Segurança Computacional
Lista de Exercícios 1

Luca Heringer Megiorin - 231003390
Universidade de Brasília

21 de Abril, 2025

1 Introdução

A comunicação é uma parte essencial da vivência em sociedade. Desde a antiguidade, porém, questões militares e políticas impulsionaram a necessidade de passar informações em segredo, sendo a cifra de transposição usadas por espartanos e a cifra de substituição usada por César uns dos primeiros registros históricos da criptografia para esses contextos [1]. Da mesma forma que havia a necessidade do segredo, os inimigos desses povos desejavam decifrar tais textos cifrados com as informações que possuíssem, prática que, atualmente, é chamada de criptoanálise. Com o advento da computação, algoritmos matemáticos desenvolvidos até então, e muitos outros que vieram depois, passaram a ser usados para criptografar mensagens, não só para a segurança de dados confidenciais políticos ou militares, mas também para dados pessoais.

Neste documento, serão exploradas as cifras de deslocamento (seção 2) e de transposição colunar (seção 3), bem como as suas respectivas criptoanálises por força bruta e análise de frequência, possuindo apenas o texto cifrado (e a acesso ao algoritmo de encriptação e decriptação). O código em C desenvolvido está disponível no github (vide o apêndice). Vale notar que o uso do tipo `string` nada mais passa que um `typedef char* string`.

2 Cifra de Deslocamento

A cifra de deslocamento se trata de uma cifra de substiuição na qual todas as letras do alfabeto serão substituídas por outra letra do alfabeto com um deslocamento escolhido pela chave. O exemplo mais famoso dessa cifra é a cifra de César, em que todas as letras são substituídas por aquelas 3 posições adiante (capítulo 5.2 [2]).

2.1 Algoritmo da Cifra de Deslocamento

Esse tipo de cifra utiliza-se da aritmética modular, de modo que a letra Z, ao ser deslocada, irá voltar para o início do alfabeto e, dado que o português tem um alfabeto de 26 letras (ignorando sinais ortográficos), pode-se dizer que, para encriptar uma mensagem M com uma chave K, a função seria dada por:

$$Enc_K(M) = p + K \pmod{26}\text{ , para cada caracter } p \in M.$$

De semelhante modo à função de encriptação, a função para decriptar texto cifrado C com uma chave K é dada por:

$$Dec_K(C) = p - K \pmod{26}\text{ , para cada caracter } p \in C.$$

Com isso, foi-se desenvolvido um código para encriptar (código 1) e outro para decriptar (código 2) uma mensagem utilizando a cifra de deslocamento, (presente no arquivo *shift_cipher.c*):

```
// Encryption using shift cipher: O(n), where n = strlen(plain_text)
string enc_shift(string plain_text, int key){
    int len = strlen(plain_text);
    string cipher_text = malloc(sizeof(char)*len + 1);

    if (!cipher_text) return NULL;

    for (int i = 0; i < len; i++){
        // Doesn't shift it character isn't in the alphabet
        if (isalpha(plain_text[i])){
            if (isupper(plain_text[i])){
                cipher_text[i] = mod((plain_text[i] - 'A' + key),
                    ↪ APLHA_LEN) + 'A';
            } else if (islower(plain_text[i])){
                cipher_text[i] = mod((plain_text[i] - 'a' + key),
                    ↪ APLHA_LEN) + 'a';
            }
        } else {
            cipher_text[i] = plain_text[i];
        }
    }
    cipher_text[len] = '\0';
    return cipher_text;
}
```

Código 1: Função de Encriptação da Cifra de Deslocamento, presente no arquivo *shift_cipher.c*

Como visto no código 1 acima, a complexidade de tempo do algoritmo de encriptar é O(n) para todos os casos (n é o tamanho do texto em claro), dado que o algoritmo apenas soma o deslocamento (em módulo 26) a cada caracter do texto em claro. Isso também é visto na função de decriptar (código 2) abaixo, cuja complexidade de tempo também será O(n), sendo n o tamanho do texto cifrado. Como o tamanho da mensagem não é alterado, pode-se dizer que o tempo para encriptar ou decriptar uma mensagem

cresce linearmente proporcional ao tamanho do texto em claro. Vale ressaltar que a função `mod()` implementada (presente no arquivo *operation_functions.c*) apenas usa o operador de módulo nativo do C (%) e garante que o resultado é sempre positivo (nesse caso, um número de 0 a 25).

```c
// Decryption using shift cipher: O(n), where n = strlen(cipher_text)
string dec_shift(string cipher_text, int key){
    int len = strlen(cipher_text);
    string plain_text = malloc(sizeof(char)*len + 1);

    if (!plain_text) return NULL;

    for (int i = 0; i < len; i++){
        // Doesn't shift it character isn't in the alphabet
        if (isalpha(cipher_text[i])){
            if (isupper(cipher_text[i])){
                plain_text[i] = mod((cipher_text[i] - 'A' - key),
                    ↪ APLHA_LEN) + 'A';
            } else if (islower(cipher_text[i])){
                plain_text[i] = mod((cipher_text[i] - 'a' - key),
                    ↪ APLHA_LEN) + 'a';
            }
        } else {
            plain_text[i] = cipher_text[i];
        }
    }
    plain_text[len] = '\0';
    return plain_text;
}
```

Código 2: Função de Decriptação da Cifra de Deslocamento, presente no arquivo *shift_cipher.c*

O código desenvolvido possui uma interface ao ser executado, que espera usuário escolher uma mensagem e uma chave, que será usada para encriptar e decriptar a mensagem, como pode ser vista no exemplo abaixo, que simula a visão do usuário:

```
SHIFT CIPHER: Encryption and Decryption mode.
Description: Write your plaintext and choose a key (between 0 and 25)
   for the encryption system.
The system will show the ciphertext as well as the decrypted message
   afterwards.
NOTE: Resulting plaintexts and ciphertexts won't have any spaces nor
   punctuation. Non ASCII characters won't be processed (so reults
   may be underwhelming).

Write your plain_text: Eu sou a mensagem, o texto em claro M!
Input a Key (between 0 and 25): 7
--------------------------------------------------------------------------
SHIFT CIPHER: Encryption and Decryption Mode.

Plaintext chosen: EusouamensagemotextoemclaroM
Chosen Key: 7
Resulting Ciphertext: LbzvbhtluzhnltvaleavltjshyvT
- Time elapsed during encryption: 0.002400 ms
Message decrypted: EusouamensagemotextoemclaroM
- Time elapsed during decryption: 0.001800 ms
```

```
Press ENTER key to return.
```

Terminal 1: Exemplo de Encriptação e Decriptação com Cifra de Deslocamento

O terminal 1 mostra a encriptação e decriptação da mensagem "Eu sou a mensagem, o texto em claro M!" com a chave $K = 7$. A mensagem passa por um pré-processamento, que retira espaços e outras pontuações, e é enviada para a função de encriptação. Desta forma, a mensagem que é encriptada, na realidade, é "Eusouamensagemotextoemclaroм". Ao somar 7 em cada uma das letras, é obtido o texto cifrado "LbzvbhtluzhnltvaleavltjshyvT" e, ao decriptá-lo, volta-se para o texto inicial. Vale notar que o tempo tomado pela funçao de encriptação e decriptação foi extremamente pequeno, e ambos os tempos são próximos uns dos outros, refletindo a complexidade O(n) abordada previamente.

2.2 Criptoanálise da Cifra de Deslocamento

A segurança de um algoritmo é fortemente ligada à segurança da chave, uma vez que, se descoberta, a mensagem pode ser decriptografada. Dessa forma, a segurança está mais relacionada à chave do que à complexidade de tempo do algoritmo, ou seja, O(n) não necessariamente significa ser mais, ou menos, seguro que uma complexidade maior. Tudo depende de como será feita a criptoanálise e de quão forte é a chave. Todavia, assim como um algoritmo com complexidade baixa deixa a encriptação e decriptação mais eficientes, este também deixará a criptoanálise mais eficiente no momento em que o criptoanalista usar a função de decriptação junto a uma chave que seu algoritmo encontrou.

