实验报告（古典密码）

【实验目的】

1、 通过本次实验，了解古典加密算法思想，掌握常见的古典密码。
2、 学会应用古典密码以及针对部分古典密码的破译。

【实验环境】

- Python 3.9.7 64-bit

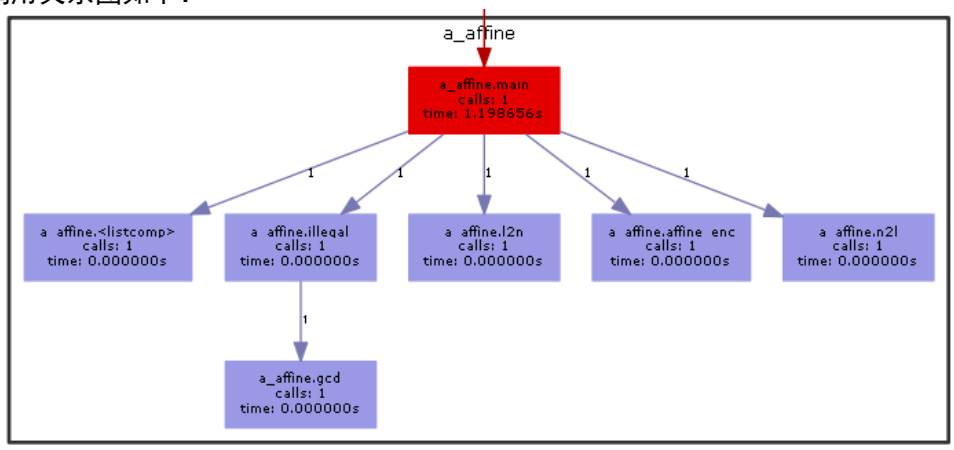
【实验内容】

一、仿射密码

1、算法流程

- 判断密钥k是否合法（`illegal()`函数）；
- 将 letter 转化为 number（`l2n()`函数）；
- 加密：$c=(k\times p+b)\ \mathrm{mod}\ 26$（`affine_enc()`函数），
- 解密：$p=k^{-1}(c-b)\ \mathrm{mod}\ 26$（`affine_dec()`函数）；
- 将 number 转化为 letter 并输出（`n2l()`函数）。

函数调用关系图如下：



2、测试样例及结果截图

```
3 17
thisisatestforaffinecipher
1
wmptptrwdtwghqrggpedxpkmdq
Process finished with exit code 0
```

3、讨论与思考

- 注意字符串读入时出现的'\n'和'\r'，可用`strip().replace()`进行替换。
- 每次加解密前先判断密钥k是否合法。

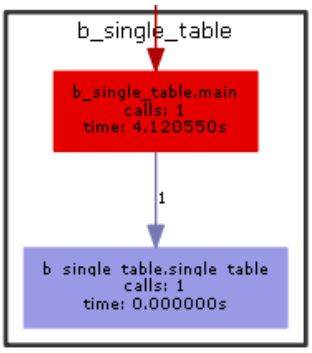
二、单表代替密码

1、算法流程

- 输入置换表中的t_1,t_2，明/密文s；

- 按置换表进行替换（single_table()函数）并输出。

函数调用关系图如下：



2、测试样例及结果截图

```
abcdefghijklmnopqrstuvwxyz
qazwsxedcrfvtgbyhnujmiklop
doyouwannatodance
1
wbobmkqggqjbwqgzs
Process finished with exit code 0
```

3、讨论与思考

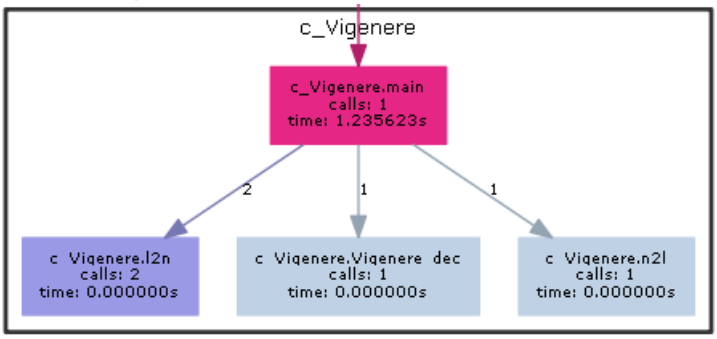
- 可以用 python 自带的.index()函数通过索引查找进行替换。

三、维吉尼亚密码

1、算法流程

- 输入密钥k，明/密文s，将其转化为 number（l2n()函数）；
- 加密：$c=(p+k) \bmod 26$（Vigenere_enc()函数），
- 解密：$p=(c-k) \bmod 26$（Vigenere_dec()函数）；
- 转化为 letter 并输出（n2l()函数）。

