**齐鲁工业大学实验报告** 成绩

课程名称 密码分析 指导教师 孔志刚 实验日期 2021.10

院（系） 网络空间安全学院 专业班级 网安19-1 实验地点 南213214

学生姓名 焦翔宇 学号 201903150004 同组人 徐弘硕、陈宇隆

# 密码分析过程

## 待解决问题

**分析未知密码密文：**

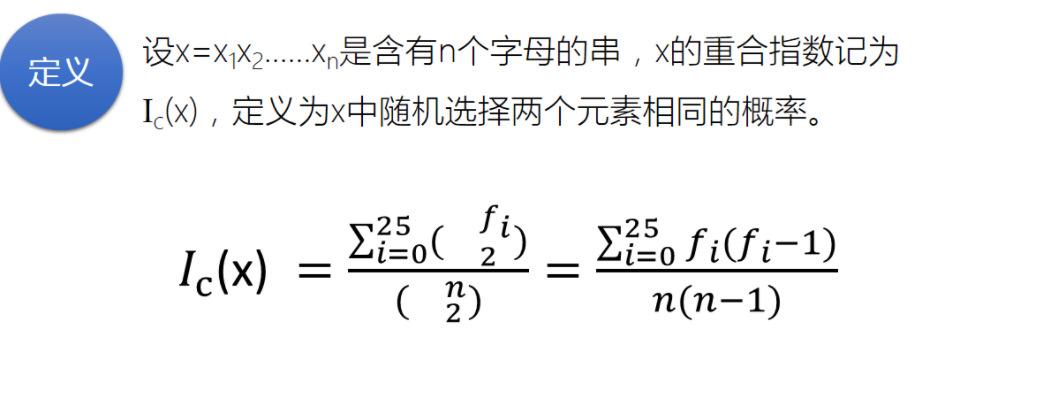
**BNVSNSIHQCEELSSKKYERIFJKXUMBGYKAMQLJTYAVFBKVTDVBPVVRJYYLAOKYMPQSCGDLFSRLLPROYGESEBUUALRWXMMASAZLGLEDFJBZAVVPXWICGJXASCBYEHOSNMULKCEAHTQOKMFLEBKFXLRREDTZXCIWBJSICBGAWDVYDHAVEJXZIBKCGJIWEAHTTOEWTUHKRQVVRGZBXYIREMMASCSPBHLHJMBLRFFJELHWEYLWISTFVVYEJCMHYUYRUFSFMGESIGRLWALSWMNUHSIMYYITCCQPZSICEHBCCMZFEGVJYOCDEMMPGHVAAUMELCMOEHVLTIPSUYILVGFLMVWDVYDBTHFRAYISYSGKVSUUHYHGGCKTMBLRX**

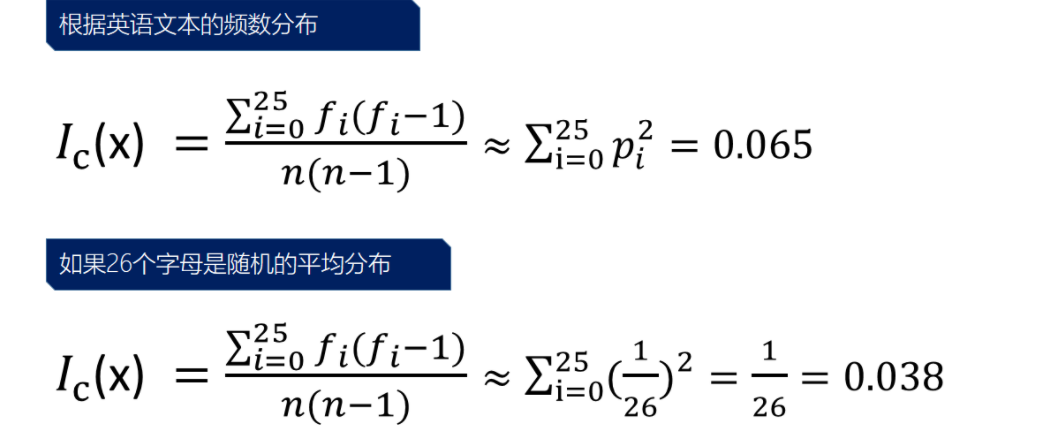
## 题目分析

移位、仿射、代换密码都是单表代换，维吉尼亚密码是多表代换。单表代换不会改变文本的重合指数，多表代换会使重合指数发生变化，由原英文文本的0.065向完全随机文本的重合指数0.038变化。通过验证密文的重合指数可以区分出单表代换和多表代换。其中单表代换中的移位密码和仿射密码又可以通过穷举密钥加明文指数测试的方法求解出明文，如果不能给出明文解。说明不是移位密码或仿射密码，大概率是代换密码，需要通过频数分析的方法来求解。

