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：　　1348

参赛队员　(签名)　：

队员 1：

队员 2：

队员 3：

参赛队教练员 (签名)：

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）：1348

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

关 键 词　　替换式密码破译　频率分析　单字母替换

摘　　要：

对于仅有密文的破译，我们只能采用唯密文攻击的方式。在密文长度较短的情形下，可以采取的较好的策略是穷搜索。但如果密文很长，我们采用的攻击方式是对密文中的字母做各类频率统计分析：分别统计出密文中每个字母、双连字母出现的频数，以及在每个单词的第一位、第二位、第三位及末位上各个字母出现的频数并进行排序，将排序的结果与字母频率表进行对比，分别作出相应的字母替换表。最后，按照一定的规则比对得到的几张字母替换表，就可以最终确定部分的字母替换规定。以上步骤，我们将编写算法，并利用 C-Free 软件编程进行运算。而对于密文中剩下的那些还未确定的字母，则辅以英文单词的书写规律进行人工判断。至此，我们就可以得到完整的字母替换表，对应此表，将密文转化为明文，就完成了我们的破译工作。

在对上述破译模型进行破译能力的评估时，我们结合设计原理与程序的编写及运行的实际情况，确定了以下三个评估标准：

1.“所得的字母替换表中确定的字母个数”。事实上，在模型中采用的主要规则以及规则组合的不同，会使得我们得到的字母替换表中，可以确定的替换的个数也不同。一般情况下，由程序得到的替换越多，密文破译后的准确性就越高。

2.“程序得到字母替换表所需的时间”。对于密码破解性能的测试还要看指定时间范围内可以运行多少次的运算。因此，我们在模型算法中，加上了计算运行时间的程序（以毫秒为单位），使运行结果能显示程序运行所需的时间。在某种意义上，这个时间越短，则破译能力越强。

3.“破译后得到的明文的可靠性”。模型的最终目标是破译密文，得到需要的明文。因此，明文的合理性也是我们需要关注的重点。考虑到计算机的局限，这一过程，我们将人工检验明文中单词的出错率，明文整体的语法问题以及语句的通顺程度，并以此作为对模型的评估的最后一条标准。

参赛队号: 1348

所选题目: B 题

参赛密码 _____

（由组委会填写）

英文摘要（选填）

In occasion that attacker is assumed to have access only to a set of ciphertexts, the only model could be assume is ciphertext-only attack. In situation where ciphertext is relatively short, the better strategy is brute-force search. However in situation ciphertext is relatively long, the attack mode to adopt is to make statistical analysis of frequence of letters in cypertext: collect statistics separately of the times each single letter and two consecutive letters appear in the ciphertext, as well as the times each letter in alphabet holds the first, second, third and the last position of each word, rank them and compare its order with the Letter Frequency, replacement charts of correspondent letters shall be made according to that. Finally, compare the replacement charts which was obtained previously in certain rules, shall the replacement rules of letters be partially defined. For previous procedures, we made algorithm of them and computed them by programming with C-Free. On the other hand, for those letters left uncertain in ciphertext, artificial judgement shall be helpful with the writing regulation of English words. At this point, the entire replacement charts is accessible, according to which, the ciphertext has been transformed into plaintext, so as to finish the work of decipherment.

In the evaluation of it on the crack model, we combine the principle and the program design of the programming and operation situation, identified the following three evaluation criteria:

1."The number of letters we can determine in the table ". In fact, the main rules adopted in the model and the combinations of rules will make us get different letter substitution table, so the number of letters we can determine will be different. Generally, the more replacement a program can do, the higher precision outcome we can get.

2."Time the program need to get the letter substitution table ". The password cracking performance test depends on the times operation can run on a specified time as well. Therefore, so in our model, we add a program to calculate the running time of the main program (in milliseconds). In a sense, the less time it takes, the stronger it is.

3."The reliability of decoded plaintext". The ultimate goal is to decipher the model and get the plaintext that we need. Therefore, the reliability of decoded plaintext is also the key point what we concern of. Considering the limitations of the computer, in this process, we will test the word error rate, grammar, and the expression of sentences by ourselves. And that is a standard as the assessment of the model.

Key: Replace type password, Cryptanalysis, Frequency analysis, Single letter substitution

替换式密码

1 问题的重述

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。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。较为复杂的形式是以多个字母为一个单元，整体替换成其它的字符。这个映射方法被称为密码表，拿到密码表的人就能够将密文破译成明文。单字母替换加密是在古代就使用过的一种加密方法，但由于其容易被破解，所以在现代需要加密的场合已经很少使用。

单字母替换加密的破译方法有频率分析等。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”。但当获取的密文篇幅不是很大时，光凭借频率分析是不足以破译全部密文的。往往还要熟知该种语言的人，经过对可能出现的词汇及字母组合进行分析，才能完整地破译密码。

第一阶段问题：

问题 1：假设明文是由现代通常使用的英语写成的。现在我们获取了一些由单字母加密方法加密的密文。请你建立合理的数学模型，设计一个算法，来自动化地破译密文。

问题 2：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。

为了问题简单起见，我们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。在设计算法时，如果需要，可以参考英文语料库的数据，例如免费的 COCA[1] 等相关资料。

2 模型的假设与符号的说明

2.1 模型的假设

（1）假设明文是由现代通常使用的英语写成的；

（2）假设密文都是由小写的英文字母构成；

（3）假设密码表仅是针对26个字母，每个单词之间的空格以及标点符号都会保留；

（4）假设所给密文篇幅较长，即密文单词数量在 20 个以上。

（5）假设所给的明文不是选自专业性较强的学术文章，并且在文章中各字母出现的可能性符合一般规律。

2.2 符号的说明

符号	意义	符号	意义
α_i	26 个字母中的其中一个	β_i	α_i 字母出现的频数
φ_i	第 i 个位置上的符号（包含字母、空格和标点）	γ_i	第 i 个位置上的字母
ω_i	单词首字母	θ_i	单词第二位字母

3 问题的分析

3.1 问题一的分析

单表替代密码对明文中的所有字母都使用一个固定的映射，即使明文中的每一个字母都被另一个固定的字母替换。[1]在本题中，我们知道的仅有一段密文，因此，我们能够采取的破解方式是唯密文攻击。

在密文长度较短的情形下，可以采取的较好的策略是穷搜索。但如果消息足够长，

则可以采用的攻击是对各种字母做频率统计分析。此分析基于这样一个事实：在大多数英文文章中，每个字母出现的频率，双连字母出现的频率以及三连字母出现的频率都是不相同的，并且，每个字母出现在单词的各个位置的概率也是不同的。在这样的事实规则下，我们可以设计程序，依照各种字母频率规则，分别统计出密文中各类字母出现的频数，将其按频数由高到低排列，再分别对照各自的频率规则表，作出相应的字母替换表。最后，按照一定的频率规则，就可以确定一些明、暗文中字母的替换规定，而对于密文中剩下的那些还未确定的字母，则辅以英文单词的书写规律进行人工判断。

在破译过程中，我们主要要用到的字母频率规则[2]如下：

【规则 1】Frequencies of the letters in the English language（最常用英文字母频率）：

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

【规则 2】单字母只有 i 和 a 两个

【规则 3】The most common first letter in a word in order of frequency（单词中最常见的首字母）：

T, O, A, W, B, C, D, S, F, M, R, H, I, Y, E, G, L, N, O, U, J, K

【规则 4】The most common second letter in a word in order of frequency（单词中最常见的第二个字母）：