函数调用关系图如下（解密样例）：



2、测试样例及结果截图

```
interesting
zhonghuaminzuweidafuxing
1
huhrxlmtuvthhpizhsyckovt
Process finished with exit code 0
```

3、讨论与思考

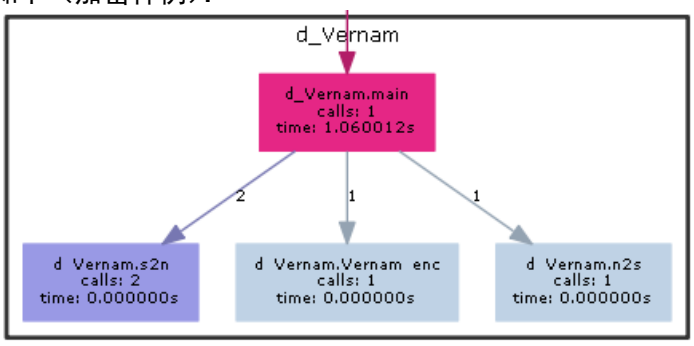
- 仍需要注意字符串读入时出现的'\n'和'\r'。

四、弗纳姆密码

1、算法流程

- 输入密钥k，明/密文s，将其转化为 number（s2n()函数）；
- 加密：$c = p \oplus k$（Vernam_enc()函数），
- 解密：$c = p \oplus k$（Vernam_dec()函数）；
- 转化为 letter 并输出（n2s()函数）。

函数调用关系图如下（加密样例）：



2、测试样例及结果截图

```
<v9nk[P
h,wX^%hy-rO?<>i.j\:y
0
TZN65~8E[K!TgnUXS2Q"
Process finished with exit code 0
```

3、讨论与思考

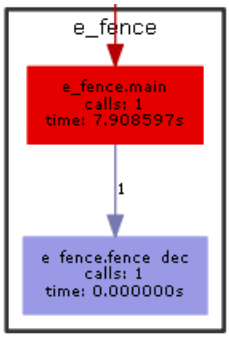
- 同弗吉尼亚密码一样，可用mod len(k)进行分组计算。
- python 中直接用^进行异或计算。

五、栅栏密码

1、算法流程

- 输入密钥k，明/密文s；
- 加密：将明文分别间隔k个字符输出得到密文（fence_enc()函数），
- 解密：将密文拆分为k行，按列输出得到明文（fence_dec()函数）。

函数调用关系图如下（解密样例）：



2、测试样例及结果截图

```
3
whateverisworthdoingisworthdoingwell
1
wtesrdnsrdneherwtogwtoglaviohiiohiwl

Process finished with exit code 0
```

3、讨论与思考

- 解密时比想象中的要复杂，需要注意$k \nmid \text{len}(s)$的情况

六、希尔密码

1、算法流程

- 输入n，密钥矩阵K，明/密文s，将s转化为 number（`l2n()`函数）；
- 加密：$C_{1\times n}=(P_{1\times n}\cdot K_{n\times n})\ \bmod\ 26$，
- 解密：$P_{1\times n}=(C_{1\times n}\cdot K_{n\times n}^{-1})\ \bmod\ 26$；
- 将加/解密结果转化为 letter（`n2l()`函数）并输出。

伪代码：

Algorithm 1：矩阵行列式

Input: n阶方阵A
Output: A的行列式$|A|$

```
function "det( )"
    if A为1阶 then return A_11
    else
        det_A ← 0
        /* A*_ij： A_ij的代数余子式 */
        for j ← 1 to n do
            det_A ← det_A + (-1)^(1+j) × A_1j · det(A*_1j)
        end for
        return det_A
    end if
end function
```