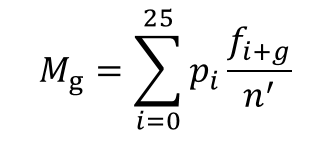
**相关概念**

### 重合指数Ic





### 明文指数Mg



Mg是英语文本的关于频数分布的度量。

Ic用于判断密文是否是单表单换的结果。

Mg是判断英语文本是否不是明文(有意义的文本)。

可以用Mg来判断是否是明文。

## 实验过程

### 1、求密文的重合指数，判断是哪一种加密方式

经计算密文的重合指数mg=0.041814,基本上可以确定加密方式是维吉尼亚密码。

下表为密文中各个字母出现次数和对应频率。



### 2、分析维吉尼亚密码

#### (1)确定分组数m。

#### 方法一：Kasiski测试法

根据重复出现的密文串的位置来确定m的值，找出重复出现的长度为3的密文串，计算它们之间的距离，并求解最大公约数来确定分组长度。由于密文长度太短，又或者是该密文是特殊构造的，用Kasiski测试法没有找到重复出现3次以上的长度为3的字符串。

**Kasiski代码**

1. **int** Kasiski(string s)
2. {
3. **int** m = 0;
4. **int** len = s.length();
5. **for** (**int** I = 0; I < len – 2; i++)
6. {
7. string chr = “”;
8. chr = chr + s[i] + s[I + 1] +s[I + 2];
9. **int** c = 0;//记录出现次数
10. **int** index[100];
11. memset(index, 0, **sizeof**(index));
12. **for** (**int** j = 0; j < len – 2; j++)
13. {
14. string tmp = “”;
15. tmp = tmp + s[j] + s[j + 1] +s[j + 2];
17. **if** (chr == tmp)
18. {
19. index[c++] = j;
20. }
21. }
22. **if** (c > 2)
23. {
24. **for** (**int** k = 1; k < c; k++)//计算距离
25. {
26. index[k] = index[k] – index[0];
27. }
28. **int** g = gcd(index[1], index[2]);
29. **for** (**int** k = 3; k < c; k++)
30. {
31. g = gcd(g, index[k]);
32. }
33. **if** (m == 0)m = g;
34. **else** {
35. **if** (m == g)m = g;
36. **else** m = gcd(m, g);
37. }
38. }
39. }
40. **return** m;
41. }

#### 方法二：穷举m值，用重合指数Ic验证

下表为分组为m时，对应各分组重合指数以及重合指数均值(部分表格，详细表格见附录1)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| m |  |  |  |  |  |  | 均值 |
| m=1 | 0.0418 |  |  |  |  |  | 0.0418 |
| m=2 | 0.0456 | 0.0464 |  |  |  |  | 0.046 |
| m=3 | 0.0446 | 0.0481 | 0.0484 |  |  |  | 0.0471 |
| m=4 | 0.0471 | 0.0575 | 0.0465 | 0.0475 |  |  | 0.0497 |
| m=5 | 0.045 | 0.0425 | 0.0418 | 0.0455 | 0.037 |  | 0.0424 |
| **m=6** | **0.0512** | **0.0635** | **0.055** | **0.0698** | **0.0576** | **0.0698** | **0.0612** |