Dessa forma, apesar de a cifra de deslocamento possuir uma complexidade de encriptação e decriptação desejável, como há apenas 26 possibilidades de chave (sendo uma delas o deslocamento 0, ou seja, não há substituição), a criptoanálise da cifra de deslocamento pode ser facilmente feita por força bruta. O código abaixo (presente no arquivo *shift_cipher.c*) representa o algoritmo de criptoanálise por força bruta, o qual recebe um texto cifrado e retorna a chave encontrada (caso o usuário escolha alguma das chaves) ou retorna -1 para indicar que não encontrou chave (isso ocorre apenas se o usuário não escolher nenhuma chave):

```
1  // Brute force cryptoanalysis for shift cipher: O(n), where n =
   ↪ strlen(cipher_text)
2  int brute_shift_cryptoanalysis(string cipher_text, double* ui_time){
3      int len = strlen(cipher_text);
4      string found_text = malloc(sizeof(char)*len + 1);
5      if (!found_text) return -1;
6
7      double start_time, end_time;
8      for (int i = 0; i < APLHA_LEN; i++){
9          found_text = dec_shift(cipher_text, i);
10
11         start_time = get_time_ms();
12         if (shift_cipher_ui_brute(i,found_text)) {
13             end_time = get_time_ms();
14             *ui_time += end_time - start_time;
15             // UI stuff (doesn't impact significantly complexity)
```

```
            free(found_text);
            return i;             // The key used
        }
        end_time = get_time_ms();
        *ui_time += end_time - start_time;
    }

    // Error (NO FOUND TEXT)
    free(found_text);
    return -1;
}
```

Código 3: Função de Criptoanálise por Força Bruta da Cifra de Deslocamento

É possível notar que o loop ocorre no máximo 26 vezes (denotado por `ALPHA_LEN`) e, para cada iteração, é chamada a função de decriptar (que tem complexidade de O(n)). Sendo assim, a complexidade desse algoritmo é de O(26 * n), ou O(n), dado que 26 é uma constante. Isso mostra que é uma cifra fácil de ser quebrada por força bruta, dado que, sendo a complexidade igual à do algoritmo de encriptação, o tempo gasto até achar a mensagem correta se aproxima do tempo de encriptar, que, como dito anteriormente, é extremamente rápido. Neste caso, a complexidade do algoritmo influenciou na facilidade da criptoanálise, de forma que a decifração é mais rápida, mas é o fato de haver apenas 26 possibilidades de chaves que realmente permitiu que a decifração pudesse se aproveitar da velocidade do algoritmo de decriptação da melhor forma. Isso pode ser visto no exemplo do terminal 2 abaixo:

```
SHIFT CIPHER: Cryptanalysis Mode.
Description: Write your plaintext. A key will be set pseudo-randomly.
Choose a method for performing the cryptanalysis afterwards.
NOTE: Resulting plaintexts and ciphertexts won't have any spaces nor
    punctuation. Non ASCII characters won't be processed (so reults
    may be underwhelming).

Write your plain_text: Eu sou a mensagem, o texto em claro M!
A key has been chosen // COMENTÁRIO1: chave escolhida
    pseudo-aleatoreamente
--------------------------------------------------------------------------
SHIFT CIPHER: Cryptanalysis Mode.
Choose method of cryptanalysis:
| 1. Brute Force
| 2. Frequency Analysis
| 3. Return

Your current ciphertext: CsqmsykclqyeckmrcvrmckajypmK

Type the number to select your answer: 1 // COMENTÁRIO2: escolha da
    criptoanálise por força bruta
--------------------------------------------------------------------------
//COMENTÁRIO3: primeira iteração (as outras serão puladas para não
    tomar muito espaço, mas seguem esse padrão)
SHIFT CIPHER: Cryptanalysis Mode (Brute Force)
Key used: 0
Possible message: CsqmsykclqyeckmrcvrmckajypmK
```

```
Choose an option:
| 1. Continue, this isn't the message I want.
| 2. Return, this is the message I'm looking for.

Type the number to select your answer: 1
------------------------------------------------------------------------
//COMENTÁRIO4: ao encontrar a chave, usuário seleciona a opção 2
SHIFT CIPHER: Cryptanalysis Mode (Brute Force)
Key used: 24
Possible message: EusouamensagemotextoemclaroM

Choose an option:
| 1. Continue, this isn't the message I want.
| 2. Return, this is the message I'm looking for.

Type the number to select your answer: 2
------------------------------------------------------------------------
//COMENTÁRIO5: resultado final: a chave éretornada e o texto é
    decriptado usando esta chave. O tempo total gasto na criptoanálise
    é mostrado, ignorando o tempo em que o usuário fazia a escolha.
SHIFT CIPHER: Cryptanalysis Mode (Brute Force)
Resulting Ciphertext: CsqmsykclqyeckmrcvrmckajypmK
Key found through cryptanalysis: 24
Message deciphered through cryptanalysis: EusouamensagemotextoemclaroM
- Time elapsed during cryptanalysis (Ignoring time spent on UI
    prompting): 0.096100 ms

Press ENTER key to return.
```

Terminal 2: Exemplo de Criptoanálise por Força Bruta com Cifra de Deslocamento

O exemplo acima usou a força bruta e em 25 iterações encontrou a chave ($K = 24$) para o texto cifrado. O tempo tomado foi relativamente rápido, pois foram necessárias algumas iterações em que o texto cifrado era decriptado, mas pode-se ver que ele se aproxima do tempo tomado pelo algoritmo de decriptação (quando é levado em conta que este foi chamado 25 vezes). Isso é melhorado quando a análise por frequências é utilizada.

Como a cifra de deslocamento substitui uma letra por outra, isso implica que haverá uma variação na frequência em que as letras irão aparecer [3]. Desta forma, foi desenvolvido um algoritmo que checasse as frequências das letras utilizadas e testasse a chave mais provável baseada no deslocamento das frequências a partir da letra mais comum do português (letra "A") e itera com base nas letras mais frequentes do português em ordem (frequências presentes em [5]). O código descrito abaixo está presente no arquivo *shift_cipher.c*:

```
1
2 //Frequency Cryptanalysis -- Complexity: O(n), where n =
    ↪ strlen(cipher_text)
3
4 static float SINGLE_FREQ[] = {13.9, 1.0, 4.4, 5.4, 12.2, 1.0, 1.2,
    ↪ 0.8, 6.9, 0.4, 0.1, 2.8, 4.2, 5.3, 10.8, 2.9, 0.9, 6.9, 7.9,
    ↪ 4.9, 4.0, 1.3, 0.0, 0.3, 0.0, 0.4};
5 // Order of most frequent letters:    a  e  o   s   i  r   d  n   t
    ↪ c  m   u   p   l   v   g  b  f  q   h  j  z   x   k   w   y
```

```
static int FREQUENT_SINGLE_INDEX[] = {0, 4, 14, 18, 8, 17, 3, 13, 19,
    ↪ 2, 12, 20, 15, 11, 21, 6, 1, 5, 16, 7, 9, 25, 23, 10, 22, 24};

int freq_shift_cryptoanalysis(string cipher_text, double* ui_time){

    float currentSingleFreq[APLHA_LEN] = {0};
    int len = strlen(cipher_text);
    if (len == 0){
        return -1; // Error
    }

    for (int i = 0; i < len; i++){
        // Frequency analysis will only care about ASCII alphabet
           ↪ characters
        if (isalpha(cipher_text[i])){
            if (isupper(cipher_text[i])){
                currentSingleFreq[(cipher_text[i] - 'A')]++;
            } else if (islower(cipher_text[i])){
                currentSingleFreq[(cipher_text[i] - 'a')]++;
            }
        }
    }

    // Finishing table (dividing by len to get frequency in current
       ↪ cipher text)
    int max = 0;  // Stores the index to the 2 highest frequencies
    for (int i = 0; i < APLHA_LEN; i++){
        currentSingleFreq[i] = currentSingleFreq[i] * 100/len;

        // Gets the two highest frequencies indexes
        if (currentSingleFreq[i] > currentSingleFreq[max]){
            max = i;
        }
    }

    // Checks for most likely frequencies:
    int possible_key;
    string found_text = malloc(sizeof(char)*len + 1);
    if (!found_text) return -1;

    int steps =
       ↪ sizeof(FREQUENT_SINGLE_INDEX)/sizeof(FREQUENT_SINGLE_INDEX[0]);
    double start_time;
    double end_time;
    for (int i = 0; i < steps; i++){
        possible_key = mod(max - (FREQUENT_SINGLE_INDEX[i]),
           ↪ APLHA_LEN);
        found_text = dec_shift(cipher_text, possible_key);


        start_time = get_time_ms();
        if (shift_cipher_ui_freq(possible_key, found_text, max + 'A',
           ↪ currentSingleFreq[max], FREQUENT_SINGLE_INDEX[i] + 'A',
           ↪ SINGLE_FREQ[FREQUENT_SINGLE_INDEX[i]], i + 1)) {
            end_time = get_time_ms();
            *ui_time += end_time - start_time;
```

```
            // UI stuff (doesn't impact significantly complexity)
            free(found_text);
            return possible_key;      // The key used
        }
        end_time = get_time_ms();
        *ui_time += end_time - start_time;
    }

    // Error (NO FOUND TEXT)
    free(found_text);
    return -1;
}
```