Algorithm 2：矩阵求逆

Input: n阶方阵A
Output: A^{-1}

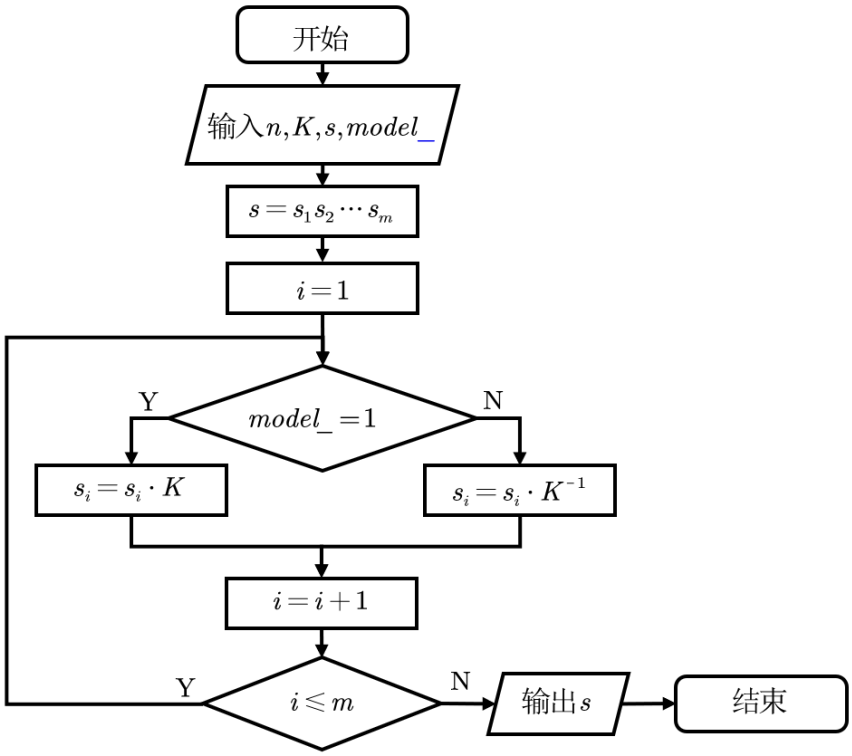
```
function "inv( )"
    inv_A ← O_n×n    /* initialization */
    for (i,j) ← (1,1) to (n,n) do
        inv_A_ij ← (-1)^(i+j) × det(A*_ij) · [det(A)]^-1 mod 26
    end for
    return inv_A
end function
```

Algorithm 3： 希尔密码

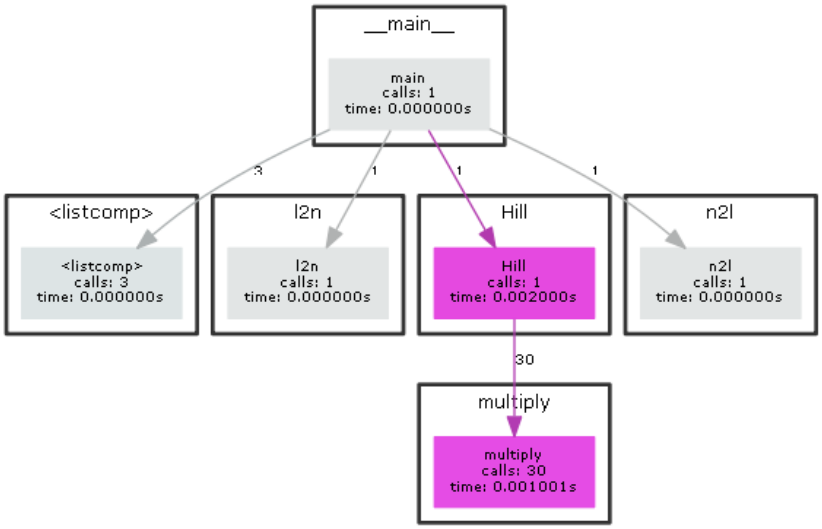
Input: 矩阵维度n, 密钥矩阵K, 明/密文s
Output: 加/解密结果s

