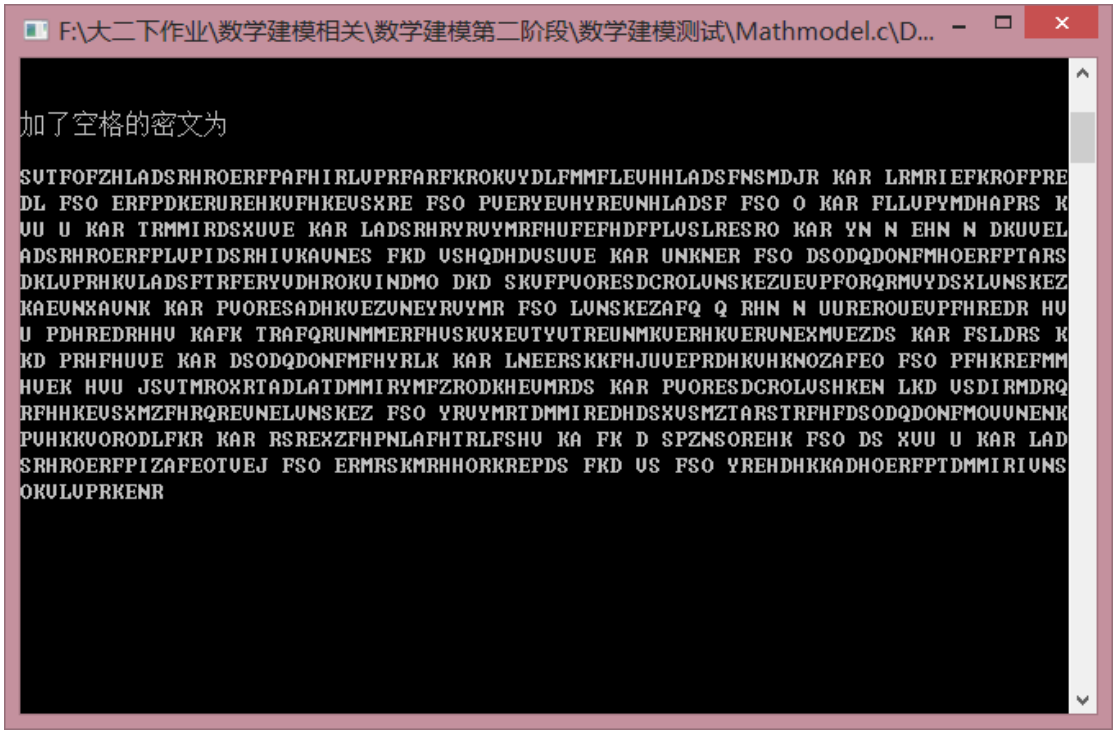

图 27

6.6 部分密码表试解密

用已知的密码表对密文进行解密，所得结果如下：

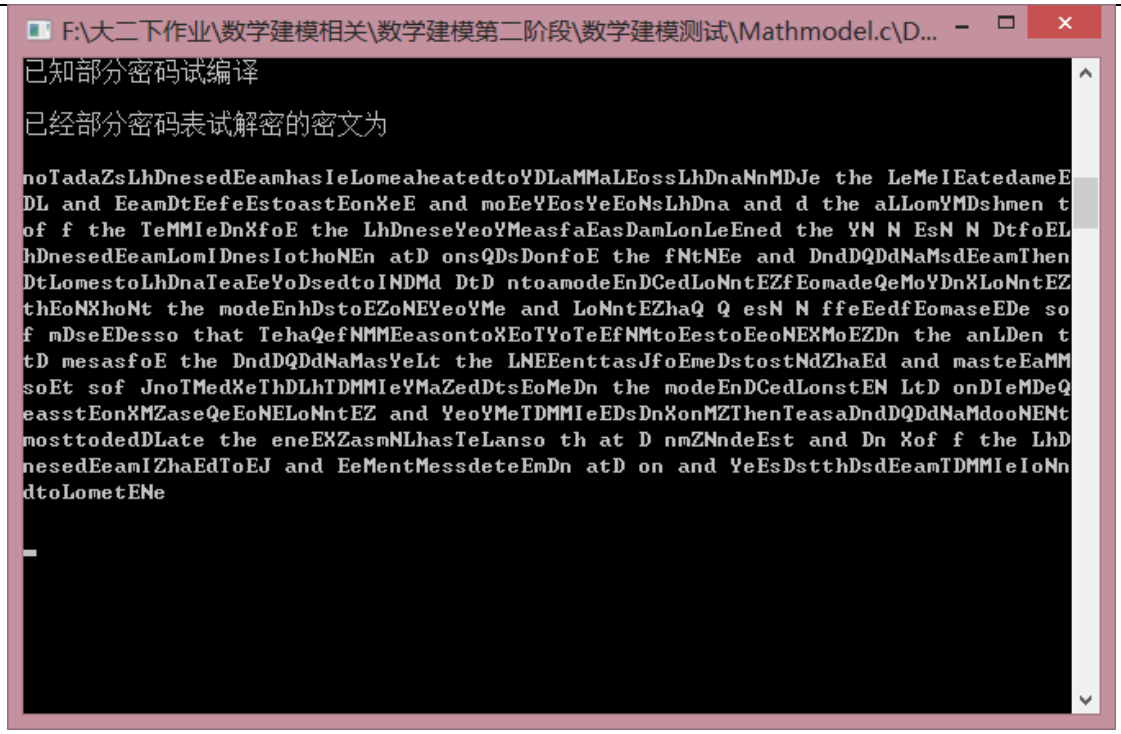

图 28

对应的文章为：

noTadaZsLhDnesedEeamhasIeLomeaheatedtoYDLaMMaLEossLhDnaNnMDJe the
（结合前后字母组合容易发现这里是 heated to）
LeMeIEatedameEDL and EeamDtEefeEstoastEonXeE and moEeYEosYeEoNsLhDna and d the
（容易发现这里是 more）
aLLomYMDshmen tof f the TeMMIeDnXfoE the LhDneseYeoYMeasfaEasDamLonLeEned the
（容易判断这里肯定发生了字母增删改的错误）
YN N EsN N DtfoELhDnesedEeamLomIDnesIothoNEn atD onsQDsDonfoE the fNtNEe and
DndDQDdNaMsdEeamThenDtLomestoLhDnaTeaEeYoDsedtoINDMd DtD
ntoamodeEnDCedLoNntEZfEomadeQeMoYDnXLoNntEZthEoNXhoNt the
modeEnhDstoEZoNEYeoYMe and LoNntEZhaQ Q esN N ffeEedfEomaseEDe sof mDseEDesso
that TehaQefNMMEeasontoXEoTYoTeEfNMtoEestoEeoNEXMoEZDn the anLDen ttD mesasfoE
the DndDQDdNaMasYeLt the LNEEenttasJfoEmeDstostNdZhaEd and masteEaMMsoEt sof
JnoTMedXeThDLhTDMMIeYMaZedDtsEoMeDn the modeEnDCedLonstEN LtD
onDIeMDeQeasstEonXMZaseQeEoNELoNntEZ and
YeoYMeTDMMIeEDsDnXonMZThenTeasaDndDQDdNaMdooNENtmosttodedDLate the
eneEXZasmNLhasTeLanso th at D nmZNndeEst and Dn Xof f the LhDnesedEeamIZhaEdToEJ
（容易发现这里是 dream）（进而可以推测出 dEeam 前面的 LhDnese 为 chinese，LhDnese
不可能发生增删改错误，因为这段文字前面也出现过这个）
and EeMentMessdeteEmDn atD on and YeEsDstthDsdEeamTDMMIeIoNndtoLometENe
（容易发现这里是 to come）

故由上面的分析可以得到密文 E 对应明文 R，密文 L 对应明文 C，密文 D 对应明文 I，密文 V 对应明文 O，密文 U 对应明文 F，故此时的密码表为：

密文-明文对应表																										
密文	F		L	O	R	U		A	D				P	S	V			E	H	K						
明文	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

表 19

此时也得到单词 LVPR（come），OERFP（dream），LADSRHR（chinese），VU（of），KV（to），PVER（more）

6.7 再次对结构化单词间隔划分

我们得到新的单词 LVPR（come），OERFP（dream），LADSRHR（chinese），VU（of），KV（to），PVER（more），我们加上这些新的单词在对原文进行结构化单词间隔划分，所得结果如下：

