SERVIÇO PÚBLICO FEDERAL - MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE VIÇOSA - UFV
CAMPUS FLORESTAL

Trabalho prático 3 - Projeto e análise de algoritmos.

Bruno Vicentini Ribeiro - 5907
Erich Pinheiro Amaral - 5915
Fabrício Henrique Viana Albino - 5925
Pedro Paulo Paz do Nascimento - 5937
Vitor Mathias Andrade - 5901

Florestal - MG
2025

Sumário

Introdução

O presente trabalho aborda a implementação de um sistema interativo em linguagem C voltado para a simulação de um processo de criptoanálise baseado em substituição alfabética. A proposta integra conteúdos estudados na disciplina de Projeto e Análise de Algoritmos, permitindo aplicar conceitos fundamentais relacionados à análise de frequência, técnicas de casamento exato e aproximado de padrões e manipulação incremental de chaves.

O problema é contextualizado por meio da narrativa das profecias dos “Herdeiros de Chrysos”, na qual o grupo recebe um texto claro que deve ser inicialmente criptografado utilizando uma cifra de deslocamento. A partir do texto cifrado produzido, o sistema desenvolvido tem como objetivo auxiliar o usuário na reconstrução da chave e na recuperação do texto original, oferecendo recursos como visualização do estado atual da decifragem, sugestões baseadas em frequências, buscas por padrões no texto cifrado ou parcialmente decifrado e ajustes manuais da chave.

A implementação proposta permite consolidar, de forma prática, o entendimento dos algoritmos explorados ao longo da disciplina, entre eles, KMP, Shift-And e sua variante aproximada, além de reforçar a importância do uso adequado de estruturas de dados, organização modular e interação com arquivos. Assim, o trabalho se fundamenta na união entre narrativa, técnicas clássicas de criptoanálise e ferramentas de análise de algoritmos, tendo como objetivo final a reconstrução completa da chave de substituição aplicada ao texto.

Organização

Na figura 1, observa-se o panorama geral da organização do projeto, no qual estão elencados os arquivos necessários para o trabalho. Esses arquivos foram separados em pastas para garantir melhor organização e funcionamento do código.

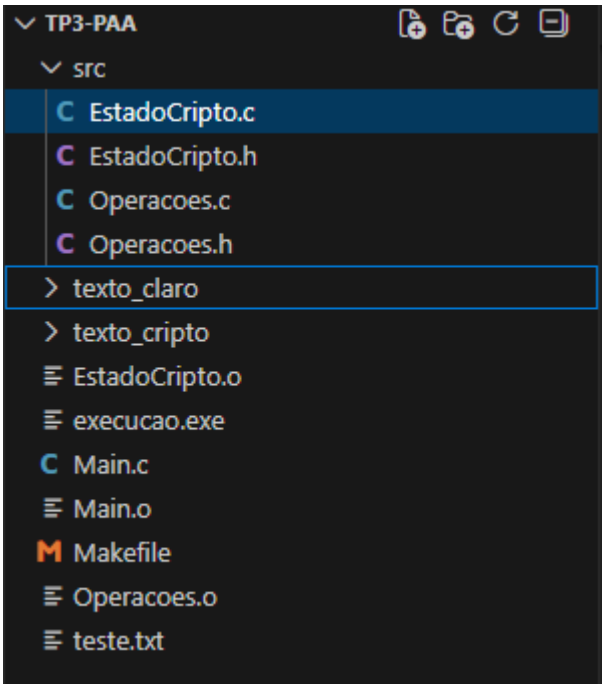


Figura 1: Repositório do projeto.

Desenvolvimento

- Estrutura de dados:

Para que seja possível executar todas as funções previstas, foi necessário criar duas estruturas de dados: “EstadoCripto” (figura 2), e “Frequencia” (figura 3).

```
typedef struct {
    char alfabetoOriginal[TAMANHO_ALFABETO + 1];
    char chaveDecifracao[TAMANHO_ALFABETO + 1];
    char textoCifrado[TAMANHO_MAX_TEXTO];
    char textoParcial[TAMANHO_MAX_TEXTO];
    int shiftCriptografia;
} EstadoCripto;
```

Figura 2: Estrutura “EstadoCripto”.

A estrutura “EstadoCripto” armazena um vetor com o alfabeto original, um vetor com a chave para decifrar, um vetor com o texto decifrado e o texto parcialmente decifrado, além de armazenar qual foi o valor do “shift” de deslocamento.

