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：1386 队

参赛队员　(签名)　：

队员 1：

队员 2：

队员 3：

参赛队教练员 (签名)：　数模指导小组

参赛队伍组别（例如本科组）：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：（请各个参赛队提前填写好）：

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　替换式密码自动化破译算法

关 键 词　频率攻击 时间复杂度 单字母加密 自动化破译

摘　　要：

古典密码是密码学的起源，它是由基于字符的密码算法构成，可用并机械操作实现加解密。目前解密行之有效的方法则是频率分析法，但是利用传统频率分析法，拥有运算量大，计算时间长，需要过多的人工干预等缺点。所以本文提出一种创新的全自动解密算法，对单字母加密方法的密文进行解密。

模型一：我们对传统频率法进行改进，建立了一种新的全自动化解密模型，不但能大量减少运算时间，而且可精确高效破译密文。

首先我们在模型准备中统计出英文固有的各种频率数据，并建立给定字母长度 15-24 的长单词库群和字母长度 1-8 字典库群，使模型可实现单词的快速搜索和匹配。模型一的算法分为三个模块：（1）频率攻击：先统计密文中单字母、双字母组合频率并分组，再对分组频率进行匹配，从而确定最高频字母 *t, h, e*。（2）长单词攻击：先将已破译的少量字母代入密文，再从中提取最长的单词代入相应长度的长单词字典库进行匹配，从而逐步破译更多密钥字母。因单词越长，数目越少，故匹配成功后正确率很高。（3）动态逐词攻击：先将已破译密钥字母代入密文，并对密文进行以单词为单位的扫描。再将单词逐个在字典库中进行匹配，得到相似单词组；其次累加记录每个相似单词中匹配的字母密钥，直至获取充足信息量；最终选取累计次数最高的匹配为最终密钥，破译出完整密钥。经多实例统计验证，该解密算法破译大约 5 万字母数的文章，用时仅 2.75s。

模型二：为评价模型一中解密算法模型的破译能力，我们从以下两个方面进行考虑：（1）算法破译的准确性：通过统计分析由全自动解密模型算法破译的多篇加密文章，得出密钥字母正确个数稳定在 20 个左右，密钥字母的正确率均高于 80%，全文平均字母正确率稳定在 96%；（2）算法破译密文的时间复杂度：首先结合算法三个模块程序的复杂度分析得，时间复杂度 $F(n)$和密文总字母数 n 的关系为 $F(n)=(C2+2)n+C1$，简记 $O(n)$，其中 $C1, C2$ 是依赖于字典库总词数的可变常数。其次因为模型算法只需要提取文章中前面部分单词，便可破译出≥24 个密钥，文章中剩余单词量并不影响算法破译时间，所以文章越长，单位时间内的破译效率越高。

综上所述，该模型能高效准确的解密出密文，算法运行速度快耗时极短，克服了传统频率分析法耗时长，运算效率不高的缺点。

另外，我们还对模型的改进和推广方向以及优缺点进行了讨论。

参赛队号： 1386

所选题目： B　**题**

参赛密码 _____

（由组委会填写）

Abstract

Decryption method is frequency analysis, but the use of traditional frequency analysis method, with large computation, the computation time is long, shortcomings and so on need too much human intervention.So this paper proposes an innovative decryption algorithm, encryption method of single letter the encrypted cryptograph.

Model 1: We have to improve the traditional method, a new automatic decryption model is established, not only can substantially reduce computing time, and can be accurate and efficient decoding ciphertext. First we have the data needed to measure out the model of the model, the letters length of 15 to 24 long words dictionary library 1-8 of the dictionary, in the model 1 is divided into three steps: (1) the frequency of cipher attack first, and the statistical frequency of single letters, double letter combinations and grouping, and then grouped frequency matching, analysis of basic to determine the highest frequency of the letter e, he, th, to determine t, h, e, (2) long words on ciphertext attacks, namely in the crack of a small amount of high frequency letters after generation into the ciphertext, extract the longest word in long words dictionary library matching gradually deciphering secret key more letters.Because of the longer the words, the less the number, so after the success of the matching accuracy is higher.(3) the last will be the key letters substitution cipher, and the cipher text with the word for the unit of the scan.Match the words in a dictionary in the library one by one, get the similar words;Accumulated each similar word matching letters secret key again, until the record number of secret key letter > = 24, eventually to every secret key letters, select the highest cumulative number of matching as the ultimate secret key, output the complete key.Verified by multi-instance statistics, the decryption algorithm to decipher the number of articles about 50000 letters, it only needs 2.75 s.

Model 2: To evaluate the model a decryption algorithm decoding ability of the model, we from the following two aspects: (1) considering the accuracy of the algorithm decoding;Tested much of the article, through the Matlab statistic can be concluded that the key letters correct number stability in 20 , the key letters correctly about 80%, average letters in full accuracy of about 96% (2) break the time complexity of cipher algorithm.The less as the number of letters, decoding time grows with the increase of the number of letters.While the number of letters after reaching a certain threshold, decoding time tends to be stable.Because model algorithm only need to extract the partial words in the article the amount of information that can be decoded key.Article the rest of the word does not affect the algorithm decoding time, so the longer the articles, per unit time decoding efficiency is higher.

To sum up, this model can efficiently accurately to decrypt the ciphertext, algorithm fast time shortly, overcomes the traditional frequency analysis method time-consuming, the disadvantage of operation efficiency is not high.In addition, we also the improvement and popularization of direction as well as the advantages and disadvantages of the model are discussed.