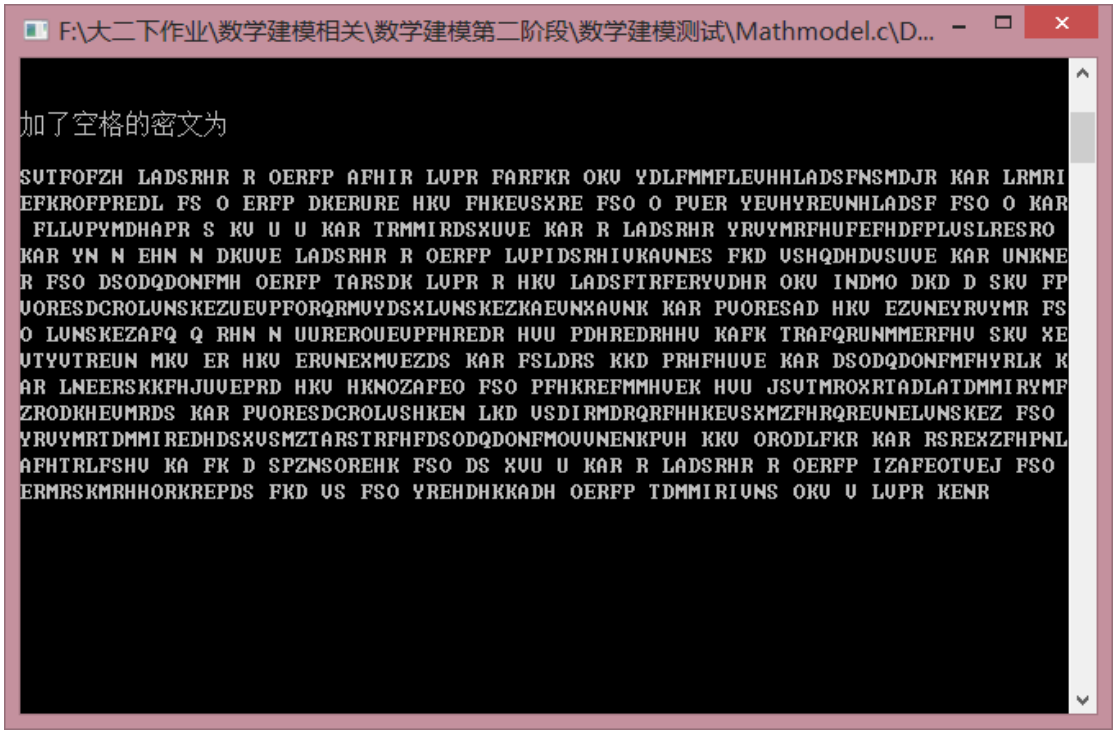

图 29

6.8 部分密码表试解密

用已知的密码表对密文进行解密，所得结果如下：

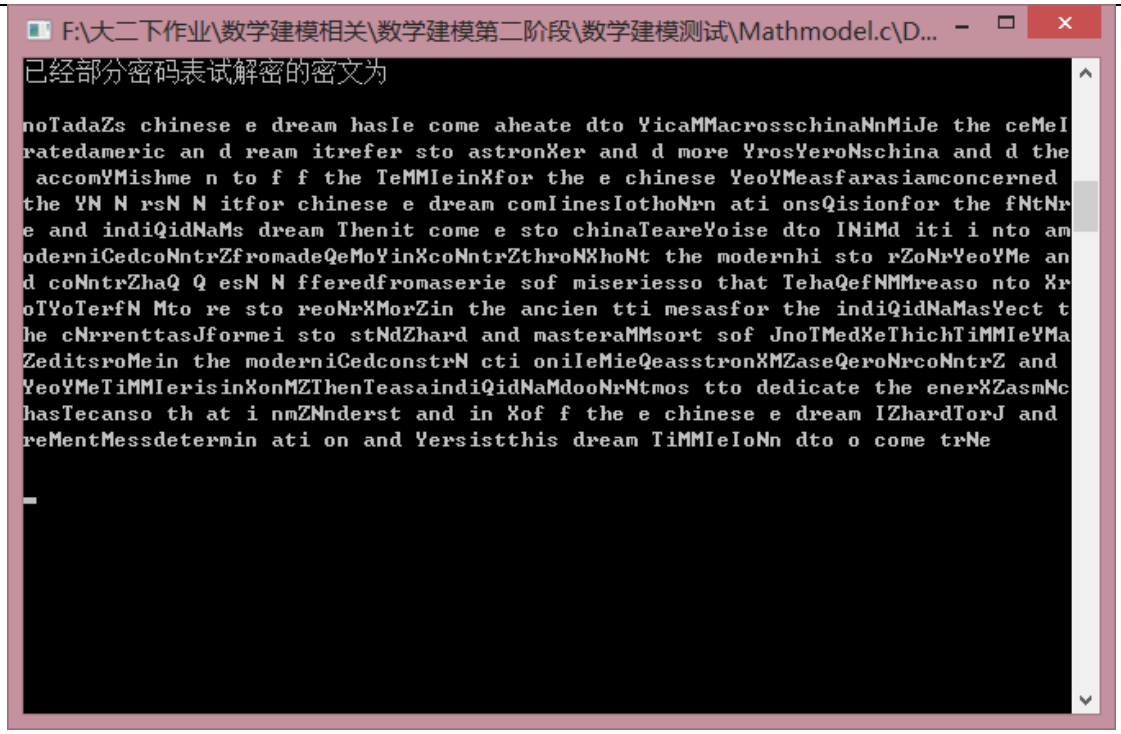

图 30

对应的文字如下：

noTadaZs chinese e dream hasIe come aheate dto YicaMMacrosschinaNnMiJe the
（根据语意义结合后文结合自身结构容易知道这个单词为 nowadays）
（容易知道这里发生了字母的增加，增加了一个字母 e）
（容易知道 hasIe come aheate dto Yic 是 has become a heated topic）
ceMeIratedameric an d ream itrefer sto astronXer and d more YrosYeroNschina and
d the
（容易知道这里为 american dream）
（容易知道这里为 refers to a stronger）
（容易知道这里发生了字母增删改，多插了一个字母 d）
accomYMishme n to f f the TeMMIeinXfor the e chinese YeoYMeasfarasiamconcerned
the YN N
（容易知道这个单词为 accomplishment）
（容易知道这里发生了字母的增加，增加了一个字母 f）
（容易知道这里发生了字母的增加，增加了一个字母 e）
rsN N itfor chinese e dream comIinesIothoNrn ati onsQisionfor the fNtNre and
indiQidNaMs
（容易知道这里发生了字母的增加，增加了一个字母 e）

dream Thenit come e sto chinaTeareYoise dto INiMd iti i nto
（容易知道这个单词为 when it）
amoderniCedcoNntrZfromadeQeMoYinXcoNntrZthroNXhoNt the modernhi sto
rZoNrYeoYMe
（容易知道这个单词为 a modernized country）
（容易知道这个单词为 modern history）

and coNntrZhaQ Q esN N fferedfromaserie sof miseriesso that TehaQefNMMreaso nto XroTYoTerfN Mto re sto reoNrXMorZin the ancien tti mesasfor the indiQidNaMasYect the cNrrenttasJformei sto stNdZhard and masteraMMsort sof JnoTMedXeThichTiMMIeYMaZeditsroMein the moderniCedconstrN cti oniIeMieQeasstronXMZaseQeroNrcoNntrZ and YeoYMeTiMMIerisinXonMZThenTeasaindiQidNaMdooNrNtmos tto dedicate the enerXZasmNchasTecanso th at i nmZNnderst and in Xof f the e chinese e dream IZhardTorJ and

（容易知道这个单词为 as we can）

reMentMessdetermin ati on and Yersistthis dream TiMMIeIoNn dto o come trNe

（容易知道这个单词为 true）

故由上面的分析可以得到密文 N 对应明文 U，密文 T 对应明文 W，密文 Z 对应明文 Y，密文 C 对应明文 Z，密文 Y 对应明文 P，密文 M 对应明文 L，密文 X 对应明文 G，密文 I 对应明文 B，故此时的密码表为：

密文-明文对应表																										
密文	F	I	L	O	R	U	X	A	D			M	P	S	V	Y		E	H	K	N		T		Z	C
明文	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

表 20

此时也得到单词 UEVP（from），LVNSKEZ（country），PVORES（modern），DK（it），DS（in），FH（as），TR（we），LFS（can）

6.9 再次对结构化单词间隔划分

我们得到新的单词 UEVP（from），LVNSKEZ（country），PVORES（modern），DK（it），DS（in），FH（as），TR（we），LFS（can），我们加上这些新的单词在对原文进行结构化单词间隔划分，所得结果如下：