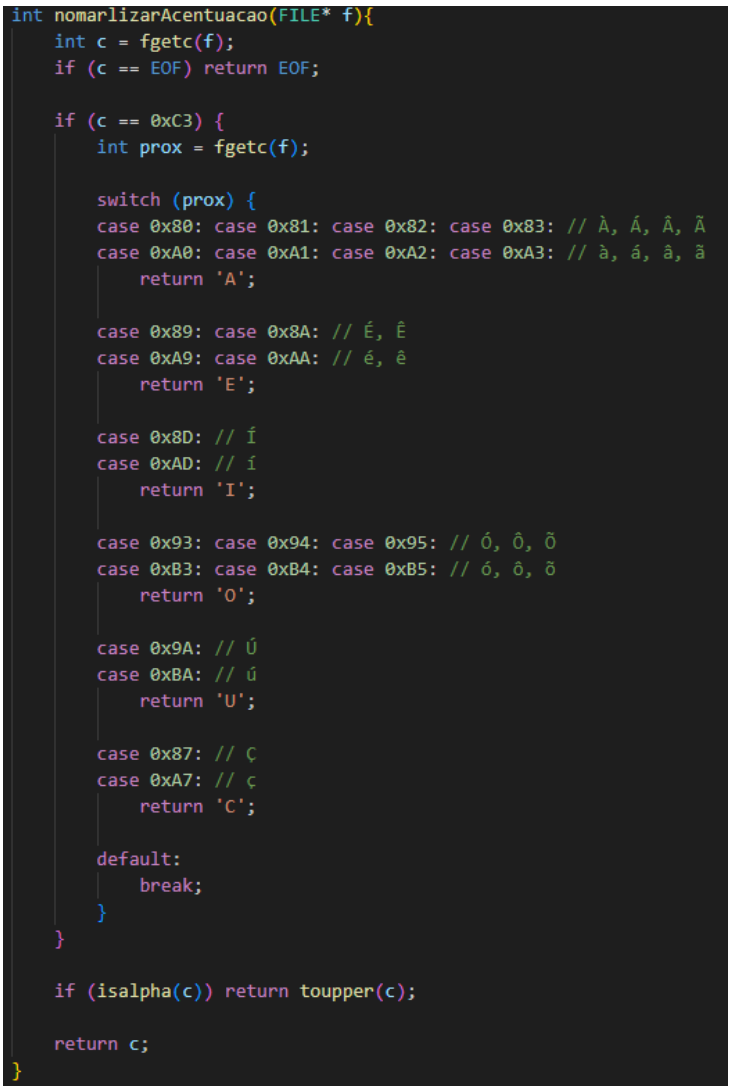
```
typedef struct {
    char letra;
    int quantidade;
    double frequencia;
}Frequencia;
```

Figura 3: Estrutura “Frequencia”.

A estrutura “Frequencia” armazena a letra que será avaliada a frequência, a quantidade de vezes que ela aparece, além do valor que será exibido em porcentagem.

- Inicialização e normalização do texto:

A primeira etapa do sistema consiste na preparação do ambiente de análise. Ao carregar um arquivo de texto claro, o programa não apenas realiza a leitura, mas aplica um processo de normalização. Caracteres acentuados (como “á”, “õ” e “ç”) são convertidos para seus equivalentes não acentuados e todo o texto é transformado para letras maiúsculas. Isso garante que a criptografia subsequente opere sobre um alfabeto padrão de 26 caracteres, simplificando a análise estatística sem perda de generalidade.



```c
int nomarlizarAcentuacao(FILE* f){
    int c = fgetc(f);
    if (c == EOF) return EOF;

    if (c == 0xC3) {
        int prox = fgetc(f);

        switch (prox) {
        case 0x80: case 0x81: case 0x82: case 0x83: // À, Á, Â, Ã
        case 0xA0: case 0xA1: case 0xA2: case 0xA3: // à, á, â, ã
            return 'A';

        case 0x89: case 0x8A: // É, Ê
        case 0xA9: case 0xAA: // é, ê
            return 'E';

        case 0x8D: // Í
        case 0xAD: // í
            return 'I';

        case 0x93: case 0x94: case 0x95: // Ó, Ô, Õ
        case 0xB3: case 0xB4: case 0xB5: // ó, ô, õ
            return 'O';

        case 0x9A: // Ú
        case 0xBA: // ú
            return 'U';

        case 0x87: // Ç
        case 0xA7: // ç
            return 'C';

        default:
            break;
        }
    }

    if (isalpha(c)) return toupper(c);

    return c;
}
```