1. **function** "Hill( )"
2. 将s以n个一组分为s_i $/*$ n阶行向量 $*/$
3. **for** each s_i **do**
4. $s_i \leftarrow s_i \cdot K$ **if** "加密" **else** $s_i \leftarrow s_i \cdot \mathrm{inv}(K)$
5. **end for**
6. **return** inv_A
7. **end function**

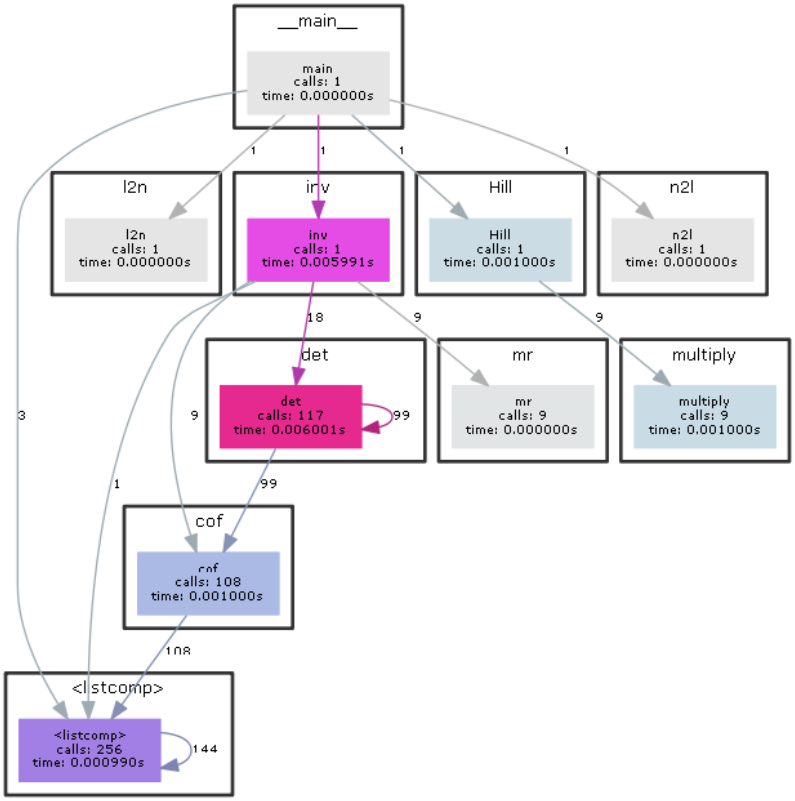
流程图：



函数调用关系图如下（加密样例）：



函数调用关系图如下（解密样例）：



2、测试样例及结果截图

```
3
10 19 13
19 24 5
24 9 2
kibounohanatsunaidakizunagaimabokuranomunenonakaniarukarakeshitechirukotohanaiikiruchikara
1
qbqafwnsjoczodwqlusaoggjkoewyiwdzdihipdxavgyfigywxbkjdlshkgeccnmfyzmvqvensjkhduwoapnuzvlsh

Process finished with exit code 0
```

3、讨论与思考

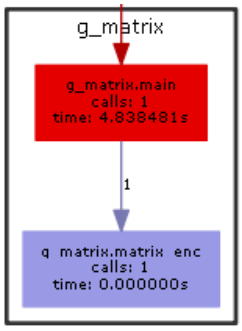
- 该算法难度较大，主要在于矩阵求逆。对于矩阵求逆，常见的有伴随矩阵和矩阵变换两种方法，我们采用的前者，并通过迭代（按列展开）来计算行列式，显然**以时间的代价换取了算法的简洁程度**。
- 笔者发现该实验测试时最大运行时间达到了 890ms，这也印证了上面说的一点，同时意味着在做希尔密码明文攻击时面临着超时的风险，这将在**实验九**继续讨论。

七、矩阵密码

1、算法流程

- 输入密钥长度n，密钥k，明/密文s；
- 加密（`matrix_enc()`函数）：以n个一组将s分组，按照密钥k的顺序输出，
- 解密（`matrix_dec()`函数）：将s分为n组，按密钥逆序输出。

函数调用关系图如下（加密样例）：



2、测试样例及结果截图

```
7
4312567
attackpostponeduntiltwoamxyz
1
ttnaaptmtsuoaodwcoixknlypetz
Process finished with exit code 0
```