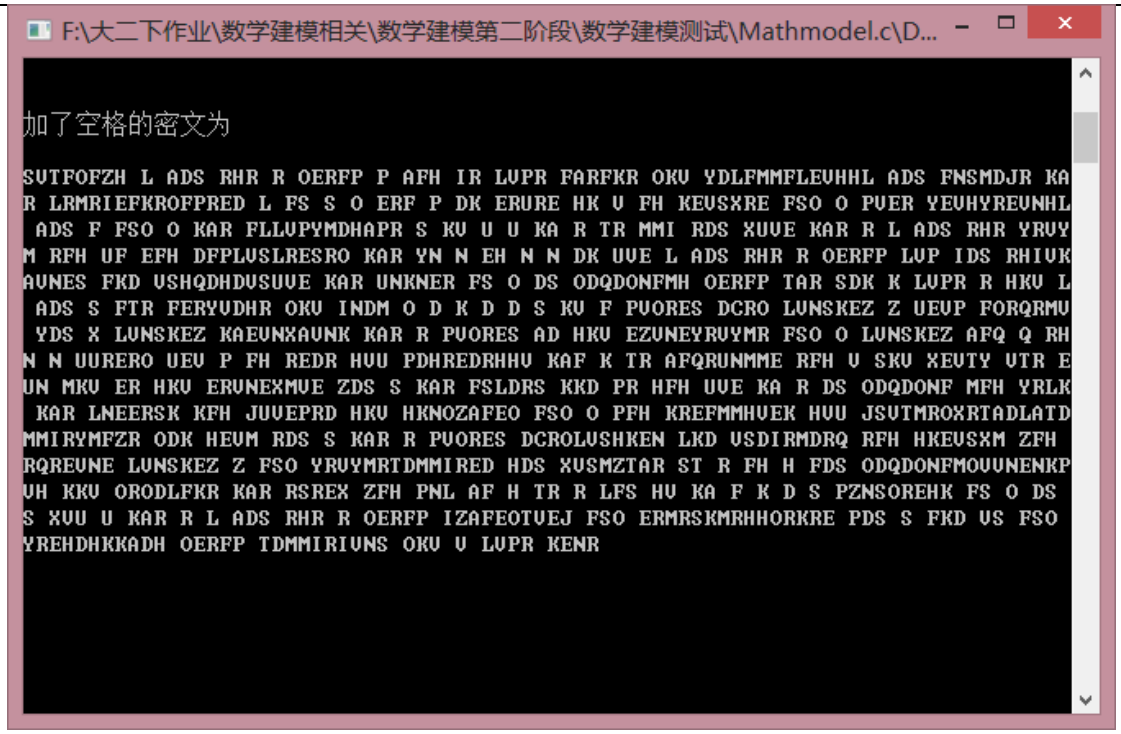

图 31

6.10 部分密码表试解密

用已知的密码表对密文进行解密，所得结果如下：

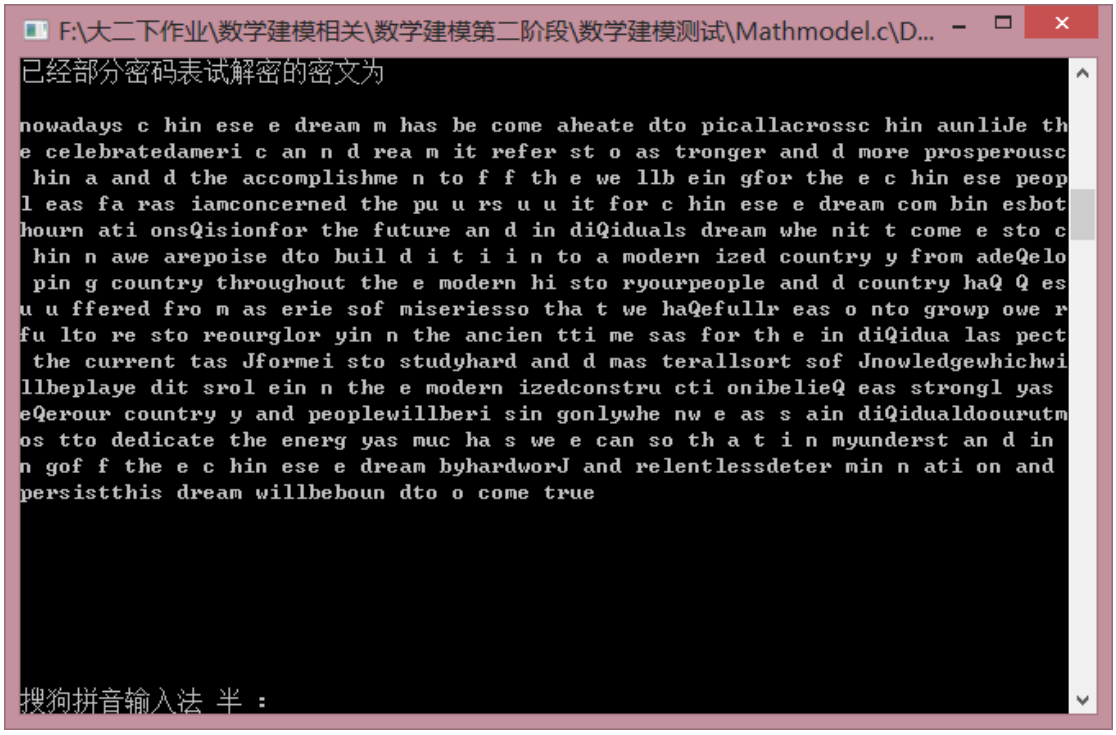

图 32

对应的文字如下：（红字表示每段密文对应的明文）

nowadays c hin ese e dream m has be come aheate dto picallacrossc hin aunliJe the

Nowadays Chinese dream has become a heated topic all across China Unlike the
celebratedameri c an n d rea m it refer st o as tronger and d more nergy y as hin a and d the
celebrated American dream,it refers to a stronger and more prosperous China and the
accomplishme n to f f th e we llb ein gfor the e c hin ese nergy eas nergy iamconcerned the pu u rs
accomplishment of the well-being for the Chinese people.As far as I am concerned,the pursuit
u u it for c hin ese e dream com bin esbothourn ati onsQisionfor the future an d in diQiduals dream
for Chinese Dream combines both our nation’s vision for the future and individual’s dream
whe nit t come e sto c hin n awe arepoise dto buil d I t I I n to a modern ized country y from
When it comes to China,we are poised to build it into a modernized country from
adeQelo pin g country throughout the e modern hi sto ryourpeople and d country haQ Q esu u
a developing country.Throughout the modern history,our people and country have suffered
ffered fro m as erie sof miseriesso tha t we haQefullr eas o nto growp owe rfu lto re sto reourglor
from a series of miseries so that we have full reason to grow powerful to restore our glory
yin n the nergy tti me sas for th e in diQidua las pect the current tas Jformei sto studyhard and d
in the ancient times.As for the individual aspect,the current task for me is to study hard and
mas terallsort sof Jnowledgewhichwillbeplaye dit srol ein n the e modern izedconstru cti
master all sorts of knowledge which will be played its role in the modernized construction
onibelieQ eas nergy y as eQerour country y and peoplewillberi sin gonlywhe nw e as s ain
I believe as strongly as ever our country and people will be rising only when we as a
diQidualdoourutmos tto dedicate the nergy yas muc ha s we e can so th a t I n myunderst an d in n
individual do our utmost to dedicate the energy as much as we can.So that in my understanding
gof f the e c hin ese e dream byhardworJ and relentlessdeter min n ati on and persistthis dream
of the Chinese Dream.By hard work and relentless determination and persist,this dream

willbeboun dto o come true
will be bound to come true

故由上面的分析可以得到密文 Q 对应明文 V，密文 J 对应明文 K，故此时密码表为：

密文-明文对应表																									
密文	F	I	L	O	R	U	X	A	D		J	M	P	S	V	Y		E	H	K	N	Q	T		Z	C
明文	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

表 21

密文中的 G，W，B 三个字母没有出现过，所以没有他们的对应关系，所以明文字母 J，Q，X 没有对应，所以上面为完整密文表。

密文传输信道干扰造成的影响不大，经过解密检查最后发现这段密文增了 5 个字母，删了 5 个字母，改了 6 个字母，增加字母概率 P1=5/886=0.56%，删除字母频率 P2=5/886=0.56%，修改字母频率 P3=6/886=0.68%.