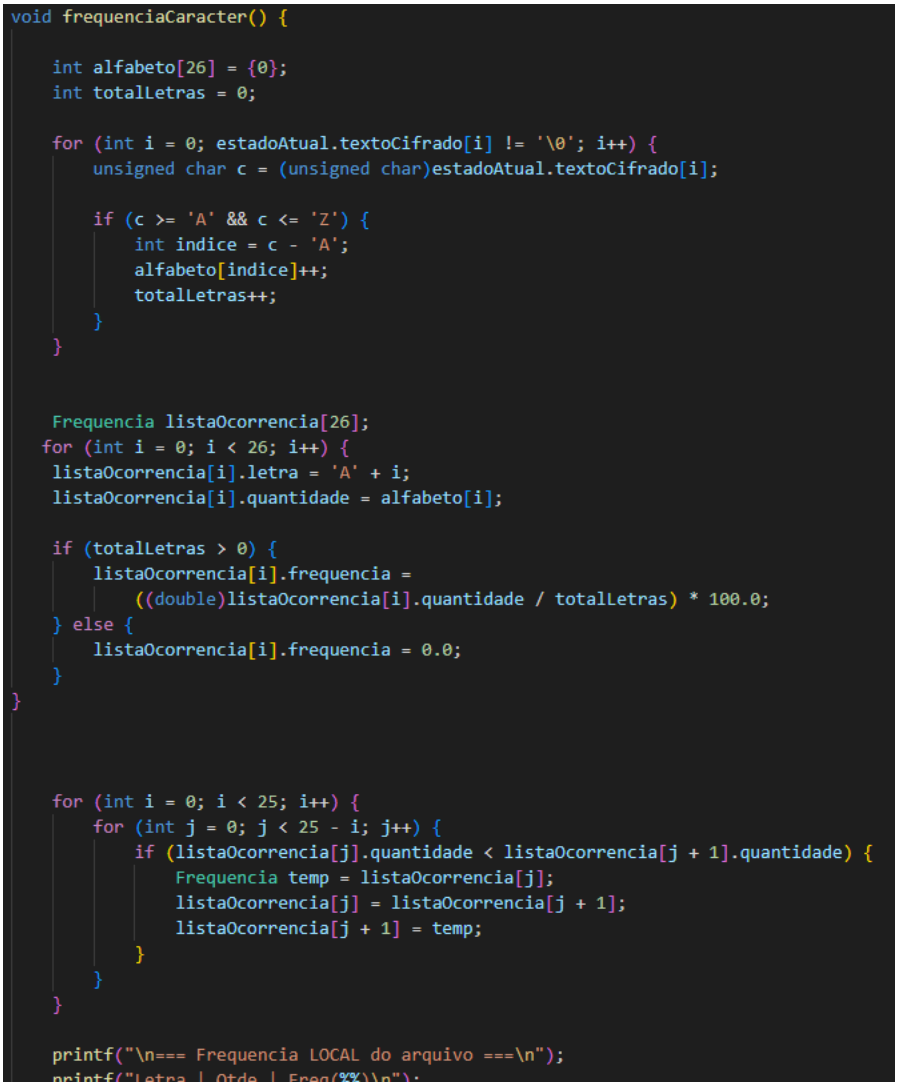
Figura 4: Normalização do texto.

- Análise de frequência:

Uma das principais ferramentas implementadas para quebrar a cifra é a análise de frequência. O algoritmo conta a incidência de cada caractere no texto cifrado local e, para aumentar a precisão, realiza também uma contagem global considerando todos os 12 arquivos de profecia disponíveis, assumindo que todos foram cifrados sob a mesma lógica.

Os dados coletados são comparados com a tabela de frequência padrão da língua portuguesa. O sistema então sugere um "chute" inicial de mapeamento, associando as letras mais frequentes do texto cifrado (como “A”, “E” e “O”) às letras mais comuns do idioma, oferecendo ao usuário um ponto de partida estatisticamente embasado para a decifração.

A função que é responsável por fazer a análise da frequência e o “chute” de mapeamento é a “frequenciaCaracter” (figura 5).



```
void frequenciaCaracter() {

    int alfabeto[26] = {0};
    int totalLetras = 0;

    for (int i = 0; estadoAtual.textoCifrado[i] != '\0'; i++) {
        unsigned char c = (unsigned char)estadoAtual.textoCifrado[i];

        if (c >= 'A' && c <= 'Z') {
            int indice = c - 'A';
            alfabeto[indice]++;
            totalLetras++;
        }
    }


    Frequencia listaOcorrencia[26];
   for (int i = 0; i < 26; i++) {
    listaOcorrencia[i].letra = 'A' + i;
    listaOcorrencia[i].quantidade = alfabeto[i];

    if (totalLetras > 0) {
        listaOcorrencia[i].frequencia =
            ((double)listaOcorrencia[i].quantidade / totalLetras) * 100.0;
    } else {
        listaOcorrencia[i].frequencia = 0.0;
    }
}




    for (int i = 0; i < 25; i++) {
        for (int j = 0; j < 25 - i; j++) {
            if (listaOcorrencia[j].quantidade < listaOcorrencia[j + 1].quantidade) {
                Frequencia temp = listaOcorrencia[j];
                listaOcorrencia[j] = listaOcorrencia[j + 1];
                listaOcorrencia[j + 1] = temp;
            }
        }
    }

    printf("\n=== Frequencia LOCAL do arquivo ===\n");
    printf("Letra | Qtde | Freq(%%)\n");
```

Figura 5: Função “frequenciaCaracter”.

- Casamento:

Para refinar a chave sugerida pela frequência, foram implementados algoritmos eficientes de busca textual, permitindo ao usuário encontrar palavras ou fragmentos suspeitos no texto cifrado:

  - Casamento exato (Shift-And): para buscas precisas, utilizou-se o algoritmo Shift-And. Este método tira proveito das operações paralelismo de bits do processador. O padrão de busca é pré-processado em máscaras de bits, e o estado da busca é mantido em uma variável inteira que é atualizada a cada caractere lido. Isso permite verificar a ocorrência do padrão com extrema rapidez, essencialmente em tempo linear (figura 6).

```c
void casamentoExato(char* padrao) {
    //Algoritmo Shift-And
    int m = strlen(padrao);
    int n = strlen(estadoAtual.textoCifrado);

    if (m > 64) {
        printf("Erro: O padrão e muito longo para o algoritmo Shift-And (max 64 caracteres).\n");
        return;
    }

    unsigned long M[256]; //Tabela M

    for (int i = 0; i < 256; i++) {
        M[i] = 0;
    }
    for (int j = 0; j < m; j++) {
        unsigned char c = (unsigned char)padrao[j];
        M[c] |= (1UL << j);
    }

    unsigned long R = 0;
    int ocorrencias = 0;

    for (int i = 0; i < n; i++) {
        unsigned char c = (unsigned char)estadoAtual.textoParcial[i];

        R = ((R << 1) | 1UL) & M[c];

        if (R & (1UL << (m - 1))) ocorrencias++;
    }

    printf("\nOcorrencias: %d\n", ocorrencias);
}
```

Figura 6: Algoritmo de casamento exato (Shift-And).