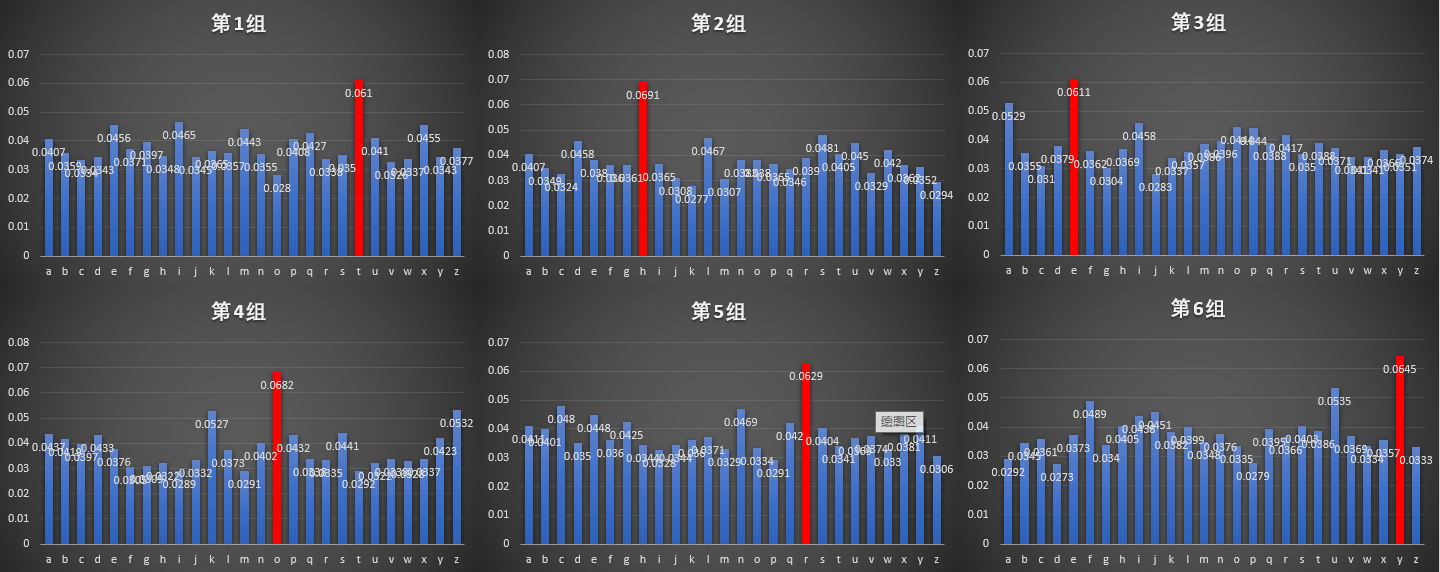
发现m=6和m=12时，重合指数均接近0.065，所以基本可以确定m=6。

**代码**

1. **double** mmg = 0;
2. fprintf(fp, "输出分组为m的各组重合指数\n");
3. **for** (**int** i = 1; i <= 20; i++)
4. {
5. string tmp[20];
6. **for** (**int** j = 0; j < s.length(); j++)
7. {
8. tmp[j % i] += s[j];
9. }
10. mmg = 0;
11. fprintf(fp,"m=%d,", i);
12. **for** (**int** j = 0; j < i; j++)
13. {
14. tmp[j][tmp[j].length()] = '\0';
15. fprintf(fp,"%.4f,", Mg(tmp[j], **true**));
16. mmg = mmg + Mg(tmp[j], **true**);
17. }
18. mmg = mmg / i;
19. **int** n = 20 - i;
20. **while** (n--)fprintf(fp, ",");
21. fprintf(fp,"%.4f\n", mmg);
22. **if** (mmg >= 0.06)
23. {
24. **if** (m == 0||m==1)m = i;
25. **else** m = gcd(m, i);
26. }
27. }
28. cout <<"m=" << m << endl;

#### (2)求解加密密钥

维吉尼亚密码的每一组是一个简单的移位密码，可以通过穷举密钥的方法来破解，并用明文指数来判断是否是明文。（详细图表见附录1）



经计算，密钥为**theory**

#### (3)根据密钥求解明文

明文：

**igrewupamongslowtalkersmeninparticularwhodroppedwordsafewatatimelikebeansinahillandwhenigottominneapoliswherepeopletookalakewobegoncommatomeantheendnfastoryicouldntspeakawhnlesentenceincompanyandwasconsiderednottoobriahtsoienrolledinaspeechcouqsetaughtbyorvillesandthefounderofreflexiverelaxologyaselfhypnotictechniquethatenabledapersontospeakuptothreehundredwordsperminute**

整理：

**l grew up anong slow talkers,men in particular,who dropped words a few at a time like beansin a hill, and when I got to Minneapolis where people took a Lake Wobegon comma to mean theend of a story, I couldn't speak a whole sentence in company and was considered not too bright.So I enrolled in a speech course taught by Orville Sand, the founder of reflexive relaxology, aself-hypnotic technique that enabled a person to speak up to three hundred words per minute.**