最后得到的密文对应的明文为：

Nowadays,Chinese Dream has become a heated topic all across China.Unlike the celebrated American dream,it refers to a stronger and more prosperous China and the accomplishment of the well-being for the Chinese people.

As far as i am concerned,the pursuit for Chinese Dream combines both our nation's vision for the future and individual's dream.When it comes to China,we are poised to build it into a modernized country from a developing country.Throughout the modern history,our people and country have suffered from a series of miseries so that we have full reason to grow powerful to restore our glory in the ancient times.

As for the individual aspect,the current task for me is to study hard and master all sorts of knowledge which will be played its role in the modernized construction.i believe as strongly as ever our country and people will be rising only when we as a individual do our utmost to dedicate the energy as much as we can.

So that in my understanding of the Chinese Dream.By hard work and relentless determination and persist,this dream will be bound to come true.

5.1.3 模型的评价

优点：

1) 这种模型是通过逐渐找明文密文字母之间的对应关系的方法来获得密码表，通过字母频率分析、高频字母的结构化处理、任意文章进行同步推演、结构化单词间隔划分、部分密码表试解密这五种方法循环使用找到明文密文之间的对应关系，所以这种模型可以非常快速有效的破解任何单字母加密的密文，无论加密方式是多么的复杂。

2) 由于这种模型步步递推找匹配率高的明文密文对应关系，所以这种算法的效率和准确率非常高，程序的时间复杂度和空间复杂度特别低，时间复杂度和空间复杂度都是 o（n）。
3) 由于这种模型用了高频字母的结构化处理和结构化单词间隔划分的方法，所以对没有标点和间隔的密文特别有效。
4) 这种算法以频率统计为基础，寻找高频字母的结构化特点，所以噪声干扰（密文字母增删改）对这种算法的影响性非常小。
5) 但是由于此种算法是基于字母频率分析，所以对密文的长度有一定的限制，一般密文长度达到 200 个字母，正确率和效率就特别高。

5.2 隐含马尔可夫模型

5.2.1 模型的建立与实例

现在我们获取了一串有干扰的密文 $O_1O_2O_3$……，其密文长度为 T。最终我们破解密文得到的明文为 $s_1s_2s_3$……,现在就是要找到条件概率 P(s1,s2,s3,…|O1,O2,O3….)达到最大值的 s1,s2,s3,…。当然，这个概率不容易直接求出，于是我们可以间接地计算它。利用贝叶斯公式并且省掉一个常数项，可以把上述公式等价变换成 P(O1,O2,O3,…|s1,s2,s3….)*P(s1,s2,s3,…)其中 P(O1,O2,O3,…|s1,s2,s3….)表示 s1,s2,s3…被读成 o1,o2,o3,…的可能性,而 P(s1,s2,s3,…)表示 s1,s2,s3,…本身能够表达为一个有意义的信息的可能性。而

s_i 只由 s_{i-1} 决定，即 s1,s2,s3,…是一个马尔可夫链。
第 i 时刻的接收信号 oi 只由发送信号 si 决定。

表 22

由已知条件有

每个字符可能出现的情况	概率
传输过程中被丢失	$Q_1=P_1$
其后添加了一个随机字符	$Q_2=P_2$
被篡改为一个随机字符	$Q_3=P_3$
正常传输	$Q_0=P_1-P_2-P_3$

表 23

根据独立输出假设,即 P(o1,o2,o3,…|s1,s2,s3….)=P(o1|s1)*P(o2|s2)*P(o3|s3)…，总共就会出现 $(A_4^4)^T$ 种组合。那么我们就可以很容易利用算法 Viterbi 找出上面式子的最大值，进而找出要识别的明文 s1,s2,s3,…。这个过程可用格形图近似表示：

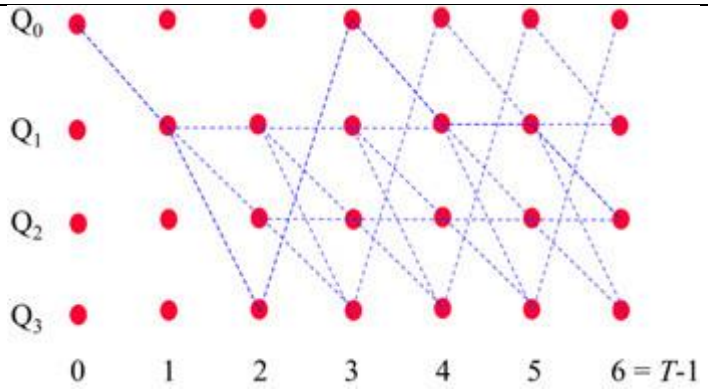

图 33

设$\alpha_t(i)$表示 t 时刻状态为 i 的所有前面状态路径的最大概率，则存在以下迭代关系：$\alpha_t(i)=\max_j(\alpha_{t-1}(j)*a_{ji}*b_{ik_t})$，$a_{ji}$表示从状态 j 转移到状态 i 的概率，b_{ik_t}表示状态 i 到观察状态 kt 的概率。时刻 t=1 时表示初始状态概率,这是求最大概率的过程。

5.2.2 模型的评价

优点：实现快速精确，双重随机过程----具有一定状态数的隐马尔可夫链和显示随机函数集。

不足：Viterbi 译码算法随着约束长度的增加算法的复杂度增加很快。

5.3 从替换式密码延伸的分组密码破解——DES 算法破解密码

5.3.1 模型的建立与实例

替换式密码属于古典密码，其中的一些只需使用铅笔和纸张的手动加密密码，都不再经常使用。然而，即使到了今天，替换加密的概念仍在进步，从一个够新奇的角度来看，替换式密码的存在以及不断发展促进了现代密码学的产生和发展。现代位元导向式的分组密码（如DES及AES）仍可视为使用大量二进制字母的替换加密。此外，分组密码通常包含较小的替换表，名为S-box，其同时包含逻辑异或算法。
我们查阅资料得到DES算法作为美国数据加密标准，是1972年美国IBM公司研制的对称密码体制加密算法。其密钥长度为56位，还有8位作为奇偶校验位，是一种用56位密钥来加密64位数据的方法。明文按64位进行分组，将分组后的明文组和56位的密钥按位替代或交换的方法形成密文组的加密方法。DES工作的基本原理是，其入口参数有三个:key、data、mode。 key为加密解密使用的密钥，data为加密解密的数据，mode为其工作模式。当模式为加密模式时，明文按照64位进行分组，形成明文组，key用于对数据加密，当模式为解密模式时，key用于对数据解密。实际运用中，密钥只用到了64位中的56位，这样才具有高的安全性。
由于在实际生活中信息的传输总是难免存在干扰，而替换式密码和分组密码同为对称密码，我们在接下来的文章中将要用到破解分组密码的DES方法来破解替换式密码。首先我们需要证明DES加密算法和解密算法是可逆的，进而通过对加密算法的研究得到解密算法，最后破解我们的替换式密码。证明：