- Casamento aproximado: reconhecendo que o texto pode estar apenas parcialmente decifrado ou conter erros, implementou-se uma variação do Shift-And capaz de tolerar erros de substituição. O algoritmo mantém múltiplos vetores de estado, onde cada vetor R[j] representa o casamento do padrão com até “j” erros. Dessa forma, é possível localizar palavras que "quase" correspondem ao padrão desejado, facilitando a identificação de palavras onde apenas algumas letras ainda estão cifradas incorretamente (figura 7). Além disso, após a inserção do padrão e da quantidade de erros tolerados, são exibidas as posições de cada ocorrência e a quantidade total de ocorrências.

```c
void casamentoAproximado(char* padrao, int k) {
    int m = strlen(padrao); // é o tamanho do padrão
    int n = strlen(estadoAtual.textoParcial); // é o tamanho do texto decifrado

    if (m > 64) {
        printf("Erro: O padrao e muito longo (max 64 caracteres).\n");
        return;
    }
    char* textoDecifradoTemp = (char*) malloc((n + 1) * sizeof(char));
    if (!textoDecifradoTemp) return;

    for (int i = 0; i < n; i++) {
        char c = estadoAtual.textoParcial[i];
        if (isalpha(c)) {
            int indice = c - 'A';
            if (estadoAtual.chaveDecifracao[indice] != '\0') {
                textoDecifradoTemp[i] = estadoAtual.chaveDecifracao[indice];
            } else {
                textoDecifradoTemp[i] = c;
            }
        } else {
            textoDecifradoTemp[i] = c;
        }
    }
    textoDecifradoTemp[n] = '\0';

    unsigned long M[256];
    for (int i = 0; i < 256; i++) M[i] = 0;

    for (int j = 0; j < m; j++) {
        unsigned char c = (unsigned char)padrao[j];
        M[c] |= (1UL << j);
    }
    unsigned long R[k + 1];
    unsigned long R_antigo[k + 1];

    for(int j=0; j <= k; j++) R[j] = 0;

    int ocorrencias = 0;

    printf("\n--- Ocorrencias Aproximadas (Erro max: %d) ---\n", k);
    for (int i = 0; i < n; i++) {
        unsigned char c = (unsigned char)textoDecifradoTemp[i];

        for(int j=0; j <= k; j++) R_antigo[j] = R[j];
```

Figura 7: Algoritmo de casamento aproximado (Shift-And aproximado).

- Gerenciamento dinâmico de chave:

A interação com o usuário é centralizada na manipulação da chave de criptografia. A função de alteração de chave foi desenvolvida para garantir a integridade do mapeamento: ela assegura que cada letra original seja mapeada por apenas uma letra cifrada (relação 1:1). Caso o usuário insira um mapeamento conflitante, o sistema remove automaticamente a associação anterior, prevenindo inconsistências na decifração final (figura 8).

```c
void alterarChave(char charCifrado, char charOriginal) {
    charCifrado = toupper(charCifrado);
    charOriginal = toupper(charOriginal);

    if (charCifrado >= 'A' && charCifrado <= 'Z' &&
        charOriginal >= 'A' && charOriginal <= 'Z')
    {
        // Remove mapeamento antigo do caractere ORIGINAL, se existir
        for (int i = 0; i < TAMANHO_ALFABETO; i++) {
            if (estadoAtual.chaveDecifracao[i] == charOriginal) {
                estadoAtual.chaveDecifracao[i] = '\0';
                printf("Aviso: Mapeamento antigo %c -> %c removido.\n", 'A' + i, charOriginal);
            }
        }

        // Verifica se a letra cifrada já estava mapeada (sobrescreve)
        if (estadoAtual.chaveDecifracao[charCifrado - 'A'] != '\0') {
            printf("Aviso: O caractere cifrado '%c' já estava mapeado. Mapeamento sobrescrito.\n", charCifrado);
        }

        // Aplica o novo mapeamento (Cifrado -> Original)
        estadoAtual.chaveDecifracao[charCifrado - 'A'] = charOriginal;
        printf("Registrado: %c (cifrado) -> %c (original)\n", charCifrado, charOriginal);
    } else {
        printf("Erro: Caracteres inválidos. Use apenas letras de A-Z.\n");
    }
}
```

Figura 8: Alteração de mapeamento de chave.