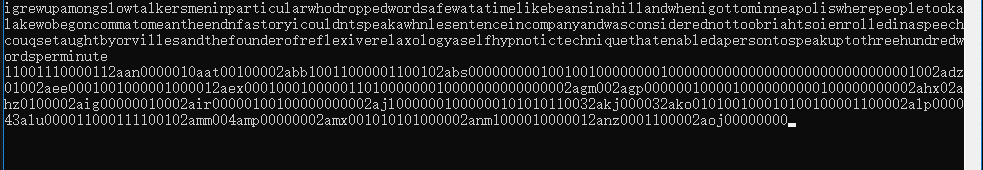
经验证，该明文是有明确意义的字符串，因此可以确定解密过程是正确的。

# 遇到的一些问题

## Kasiski无法正确求解m

在使用Kasiski方法求解m时，并未正确求解出m，经过分析发现，该密文串中没有出现过3次以上的长度为3的字符串，因此，采用该方法不能求出m。猜测可能原因一个是密文太短，另一个是加密密钥特性够好，将相同的明文加密成了不同的密文。再次观察明文之后发现一个问题，明文字符串中的重复出现的长度为3的字符串的次数就很少，再经过分组加密，很容易就被隐藏起来，因此，Kasiski方法分析不出来是正常的。

通过穷举长度为3的密钥，对明文加密后进行Kasiski分析，发现均不能找到正确的m，且有一部分m值异常大，明显是一个错误值。基本上可以确定是明文本身重复字符串较少导致的Kasiski不能正确求解。

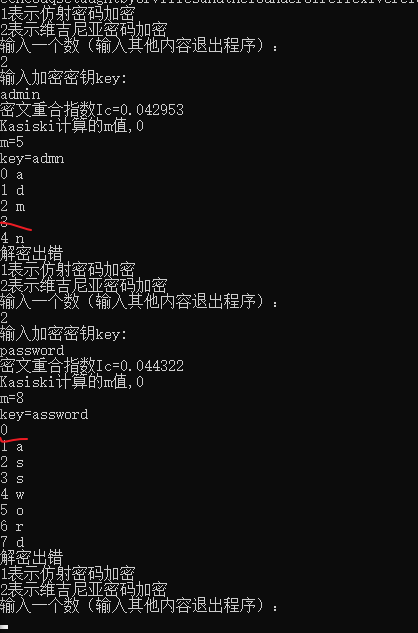




## 求解密钥异常

求解出来的m都是正确的，但是在求解每一个分组的时候，部分分组没有得到结果，经过调试分析发现，原因在于密文串较短，在分组较长或者是一些特殊分组的情况下，求解得到的Mg是小于设置的阈值0.06的，这也就导致了，无法得到正确解。

如图所示



找到问题的原因之后，问题就好解决了，我们在设置阈值的同时，增加一个参数Mg与0.065差值的绝对值来作为辅助参考，在穷举密钥时，每次记录下最小值（即最接近0.065的值）所对应的密钥，如果所有密钥对应的Mg都小于0.06，就将Mg最接近0.065的密钥作为密钥来使用。



# 分析总结

通过使用重合指数，确定了该未知密码采用的是维吉尼亚密码加密方式加密，进而通过使用重合指数和明文指数对密文进行分析破解，成功得到加密密钥theory，并用密钥解出正确明文。

# 附录1 表格

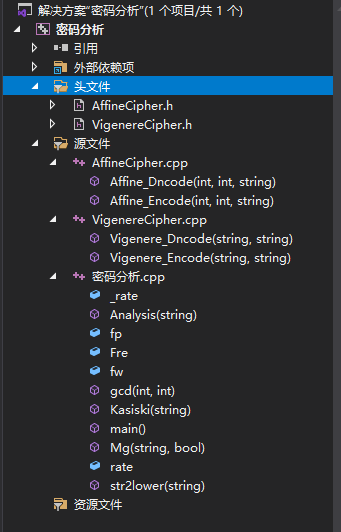
m取不同值时，对应的所有Ic值（m=1,2,3…,20）。



m=6时，各个分组求解密钥过程图表。

# 附录2 代码

## 代码项目结构



## AffineCipher.h

1. #pragma once
2. #include <iostream>
3. **using** **namespace** std;
4. #ifndef AFFINECIPHER\_H
5. #define AFFINECIPHER\_H
6. string Affine\_Encode(**int** a, **int** b, string s);
7. string Affine\_Dncode(**int** a, **int** b, string s);
8. #endif