H, O, E, I, A, U, N, R, T

【规则 5】The most common third letter in a word in order of frequency（单词中最常见的第三个字母）：

E, S, A, R, N, I

【规则 6】The most common last letter in a word in order of frequency（单词中最常见的末字母）：

E, S, T, D, N, R, Y, F, L, O, G, H, A, K, M, P, U, W

【规则 7】The most common digraphs on order of frequency（最常用的双连字母）：

TH, HE, AN, IN, ER, ON, RE, ED, ND, HA, AT, EN, ES, OF, NT, EA, TI, TO, IO, LE, IS, OU, AR, AS, DE, RT, VE,ON, ST, NT,NG, OR, ET, IT, AR, TE, SE

3.2 问题二的分析

综合此次我们破译密码的原理与程序的编写及运行的实际情况，我们决定以以下三个标准评估我们模型破译密码能力：

首先，我们采用字母频率分析的方法，即以密文中各字母出现的频数对应已知的频率表来确定字母的替换。在这一过程中，一方面，规则本身是人为总结的，它所具备的科学性不是很高；另一方面可选取的规则较多，而需要编写算法作为主要判断依据的规则却不可以过多。因此，在模型中采用的规则以及规则组合的不同，会使得我们得到的字母替换表中，可以确定的替换的个数也不同。一般情况下，由程序得到的替换越多，密文破译后的准确性就越高。在这样的事实下，我们将“程序运行所得的字母替换表中确定的替换的个数的多少”作为我们评估模型破译密码能力的第一条标准。

其次，对于密码破解性能的测试还要看指定时间范围内可以运行多少次的运算，通常情况下都会根据算法复杂程度的不同以每秒千次或每秒百万次作为衡量单位。[3]因此，我们在模型算法中，特意加上了计算运行时间的程序（以毫秒为单位）。在最后得到的结果中，会显示我们的程序破译一段超过 1000 个字母的密文所需的时间，在某种意义上，这个时间越短，则破译能力越强。综上所述，我们将“程序得到字母替换表所需的时间长短”作为我们评估模型破译密码能力的第二条标准。

最后，我们将在程序得到的字母替换表的基础上，辅以英文书写中的规律，人工确定剩余的字母的替换，由此，就可以得到完整的字母替换表。依照我们做出的字母替换表，将密文翻译成明文。阅读明文，检验明文中单词的出错率以及明文整体的语句通顺

程度，而这，也是我们评估模型破译密码能力的最后一条准则：“估计破译的明文的可靠性”。

4 模型的建立

4.1 问题一模型的建立与求解

在该部分中，为了更直观地反映破译过程，我们将以下面这段密文为例进行程序演示。

密文：

lw nsoz bnwvw baybnv bs qw vwoh wdizwrb, bnpb poo uwr paw xawpbwz wmypo, bnpb bnwj paw wrzslwz qj bnwia xawpbsa libn xwabpir yrpoiwrpqow aienbv, bnpb pusre bnwvw paw oihw, oiqwabj prz bnw fyavyib sh npffirwvv. bnpb bs vwxyaw bnwvw aienbv, esdwaruwrbv paw irvbibybwz pusre uwr, zwaidire bnwia tyvb fslwav hasu bnw xsrvwrb sh bnw esdwarwz, bnpb lnwrwdwa prj hsau sh esdwaruwrb qwxsuwv zwvbayxbidw sh bnwvw wrzv, ib iv bnw aienb sh bnw fwsfow bs pobwa sa bs pqsoivn ib, prz bs irvbibybw rwl esdwaruwrb, opjire ibv hsurzpbisr sr vyxn fairxifowv prz saeprikire ibv fslwav ir vyxn hsau, pv bs vwwu usvb oicwoj bs whhwxb bnwia vphwbj prz npffirwvv. fayzwrxw, irzwwz, lioo zixbpbw bnpb wsdwaruwrbv osre wvbpboivnwz vnsyoz rsb bw xnprewz hsa oienb prz baprviwrb xpyvwv; prz pxxsazireoj poo wgfwaiwrxw npbn vnwlr, bnpb uprcirz paw usaw zivfsvwz bs vyhhwa, lniow wdiov paw vyhhwapbow, bnpr bs aienb bnwuvwodwv bj pbsoivnire bnw hsauv bs lnixn bnwj paw pxxyvbsuwz. qyb lnwr p osre bapir sh pbyvwv prz yvyafpbisrv, fyavyire irdpaipboj bnw vpuw sbtwxb wdirxwv p zwvier bs awzyxw bnwu yrzwa pbvsoybw zwvfsbivu, ib iv bnwia aienb, ib iv bnwia zybj, bs bnasl shh vyxn esdwaruwrb, prz bs fasdizw rwl eypazv hsa bnwia hybyaw vwxyaiby.

4.1.1 单个字母频率分析

现在有这样一个事实，在大多数英文文字中，字母出现的频率是不一样的。一般来说，单词中 e 出现的频率比 x 的频率要大得多。在该模型中，我们首先分析密文中单个字母出现的频数（以频数代替频率）：

记

$$\alpha_1 = a，\ \alpha_2 = b，\ \cdots\cdots，\ \alpha_{25} = y，\ \alpha_{26} = z，$$

由 CF 程序自动分析密文中的每一个字母，但程序运行的过程中不计空格和标点。将 α_i 出现的频数记为 β_i，然后成对列出 α_i 和 β_i。

再选用由 Beker 和 Piper 统计得出的 Beker-Piper 字母频率表（见表 1），将上述统计出的各字母频数 β_i 按由高到低的顺序依次对应到 Beker-Piper 字母频率表，得出由单个字母频数得出的单个字母替换表。

表 1　Beker-Piper 字母频率表

a	b	c	d	e	f	g	h	i	j
0.082	0.015	0.028	0.043	0.127	0.022	0.020	0.061	0.070	0.002
k	l	m	n	o	p	q	r	s	t
0.008	0.040	0.024	0.067	0.075	0.019	0.001	0.060	0.063	0.091
u	v	w	x	y	z				
0.028	0.010	0.023	0.001	0.020	0.001				

下面，我们用所给密文进行第一轮演示：（程序见附件）

首先，输入密文后就能得到密文中各个字母出现的频数，将密文中各字母按频数由高到低的顺序排列，根据 Beker-Piper 字母频率表，给出单个字母替换图（见图 1），并制成表格（见表 2）。