$$T(L,R)=(R,L)$$
$$F_{Ki}(L,R)=(L+f(R,k_i),R)$$

则有：

$$T^2(L,R)=T(R,L)=I(L,R)$$
$$T=T^{-1}$$

同理：

$$F_{Ki}^{\ 2}(L,R)=F_{Ki}(L+f(R,k_i),R)$$

$$=(L+\ f(R,k_i)+\ f(R,k_i),R)$$

$$=(L,R)$$

有：

$$F_{Ki}=\ F_{Ki}^{-1}$$
$$(F_{Ki}T)(TF_{Ki})=\ F_{Ki}\ F_{Ki}=I$$

则有：

$$(T\ F_{Ki})^{-1}=\ F_{Ki}T$$

加密过程可以写成：

$$DES_k=（IP）^{-1}\ F_{K16}T\ F_{K15}T\cdots\cdots（IP）$$

解密过程可以写成：

$$DES_k^{\ -1}=(IP)^{-1}\ F_{K1}T\ F_{K2}T\cdots\cdots（IP）$$

进而可以得到：

$$（DES_k^{\ -1}）*（DES_k）=I$$

即：

DES加密算法和解密算法是可逆的。

我们将获取的密文分成以64个字符为单位的小单元，对每一个单元使用DES解密算法，尝试获得最后的明文。下面是DES解密算法执行过程：

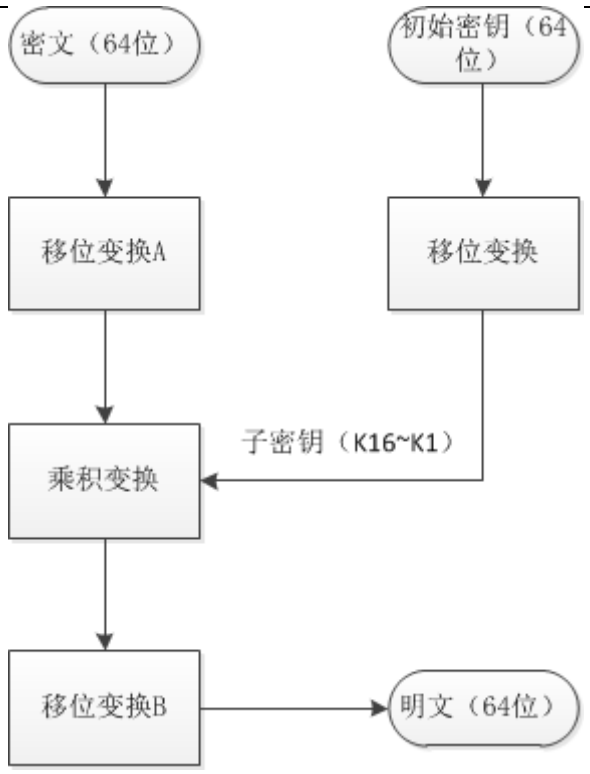

图34

在DES算法中，密文中一个字符仅由对应的明文以及该字符前面的一个字符所决定。我们将64位字符标号，将其中标号为：8,16,24,……64作为奇偶校验段，把剩下的56位字符平均分成两份，左边为

L_0:57,49,41,33,25,17,9,1,58,50,42,34,26,18,10,2,59,51,43,35,27,19,11,3,60,52,44,36

右边为R_0:

63,55,47,39,31,23,15,7,61,53,45,37,29,21,15,5,62,54,46,38,30,22,16,6,28,20,12,4

K_i会从64位经过若干次变化得到48位的密钥,每一轮迭代经过函数f变换异或和左右两边交换。在函数f中进行了8次S变换（置换表）和S-盒（交叉变换），而PC-1和PC-2循环移位产生子密钥，一次次的迭代得到明文。

5.3.2 模型的评价

优点：

1） 在密文规模比较大的时候，分成64位字符的单元是很合适的
2） 可以结合已知明文攻击和选择明文攻击来破译分组密码
3） 解密过程和加密过程是互逆的，在解密的时候可以综合加密过程进行
4） 对于密钥比较短小的情况可以使用穷举法直接攻击获取明文

不足：

1） 在缩小密钥空间的 时候时空复杂度可能会比较大

六、参考文献

1. 李志林，欧宜贵.数学建模及典型案例分析.化学工业出版社.[2007 年 4 月]
2. 吴建国主编.数学建模案例精编.中国水利水电出版社.[2005 年]
3. 赵静，但琦主编.数学建模与数学实验.高等教育出版社.[2008 年]
4. 司守奎，孙玺菁.数学建模算法与应用.国防工业出版社.[2014 年 9 月]
5. 韩中庚.数学建模方法及其应用.高等教育出版社. [2005 年]
6. 数学之美 系列三 -- 隐含马尔可夫模型在语言处理中的应用 http://www.360doc.com/content/11/0304/22/6140946_98202082.shtml
7. 解密算法_IT168 文库 http://wenku.it168.com/tag/103113_0_1.shtml
8. 传统密码学的基本原理. 正修科技大学资讯管理系. [2011-11-28].
9. 密码学原理与技术. 资讯安全宣导专区稻江科技暨管理学院. [2011-11-28].
10. ICCL-资讯密码暨建构实验室, 王旭正、柯宏叡.密码学与网络安全 [理论、实务与应用]. 博硕文化. 2011 年 [2004 年].ISBN 957-527-690-6.
11. Salomma, Arto. 1.2. public-key cryptography. 香港: 国防工业出版社. [1990].ISBN 7-118-01777-9.
12. 替换式密码 - 维基百科，自由的百科全书 http://zh.wikipedia.org/wiki/%E6%9B%BF%E6%8D%A2%E5%BC%8F%E5%AF%86%E7%A0%81
13. 文献穷举法_百度百科 http://baike.baidu.com/link?url=xCnsMVNO33q7wYAfcfzcYqM9XE1LLaZdlnu2vQYcU4bmXdEVD1cv46Qss7tV17xj2ntQWTfGv0EB_9tDn8HURa
14. 频率分析_百度百科 http://baike.baidu.com/view/2054214.htm?qq-pf-to=pcqq.discussion
15. Hill2 密码加密解密_百度文库 http://wenku.baidu.com/link?url=fsRthBj31TQQh1FCB740vyPMYbTKEDaxrKs_caajUeYpVorqPMpcpzfV9wyz-vx3SUL287NylkOZl9jZOjeLlPgEyJg3L4ZcHuVWflf0Tcu&qq-pf-to=pcqq.discussion