## AffineCipher.cpp

1. #include <iostream>
2. **using** **namespace** std;
3. string Affine\_Encode(**int** a, **int** b, string s)//仿射密码加密函数
4. {
5. **if** (a == 0 || a % 2 == 0) {
6. **return** "Error";
7. }
8. **int** len = s.length();
9. string C;
10. **for** (**int** i = 0; i < len; i++)
11. {
12. C = C + **char**((a \* **int**(s[i] - 'a') + b + 26) % 26 + 'a');
13. }
14. **return** C;
15. }
16. string Affine\_Dncode(**int** a, **int** b, string s)//仿射密码解密函数
17. {
18. **if** (a == 0 || a % 2 == 0) {
19. **return** "Error";
20. }
21. **int** len = s.length();
22. string P;
23. **for** (**int** i = 0; i < len; i++)
24. {
25. P = P + **char**((a \* **int**(s[i] - 'a' - b + 26)) % 26 + 'a');
26. }
27. **return** P;
28. }

## VigenereCipher.h

1. #pragma once
2. #include <iostream>
3. **using** **namespace** std;
4. #ifndef VIGENERECIPHER\_H
5. #define VIGENERECIPHER\_H
6. string Vigenere\_Encode(string s,string key);
7. string Vigenere\_Dncode(string s,string key);
8. #endif
9. #pragma once

## VigenereCipher.cpp

1. #include "VigenereCipher.h"
3. string Vigenere\_Encode(string s, string key)
4. {
5. **int** len1 = s.length(), len2 = key.length();
6. string c;
7. **for** (**int** i = 0; i < len1; i++)
8. {
9. c = c + **char**((**int**(s[i] - 'a') + (key[i%len2]-'a') + 26) % 26 + 'a');
10. }
11. **return** c;
12. }
14. string Vigenere\_Dncode(string s, string key)
15. {
16. **int** len1 = s.length(), len2 = key.length();
17. string m;
18. **for** (**int** i = 0; i < len1; i++)
19. {
20. m = m + **char**((**int**(s[i] - 'a') - (key[i % len2] - 'a') + 26) % 26 + 'a');
21. }
22. **return** m;
23. }

