Homework Set 1

by Mingcen Wei [redacted] [redacted]

Answer 1.

Part A.

Let $A = \{s \in S \mid \Pr[X = s] > \Pr[Y = s]\}$, then

$$\begin{aligned}
\frac{1}{2}\sum_{s \in S} |\Pr[X = s] - \Pr[Y = s]| &= \frac{1}{2}\sum_{s \in A} (\Pr[X = s] - \Pr[Y = s]) + \\
&\quad \frac{1}{2}\sum_{s \in S - A} (\Pr[Y = s] - \Pr[X = s]) \\
&= \frac{1}{2}\sum_{s \in A} (\Pr[X = s] - \Pr[Y = s]) + \\
&\quad \frac{1}{2}\left(\left(1 - \sum_{s \in A} \Pr[Y = s]\right) - \left(1 - \sum_{s \in A} \Pr[Y = s]\right)\right) \\
&= \frac{1}{2}\sum_{s \in A} (\Pr[X = s] - \Pr[Y = s]) + \frac{1}{2}\sum_{s \in A} (\Pr[X = s] - \Pr[Y = s]) \\
&= \sum_{s \in A} (\Pr[X = s] - \Pr[Y = s]) \\
&= \max_{T \subseteq S} (\Pr[X \in T] - \Pr[Y \in T]).
\end{aligned}$$

Part B.

$$\begin{aligned}
\Pr[D(X) = 1] - \Pr[D(Y) = 1] &= \Pr[X \in D^{-1}(1)] - \Pr[Y \in D^{-1}(1)] \\
&\leqslant \max_{T \subseteq S} (\Pr[X \in T] - \Pr[Y \in T]) \\
&= \Delta(X, Y).
\end{aligned}$$

Part C.

$$\begin{aligned}
\Delta(X, Y) &= \max_{T \subseteq S} (\Pr[X \in T] - \Pr[Y \in T]) \\
&\geqslant \max_{\mathrm{domain}(D) \subseteq S} (\Pr[X \in D^{-1}(1)] - \Pr[Y \in D^{-1}(1)]) \\
&\geqslant \max_{D} (\Pr[D(X) = 1] - \Pr[D(Y) = 1]) \qquad (*) \\
&\geqslant \Pr[D(X) = 1] - \Pr[D(Y) = 1] \\
&\quad \text{where } D(s) = 1 \text{ iff } \Pr[X = s] > \Pr[Y = s], \forall s \in S, \text{and } 0 \text{ otherwise} \\
&= \sum_{s \in A} (\Pr[X = s] - \Pr[Y = s]) \text{ where } A = \{s \in S \mid \Pr[X = s] > \Pr[Y = s]\} \\
&= \Delta(X, Y) \text{ according to } \textbf{Part A.}
\end{aligned}$$

Answer 2.

This proposition is obviously true when $|\mathcal{K}| \geqslant |\mathcal{M}|$.

Now assume $|\mathcal{K}| < |\mathcal{M}|$. Given any $c \in \mathcal{C}$, there are at most $|\mathcal{K}|$ elements of $|\mathcal{M}|$ belonging to $\mathrm{Dec}^{-1}(\mathcal{K}, c) = \{m \in \mathcal{M} \mid \exists k \in \mathcal{K} \ s.t. \ m = \mathrm{Dec}(k, c)\}$. Thus $\forall c \in \mathcal{C}, \Pr[M \in \mathrm{Dec}^{-1}(\mathcal{K}, c)] \leqslant \frac{|\mathcal{K}|}{|\mathcal{M}|}$. $\therefore$ $\forall k_0 \in \mathcal{K}, m_0 \in \mathcal{M}, \Pr[M \in \mathrm{Dec}^{-1}(\mathcal{K}, \mathrm{Enc}(k_0, m_0))] \leqslant \frac{|\mathcal{K}|}{|\mathcal{M}|}$. $\therefore$ $\forall m_0 \in \mathcal{M}$,

$$\begin{aligned}
\Pr[M \in \mathrm{Dec}^{-1}(\mathcal{K}, \mathrm{Enc}(K, m_0))] &= \sum_{k_0 \in \mathcal{K}} \Pr[M \in \mathrm{Dec}^{-1}(\mathcal{K}, \mathrm{Enc}(K, m_0)), K = k_0] \\
&= \sum_{k_0 \in \mathcal{K}} \Pr[M \in \mathrm{Dec}^{-1}(\mathcal{K}, \mathrm{Enc}(k_0, m_0)), K = k_0] \\
&= \sum_{k_0 \in \mathcal{K}} \Pr[M \in \mathrm{Dec}^{-1}(\mathcal{K}, \mathrm{Enc}(k_0, m_0))] \Pr[K = k_0] \\
&\leqslant \sum_{k_0 \in \mathcal{K}} \frac{|\mathcal{K}|}{|\mathcal{M}|} \frac{1}{\mathcal{K}} \\
&= \frac{|\mathcal{K}|}{|\mathcal{M}|}.
\end{aligned}$$

$\therefore$ $\exists m_1 \in \mathcal{M} \ s.t. \ \Pr[m_1 \in \mathrm{Dec}^{-1}(\mathcal{K}, \mathrm{Enc}(K, m_0))] \leqslant \Pr[M \in \mathrm{Dec}^{-1}(\mathcal{K}, \mathrm{Enc}(K, m_0))] \leqslant \frac{|\mathcal{K}|}{|\mathcal{M}|}$. Since $\mathrm{Enc}(\mathcal{K}, m_1) = \{c \in \mathcal{C} \mid \exists k \in \mathcal{K} \ s.t. \ m_1 = \mathrm{Dec}(k, c)\} \subseteq \mathcal{C}$, we have

$$\begin{aligned}
\Delta(\mathrm{Enc}(K, m_0), \mathrm{Enc}(K, m_1)) &\geqslant \Pr[\mathrm{Enc}(K, m_1) \in \mathrm{Enc}(\mathcal{K}, m_1)] - \\
&\quad \Pr[\mathrm{Enc}(K, m_0) \in \mathrm{Enc}(\mathcal{K}, m_1)] \\
&= 1 - \Pr[m_1 \in \mathrm{Dec}^{-1}(\mathcal{K}, \mathrm{Enc}(K, m_0))] \\
&\geqslant 1 - \frac{|\mathcal{K}|}{|\mathcal{M}|}.
\end{aligned}$$

Answer 3.