Keywords: *Frequency attacks，Time complexity，Single letter encryption，Automatic decipher*

一、问题重述

1.1 背景

当今社会处于信息时代，随着计算机互联网络的高速发展，整个世界被越来越紧密地联系在一起。资讯的发达在为人类社会带来巨大便利的同时，也产生了一系列严重的问题，信息安全就是其中突出的一个。由于网络中存储着许多重要、敏感数据，因此，信息安全已经越来越受到人们的关注。为确保信息的安全性，人们在技术上主要采用密码学的方法。

密码学是研究如何进行密写及如何解密的学科，它是一个高度专业化的研究领域，对数据的传送和通信安全有着重大的意义。因此，作为一门新兴学科，引起了数学家和计算机工作者的日益关注。

1.2 问题的提出

假设明文是由现代通常所使用的英语写成的。现在我们从题中所给数据库获取一些由单字母加密方法加密的密文。

1.根据题目的要求建立合理的数学模型，设计一个解密的算法，可以自动化地实现破译密文。

2.在确定算法并实施算法之后，设计一个衡量破译能力的标准，来评价破译算法的破译能力。

二、模型假设

① 为了简化问题，本文假设密码表仅是针对26个字母的，每个单词之间的空格，以及标点符号仍然会保留。

② 在长单词攻击时，假设被攻击的长单词均为名词，因此不考虑动词因时态而发生的长度变化。（如动词*ing*形式和*ed*形式）

③ 假设在模型一中双字母的低频组合忽略不计。

三、符号说明

符号	符号说明
f	表示字母在英文语料库中出现的频率
i	表示26个字母按照字母表顺序的第i个字母
Φ	表示密文转化为明文的算法
Φ^{-1}	表示明文转化为密文的算法
W_i	表示密钥中第i个字母

w_i	表示W_i所对应的明文
C	表示字母密钥正确率
C_i	表示i个密钥是否正确，用0,1区分正确与否
P	表示全文字母数正确率
Q	表示所有正确的字母个数
Z	表示全文中所有单词数
$F(n)$	表示解密算法的时间复杂度

四、模型的建立与求解

4.1 模型一的分析与求解

4.1.1 模型一的分析

在密码学中，有许多的的编制密钥的方法，其中较为简单的是替换式密码，即将文中出现的字符一对一地替换成其它的符号。虽然频率分析是现在单字母替换密码最常用的破译方法，但是频率分析存在许多缺点，如运算量大，计算时间长，需要过多的人工干预等。

因此我们需要在传统的频率法基础上进行改进，建立新的解密模型，使其能够自动化解密，减少运算时间，更加精确高效破译密文（模型步骤见下图）。

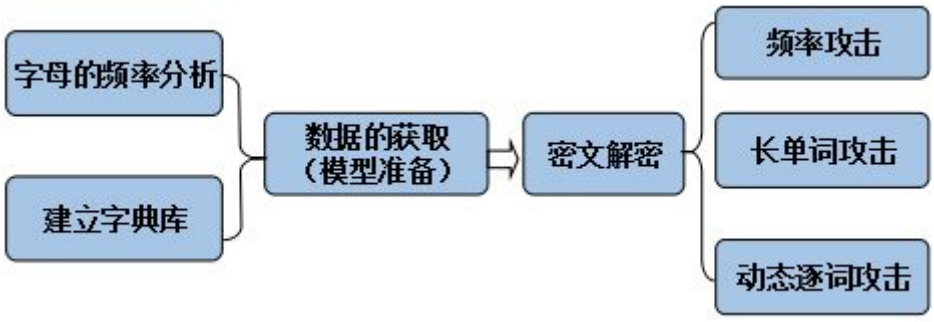

图 1：模型一的问题分析图

4.1.2 模型的准备

（一）首先在题目中所给*COCA*语料库[1]中下载所有的参考文章，将超过百篇英文文章合并。我们在语料库中统计长度$l(15 \le l \le 24)$的单词，建立对应单词长度为l的长单词字典库，为模型算法中长单词攻击做准备。同时，我们再选用一本常用英文字典[3](大小630*kb*)，建立常用英文单词字典库（后称字典库）。

并且，将字典库也拆分成单词长度为$l(1 \le l \le 8)$的字典库。

（二）在语料库中, 对 26 个英文字母分别进行测频，可以发现各字母出现的频率非常稳定，且频率如下：

表 1：26 个英文字母在英语语料库中出现的频率

A	*B*	*C*	*D*	*E*	*F*	*G*
0.0822	0.0158	0.0296	0.0383	0.1217	0.0210	0.0216
H	*I*	*J*	*K*	*L*	*M*	*N*
0.0524	0.0719	0.0021	0.0090	0.0413	0.0255	0.0696
O	*P*	*Q*	*R*	*S*	*T*	*U*
0.0768	0.0211	0.0010	0.0612	0.0664	0.0908	0.0275
V	*W*	*X*	*Y*	*Z*	-	-
0.0108	0.0202	0.0018	0.0194	0.0011	-	-

（三）将以上字母的出现频率进行分组，得到以下英文字母频率分组表。

表 2：各英语字母频率分组表

极高频率字母组 (0.12)	*E*
次高频字母组 (0.06 ~ 0.09)	*T A O I N S H R*
中等频率字母组 (0.04)	*D L*
低频率字母组 (0.015 ~ 0.03)	*C* U M W F G Y P B
甚低频率字母组 (< 0.01)	*V K* J X Q Z

（四）将字母两两进行组合，在语料库中对双字母组合的统计测频（测试字母数近 2000 万）。统计得到双字母频率表。由于双字母组合情况过多（26×26 种）并且低频组合频率接近，我们只考虑高频双字母组合，绘制如下双字母组合频率前十名柱状图：

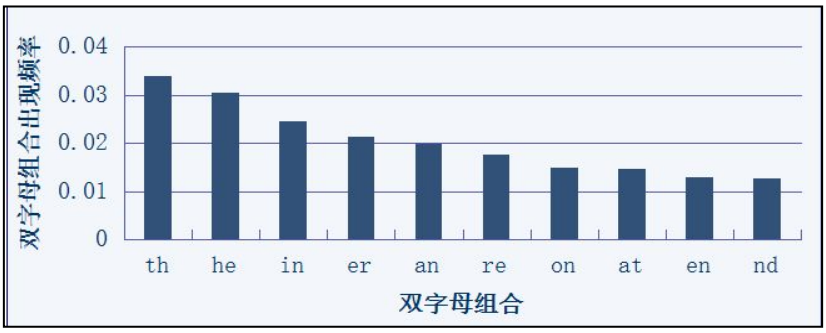

图 2：双字母组合频率前十名

根据上图我们将双字母组合也进行频率分类，得到双字母组合的频率分组（表 3）。

表 3：双字母组合的频率分组

极高频率双字母组(0.03)	*th he*
次高频双字母组(0.015～0.025)	*in er an re*
中频率双字母组(0.015)	*on at en nd*

（五）最后，在语料库中，统计出 3 个字母单词的出现次数，得到 3 个字母单词出现次数的前七位（测试基数为10000）。

表 4：3 个字母单词的出现次数（单位：次）

the	*and*	*for*	*are*	*was*	*not*	*but*
3542	1665	479	378	351	330	216

由上表分析，可以得出 *the* 和 *and* 这两个单词的出现频率为 35.42% 和 16.65%，所占的比例相对较大，因此 t,h,e 和 a,n,d 容易被破译。

