find_length.py

```
import string
from collections import Counter

# 密文
s =
'IYMYSILONRFNCQXQJEDSHBUIBCJUZBOLFQYSCHATPEQGQJEJNGNXZWHHGWFSUKULJQACZKKJOAAHGKEM
TAFGMKVRDOPXNEHEKZNKFSKIFRQVHHOVXINPHMRTJPYWQGJWPUUVKFPOAWPMRKKQZWLQDYAZDRMLPBJKJ
OBWIWPSEPVVQMBCRYVCRUZAAOUMBCHDAGDIEMSZFZHALIGKEMJJFPCIWKRMLMPINAYOFIREAOLDTHITDV
RMSE'

# 字母映射到它的前一个字母（恢复第一个切片后的字符串到维吉尼亚的加密形式）
letter_back = {letter: string.ascii_uppercase[idx - 1] for idx, letter in
enumerate(string.ascii_uppercase)}


# 计算重合指数ic
def get_ic(text):
    n = len(text)
    frequency = Counter(text)  # 统计每一个大写字母的频率
    up = sum(f * (f - 1) for f in frequency.values())  # 分子
    down = n * (n - 1)  # 分母
    ic = up / down
    return ic


# 把密文恢复成标准维吉尼亚的形式
def text_recover(text, l):
    # 转为列表，便于替换
    after = list(text)

    # 对从 l 到 len(text) 的字符进行不同次数的映射
    for i in range(l, len(text), l):
        # 映射的次数是 i // l（即从第 l 到 2l 映射一次，从 2l 到 3l 映射两次，以此类推）
        times = i // l
        # 将 i 到 i + l 的字符段逐个字符进行 times 次映射
        for j in range(i, min(i + l, len(text))):
            after[j] = times_back(after[j], times)

    # 将字符列表转换回字符串
    return ''.join(after)


# 多次映射函数
def times_back(char, times):
    # 根据字典映射 char 字符，并且根据 times 进行多次映射
    for _ in range(times):
        char = letter_back[char]
    return char


# 找最可能的密文长度，逐个尝试长度l
def length_find(text, max_len):
```

```
    ic_avg = []
    for l in range(1, max_len + 1):
        # 按当前长度恢复密文
        recover = text_recover(text, l)
        ics = 0
        for i in range(l):
            part_text = recover[i::l]  # 提取一个子序列
            ic = get_ic(part_text)  # 计算这个子序列的ic
            ics += ic  # 记录ic之和

        ic_avg.append(ics / l)  # 记录l个子串的平均ic值

    print(ic_avg)
    # 找到ic_avg列表中比0.065大最多的下标，加1，即为最可能的密钥长度
    length = 1 + max(enumerate(ic_avg), key=lambda x: (x[1] - 0.065))[0]
    return length


key_length = length_find(s, 10)
print("最可能的密钥长度：", key_length)  # 5

text_recovered = text_recover(s, key_length)
print("修复后的密文：\n" + text_recovered)
#IYMYSHKNMQDLAOVNGBAPDXQEXXEPUWIFZKSLVATMHWIYIAVAEXDNPMXWVLUHIYIZXDNPMXWVAMMSRVPX
DKPQWTEAMXXFVMPLRGURLYQOLWVAMMSZBMRSKPUWLRAYSHKXQVUVKFPNZVOLPIIOXTINAVWVZNIGKWEFD
IVQCPILXINNIETTIPMTHKPQQDJBQRVROURVRZFMRLTMXTRVPXTTPZMRFTAVTUXQVHFVMPXKGURIYMNYHZ
VQWH
```

find_key.py

```
from collections import Counter
import string

# 字母映射到数字
letter_to_num = {letter: idx for idx, letter in
enumerate(string.ascii_uppercase)}

# 数字映射到字母
num_to_letter = {idx: letter for idx, letter in
enumerate(string.ascii_uppercase)}

# 字母频率
letter_p = {'A': 0.082, 'B': 0.015, 'C': 0.028, 'D': 0.043, 'E': 0.127, 'F':
0.022, 'G': 0.020, 'H': 0.061,
            'I': 0.070, 'J': 0.002, 'K': 0.008, 'L': 0.040, 'M': 0.024, 'N':
0.067, 'O': 0.075, 'P': 0.019,