O resultado desse processo interativo é visualizado em tempo real, com o texto parcialmente decifrado sendo exibido com destaque visual para facilitar a distinção entre caracteres cifrados e decifrados (figura 9).

```
 === Chave ===
ABCDEFGHIJKLMNOPQRSTUVWXYZ
F

=== Texto Parcialmente Decifrado ===
K XKOMQF ZW FLEBWJEW, KJZF K YKJDFYEIFJPK BHKNFOYF F KO BEHKOKBKO JWOYFI. JK FJPWJPK, WMQE FOPW WJWTWCKNWO, K XHWOBFIK, K DFNZFENK ZF YDNUOKO MQF ZFOWBEW W YDWIW ZW NWVWK: RKYF NFWHIFJPF WXNWYWNEW W EJBWIEW F ZFOWBEWNEW
W LNKBFYEW, YNWRWJZK KO FOLEJDKO ZW ZQREZW JW WNRKNF OWCNWZW ZW OWXFZKNEW?

"PQ PNWJOYFJZFNWO KO LEYKO ZW LQNFVW F NFPKNJWNWO W YKNNQLYWK F W WZRFNOEZWZF."
```

Figura 9: Distinção entre caracteres decifrados.

Compilação e execução

Para facilitar a compilação e execução do programa, foi utilizado um arquivo Makefile. Para compilar, utiliza-se o seguinte comando: `make`

Após isso, o programa é executado seguindo o seguinte formato: `.\execucao.exe`

Em seguida, é necessário informar ao programa o nome do arquivo com o texto claro e o nome do arquivo que será gerado com a criptografia. É válido destacar que o arquivo do texto claro está dentro da pasta “texto_claro” e o texto criptografado está dentro da pasta “texto_cripto”. Logo, ao informar o nome do arquivo, deve-se escrever na forma: “texto_claro/Anaxa.txt”.

```
--- Trabalho Pratico 3 - Criptoanalise ---
Informe o nome do arquivo de texto claro a ser lido (ex: texto_claro/Anaxa.txt): texto_claro/Anaxa.txt
Informe o nome do arquivo para escrever o texto criptografado (ex: texto_cripto/Anaxa.txt): texto_cripto/Anaxa.txt
```

Figura 10: Inserção do nome dos arquivos.

Resultados

Os testes realizados demonstraram a eficácia das ferramentas de criptoanálise implementadas. A seguir, apresentamos os resultados obtidos em cada etapa crítica do processo de decifração.

- Cifragem e estado inicial:

Logo após a leitura do arquivo, o sistema aplicou corretamente a Cifra de Deslocamento com um shift aleatório, gerando um texto criptografado. A interface inicial (opção 1) apresentou o texto cifrado e a chave vazia, confirmando que o sistema estava pronto para a análise sem revelar a resposta prematuramente (figura 11).

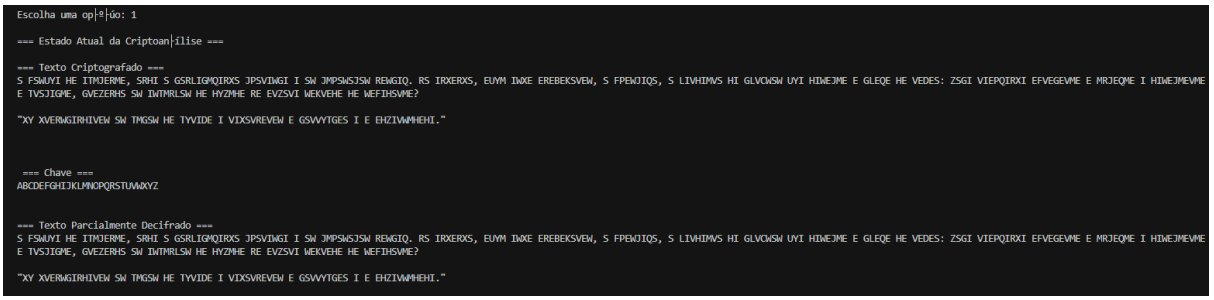


Figura 11: Opção 1 em estado inicial.

- Análise de frequência:

Ao selecionar a opção 2, o algoritmo calcula a frequência dos caracteres no texto cifrado e compara com a tabela global (obtida dos 12 arquivos de profecia) (figura 12).