图 1　单个字母频率分析运行图

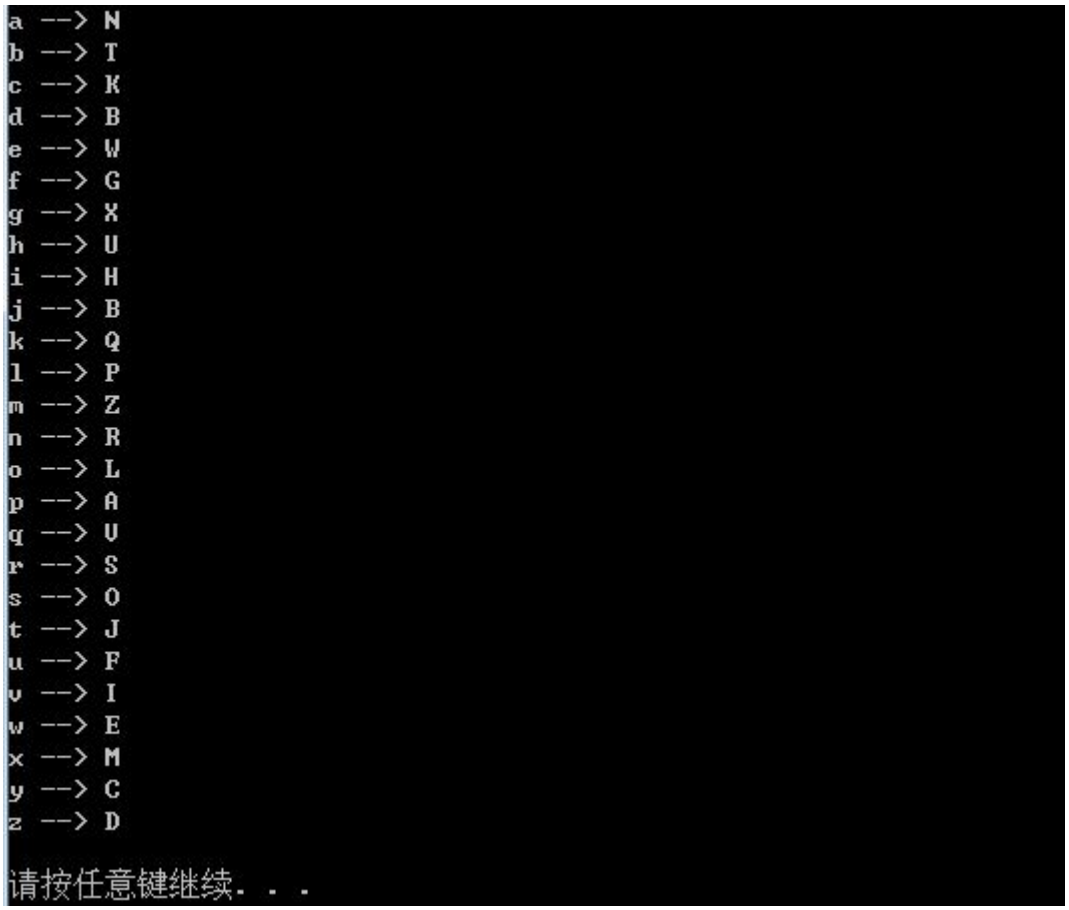


表 2　单个字母替换表

密文字母表	A	B	C	D	E	F	G	H
明文字母表	N	T	K	Y	W	G	X	U
密文字母表	I	J	K	L	M	N	O	P
明文字母表	H	B	Q	P	Z	R	L	A
密文字母表	Q	R	S	T	U	V	W	X
明文字母表	V	S	O	J	F	I	E	M
密文字母表	Y	Z						
明文字母表	C	D						

4.1.2 对单字母单词的分析

在英文中由单个字母组成的单词只有 a 和 i，具有很大的特殊性。由于密文中保留了单词间的空格及标点符号，所以，我们可以根据在第 i 个位置上的符号（包含字母、空格和标点）φ_i 的前一位 φ_{i-1} 和后一位 φ_{i+1} 上的符号来判断 φ_i 是否是单个字母组成的单词。即，

若 φ_{i-1} 和 φ_{i+1} 均为空格或标点，则输出 φ_i 上的字母，该字母对应明文中的字母 a 或者字母 i。

下面，我们用所给密文进行第二轮演示：（程序见附件）

输入密文，得到如下运行结果（见图 2），

图 2　单字母单词分析运行图

说明密文中的字母 p 对应明文中的 a 或者 i，制成表格（见表 3）

表 3　单字母单词替换表

密文字母表	P
明文字母表	A
或	
密文字母表	P
明文字母表	I

4.1.3 双连字母频率分析

仅凭借上述得到的单个字母替换表和单字母单词分析还不足以确定正确的字母替换，现在我们需要做的是观察双连字组合的频率。这里，我们将密文中第 i 个位置上的字母记为 γ_i（这里空格和标点不计位置），由 CF 程序自动统计双连字母 $\gamma_i\gamma_{i+1}$ 出现的频数。再根据【规则 7】，将双连字母 $\gamma_i\gamma_{i+1}$ 的频数按由高到低的顺序依次对应到双连字母频率表。

下面，我们用所给密文进行第三轮演示：（程序见附件）

输入密文，得到如下运行结果（见图 3），并制成表格（见表 4）。

图 3　双连字母频率分析运行图

```
bw --->  LE
pa --->  IS
uw --->  OU
dw --->  AR
iv --->  AS
oi --->  DE
rb --->  RT
vy --->  VE
wi --->  ON
bv --->  ST
ie --->  NT
sd --->  NG
sh --->  OR
sr --->  ET
vb --->  IT
yb --->  AR
ar --->  TE
bi --->  SE
```

表 4　双连字母替换表

密文双字母表	BN	BW	PB	IR	WR	BS	WA	WV
明文双字母表	TH	HE	AN	IN	ER	ON	RE	ED
密文双字母表	PR	AW	NP	RZ	VW	IB	RE	AI
明文双字母表	ND	HA	AT	EN	ES	OF	NT	EA
密文双字母表	SA	WZ	ZW	BW	PA	UW	DW	IV
明文双字母表	TI	TO	IO	LE	IS	OU	AR	AS
密文双字母表	OI	RB	VY	WI	BV	IE	SD	SH
明文双字母表	DE	RT	VE	ON	ST	NT	NG	OR
密文双字母表	SR	VB	YB	AR	BI			
明文双字母表	ET	IT	AR	TE	SE			

4.1.4 单词各位置字母频率分析

英文单词还有一个特殊性，单词各个位置上字母出现的频率也有一定的差别，即首字母多出现 T，O，A，W 等，第二位出现字母 H，O，E，I，A 的频率计较高，以此类推，我们可以根据密文中单词不同位置上频数得出各自的字母替换表。

（ｉ）首字母频率分析

我们记空格或者标点后的第一个字母分别为

$$\omega_1，\ \omega_2，\ \cdots\cdots，\ \omega_i，\ (i \le 26)$$

由 CF 程序统计各个字母重复出现的次数，即统计 ω_i 的频数。再根据【规则 3】，将首字母出现的频数按由高到低的顺序依次对应到首字母频率表。

下面，我们用所给密文进行第四轮演示：（程序见附件）

输入密文，得到如下运行结果（见图 4），并制成表格（见表 5）.

图 4　首字母频率分析运行图

```
输出第1个字母个数：

a的个数为：6
b的个数为：50
c的个数为：0
d的个数为：0
e的个数为：6
f的个数为：8
g的个数为：0
h的个数为：8
i的个数为：13
j的个数为：0
k的个数为：0
l的个数为：7
m的个数为：0
n的个数为：4
o的个数为：7
p的个数为：30
q的个数为：4
r的个数为：3
s的个数为：11
t的个数为：1
u的个数为：5
v的个数为：12
w的个数为：10
x的个数为：6
y的个数为：3
z的个数为：7
b --> T
p --> O
i --> A
v --> W
s --> B
w --> C
f --> D
h --> S
l --> F
o --> M
z --> R
a --> H
e --> I
x --> Y
u --> E
n --> G
q --> L
r --> N
y --> O
t --> U
c --> J
d --> K
```

表 5　首字母替换表

密文首字母表	B	P	I	V	S	W	F	H
明文首字母表	T	O	A	W	B	C	D	S

密文首字母表	L	O	Z	A	E	X	U	N
明文首字母表	F	M	R	H	I	Y	E	G
密文首字母表	Q	R	Y	T	C	D		
明文首字母表	L	N	O	U	J	K		

（ii）第二位字母频率分析

我们记空格或者标点后的第二个字母分别为

$$\theta_1，\ \theta_2，\ \cdots\cdots，\ \theta_i，\ (i \le 26)$$

与（i）同原理，统计θ_i的频数，根据【规则 4】，得出第二位字母频率表。

下面，我们用所给密文进行第五轮演示：（程序见附件）

输入密文，得到如下运行结果（见图 5），并制成表格（见表 6）.