4.1.3 模型的建立

通过以上模型准备，我们建立了长单词字典库（$15 \le l \le 24$）和英文单词字典库（$1 \le l \le 8$），并将长单词字典库按照字母的个数进行划分，如单位长度为 15 的划分成一个组。另外对英语单词字典库也进行了同样的分组，这样可以方便快捷地进行单词的匹配。模型一的准备也对密文破译和模型建立提供数据支持。我们将模型分为三部分，步骤流程图如下：

图 3：模型一流程图

（一）频率攻击

假设以W_i表示密钥中第i个字母，以w_i表示W_i所对应的明文，其中$i=1,2,3...26$，表示26个英文字母。令

$$\Phi=\begin{bmatrix} W_1 & W_2 & \cdots & W_{26} \\ w_1 & w_2 & \cdots & w_{26} \end{bmatrix},\quad \Phi^{-1}=\begin{bmatrix} w_1 & w_2 & \cdots & w_{26} \\ W_1 & W_2 & \cdots & W_{26} \end{bmatrix}$$

其中Φ表示的是密文转化成明文的算法，Φ是可逆的，Φ^{-1}是明文转化成密文的算法。针对此问题我们则建立解密模型步骤如下：

Step 1：首先从密文本身出发，首先统计出密文中所有字母的频率及两两字母组合的频率，并将按照频率的高低进行分组；

Step 2：统计频率分组之后，将模型准备中相应的频率进行匹配，自然密文中字母频率最高的3个及双字母组合频率最高的3个对应明文中出现频率最高的3个字母和双字母组合，这样就可以确定出密文中出现频率较高的一部分字母，不妨假设字母a,e,i,o最先被破译；

频率攻击流程图如下：

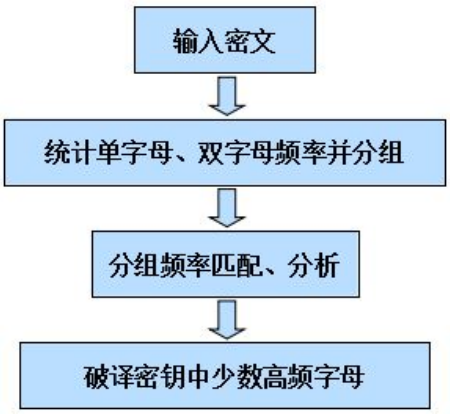

图 4 ：频率攻击流程图

（二）长单词攻击

通过以上方法只能匹配并确定少量字符，因而在此基础上，我们设计了一种长单词攻击算法。

Step 3：首先我们再次对密文进行扫描，找出密文中最长的单词长度记为L，最长的单词是*ksddhfkrymrbh*；假设频率攻击已经将$y\to a,\mathrm{h}\to e,\mathrm{r}\to i,\mathrm{s}\to o,$的配对破译。

Step 4：最长的单词经过之前的匹配破译就是$*o**e**ia*i*e$，其中*号代表的字符是没有被破译的。

Step 5：然后筛选出长度为L的长单词字典库中所有符合$*o**e**ia*i*e$规律的单词。（另外，因为长单词字典库内单词量极少，所以符合这种规律的单词数量较少，可能几十个，但对于长单词字典库来说只是小数目，可以大大节省搜索比对时间。）

Step 6：再针对这几十个备选单词，进行字母位置规律的匹配。我们利用 *ksddhfkrymrbh* 中 *dd* 字母位置及其他字母的重复性，筛选找到唯一匹配成功的单词 *commercialize*，也就成功破译出相应的配对 $k \to c, d \to m$..... 至此最长单词破译攻击完成，根据此方法我们就破译了更多的密钥配对。

长单词攻击流程图如下：

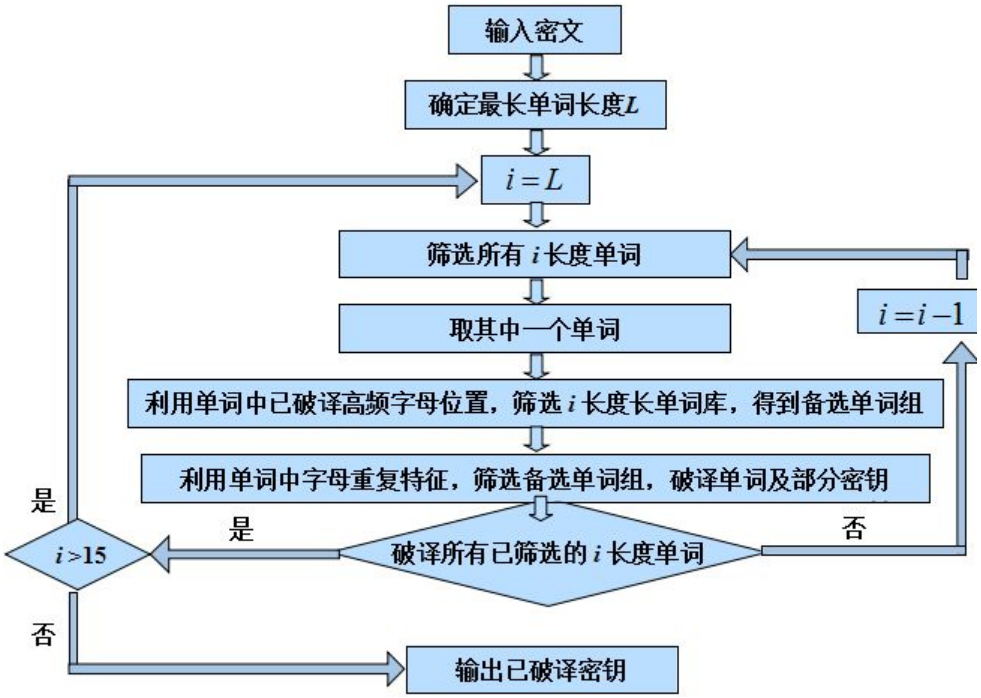

图 5：长单词攻击流程图

（三）动态逐词攻击

Step 7: 由长单词攻击破译出的 11 个（例文到此步破译出 11 个，个数不定）密钥回代到密文中，对密文以单词为单位进行扫描，将提取的每一个单词代入英语字典库进行比对，得到该单词的相似单词组；然后，累计记录相似单词与原密文单词中对应位置的密钥匹配字母对，直至已记录的密钥字母>=24 时，停止对密文的扫描(因为最低频字母出现概率极低，找出极低频单词对算法的时间成本影响过大，所以不扫描后面的字母)。最后，对已记录的每个密钥字母对，选取其累计次数最高的匹配为最终密钥。

