Film Box

GitHub

Ana Paula Natsumi Yuki
Theo Diniz Viana
Gabriel Henrique Pereira Carvalhaes
Lucas Teixeira Reis

Formulário Técnico

1. **Qual campo textual foi escolhido para aplicar os algoritmos de casamento de padrões? Por quê?**

- O campo textual escolhido foi o título do filme. O motivo dessa escolha foi a relevância para o negócio, pois a funcionalidade de busca textual por título é a mais crítica para a experiência do usuário. Além disso, títulos são cadeias de caracteres relativamente curtas em que a busca por casamento de padrões se mostra uma operação eficiente.

2. **Explique o funcionamento do KMP implementado.**

- O KMP implementado utiliza a Tabela de Prefixo / Sufixo Mais Longo (LPS Array) para otimizar o processo de busca e garantir a complexidade linear O(N + M) (onde N é o tamanho do texto e M é o tamanho do padrão).

    1. **Pré-processamento (computeLPSArray):**

        - Cria a LPS Array, que armazena, para cada posição do padrão, o comprimento do maior prefixo do padrão que é também um sufixo do prefixo atual.

        - Esta tabela age como um "mapa de recuo", indicando o máximo de caracteres que o padrão pode ser deslizado após uma falha (*mismatch*) sem precisar retroceder no texto.

    2. **Casamento (search):**

        - O casamento ocorre da esquerda para direita, usando dois ponteiros: i (texto) e j (padrão).

        - **Sucesso (P[j] == T[i]):** Ambos os ponteiros (i e j) avançam.

        - **Falha (P[j] != T[i]):** O ponteiro do texto ‘i’ permanece parado. O ponteiro do padrão ‘j’ é recuado para o índice sugerido pelo LPS Array (j = lps[j - 1]). Isso alinha o maior prefixo casado com o sufixo

correspondente e tenta continuar o casamento do ponto onde a falha ocorreu.

- **Princípio Central:** O ponteiro do texto nunca retrocede, garantindo a eficiência O(N).

3. **Explique o funcionamento do Boyer-Moore implementado**

- O algoritmo Boyer-Moore implementado foca na Heurística do Caractere Ruim. Seu objetivo é deslizar o padrão o máximo possível após encontrar um mismatch, comparando o padrão com o texto da direita para a esquerda.

1. **Pré-processamento (computeBadCharacterShift):**

   - Cria uma tabela que mapeia cada caractere do alfabeto para a sua última posição de ocorrência no padrão.

2. **Casamento (search):**

   - A comparação começa do final do padrão (j = M - 1) para a esquerda.

   - **Falha (Mismatch):** Se o caractere do texto (T[s+j]) for diferente do caractere do padrão (P[j]),a heurísitca é aplicada:

     - **Caractere Ruim:** O caractere T[s +j] é o “caractere ruim” que causou a falha.

     - **Cálculo do Salto (Shift):** O algoritmo calcula o salto máximo necessário para alinhar a última concorrência conhecida do “caractere ruim” no padrão com a posição atual do texto.

     - **Fórmula:** Shift = max(1,j - badChar[T[s + j]])

     - O padrão é deslizado em shift posições ignorando uma parte do texto.

3. **Princípio Central:** Ao começar a busca pelo final do padrão e usar a tabela de ocorrências, o Boyer-Moore frequentemente atinge um desempenho superlinear (na prática, muito rápido), pois ele pula comparações em grandes pedaços do texto.

4. **Descreva como integrou os algoritmos ao sistema.**

A integração seguiu a arquitetura de camadas do Spring Boot, encapsulando a lógica de AEDS e expondo-a via REST:

1. **Criação das Classes de Algoritmo:**
    - As lógicas puras de KMP e Boyer-Moore foram criadas em um novo pacote (`com.example.tpaedsiii.repository.bd.search`) como métodos estáticos para fácil reuso.
2. **Camada de Repositório (`FilmeRepository.java`):**
    - Foram adicionados os métodos `searchByTitleKMP()` e `searchByTitleBoyerMoore()`.
    - Integração de Dados: Esses métodos chamam `hashFilmes.readAll()` para realizar uma varredura completa de todos os títulos de filmes.
    - Aplicação do Algoritmo: Para cada filme lido, o método aplica `KMP.search()` (ou `BoyerMoore.search()`) no título, e se houver correspondência, adiciona o `Filme` aos resultados.
3. **Camada de Serviço (`FilmeService.java`):**
    - Adicionou-se o método *pass-through* (`buscarPorTituloKMP`, etc.) para isolar a lógica de negócio (se houvesse) da camada de interface.
4. **Camada de Controle (`FilmeController.java`):**
    - Foram criados os novos *endpoints* REST GET para acesso público:
        - `/api/filmes/search/kmp?pattern={termo}`
        - `/api/filmes/search/bm?pattern={termo}`
    - O controlador recebe o `pattern` e delega a chamada ao `FilmeService`.
5. **Interface Gráfica (`index.html`):**
    - Uma nova seção de formulário foi criada, permitindo que o usuário insira o padrão, selecione o algoritmo (KMP/BM) e execute a busca via AJAX, exibindo os resultados formatados em JSON.

5. **Quais dificuldades encontrou na implementação dos dois algoritmos?**

As principais dificuldades encontradas (ou potenciais dificuldades em implementações complexas) são:

1. **Lógica do KMP (Tabela LPS):** A maior dificuldade é garantir a precisão da Tabela LPS. Um erro de "off-by-one" (erro de índice) no cálculo do `computeLPSArray` ou na lógica de recuo do ponteiro `j` pode invalidar todo o algoritmo e levar a uma falha na correspondência (como o problema que você enfrentou no teste).

2. **Lógica do Boyer-Moore (Heurísticas):**

    - **Bad Character Rule (Regra do Caractere Ruim):** Garantir que o cálculo do salto (`Shift`) após um *mismatch* seja o máximo possível, mas seguro, é complexo. Erros no mapeamento da última ocorrência (`badChar` array) são comuns.

    - **Heurística Opcional (Good Suffix):** Integrar a Heurística do Sufixo Bom, que lida com o caso em que o *suffix* casado é conhecido, aumenta drasticamente a complexidade do pré-processamento. (Embora sua implementação tenha sido simplificada para focar apenas no Caractere Ruim).

3. **Integração de Performance (AEDS vs. Aplicação):** A dificuldade conceitual foi aplicar algoritmos O(N+M) em um sistema que já exige um Full Scan O(N) para resgatar os dados. Isso significa que, embora o KMP/BM sejam algoritmos rápidos, a operação total continua limitada pela leitura sequencial dos N registros do seu sistema de persistência customizado (`hashFilmes.readAll()`).