图 5　第二位字母频率分析运行图

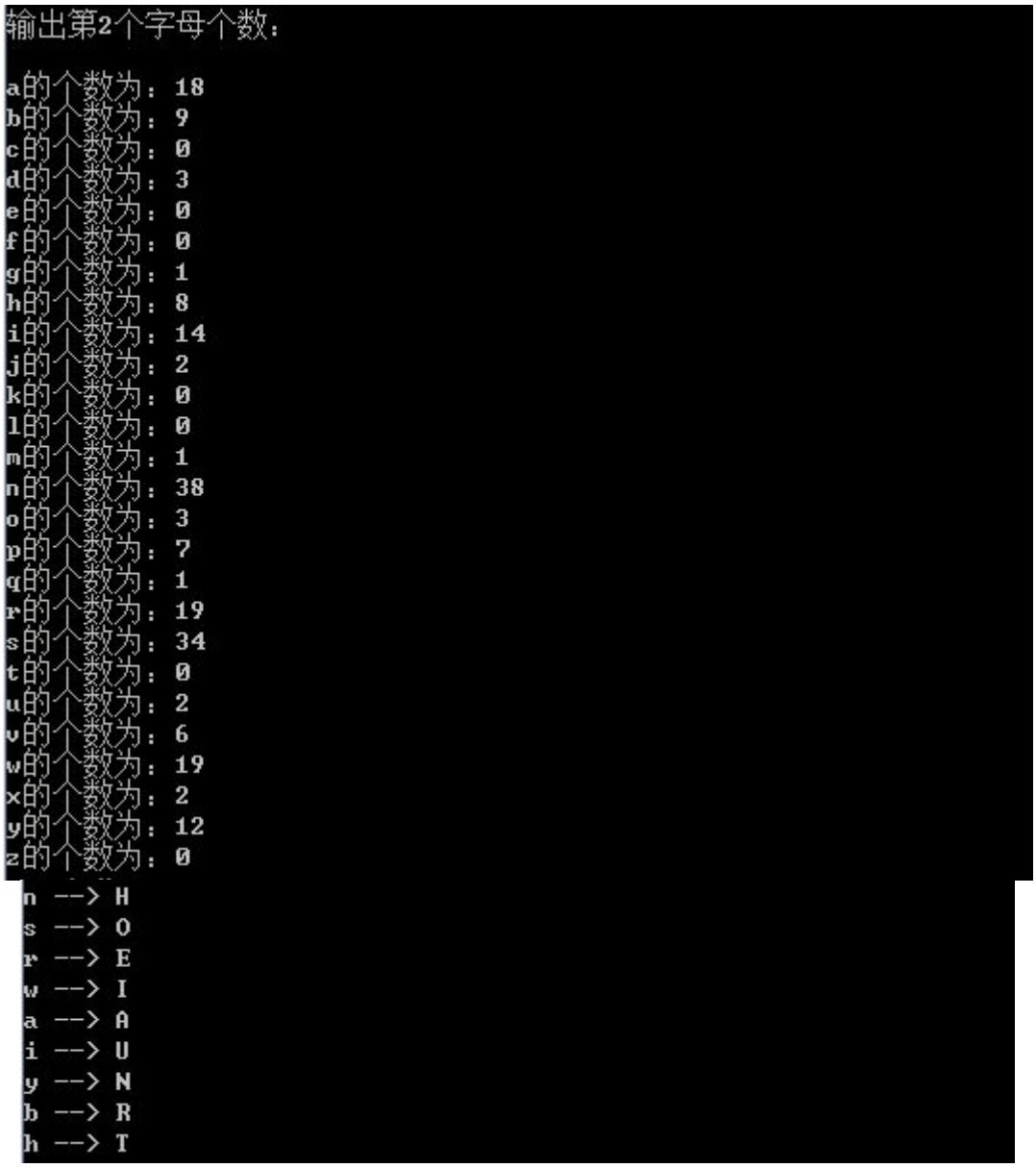


表 6　第二位字母替换表

密文字母表	A	B	C	D	E	F	G	H
明文字母表	A	R						T
密文字母表	I	J	K	L	M	N	O	P
明文字母表	U					H		
密文字母表	Q	R	S	T	U	V	W	X
明文字母表		E	O				I	
密文字母表	Y	Z						
明文字母表	N							

（iii）第三位字母与末字母频率分析

对于第三位字母与末字母的分析，同（ｉ）、（ii），这里不再赘述。

下面，我们用密文进行第六、七轮演示：（程序见附件）

输入密文，得到如下运行结果（见图 6、图 7），并制成表格（见表 7、表 8）。

图 6　第三位字母频率分析运行图

```
o的个数为：5
p的个数为：13
q的个数为：1
r的个数为：6
s的个数为：8
t的个数为：1
u的个数为：1
v的个数为：12
w的个数为：34
x的个数为：9
y的个数为：6
z的个数为：13
w --> E
p --> S
z --> A
v --> R
a --> N
x --> I
```

表 7　第三位字母替换表

密文字母表	A	B	C	D	E	F	G	H
明文字母表	N							
密文字母表	I	J	K	L	M	N	O	P
明文字母表								S
密文字母表	Q	R	S	T	U	V	W	X
明文字母表						R	E	I
密文字母表	Y	Z						
明文字母表		A						

图 7　末字母频率分析运行图

```
输出最后一个字母：

a的个数为：14
b的个数为：28
c的个数为：0
d的个数为：0
e的个数为：9
f的个数为：0
g的个数为：0
h的个数为：8
i的个数为：0
j的个数为：11
k的个数为：0
l的个数为：3
m的个数为：0
```

