

# Lista de Exercícios 01

## Segurança Computacional: Cifras de César e Transposição

João Carlos Gonçalves de Oliveira Filho, 232009511

<sup>1</sup>Departamento de Ciência da Computação – Universidade de Brasília (UnB)  
CIC0201 - Segurança Computacional

232009511@aluno.unb.br

**Resumo.** Este relatório apresenta a implementação e análise das cifras de César (shift cipher) e de Transposição, abordando técnicas de criptoanálise como ataques de força bruta e análise de distribuição de frequência de letras, digrafos e trigrafos da língua portuguesa.

### 1. Introdução

Todos os códigos dos algoritmos implementados estão disponíveis no seguinte repositório do GitHub:

<https://github.com/ogjoaoc/lista-1-segcomp-unb>

Esta é a primeira lista de exercícios da disciplina de Segurança Computacional, cujo foco é o estudo da criptografia clássica. Nesta lista, foram exploradas seguintes técnicas criptográficas fundamentais:

- **Cifra de César:** Cifra por substituição com deslocamento fixo.
- **Cifra de Transposição:** Reorganização da ordem das letras de forma matricial.

Para cada cifra, foram implementados algoritmos para cifração/decifração e dois métodos de ataque (força bruta e análise de distribuição de frequência).

### 2. Especificações

#### 2.1. Implementação

Foram desenvolvidos em C++ os seguintes módulos:

- cifra\_cesar.cpp - Módulos principais para cifra por deslocamento.
- cifra\_transposicao.cpp - Módulos principais para cifra por transposição.
- menu\_cifra\_cesar.cpp - Interface de menu para Cifra de César.
- menu\_cifra\_transposicao.cpp - Interface de menu para Cifra de Transposição.

#### 2.2. Execução

Compilação: `g++ menu_cifra_cesar.cpp -o run_cesar.out`

Execução: `./run_cesar.out`

#### 2.3. Testes

- Textos em português sem acentos, com ou sem pontuação.
- Tabelas de frequência de letras/digrafos/trigrafos.
- input.txt - Arquivo de entrada utilizado no algoritmo da cifra por deslocamento.
- input2.txt - Arquivo de entrada utilizado no algoritmo da cifra por transposição.

### 3. Cifra de César

#### 3.1. Funcionamento

A implementação da Cifra de César foi desenvolvida com a ideia de substituir cada caractere com base em um deslocamento de tamanho fixo. Segue abaixo o código da função de encriptação. As outras funções principais realizam:

- Decodificação: Aplicação do deslocamento inverso.
- Ataque por força bruta: Testa todas as chaves de tamanho  $K$ .
- Ataque por análise de distribuição de frequências: Testa as diferentes frequências de letras com chaves alternativas.

```
/**
 * @brief codifica um texto usando cifra de cesar
 * @param mensagem texto original
 * @param CHAVE chave de deslocamento
 * @return texto cifrado
 */
string codificaCifraCesar(string &mensagem, int CHAVE) {
    string mensagem_criptografada = mensagem;
    for(char &c : mensagem_criptografada) {
        if(isalpha(c)) {
            char base = islower(c) ? 'a' : 'A';
            c = (c - base + CHAVE) % 26 + base;
        }
    }
    return mensagem_criptografada;
}
```

Figure 1. Implementação da encriptação - Cifra de César

O algoritmo segue as equações básicas:

$$E(c) = (c + K) \mod 26 \quad (\text{Encriptação}) \quad (1)$$

$$D(c) = (c - K + 26) \mod 26 \quad (\text{Decriptação}) \quad (2)$$

Em que  $c$  representa o caractere original e  $K$  é a chave de deslocamento.

Durante o processo de encriptação/decriptação básico, o algoritmo realiza  $N$  iterações, sendo  $N$  o tamanho do texto. Há operações de atribuição e deslocamento dos caracteres no processo, mas no pior caso, a complexidade de tempo e espaço do algoritmo é  $O(N)$ . Esse algoritmo pode ser aplicado em textos curtos, ou para fins educacionais, já que atualmente ele não garante tanta segurança.