七、附录

代码

```
#include "stdafx.h"
#include "string.h"
//全局变量区
char aticle[5000];
char cipher[5000];
```

```
char randomarticle[5000];
//函数声明区
int readcipher();
int preparecipher();
int creatcipher();
int letterfrequency();
int findwordsstructure();
int RandomArticleStructure();
int RandomArticleletterfrequency();
int Useletterfrequency(char somewords[]);
int Usefindwordsstructure(char usearticle,char letter,int m,int n);
int AddBlankCipher();
int Trydecipher();
//函数区
int preparecipher()//准备密文，去标点空格，全变大写
{

    char ch;
    int i=0;
    int k=0;//忘记这个 k，用 i 弄错了好多次
    FILE *fp=fopen("a.txt","r");//打开文件
        if(!fp)
    {
        printf("can't open file\n");
        return -1;
    }
        ch= fgetc(fp);
        while(!feof(fp))
    {
        if((ch>='a'&&ch<='z')){
            aticle[k]=ch-32;
            k++;
        }
        else if((ch>='A'&&ch<='Z')){
            aticle[k]=ch;
            k++;
        }
        i++;
        ch= fgetc(fp);
    }
        aticle[k]='\0';
        printf("这段密文有%d 个字母\n",k);
        printf("%s",aticle);

        printf("\n");
    fclose(fp);
    return 0;
}

//创建密文
int creatcipher()//创建密文，去标点空格
{
    char ch;
    int i=0;
    FILE *fp=fopen("cipher.txt","w");
        if(!fp)
```

```
	{
		printf("can't open file\n");
		return -1;
	}
	fprintf(fp,aticle);
	fclose(fp);
	return 0;
}


//读获取的密文
int readcipher()//读获取的密文
{

	char ch;
	int i=0;
	FILE *fp=fopen("gotcipher.txt","r");//打开文件
		if(!fp)
	{
		printf("can't open file\n");
		return -1;
	}
		ch= fgetc(fp);
		while(!feof(fp))
	{
		cipher[i]=ch;
		i++;
		ch= fgetc(fp);
	}
		cipher[i]='\0';
		printf("需要破译的密文为\n\n");
		printf("%s\n",cipher);
		printf("\n");
		printf("这段密文有%d 个字母\n\n",i);
	fclose(fp);
	return 0;
}
//各个字母在密文中出现的频率
int letterfrequency(){
	char ch;
	int i;
	int j;
	int k;
	int sum=0;
	int charactertimes[30];
	int characternum=0;
	for(j=0;j<26;j++)
			charactertimes[j]=0;
	//字母计算频数概率
	for(i=0;cipher[i]!='\0';i++){
		ch=cipher[i];
		characternum++;
		if((ch>='A'&&ch<='Z')){
			charactertimes[ch-65]++;}
	}
	printf("密文中总共有字母%d 个\n\n",characternum);
```

```
	//之前这里 25 数组越界
	for(j=0;j<26;j++){
		printf("密文中字母%c 出现的次数%d\n",j+65,charactertimes[j]);
		sum=sum+charactertimes[j];
	}
	printf("\n\n\n");
	printf("这些字母出现的次数之和为%d\n\n",sum);
	printf("这些字母分别出现的概率为\n\n");
	for(j=0;j<26;j++){
		printf("密文中字母%c 出现的概率为%6.2f%%\n",j+65,100*(charactertimes[j])/(float)sum);

	}
	return 0;
}


//找到密文中出现频率高的字母对应的频率高的结构结构
int findwordsstructure()
{
	char wordstructureR[300][10];
	char sortwordstructureR[300][10];
	char sortwordstructureNum[300];
	char replaceword[10];
	int i=0;
	int j=0;
	int k=0;
	int structurenum=0;
	int changeline=0;
	int before3;
	int before2;
	int before1;
	int behind3;
	int behind2;
	int behind1;
	char ch;
	int characternum=0;
	for(i=0;cipher[i]!='\0';i++){
		ch=cipher[i];
		characternum++;
	}
	int state=0;
	//printf("%d\n",characternum);
	//printf("%s\n",cipher);
	printf("密文中形如**R 的结构有\n\n");
	for(i=5;i<characternum-5;i++){
		ch=cipher[i];
		//这里本来应该是等于的写成了赋值
		if(ch=='R'){
			changeline++;
			before3=i-3;
			before2=i-2;
			before1=i-1;
			behind3=i+3;
			behind2=i+2;
			behind1=i+1;
			printf(" %c%c%c    ",cipher[before2],cipher[before1],cipher[i]);
```

```
                replaceword[0]=cipher[i-2];
                replaceword[1]=cipher[i-1];
                replaceword[2]=cipher[i];
                replaceword[3]='\0';
                //printf("0000%s\n",replaceword);
                //**R 型的单词全部存进了 wordstructureR
                strcpy(wordstructureR[structurenum],replaceword);
                structurenum++;
                if(changeline%5==0)
                        printf("\n");
        }
    }
    /*for(i=0;i<structurenum;i++)
    printf("wordstructureR   %s   \n",wordstructureR[i]);*/
    printf("\n\n");


    //按字母顺序输出 wordstructureR 中的单词
    //printf("按字母排序后的单词单词\n");
    //sortwordstructureR 里面储存了排序后的单词
    printf("按字母顺序输出的**R 的结构的结构\n");
    k=0;
    for(i='A';i>='A'&&i<='Z';i++){
        for(j=0;j<structurenum;j++){
            if((wordstructureR[j][0]==i)){
                state=1;
                printf("%s   ",wordstructureR[j]);
                strcpy(sortwordstructureR[k++],wordstructureR[j]);}
        }
        if(state==1){
            puts("\n");
            state=0;
        }
    }
    printf("密文中总共有%d 个**R 结构的单词\n\n",structurenum);
    /*for(i=0;i<structurenum;i++)
    printf("sortwordstructureR   %s   \n",sortwordstructureR[i]);*/

    //用 sortwordstructureNum 记录每个**R 单词出现的次数，下标相对应，重复的记在第一次出现
的位置
    for(i=0;i<structurenum;i++){//初始化
        sortwordstructureNum[i]=0;
    }
    //后面的每一个和前面做比较，发现相同，前面加 1，发现不同，自己加 1
    //state 为 0 表示没加，等于 1 表示加了
    state=0;
    sortwordstructureNum[0]=1;
    for(i=1;i<structurenum;i++){
        for(j=0;j<i;j++){
            if(strcmp(sortwordstructureR[i],sortwordstructureR[j])==0){
                sortwordstructureNum[j]++;
                state=1;
                break;
            }
            else
                state=0;
```

```
    }
        if(state==0){
            sortwordstructureNum[i]++;
            state=1;
        }
    }
    for(i=0;i<structurenum;i++){
        if(sortwordstructureNum[i]!=0)
        printf(" 第 %3d 号 位 置 的 结 构 %s 出 现 了 %3d 次 \n",i,sortwordstructureR[i],sortwordstructureNum[i]);
    }


    //出现频率比较高的单词结构
    printf("\n 出现频率比较高的**R 结构\n\n");
    for(i=0;i<structurenum;i++){
        if((sortwordstructureNum[i]!=0)&&(sortwordstructureNum[i]>=4))
        printf(" 第 %3d 号 位 置 的 结 构 %s 出 现 了 %3d 次 \n",i,sortwordstructureR[i],sortwordstructureNum[i]);
    }
    return 0;
}
//和随便网上找的文章做对比
int RandomArticleStructure()
{
    char ch;
    int i=0;
    int k=0;//忘记这个 k，用 i 弄错了好多次
    FILE *fp=fopen("randomarticle.txt","r");//打开文件
        if(!fp)
    {
        printf("can't open file\n");
        return -1;
    }
        ch= fgetc(fp);
        while(!feof(fp))
    {
        if((ch>='a'&&ch<='z')){
            randomarticle[k]=ch-32;
            k++;
        }
        else if((ch>='A'&&ch<='Z')){
            randomarticle[k]=ch;
            k++;
        }
        i++;
        ch= fgetc(fp);
    }
        randomarticle[k]='\0';
        printf("这段随意找的文章有%d 个字母\n",k);
        printf("%s",randomarticle);

        printf("\n");
    fclose(fp);
    return 0;
}
```

```
int RandomArticleletterfrequency(){
	char ch;
	int i;
	int j;
	int k;
	int sum=0;
	int charactertimes[30];
	int characternum=0;
	for(j=0;j<26;j++)
			charactertimes[j]=0;
	//字母计算频数概率
	for(i=0;randomarticle[i]!='\0';i++){
		ch=randomarticle[i];
		characternum++;
		if((ch>='A'&&ch<='Z')){
			charactertimes[ch-65]++;}
	}
	printf("这段随意找的文章中总共有字母%d 个\n",characternum);
	//之前这里 25 数组越界
	for(j=0;j<26;j++){
		printf("这段随意找的文章中字母%c 出现的次数%d\n",j+65,charactertimes[j]);
		sum=sum+charactertimes[j];
	}
	printf("这些字母出现的次数之和为%d\n",sum);
	for(j=0;j<26;j++){
		printf("这段随意找的文章中字母%c 出现的概率为%6.2f%%\n",j+65,100*(charactertimes[j])/(float)sum);

	}
	return 0;
}

int Useletterfrequency(char somewords[]){
	char ch;
	int i=0;
	int j;
	int k;
	int sum=0;
	int charactertimes[30];
	int characternum=0;

	FILE *fp=fopen("frequency.txt","w");
		if(!fp)
	{
		printf("can't open file\n");
		return -1;
	}

	for(j=0;j<26;j++)
			charactertimes[j]=0;
	//字母计算频数概率
	for(i=0;somewords[i]!='\0';i++){
		ch=somewords[i];
		characternum++;
```

```
            if((ch>='A'&&ch<='Z')){
                charactertimes[ch-65]++;}
        }
        printf("密文中总共有字母%d 个\n",characternum);
        //之前这里 25 数组越界
        for(j=0;j<26;j++){
            printf("密文中字母%c 出现的次数%d\n",j+65,charactertimes[j]);
            sum=sum+charactertimes[j];
        }
        printf("这些字母出现的次数之和为%d\n",sum);
        for(j=0;j<26;j++){
            printf("密文中字母%c 出现的概率为%6.2f%%\n",j+65,100*(charactertimes[j])/(float)sum);
            fprintf(fp,"密文中字母%c 出现的概率为%6.2f%%\n",j+65,100*(charactertimes[j])/(float)sum);

        }
        printf("出现概率较低或者较高的单词\n\n");
        for(j=0;j<26;j++){
            if(((100*(charactertimes[j])/(float)sum)>0)&&((100*(charactertimes[j])/(float)sum)<1)){
                printf("密文中字母%c出现的概率为%6.2f%%\n",j+65,100*(charactertimes[j])/(float)sum);}
        }
        fclose(fp);
        return 0;
}

int Usefindwordsstructure(char usearticle[],char letter,int m,int n){
        char wordstructureR[300][10];