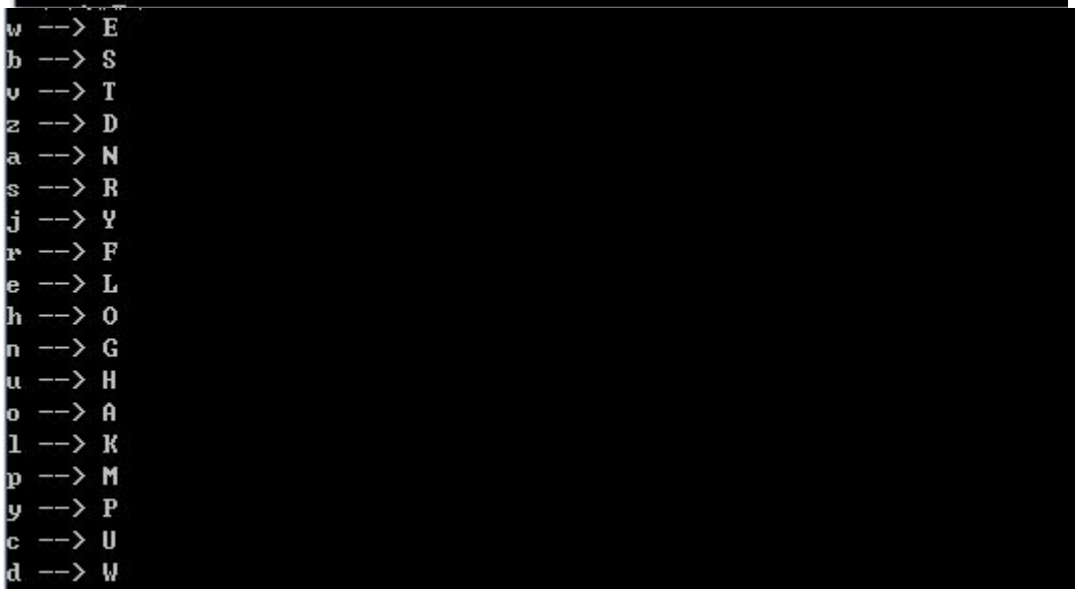


表 8　末字母替换表

密文字母表	A	B	C	D	E	F	G	H
明文字母表	N	S	U	W	L			O
密文字母表	I	J	K	L	M	N	O	P
明文字母表		Y		K		G	A	M
密文字母表	Q	R	S	T	U	V	W	X
明文字母表		F	R		H	T	E	
密文字母表	Y	Z						
明文字母表	P	D						

根据表 2 至表 8 的对应结果，以及频率的规则，我们先取各表频数最高的两位，比对后将这些概率大的几对字母替换判为正确；然后，对比几张表格，将表格中对应重合高的替换判断为正确替换。由此得到 7 对替换。

4.1.5 单词匹配分析

利用已经得到的这 7 对替换，将密文中的这 7 个字母用相应字母替换后，输入程序，

再用最常见的 50 个单词在程序中进行匹配，得出其它的一些替换（程序见附件，运行结果见图 8），并制成表格（见表 9）。

图 8　单词匹配分析运行图

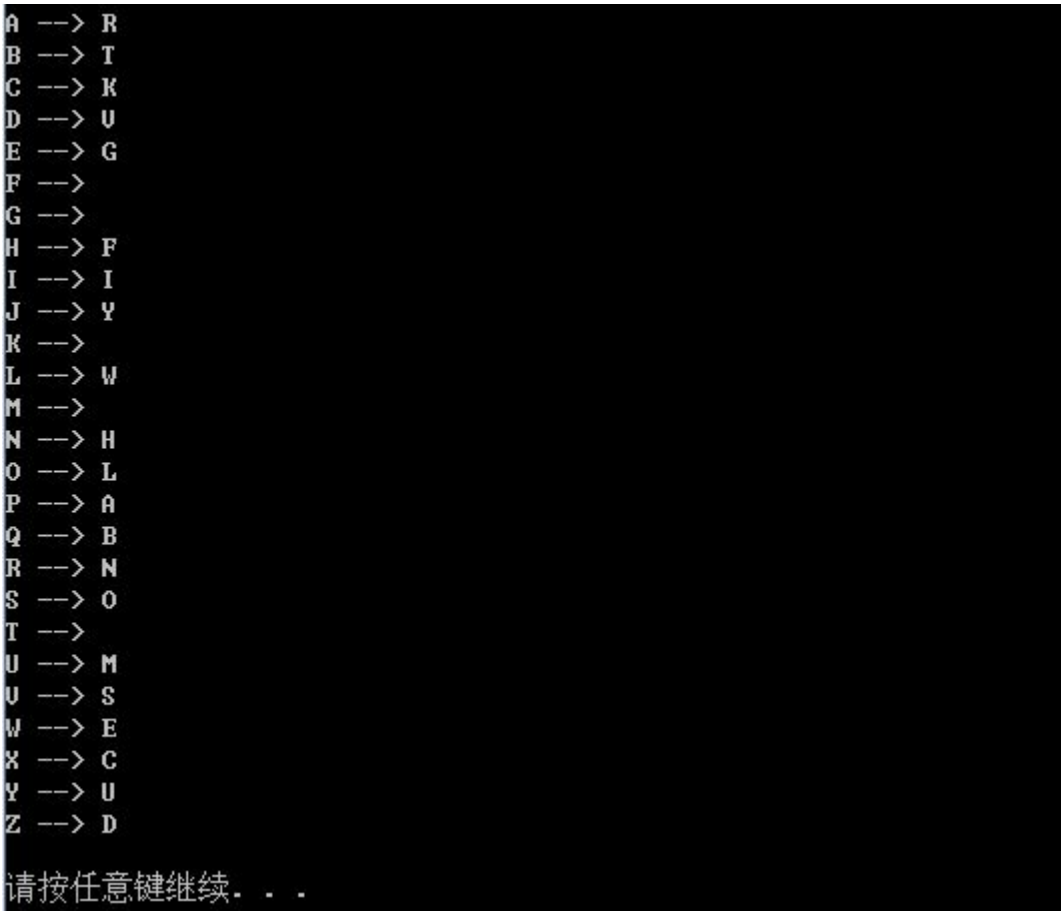


表 9　单词匹配字母替换表

密文字母表	A	B	C	D	E	F	G	H
明文字母表	R	T	K	V	G			F
密文字母表	I	J	K	L	M	N	O	P
明文字母表	I	Y		W		H	L	A
密文字母表	Q	R	S	T	U	V	W	X
明文字母表	B	N	O		M	S	E	C
密文字母表	Y	Z						
明文字母表	U	D						

4.1.6 人工辅助判断

根据表 2 至表 8，我们能大致确定一些字母的替换（见表 9），由此我们可以知道密文中还有 F、G、K、M、T 这 5 个字母没有确定，明文中则剩下 J、P、Q、X、Z 这 5 个字母。现在，我们将借助英文单词书写规律进行人工确定。

例如，密文中的 npffirwvv，我们根据表 9 可将其替换为 ha**iness，我们很容易就

能猜出单词是 happiness，因此密文中的字母 F 应与 P 替换。以此类推，我们可以推出剩余替换是 G——X，K——Z，M——Q，T——J。因此，我们得到最终的字母替换表（见表 10）为

表 10　字母替换表

密文字母表	A	B	C	D	E	F	G	H
明文字母表	R	T	K	V	G	P	X	F
密文字母表	I	J	K	L	M	N	O	P
明文字母表	I	Y	Z	W	Q	H	L	A
密文字母表	Q	R	S	T	U	V	W	X
明文字母表	B	N	O	J	M	S	E	C
密文字母表	Y	Z						
明文字母表	U	D						

4.2 问题二模型的建立与求解

4.2.1 第一条标准：程序运行所得的字母替换表中确定的替换个数

我们采用字母频率分析的方法，即以密文中各字母出现的频数对应已知的频率表来确定字母的替换。在模型中采用的规则以及规则组合的不同，会使得我们得到的字母替换表中，可以确定的替换的个数也不同。一般情况下，由程序得到的替换越多，密文破译后的准确性就越高。这里，我们用程序替换出字母的个数与字母总数之比来定义算法的破译率，即