Código 4: Função de Criptoanálise por Análise de Frequência com Cifra de Deslocamento, presente no arquivo *shift_cipher.c*

Apesar de o algoritmo assemelhar-se a um algoritmo de força bruta, ele possui a vantagem de que, quanto maior a amostra, mais as frequências das letras irão se aproximar da realidade. Sendo assim, uma mensagem como "Eu sou a mensagem, o texto em claro M!"(terminal 3 fará o código ter um comportamento próximo de força bruta, enquanto que o parágrafo inicial do livro *A Metamorfose*, de Franz Kafka, possui uma melhor distribuição e, portanto, é decifrado logo na primeira iteração (terminal 4):

```
SHIFT CIPHER: Cryptanalysis Mode.
Description: Write your plaintext. A key will be set pseudo-randomly.
Choose a method for performing the cryptanalysis afterwards.
NOTE: Resulting plaintexts and ciphertexts won't have any spaces nor
   punctuation. Non ASCII characters won't be processed (so reults
   may be underwhelming).

Write your plain_text: Eu sou a mensagem, o texto em claro M!
A key has been chosen // COMENTÁRIO1: chave escolhida
   pseudo-aleatoreamente
--------------------------------------------------------------------------
SHIFT CIPHER: Cryptanalysis Mode.
Choose method of cryptanalysis:
| 1. Brute Force
| 2. Frequency Analysis
| 3. Return

Your current ciphertext: HxvrxdphqvdjhprwhawrhpfodurP

Type the number to select your answer: 2 // COMENTÁRIO2: escolha da
   criptoanálise por análise de frequência
--------------------------------------------------------------------------
//COMENTÁRIO3: primeira iteração (as outras serão puladas para não
   tomar muito espaço, mas seguem esse padrão)
SHIFT CIPHER: Cryptanalysis Mode (Frequency Analysis)
Iteration 1:
- Key used: 7.
- Frequency of letter 'H' in ciphertext: 17.857143.
- Most likely to be the letter: 'A', with a frequency of: 13.900000.
Possible message: AqokqwiajowcaikpatpkaiyhwnkI

Choose an option:
```

```
| 1. Continue, this isn't the message I want.
| 2. Return, this is the message I'm looking for.

Type the number to select your answer: 1
-------------------------------------------------------------------------
//COMENTÁRIO4: ao encontrar a chave, usuário seleciona a opção 2
SHIFT CIPHER: Cryptanalysis Mode (Frequency Analysis)
Iteration 2:
- Key used: 3.
- Frequency of letter 'H' in ciphertext: 17.857143.
- Most likely to be the letter: 'E', with a frequency of: 12.200000.
Possible message: EusouamensagemotextoemclaroM

Choose an option:
| 1. Continue, this isn't the message I want.
| 2. Return, this is the message I'm looking for.

Type the number to select your answer: 2
-------------------------------------------------------------------------
//COMENTÁRIO5: resultado final: a chave éretornada e o texto é
    decriptado usando esta chave. O tempo total gasto na criptoanálise
    é␣mostrado, ignorando o tempo em que o usuário fazia a escolha.
SHIFT CIPHER: Cryptanalysis Mode (Frequency Analysis)
Resulting Ciphertext: HxvrxdphqvdjhprwhawrhpfodurP
Key found through cryptanalysis: 3
Message deciphered through cryptanalysis: EusouamensagemotextoemclaroM
- Time elapsed during cryptanalysis (Ignoring time spent on UI
    prompting): 0.010900 ms

Press ENTER key to return.

Press ENTER key to return.
```

Terminal 3: Exemplo de Criptoanálise por Análise de Frequência de uma mensagem curta com Cifra de Deslocamento

```
SHIFT CIPHER: Cryptanalysis Mode.
Description: Write your plaintext. A key will be set pseudo-randomly.
Choose a method for performing the cryptanalysis afterwards.
NOTE: Resulting plaintexts and ciphertexts won't have any spaces nor
    punctuation. Non ASCII characters won't be processed (so reults
    may be underwhelming).

Write your plain_text: Quando certa manha Gregor Samsa acordou de
    sonhos intranquilos, em sua cama meta-morfoseado num inseto
    monstruoso. Estava deitado sobre suas costas duras como couraca e,
    ao levantar um pouco a cabeca, viu seu ventre abaulado, marrom,
    dividido por nervuras arqueadas, no topo de qual a coberta,
    prestes a deslizar de vez, ainda mal se sustinha. Suas numerosas
    pernas, lastimavelmente finas em comparacao com o volume do resto
    do corpo, tremulavam desamparadas diante dos seus olhos.
A key has been chosen // COMENTÁRIO1: chave escolhida
    pseudo-aleatoreamente
-------------------------------------------------------------------------
SHIFT CIPHER: Cryptanalysis Mode.
Choose method of cryptanalysis:
```

```
| 1. Brute Force
| 2. Frequency Analysis
| 3. Return

Your current ciphertext:
PtzmcnbdqszlzmgzFqdfnqRzlrzzbnqcntcdrnmgnrhmsqzmpthknrdl
rtzbzlzldszlnqenrdzcnmtlhmrdsnlnmrsqtnrnDrszuzcdhszcnrna
qdrtzrbnrszrctqzrbnlnbntqzbzdznkduzmszqtlontbnzbzadbzuht
rdtudmsqdzaztkzcnlzqqnlchuhchcnonqmdqutqzrzqptdzczrmnsno
ncdptzkzbnadqszoqdrsdrzcdrkhyzqcdudyzhmczlzkrdrtrshmgzRt
zrmtldqnrzrodqmzrkzrshlzudkldmsdehmzrdlbnlozqzbznbnlnunk
tldcnqdrsncnbnqonsqdltkzuzlcdrzlozqzczrchzmsdcnrrdtrnkgnr

Type the number to select your answer: 2 // COMENTÁRIO2: escolha da
    criptoanálise por análise de frequência
--------------------------------------------------------------------------
//COMENTÁRIO3: primeira iteração (chave já foi encontrada)
SHIFT CIPHER: Cryptanalysis Mode (Frequency Analysis)
Iteration 1:
- Key used: 25.
- Frequency of letter 'Z' in ciphertext: 15.267176.
- Most likely to be the letter: 'A', with a frequency of: 13.900000.
Possible message:
QuandocertamanhaGregorSamsaacordoudesonhosintranquilosem
suacamametamorfoseadonuminsetomonstruosoEstavadeitadosob
resuascostasdurascomocouracaeaolevantarumpoucoacabecaviu
seuventreabauladomarromdivididopornervurasarqueadasnotop
odequalacobertaprestesadeslizardevezaindamalsesustinhaSu
asnumerosaspernaslastimavelmentefinasemcomparacaocomovol
umedorestodocorpotremulavamdesamparadasdiantedosseusolhos

Choose an option:
| 1. Continue, this isn't the message I want.
| 2. Return, this is the message I'm looking for.

Type the number to select your answer: 2
--------------------------------------------------------------------------
//COMENTÁRIO4: resultado final: a chave éretornada e o texto é
    decriptado usando esta chave. O tempo total gasto na criptoanálise
    é␣mostrado, ignorando o tempo em que o usuário fazia a escolha.
SHIFT CIPHER: Cryptanalysis Mode (Frequency Analysis)
Resulting Ciphertext:
PtzmcnbdqszlzmgzFqdfnqRzlrzzbnqcntcdrnmgnrhmsqzmpthknrdl
rtzbzlzldszlnqenrdzcnmtlhmrdsnlnmrsqtnrnDrszuzcdhszcnrna
qdrtzrbnrszrctqzrbnlnbntqzbzdznkduzmszqtlontbnzbzadbzuht
rdtudmsqdzaztkzcnlzqqnlchuhchcnonqmdqutqzrzqptdzczrmnsno
ncdptzkzbnadqszoqdrsdrzcdrkhyzqcdudyzhmczlzkrdrtrshmgzRt
zrmtldqnrzrodqmzrkzrshlzudkldmsdehmzrdlbnlozqzbznbnlnunk
tldcnqdrsncnbnqonsqdltkzuzlcdrzlozqzczrchzmsdcnrrdtrnkgnr
Key found through cryptanalysis: 25
Message deciphered through cryptanalysis:
QuandocertamanhaGregorSamsaacordoudesonhosintranquilosem
suacamametamorfoseadonuminsetomonstruosoEstavadeitadosob
resuascostasdurascomocouracaeaolevantarumpoucoacabecaviu
seuventreabauladomarromdivididopornervurasarqueadasnotop
odequalacobertaprestesadeslizardevezaindamalsesustinhaSu
```

```
asnumerosaspernaslastimavelmentefinasemcomparacaocomovol
umedorestodocorpotremulavamdesamparadasdiantedosseusolhos
- Time elapsed during cryptanalysis (Ignoring time spent on UI
    prompting): 0.037600 ms

Press ENTER key to return.
```