Plaintext:

cryptographyisanindispensabletoolusedtoprotectinformationincomputingsystemsitisusedevery
whereandbybillionsofpeopleworldwideonadailybasisitisusedtoprotectdataatrestanddatainmotion

Cryptography is an indispensable tool used to protect information in computing systems. It is used everywhere and by billions of people worldwide on a daily basis. It is used to protect data at rest and data in motion.

— Oh! It seems that this is an excerpt from *A Quantum Leap in Cryptography: Interview with Grégoire Ribordy From ID Quantique*.

Cipher:

`enjoy`

Steps:

1. First I tried the *index of coincidence method* — computing sums of squared frequencies of characters in the spaced substrings with various interval lengths (see A.1). However, due to the short length of the ciphertext, this method didn't help much.

2. Then I tried KASISKI*'s method*, and found that the substring `olh` appeared twice in the ciphertext (see A.2).

3. I guessed that `the` might be mapped to `olh`. However, when I mapped the ciphertext back, it seemed nonsense (see A.3).

4. Then I computed the distance between the two appearances and the result was `70` $= 2 \times 5 \times 7$ (see A.4). Thus the cipher length was likely one in 5, 7, 10, 14, 35, 70 (1 and 2 are omitted since `olh` has 3 characters).

5. To ensure that I could find the cipher, I decided to use the *brute force method* to try all ciphers no longer than 4 characters (see A.6). The time complexity $T_1 = \sum_{1 \leqslant k \leqslant 4} 26^k = 475254$, and running the program took me several minutes (thus if I chose to try all 5 letter ciphers, this program would take me more than an hour).

6. To find meaningful strings from the output file obtained in 5, I used the Python library *Nostril: Nonsense String Evaluator*[1]. It gave an empty output file (see A.7). In view of the low false positive rate of the library, I deduced that the cipher length was longer than 5 characters.

7. Then I started to try 5 letter ciphers. I calculated the frequencies of each letter in the spaced substrings (see A.8). Again, this information was of little use because of the limited size of the ciphertext.

8. The letter `e` appears very frequently throughout English language. Therefore I believed it would appear in every spaced substring. I used brute force to try every possiblity (see A.5). The spaced substrings contained 14, 16, 13, 14, 16 distinct letters, repectively (see the output of A.8), so the time complexity $T_2 = 14 \times 16 \times 13 \times 14 \times 16 = 652288 \ll 11881376 = 26^5$. This time the program took me several minutes as well.

9. Once more, I used *Nostril* to find meaningful string from the output file obtained in 8. The result was a 41 line file (see A.7). Hooray! This time I finally found a meaningful string in line 19 of the file.

10. In the end, I used the plaintext and the ciphertext to compute the cipher (see A.9). It was `enjoy`, and I enjoyed this journey of cracking code very much!

Answer 4.

Definition. Fix an integer $\ell > 0$, $\mathcal{M} = \mathcal{C} = \{a, b, ..., z\}^{\leqslant \ell}$, and $\mathcal{K} = S_\ell$ (the permutation group of order ℓ). Let $f: \{a, b, ..., z\} \to \{0, 1, ..., 25\}$ be the bijective ordinal mapping.

- Gen: choosing a key from $\mathcal{K}$ uniformly. $k \leftarrow \mathsf{Gen}(1^n)$.
- Enc: given a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, $c := \mathsf{Enc}(k, m) = \mathrm{map}(f^{-1}, \mathrm{map}(k, \mathrm{map}(f, m)))$, i.e. $\mathsf{Enc}(k, \overline{\alpha_1\alpha_2...\alpha_j}) = \overline{f^{-1}(k(f(\alpha_1)))f^{-1}(k(f(\alpha_2)))...f^{-1}(k(f(\alpha_j)))}$.
- Dec: the same as Enc, except that it use k^{-1} instead of k.

Answer 5.

a) No. For instance, $\Pr[M = 0 \mid C = 0] = \frac{1}{3}$, while $\Pr[M = 1 \mid C = 0] = \Pr[M = 2 \mid C = 0] = \Pr[M = 3 \mid C = 0] = \Pr[M = 4 \mid C = 0] = \frac{1}{6}$.

[1] Its documentation says:

> Nostril is the *Nonsense String Evaluator*: a Python module that infers whether a given short string of characters is likely to be random gibberish or something meaningful.
>
> ...
>
> Nostril uses a combination of heuristic rules and a probabilistic assessment. It is not always correct (see below). It is tuned to reduce false positives: it is more likely to say something is *not* gibberish when it really might be. This is suitable for its intended purpose of filtering source code identifiers – a difficult problem, incidentally, because program identifiers often consist of acronyms and word fragments jammed together (e.g., "kBoPoMoFoOrderIdCID", "ioFlXFndrInfo", etc.), which can challenge even humans. Nevertheless, on the identifier strings from the Loyola University of Delaware Identifier Splitting Oracle, Nostril classifies over 99% correctly.
>
> Nostril is reasonably fast: once the module is loaded, on a 4 Ghz Apple OS X 10.12 computer, calling the evaluation function returns a result in 30–50 microseconds per string on average.

b) Yes. Because for each $m\in\mathcal{M}, c\in\mathcal{C}$, $\Pr[M=m\mid C=c]\equiv\frac{1}{|\mathcal{M}|}$.

Answer 6.

Example. Given $\varepsilon>0$, there exists an integer $n>\frac{1}{\varepsilon}+1$. Let $\mathcal{M}=\mathcal{C}=\{0,1,2,\ldots,n\},\mathcal{K}=\{0,1,2,\ldots,(n-1)\},|\mathcal{K}|<|\mathcal{M}|$. Let $\mathrm{Enc}(m,k)=[(m+k) \bmod (n+1)]$, $\mathrm{Dec}(c,k)=[(c-k) \bmod (n+1)]$. Then given any distinct $m_0,m_1\in\mathcal{M}$, if $k\in\mathcal{K}-\{[(m_0+n) \bmod (n+1)],[(m_1+n) \bmod (n+1)]\}$, $\mathcal{A}$ has a chance $\frac{1}{2}$ of success; otherwise if $k=[(m_0+n) \bmod (n+1)]$, $\mathcal{A}$ can deduce that $M=m_1$, and if $k=[(m_1+n) \bmod (n+1)]$, $\mathcal{A}$ can deduce that $M=m_0$. Thus

$$\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}{=}1]=\frac{n-3}{n-1}\times\frac{1}{2}+\frac{2}{n-1}\times 1=\frac{1}{2}+\frac{1}{n-1}\leqslant\frac{1}{2}+\varepsilon.$$

Proposition. (Lower Bound of the Cardinality of the Key Space)

If $\Pr[\mathit{PrivK}^{eav}_{\mathcal{A},\Pi}{=}1]\leqslant\frac{1}{2}+\varepsilon$, *then* $|\mathcal{K}|\geqslant|\mathcal{M}|\,(1-\varepsilon)$.