$$\text{破译率}=\frac{\text{替换出的字母个数}}{\text{字母总数}}\times 100\%$$

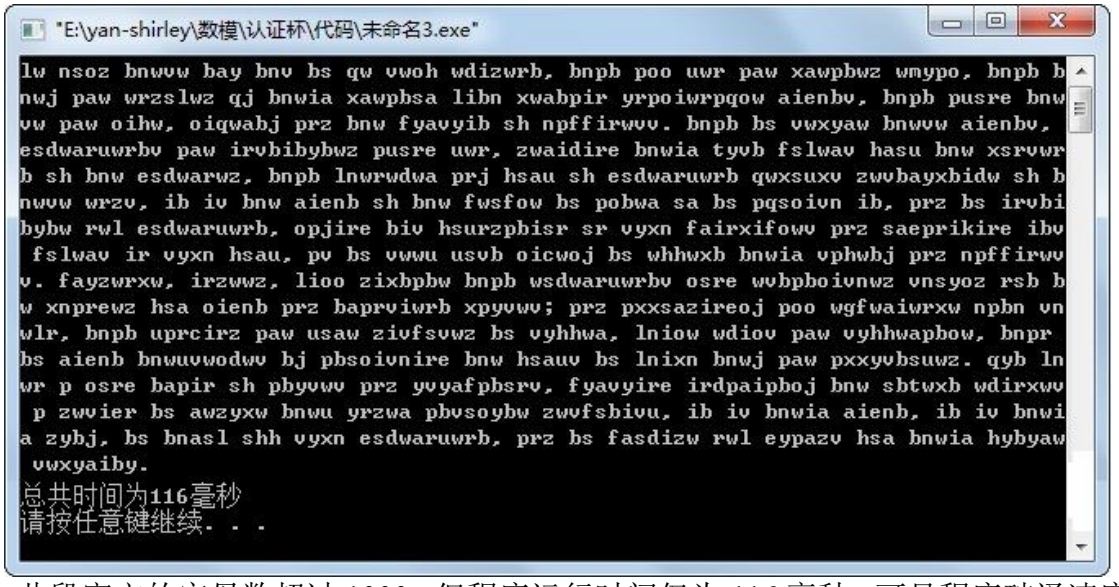
对例文进行问题一的程序运行后，我们已得到 21 个字母相应的替换，因此我们易得，程序对此密文的破译率达到 80.77%。此后，我们也用该程序试验过几篇密文，其破译率维持在 80%左右，可见，此程序的破译能力较高。

4.2.2 第二条标准：程序得到字母替换表所需的时间

对于密码破解性能的测试还要看指定时间范围内可以运行多少次的运算，因此，我们在模型算法中，加上了计算运行时间的程序（见附件）（以毫秒为单位），在最后得到的结果中，会显示我们的程序破译密文所需的时间，以此来衡量程序的破译能力。

运行结果如下图 9。

图 9　程序运行时间



此段密文的字母数超过 1000，但程序运行时间仅为 116 毫秒，可见程序破译速度较快。

4.2.3 第三条标准：破译明文的可靠性

依照我们破译出的字母替换表，将密文翻译成明文。阅读明文，检验明文中单词的出错率以及明文整体的语句通顺程度，以此作为判断本文模型破译能力的重要标准。

以下为我们破译出的明文：

we hold these truths to be self-evident, that all men are created equal, that they are endowed by their creator with certain unalienable rights, that among these are life, liberty and the pursuit of happiness. that to secure these rights, governments are instituted among men, deriving their just powers from the consent of the governed, that whenever any form of government becomes destructive of these ends, it is the right of the people to alter or to abolish it, and to institute new government, laying its foundation on such principles and organizing its powers in such form, as to them shall seem most likely to effect their safety and happiness. prudence, indeed, will dictate that governments long established should not be changed for light and transient causes; and accordingly all experience hath shewn, that mankind are more disposed to suffer, while evils are sufferable, than to right themselves by abolishing the forms to which they are accustomed. but when a long train of abuses and usurpations, pursuing invariably the same object evinces a design to reduce them under absolute despotism, it is their right, it is their duty, to throw off such government, and to provide new guards for their future security.

通过我们的阅读检验后发现，破译的明文中单词的出错率为 0，整片文章语句通顺，没有明显的错误。

综上三条标准的评估结果，我们认为我们模型的破译能力较高，可行性较大。

5 存在的问题

1、本文所采用的频率分析规则仅适用于一般性的文章，对于那些专业性较强的文章以及一些具有特殊情况的文章，例如 Ernest Vincent Wright 的《Gadsby》一书中为用到字母 e，用此类的文章作为明文的密文都不适用本模型。

2、程序不一定能得到所有的字母替换，有时需要依靠人工破译，这就需要破译人员具有一定的英语素养，这就是模型存在的局限性。

6 参考文献

[1]吴干华. 基于频率分析的代替密码破译方法及其程序实现[J]. 福建电脑,2006,09:125+127.

[2]王彩霞. 密码分析中几种方法的研究及其设计与实现[D].西北大学,2004.

[3]吴俊杰. 密码的编制与破译:破译时间的研究[J]. 中国信息技术教育,2014,19:78-79.

附件一：

【单个字母频数分析】

```
#include<stdio.h>
#include<string.h>
const int max_L=1000;
void main()
{
	char text1[max_L],text2[max_L];
	int text_a[26];
	char text_b[26]={'E','T','A','O','I','N','S','H','R','D','L','C','U','M','W','F','G','Y','P','B','V','K','J','X','Q','Z'};
	char text_c[26];
	int i,j,k,max;
	for(i=0;i<max_L;i++){
		text1[i]='\0';
		text2[i]='\0';
	}
	for(i=0;i<26;i++) text_a[i]=0;//初始化 text1,text2,text_a

	gets(text1);
	int len=strlen(text1);
	for(i=0;i<len;i++)
text_a[0]的值  类推
	{
		if(text1[i]>='a'&&text1[i]<='z'){
			k=text1[i]-'a';
			text_a[k]+=1;
		}
	}
	for(i=0;i<26;i++)
	{
		printf("%c 的个数为：%d\n",i+'a',text_a[i]);
	}

	for(k=0;k<26;k++){
		for(i=1,max=0;i<26;i++)
		{
			if(text_a[i]>text_a[max]) max=i;
		}
		text_c[k]=max+'a';
		text_a[max]=-1;
	}
	for(i=0;i<26;i++){
		printf("%c ",text_c[i]);
	}
	printf("\n");
	 for(i=0;i<26;i++){
		printf("%c --->    %c\n",text_c[i],text_b[i]);
	}
	for(i=0;i<len;i++){
		if(text1[i]<'a'||text1[i]>'z'){
		text2[i]=text1[i];
		continue;
		}
		for(j=0;j<26;j++)
		{
			if(text_c[j]==text1[i])
			break;
```

```
            }
            text2[i]=text_b[j];
        }
        text2[i]='\0';
        puts(text2);
}
```

【查找单个字母单词】

```
#include<stdio.h>
#include<string.h>
char *tok_p;
char * my_strtok()
{
        if(*tok_p=='\0')
        return NULL;
        int i=0;
        for(;i<strlen(tok_p);i++)
        if(tok_p[i]<'a'||tok_p[i]>'z')
        {
            tok_p[i]='\0';
        }
        char *p;
        p=tok_p;
        tok_p+=i;

        return p;
}
void main()
{
        char input[1000];
        char *p;
        tok_p=input;
        gets(input);
        p=my_strtok();
        while(p){
            if(strlen(p)==1)
            puts(p);
            p=my_strtok();
        }
}
```

