Universidade Federal
de São João del-Rei

Ciência da Computação
Projeto e Análise de Algoritmos

Documentação Trabalho Prático 3
A Substring Perdida

Lucas Eduardo Bernardes de Paula

Messias Feres Curi Melo

2024.1

1 Introdução

Em um reino distante, onde a magia fazia parte do cotidiano e as palavras carregavam poderes secretos, uma antiga lenda despertou a curiosidade dos sábios e guerreiros. Conta-se que, há muito tempo, uma substring de poder incomensurável foi perdida nas entrelinhas de textos místicos, aguardando para ser descoberta. Temendo que esse poder caísse em mãos erradas, os sábios do reino decidiram reunir os mais habilidosos buscadores para empreender uma jornada na tentativa de recuperar essa misteriosa substring.

Entre os escolhidos estava um jovem aprendiz de magia, ávido por aventuras e desafios. O Grande Mago, protetor dos segredos ocultos, confiou-lhe a missão de encontrar essa substring perdida, que se esconde nas profundezas de palavras encantadas. Armado com um pergaminho mágico contendo a string principal e a substring a ser encontrada, a tarefa era analisar diferentes trechos de palavras encantadas e localizar a substring em meio a elas. A precisão e a rapidez seriam cruciais, pois a jornada estava repleta de armadilhas e enigmas complexos.

No mundo real, o problema de buscar uma substring em um texto é amplamente aplicável, desde sistemas de busca em grandes bases de dados até a análise de sequências genéticas. Para resolver esse desafio, utilizam-se dois algoritmos poderosos e eficientes: Knuth-Morris-Pratt (KMP) e Boyer-Moore-Horspool. O algoritmo KMP se destaca por sua habilidade de realizar a busca em tempo linear, evitando comparações redundantes. Por outro lado, o algoritmo Boyer-Moore-Horspool, com sua estratégia de saltos nas comparações, é extremamente eficaz em contextos práticos onde a busca precisa ser rápida e precisa.

Assim, torna-se necessário desenvolver um programa que encontre a solução de maneira otimizada, ao mesmo tempo em que se garanta a facilidade de manutenção do código, alcançada por meio de boas práticas de programação e documentação adequada, conforme demonstrado adiante.

2 Estruturando a Solução

Levando em consideração o problema apresentado, é de grande importância uma boa estruturação de código. Por isso, é imprescindível definir como será feita a estruturação antes de começar a programar definitivamente.

Conforme ilustrado no fluxograma da Figura 1, o código inicia sua execução ao receber os argumentos de entrada. Em seguida, ele cria arrays para armazenar tanto o texto quanto o padrão a ser pesquisado, além de registrar os intervalos que serão analisados. Dependendo da estratégia especificada via linha de comando, o algoritmo correspondente é executado repetidamente para buscar o padrão nos

intervalos fornecidos. Finalmente, o código grava em um arquivo se o padrão foi encontrado em cada intervalo e exibe o tempo total de execução no terminal.

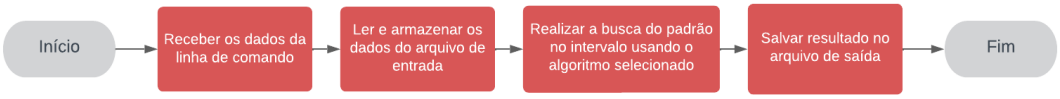

Figure 1: Fluxograma de funcionamento

2.1 Leitura de arquivos

Para receber as informações necessárias para a busca mágica, o programa utiliza um arquivo de entrada estruturado da seguinte forma: na primeira linha, é fornecida uma string de tamanho n, que representa a sequência principal onde a substring será procurada. A segunda linha contém uma string de tamanho m, que corresponde à substring que deve ser encontrada. Na terceira linha, um inteiro k indica o número de consultas a serem realizadas. A seguir, as próximas k linhas descrevem as consultas, onde cada linha contém dois inteiros a e b, representando os intervalos na sequência principal em que a substring será buscada. A compilação do código é gerenciada por um `Makefile`, que automatiza o processo, criando o executável final ao inserir o comando `make` no diretório do projeto. Para executar o programa, basta utilizar o comando `./tp3`, seguido da estratégia desejada e do arquivo de entrada, conforme especificado na linha de comando:

./tp3 <estrategia> entrada.txt

Para receber esses dados é necessário utilizar os parâmetros $argc$ e $argv$, recebidos pela função $main()$, verificando se as entradas são válidas para o funcionamento correto do programa.

Após obter o endereço do arquivo de entrada, a leitura completa do mesmo é realizada utilizando a função `getline()`, que pertence à biblioteca `stdio.h`.

2.2 Estruturar o texto e o padrão

A estruturação do texto e do padrão a ser pesquisado é realizada a partir da leitura do arquivo, onde são gerados dois arrays com base no número de caracteres lidos pela função `getline()` em cada chamada. Esses arrays contêm os caracteres do texto e do padrão, respeitando o tamanho necessário para armazenar o conteúdo lido da entrada.