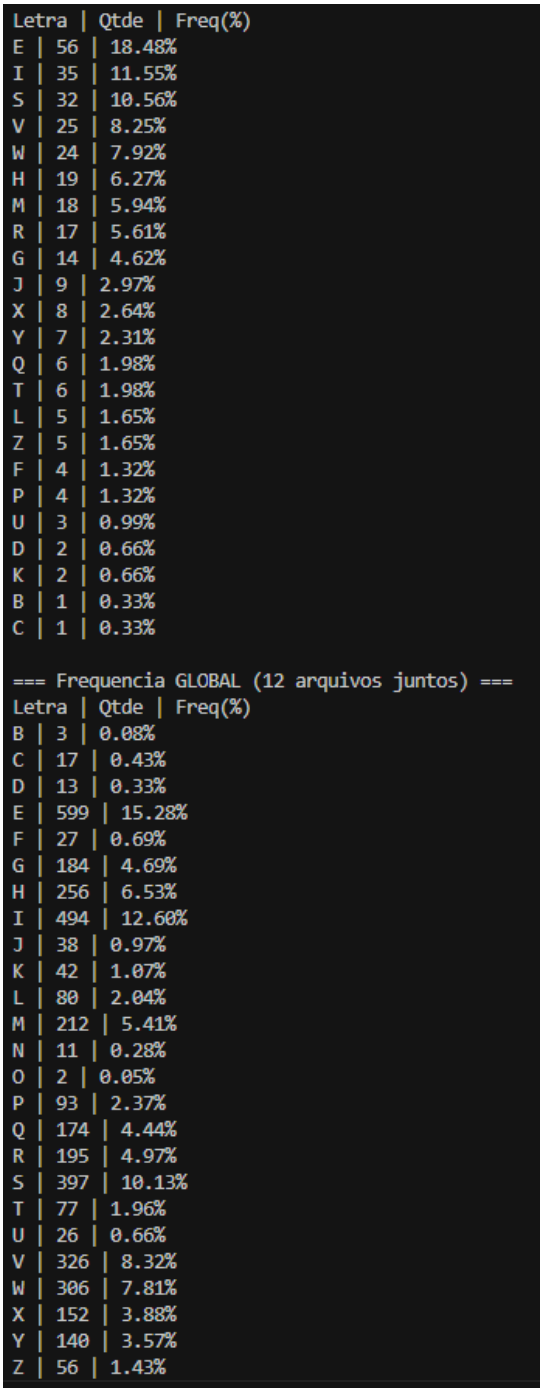


Figura 12: Tabelas de frequência.

A tabela comparativa exibida no terminal facilitou a visualização das discrepâncias estatísticas (figura 13).

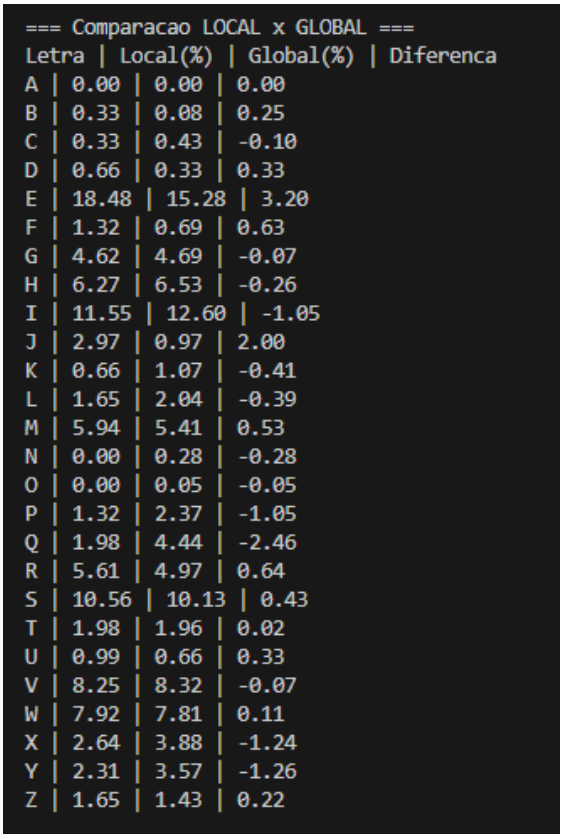


Figura 13: Tabela comparativa.

A sugestão automática de mapeamento exibida após a tabela comparativa, mostrou-se precisa para as vogais mais comuns (como “A” e “E”), oferecendo um ponto de partida sólido para a quebra da cifra (figura 14).