### 3.1.1. Ataque por Força Bruta

Nesse viés, a abordagem de ataque por força bruta utiliza o mesmo processo da decifração simples, porém realizando  $K$  iterações, sendo  $K$  o tamanho máximo do alfabeto, para que assim seja possível testar todos os possíveis deslocamentos, garantindo ao final uma complexidade de  $O(26 * N)$ .

- **Vantagens:**
  - Simplicidade de implementação.
  - Garantia de encontrar a solução no espaço de  $K$  chaves.
  - Eficaz para textos curtos (até  $10^6$  caracteres) com um alfabeto razoável.
- **Limitações:**
  - Inviável para cifras com espaço de chaves grande.
  - Requer avaliação manual dos resultados para textos curtos.

### 3.1.2. Análise de Distribuição de Frequência

Na abordagem por análise da distribuição de frequência, é preciso calcular com base no texto cifrado, a frequência de cada letra. Em seguida, 26 iterações são feitas para testar todos os deslocamentos possíveis. Para cada deslocamento, é gerado um erro total, e a chave escolhida é a que possui o menor erro absoluto. Com essa estratégia, a complexidade de tempo fica  $O(26^2 + N)$ , já que iteramos por todos os possíveis deslocamentos ( $K$ ), e para cada deslocamento, contabilizamos o erro com base na frequência de cada letra do alfabeto.

- **Vantagens:**
  - Mais rápida para aplicações com textos longos ( $> 100$  caracteres).
  - Não requer avaliação manual.
- **Limitações:**
  - Precisão reduzida para textos curtos.
  - Depende da similaridade do texto com a distribuição padrão.

**Table 1. Desempenho comparativo dos métodos de ataque**

Método	Complexidade	Melhor Caso
Força Bruta	$O(26N)$	Textos curtos
Análise Freq.	$O(26^2 + N)$	Textos longos
Combinado	$O(26N + 26^2)$	Todos os casos

## 4. Cifra de Transposição

### 4.1. Funcionamento

A Cifra de Transposição colunar reorganiza os caracteres do texto original em uma matriz, permutando as colunas de acordo com uma chave. Para fins de otimização e simplicidade, a chave recomendada possui tamanho  $< 9$  e não possui letras repetidas. A Figura 2 mostra a implementação da função de codificação.

```
/**
 * @brief codifica uma mensagem usando cifra de transposicao.
 * @param mensagem mensagem original (plaintext).
 * @param chave chave para cifragem.
 * @return mensagem criptografada (ciphertext).
 */
string codificaTransposicao(string mensagem, string chave) {
    int linhas = (mensagem.length() + chave.length() - 1) / chave.length();
    int colunas = chave.size();
    vector<vector<char>> matriz(linhas, vector<char> (colunas));
    int indice = 0;
    for(int i = 0; i < linhas; i++) {
        for(int j = 0; j < colunas; j++) {
            if(indice < mensagem.length()) {
                matriz[i][j] = mensagem[indice++];
            } else {
                matriz[i][j] = 'X';
            }
        }
    }
    string mensagem_codificada = "";
    vector<string> auxiliar;
    for(int j = 0; j < colunas; j++) {
        string aux = "";
        for(int i = 0; i < linhas; i++) {
            aux += matriz[i][j];
        }
        auxiliar.push_back(aux);
    }
    vector<pair<char, int>> permutacao = geraPermutacao(chave);
    for(int i = 0; i < colunas; i++) {
        mensagem_codificada += auxiliar[permutacao[i].second - 1];
    }
    return mensagem_codificada;
}
```