【双连字母频率分析】

```
#include<stdio.h>
#include<string.h>
const int letter_num=26;
const int max_L=1000;
int text_a[letter_num][letter_num];
int main()
{
        char text1[max_L],text2[max_L];
        char text_b[letter_num*letter_num][3]={"TH","HE","AN","IN","ER","ON","RE", "ED", "ND", "HA",
"AT", "EN", "ES", "OF", "NT", "EA", "TI", "TO", "IO", "LE", "IS", "OU", "AR", "AS", "DE", "RT",
"VE","ON", "ST", "NT","NG", "OR", "ET", "IT", "AR", "TE", "SE",};
        char text_c[letter_num*letter_num][3];
        int i,j,k,l,max1,max2;
```

```
//    printf("len=%d",strlen(text_b[1]));
//    for(i=0;i<26;i++) text_a[i]=0;
      for(int i=1;i<max_L;i++)
            text2[i]='\0';
      for(i=0;i<letter_num*letter_num;i++)
      for(j=0;j<3;j++)
            text_c[i][j]='\0';
      gets(text1);
      int len=strlen(text1);
      for(i=0;i<len-1;i++)
      {
            if(text1[i]>='a'&&text1[i]<='z' && text1[i+1]>='a'&&text1[i+1]<='z'){
                  k=text1[i]-'a';
                  l=text1[i+1]-'a';
                  text_a[k][l]+=1;
            }
      }
      for(i=0;i<26;i++)
      {
            //printf("%c 的个数为：%d\n",i+'a',text_a[i]);
      }

      for(i=0;i<len-1;i++)
      {
            for(k=0,max1=0,max2=0;k<26;k++)
            {
                  for(l=0;l<26;l++)
                  {
                        if(text_a[k][l]>text_a[max1][max2])
                        {
                              max1=k;
                              max2=l;
                        }
                  }
            }
            if(text_a[max1][max2]==0)
            break;
            text_c[i][0]=max1+'a';
            text_c[i][1]=max2+'a';
            //          i++;
            text_a[max1][max2]=-1;
      }


      for(i=0;i<26;i++){
            //printf("%c ",text_c[i]);
      }
      printf("\n");
      //映射表
        for(i=0;i<37;i++){
            printf("%s --->    %s\n",text_c[i],text_b[i]);
      }
      for(i=0;i<len;i++)
            text2[i]=text1[i];
      for(i=0;i<len-1;i++){
      if((text1[i]<'a'||text1[i]>'z')||(text1[i+1]<'a'||text1[i+1]>'z')){
      //    text2[i]=text1[i];
```

```
//            text2[i+1]=text1[i+1];
      //      i++;
            continue;
            }

            for(j=0;j<38-1;j++)
            {
                  if(text_c[j][0]=text1[i]&&text_c[j][1]==text1[i+1])
                  break;
            }
            if(j==38-1)
            {
                  text2[i]=text1[i];
            }
            else
            {
                  text2[i]=text_b[j][0];
                  text2[i+1]=text_b[j][1];
            }
      }
      text2[i+1]='\0';
      puts(text2);
      return 0;
}
```

【首字母频率分析】（第二位、第三位字母频率分析程序中改变代码中 flag 的值）

```
#include<string.h>
#include<stdio.h>
int flag=1;
int flag2=0;
char *tok_p;
char * my_strtok()
{
      char *p;
      p=tok_p;
      if(*p=='\0')
      return NULL;
      int i=0;
      for(;i<strlen(tok_p);i++)
      if(tok_p[i]<'a'||tok_p[i]>'z')
      {
            tok_p[i]='\0';
      }


      tok_p+=i;

      return p;
}
int main(void)
{
      char input[100];
      int text_a[26],i,j;
      char *p;
      char      text_flag[4][27]={"TOAWBCDSFMRHIYEGLNOUJK                    ","HOEIAUNRT
","ESARNI                              ","ESTDNRYFLOGHAKMPUW            "};
```

```
    char text_falg2[26];
    int max;
    gets(input);
    tok_p=input;
    for(i=0;i<26;i++) text_a[i]=0;
    p=my_strtok();
    while(p){
        if(flag2==-1)
        flag=strlen(p);
        if(strlen(p)>=flag){
            int k=p[flag-1]-'a';
            text_a[k]+=1;
        }
    //  printf("%c\个数为：%d",*p);

        p=my_strtok();
    }

        for(i=0;i<26;i++)
    {
        printf("%c 的个数为：%d\n",i+'a',text_a[i]);
    }
    for(j=0;j<26;j++)
    {
        for(i=0,max=0;i<26;i++)
        if(text_a[i]>text_a[max])
        max=i;
        text_falg2[j]=max+'a';
        text_a[max]=-1;
    }
//  printf("%c",text_falg2[0]);
    //text_falg2[0]->text_falg[0];
    if(flag2!=-1)
    {
        for(i=0;i<26;i++)
        printf("%c --> %c\n",text_falg2[i],text_flag[flag-1][i]);
    }
    else
    {
        for(i=0;i<26;i++)
        printf("%c --> %c\n",text_falg2[i],text_flag[3][i]);
    }
}
```

【末位字母频率分析】

```
#include<string.h>
#include<stdio.h>
int flag=1;
int flag2=-1;
char *tok_p;
char * my_strtok()
{
    char *p;
    p=tok_p;
    if(*p=='\0')
    return NULL;
```

```
    int i=0;
    for(;i<strlen(tok_p);i++)
    if(tok_p[i]<'a'||tok_p[i]>'z')
    {
        tok_p[i]='\0';
    }
    tok_p+=i;
    return p;
}
int main(void)
{
    char input[100];
    int text_a[26],i,j;
    char *p;
    char    text_flag[4][27]={"TOAWBCDSFMRHIYEGLNOUJK                    ","HOEIAUNRT
","ESARNI                    ","ESTDNRYFLOGHAKMPUW          "};
    char text_falg2[26];
    int max;
    gets(input);
    tok_p=input;
    for(i=0;i<26;i++) text_a[i]=0;
    p=my_strtok();
    while(p){
        if(flag2==-1)
        flag=strlen(p);
        if(strlen(p)>=flag){
            int k=p[flag-1]-'a';
            text_a[k]+=1;
        }
    //  printf("%c\个数为：%d",*p);
        p=my_strtok();
    }
        for(i=0;i<26;i++)
    {
        printf("%c 的个数为：%d\n",i+'a',text_a[i]);
    }
    for(j=0;j<26;j++)
    {
        for(i=0,max=0;i<26;i++)
        if(text_a[i]>text_a[max])
        max=i;
        text_falg2[j]=max+'a';
        text_a[max]=-1;
    }
//  printf("%c",text_falg2[0]);
    //text_falg2[0]->text_falg[0];
    if(flag2!=-1)
    {
        for(i=0;i<26;i++)
        printf("%c --> %c\n",text_falg2[i],text_flag[flag-1][i]);
    }
    else
    {
        for(i=0;i<26;i++)
        printf("%c --> %c\n",text_falg2[i],text_flag[3][i]);
    }
}
```

【单词匹配分析】

```
#include<stdio.h>
#include<string.h>
#define MAX_size 2000
int flag=1,degree=0;
void Index(char str[],char word[],int position[])
{
 int i,len_str,len_word,pos_str,pos_word,k=0,word_number=0;
 len_word=strlen(word);
 len_str=strlen(str);
 for(i=0;i<len_str;)
 {
  while(str[i]==' ')
   i++;
  word_number++;
  for(pos_str=i,pos_word=0;pos_str<len_str && pos_word<len_word;pos_str++,pos_word++)
  {
   if(str[pos_str]!=word[pos_word])
    break;
  }
  if(pos_word==len_word && (str[pos_str]=='\0'|| str[pos_str]==' '))
  {
   position[k++]=word_number;
   degree++;
   flag=0;
  }
  else
  {
   while(str[pos_str]!=' ' && pos_str<len_str)
    pos_str++;
  }
  i=pos_str;
 }
}
void main()
{
 char str[MAX_size],word[50];

 int position[100],i;
 printf("input: \n");
 gets(str);
 printf("need: \n");
 gets(word);
 Index(str,word,position);
  printf("%c --> %c\n",text_falg2[i],text_flag[flag-1][i]);
}
```