3、讨论与思考

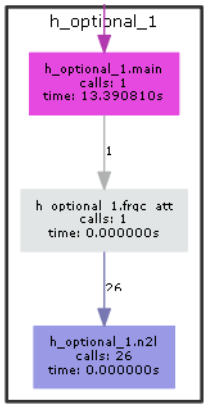
- 通过.index()找索引。

八、加法密码の字母频率攻击

1、算法流程

- 输入加密后的文本s；
- 统计出现频率最大的字母ε（frqc_att()函数，调用了 n2l()函数遍历‘a’～‘z’）；
- 输出$(\varepsilon-\text{‘}e\text{’}) \bmod 26$。

函数调用关系图如下：



2、测试样例及结果截图

```
chnqyhnsmypyhnsmypyhnbyspinyxgswcnsnbyqilmnjfuwynifcpychugylcwuguchcmmoymmesbcablunyizpcify
 hwyuhxgilyjyijfyfcpychavyfiqnbyjipylnsfchynbuhuhsqbylyyfmywuhnxyhscnmnloyvonypylsvixsmncff
 quhnmnifcpybylynbcmwcnsufqusmainujligcmyzilsiogcabnvyufcyuhcffomcihvoncnmnbylydomnuliohxnb
 ywilhyluhxeyyjmsioaichacnmwcnsxlyugmuhxcguvcaxlyugyl
20

Process finished with exit code 0
```

3、讨论与思考

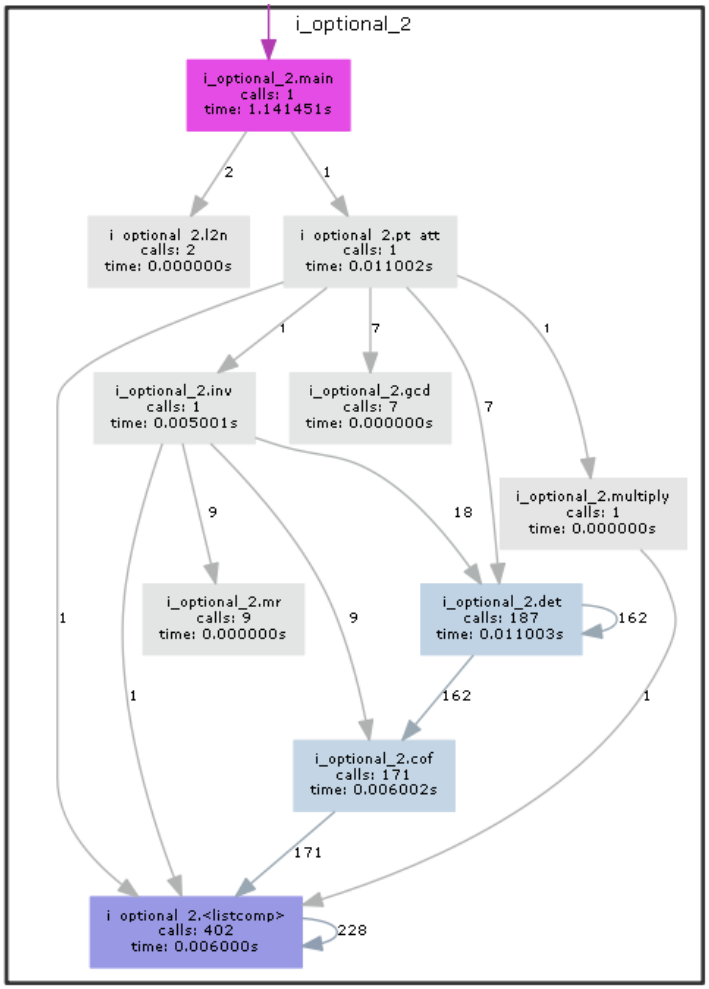
- 在 python 中，直接用`.count()`统计字符出现次数，用 `max()`寻找最大次数，用`.index()`记录最大次数的索引（即出现次数最多字母对应的 number）。

九、希尔密码の已知明文攻击

1、算法流程

- 输入密钥维数n，明文m，密文c，将m,c转化为 number（`l2n()`函数）；
- 将m,c以n个为一组分组，随机选取m的n个组排列为明文矩阵M（n阶方阵），判断M是否可逆（$\gcd(|M|,26)=1$），可逆则返回$K=M_{n\times n}^{-1}\cdot C_{n\times n}$；
- 格式化输出K。