Terminal 4: Exemplo de Criptoanálise por Análise de Frequência com o parágrafo inicial do livro "A Metamorfose"com Cifra de Deslocamento

Como, essa análise por frequência nada mais passa de um ataque de força bruta, mas com uma heurística de frequência, a complexidade do algoritmo é de O(n). Vale notar que, mesmo sendo da mesma complexidade de O(n) que o algoritmo de força bruta, ao forçar a chave como 25, foi feito um teste com o mesmo parágrafo inicial de Kafka e o tempo tomado foi de 0.547900 ms, o que mostra que a análise de frequência tem uma melhor eficiência para textos maiores.

3 Cifra de Transposição Colunar

Para a segunda parte do exercício, era necessário desenvolver um algoritmo de cifra de transposição. O algoritmo escolhido foi de transposição colunar com preenchimento, sendo uma variação dele descrita em [4].

3.1 Algoritmo da Cifra de Transposição Colunar

Esse tipo de cifra consiste em permutar colunas com partes do texto em claro escrito em uma matriz da largura do tamanho da chave com base na ordem alfabética das letras (que não se repetem) da chave. Por exemplo, a mensagem "Eu sou a mensagem, o texto em claro M!", após retirar espaços e pontuações, ficaria distribuída dessa forma com uma chave "SENHA":

S	**E**	**N**	**H**	**A**
5	2	4	3	1
E	U	S	O	U
A	M	E	N	S
A	G	E	M	O
T	E	X	T	O
E	M	C	L	A
R	O	M	#	#

Tabela 1: Distribuição da mensagem em matriz formada pela chave "SENHA"

Os espaços que sobram na matriz são preenchidos com *nulls*, que, neste caso, são os caracteres "#". Assim, o texto cifrado formado, seguindo a ordem alfabética das letras

da chave, iria ser a sequência das letras das colunas USOOA# UMGEMO ONMTL# SEEXCM EAATER, ou seja, "usooa#umgemoonmtl#seexcMEaater".

Com base nessa lógica, foi desenvolvido o algoritmo para encriptar. Para permitir que fossem exploradas diversos casos de criptoanálise, a chave, apesar de ter tamanho fixo, pode ter o seu tamanho escolhido antes de executar os algoritmos (o tamanho da chave é conhecido por todos, pois representa a simulação de um algoritmo com tamanho de chave fixo, apenas com a diferença de que o usuário pode escolher qual será o tamanho fixo). Dado que não é possível repetir as letras da chave, o tamanho mínimo é 2 (pois o tamanho 1 não permite permutação) e o tamanho máximo é 26. O resultado é visto a seguir (código presente no arquivo *transposition_cipher.c*):

```
1  // Encryption using shift cipher: O(|K|^2 + N), where:
2  // -  N  is roughly the length of the plaintext (+ necessary padding)
3  // - |K| is the length of the key K
4  string enc_transp(string plain_text, string key){
5      int msg_len = strlen(plain_text);                          // Length
       ↪ of message
6      int key_len = strlen(key);                                 // Length
       ↪ of key (width of cipher matrix)
7      int matrix_height = ceil(msg_len/ (float) key_len);        // Height
       ↪ of cipher matrix
8      int len = matrix_height * key_len;                         // Lenght
       ↪ of ciphertext
9
10     string cipher_text = malloc(sizeof(char)*len + 1);
11     if (!cipher_text) return NULL;
12
13
14     // Order of indexes for columnar transposition -- O(|K|), where
       ↪ |K| is the length of the key K
15     int key_order[key_len];
16     for (int i = 0; i < key_len; i++){
17         key_order[i] = i;
18     }
19
20     // Simple bubble sorting (to know the order of indexes related to
       ↪ key) --  O(|K|^2), where |K| is the length of the key K
21     for (int i = 0; i < key_len; i++){
22         for (int j = 0; j < i; j++){
23             if((tolower(key[key_order[j]]) - 'a') >
                   ↪ (tolower(key[key_order[i]]) - 'a')){
24                 int temp = key_order[i];
25                 key_order[i] = key_order[j];
26                 key_order[j] = temp;
27             }
28         }
29     }
30
31     // Building matrix for transposing ciphertext --  O(N), where N is
       ↪ roughly the length of the plaintext (+ necessary padding)
32     char cipher_matrix[matrix_height][key_len];
33     int plainIndex;
34     for (int i = 0; i < matrix_height; i++){
35         for (int j = 0; j < key_len; j++){
```

```
            plainIndex = (key_len * i) + j;
            if (plainIndex >= msg_len){          // Padding
                cipher_matrix[i][j] = '#';
            } else {
                cipher_matrix[i][j] = plain_text[(key_len * i) + j];
            }
        }
    }

    // Creating ciphertext by concatenating matrixes on the order
       ↪ defined by orderIndex --  O(N), where N is roughly the
       ↪ length of the plaintext (+ necessary padding)
    int index = 0;
    for (int orderIndex = 0; orderIndex < key_len; orderIndex++){
        for (int rowIndex = 0; rowIndex < matrix_height; rowIndex++){
            cipher_text[index] =
               ↪ cipher_matrix[rowIndex][key_order[orderIndex]];
            index++;
        }
    }

    cipher_text[len] = '\0';
    return cipher_text;
}
```

Código 5: Função de Encriptação com Cifra de Transposição Colunar, presente no arquivo *transposition_cipher.c*

Quanto à decriptação, o processo inverso é feito: uma matriz é criada e ela é preenchida com o texto cifrado na ordem da chave, ou seja, para o texto cifrado obtido acima e a chave "SENHA", os primeiros $N/|K|$ caracteres seriam escritos na coluna correspondente à letra "A", em que N é o tamanho do texto cifrado e $|K|$ é o tamanho da chave. O processo continuaria na mesma lógica, como na sequência abaixo:

S	**E**	**N**	**H**	**A**
5	2	4	3	1
				U
				S
				O
				O
				A
				#

S	**E**	**N**	**H**	**A**
5	2	4	3	1
	U			U
	M			S
	G			O
	E			O
	M			A
	O			#

...