Step 8: 最后将剩余的 2 个字母相互交换位置或保持不变，得到完全密钥，通过以上密钥便可对密文进行完全破译。

动态逐词攻击流程图如下：

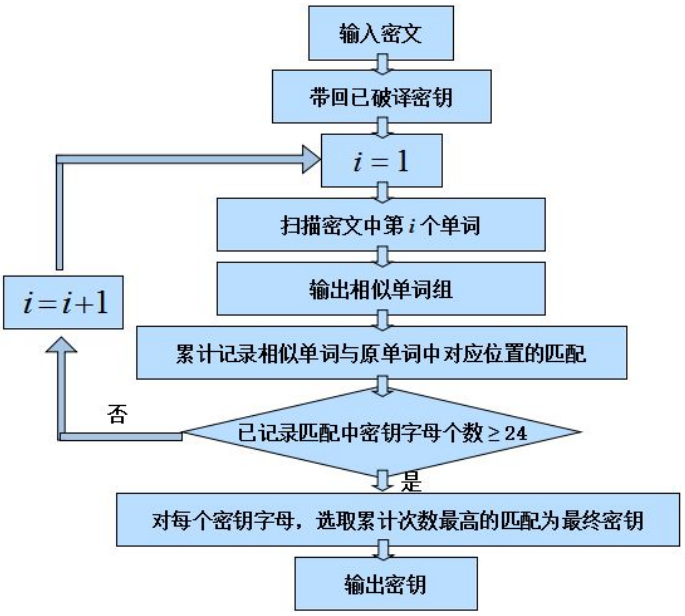

图 6：动态逐词攻击流程图

4.1.4 模型的求解

我们选取所给语料库中的一篇英语文献作为例子，先将这篇文章进行随机加密，然后通过设计出来的算法进行解密。

Step 1: 首先统计密文中单字母和双字母的出现频率，再将其高频部分(前三)与模型准备中统计频率里的高频部分进行匹配(如表 5)，能够确定出字母 t,h,e 。

表 5：明文与密文高频单、双字母对照表

明文中出现频率最高的				密文中出现频率最高的			
双字母	*th*	*he*	*in*	双字母	*ou*	*od*	*gf*
单字母	*e*	*t*	*h*	单字母	*d*	*g*	*j*

如上表所示，首先在明文中扫描出单字母频率最高的是 e，双字母频率前两位的是 th,he，而 e 和 he 中的 e 互相对应相等，那么所对应的密文也应该是相同位置上的字母对应相等，然后可以依次推导出字母 t,h 即得到密钥字母对应规则。$u \to t, o \to h, d \to e$ 。

Step 2: 其次搜索密文中最长的单词（记长度为 L ）进行长单词攻击，将以确定的字母 t,h,e 代到密文中进行替换，对长单词已经进行替换的位置进行标记。例如：对单词密文 *tdgwudqgwwdfwnorjjdf* 进行已破译高频字母密钥，替换变成 **e**te****e*******e** 。

再提取该单词与长单词词库中同长度的英语单词进行比对，必须满足 **e**te****e*******e** 词形式的单词被筛选出来，由于可匹配的长单词个数较少且比对位置精确因而能直接替换出其他 * 号的部分。

Step 3：然后依次对长度 $L-1$，$L-2...$，15 的单词重复 step 2 的破译过程，大致进行 4－5 次这样的循环。在例文中已经破解出密钥中 11 个对应字母如下（第一排为明文，第二排问密文，进行一一对应）

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*
r		*n*		*d*	*j*	*t*	*o*	*g*					*f*					*w*	*u*		*q*					

Step 4: 然后将 11 个已破译的字母代回到原来的密文，得到未完全破译的单词如：*hZYe*。遍历文中所有未完全破译的单词，将每一个未完全破译的单词代入字典库里进行匹配。

EX：将满足 *h**e* 形式的单词放入英文字典中进行比对，可以得到以下单词 *huge*，*hide*，*hope*，*have*，*hive*，*hire*，*here*，*home* 等等。

假设单词从左至右的字母位置的顺序为1,2,3,4，则*号的位置为2,3，通过匹配可以得到以下的多种对应方式：

表 6：*he* 形式单词的明/密文转换**

单词	密文换明文	密文换明文
hive	$Z \to i$	$Y \to v$
hire	$Z \to i$	$Y \to r$
have	$Z \to a$	$Y \to v$
here	$Z \to e$	$Y \to r$

从上表可以看出 *h**e* 符合这种规律的单词可以有多种组合形式。在完成所需信息量单词匹配后通过密文转化为明文的累积总次数进行分析：

表 7：明文和密文个字母之间转换

密文换明文	出现的次数
$Z \to i$	1876 次
$Z \to a$	543 次
$Z \to e$	382 次
$Y \to v$	3572 次
$Y \to r$	496 次

于是我们得到这个单词 *hZYe* 在位置 2,3 上所有对应字母的频率，根据出现次数可以基本判定密文 *Z* 对应的明文 *i* 和密文 *Y* 对应的明文 *v*。依次类推我们遍历全文所有未完全破译的单词，将未能破译的字母进行频率的分析，出现次数最高的解密方式则是该字母的密钥。进而可以确定具体的单词。

4.1.3.1 解密模型结论分析

通过本文建立的解密算法进行对加密的文章进行解密，并且在加密的过程中自动将原文中的大写字母转化为小写字母。

（1）以一篇英文文章为例：首先通过随机数列生成密钥字母对照表。例如：

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*
f	*n*	*h*	*i*	*d*	*s*	*y*	*g*	*l*	*p*	*b*	*o*	*u*	*t*	*v*	*c*	*z*	*k*	*e*	*w*	*a*	*x*	*q*	*r*	*j*	*m*

（2）然后对原文以一一对应的方式进行加密，图 8 为原文即明文，图 9 为加密后的文章即密文，我们截取了一部分进行对比:

When Allan Bloom wrote The Closing of the American Mind , he was probably not thinking of the " closed mind " under the image of a contraceptive . Still less would anyone be likely to think of the " open mind " that way .