函数调用关系图如下：



函数名	函数功能
`l2n()`	letter 转 number（0～25）
`pt_attack()`	明文攻击（plaintext attack）总函数
`inv()`	矩阵求逆（inverse）
`gcd()`	求最大公因数（greatest common divisor）
`mr()`	求模逆（mod reverse）
`det()`	矩阵行列式（递归）
`multiply()`	矩阵乘法（方阵×方阵）
`cof()`	求代数余子式（algebraic cofactor）

2、测试样例及结果截图

```
3
thisistestfortheknownplaintextattack
wvlomkzsbyjjwdisydygrqtfrpnfxnhthysu
7 2 15
25 25 6
0 2 19

Process finished with exit code 0
```

3、讨论与思考

- 考虑到明文矩阵是否可逆的不确定性，笔者采用 `random.sample()`对明文组进行随机选取并判断其是否可逆。
- 在**实验六**中提到的超时风险也如期而至了，但注意到每次测试都只有一个测试点超时，且每次超时的组别都不一样。笔者有理由相信这是`random.sample()`的随机性导致，经过多次提交，总会有一次~~运气好~~能够全部通过。
- 幸运的是第四次便过了（如图），且最大运行时间基本都在 990ms，因此该算法徘徊在超时边缘。~~**换句话说，该算法最大化了用时间代价换取的空间和算法简洁程度。**~~

8098	2022-03-21 20:34:51	Accepted	Python	995ms	11900KB	
8097	2022-03-21 20:34:31	Unaccepted	Python	996ms	11920KB	
8096	2022-03-21 20:34:06	Unaccepted	Python	999ms	11904KB	
8095	2022-03-21 20:33:42	Unaccepted	Python	941ms	11912KB	

十、单表代替密码の字母频率攻击

1、算法流程

- 读入文件 ciphertext.txt，并用字符串 ct 储存；
- 统计 ct 中各字母的出现频率；
- 对应自然语言中的字母频率得到置换表，将密文通过该表反置换得到伪明文；
- 通过伪明文中的一些伪单词更新置换表；
- 循环上一步操作直到得到真明文；
- 利用 `wordninja.split()`分词得到完整文章。

2、测试样例及结果截图

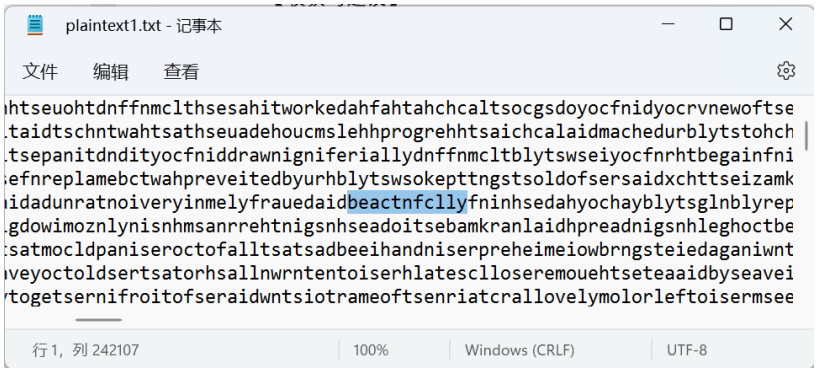
```
laksjdhfgpzoxicuvmqnwberyt

Process finished with exit code 0
```