Proof. According to **Answer 2**, if $\forall m_0,m_1\in\mathcal{M},\ \Delta(\mathrm{Enc}(K,m_0),\mathrm{Enc}(K,m_1))\leqslant\varepsilon$, then $1-\frac{|\mathcal{K}|}{|\mathcal{M}|}\leqslant\varepsilon$, i.e. $|\mathcal{K}|\geqslant|\mathcal{M}|\,(1-\varepsilon)$. Therefore we only need to prove that $\forall m_0,m_1\in\mathcal{M},\ \Delta(\mathrm{Enc}(K,m_0),\mathrm{Enc}(K,m_1))\leqslant\varepsilon$.

$\forall m_0,m_1\in\mathcal{M}$, let $S=\{c\in\mathcal{C}\mid\Pr[\mathrm{Enc}(K,m_0)]>\Pr[\mathrm{Enc}(K,m_1)]\}$. $\mathcal{A}$ has a chance no greater than $\left(\frac{1}{2}+\varepsilon\right)$ to sucessfully distinguish $\mathrm{Enc}(K,m_0)$ and $\mathrm{Enc}(K,m_1)$. If whenever $c\in S$, $\mathcal{A}$ will output m_0, and m_1 otherwise, then $\mathcal{A}$ can perform best on average. Thus

$$\begin{aligned}
\frac{1}{2}+\varepsilon &\geqslant \sum_{c\in S}\Pr[\mathrm{Enc}(K,m_0)=c]\Pr[m_0\text{ is chosen}]+\\
&\quad\sum_{c\in\mathcal{C}-S}\Pr[\mathrm{Enc}(K,m_1)=c]\Pr[m_1\text{ is chosen}]\\
&= \frac{1}{2}\sum_{c\in S}\Pr[\mathrm{Enc}(K,m_0)=c]+\frac{1}{2}\left(1-\sum_{c\in\mathcal{C}}\Pr[\mathrm{Enc}(K,m_1)=c]\right)\\
&= \frac{1}{2}+\sum_{c\in S}(\Pr[\mathrm{Enc}(K,m_0)=c]-\Pr[\mathrm{Enc}(K,m_1)=c])\\
&= \frac{1}{2}+(\Pr[\mathrm{Enc}(K,m_0)\in S]-\Pr[\mathrm{Enc}(K,m_1)\in S])\\
&= \frac{1}{2}+\max_{T\subseteq\mathcal{C}}(\Pr[\mathrm{Enc}(K,m_0)\in T]-\Pr[\mathrm{Enc}(K,m_0)\in T])\\
&= \frac{1}{2}+\Delta(\mathrm{Enc}(K,m_0),\mathrm{Enc}(K,m_1))
\end{aligned}$$

Then we have $\Delta(\mathrm{Enc}(K,m_0),\mathrm{Enc}(K,m_1))\leqslant\varepsilon$. □

Appendix

A Python Programs and Outputs

For the source code and some of the results files, visit https://github.com/mingcenwei/pku course fundamentals of cryptography.

A.1 sums_of_squared_frequencies.py

Code

```
from itertools import takewhile, count
from string import ascii_lowercase

CIPHERTEXT = "GEHDRSTAONLLRGYRVWRGWCNBQEOUSRSBUIQIQCCNVBCSA" + \
    "XVWTMVZJHGSARBASZYIRMAPGWWGNAQMGRGSWRMSTIEH" + \
    "KFIENOLHOHPGPYRCLWBODCSCUSUSEURUMQNCLEQJWJCOJ" + \
    "GGWVCWQYFNRRSCACRIPCRYXNJHPIFCOLHQJHYMAVCRMBW"
MAX_CIPHER_LENGTH = 10

CIPHERTEXT = CIPHERTEXT.lower()

for cipherLength in range(1, MAX_CIPHER_LENGTH + 1):
    substrings = [''.join(
        CIPHERTEXT[start_i + j * cipherLength] for j in takewhile(lambda j:
start_i + j * cipherLength < len(CIPHERTEXT), count())
        ) for start_i in range(cipherLength)]
    sumsOfSquaredFrequencies = [round(sum(
        substring.count(letter) ** 2 for letter in ascii_lowercase
        ) / (len(substring) ** 2), 3) for substring in substrings]
    print(sumsOfSquaredFrequencies)
```

Output

```
[0.05]

[0.054, 0.056]

[0.069, 0.058, 0.057]

[0.054, 0.07, 0.063, 0.06]

[0.113, 0.079, 0.103, 0.091, 0.092]

[0.084, 0.087, 0.069, 0.078, 0.082, 0.07]

[0.074, 0.086, 0.083, 0.085, 0.12, 0.085, 0.075]

[0.07, 0.081, 0.083, 0.087, 0.07, 0.091, 0.087, 0.079]

[0.115, 0.095, 0.105, 0.105, 0.095, 0.085, 0.08, 0.069, 0.091]

[0.123, 0.086, 0.154, 0.117, 0.16, 0.142, 0.117, 0.123, 0.093, 0.08]
```

A.2 find_repeated_substrings.py

Code

```
CIPHERTEXT = "GEHDRSTAONLLRGYRVWRGWCNBQEOUSRSBUIQIQCCNVBCSA" + \
    "XVWTMVZJHGSARBASZYIRMAPGWWGNAQMGRGSWRMSTIEH" + \
    "KFIENOLHOHPGPYRCLWBODCSCUSUSEURUMQNCLEQJWJCOJ" + \
    "GGWVCWQYFNRRSCACRIPCRYXNJHPIFCOLHQJHYMAVCRMBW"
TARGET_WORD = "the"

CIPHERTEXT = CIPHERTEXT.lower()
TAEGET_WORD = TARGET_WORD.lower()

stringFrequencies = dict()

for i1 in range(len(CIPHERTEXT)   len(TAEGET_WORD) + 1):
    word = CIPHERTEXT[i1: i1 + len(TAEGET_WORD)]
    if word in stringFrequencies.keys():
        continue
    else:
        count = 1
        for i2 in range(i1 + 1, len(CIPHERTEXT)   len(TAEGET_WORD) + 1):
            if word == CIPHERTEXT[i2: i2 + len(TAEGET_WORD)]:
                count += 1
            else:
                continue
        if count == 1:
            continue
        else:
            stringFrequencies[word] = count

sortedList = sorted(stringFrequencies.items(), key=(lambda kv: kv[1]),
reverse=True)
print(sortedList)
```