图 8 ：英语文章原文

qgdt fooft novvu qkvwd wgd hovelty vs wgd fudklhft ulti , gd qfe ckvnfnoj tvw wgltblty vs wgd " hovedi ulti " atidk wgd lufyd vs f hvtwkfhdcwlxd . ewloo odee qvaoi ftjvtd nd olbdoj wv wgltb vs wgd " vcdt ulti " wgfw qfj .

图 9：英文文章密文

（3）最后我们将加密后的密文运用我们解密模型进行解密后得到：

when allan bloom wrote the closing of the american minj , he was probablu not thinking of the " closej minj " dnjer the image of a contraceptive . still less wodlj anuone be likelu to think of the " open minj " that wau .

图 10：英文文章解密后

解密之后，密文中的字母（第一排）所对应的解密后字母（第二排）

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*
f	*n*	*h*	*i*	*d*	*s*	*y*	*g*	*l*	*p*	*b*	*o*	*u*	*t*	*v*	*c*	*z*	*k*	*e*	*w*	*a*	*x*	*q*	*r*	*j*	*m*

然后通过与上面的密钥进行对比，通过对比可准确得出 21 个字母所对密钥，下图框出的5个字母则是匹配出现错误的字母，但是剩余 5 个字母由于出现频率过低，无法精确匹配。并且密钥可经过极少量的简单人工识别，得到完全准确的密钥。

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*
f	*n*	*h*	*i*	*d*	*s*	*y*	*g*	*l*	*p*	*b*	*o*	*u*	*t*	*v*	*c*	*z*	*k*	*e*	*w*	*a*	*x*	*q*	*r*	*j*	*m*
a	*b*	*c*	*j*	*e*	*f*	*g*	*h*	*i*	*x*	*k*	*l*	*m*	*n*	*o*	*p*	*q*	*r*	*s*	*t*	*d*	*v*	*w*	*y*	*u*	*z*

第一行字母表示标准明文字母表，第二行表示密文对应字母表，第三行表示解密密文所得的明文字母表。其中 26 个字母已有 21 个字母被确定出来，未确定的都是低频字母。仅占所有文章字母的2.1%。完全不会影响读者的阅读，因此这个解密模型所解密出来的文章准确率较高，能够较为精准完全解出密文。

4.2 模型二的分析与求解

4.2.1 模型二的分析

由于模型一的解密模型能够将密文通过设计的算法译成明文，但是为了评价算法的破译能力，我们需要建立一个评价模型。评价密文破译模型的破译能力，我们以运行时间长短、准确性和运算简便，这三个个方面评价模型破译能力。模型二问题分析流程图如下：

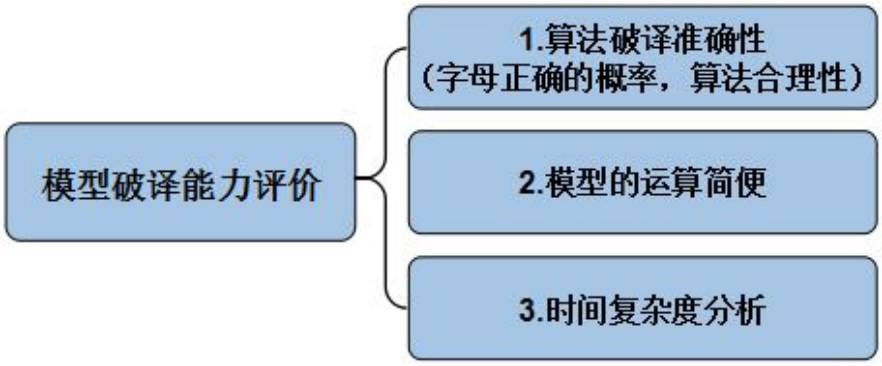

图 7：模型二的问题分析流程图

4.2.2 模型二的建立

鉴于对模型破译能力的评价，主要体现在两个方面，一个是算法破译的准确性，算法破译密文的时间以及算法易于理解，易于编码，易于调试。

首先分析算法破译的准确性，我们以下两个方面进行分析衡量。

Part 1：字母密钥的正确率：

$$C=\frac{\sum_{i=1}^{26}C_i}{26}\times 100\%$$

其中 C 表示字母密钥正确率，C_i 表示 i 个密钥是否正确。其中：

$$C_i=\begin{cases}1 & ,\text{第}i\text{个字母正确}\\ 0 & ,\text{第}i\text{个字母不正确}\end{cases}$$

Part 2：全文字母正确率：

$$P=\frac{Q}{Y}\times 100\%$$

其中 P 表示全文字母正确率，Q 表示全文正确的字母个数，Y 表示全文字母个数。

其次再通过计算时间的复杂度来衡量解密算法的破译效率。

Part 3：通过破译每万字母的所需时间作为衡量标准，以衡量算法解密文章的时间复杂度，详见模型二的求解（二）。

4.2.3 模型二的求解

（一）算法破译准确性

经过统计分析由解密模型算法破译的多篇加密文章，可以得出密钥字母正确

个数稳定在 20 个左右，密钥字母的正确率约80%，全文平均字母正确率约为96%，说明解密算法的准确率较高，那么解密出来的文章则完全不会影响读者的阅读。（见下表 11）

表 11：密钥字母匹配数及密问字母正确率

	Issue 1	Issue 2	Issue 3	Issue 4	Issue 5
密钥字母匹配个数	19	21	18	21	20
密文字母正确率	97.9%	97.4%	95.0%	94.3%	97.3%
	Issue 6	Issue 7	Issue 8	Issue 9	Issue 10
密钥字母匹配个数	20	21	19	19	20
密文字母正确率	96.4%	94.3%	97.8%	94.9%	96.3%

（二）时间复杂度分析

算法的复杂性是算法效率的度量，是评价算法优劣的重要依据。一个算法的复杂性的高低体现在运行该算法所需要的计算机资源的多少上面，所需的资源越多，我们就说该算法的复杂性越高；反之，所需的资源越低，则该算法的复杂性越低。计算机的资源，最重要的是时间和空间（即存储器）资源。因而，算法的复杂性有时间复杂性和空间复杂性之分。不言而喻，对于任意给定的问题，设计出复杂性尽可能低的算法是我们在设计算法时追求的一个重要目标；另一方面，当给定的问题已有多种算法时，选择其中复杂性最低者，是我们在选用算法适应遵循的一个重要准则。因此，算法的复杂性分析对算法的设计或选用有着重要的指导意义和实用价值.

