Cryptography 1st Project

Le Duy Nguyen **xnguye27**

1. Overview

The `affine cipher` is a type of monoalphabetic substitution cipher[1] and it consists of 2 keys, let's call it a and b, The key a must be a coprime[2] with m, where m is the number of allowed letters.
For standard English alphabet, which is used in this project, m is equal to 26 and a is a set of 12 elements $\{1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25\}$

2. Implementation

2.1. Encryption

The encryption is pretty straightforward, each character in the text will be encrypted using the following function

$$E(c) = (a * c_i + b) \bmod m$$

Whitespaces and newline characters will be put right into the output.

Pseudocode

```
for char in text
    if whitespace(char) or char == '\n'
        put(char)
    else
        put(encrypt_char(char))
```

2.2. Decryption

Same as encryption, each character in the text will be decrypted using the following function

$$D(c) = a^{-1} * (c_i - b) \bmod m$$

Decrypting is possible thanks to a and m being coprime[2], the multiplivative inverse[3] always exists for **any element** of a.

Pseudocode

```
for char in text
    if whitespace(char) or char == '\n'
        put(char)
    else
        put(decrypt_char(char))
```

2.2.1. Finding the Multiplicative Inverse

There are serveral algorithms to easily find the multiplicative inverse, but with the **limited set of input** in this case, it is **more efficient** to just map it directly.

```
1 -> 1
3 -> 9
5 -> 21
7 -> 15
9 -> 3
11 -> 19
15 -> 7
17 -> 23
19 -> 11
21 -> 5
23 -> 17
25 -> 25
```

2.3. Breaking the code

2.3.1. Basic Idea

Breaking the affine cipher can be done by bruteforcing[4] it.
The number of key variants is $|a| * m$, in this case $12 * 26 = 326$.
In each iteration of bruteforcing process:

1. **decrypt the text** by the current keys
2. analyze the **frequency of each character** in the decrypted text
3. compare the frequency against the frequency of the target language and calculate its chi value[5]
4. store the **lowest chi value** as the result

The time complexity[7] of this algorithm is $O(n)$ for n is the number of character in the encrypted text

Pseudocode

```
lowest_chi_value = MAX
key = { a = 1, b = 0 }

for key_a in ALL_POSSIBLE_KEY_A
    for key_b in 0 to ALPHABET_LENGTH
        decrypted = decrypt(text, key_a, key_b)
        frequency = frequency_analyze(decrypted)
        chi_value = chi_squared_test(frequency)

        if chi_value < lowest_chi_value
            lowest_chi_value = chi_value
            key = { key_a, key_b }

return key
```

2.3.2. A Better Approach

Disclaimer: My implementation in `C` uses brute force attack, anything after this is just **on paper** because the permutation thing is extra complex, I did try to implement it but the code isn't that much readable with **7 level** of nested loop, and the outcome isn't that much faster than the brute force approach for this project inputs.

Without having to decrypt it **326 times**, we can directly **rearrange the frequency** of secret text and compare it against the frequency of target language. To get the keys after have the desired frequency, just simply loop through **every possible key *a***, generate key b for **every** character, in case all of them have the **same** key b => **BOMBA**!
The number of permutation is 26!, which sounds terrible, but we don't have to go through **every permutations**, just go through variations that look **nearly identical** to the frequency we want.
If we look at the frequency of use letters in Czech, the characters that most likely to appear are `EOAN` and characters that not likely to appear is `Q` and `W`. Those top rank character **can not** be in the bottom rank and vice versa.
The longer the text is, the smaller number of variations will be.

Pseudocode

```
lowest_chi_value = MAX
encrypted_frequency = frequency_analyze(text)
decrypted_frequency_mapping

for frequency_mapping in nearly_identical_variations(encrypted_frequency)
    chi_value = chi_squared_test(frequency_mapping)

    if chi_value < lowest_chi_value
        lowest_chi_value = chi_value
        decrypted_frequency_mapping = frequency_mapping

loop_a: for key_a in ALL_POSSIBLE_KEY_A
    key_b = get_key_b(key_a, decrypted_frequency_mapping[0])

    for character_map in decrypted_frequency_mapping[1..]
        if key_b != get_key_b(key_a, character_map)
            continue loop_a

    return { key_a, key_b }
```

2.3.3. Get Key b

Knowing how the encryption work, and if we have a, `original character` and `encrypted character`, let's call them o and e. By reverse the equation, we got this

$$(a * o_i - e_i) \bmod m = b$$

One problem is that e_i can be greater than $a * o_i$ which will give **negative value**, but thanks to Chinese remainder theorem[6], the problem can be solved by **adding m before subtracting** e_i, the final equation looks like this

$$(a * o_i + m - e_i) \bmod m = b$$

2.3.4. Frequency of Use Letters in Czech

The frequency of use letters in Czech **without diacritics** according to Slovo a slovesnost.

```
A: 0.08455 %
B: 0.01558 %
```

```
C: 0.02556 %
D: 0.03624 %
E: 0.10675 %
F: 0.00273 %
G: 0.00273 %
H: 0.01271 %
I: 0.07623 %
J: 0.02119 %
K: 0.03737 %
L: 0.03842 %
M: 0.03227 %
N: 0.06617 %
O: 0.08698 %
P: 0.03413 %
Q: 0.00001 %
R: 0.04914 %
S: 0.05321 %
T: 0.05769 %
U: 0.03942 %
V: 0.04662 %
W: 0.00009 %
X: 0.00076 %
Y: 0.02981 %
Z: 0.03194 %
```

3. Terminologies

3.1. Monoalphabetic Substitution Cipher

A type of classical cryptography. Each letter is mapped into another letter **independently** using a simple math function.

3.2. Coprime

Two number are coprime when the **only positive integer** that is a **divisor** of both of them is 1.

3.3. Modular Multiplicative Inverse

Given an integer a with respect to modulus m, if another integer x that is $(a * x) \bmod m = 1$, then x is the modular multiplicative inverse of a and it often seen as a^{-1}.

3.4. Brute Force Attack

It is a general problem-solving technique, it can solve basically **every problem** that does not have infinite cases by iterate through every possible candidates to find the right answer.

3.5. Chi-squared Test

The test calculates a value called the Chi-squared statistic, which measures the degree of difference between expected and observed frequencies.
In other word, the lowest value of the test means that the decryption was successful with high probability.

$$\mathrm{X}^2 = \sum_{i=0}^{n} \frac{x_i^2}{m_i} - n$$

3.6. Chinese Remainder Theorem

It states that if one knows the remainders of the Euclidean division of an integer n by several integers, then one can determine **uniquely** the remainder of the division of n by the product of these integers.

3.7. Time Complexity

The computational complexity that describes the **amount of computer time** it takes to run an algorithm.

4. Resources

This project cannot be done without **any** of these

- Information from the assignment by Ing. Daniel Snášel
- Presentations for cryptography course by Dr. Ing. Petr Hanáček
- https://en.wikipedia.org/wiki/Modular_multiplicative_inverse
- https://en.wikipedia.org/wiki/Frequency_analysis
- https://en.wikipedia.org/wiki/Affine_cipher
- https://en.wikipedia.org/wiki/Brute-force_attack
- https://en.wikipedia.org/wiki/Chi-squared_test
- https://en.wikipedia.org/wiki/Combinatorics
- http://sas.ujc.cas.cz/archiv.php?art=2913