        char sortwordstructureR[300][10];
        char sortwordstructureNum[300];
        char replaceword[10];
        int i=0;
        int j=0;
        int k=0;
        int z=0;
        int y=0;
        int structurenum=0;
        int changeline=0;
        int before4;
        int before3;
        int before2;
        int before1;
        int behind4;
        int behind3;
        int behind2;
        int behind1;
        char ch;
        int characternum=0;
        printf("000000000%s\n",usearticle);
        for(i=0;usearticle[i]!='\0';i++){
            ch=usearticle[i];
            characternum++;

        }
        int state=0;
        printf("%d\n",characternum);;
        //按照实际情况输出
```

```
    printf("密文中形如");
    for(y=i,z=n-1;z>=1;z--){
                printf("*");
                y=y-1;
            }
    printf("%c",letter);
    for(y=i,z=n;z<=m-1;z++){
                printf("*");
                y++;
            }
    printf("的结构有\n");

    for(i=5;i<characternum-5;i++){
        ch=usearticle[i];
        //这里本来应该是等于的写成了赋值
        if(ch==letter){
            changeline++;
            before4=i-4;
            before3=i-3;
            before2=i-2;
            before1=i-1;
            behind4=i+4;
            behind3=i+3;
            behind2=i+2;
            behind1=i+1;
            //printf(" %c%c%c    ",cipher[before2],cipher[before1],cipher[i]);
            //y=i 放在了循环体里面，所以 Y 的值始终一样
            for(y=i,z=n-1;z>=0;z--){
                replaceword[z]=usearticle[y];
                y=y-1;
            }
            for(y=i,z=n;z<=m-1;z++){
                replaceword[z]=usearticle[y+1];//这里把 y+1 写成了 y++
                y++;
            }
            replaceword[m]='\0';
            //printf("0000%s\n",replaceword);
            //**R 型的单词全部存进了 wordstructureR
            strcpy(wordstructureR[structurenum],replaceword);
            structurenum++;
            if(changeline%5==0)
                printf("\n");
        }
    }
    printf("文中形如");
    for(y=i,z=n-1;z>=1;z--){
                printf("*");
                y=y-1;
            }
    printf("%c",letter);
    for(y=i,z=n;z<=m-1;z++){
                printf("*");
                y++;
            }
    printf("的结构有\n\n");
```

```
    /*for(i=0;i<structurenum;i++)
    printf("wordstructureR    %s    \n",wordstructureR[i]);
*/
    //按字母顺序输出 wordstructureR 中的单词
    //printf("按字母排序后的单词单词\n");
    //sortwordstructureR 里面储存了排序后的单词
    printf("按字母顺序输出特定结构排序后的单词\n");
    k=0;
    for(i='A';i>='A'&&i<='Z';i++){
        for(j=0;j<structurenum;j++){
            if((wordstructureR[j][0]==i)){
                state=1;
                printf("%s    ",wordstructureR[j]);
                strcpy(sortwordstructureR[k++],wordstructureR[j]);}
        }
        if(state==1){
            puts("\n");
            state=0;
        }
    }
    //按照实际情况输出
    printf("文中形如");
    for(y=i,z=n-1;z>=1;z--){
                printf("*");
                y=y-1;
            }
    printf("%c",letter);
    for(y=i,z=n;z<=m-1;z++){
                printf("*");
                y++;
            }
    printf("的结构有\n\n");


    /*for(i=0;i<structurenum;i++)
    printf("sortwordstructureR    %s    \n",sortwordstructureR[i]);*/

    //用 sortwordstructureNum 记录每个**R 单词出现的次数，下标相对应，重复的记在第一次出现的位置
    for(i=0;i<structurenum;i++){//初始化
        sortwordstructureNum[i]=0;
    }
    //后面的每一个和前面做比较，发现相同，前面加 1，发现不同，自己加 1
    //state 为 0 表示没加，等于 1 表示加了
    state=0;
    sortwordstructureNum[0]=1;
    for(i=1;i<structurenum;i++){
        for(j=0;j<i;j++){
            if(strcmp(sortwordstructureR[i],sortwordstructureR[j])==0){
                sortwordstructureNum[j]++;
                state=1;
                break;
            }
            else
                state=0;
    }
```

```
            if(state==0){
                sortwordstructureNum[i]++;
                state=1;
            }
        }
        for(i=0;i<structurenum;i++){
            if(sortwordstructureNum[i]!=0)
            printf(" 储 存 在 第 %3d 号 位 置 的 结 构 在 文 中 %s 出 现 了 %3d 次 \n",i,sortwordstructureR[i],sortwordstructureNum[i]);
        }


        //出现频率比较高的单词结构
        //按照实际情况输出
        printf("\n 文中出现频率比较高的");
        for(y=i,z=n-1;z>=1;z--){
                        printf("*");
                        y=y-1;
                    }
        printf("%c",letter);
        for(y=i,z=n;z<=m-1;z++){
                        printf("*");
                        y++;
                    }
        printf("的结构有\n\n");
        for(i=0;i<structurenum;i++){
            if((sortwordstructureNum[i]!=0)&&(sortwordstructureNum[i]>=4))
            printf(" 储 存 在 第 %3d 号 位 置 的 结 构 在 文 中 %s 出 现 了 %3d 次 \n",i,sortwordstructureR[i],sortwordstructureNum[i]);
        }
        return 0;


}
//加空格后的密文
int AddBlankCipher(){
        int k=0;
        int i=0;
        int state=0;
        char ch;
        char addblank[5000];
        char useaddblank[5000];
        //读入文件 addblankcipher1.txt 的内容
        FILE *fp=fopen("cipher.txt","r");//打开文件
            if(!fp)
        {
            printf("can't open file\n");
            return -1;
        }
            printf("加空格后的密文\n");
            ch= fgetc(fp);
            while(!feof(fp))
        {
                    addblank[k]=ch;
                    k++;
                    ch= fgetc(fp);
```

```
    }
        addblank[k]='\0';
        fclose(fp);
        //printf("addblank 有%d 个字母\n",k);
        printf("%s",addblank);
    //具体加空格过程
        state=0;
        for(i=0,k=0;addblank[i]!=EOF;i++){
            //printf("%c",addblank[i]);
            //六个单词

    if((addblank[i+1]=='L')&&(addblank[i+2]=='A')&&(addblank[i+3]=='D')&&(addblank[i+4]=='S')&&
(addblank[i+5]=='R')&&(addblank[i+6]=='H')&&(addblank[i+7]=='R')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;

            }

    if((addblank[i-6]=='L')&&(addblank[i-5]=='A')&&(addblank[i-4]=='D')&&(addblank[i-3]=='S')&&(a
ddblank[i-2]=='R')&&(addblank[i-1]=='H')&&(addblank[i]=='R')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;
            }

    if((addblank[i+1]=='L')&&(addblank[i+2]=='V')&&(addblank[i+3]=='N')&&(addblank[i+4]=='S')&&
(addblank[i+5]=='K')&&(addblank[i+6]=='E')&&(addblank[i+7]=='Z')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;

            }

    if((addblank[i-6]=='L')&&(addblank[i-5]=='V')&&(addblank[i-4]=='N')&&(addblank[i-3]=='S')&&(a
ddblank[i-2]=='K')&&(addblank[i-1]=='E')&&(addblank[i]=='Z')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;
            }
            //六个单词

    if((addblank[i+1]=='P')&&(addblank[i+2]=='V')&&(addblank[i+3]=='O')&&(addblank[i+4]=='R')&&
(addblank[i+5]=='E')&&(addblank[i+6]=='S')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;

            }

    if((addblank[i-5]=='P')&&(addblank[i-4]=='V')&&(addblank[i-3]=='O')&&(addblank[i-2]=='R')&&(a
ddblank[i-1]=='E')&&(addblank[i]=='S')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;
            }
            //五个单词
```

```
    if((addblank[i+1]=='O')&&(addblank[i+2]=='E')&&(addblank[i+3]=='R')&&(addblank[i+4]=='F')&&
(addblank[i+5]=='P')){
                    useaddblank[k++]=addblank[i];
                    useaddblank[k++]=' ';
                    state=1;

                }

    if((addblank[i-4]=='O')&&(addblank[i-3]=='E')&&(addblank[i-2]=='R')&&(addblank[i-1]=='F')&&(a
ddblank[i]=='P')){
                    useaddblank[k++]=addblank[i];
                    useaddblank[k++]=' ';
                    state=1;
                }
                //四个单词


    if((addblank[i+1]=='K')&&(addblank[i+2]=='A')&&(addblank[i+3]=='F')&&(addblank[i+4]=='K')){
                    useaddblank[k++]=addblank[i];
                    useaddblank[k++]=' ';
                    state=1;

                }

    if((addblank[i-3]=='K')&&(addblank[i-2]=='A')&&(addblank[i-1]=='F')&&(addblank[i]=='K')){
                    useaddblank[k++]=addblank[i];
                    useaddblank[k++]=' ';
                    state=1;
                }

    if((addblank[i+1]=='P')&&(addblank[i+2]=='V')&&(addblank[i+3]=='E')&&(addblank[i+4]=='R')){
                    useaddblank[k++]=addblank[i];
                    useaddblank[k++]=' ';
                    state=1;

                }

    if((addblank[i-3]=='P')&&(addblank[i-2]=='V')&&(addblank[i-1]=='E')&&(addblank[i]=='R')){