...

S	**E**	**N**	**H**	**A**
5	2	4	3	1
E	U	S	O	U
A	M	E	N	S
A	G	E	M	O
T	E	X	T	O
E	M	C	L	A
R	O	M	#	#

Tabela 2: Distribuição da mensagem em matriz formada pela chave "SENHA"na hora de decriptar

Ao final, retira-se o preenchimento, e obtém-se a mensagem original "EusouamensagemotextoemclaroM". O algoritmo foi desenvolvido no arquivo *transposition_cipher.c* e é visível abaixo:

```
// Decryption using shift cipher: O(|K|^2 + N), where:
// -  N  is the length of the ciphertext
```

```
// - |K| is the length of the key K
string dec_transp(string cipher_text, string key){
    int len = strlen(cipher_text);                          // Lenght of
        ↪ ciphertext
    int key_len = strlen(key);                              // Length of
        ↪ key (width of cipher matrix)
    int matrix_height = len / key_len;                      // Height of
        ↪ cipher matrix
    //int msg_len = len - ;                                   // Length
        ↪ of message

    //string cipher_text = malloc(sizeof(char)*len + 1);
    //if (!cipher_text) return NULL;


    // Order of indexes for columnar transposition -- O(|K|), where
        ↪ |K| is the length of the key K
    int key_order[key_len];
    for (int i = 0; i < key_len; i++){
        key_order[i] = i;
    }

    // Simple bubble sorting (to know the order of indexes related to
        ↪ key) --  O(|K|^2), where |K| is the length of the key K
    for (int i = 0; i < key_len; i++){
        for (int j = 0; j < i; j++){
            if((tolower(key[key_order[j]]) - 'a') >
                ↪ (tolower(key[key_order[i]]) - 'a')){
                int temp = key_order[i];
                key_order[i] = key_order[j];
                key_order[j] = temp;
            }
        }
    }

    // Decrypting ciphertext by reading matrix in the right order
        ↪ matrixes on the order defined by orderIndex --  O(N), where
        ↪ N is roughly the length of the plaintext (+ necessary
        ↪ padding)
    int index = 0;
    int paddingN = 0;
    char cipher_matrix[matrix_height][key_len];
    for (int orderIndex = 0; orderIndex < key_len; orderIndex++){
        for (int rowIndex = 0; rowIndex < matrix_height; rowIndex++){
            cipher_matrix[rowIndex][key_order[orderIndex]] =
                ↪ cipher_text[index];
            if (cipher_text[index] == '#'){
                paddingN++;
            }
            index++;
        }
    }

    // Building matrix for transposing ciphertext --  O(N), where N is
        ↪ the length of the ciphertext
    string plain_text = malloc(sizeof(char)*(len - paddingN) + 1);
```

```
    int plainIndex = 0;
    for (int i = 0; i < matrix_height; i++){
        for (int j = 0; j < key_len; j++){
            if (cipher_matrix[i][j] != '#'){//plainIndex < (len -
                ↪ paddingN)) {
                plain_text[plainIndex] = cipher_matrix[i][j];
                plainIndex++;
            }
        }
    }

    plain_text[len - paddingN] = '\0';
    return plain_text;
}
```

Código 6: Função de Decriptação com Cifra de Transposição Colunar, presente no arquivo *transposition_cipher.c*

Ao analisar, ambos os algoritmos no código 8 e código 6 possuem uma complexidade de O($|K|^2 + N$), em que $|K|$ é o tamanho da chave e N é o tamanho do texto em claro somado à quantidade de preenchimentos, ou seja, é o tamanho do texto cifrado resultante. Esse crescimento de complexidade comparado à cifra de deslocamento se dá em razão da chave poder ter diversos tamanhos fixos escolhidos. Caso a chave fosse de tamanho constante, seria possível dizer que a complexidade seria de O(N), mas isso seria ignorar o fato de que, se a chave tiver um tamanho diferente, haverá um impacto na execução do código, em razão do *bubble sort* usado para definir a ordem das colunas baseado na ordem das letras na chave. O exemplo utilizado acima pode ser visto em ação abaixo:

```
TRANSPOSITION CIPHER (COLUMNAR): Encryption and Decryption mode.
Description: Write your plaintext and choose a key (between 0 and 25)
    for the encryption system.
The system will show the ciphertext as well as the decrypted message
    afterwards.
NOTE: Resulting plaintexts and ciphertexts won't have any spaces nor
    punctuation. Non ASCII characters won't be processed (so reults
    may be underwhelming).

Write your plain_text: Eu sou a mensagem, o texto em claro M!
How long is your key? Input a number between 2 and 26: 5
Input key of your chosen length (No repeating characters, ASCII only):
    senha
--------------------------------------------------------------------------
TRANSPOSITION CIPHER (COLUMNAR): Encryption and Decryption Mode.

Plaintext chosen: EusouamensagemotextoemclaroM
Chosen Key: senha
Resulting Ciphertext: usooa#umgemoonmtl#seexcMEaater
- Time elapsed during encryption: 0.029400 ms
Message decrypted: EusouamensagemotextoemclaroM
- Time elapsed during decryption: 0.001400 ms

Press ENTER key to return.
```

Terminal 5: Um exemplo de Encriptação e Decriptação com a Transposição Colunar

```
TRANSPOSITION CIPHER (COLUMNAR): Encryption and Decryption mode.
Description: Write your plaintext and choose a key (between 0 and 25)
   for the encryption system.
The system will show the ciphertext as well as the decrypted message
   afterwards.
NOTE: Resulting plaintexts and ciphertexts won't have any spaces nor
   punctuation. Non ASCII characters won't be processed (so reults
   may be underwhelming).

Write your plain_text: Essa mensagem e um pouco mais longa que a
   anterior, mas a chave continua do mesmo tamanho
How long is your key? Input a number between 2 and 26: 5
Input key of your chosen length (No repeating characters, ASCII only):
   senha
--------------------------------------------------------------------------
TRANSPOSITION CIPHER (COLUMNAR): Encryption and Decryption Mode.

Plaintext chosen: Essamensagemeumpoucomaislongaqueaanteriormasachave
continuadomesmotamanho
Chosen Key: senha
Resulting Ciphertext: mgmolqnoaeioon#snmoaneemhoueaoaaucsaaisvtdma#
sseuigaraanasm#Eeepmoutrccnmth
- Time elapsed during encryption: 0.005200 ms
Message decrypted: Essamensagemeumpoucomaislongaqueaanteriormasachave
continuadomesmotamanho
- Time elapsed during decryption: 0.002700 ms

Press ENTER key to return.
```

Terminal 6: Um exemplo de Encriptação e Decriptação de uma mensagem mais longa com a Transposição Colunar

```
TRANSPOSITION CIPHER (COLUMNAR): Encryption and Decryption mode.
Description: Write your plaintext and choose a key (between 0 and 25)
   for the encryption system.
The system will show the ciphertext as well as the decrypted message
   afterwards.
NOTE: Resulting plaintexts and ciphertexts won't have any spaces nor
   punctuation. Non ASCII characters won't be processed (so reults
   may be underwhelming).

Write your plain_text: Eu sou a mensagem, o texto em claro M!
How long is your key? Input a number between 2 and 26: 8
Input key of your chosen length (No repeating characters, ASCII only):
   senhabig
--------------------------------------------------------------------------
TRANSPOSITION CIPHER (COLUMNAR): Encryption and Decryption Mode.

Plaintext chosen: EusouamensagemotextoemclaroM
Chosen Key: senhabig
Resulting Ciphertext: uee#amm#usxretl#ogoMmoc#satoEnea
- Time elapsed during encryption: 0.003400 ms
Message decrypted: EusouamensagemotextoemclaroM
- Time elapsed during decryption: 0.002000 ms
```

```
Press ENTER key to return.
```

Terminal 7: Outro exemplo de Encriptação e Decriptação com a Transposição Colunar

Ao comparar com o primeiro exemplo (terminal 5), é possível perceber que o tempo aumenta tanto em função do tamanho do texto em claro (terminal 6) quanto em função do tamanho da chave (terminal 7), corroborando a ideia de que a complexidade de tempo é dada por O($|K|^2 + N$).