## 密码分析.cpp

1. // 密码分析.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
2. //
3. #include <iostream>
4. #include<algorithm>
5. #include"AffineCipher.h"
6. #include"VigenereCipher.h"
7. **using** **namespace** std;
8. **FILE**\* fp;
9. **FILE**\* fw;
10. **double** rate[26] = { 0.082,0.015,0.028,0.043,0.127,0.022,0.02,0.061,0.07,0.002,0.008,0.04,0.024,0.067,0.075,0.019,0.001,0.060,0.063,0.091,0.028,0.01,0.023,0.001,0.02,0.001 };
11. **double** \_rate[26];//密文中各个字母出现的概率
12. **int** Fre[26];//字母出现频数
13. string str2lower(string s)
14. {
15. //大写转小写
16. **for** (**int** i = 0; i < s.length(); i++)
17. {
18. **if** (s[i] >= 'A' && s[i] <= 'Z')s[i]+=32;
19. }
20. **return** s;
21. }
22. **int** gcd(**int** a, **int** b) {
23. //计算最大公约数
24. **if** (b == 0) **return** a;
25. **return** gcd(b, a % b);
26. }
27. **double** Mg(string s, **bool** m=**false**)
28. {
29. //计算明文指数或重合指数，第二个参数默认缺省为false，若m为true，则计算重合指数，否则计算明文指数。
30. **int** len = s.length();
31. **double** mg = 0;
32. memset(Fre, 0, **sizeof**(Fre));
33. **for** (**int** i = 0; i < len; i++)
34. {
35. Fre[s[i] - 'a']++;
36. }
38. **for** (**int** i = 0; i < 26; i++)
39. {
40. \_rate[i] = (**double**)Fre[i] / (**double**)len;
42. }
43. **if** (m)//计算重合指数
44. {
45. **for** (**int** i = 0; i < 26; i++)
46. {
47. mg = mg + (\_rate[i] \* (((**double**)Fre[i]-1) / ((**double**)len-1)));
48. }
49. **return** mg;
50. }
51. **for** (**int** i = 0; i < 26; i++)
52. {
53. mg = mg + ((**double**)Fre[i] / (**double**)len) \* rate[i];
54. }
55. **return** mg;
56. }
57. **int** Kasiski(string s)
58. {
59. **int** m = 0;
60. **int** len = s.length();
61. **for** (**int** i = 0; i < len - 2; i++)
62. {
63. string chr = "";
64. chr = chr + s[i] + s[i + 1] +s[i + 2];
65. **int** c = 0;//记录出现次数
66. **int** index[100];
67. memset(index, 0, **sizeof**(index));
68. **for** (**int** j = 0; j < len - 2; j++)
69. {
70. string tmp = "";
71. tmp = tmp + s[j] + s[j + 1] +s[j + 2];
72. **if** (chr == tmp)
73. {
74. index[c++] = j;
75. }
76. }
77. **if** (c > 2)
78. {
79. **for** (**int** k = 1; k < c; k++)//计算距离
80. {
81. index[k] = index[k] - index[0];
82. }
83. **int** g = gcd(index[1], index[2]);
84. **for** (**int** k = 3; k < c; k++)
85. {
86. g = gcd(g, index[k]);
87. }
88. **if** (m == 0)m = g;
89. **else** {
90. **if** (m == g)m = g;
91. **else** m = gcd(m, g);
92. }
93. }
94. }
95. **return** m;
96. }
97. string Analysis(string s)
98. {
99. s = str2lower(s);
100. **double** mg;//明文指数
101. **double** Ic;//重合指数
102. Ic = Mg(s, **true**);
103. printf("密文重合指数Ic=%f\n", Ic);
104. fw = fopen("字母出现概率表.csv", "w");
105. fprintf(fw, "字母,频数,频率\n");
106. **for** (**int** i = 0; i < 26; i++)
107. {
108. fprintf(fw, "%c,%d,%.3f\n", (**char**)i + 'A', Fre[i], \_rate[i]);
109. }
110. fclose(fw);
111. **if** (Ic > 0.06)//Ic=0.065,说明不是维吉尼亚密码，是个单表代换
112. {
113. //先暴力穷举秘钥，判断是不是移位密码或者仿射密码，用明文指数来判断解密出来的明文是否有意义
114. **bool** f = **true**;
115. **for** (**int** i = 1; i < 26; i = i + 2)
116. {
117. **if** (i == 13)**continue**;
118. **for** (**int** j = 1; j < 26; j++)
119. {
120. string m = Affine\_Dncode(i, j, s);
121. mg = Mg(m);
122. **if** (mg > 0.06)
123. {
124. f = **false**;
125. cout << i << " " << j << ":" << m << endl << endl;
126. }
127. }
128. }
129. **if** (f)
130. {
131. cout << "加密方式不是仿射密码或维吉尼亚密码" << endl;
132. }
134. }
135. **else** //mg！=0.065
136. {
137. **if** ((fp = fopen("维吉尼亚分析表.csv", "w")) == NULL)
138. {