Output

```
[('olh', 2)]
```

A.3 known_substrings_plaintext_attack.py

Code

```
CIPHERTEXT = "GEHDRSTAONLLRGYRVWRGWCNBQEOUSRSBUIQIQCCNVBCSA" + \
    "XVWTMVZJHGSARBASZYIRMAPGWWGNAQMGRGSWRMSTIEH" + \
    "KFIENOLHOHPGPYRCLWBODCSCUSUSEURUMQNCLEQJWJCOJ" + \
    "GGWVCWQYFNRRSCACRIPCRYXNJHPIFCOLHQJHYMAVCRMBW"
TARGET_WORD = "the"
ENCRYPTED_STRING = "olh"
CIPHER_LENGTH = 5

CIPHERTEXT = CIPHERTEXT.lower()
TAEGET_WORD = TARGET_WORD.lower()
ENCRYPTED_STRING = ENCRYPTED_STRING.lower()

forwardShifts = [(ord(ENCRYPTED_STRING[i])   ord(TAEGET_WORD[i])) % 26 for i
in range(len(TAEGET_WORD))]
encyptedStringIndex = CIPHERTEXT.find(ENCRYPTED_STRING)
partiallyDecryptedString = ""
for i in range(len(CIPHERTEXT)):
    offset = (i   encyptedStringIndex) % CIPHER_LENGTH
    if offset < len(TARGET_WORD):
        o1 = ord(CIPHERTEXT[i])   forwardShifts[offset]
        decryptedChar = chr(o1) if o1 >= ord("a") else chr(o1 + 26)
        partiallyDecryptedString += decryptedChar
    else:
        partiallyDecryptedString += chr(ord(CIPHERTEXT[i]) + ord("A")
ord("a"))
print(partiallyDecryptedString)
```

Output

```
dEHinpTAtjiLRluoVWwctCNgmbOUxnpBUnmfQChjsBCxwuVWyisZJmcpARgwpZYnnjAPlst

GNfmjGRlotRMxpfEHpbfENtheOHucmYRhhtBOiypCUxqpEUwqjQNhhbQJbfzOJlctVCbmvFN

wnpCAhnfPCwuuNJmlfFCtheQJmujAVhnjBW
```

A.4 get_gaps_between_repeated_substrings.py

Code

```
CIPHERTEXT = "GEHDRSTAONLLRGYRVWRGWCNBQEOUSRSBUIQIQCCNVBCSA" + \
    "XVWTMVZJHGSARBASZYIRMAPGWWGNAQMGRGSWRMSTIEH" + \
    "KFIENOLHOHPGPYRCLWBODCSCUSUSEURUMQNCLEQJWJCOJ" + \
    "GGWVCWQYFNRRSCACRIPCRYXNJHPIFCOLHQJHYMAVCRMBW"
POSSIBLE_ENCRYPTED_STRINGS = ("olh", )

CIPHERTEXT = CIPHERTEXT.lower()
POSSIBLE_ENCRYPTED_STRINGS = [string.lower() for string in
POSSIBLE_ENCRYPTED_STRINGS]

for string in POSSIBLE_ENCRYPTED_STRINGS:
    possible_encrypted_string_indices = list()
    start_i = CIPHERTEXT.find(string)
    while start_i !=  1:
        possible_encrypted_string_indices.append(start_i)
        start_i = CIPHERTEXT.find(string, start_i + 1)

    for i in range(len(possible_encrypted_string_indices)   1):
        print(possible_encrypted_string_indices[i + 1]
possible_encrypted_string_indices[i], end=" ")
    print()
```