3.2 Criptoanálise da Cifra por Transposição Colunar

Como abordado anteriormente, a segurança do algoritmo está relacionada à força da chave. Neste caso, considerando que o que importa na chave é apenas a ordem das letras, e não a letra em si (ou seja, uma chave "ABC"seria equivalente a uma chave "DEF"), a quantidade de chaves possíveis para uma chave de tamanho $|K|$ é $|K|!$.

Sendo assim, um algoritmo de força bruta teria que ver, no pior caso, $|K|!$ possibilidades, e esse foi o algoritmo desenvolvido (código presente no arquivo *transposition_cipher.c*):

```
1  // Brute force cryptoanalysis for shift cipher: O(|K|! * (|K|^2 + N))
2  string brute_transp_cryptoanalysis(string cipher_text, int key_len,
       ↪ double* ui_time){
3      double start_time, end_time;
4      string possible_key = get_transp_key(2, key_len); // Generates a
           ↪ non-permutated key
5
6      string found_text = dec_transp(cipher_text, possible_key);    //
           ↪ Applies non-permutated key
7
8      // First attempt
9      start_time = get_time_ms();
10     if (transp_cipher_ui_brute(possible_key,found_text)) {
11         end_time = get_time_ms();
12         *ui_time += end_time - start_time;
13         // UI stuff (doesn't impact significantly complexity)
14         free(found_text);
15         return possible_key;            // The key used
16     }
17     end_time = get_time_ms();
18     *ui_time += end_time - start_time;
19
20     // Array for keeping track of permutations -- O(|K|) , where |K|
           ↪ is the length of the key K
21     int permuteTracker[key_len];
22     for (int i = 0; i < key_len; i++) {
23         permuteTracker[i] = 0;
24     }
25
26     // Loop that iterates through all possible keys -- O(|K|! * (|K|^2
           ↪ + N))
27     int i = 0;
28     while (i < key_len) {
```

```c
        if (permuteTracker[i] < i) {
            if (i % 2 == 0) {
                swap_char(possible_key, possible_key + i);
            } else {
                swap_char(possible_key + permuteTracker[i],
                    ↪ possible_key + i);
            }

            found_text = dec_transp(cipher_text, possible_key);
                ↪ // O(|K|^2 + N)

            start_time = get_time_ms();
            if (transp_cipher_ui_brute(possible_key,found_text)) {
                end_time = get_time_ms();
                *ui_time += end_time - start_time;
                // UI stuff (doesn't impact significantly complexity)
                free(found_text);
                return possible_key;            // The key used
            }
            end_time = get_time_ms();
            *ui_time += end_time - start_time;

            permuteTracker[i]++;
            i = 0;
        } else {
            permuteTracker[i] = 0;
            i++;
        }
    }

    // Error (NO FOUND TEXT)
    free(found_text);
    free(possible_key);
    return NULL;
}
```

Código 7: Algoritmo de Ataque a Força Bruta com Cifra de Transposição Colunar, presente no arquivo *transposition_cipher.c*

É possível perceber que, como a função, no pior dos casos, cobre todas as permutações ($|K|!$) e, a cada permutação, é chamada a função de decriptação, que tem complexidade O($|K|^2 + N$), a complexidade deste algoritmo é dada por O($|K|! \times (|K|^2 + N)$), algo significativamente maior que o ataque em força bruta da cifra de deslocamento. Isso pode ser visto no exemplo a seguir:

```
TRANSPOSITION CIPHER (COLUMNAR): Cryptanalysis Mode.
Description: Write your plaintext. A key will be set pseudo-randomly.
Choose a method for performing the cryptanalysis afterwards.
NOTE: Resulting plaintexts and ciphertexts won't have any spaces nor
   punctuation. Non ASCII characters won't be processed (so reults
   may be underwhelming).

Write your plain_text: Eu sou a mensagem, o texto em claro M!
How long is your key? Input a number between 2 and 26: 5
A key has been chosen // COMENTÁRIO1: chave escolhida
```

```
   pseudo-aleatoreamente
------------------------------------------------------------------------
TRANSPOSITION CIPHER (COLUMNAR): Cryptanalysis Mode.
Choose method of cryptanalysis:
| 1. Brute Force
| 2. Frequency Analysis
| 3. Return

Your current ciphertext: onmtl#Eaaterusooa#umgemoseexcM

Type the number to select your answer: 1 // COMENTÁRIO2: escolha da
   criptoanálise por força bruta
------------------------------------------------------------------------
//COMENTÁRIO3: primeira iteração (as outras serão puladas para não
   tomar muito espaço, mas seguem esse padrão)
TRANSPOSITION CIPHER (COLUMNAR): Cryptanalysis Mode (Brute Force)
Key used: abcde
Possible message: oEuusnasmemaogettoexleamcroM

Choose an option:
| 1. Continue, this isn't the message I want.
| 2. Return, this is the message I'm looking for.

Type the number to select your answer: 1
------------------------------------------------------------------------
//COMENTÁRIO4: ao encontrar a chave, usuário seleciona a opção 2
   (ocorreu na iteração 59)
TRANSPOSITION CIPHER (COLUMNAR): Cryptanalysis Mode (Brute Force)
Key used: bdeac
Possible message: EusouamensagemotextoemclaroM

Choose an option:
| 1. Continue, this isn't the message I want.
| 2. Return, this is the message I'm looking for.

Type the number to select your answer: 2
------------------------------------------------------------------------
//COMENTÁRIO5: resultado final: a chave éretornada e o texto é
   decriptado usando esta chave. O tempo total gasto na criptoanálise
   é␣mostrado, ignorando o tempo em que o usuário fazia a escolha.
TRANSPOSITION CIPHER (COLUMNAR): Cryptanalysis Mode (Brute Force)
Resulting Ciphertext: onmtl#Eaaterusooa#umgemoseexcM
Key found through cryptanalysis: bdeac
True key (NOT USED FOR DECIPHERING, ONLY SHOWN FOR COMPARISON): hvzdl
Message deciphered through cryptanalysis: EusouamensagemotextoemclaroM
- Time elapsed during cryptanalysis (Ignoring time spent on UI
   prompting): 0.253000 ms

Press ENTER key to return.
```

Terminal 8: Exemplo de Criptoanálise por Força Bruta com Cifra de Transposição Colunar

Ao comparar o exemplo acima com o de força bruta da cifra de deslocamento (terminal 2), o tempo de execução desse ataque (0.253000 ms) demorou mais do que o dobro do ataque à cifra por deslocamento (0.096100 ms), o que mostra que esta cifra é mais

segura, a não ser que o tamanho da chave seja $|K| \leq 4$, uma vez que $4! = 24 \leq 26$, que são as possibilidades de chave para a cifra de deslocamento. Caso $|K| \leq 4$, haverá menos iterações máximas que a cifra por deslocamento, o que possibilitaria um tempo de execução mais rápido, porém a complexidade do algoritmo de decriptação pode influenciar nos resultados, por ser mais complexa que a do algoritmo de substituição por deslocamento.

Da mesma forma que o ataque por força não se mostrou tão eficiente quanto era na cifra de deslocamento, o ataque por análise de frequência também é afetado. Como não há substituição de letras, a frequência na qual uma letra aparece não é alterada, mas a frequência em que pares de letras (dígrafos) aparecem é [4]. Isso permite a contagem de pontos (*scores*) entre combinações diferentes de colunas e classificá-las em função dos maiores *scores*, ou seja, em ordem da maior probabilidade (metodologia descrita em [4] e frequências obtidas de [5]). Apesar disso, a acurácia deste algoritmo não é tão favorecida quanto a do seu correspondente na cifra de deslocamento, pois pode haver casos em que uma palavra é terminada em uma coluna e outra é iniciada na outra, mas o dígrafo formado entre as suas letras não é comum e, portanto, a desfavorece na hora de realizar a análise, mesmo sendo uma situação possível de acontecer.