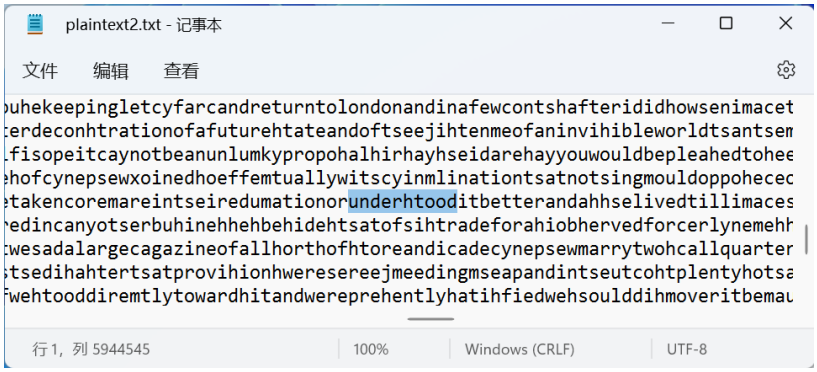
字母频率统计：

```
{'t': 0.068, 'v': 0.097, 'p': 0.133, 'r': 0.156, 'z': 0.789, 'b': 0.976, 'a': 1.469,
  'u': 1.714, 'y': 1.951, 'h': 2.019, 'd': 2.318, 'e': 2.358, 'k': 2.514, 'x': 2.53,
  'w': 2.819, 'o': 3.991, 's': 4.466, 'm': 5.961, 'q': 6.174, 'f': 6.302, 'g':
 6.747, 'i': 6.944, 'c': 7.594, 'l': 8.045, 'n': 9.271, 'j': 12.594}
```

