Universidade Federal
de São João del-Rei

Trabalho Prático 3

Leonardo Patias Johansson

202050055

Projeto e Análise de Algoritmos

Ciência da Computação

São João Del Rei
2025.1

Introdução

- **Objetivo do Trabalho:** Aplicar conceitos dos algoritmos de processamento de caracteres e compressão de arquivos vistos em sala de aula.

- **Definição do problema:** Recuperar ocorrências de padrões em arquivos comprimidos e não comprimidos. O código recebe um texto contendo (n) caracteres, o padrão contendo (m) caracteres, o número de erros ($0 \leq k < m$), e imprime todas as ocorrências do padrão no texto.

Parte 1: Busca Aproximada em Arquivos Não Comprimidos

- **Objetivo:** Usar 2 Algoritmos para casamento aproximado. Comparar o desempenho dos dois algoritmos para valores de diferentes (k) sendo o número de erros aceitáveis.

- **Algoritmos Utilizados:**

    - **Programação Dinâmica:**

        - **Descrição:** A Distância de Levenshtein mede o número mínimo de edições de um único caractere (inserções, deleções ou substituições) necessárias para transformar uma string em outra.
          A Programação Dinâmica é utilizada para calcular essa distância de forma eficiente.
        - **Complexidade:** A complexidade: O(nm)

    - **Algoritmo Shift-And:**

        - **Descrição:** O algoritmo Shift-And é um método eficiente para busca de padrões que utiliza operações bit a bit para realizar comparações em paralelo. Para o casamento aproximado, ele é baseado no algoritmo Shift-And com erros.
        - **Complexidade:** A complexidade: O(nm/w)

- **Entrada e Saída:**

  - Primeiro, utilize o comando (`make`) no terminal, que vai compilar o código e criar o executável (`tp3_parte1`). Depois de usar o (`make`), execute o programa assim:

    ./tp3_parte1 <algoritmo> <arquivo_texto> <arquivo_padroes> <k>

    `Onde:`

    - `<algoritmo>:`
      `1 para Programação Dinâmica`
      `2 para Shift-And.`
    - `<arquivo_texto>: Caminho para o arquivo de texto.`
    - `<arquivo_padroes>: Caminho para o arquivo contendo os padrões, um por linha.`
    - `<k>: O número máximo de erros aceitáveis.`

  - Saída no Terminal:

```
leonardopj@Leonardo:~/Codigos/Trabalho 3/parte1$ ./tp3_parte1 1 texto_grande.txt padroes_grande.txt 0
--- Iniciando Busca ---
Algoritmo selecionado: Programação Dinâmica
Com k = 0
---------------------------------------
Padrão 'Capitu': Tempo: 0.30 ms | Comparações: 48138
Padrão 'seminário': Tempo: 0.47 ms | Comparações: 80230
```

```
leonardopj@Leonardo:~/Codigos/Trabalho 3/parte1$ ./tp3_parte1 2 texto_grande.txt padroes_grande.txt 0
--- Iniciando Busca ---
Algoritmo selecionado: Shift-And
Com k = 0
---------------------------------------
Padrão 'Capitu': Tempo: 0.04 ms | Comparações: 8023
Padrão 'seminário': Tempo: 0.03 ms | Comparações: 8023
```

    A saída principal, contendo o padrão e as posições exatas de suas ocorrências no texto, será gravada no arquivo:

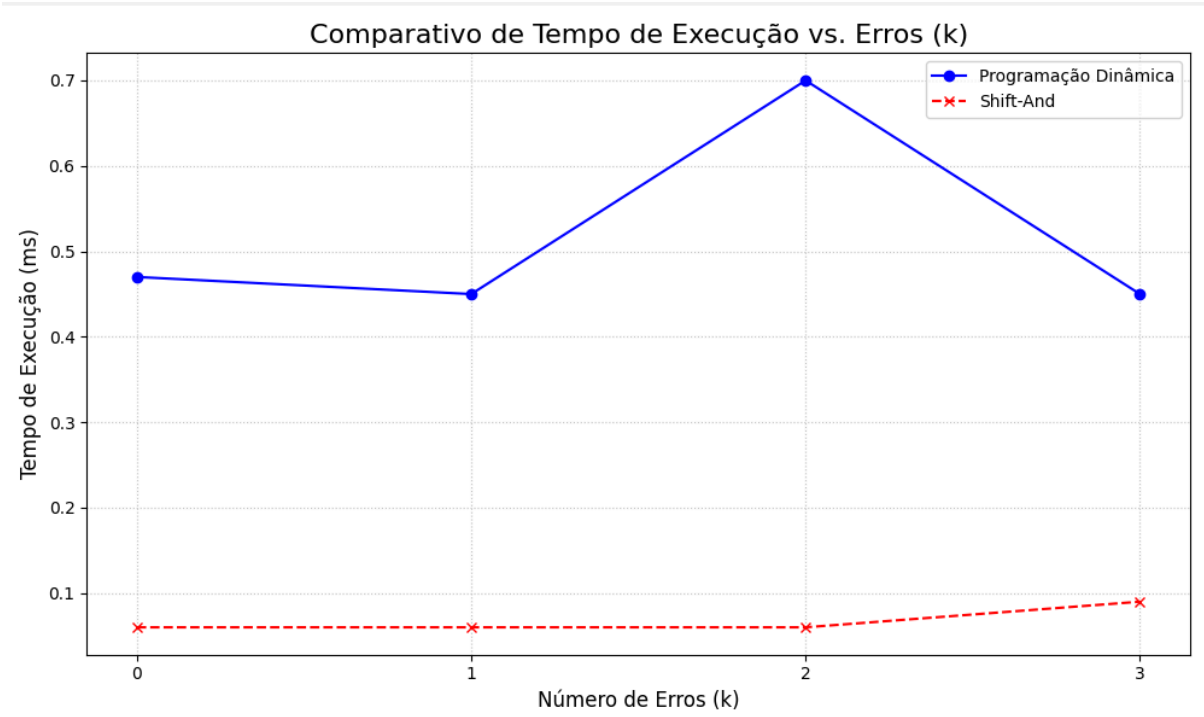
    `Arquivo Saída.`

```
parte1 > Arquivo Saida
1   Capitu 1530 1637 1848 2163 2415 2422 2932 3367 4103 4150 4528 4931 5110 5339 5771 5923 6240 6503 6993 7629 7824
2   seminário 1959 2909 4394 5216 5674 6339 6443 6747 6760 7493 7738
3
```