Assim, o código desenvolvido, presente no arquivo *transposition_cipher.c*, pode ser visto abaixo, em que a estrutura definida por *digraphPair* guarda o índice da coluna à esquerda, índice da coluna à direita, e o *score* dessa combinação:

```
/* Frequency Cryptanalysis

Uses the frequency table of digraphs (available in
    ↪ https://www.dcc.fc.up.pt/~rvr/naulas/tabelasPT/):
Choose row as first letter and column as second letter

*/


float DIGRAPH_FREQ[26][26] // DEFINIDO NO ARQUIVO DE
    ↪ transposition\_cipher.c

string freq_transp_cryptoanalysis(string cipher_text, int key_len,
    ↪ double* ui_time){
    double start_time, end_time;
    int len = strlen(cipher_text);                          // Lenght of
        ↪ ciphertext
    int matrix_height = len / key_len;                      // Height of
        ↪ cipher matrix

    // Building a (currently unordered) matrix with given ciphertext
        ↪ --  O(N), where N is the length of the ciphertext
    int paddingN = 0;
    int index = 0;
    char cipher_matrix[matrix_height][key_len];
    for (int rowIndex = 0; rowIndex < matrix_height; rowIndex++){
        for (int columnIndex = 0; columnIndex < key_len;
            ↪ columnIndex++){
            cipher_matrix[rowIndex][columnIndex] = cipher_text[index];
            index++;
            if (cipher_text[index] == '#'){
                paddingN++;
```

```
            }
        }
    }

    // Creating column pairs scores matrix -- O(|K| * (|K| - 1)) =>
       ↪ O(|K|^2)*
    digraphPair digraphScores[key_len][key_len - 1];
    for (int i = 0; i < key_len; i++) {
        for (int j = 0; j < (key_len - 1); j++) {
            digraphScores[i][j].fstColumn = -1;
            digraphScores[i][j].sndColumn = -1;
            digraphScores[i][j].score = 0;
        }
    }

    // Building scores for column pairs based on digraphs frequency --
       ↪ O(|K| * (|K| - 1) * N/|K|) => O(|K| * N)
    int sndIndex, firstLetter, secondLetter;
    for (int i = 0; i < key_len; i++) {
       ↪                                          // Repeats |K| times
        sndIndex = 0;
        for (int j = 0; j < (key_len - 1); j++) {
           ↪                                    // Repeats |K - 1| times
            if (i == sndIndex){
                sndIndex++;
            }
            for (int rowIndex = 0; rowIndex < matrix_height;
               ↪ rowIndex++){     // Repeats matrix_height = N/|K|
               ↪ times
                digraphScores[i][j].fstColumn = i;
                digraphScores[i][j].sndColumn = sndIndex;
                firstLetter = cipher_matrix[rowIndex][i];
                secondLetter = cipher_matrix[rowIndex][sndIndex];

                if (firstLetter != '#' && secondLetter != '#'){
                    digraphScores[i][j].score +=
                       ↪ DIGRAPH_FREQ[(tolower(firstLetter) -
                       ↪ 'a')][(tolower(secondLetter) - 'a')];
                }
            }
            sndIndex++;
        }
    }

    // Bubble sorting each column scores, so that highest scores pairs
       ↪ are chosen first -- O(|K|^3)
    for (int digraphColumn = 0; digraphColumn < key_len;
       ↪ digraphColumn++){                // Repeats |K| times
        // Bubble sort for a single column -- O(|K-1|^2) => O(|K|^2)
        for (int i = 0; i < (key_len - 1); i++){
           ↪                                              // Repeats |K -
           ↪ 1| times
            for (int j = 0; j < i; j++){
               ↪                                                    //
               ↪ Repeats roughly |K - 1| times
                if(digraphScores[digraphColumn][j].score <
```

```
                    ↪ digraphScores[digraphColumn][i].score){
68                  digraphPair temp = digraphScores[digraphColumn][i];
69                  digraphScores[digraphColumn][i] =
                        ↪ digraphScores[digraphColumn][j];
70                  digraphScores[digraphColumn][j] = temp;
71              }
72          }
73      }
74  }
75
76  // Creating a priority list for which column should be the
        ↪ starting one (based on scores) -- O(|K|)
77  int columnPriority[key_len];
78  for (int i = 0; i < key_len; i++){
79      columnPriority[i] = digraphScores[i][0].fstColumn;
80  }
81
82  // Bubble sorts the priority list based on scores -- O(|K|^2)
83  for (int i = 0; i < key_len; i++){
84      for (int j = 0; j < i; j++){
85          if(digraphScores[columnPriority[j]][0].score <
                ↪ digraphScores[columnPriority[i]][0].score){
86              int temp = columnPriority[i];
87              columnPriority[i] = columnPriority[j];
88              columnPriority[j] = temp;
89          }
90      }
91  }
92
93
94  int keyOrder[key_len];          // Stores the order of the
        ↪ columns, which translates to a possible key
95  int currentColumns[key_len];    // Stores which of the (key_len -
        ↪ 1) digraph combinations each column is using
96  int used[key_len];              // 0 if column combination wasn't
        ↪ used yet
97  float currentScores[key_len - 1];
98
99  // Setting up the previously mentioned arrays -- O(|K|)
100 for (int i = 0; i < key_len; i++){
101     keyOrder[i] = -1;
102     currentColumns[i] = 0;
103     used[i] = 0;
104     if (i < key_len - 1){
105         currentScores[i] = 0;
106     }
107 }
108
109
110 string possible_key;
111 string found_text;
112 int startingColumn;
113
114 // Loops through all possible key combinations, based on the
        ↪ highest scores -- O(|K|! * |K|^2 + N), since:
115 // 1 - Tests through all possible combinations (worst case) --
```

```
        ↪ O(|K|!)
116     // 2 - For each combination, tries to decrypt using possible key
            ↪ -- O(|K|^2 + N)
117     for (int scIndex = 0; scIndex < key_len; scIndex++) {
118         startingColumn = columnPriority[scIndex];
119         keyOrder[0] = startingColumn;
120         used[startingColumn] = 1;
121
122         int i = 1; // Index for building keyOrder
123         while (i >= 1) {
124             if (i == key_len) {   // If we have a full permutation,
                    ↪ use keyOrder to decrypt ciphertext=
125                 possible_key = generate_array_transp_key(keyOrder,
                        ↪ key_len); // Generates key from keyOrder array
                        ↪ -- O(|K|)
126                 found_text = dec_transp(cipher_text, possible_key);
                        ↪          // Decrypts using possible key --
                        ↪ O(|K|^2 + N)
127
128                 start_time = get_time_ms();
129                 if
                        ↪ (transp_cipher_ui_freq(possible_key,found_text,currentScores,ke
                        ↪ {
130                     end_time = get_time_ms();
131                     *ui_time += end_time - start_time;
132                     // UI stuff (doesn't impact significantly
                            ↪ complexity)
133                     free(found_text);
134                     return possible_key;            // The key used
135                 }
136                 end_time = get_time_ms();
137                 *ui_time += end_time - start_time;
138
139                 // If possible_key wasn't accepted, backtrack to find
                        ↪ next available option
140                 i--;
141                 used[keyOrder[i]] = 0;   // Resets previous column
                        ↪ (frees from it being used)
142                 currentColumns[i]++;     // Sets to try the next
                        ↪ highest score pair for the previous column
143                 continue;
144             }
145
146             int leftCol = keyOrder[i - 1];
147             int foundNext = 0;
148
149             while (currentColumns[i] < key_len - 1) {      // Loops
                    ↪ until it finds a valid pair or exceeds key_len - 1
                    ↪   -- O(|K| - 1 ) => O(|K|)
150                 int candidateCol =
                        ↪ digraphScores[leftCol][currentColumns[i]].sndColumn;
151                 if (!used[candidateCol]) {                // If
                        ↪ column isn't already in keyOrder
152                     keyOrder[i] = candidateCol;
153                     used[candidateCol] = 1;
154                     currentScores[i - 1] =
```

```
                        ↪ digraphScores[leftCol][currentColumns[i]].score;
155                     if (i < (key_len - 1)){
156                         currentColumns[i + 1] = 0;              //
                                ↪ Resets the next column's pair index, so
                                ↪ that all combinations are tried
157                     }
158                     foundNext = 1;
159                     i++;
160                     break;
161                 }
162
163                 currentColumns[i]++;    // If pair isn't available, try
                        ↪ next pairing
164             }
165
166
167             if (!foundNext) {       // If no pair was found, it means
                    ↪ there are no more possible combinations with this
                    ↪ order, so backtrack
168                 if (i > 0) {  // If i hasn't reached the starting
                        ↪ column, continue backtracking
169                     i--;
170                     used[keyOrder[i]] = 0;    // Resets previous column
                            ↪ (frees from it being used)
171                     currentColumns[i]++;      // Sets to try the next
                            ↪ highest score pair for the previous column
172
173                 } else {    // Otherwise, change the starting column by
                        ↪ breaking from this loop (every combination was
                        ↪ already explored)
174                     break;
175                 }
176             }
177         }
178
179         // Reset for next startingColumn -- O(|K|)
180         for (int j = 0; j < key_len; j++) {
181             used[j] = 0;                    // Sets every column to unused
182             currentColumns[j] = 0;          // Resets every combination,
                    ↪ to that algorithm gets the highest score pair
183             keyOrder[j] = -1;               // Sets keyOrder to an invalid
                    ↪ state
184         }
185     }
186
187     // Error (NO FOUND TEXT)
188     free(found_text);
189     free(possible_key);
190     return NULL;
191 }
```