(1)针对模型一算法的时间复杂度分析，我们做如下分析：

Step 1: 导入密文，对所有字母统一化(转化为全小写)，所消耗的时间为n。

Step 2: 扫描整篇密文计算每个字母出现频率及字母两两组合出现的频率所消耗的时间为n。

Step 3: 确定出高频字母后，对文中最长单词进行替换并与字典进行比对，所消耗时间为C_1（常数）。

Step 4: 再次对密文从头开始扫描，每扫入一个单词代入英语词典库进行对比，所消耗时间为$C_2 n$。

综上，则$F(n)=(C_2+2)n+C_1$，所以时间复杂度满足$O(n)$，其中C_1, C_2是依赖于字典库总词数的常数。下显示出，在测试的 10 篇文章中的程序运行时间和密文字母数：

表 12： 10 篇文章中程序运行时间和密文字母数

	Issue 1	Issue 2	Issue 3	Issue 4	Issue 5
程序运行时间(s)	0.85	2.75	12.55	50.75	7.65
密文字母数	228,083	50,370	52,174	25,790	570,147

	Issue 6	Issue 7	Issue 8	Issue 9	Issue 10
程序运行时间(s)	3.15	18.55	5.35	25.75	14.55
密文字母数	14,418	241,617	77,433	201,026	302,021

(2)我们还探讨了程序运行时间和密文字母数的关系，通过*Matlab*画图，可以得到同一篇文章破译时间随字母数量的变化情况图：

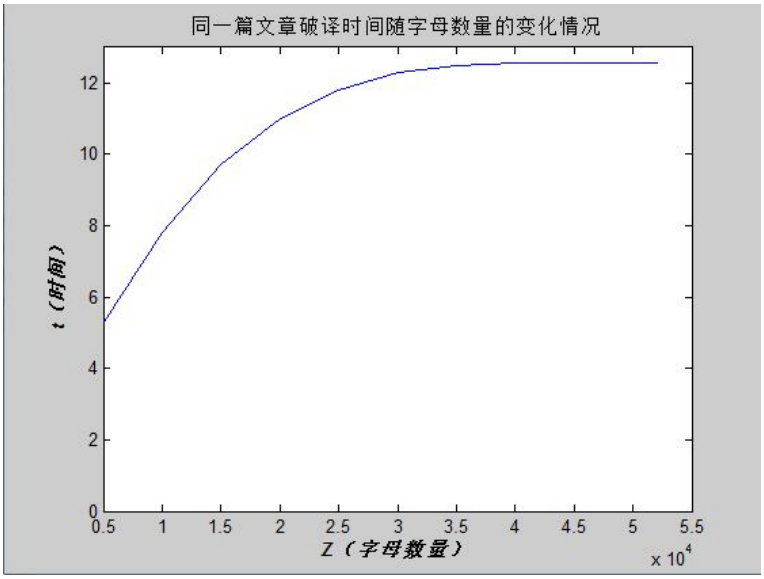

图 8：同一篇文章破译时间随字母数量的变化情况

上图可以看出，程序运行时间和密文字母数在时间大致字母数5,000至35,000的范围内是呈正比的关系，即密文字数越多则程序运行的时间越长，但在字母数大于35,000后，程序运行时间相对稳定，因为如果对全文的每一个单词进行扫描，破译时间必定随字母数量的增大而增长。但是我们动态逐词攻击的算法只扫描文章的前一部分，当扫描的单词信息量足够破译出≥24个字母，就对文章停止扫描。文章停止扫描后的剩余字母个数并不对算法破译时间产生影响，所以文章越长，单位时间内的破译效率越高。

五、模型的优缺点与评价推广

5.1 模型的优点

1. 我们对语料库做了字母测频，包括单字母、双字母以及三个字母，因此测试出来的频率相当稳定。（单词基数大概为2000万）

2. 此外我们还建立了自己的长单词库和英语单词库，这样由点及面进行的频率攻击所得到的破译密码具有较高可信度。

3. 我们解密算法解密的成功率为96.16%，因为这些字母都是极低频的字母，所以不能完全解密所有字母，但是绝大多数未破译字母完全可以通过人为干预识别出来，完全不会影响读者的阅读。

4. 本模型拥有最优的时间复杂度$O(n)$，对于较长的密文能在短时间（10s之内）进行破解。

5.2 模型的缺点

1.在模型二中，由于英文单词本身的复杂性，难以考虑完全的情况，设计万能的算法，因此对于部分单词的解密不能实现全自动，仍需要引入极少量人为干预。

2.当遇到专业性很强的文章时，文章中的高频词与统计的高频词出入较大，因此算法破解的准确率就会大大降低。

5.3 模型的改进

改进：现在算法仅对单字母加密的文章进行解密，如果是双字母表、多字母表就需要对算法重复运行两次甚至多次才能得到结果，算法的改进方向是能自动识别当前破解的文章是否为最终目标，如果是，程序运行结束，如果不是程序进行下一次循环。

六、参考文献

[1] 语料库：http://corpus.byu.edu/full-text/formats.asp

[2] 吴干华，基于频率分析的替代密码破译方法及其程序实现，福建电脑，第 9 期，2006 年

[3]http://wenku.baidu.com/link?url=Bb4eJdX7wm3iCxilqjKxkjL3X9H6Ej7ueSS6OSwUV1GuIIhFADmR-BlHjKkFCgPKaFZIdNhWPzkSYXe3cb4n_TQV1InpCAdRT5O1EHK3Uce

[4] G.Navarro,M.Raffinot,Flexible Pattern Matching in Strings(柔性字符串匹配)，2007.3

七、附录

一．图表

表 12 双字母组合频率最高前 10 名

th	*he*	*in*	*er*	*an*
0.0340	0.0304	0.0245	0.0214	0.0197
re	*on*	*at*	*en*	*nd*
0.0175	0.0148	0.0145	0.0129	0.0127