Output

```
70
```

A.5 shift_text_backward.py

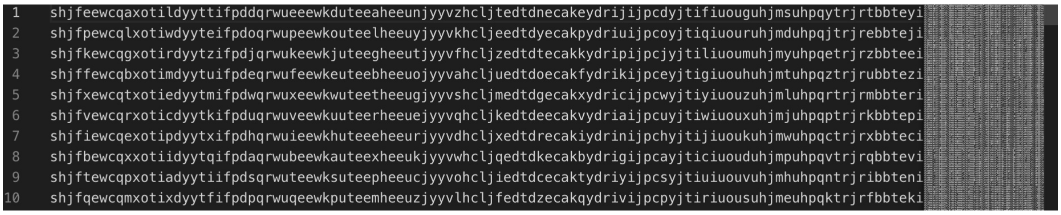
Code

```
from itertools import product

def shiftBack(ciphertext, step, plain_char, cipher_char, start_i):
    shift = (ord(cipher_char.lower())   ord(plain_char.lower())) % 26

    decryptedText = ""
    for i in range(len(ciphertext)):
        if (i   start_i) % step == 0:
            o1 = ord(ciphertext[i].lower())   shift
            char = chr(o1) if o1 >= ord("a") else chr(o1 + 26)
            decryptedText += char
        else:
            decryptedText += ciphertext[i]

    return decryptedText

if __name__ == "__main__":
    FILENAME = "brute_force_results.txt"
    CIPHERTEXT = "GEHDRSTAONLLRGYRVWRGWCNBQEOUSRSBUIQIQCCNVBCSA" + \
    "XVWTMVZJHGSARBASZYIRMAPGWWGNAQMGRGSWRMSTIEH" + \
    "KFIENOLHOHPGPYRCLWBODCSCUSUSEURUMQNCLEQJWJCOJ" + \
    "GGWVCWQYFNRRSCACRIPCRYXNJHPIFCOLHQJHYMAVCRMBW"
    CIPHER_LENGTH = 5
    # FIRST_N_TO_TRY = 4
    FIRST_N_TO_TRY = 2
    # MOST_COMMON_LETTERS = ("e", "t", "a", "i")
    MOST_COMMON_LETTERS = ("e", )
    FREQUENCY_LISTS = [
        [(’s’, 0.194), (’i’, 0.139), (’m’, 0.139), (’w’, 0.139), (’e’,
0.056), (’h’, 0.056), (’v’, 0.056), (’x’, 0.056), (’c’, 0.028), (’g’,
0.028), (’l’, 0.028), (’p’, 0.028), (’r’, 0.028), (’y’, 0.028)],
        [(’b’, 0.111), (’e’, 0.111), (’q’, 0.111), (’a’, 0.083), (’c’,
0.083), (’o’, 0.083), (’v’, 0.083), (’f’, 0.056), (’g’, 0.056), (’z’,
0.056), (’l’, 0.028), (’n’, 0.028), (’p’, 0.028), (’r’, 0.028), (’t’,
0.028), (’y’, 0.028)],
        [(’c’, 0.139), (’j’, 0.139), (’n’, 0.139), (’r’, 0.111), (’u’,
0.111), (’h’, 0.083), (’w’, 0.083), (’a’, 0.056), (’m’, 0.028), (’o’,
0.028), (’p’, 0.028), (’v’, 0.028), (’y’, 0.028)],
        [(’c’, 0.143), (’g’, 0.114), (’r’, 0.114), (’s’, 0.114), (’h’,
0.086), (’o’, 0.086), (’b’, 0.057), (’d’, 0.057), (’i’, 0.057), (’w’,
0.057), (’a’, 0.029), (’k’, 0.029), (’p’, 0.029), (’t’, 0.029)],
        [(’r’, 0.171), (’g’, 0.114), (’l’, 0.114), (’q’, 0.114), (’y’,
0.086), (’a’, 0.057), (’n’, 0.057), (’u’, 0.057), (’c’, 0.029), (’f’,
0.029), (’j’, 0.029), (’m’, 0.029), (’p’, 0.029), (’s’, 0.029), (’t’,
0.029), (’w’, 0.029)]]

    CIPHERTEXT = CIPHERTEXT.upper()

    with open(FILENAME, "w") as file:
        # for indexTuple in product(range(FIRST_N_TO_TRY),
repeat=CIPHER_LENGTH):
        ranges = [range(len(l)) for l in FREQUENCY_LISTS]
        for indexTuple in product(*ranges):
            for indexTuple2 in product(MOST_COMMON_LETTERS,
repeat=CIPHER_LENGTH):
                decryptedText = CIPHERTEXT
                for i in range(CIPHER_LENGTH):
                    decryptedText = shiftBack(decryptedText, CIPHER_LENGTH,
indexTuple2[i], FREQUENCY_LISTS[i][indexTuple[i]][0], i)

                print(decryptedText, file=file)
```

Output

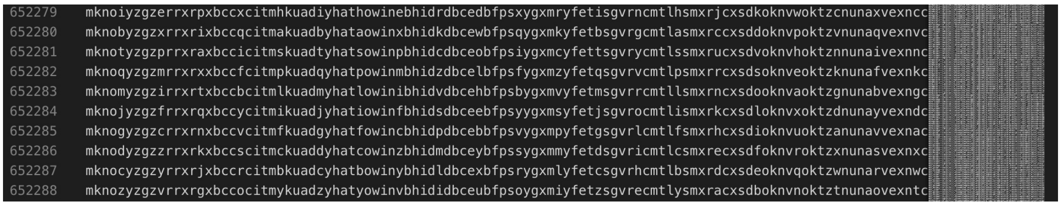
A 652288 line large file (size: 116,759,552 bytes).

The first 10 lines:



```
shjfeewcqaxotildyyttifpddqrwueeewkduteeaheeunjyyvzhcljtedtdnecakeydrijijpcdyjtifiuouguhjmsuhpqytrjrtbbteyi
shjfpewcqlxotiwdyyteifpdoqrwupeewkouteelheeuyjyyvkhcljeedtdyecakpydriuijpcoyjtiqiuouruhjmduhpqjtrjrebbteji
shjfkewcqgxotirdyytzifpdjqrwukeewkjuteegheeutjyyvfhcljzedtdtecakkydripijpcjyjtiliuoumuhjmyuhpqetrjrzbbteei
shjffewcqbxotimdyytuifpdeqrwufeewkeuteebheeuojyyvahcljuedtdoecakfydrikijpceyjtigiuouhuhjmtuhpqztrjrubbtezi
shjfxewcqtxotiedyytmifpdwqrwuxeewkwuteetheeugjyyvshcljmedtdgecakxydricijpcwyjtiyiuouzuhjmluhpqrtrjrmbbteri
shjfvewcqrxoticdyytkifpduqrwuveewkuuteerheeuejyyvqhcljkedtdeecakvydriaijpcuyjtiwiuouxuhjmjuhpqptrjrkbbtepi
shjfiewcqexotipdyytxifpdhqrwuieewkhuteeeheeurjyyvdhcljxedtdrecakiydrinijpchyjtijiuoukuhjmwuhpqctrjrxbbteci
shjfbewcqxxotiidyytqifpdaqrwubeewkauteexheeukjyyvwhcljqedtdkecakbydrigijpcayjticiuouduhjmpuhpqvtrjrqbbtevi
shjftewcqpxotiadyytiifpdsqrwuteewksuteepheeucjyyvohcljiedtdcecaktydriyijpcsyjtiuiuouvuhjmhuhpqntrjribbteni
shjfqewcqmxotixdyytfifpdpqrwuqeewkputeemheeuzjyyvlhcljfedtdzecakqydrivijpcpyjtiriuousuhjmeuhpqktrjrfbbteki
```