            'Q': 0.001, 'R': 0.060, 'S': 0.063, 'T': 0.091, 'U': 0.028, 'V':
0.010, 'W': 0.023, 'X': 0.001,
            'Y': 0.020, 'Z': 0.001}

# 已经变成普通维吉尼亚加密的密文
```

```
s =
'IYMYSHKNMQDLAOVNGBAPDXQEXXEPUWIFZKSLVATMHWIYIAVAEXDNPMXWVLUHIYIZXDNPMXWVAMMSRVPX
DKPQWTEAMXXFVMPLRGURLYQOLWVAMMSZBMRSKPUWLRAYSHKXQVUVKFPNZVOLPIIOXTINAVWVZNIGKWEFD
IVQCPILXINNIETTIPMTHKPQQDJBQRVROURVRZFMRLTMXTRVPXTTPZMRFTAVTUXQVHFVMPXKGURIYMNYHZ
VQWH'

# 密钥长度
l = 5

# 密钥字
key = []


# 计算当前密文字g对应的Mg，s为当前密文段
def get_Mg(g, s):
    Mg = 0
    frequency = Counter(s)  # 统计密文中每一个大写字母的频率
    # 累加26个英文单词的情况
    for i in range(26):
        now_letter = num_to_letter[i]  # 当前字母
        pi = letter_p[now_letter]
        n = len(s)
        f = frequency[num_to_letter[(i + g) % 26]]  # i+g加密后字母在密文中出现的频率
        Mg += pi * f / n
    return Mg


# 切分子字符串，每一个子字符串的加密字相同
for i in range(l):
    Mgs = []
    part_text = s[i::l]
    # 循环尝试26个密钥字
    for g in range(26):
        Mg = get_Mg(g, part_text)
        Mgs.append(Mg)
    print(Mgs)
    # 第i个子字符串的所有Mg计算完成，找到最接近0.065的
    num = max(enumerate(Mgs), key=lambda x: (x[1] - 0.065))[0]
    print("第", i, "个密钥字为：", num_to_letter[num])
    print("对应的Mg值为：", Mgs[num])
    key.append(num_to_letter[num])

keyword = ''.join(key)
print("找到最可能的密钥字为：", keyword)  # PRIME
```

declasssify.py

```
import string

# 字母映射到数字
letter_to_num = {letter: idx for idx, letter in
enumerate(string.ascii_uppercase)}

# 数字映射到字母
```

```python
num_to_letter = {idx: letter for idx, letter in
enumerate(string.ascii_uppercase)}

# 密文字
key = list("PRIME")
l = len(key)

key_num = [letter_to_num[element] for element in key]

# 已经变成普通维吉尼亚加密的密文
s =
'IYMYSHKNMQDLAOVNGBAPDXQEXXEPUWIFZKSLVATMHWIYIAVAEXDNPMXWVLUHIYIZXDNPMXWVAMMSRVPX
DKPQWTEAMXXFVMPLRGURLYQOLWVAMMSZBMRSKPUWLRAYSHKXQVUVKFPNZVOLPIIOXTINAVWVZNIGKWEFD
IVQCPILXINNIETTIPMTHKPQQDJBQRVROURVRZFMRLTMXTRVPXTTPZMRFTAVTUXQVHFVMPXKGURIYMNYHZ
VQWH'

answer = []

for i in range(0, len(s), l):
    for j in range(i, min(i + l, len(s))):
        text_num = (letter_to_num[s[j]] - key_num[j % l]) % 26
        text = num_to_letter[text_num]
        answer.append(text)

answer = ''.join(answer)
print(answer)
#THEMOSTFAMOUSCRYPTOLOGISTINHISTORYOWESHISFAMELESSTOWHATHEDIDTHANTOWHATHESAIDANDT
OTHESENSATIONALWAYINWHICHHESAIDITANDTHISWASMOSTPERFECTLYINCHARACTERFORHERBERTOSBO
RNEYARDLEYWASPERHAPSTHEMOSTENGAGINGARTICULATEANDTECHNICOLOREDPERSONALITYINTHEBUSI
NESS
# 整理成可读的：The most famous cryptologist in history owes his fame less to what
he did than to what he said, and to the sensational way in which he said it. And
this was most perfectly in character, for Herbert Osborne Yardley was perhaps the
most engaging, articulate, and technicolored personality in the business.
```