二、程序

```
tic
clc,clear;
A=fopen('w_acad_2010.txt','r');
b=fread(A);%原文章导入的原始数据
[len,wit] =size(b);
```

```
jieb2 = zeros(len,wit);%加密后的文章
for i= 1:len
    while(b(i)>=65 && b(i) <= 90)
        b(i) = b(i)-'A'+'a';
    end
end
begin = 97:122;          %  标准字幕顺序表
final = rendprim(1:26);%随机生成加密所对字母表
for i = 1:len
    if(b(i) >= 97 && b(i) <=122)
        jieb2(i) = begin(final(b(i)-'a'+1));
    else jieb2(i) = b(i);
    end
end
fid = fopen('w_acad_20110mi.txt','w');%加密后的文章到出
fprintf(fid,'%c',jieb2);
fclose(fid);

A=fopen('w_acad_20110mi.txt','r');%加密后的文章导入
b=fread(A);%文章导入的原始数据
[len,wit] =size(b);

load word15.mat;
load word16.mat;
load word17.mat;
load word18.mat;
load word19.mat;
load word20.mat;
load word21.mat;
load word23.mat;
load word24.mat;

%load word_len1.mat;
load word_len2.mat;
load word_len3.mat;
load word_len4.mat;
load word_len5.mat;
load word_len6.mat;
load word_len7.mat;
load word_len8.mat;
load count_word_len.mat;

for i= 1:len
```

```
    while(b(i)>=65 && b(i) <= 90)
        b(i) = b(i)-'A'+'a';
    end
end

level = zeros(1,26);

pr = zeros(1,26);%  每个字母在所有中的出现次数
cum=0;                  %所有的字母总数
for i = 1:len
    if((b(i) >= 97 && b(i) <= 122))
        pr(b(i)-'a'+1) =    pr(b(i)-'a'+1)+1;
        cum = cum+1;
    end
end
pri = pr/cum; %  每个字母在所有中的频率
sor_pri = sort(pri);

[m_al,n_al] = find(pri>sor_pri(23),3); % m,n 表示最高频率的 3 个字母所在的位置
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%5

front = 1;
rear = 1;
maxL= 0;
while(front <= len-1)
    if((b(front) <97 || b(front) >122)&& (b(front+1) >= 97 && b(front+1) <=122))
        rear = front;
    end
    if((b(front)  >=  97  &&  b(front)  <=  122)&&  (b(front+1)  >=  97  ||  b(front+1)
<=122))
            if(maxL < front-rear)
                maxL = front - rear;
            end
    end
    front = front +1;
end


front =1;
rear =1;
count = zeros(1,6); %最长长度的单词出现的次数
max1 = 0;   %统计单词的长度
```

```
two_als = zeros(26,26);     %字母两两组合出现的频率
count3 = 0;         %计算单词长度为三的单词数
dL1=0;
dL2=0;
dL3=0;
dL4=0;
dL5=0;
dL6=0;
for front=1:len-1
      x= b(front);
      y= b(front+1) ;

      if((x<97 || x >122)&& (y>= 97 && y<=122))
            rear = front;
      end

      if (x>='a' && x<='z') && (y >='a' && y<='z')
            two_als(x-'a'+1,y-'a'+1) =two_als(x-'a'+1,y-'a'+1)+1;
      end

      if((x >= 97 && x <= 122)&& (y < 97 || y >122))
                  max1 = front - rear;
                  if max1 > maxL-5
                        switch max1
                              case maxL-5
                                    count(1) = count(1)+1;
                                    for j = 1:max1
                                          dL1(count(1),(j)) = b(rear + j);
                                    end
                              case maxL-4
                                    count(2) = count(2)+1;
                                    for j = 1:max1
                                          dL2(count(2),(j)) = b(rear + j);
                                    end
                              case maxL-3
                                    count(3) = count(3)+1;
                                    for j = 1:max1
                                          dL3(count(3),(j)) = b(rear + j);
                                    end
                              case maxL-2
                                    count(4) = count(4)+1;
                                    for j = 1:max1
                                          dL4(count(4),(j)) = b(rear + j);
                                    end
```

```
                    case maxL-1
                        count(5) = count(5)+1;
                        for j = 1:max1
                            dL5(count(5),(j)) = b(rear + j);
                        end
                    case maxL
                        count(6) = count(6)+1;
                        for j = 1:max1
                            dL6(count(6),(j)) = b(rear + j);
                        end
                end
            end
    end
end

sor_two_als = sort(two_als);
sor2_two_als = sort(sor_two_als(end,:));
[m_two_als,n_two_als] = find(two_als>sor2_two_als(end-3),3); % m,n 表示最高频
率的 3 个字母所在的位置

for
i=1:3                       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%找出 e,h
    for j=1:3
        if n_al(i)==n_two_als(j)
            level(n_two_als(j)) = 5;
            level(m_two_als(j)) = 8;
        end
    end
end

for
i=1:3                       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%找出 t
    for j=1:3
        if n_two_als(i)==m_two_als(j) && level(n_two_als(j)) == 5;
            level(m_two_als(i)) = 20;
        end
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for k=6:-1:5
    switch maxL
```

```
        case 24
            Y=d10;
        case 23
            Y=d9;
        case 22
            Y=d8;
        case 21
            Y=d7;
        case 20
            Y=d6;
        case 19
            Y=d5;
        case 18
            Y=d4;
        case 17
            Y=d3;
        case 16
            Y=d2;
        case 15
            Y=d1;
    end

    switch k
        case 6
            Y1=dL6;
        case 5
            Y1=dL5;
        case 4
            Y1=dL4;
        case 3
            Y1=dL3;
        case 2
            Y1=dL2;
        case 1
            Y1=dL1;
      end
    flag = zeros(1,maxL);%所有标记比对的地方
    if count(k)
        for j=1:count(k)
            for i=1:maxL
                x = Y1(i)-'a'+1;
                if level(x) ~=0
                    %dL5(i) = level(x)+'a'-1;
                    flag(i) = 1;
```

```
                end
            end
        end

        [count_word,count_wo] =size(Y);    %count_word 记录字典中该长度的单词的个数
        for j = 1:count_word
            for i=1:maxL
                flag1 = 1;
                if flag(i)==1
                    x=level(Y1(i)-'a'+1)+'a'-1;
                    if Y(j,i) ~= x
                        flag1 = 0;
                        break
                    end
                end
            end
            if flag1==1
                for i=1:maxL
                    if flag(i)==0
                        x = Y1(i)-'a'+1;
                        level(x)= Y(j,i)-'a'+1;
                        flag(i)=1;
                    end
                end
            end
        end
    end
    maxL= maxL-1;
end

flag_level = zeros(1,26); %记录翻译完成字母的个数,只要有替换出结果,就标记 1

for i=1:26
    if level(i)
        flag_level(i) = 1;
%           flag_level2(i) =1;
    end
end

level2 = [];    %剩余替换所有的可能全记录
count_level2 =zeros(1,26); %记录替换的次数

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%从开始对密文进行翻译
front = 1;
rear = 1;
maxL = 0;

count_word_len1 = 0;   %扫描密文每个单词的长度