The last 10 lines:



```
mknoiyzgzerrxrpxbccxcitmhkuadiyhathowinebhidrdbcedbfpsxygxmryfetisgvrncmtlhsmxrjcxsdkoknvwoktzcnunaxvexncc
mknobyzgzxrrxrixbccqcitmakuadbyhataowinxbhidkdbcewbfpsqygxmkyfetbsgvrgcmtlasmxrccxsddoknvpoktzvnunaqvexnvc
mknotyzgzprrxraxbccicitmskuadtyhatsowinpbhidcdbceobfpsiygxmcyfettsgvrycmtlssmxrucxsdvoknvhoktznnunaivexnnc
mknoqyzgzmrrxrxxbccfcitmpkuadqyhatpowinmbhidzdbcelbfpsfygxmzyfetqsgvrvcmtlpsmxrrcxsdsoknveoktzknunafvexnkc
mknomyzgzirrxrtxbccbcitmlkuadmyhatlowinibhidvdbcehbfpsbygxmvyfetmsgvrrcmtllsmxrncxsdooknvaoktzgnunabvexngc
mknojyzgzfrrxrqxbccycitmikuadjyhatiowinfbhidsdbceebfpsyygxmsyfetjsgvrocmtlismxrkcxsdloknvxoktzdnunayvexndc
mknogyzgzcrrxrnxbccvcitmfkuadgyhatfowincbhidpdbcebbfpsvygxmpyfetgsgvrlcmtlfsmxrhcxsdioknvuoktzanunavvexnac
mknodyzgzzrrxrkxbccscitmckuaddyhatcowinzbhidmdbceybfpssygxmmyfetdsgvricmtlcsmxrecxsdfoknvroktzxnunasvexnxc
mknocyzgzyrrxrjxbccrcitmbkuadcyhatbowinybhidldbcexbfpsrygxmlyfetcsgvrhcmtlbsmxrdcxsdeoknvqoktzwnunarvexnwc
mknozyzgzvrrxrgxbccocitmykuadzyhatyowinvbhididbceubfpsoygxmiyfetzsgvrecmtlysmxracxsdboknvnoktztnunaovexntc
```

A.6 brute_force.py

Code

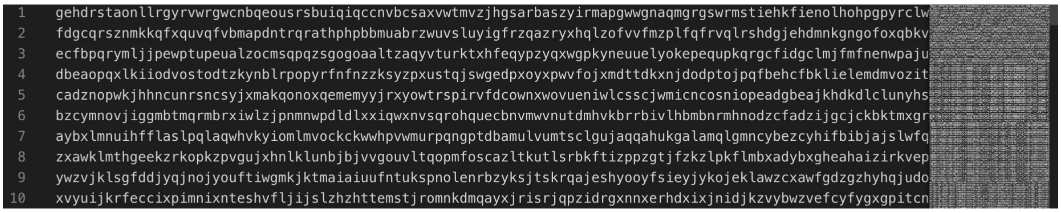
For `shift_text_backward.py`, see A.5.

```
from shift_text_backward import shiftBack
from string import ascii_lowercase
from itertools import product

FILENAME = "brute_force_results.txt"
CIPHERTEXT = "GEHDRSTAONLLRGYRVWRGWCNBQEOUSRSBUIQIQCCNVBCSA" + \
    "XVWTMVZJHGSARBASZYIRMAPGWWGNAQMGRGSWRMSTIEH" + \
    "KFIENOLHOHPGPYRCLWBODCSCUSUSEURUMQNCLEQJWJCOJ" + \
    "GGWVCWQYFNRRSCACRIPCRYXNJHPIFCOLHQJHYMAVCRMBW"
CIPHER_LENGTHS = list(range(1, 5))
ALPHABET = ascii_lowercase

CIPHERTEXT = CIPHERTEXT.upper()

with open(FILENAME, "w") as file:
    for cipher_length in CIPHER_LENGTHS:
        for shiftTuple in product(range(len(ALPHABET)),
repeat=cipher_length):
            decryptedText = CIPHERTEXT
            for i in range(cipher_length):
                decryptedText = shiftBack(decryptedText, cipher_length,
ALPHABET[0], ALPHABET[shiftTuple[i]], i)

            print(decryptedText, file=file)
```

Output

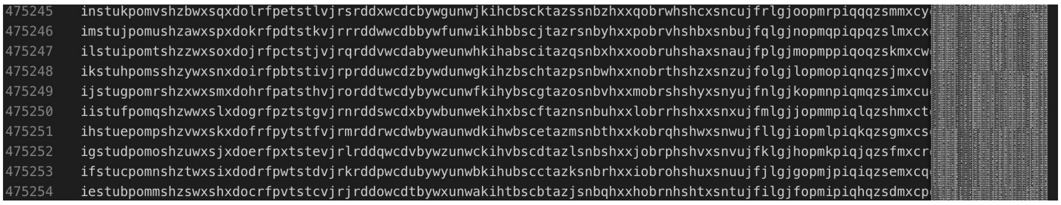
A 475254 line large file (size: 85,070,466 bytes).

The first 10 lines:



```
gehdrstaonllrgyrvwrgwcnbqeousrsbuiqiqccnvbcsaxvwtmvzjhgsarbaszyirmapgwwgnaqmgrgswrmstiehkfienolhohpgpyrclw
fdgcqrsznmkkqfxquvqfvbmapdntrqrathphpbbmuabrzwuvsluyigfrzqazryxhqlzofvvfmzplfqfrvqlrshdgjehdmnkgngofoxqbkv
ecfbpqrymljjpewptupeualzocmsqpqzsgogoaaltzaqyvturktxhfeqypzyqxwgpkyneuuelyokepequpkqrgcfidgclmjfmfnenwpaju
dbeaopqxlkiiodvostodtzkynblrpopyrfnfnzzksyzpxustqjswgedpxoyxpwvfojxmdttdkxnjdodptojpqfbehcfbklielemdmvozit
cadznopwkjhhncunrsncsyjxmakqonoxqememyyjrxyowtrspirvfdcownxwovueniwlcsscjwmicncosniopeadgbeajkhdkdlclunyhs
bzcymnovjiggmbtmqrmbrxiwlzjpnmnwpdldlxxiqwxnvsqrohquecbnvmwvnutdmhvkbrrbivlhbmbnrmhnodzcfadzijgcjckbktmxgr
aybxlmnuihfflaslpqlaqwhvkyiomlmvockckwwhpvwmurpqngptdbamulvumtsclgujaqqahukgalamqlgmncybezcyhifbibjajslwfq
zxawklmthgeekzrkopkzpvgujxhnlklunbjbjvvgouvltqopmfoscazltkutlsrbkftizppzgtjfzkzlpkflmbxadybxgheahaizirkvep
ywzvjklsgfddjyqjnojyouftiwgmkjktmaiaiuufntukspnolenrbzyksjtskrqajeshyooyfsieyjykojeklawzcxawfgdzgzhyhqjudo
xvyuijkrfeccixpimnixnteshvfljijslzhzhttemstjromnkdmqayxjrisrjqpzidrgxnnxerhdxixjnidjkzvybwzvefcyfygxgpitcn
```