                    useaddblank[k++]=addblank[i];
                    useaddblank[k++]=' ';
                    state=1;
                }

    if((addblank[i+1]=='L')&&(addblank[i+2]=='V')&&(addblank[i+3]=='P')&&(addblank[i+4]=='R')){
                    useaddblank[k++]=addblank[i];
                    useaddblank[k++]=' ';
                    state=1;

                }

    if((addblank[i-3]=='L')&&(addblank[i-2]=='V')&&(addblank[i-1]=='P')&&(addblank[i]=='R')){
                    useaddblank[k++]=addblank[i];
                    useaddblank[k++]=' ';
                    state=1;
                }
```

```
    if((addblank[i+1]=='U')&&(addblank[i+2]=='E')&&(addblank[i+3]=='V')&&(addblank[i+4]=='P')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;

            }

    if((addblank[i-3]=='U')&&(addblank[i-2]=='E')&&(addblank[i-1]=='V')&&(addblank[i]=='P')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;
            }

            //三个单词
            if((addblank[i+1]=='L')&&(addblank[i+2]=='F')&&(addblank[i+3]=='S')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;

            }
            if((addblank[i-2]=='L')&&(addblank[i-1]=='F')&&(addblank[i]=='S')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;
            }
            if((addblank[i+1]=='K')&&(addblank[i+2]=='A')&&(addblank[i+3]=='R')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;

            }
            if((addblank[i-2]=='K')&&(addblank[i-1]=='A')&&(addblank[i]=='R')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;
            }
            if((addblank[i+1]=='F')&&(addblank[i+2]=='S')&&(addblank[i+3]=='O')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;

            }
            if((addblank[i-2]=='F')&&(addblank[i-1]=='S')&&(addblank[i]=='O')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;
            }
            //三个单词
            if((addblank[i+2]=='T')&&(addblank[i+3]=='R')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;

            }
            if((addblank[i-1]=='T')&&(addblank[i]=='R')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
```

```
            state=1;
        }
        if((addblank[i+2]=='F')&&(addblank[i+3]=='H')){
            useaddblank[k++]=addblank[i];
            useaddblank[k++]=' ';
            state=1;

        }
        if((addblank[i-1]=='F')&&(addblank[i]=='H')){
            useaddblank[k++]=addblank[i];
            useaddblank[k++]=' ';
            state=1;
        }
        if((addblank[i+2]=='D')&&(addblank[i+3]=='S')){
            useaddblank[k++]=addblank[i];
            useaddblank[k++]=' ';
            state=1;

        }
        if((addblank[i-1]=='D')&&(addblank[i]=='S')){
            useaddblank[k++]=addblank[i];
            useaddblank[k++]=' ';
            state=1;
        }
        if((addblank[i+2]=='D')&&(addblank[i+3]=='K')){
            useaddblank[k++]=addblank[i];
            useaddblank[k++]=' ';
            state=1;

        }
        if((addblank[i-1]=='D')&&(addblank[i]=='K')){
            useaddblank[k++]=addblank[i];
            useaddblank[k++]=' ';
            state=1;
        }
        if((addblank[i+2]=='H')&&(addblank[i+3]=='N')){
            useaddblank[k++]=addblank[i];
            useaddblank[k++]=' ';
            state=1;

        }
        if((addblank[i-1]=='H')&&(addblank[i]=='N')){
            useaddblank[k++]=addblank[i];
            useaddblank[k++]=' ';
            state=1;
        }
        if((addblank[i+2]=='H')&&(addblank[i+3]=='N')){
            useaddblank[k++]=addblank[i];
            useaddblank[k++]=' ';
            state=1;

        }
        if((addblank[i-1]=='H')&&(addblank[i]=='N')){
            useaddblank[k++]=addblank[i];
            useaddblank[k++]=' ';
            state=1;
        }
```

```
            if((addblank[i+2]=='K')&&(addblank[i+3]=='D')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;

            }
            if((addblank[i-1]=='K')&&(addblank[i]=='D')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;
            }
            if((addblank[i+2]=='V')&&(addblank[i+3]=='U')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;

            }
            if((addblank[i-1]=='V')&&(addblank[i]=='U')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;
            }
            if((addblank[i+2]=='K')&&(addblank[i+3]=='V')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;

            }
            if((addblank[i-1]=='K')&&(addblank[i]=='V')){
                useaddblank[k++]=addblank[i];
                useaddblank[k++]=' ';
                state=1;
            }
            if(state==0){
                useaddblank[k++]=addblank[i];
                state=1;
            }
            state=0;

        }
        useaddblank[k]='\0';
        printf("\n\n\n 加了空格的密文为\n");
        printf("\n%s\n",useaddblank);

        //写入文件 addblankcipher2.txt

    fp=fopen("cipher2.txt","w");
        if(!fp)
    {
        printf("can't open file\n");
        return -1;
    }
    fprintf(fp,useaddblank);

    printf("\n");
    fclose(fp);
    return 0;
```

```
}
int Trydecipher()
{
	int k=0;
	int i=0;
	int state=0;
	char ch;
	char readaddblankcipher2[5000];
	char writeaddblankcipher1[5000];
	char trydecipher[5000];
	int m;
	int n;
	//读入文件 addblankcipher2.txt 的内容,(加空格程序已经修改的内容)
	FILE *fp=fopen("cipher2.txt","r");//打开文件
		if(!fp)
	{
		printf("can't open file\n");
		return -1;
	}
		printf("加空格后的密文\n");
		ch= fgetc(fp);
		while(!feof(fp))
	{
			readaddblankcipher2[k]=ch;
			k++;
			ch= fgetc(fp);
	}
		readaddblankcipher2[k]='\0';
		printf("已知部分密码试编译要用的加了空格的密文\n");
		printf("%s\n",readaddblankcipher2);
		fclose(fp);
	//部分密码试编译
	//已知部分密码试编译
	printf("已知部分密码试编译\n\n");
	for(i=0;readaddblankcipher2[i]!=EOF;i++){
		if(readaddblankcipher2[i]=='F')
			trydecipher[i]='a';
		else if(readaddblankcipher2[i]=='I')
			trydecipher[i]='b';
		else if(readaddblankcipher2[i]=='L')
			trydecipher[i]='c';
		else if(readaddblankcipher2[i]=='O')
			trydecipher[i]='d';
		else if(readaddblankcipher2[i]=='R')
			trydecipher[i]='e';
		else if(readaddblankcipher2[i]=='U')
			trydecipher[i]='f';
		else if(readaddblankcipher2[i]=='X')
			trydecipher[i]='g';
		else if(readaddblankcipher2[i]=='A')
			trydecipher[i]='h';
		else if(readaddblankcipher2[i]=='D')
			trydecipher[i]='i';
```

```
            else if(readaddblankcipher2[i]=='M')
                trydecipher[i]='l';
            else if(readaddblankcipher2[i]=='P')
                trydecipher[i]='m';
            else if(readaddblankcipher2[i]=='S')
                trydecipher[i]='n';
            else if(readaddblankcipher2[i]=='V')
                trydecipher[i]='o';
            else if(readaddblankcipher2[i]=='Y')
                trydecipher[i]='p';
            else if(readaddblankcipher2[i]=='E')
                trydecipher[i]='r';
            else if(readaddblankcipher2[i]=='H')
                trydecipher[i]='s';
            else if(readaddblankcipher2[i]=='K')
                trydecipher[i]='t';
            else if(readaddblankcipher2[i]=='N')
                trydecipher[i]='u';
            else if(readaddblankcipher2[i]=='T')
                trydecipher[i]='w';
            else if(readaddblankcipher2[i]=='Z')
                trydecipher[i]='y';
            else if(readaddblankcipher2[i]=='C')
                trydecipher[i]='z';

            else
                trydecipher[i]=readaddblankcipher2[i];
        }
        trydecipher[i]='\0';
        printf("已经部分密码表试解密的密文为\n\n");
            printf("%s\n",trydecipher);

        //把试解密得到的结果写入 addblankcipher1.txt

        fp=fopen("cipher3.txt","w");
            if(!fp)
        {
            printf("can't open file\n");
            return -1;
        }
        fprintf(fp,trydecipher);

        printf("\n");
        fclose(fp);

        return 0;
}
int _tmain(int argc, _TCHAR* argv[])
{
        int i=0;
        preparecipher();
        creatcipher();
        readcipher();
        letterfrequency();
        findwordsstructure();
        RandomArticleStructure();
```

```
    RandomArticleletterfrequency();
    Useletterfrequency(cipher);
    Usefindwordsstructure(cipher,'A',2,1);
    AddBlankCipher();
    Trydecipher();


    scanf("%d",&i);
    return 0;
}
```