【程序运行时间】

```
#include<string.h>
#include<stdio.h>
#include<time.h>
int flag=2;
int flag2=0;
char *tok_p;
```

```
const int max_L=2000;
char * my_strtok()
{
	if(*tok_p=='\0')
	return NULL;
	int i=0;
	for(;i<strlen(tok_p);i++)
	if(tok_p[i]<'a'||tok_p[i]>'z')
	{
		tok_p[i]='\0';
	}
	char *p;
	p=tok_p;
	tok_p+=i;

	return p;
}

void main()
{
	char text1[max_L],text2[max_L],text3[max_L];
	int text_a[26];
	char text_b[26]={'e','t','a','o','i','n','s','h','r','d','l','c','u','m','w','f','g','y','p','b','v','k','j','x','q','z'};
	char text_c[26];
	char	text_flag[4][27]={"TOAWBCDSFMRHIYEGLNOUJK	","HOEIAUNRT
","ESARNI	","ESTDNRYFLOGHAKMPUW	"};
	char text_falg2[26];
	char *p;
	int i,j,k,max,a,b;

	for(i=0;i<max_L;i++){
		text1[i]='\0';
		text2[i]='\0';
	}
	for(i=0;i<26;i++) text_a[i]=0;

	gets(text1);
	double start=clock();
	strcpy(text3,text1);
	int len=strlen(text1);
	for(i=0;i<len;i++)
	{
		if(text1[i]>='a'&&text1[i]<='z'){
			k=text1[i]-'a';
			text_a[k]+=1;
		}
	}
/*
	for(i=0;i<26;i++)
	{
		printf("%c 的个数为：%d\n",i+'a',text_a[i]);
	}
	*/
	for(k=0;k<26;k++){
		for(i=1,max=0;i<26;i++)
		{
			if(text_a[i]>text_a[max]) max=i;
```

```
        }
        text_c[k]=max+'a';
        text_a[max]=-1;
}
for(i=0;i<26;i++){
        printf("%c ",text_c[i]);
}
printf("\n");
  for(i=0;i<26;i++){
        printf("%c --->    %c\n",text_c[i],text_b[i]);
}
for(i=0;i<len;i++){
        if(text1[i]<'a'||text1[i]>'z'){
        text2[i]=text1[i];
        continue;
        }
        for(j=0;j<26;j++)
        {
                if(text_c[j]==text1[i])
                break;
        }
        text2[i]=text_b[j];
}
text2[i]='\0';
puts(text2);
printf("以上全字母\n\n");

printf("找出单字母：\n\n");
strcpy(text1,text3);
tok_p=text1;
//gets(input);
p=my_strtok();
while(p){
        if(strlen(p)==1)
        puts(p);
        p=my_strtok();
}



for(flag=1;flag<4;flag++){
        printf("输出第%d 个字母个数：\n\n",flag) ;
        strcpy(text1,text3);
        tok_p=text1;
        for(i=0;i<26;i++) text_a[i]=0;
        p=my_strtok();
        while(p){
                if(flag2==-1)
                flag=strlen(p);
                if(strlen(p)>=flag){
                        int k=p[flag-1]-'a';
                        text_a[k]+=1;
                }
        //      printf("%c\个数为：%d",*p);

                p=my_strtok();
        }
```

```
        for(i=0;i<26;i++)
    {
        printf("%c 的个数为：%d\n",i+'a',text_a[i]);
    }
    for(j=0;j<26;j++)
    {
        for(i=0,max=0;i<26;i++)
        if(text_a[i]>text_a[max])
        max=i;
        text_falg2[j]=max+'a';
        text_a[max]=-1;
    }
//  printf("%c",text_falg2[0]);
    //text_falg2[0]->text_falg[0];
    if(flag2!=-1)
    {
        for(i=0;i<26;i++)
        printf("%c --> %c\n",text_falg2[i],text_flag[flag-1][i]);
    }
    else
    {
        for(i=0;i<26;i++)
        printf("%c --> %c\n",text_falg2[i],text_flag[3][i]);
    }
}
    printf("输出最后一个字母：\n\n");
    strcpy(text1,text3);
    tok_p=text1;
    flag2=-1;
    for(i=0;i<26;i++) text_a[i]=0;
    p=my_strtok();
    while(p){
        if(flag2==-1)
        flag=strlen(p);
        if(strlen(p)>=flag){
            int k=p[flag-1]-'a';
            text_a[k]+=1;
        }
    //  printf("%c\个数为：%d",*p);

        p=my_strtok();
    }

        for(i=0;i<26;i++)
    {
        printf("%c 的个数为：%d\n",i+'a',text_a[i]);
    }
    for(j=0;j<26;j++)
    {
        for(i=0,max=0;i<26;i++)
        if(text_a[i]>text_a[max])
        max=i;
        text_falg2[j]=max+'a';
        text_a[max]=-1;
    }
//  printf("%c",text_falg2[0]);
```

```
        //text_falg2[0]->text_falg[0];
        if(flag2!=-1)
        {
            for(i=0;i<26;i++)
            printf("%c --> %c\n",text_falg2[i],text_flag[flag-1][i]);
        }
        else
        {
            for(i=0;i<26;i++)
            printf("%c --> %c\n",text_falg2[i],text_flag[3][i]);
        }

    double finish=clock();

    printf("总共时间为%.0lf 毫秒\n",finish-start);
}
```

附件二：

【常用单词表】

Rank	Word	Part of speech	Frequency	Dispersion
3	and	c	10741073	0.99
9	to	i	3856916	0.99
16	with	i	2683014	0.99
17	on	i	2485306	0.99
26	from	i	1635914	0.99
80	two	m	511027	0.99
1	the	a	22038615	0.98
5	a	a	10144200	0.98
6	in	i	6996437	0.98
7	to	t	6332195	0.98
13	for	i	3281454	0.98
22	at	i	1767638	0.98
28	not	x	1638830	0.98
33	as	c	1296879	0.98
37	can	v	1022775	0.98
43	all	d	892102	0.98
45	make	v	857168	0.98
51	one	m	768232	0.98
52	time	n	764657	0.98
57	when	c	678626	0.98
60	some	d	674193	0.98
84	way	n	470401	0.98
86	first	m	463566	0.98
93	no	a	402222	0.98
95	find	v	395203	0.98
98	give	v	384503	0.98
2	be	v	12545825	0.97
4	of	i	10343885	0.97
8	have	v	4303955	0.97
12	that	c	3430996	0.97
23	but	c	1776767	0.97
29	n't	x	1619007	0.97
32	or	c	1379320	0.97
36	their	a	1083029	0.97
38	who	p	1018283	0.97
40	if	c	933542	0.97
41	would	v	925515	0.97
48	will	v	824568	0.97
59	them	p	677870	0.97

63	take	v	670745	0.97
65	into	i	668172	0.97
73	than	c	579757	0.97
75	other	j	547799	0.97
76	how	r	538893	0.97
79	our	a	525107	0.97
81	more	r	517536	0.97
88	new	j	435993	0.97
90	day	n	432773	0.97
91	more	d	420170	0.97
99	many	d	385348	0.97