The last 10 lines:



```
instukpomvshzbwxsqxdolrfpetstlvjrsrddxwcdcbywgunwjkihcbscktazssnbzhxxqobrwhshcxsncujfrlgjoopmrpiqqqzsmmxcy
imstujpomushzawxspxdokrfpdtstkvjrrrddwwcdbbywfunwikihbbscjtazrsnbyhxxpobrvhshbxsnbujfqlgjnopmqpiqpqzslmxcx
ilstuipomtshzzwxsoxdojrfpctstjvjrqrddvwcdabyweunwhkihabscitazqsnbxhxxoobruhshaxsnaujfplgjmopmppiqoqzskmxcw
ikstuhpomsshzywxsnxdoirfpbtstivjrprdduwcdzbywdunwgkihzbschtazpsnbwhxxnobrthshzxsnzujfolgjlopmopiqnqzsjmxcv
ijstugpomrshzxwxsmxdohrfpatsthvjrorddtwcdybywcunwfkihybscgtazosnbvhxxmobrshshyxsnyujfnlgjkopmnpiqmqzsimxcu
iistufpomqshzwwxslxdogrfpztstgvjrnrddswcdxbywbunwekihxbscftaznsnbuhxxlobrrhshxxsnxujfmlgjjopmmpiqlqzshmxct
ihstuepompshzvwxskxdofrfpytstfvjrmrddrwcdwbywaunwdkihwbscetazmsnbthxxkobrqhshwxsnwujfllgjiopmlpiqkqzsgmxcs
igstudpomoshzuwxsjxdoerfpxtstevjrlrddqwcdvbywzunwckihvbscdtazlsnbshxxjobrphshvxsnvujfklgjhopmkpiqjqzsfmxcr
ifstucpomnshztwxsixdodrfpwtstdvjrkrddpwcdubywyunwbkihubscctazksnbrhxxiobrohshuxsnuujfjlgjgopmjpiqiqzsemxcq
iestubpommshzswxshxdocrfpvtstcvjrjrddowcdtbywxunwakihtbscbtazjsnbqhxxhobrnhshtxsntujfilgjfopmipiqhqzsdmxcp
```

A.7 find_meaningful_strings.py

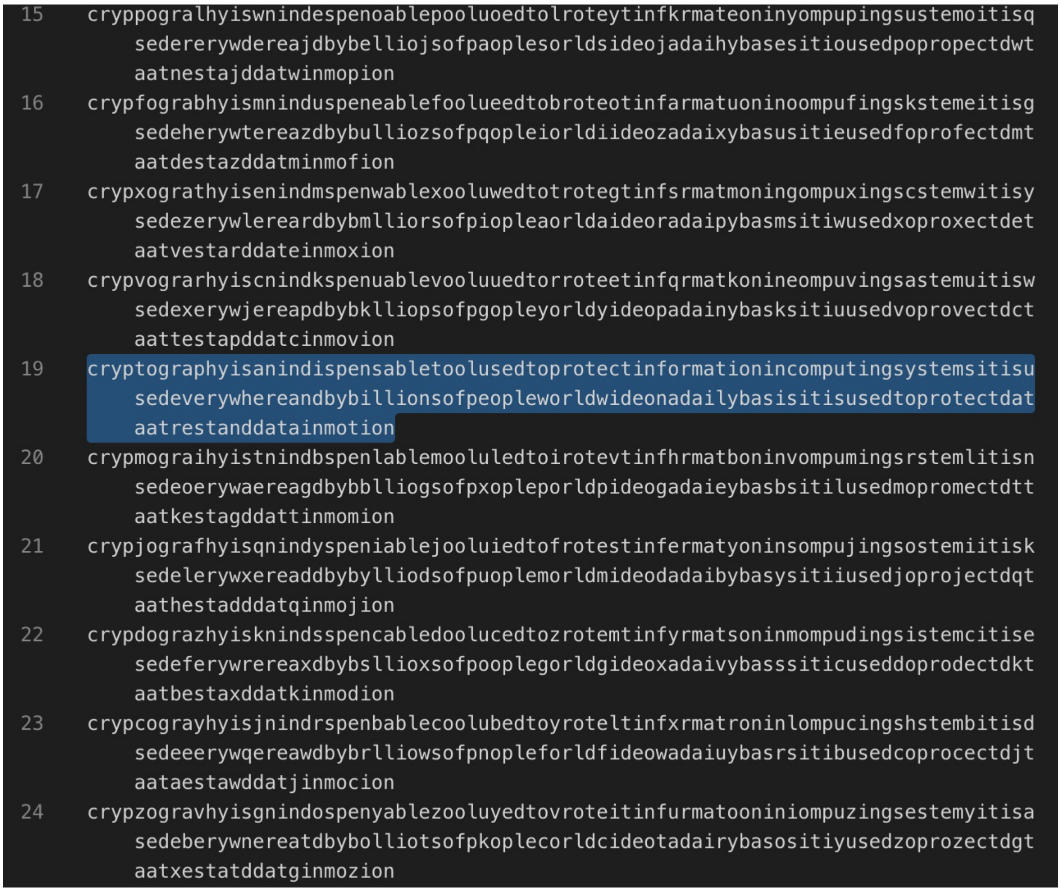
Code

```python
from nostril import nonsense

INPUT_FILENAME = "brute_force_results.txt"
OUTPUT_FILENAME = "meaningful_results.txt"

with open(INPUT_FILENAME) as input_file, open(OUTPUT_FILENAME, "w") as
output_file:
    for line in input_file:
        if not nonsense(line):
            output_file.write(line)
```

Output

An empty file when inputting the output file of A.6.