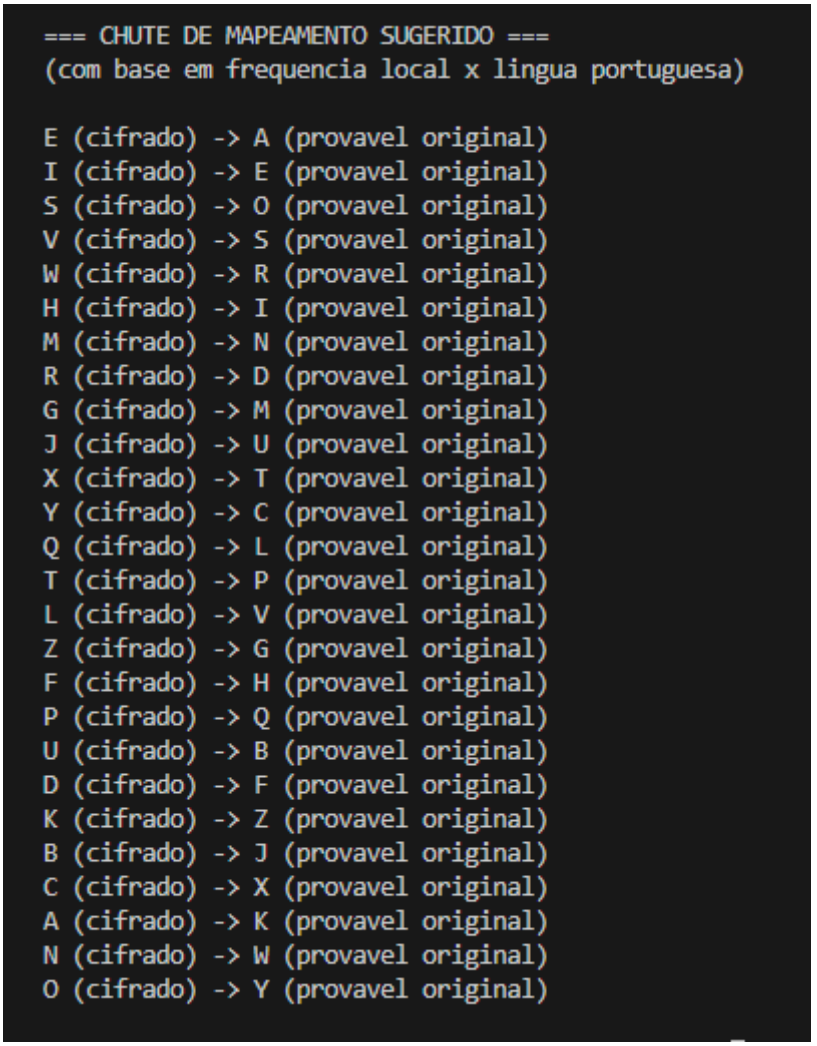


Figura 14: Mapeamento sugerido.

Após a exibição dos mapeamentos sugeridos, é perguntado ao usuário se deseja aplicar algum mapeamento. Caso queira, é necessário informar a letra cifrada que deseja decifrar, e em seguida a letra que será mapeada no lugar (figura 15).

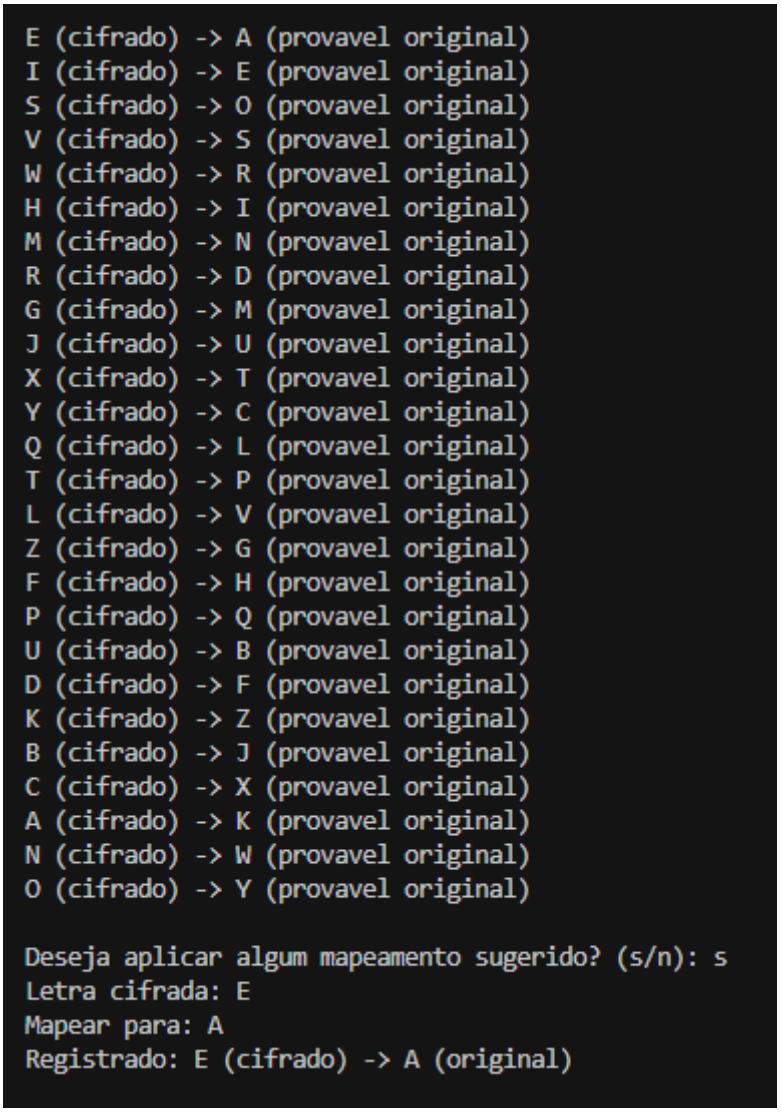


Figura 15: Aplicação de mapeamento sugerido.

- Busca e refinamento:

A utilização do algoritmo Shift-And para realizar o casamento exato de caracteres (opção 3) permite localizar padrões exatos instantaneamente, validando suspeitas sobre palavras comuns. Após a seleção da opção 3, é necessário escrever o padrão a ser procurado. Caso seja encontrado, o número de ocorrências será atualizado e exibido no terminal (figura 16).

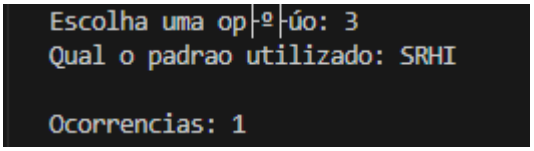


Figura 16: Casamento exato.

Ao selecionar a opção 4 para realizar o casamento aproximado de caracteres, é feita uma busca por palavras parcialmente decifradas com uma margem de erro definida pelo

usuário. Além disso, é necessário informar o padrão a ser adotado (figura 17). Nessa opção são informadas as posições das ocorrências, além do número total de ocorrências encontradas.