Figure 2. Implementação da codificação - Cifra de Transposição

O processo de encriptação se baseia na composição de uma matriz de  $K$  colunas, sendo  $K$  o tamanho da chave. Essa matriz terá  $\lceil N/K \rceil$  linhas, com cada posição sendo preenchida com um caracter do plaintext, no sentido horizontal. Em seguida, a mensagem cifrada é composta, a partir da concatenação da  $i$ -ésima coluna, sendo  $i$  o índice da permutação alfabética gerada pela chave. Essa estratégia garante uma complexidade de  $O(K * \lceil N/K \rceil + N)$ , já que ao final da composição da matriz com os caracteres do texto, é necessário reconstruir a mensagem cifrada de tamanho  $N$  com base na permutação. Na implementação utilizada, ainda há um adicional de  $K * \log K$  pela ordenação da chave. A complexidade final de tempo é  $O(K * \lceil N/K \rceil + N + K * \log K)$  e  $O(K * \lceil N/K \rceil)$  de espaço, garantindo  $O(N)$  no pior caso para tempo e espaço.

## 4.2. Ataque por Força Bruta

A abordagem de força bruta testa todas as possíveis permutações das colunas da matriz de cifração. Para uma chave de tamanho  $K$ , o espaço de busca contém  $K!$  permutações, resultando em complexidade computacional de  $\mathcal{O}(K! \times N)$ , onde  $N$  é o tamanho do texto cifrado. Nesse processo, é extremamente ineficiente explorar chaves com  $K > 9$ .

- **Vantagens:**

- Implementação direta e determinística.
- Garantia de encontrar a solução (para  $K!$  pequeno).
- Eficiente para chaves curtas.
- Independe do idioma ou conteúdo do texto original (não realiza substituições).

- **Limitações:**

- Complexidade fatorial  $\mathcal{O}(K! \times N)$
- Rapidamente inviável para  $K \geq 9$  ( $9! = 362880$  permutações)
- Requer verificação manual.
- Sensível a erros de padding.

### 4.2.1. Análise de Distribuição de Frequência

Na abordagem implementada por análise da distribuição de frequência, é preciso iterar por todas as  $K!$  possíveis permutações. Para cada permutação, é feita a decifração padrão, como também a contagem de dígrafos e trígrafos presentes no texto decifrado. A partir dessa contagem, é calculado um score para a  $i$ -ésima decifração. Ao final, é gerado um vetor de pares, que contém o texto decifrado, o score e a permutação utilizada. A decifração de maior score é escolhida como "ótima". O cálculo da complexidade é similar ao ataque de força bruta, com um adicional de  $K! \times \log K!$  para ordenação de todas as decifrações com base no score. Não consigo afirmar que essa era a abordagem mais eficiente.

- **Vantagens:**

- Automatizável via scores de dígrafos e trígrafos.
- Eficiente para chaves médias ( $7 \leq K \leq 10$ ).

- **Limitações:**

- Requer memória extra para armazenar dados de  $n$ -grafos do idioma.
- Sensível a textos especiais (técnicos, fórmulas).
- Precisão depende do tamanho mínimo do texto ( $> 3K$  caracteres).

**Table 2. Desempenho dos métodos de ataque**

Método	Complexidade	Melhor Caso
Força Bruta	$\mathcal{O}(K! \times N)$	Chaves pequenas
Análise Freq.	$\mathcal{O}(K! \times N + K! \times \log K)$	Textos longos
Combinado	$\mathcal{O}((K! \times (N + \log K)) \times N)$	N/A

## 5. Conclusão

É válido concluir que em ambas as cifras, a combinação das técnicas que utilizam força bruta e análise estatística influenciam positivamente na quebra, aumentando a eficiência e consequentemente reduzindo o custo computacional. Contudo, na cifra por transposição, ambas as estratégias se tornam inviáveis para uma chave extensa, já que a complexidade fatorial exige um grande custo computacional. Além disso, conforme os testes de cada algoritmo, o comprimento do texto a ser encriptado/decriptado tem grande impacto na eficiência.

## References

- [1] SHANNON, C. E. *Communication Theory of Secrecy Systems*.
- [2] LORENA, *Aula 04: Algoritmos das Cifras*. Prof<sup>a</sup>. Ma. Lorena S. B. Borges.