Análise de Resultados:

- **Metodologia:** Realizei os testes com um arquivo: texto_grande.txt e padroes_grande.txt onde tem um texto maior para visualizar melhor a diferença de tempos. Utilizei o (k) de 0 a 3 para alguns testes.

- **Resultados:**



- **Discussão:** com base nos resultados da pra notar que o shift-and tem uma variação bem baixa em relação ao tempo quando se aumenta os números de erros (k). Já a Programação Dinâmica acabou tendo uma oscilação em um dos testes mas mantém a mesma velocidade mudando os valores de (k).

Parte 2: Busca Exata em Arquivos Comprimidos

- **Objetivo:** Comparar o desempenho do algoritmo de Boyer-Moore Horspool (BMH) para arquivos comprimidos e não comprimidos, deve utilizar o código de Huffman com marcação para o arquivo comprimido.

- **Algoritmos Utilizados:**

- **Algoritmo de Compressão Huffman:**

  - **Descrição:** O Código de Huffman é um método de compressão de dados sem perdas que opera atribuindo códigos binários de tamanho variável aos caracteres de um texto. Caracteres mais frequentes recebem códigos mais curtos, enquanto caracteres menos frequentes recebem códigos mais longos. O processo envolve o cálculo da frequência de cada caractere no texto, a construção de uma árvore binária (árvore de Huffman) utilizando uma fila de prioridade (Min-Heap) para agrupar os caracteres com menor frequência, e a subsequente geração dos códigos binários a partir dessa árvore.

  - **Complexidade:** $O(N \cdot Cmax)$

- **Algoritmo Boyer-Moore-Horspool (BMH):**

  - **Descrição:** O algoritmo Boyer-Moore-Horspool (BMH) é um método eficiente para busca exata de padrões em texto. Ele otimiza o processo de comparação ao escanear o padrão da direita para a esquerda e utilizar a "regra do mau caractere" (bad character rule). Essa regra permite que o algoritmo realize saltos maiores no texto quando um caractere no texto não corresponde ao caractere esperado no padrão, ou quando o caractere mismatch não existe no padrão, evitando comparações desnecessárias e acelerando a busca.

  - **Complexidade:**
    Melhor caso O(N/M).
    Pior caso O(NM).

- **Busca em Arquivo Comprimido:**

    - **Abordagem:** Para a busca em arquivos comprimidos, o texto é comprimido utilizando o algoritmo de Huffman. depois, o padrão também é comprimido.

      Para comparar o desempenho e demonstrar a capacidade de buscar em dados comprimidos, foi usada a **busca linear bit a bit**.

- **Entrada e Saída:**

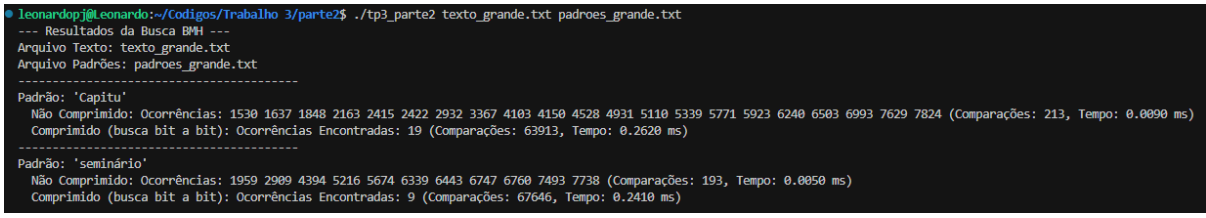
    - Primeiro, utilize o comando (`make`) no terminal, que vai compilar o código e criar o executável (`tp3_parte2`). Depois execute o programa da seguinte forma:

      ./tp3_parte2 <arquivo_texto> <arquivo_padroes>

      Onde:

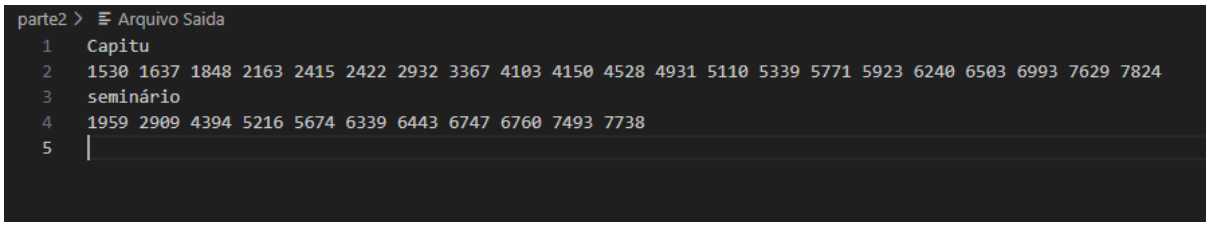
        - `<arquivo_texto>`: Caminho para o arquivo de texto.
        - `<arquivo_padroes>`: Caminho para o arquivo contendo os padrões, um por linha.

    - Saída no Terminal:



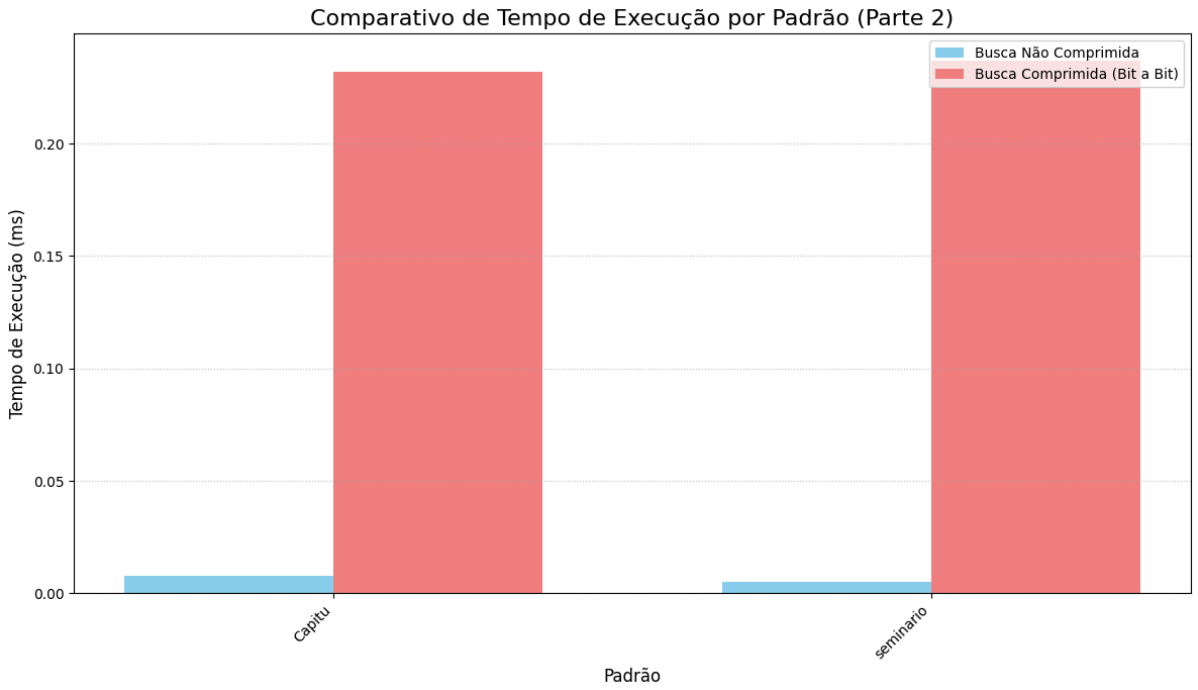
      A saída principal, contendo o padrão e as posições exatas de suas ocorrências no texto, será gravada no arquivo:

      `Arquivo Saída`.



- **Análise de Resultados:**

    - **Metodologia:** Realizei os testes com um arquivo: texto_grande.txt e padroes_grande.txt onde tem um texto maior para visualizar melhor a diferença de tempos.
    - **Resultados:**



    - **Discussão:** com base nos resultados da pra notar que a busca comprimida tem um tempo de execução bem alta em relação a busca não comprimida.

Conclusão

- O trabalho fez eu colocar em prática o que foi visto em sala de aula fazendo a implementação de algoritmos de busca de padrões e compressão de dados.
  Na Parte 1, a implementação e comparação dos algoritmos de Programação Dinâmica e Shift-And.
  Na Parte 2, a busca exata em texto não comprimido com Boyer-Moore-Horspool (BMH) e a busca em texto comprimido por Huffman, utilizando uma abordagem linear bit a bit.