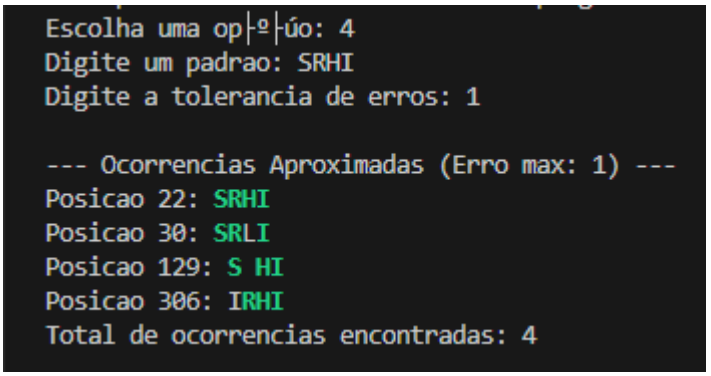


Figura 17: Casamento aproximado.

- Decifração final:

Através do processo iterativo de alterar a chave (opção 5) e verificar o texto parcialmente decifrado (figura 18),que é exibido com as letras descobertas destacadas em verde, é possível reconstruir a mensagem original completa.

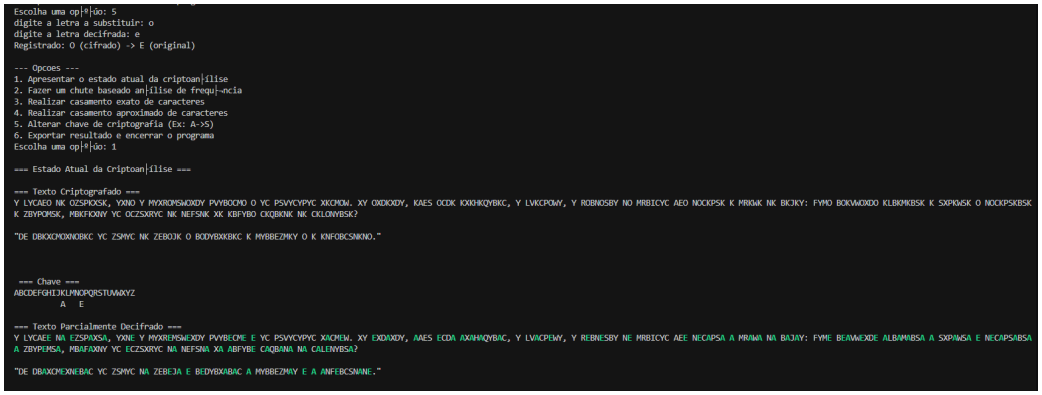


Figura 18: Opção 5 e exibição da mensagem.

A exportação final (opção 6) gera um arquivo contendo tanto a chave descoberta quanto o texto plenamente legível, concluindo com sucesso a missão proposta (figuras 19 e 20).

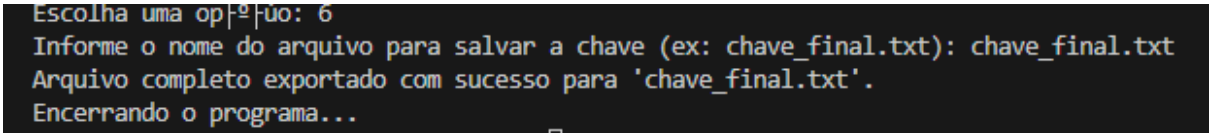


Figura 19: Mensagem da opção 6.

Figura 20: Arquivo com a chave e o texto.

Conclusão

O desenvolvimento do sistema proposto permitiu aplicar, de maneira integrada, diferentes técnicas de análise de algoritmos e métodos clássicos de criptoanálise. As funcionalidades implementadas demonstraram como operações relativamente simples, como análise de frequência, casamento exato e aproximado de padrões e ajustes progressivos da chave, podem ser combinadas para auxiliar na decifração de cifras de substituição.

Durante o processo, observou-se que o uso de algoritmos eficientes, especialmente nas etapas de busca aproximada com a variante do Shift-And, contribuiu para reduzir o custo computacional e tornar o sistema responsivo mesmo em textos maiores. A necessidade de manter um estado consistente da chave e do texto parcialmente decifrado também reforçou a importância de uma arquitetura modular e bem estruturada.

Embora o programa ainda esteja em fase de finalização, os resultados obtidos até o momento indicam um avanço significativo na reconstrução da chave e do texto claro original.

O trabalho cumpriu seu papel ao aprofundar a compreensão dos algoritmos estudados em aula, bem como ao demonstrar, de forma prática, como conceitos teóricos podem ser aplicados no contexto de criptoanálise. Conclui-se que, com as etapas restantes concluídas, o sistema atenderá integralmente às especificações propostas e servirá como ferramenta funcional e didática para a decifração do material fornecido.

Referências

1. GONZAGA, Julia. Estruturas de dados: Casamento de cadeias. Medium, 2024. Disponível em: https://juliagonnzaga.medium.com/estruturas-de-dados-casamento-de-cadeias-69ff6d50433a. Acesso em: 27 nov. 2025.

2. KHAN ACADEMY. Shift cipher (Cifra de deslocamento). Disponível em: https://pt.khanacademy.org/computing/computer-science/cryptography/ciphers/a/shift-cipher. Acesso em: 27 nov. 2025.