Código 8: Algoritmo de Análise de frequência com Cifra de Transposição Colunar, presente no arquivo *transposition_cipher.c*

Assim como na análise de frequência da cifra de deslocamento, esse algoritmo também pode ser considerado uma força bruta com heurística, pois leva em considera-

ção o *score* das combinações entre as colunas. A complexidade temporal, porém, é de $O(|K|! \times (|K|^2 + N))$, a mesma da força bruta.

```
TRANSPOSITION CIPHER (COLUMNAR): Cryptanalysis Mode.
Description: Write your plaintext. A key will be set pseudo-randomly.
Choose a method for performing the cryptanalysis afterwards.
NOTE: Resulting plaintexts and ciphertexts won't have any spaces nor
   punctuation. Non ASCII characters won't be processed (so reults
   may be underwhelming).

Write your plain_text: Eu sou a mensagem, o texto em claro M!
How long is your key? Input a number between 2 and 26: 5
A key has been chosen // COMENTÁRIO1: chave escolhida
   pseudo-aleatoreamente
--------------------------------------------------------------------------
TRANSPOSITION CIPHER (COLUMNAR): Cryptanalysis Mode.
Choose method of cryptanalysis:
| 1. Brute Force
| 2. Frequency Analysis
| 3. Return

Your current ciphertext: onmtl#Eaaterusooa#umgemoseexcM

Type the number to select your answer: 2 // COMENTÁRIO2: escolha da
   criptoanálise por análise de frequência
--------------------------------------------------------------------------
//COMENTÁRIO3: primeira iteração (as outras serão puladas para não
   tomar muito espaço, mas seguem esse padrão)
TRANSPOSITION CIPHER (COLUMNAR): Cryptanalysis Mode (Frequency
   Analysis)
- Key used: ebadc.
- Scores:
  -> 57.760006, for the column pair of indexes 4 ('e' in this key) and
     1 ('b' in this key)
  -> 38.280003, for the column pair of indexes 1 ('b' in this key) and
     0 ('a' in this key)
  -> 36.619995, for the column pair of indexes 0 ('a' in this key) and
     3 ('d' in this key)
  -> 15.559999, for the column pair of indexes 3 ('d' in this key) and
     2 ('c' in this key)
Possible message: sEouueanmseamgoxtteocelmaMro

Choose an option:
| 1. Continue, this isn't the message I want.
| 2. Return, this is the message I'm looking for.

Type the number to select your answer: 1
--------------------------------------------------------------------------
//COMENTÁRIO4: ao encontrar a chave, usuário seleciona a opção 2
   (ocorreu na iteração 37)
TRANSPOSITION CIPHER (COLUMNAR): Cryptanalysis Mode (Frequency
   Analysis)
- Key used: bdeac.
- Scores:
  -> 33.669998, for the column pair of indexes 1 ('b' in this key) and
     3 ('d' in this key)
```

```
  -> 34.810001, for the column pair of indexes 3 ('d' in this key) and
     4 ('e' in this key)
  -> 23.450001, for the column pair of indexes 4 ('e' in this key) and
     0 ('a' in this key)
  -> 12.619999, for the column pair of indexes 0 ('a' in this key) and
     2 ('c' in this key)
Possible message: EusouamensagemotextoemclaroM

Choose an option:
| 1. Continue, this isn't the message I want.
| 2. Return, this is the message I'm looking for.

Type the number to select your answer: 2
------------------------------------------------------------------------
//COMENTÁRIO5: resultado final: a chave éretornada e o texto é
   decriptado usando esta chave. O tempo total gasto na criptoanálise
   é␣mostrado, ignorando o tempo em que o usuário fazia a escolha.
TRANSPOSITION CIPHER (COLUMNAR): Cryptanalysis Mode (Frequency
   Analysis)
Resulting Ciphertext: onmtl#Eaaterusooa#umgemoseexcM
Key found through cryptanalysis: bdeac
True key (NOT USED FOR DECIPHERING, ONLY SHOWN FOR COMPARISON): hvzdl
Message deciphered through cryptanalysis: EusouamensagemotextoemclaroM
- Time elapsed during cryptanalysis (Ignoring time spent on UI
   prompting): 0.219800 ms

Press ENTER key to return.
```

Terminal 9: Exemplo de Criptoanálise por Análise de Frequência com Cifra de Transposição Colunar

Ao comparar o exemplo acima, verifica-se que a heurística proporcionou uma economia de tempo na criptoanálise (demorando, apenas 37 iterações, comparado às 59 iterações da força bruta). Apesar disso, por se tratar e uma heurística fraca, podem haver casos em que as escolhas do algoritmo o torne próximo de uma força bruta.

4 Conclusão

Por meio desse exercício, foi possível perceber a diferença entre cifras básicas de substituição e transposição e como as suas complexidades e suas escolhas de chaves podem afetar a complexidade da criptoanálise. O código desenvolvido permitiu a confirmação teórica e a sua execução está explicada no apêndice.

Referências

[1] SCHNEIDER, J. *História da criptografia*, 2024. Disponível em: <https://www.ibm.com/br-pt/think/topics/cryptography-history>. Acesso em: 19 de abril de 2025.

[2] ANDERSON, R. Security engineering : a guide to building dependable distributed systems. Indianapolis, Indiana: John Wiley & Sons, Inc., 2020.

[3] CHRISTENSEN, C. *Frequency Analysis*, 2006. Disponível em: <https://websites.nku.edu/ christensen/section%203%20frequency%20analysis.pdf>. Acesso em: 19 de abril de 2025.

[4] CHRISTENSEN, C. *Columnar Transposition*, 2015. Disponível em: <https://websites.nku.edu/ christensen/1402%20Columnar%20transposition.pdf>. Acesso em: 17 de abril de 2025.

[5] REIS ROGÉRIO *Tabelas da língua portuguesa*. Disponível em: <https://www.dcc.fc.up.pt/%7Ervr/naulas/tabelasPT/>. Acesso em: 30 de março de 2025.

A Código do Exercício

O código foi desenvolvido e testado em windows, pode haver erros inesperados em unix, mas em tese não é para haver. O código está no mesmo diretório deste documento. Para compilar, basta usar o comando `make main` e executar o arquivo gerado.

- Link para repositório: https://github.com/Luke0133/Exercicio01-SegurancaComputacional.