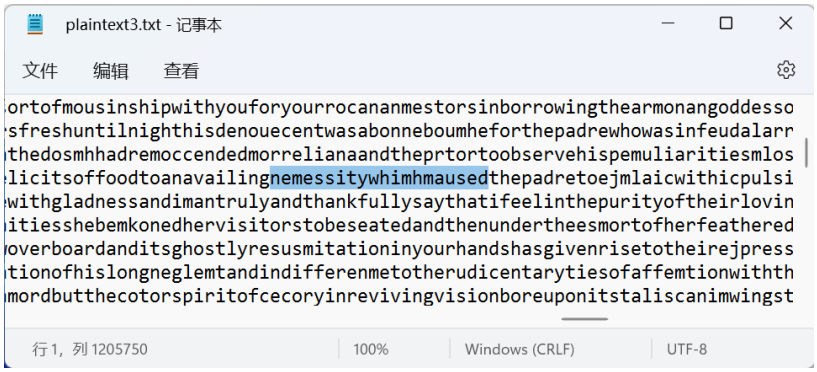
第一次置换后的伪明文：（$c \to u, n \to i$）



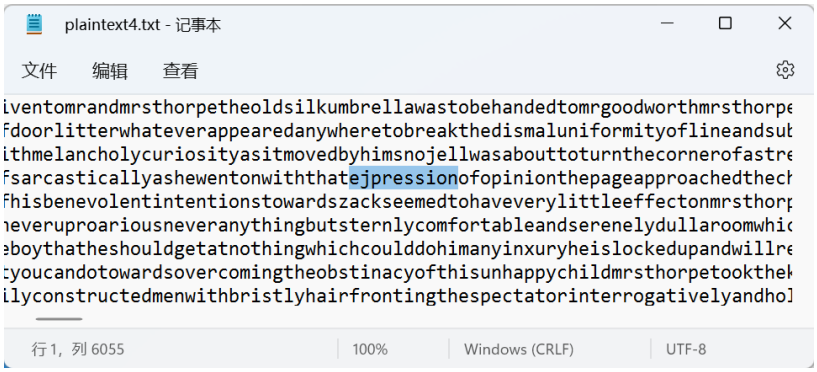
第二次置换后的伪明文：（$h \to s$）



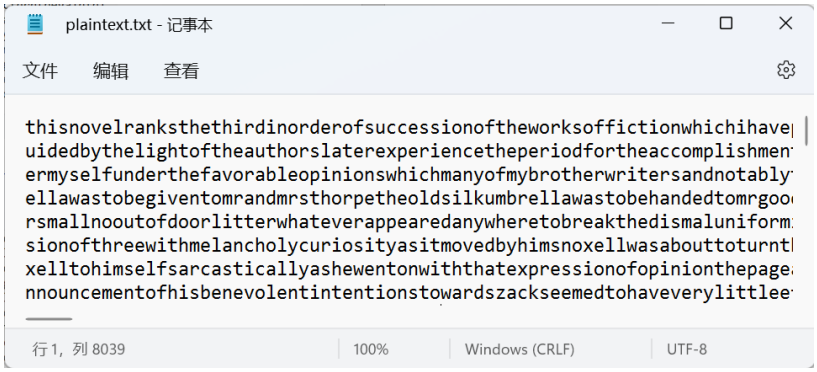
第三次置换后的伪明文：（$m \to c$）



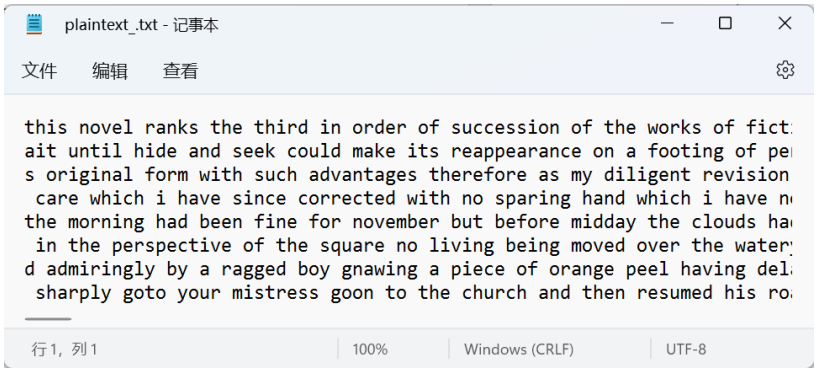
第四次置换后的伪明文：（$j \to x$）



真明文：



分词后的 plaintext：



3、讨论与思考

- 文章：*Hide and Seek* by Wilkie Collins。
- 原文链接：*http://www.online-literature.com/wilkie-collins/hide-and-seek/0/*

【收获与建议】

1、收获

- 加深了对古典密码的理解。
- 提高了编程能力和对 Python 的熟练度。
- 巩固了排版能力。

2、建议

- OJ 平台可以增加查询“错误输出”和“正确输入”的功能，以避免对不要问题（如本次的输入问题）的过多耗时，以及增强同学们的代码自纠错能力。
- 写伪代码和做流程图比较耗时，且两者均属于对算法原理的展示，略显重复。

【思考题】

1、 m 维 Hill 密码的破解效率如何？请大致用 m 的函数表示。

假设 26 个字母均匀分布，破解效率定义为 m 阶方阵中可逆矩阵（其行列式与 26 互素）的占比，这也可以看作是 m 维 Hill 密码密钥空间的大小。故共有 26^{m^2} 个 m 阶方阵，以 bit 长度表示，密钥空间大小为 $\log_2\left(26^{m^2}\right)=4.7m^2$。

m 阶方阵在 $\bmod\ 2$ 意义下可逆的个数为一般线性群 $\mathrm{GL}(m,\mathbb{Z}_2)$ 的阶数：

$$2^{m^2}\left(1-\frac{1}{2}\right)\left(1-\frac{1}{2^2}\right)\cdots\left(1-\frac{1}{2^m}\right)$$

m阶方阵在$\mathrm{mod}\ 13$意义下可逆的个数为一般线性群$\mathrm{GL}(m, \mathbb{Z}_{13})$的阶数：

$$13^{m^2}\left(1-\frac{1}{13}\right)\left(1-\frac{1}{13^2}\right)\cdots\left(1-\frac{1}{13^m}\right)$$

由中国剩余定理m阶方阵在$\mathrm{mod}\ 26$意义下可逆的个数为：

$$26^{m^2}\left(1-\frac{1}{2}\right)\left(1-\frac{1}{2^2}\right)\cdots\left(1-\frac{1}{2^m}\right)\left(1-\frac{1}{13}\right)\left(1-\frac{1}{13^2}\right)\cdots\left(1-\frac{1}{13^m}\right)$$

故 Hill 密码有效密钥空间大小约为$4.64n^2-1.7$。对于5维的 Hill 密码，大约是$114\,\mathrm{bit}$。

2、 请思考古典密码体制有什么缺陷与不足。

古典密码通常很容易被破解，容易受到仅密文攻击：

- 凯撒密码密钥空间小，类似的这些密码可以通过暴力攻击来破坏。
- 替换密码密钥空间较大，但容易受到频率分析的影响。
- 多字母密码（如 Vigenère 密码）通过使用多个替换来阻止简单的频率分析。但如 Kasiski 检查技术，仍可用来破解这些密码。