while(front <= len-1)
    if((b(front) <97 || b(front) >122)&& (b(front+1) >= 97 && b(front+1) <=122))
        rear = front;
    end
    if((b(front) >= 97 && b(front) <= 122)&& (b(front+1) >= 97 || b(front+1)
<=122))
                count_word_len1 = front - rear;
                if count_word_len1>=6 && count_word_len1<=8
                    switch count_word_len1
                        case 2
                            flag_X = zeros(1,2);
                            for i=1:2
                                X(i) = b(rear+i);
                                if level(X(i)-'a'+1)
                                    flag_X(i)=1;
                                    X1(i) = level(X(i)-'a'+1)+'a'-1;
                                else    X1(i) = X(i);
                                end
                            end
                            Y=word_len2;
                        case 3
                            flag_X = zeros(1,3);
                            for i=1:3
                                X(i) = b(rear+i);
                                if level(X(i)-'a'+1)
                                    flag_X(i)=1;
                                    X1(i) = level(X(i)-'a'+1)+'a'-1;
                                else    X1(i) = X(i);
                                end
                            end
                            Y=word_len3;
                        case 4
                            flag_X = zeros(1,4);
                            for i=1:4
                                X(i) = b(rear+i);
                                if level(X(i)-'a'+1)
                                    flag_X(i)=1;
```

```
                X1(i) = level(X(i)-'a'+1)+'a'-1;
            else    X1(i) = X(i);
            end
        end
        Y=word_len4;
    case 5
        flag_X = zeros(1,5);
        for i=1:5
            X(i) = b(rear+i);
            if level(X(i)-'a'+1)
                flag_X(i)=1;
                X1(i) = level(X(i)-'a'+1)+'a'-1;
            else    X1(i) = X(i);
            end
        end
        Y=word_len5;
    case 6
        flag_X = zeros(1,6);
        for i=1:6
            X(i) = b(rear+i);
            if level(X(i)-'a'+1)
                flag_X(i)=1;
                X1(i) = level(X(i)-'a'+1)+'a'-1;
            else    X1(i) = X(i);
            end
        end
        Y=word_len6;
    case 7
        flag_X = zeros(1,7);
        for i=1:7
            X(i) = b(rear+i);
            if level(X(i)-'a'+1)
                flag_X(i)=1;
                X1(i) = level(X(i)-'a'+1)+'a'-1;
            else    X1(i) = X(i);
            end
        end
        Y=word_len7;
    case 8
        flag_X = zeros(1,8);
        for i=1:8
            X(i) = b(rear+i);
            if level(X(i)-'a'+1)
                flag_X(i)=1;
```

```
                                    X1(i) = level(X(i)-'a'+1)+'a'-1;
                                else    X1(i) = X(i);
                                end
                            end
                            Y=word_len8;
                        end

 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%对单词进行字典查找
                        for j = 1:count_word_len(count_word_len1)
                            for i=1:count_word_len1
                                flag1 = 1;
                                if flag_X(i)==1
                                    x=level(X(i)-'a'+1)+'a'-1;
                                    if Y(j,i) ~= x
                                        flag1 = 0;
                                        break
                                    end
                                end
                            end
                            if flag1==1
                                for i=1:count_word_len1
                                    if flag_X(i)==0
                                        x = X(i)-'a'+1;
                                        count_level2(x) = count_level2(x)+1;
                                        level2(count_level2(x),x)=Y(j,i)-'a'+1;
                                        flag_level(x) = 1;
                                    end
                                end
                            end
                        end
                    end
                end
            end
            if sum(flag_level)>=25
                break
            end
            front = front +1;
        end

        sumn = zeros(26,26);
        for i = 1:26
            if level(i)==0 && count_level2(i) ~=0
                for j=1:count_level2(i)
                    x=level2(count_level2(i),i);
```

```
                sumn(i,x) = sumn(i,x)+1;
            end
        end
end

flag_level2 = zeros(1,26);%只标记前面完全正确替换的字母
flag_level3 = zeros(1,26);
for i=1:26
        if level(i) ~=0
            flag_level2(i)=1;
            flag_level3(level(i)) = 1;
        end
end

for i = 1:26
        if~flag_level2(i)
            k=max(sumn(i,:));
            [m,n]=find(sumn==k);
            if ~flag_level3(n)
                level(i) = n;
                flag_level3(n) = 1;
            end
        end
end


for i = 1:26
        for j=1:26
            if ~level(i) && ~flag_level3(j)
                level(i) = j;
                flag_level3(j)=1;
            end
        end
end

jieb2 = zeros(len,wit);%加密后的文章
for i= 1:len
        while(b(i)>=65 && b(i) <= 90)
            b(i) = b(i)-'A'+'a';
        end
end
begin = 97:122;          %  标准字幕顺序表
for i = 1:len
        if(b(i) >= 97 && b(i) <=122)
```

```
            jieb2(i) = begin(level(b(i)-'a'+1));
        else jieb2(i) = b(i);
        end
end
fid = fopen('w_acad_20110jiemi.txt','w');
fprintf(fid,'%c',jieb2);
fclose(fid);

z_q_s=0;
for i =1:26
        if level(final(i))==i
            z_q_s=z_q_s+1;
        end
end
toc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc,clear;
A1=fopen('33.txt','r');
b1=fread(A1);%文章导入的原始数据
[len1,wit1] =size(b1);

for i= 1:len1
        while(b1(i)>=65 && b1(i) <= 90)
            b1(i) = b1(i)-'A'+'a';
        end
end

A2=fopen('33jiemi.txt','r');
b2=fread(A2);%文章导入的原始数据
[len2,wit2] =size(b2);

for i= 1:len2
        while(b2(i)>=65 && b2(i) <= 90)
            b2(i) = b2(i)-'A'+'a';
        end
end

rear=1;
front=1;
count_sum=0;
count_error=0;
```

```
while(front <= len2-1)
    if((b1(front) <97 || b1(front) >122)&& (b1(front+1) >= 97 && b1(front+1)
<=122))
        rear = front;
    end
    if((b1(front) >= 97 && b1(front) <= 122)&& (b1(front+1) >= 97 || b1(front+1)
<=122))
        count_sum = count_sum+front-rear;
    end
    if b1(front)~=b2(front)
        count_error = count_error+1;
    end
    front = front +1;
end

d=count_error/count_sum
```