139. cout << "Error" << endl;
141. }
142. **int** m = Kasiski(s);
143. fprintf(fp, "Kasiski计算的m值,%d\n", m);
145. printf("Kasiski计算的m值,%d\n", m);
146. **if** (m > 20)m = 0;
147. **double** mmg = 0;
148. fprintf(fp, "输出分组为m的各组重合指数\n");
149. **for** (**int** i = 1; i <= 20; i++)
150. {
151. string tmp[20];
152. **for** (**int** j = 0; j < s.length(); j++)
153. {
154. tmp[j % i] += s[j];
155. }
156. mmg = 0;
157. fprintf(fp,"m=%d,", i);
158. **for** (**int** j = 0; j < i; j++)
159. {
161. tmp[j][tmp[j].length()] = '\0';
162. fprintf(fp,"%.4f,", Mg(tmp[j], **true**));
163. mmg = mmg + Mg(tmp[j], **true**);
164. }
166. mmg = mmg / i;
167. **int** n = 20 - i;
168. **while** (n--)fprintf(fp, ",");
169. fprintf(fp,"%.4f\n", mmg);
170. **if** (m == i && mmg < 0.06)m = 0;
171. **if** (mmg >= 0.06)
172. {
173. **if** (m == 0||m==1)m = i;
174. **else** m = gcd(m, i);
176. }
177. }
179. cout <<"m=" << m << endl;
180. string tmp[20];
181. **for** (**int** i = 0; i < s.length(); i++)
182. {
183. tmp[i % m] += s[i];
184. }
185. string tmpkey = "";
186. string kkey[20];
187. **for** (**int** i = 0; i < m; i++)
188. {
189. **double** mmgtmp = 100;//另一种评价mg的指标
190. **int** jj = 0;
191. string ttmp = tmp[i];
192. **for** (**int** j = 0; j < 26; j++)
193. {
194. string mtmp = Affine\_Dncode(1, j, ttmp);
195. **double** mgtmp = Mg(mtmp);
196. **double** a = 0.065 > mgtmp ? 0.065 - mgtmp : mgtmp - 0.065;
197. **if** (a < mmgtmp) {
198. jj = j;
199. mmgtmp = a;
200. }
201. **if** (mgtmp > 0.06) {
203. tmpkey += (**char**)(j + 'a');
205. kkey[i] += (**char**)(j + 'a');
206. //break;
207. }
208. fprintf(fp, "%.4f,", mgtmp);
210. }
211. **if** (i == tmpkey.length())
212. {
213. tmpkey += (**char**)(jj + 'a');
214. kkey[i] += (**char**)(jj + 'a');
215. }
216. fprintf(fp, "\n");
217. }
218. fclose(fp);
219. printf("key=");
220. cout << tmpkey << endl;
221. **for** (**int** i = 0; i < m; i++)
222. {
223. cout<<i<<" " << kkey[i] << endl;
224. }
225. fp= fopen("明文.txt", "w");
227. fprintf(fp, "%s\n",tmpkey.c\_str());
228. //cout << tmpkey << endl;
229. string M = "decode failed!";
230. **if** (tmpkey.length()) {
231. M = Vigenere\_Dncode(s, tmpkey);
232. **if** (Mg(M) < 0.06) {
233. M = "解密出错";
234. }
235. }
236. fprintf(fp, "%s\n", M.c\_str());
237. fclose(fp);
238. cout << M << endl;
239. **return** M;
240. }
241. **return** s;
242. }
243. **int** main()
244. {
246. string s = "BNVSNSIHQCEELSSKKYERIFJKXUMBGYKAMQLJTYAVFBKVTDVBPVVRJYYLAOKYMPQSCGDLFSRLLPROYGESEBUUALRWXMMASAZLGLEDFJBZAVVPXWICGJXASCBYEHOSNMULKCEAHTQOKMFLEBKFXLRREDTZXCIWBJSICBGAWDVYDHAVEJXZIBKCGJIWEAHTTOEWTUHKRQVVRGZBXYIREMMASCSPBHLHJMBLRFFJELHWEYLWISTFVVYEJCMHYUYRUFSFMGESIGRLWALSWMNUHSIMYYITCCQPZSICEHBCCMZFEGVJYOCDEMMPGHVAAUMELCMOEHVLTIPSUYILVGFLMVWDVYDBTHFRAYISYSGKVSUUHYHGGCKTMBLRX";
247. cout << s.length() << endl;
248. s=Analysis(s);
249. **while** (1)
250. {
251. **int** n=1;
252. cout << "1表示仿射密码加密\n2表示维吉尼亚密码加密\n输入一个数（输入其他内容退出程序）：\n";
253. cin >> n;
254. **if** (n == 1)
255. {
256. **int** a, b;
257. cout << "输入a,b" << endl;
258. cin >> a >> b;
259. string S = Affine\_Encode(a, b, s);
260. **if** (S == "Error")
261. {
262. cout << S << endl;
263. }
264. **else** {
265. Analysis(S);
266. }
267. }
268. **else** **if**(n==2){
269. string key;
270. cout << "输入加密密钥key:\n";
271. cin >> key;
272. string S = Vigenere\_Encode(s, key);
273. Analysis(S);
274. }
275. **else**
276. {
277. **break**;
278. }
279. }
280. **return** 0;
281. }