A 41 line file when inputting the output file of A.5. Below are 10 lines chosen from the file.



```
cryppogralhyiswnindespenoablepooluoedtolroteytinfkrmateoninyompupingsustemoitisqsedererywdereajdbybelliojsofpaoplesorldsideojadaihybasesitiousedpopropectdwtaatnestajddatwinmopion
crypfograbhyismnninduspeneablefooluedtobroteotinfarmatuoninoompufingskstemeitisgsedeherywtereazdbybulliozsofpqopleiorldiideozadaixybasusitieusedfoprofectdmtaatdestazddatminmofion
crypxograthyisenindmspenwablexooluwedtotrotegtinfsrmatmoningompuxingscstemwitisysedezerywlereardbybmlliorsofpiopleaorldaideoradaipybasmsitiwusedxoproxectdetaatvestarddateinmoxion
crypvograrhyiscnindkspenuablevooluuedtorroteetinfqrmatkonineompuvingsastemuitiswsedexerywjereapdbybklliopsofpgopleyorldyideopadainybasksitiuusedvoprovectdctaattestapddatcinmovion
cryptographyisanindispensabletoolusedtoprotectinformationincomputingsystemsitisusedeverywhereandbybillionsofpeopleworldwideonadailybasisitisusedtoprotectdataatrestanddatainmotion
crypmograihyistnindbspenlablemoolulедtoirotevtinfhrmatboninvompumingsrstemlitisnsedeoerywaereagdbybblliogsofpxopleporldpideogadaieybasbsitilusedmopromectdttaatkestagddattinmomion
crypjografhyisqnindyspeniablejooluiedtofrotestinfermatyoninsompujingsostemiitisksedelerywxereaddbybylliodsofpuoplemorldmideodadaibybasysitiiusedjoprojectdqtaathestadddatqinmojion
crypdograzhyisknindsspencabledoolucedtozrotemtinfyrmatsoninmompudingsistemcitisesedeferywrereaxdbybsllioxsofpooplegorldgideoxadaivybasssiticuseddoprodectdktaatbestaxddatkinmodion
crypcograyhyisjnindrspenbablecoolubedtoyroteltinfxrmatroninlompucingshstembitisdsedeeerywqereawdbybrlliowsofpnopleforldfideowadaiuybasrsitibusedcoprocectdjtaataestawddatjinmocion
crypzogravhyisgnindospenyablezooluyedtovroteitinfurmatooniniompuzingsestemyitisasedeberywnereatdbybolliotsofpkoplecorldcideotadairybasositiyusedzoprozectdgtaatxestatddatginmozion
```

A.8 get_frequencies.py

Code

```
from itertools import takewhile, count
from string import ascii_lowercase

CIPHERTEXT = "GEHDRSTAONLLRGYRVWRGWCNBQEOUSRSBUIQIQCCNVBCSA" + \
    "XVWTMVZJHGSARBASZYIRMAPGWWGNAQMGRGSWRMSTIEH" + \
    "KFIENOLHOHPGPYRCLWBODCSCUSUSEURUMQNCLEQJWJCOJ" + \
    "GGWVCWQYFNRRSCACRIPCRYXNJHPIFCOLHQJHYMAVCRMBW"
CIPHER_LENGTH = 5

CIPHERTEXT = CIPHERTEXT.lower()

substrings = [’’.join(
    CIPHERTEXT[start_i + j * CIPHER_LENGTH] for j in takewhile(lambda j:
start_i + j * CIPHER_LENGTH < len(CIPHERTEXT), count())
    ) for start_i in range(CIPHER_LENGTH)]

frequencyLists = [
    sorted(((l, round(substring.count(l) / len(substring), 3)) for l
in ascii_lowercase if substring.count(l) > 0), key=(lambda t: t[1]),
reverse=True)
    for substring in substrings]

for list1 in frequencyLists:
    print(list1)
```

Output

```
[(’s’, 0.194), (’i’, 0.139), (’m’, 0.139), (’w’, 0.139), (’e’, 0.056),
(’h’, 0.056), (’v’, 0.056), (’x’, 0.056), (’c’, 0.028), (’g’, 0.028), (’l’,
0.028), (’p’, 0.028), (’r’, 0.028), (’y’, 0.028)]

[(’b’, 0.111), (’e’, 0.111), (’q’, 0.111), (’a’, 0.083), (’c’, 0.083),
(’o’, 0.083), (’v’, 0.083), (’f’, 0.056), (’g’, 0.056), (’z’, 0.056),
(’l’, 0.028), (’n’, 0.028), (’p’, 0.028), (’r’, 0.028), (’t’, 0.028), (’y’,
0.028)]

[(’c’, 0.139), (’j’, 0.139), (’n’, 0.139), (’r’, 0.111), (’u’, 0.111),
(’h’, 0.083), (’w’, 0.083), (’a’, 0.056), (’m’, 0.028), (’o’, 0.028), (’p’,
0.028), (’v’, 0.028), (’y’, 0.028)]

[(’c’, 0.143), (’g’, 0.114), (’r’, 0.114), (’s’, 0.114), (’h’, 0.086),
(’o’, 0.086), (’b’, 0.057), (’d’, 0.057), (’i’, 0.057), (’w’, 0.057), (’a’,
0.029), (’k’, 0.029), (’p’, 0.029), (’t’, 0.029)]

[(’r’, 0.171), (’g’, 0.114), (’l’, 0.114), (’q’, 0.114), (’y’, 0.086),
(’a’, 0.057), (’n’, 0.057), (’u’, 0.057), (’c’, 0.029), (’f’, 0.029),
(’j’, 0.029), (’m’, 0.029), (’p’, 0.029), (’s’, 0.029), (’t’, 0.029), (’w’,
0.029)]
```

A.9 compute_cipher.py

Code

```
CIPHERTEXT = "GEHDRSTAONLLRGYRVWRGWCNBQEOUSRSBUIQIQCCNVBCSA" + \
    "XVWTMVZJHGSARBASZYIRMAPGWWGNAQMGRGSWRMSTIEH" + \
    "KFIENOLHOHPGPYRCLWBODCSCUSUSEURUMQNCLEQJWJCOJ" + \
    "GGWVCWQYFNRRSCACRIPCRYXNJHPIFCOLHQJHYMAVCRMBW"
PLAINTEXT = "cryptographyisanindispensabletoolusedtoprotec" + \
    "tinformationincomputingsystemsitisusedevery" + \
    "whereandbybillionsofpeopleworldwideonadailyba" + \
    "sisitisusedtoprotectdataatrestanddatainmotion"
CIPHER_LENGTH = 5

CIPHERTEXT = CIPHERTEXT.lower()
PLAINTEXT = PLAINTEXT.lower()

cipher = ""
for i in range(CIPHER_LENGTH):
    oc = ord(CIPHERTEXT[i])
    op = ord(PLAINTEXT[i])
    cipher += chr(ord("a") + (oc   op) % 26)

print(cipher)
```

Output

```
enjoy
```