2.3 Solução proposta

Para resolver o problema, foram desenvolvidas duas estratégias distintas: uma baseada no algoritmo de Knuth-Morris-Pratt (KMP) e outra no algoritmo de Boyer-Moore-Horspool (BMH). O KMP foi escolhido por sua capacidade de realizar buscas de padrões de forma eficiente, evitando comparações redundantes. Esse algoritmo utiliza uma tabela de falhas para otimizar as comparações, o que resulta em um desempenho linear em relação ao tamanho do texto e do padrão. A escolha do KMP se justifica pela necessidade de garantir eficiência em cenários onde o desempenho no pior caso é uma preocupação relevante. Na figura 2, é possível visualizar um exemplo do algoritmo.

Figure 2: Exemplo de subsequência encontrada em um texto

Por outro lado, o algoritmo Boyer-Moore-Horspool (BMH) foi escolhido por sua eficiência prática em buscas rápidas, especialmente quando o padrão é significativamente menor que o texto. O BMH reduz o número de comparações ao pular seções do texto com base na última ocorrência de caracteres, tornando-o bastante ágil em muitas situações. Contudo, seu desempenho pode se deteriorar com textos muito grandes

O funcionamento de ambos os algoritmos é descrito com mais detalhes na seção 4 da documentação

2.4 Salvar os resultados no arquivo de saída

Ao final, o resultado das consultas em cada um dos intervalos é salvo em um arquivo de saída chamado `saida.txt`, utilizando a função `fprintf` da biblioteca `stdio.h`.

Após a conclusão do programa, todas as estruturas de dados alocadas dinamicamente são liberadas, garantindo que não ocorram vazamentos de memória indesejados.

3 Estrutura de dados

Para a implementação dos algoritmos de busca de padrões (KMP e Boyer-Moore-Horspool), foi utilizada a estrutura de dados array. Esta escolha se deve à sua eficiência em termos de acesso direto e manipulação de elementos, características

essenciais para as operações frequentes de comparação de padrões. Os arrays permitem que os algoritmos realizem buscas rápidas e eficazes, otimizando o tempo de execução.

No algoritmo KMP, a tabela de falhas (failure table) é representada por um array, que armazena o comprimento do maior prefixo do padrão que também é um sufixo. Isso possibilita que o algoritmo salte partes do texto durante a busca, evitando comparações desnecessárias. Já no Boyer-Moore-Horspool, a tabela de deslocamento (shift table) também é armazenada em um array, permitindo saltos maiores no texto quando uma correspondência falha, acelerando significativamente a busca. Além disso, os arrays são utilizados para armazenar posições de correspondência, facilitando o controle do fluxo de execução.

A escolha de arrays como estrutura de dados principal garante uma solução simples, clara e eficiente, que atende aos requisitos de busca em grandes textos ou padrões. Sua utilização não só simplifica a implementação dos algoritmos, mas também assegura um desempenho otimizado, crucial para lidar com textos de tamanhos elevados e múltiplas queries. Na figura 3, pode ser visto um exemplo simples da utilização da estrutura.

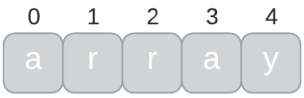

Figure 3: Exemplo de utilização do array

4 Estratégia de resolução

Conforme visto em seções anteriores, foram adotadas duas estratégias distintas para resolver o problema de busca de padrões dentro de um texto: o algoritmo Knuth-Morris-Pratt (KMP) e o algoritmo Boyer-Moore-Horspool (BMH).

4.1 Knuth-Morris-Pratt (KMP)

O algoritmo Knuth-Morris-Pratt (KMP) é uma solução eficiente para o problema de casamento de padrões em textos, destacando-se por evitar comparações redundantes durante o processo de busca. A eficiência do KMP se deve a uma etapa inicial de pré-processamento do padrão que permite ao algoritmo aproveitar informações sobre o próprio padrão para acelerar a busca.

O primeiro passo do KMP é construir uma estrutura conhecida como "tabela de falhas" ou "tabela de prefixos". Esta tabela armazena informações sobre os maiores

prefixos do padrão que são também sufixos. Essencialmente, ela captura como as partes iniciais do padrão se repetem dentro do próprio padrão. Essa tabela permite que, quando o algoritmo enfrenta uma falha durante a busca (ou seja, quando um caractere do texto não corresponde ao caractere correspondente do padrão), ele possa recomeçar a comparação a partir de uma posição que já se sabe que é potencialmente compatível, sem precisar voltar e revisar os caracteres do texto já processados.

Durante a busca, o KMP percorre o texto de maneira linear, comparando o padrão com o texto uma posição por vez. Em caso de uma correspondência parcial seguida de uma falha, a tabela de falhas indica o próximo índice do padrão que deve ser comparado. Isso significa que o algoritmo pode "pular" partes do texto que já foram analisadas, porque ele já sabe que essas partes não podem fazer parte de uma correspondência válida. Essa abordagem evita a necessidade de retroceder no texto, permitindo que o KMP prossiga de forma contínua, sem revisitar os caracteres do texto que já foram processados.

Uma das vantagens práticas do KMP é sua habilidade de lidar eficientemente com padrões que contêm repetições internas, pois o algoritmo utiliza essas repetições para minimizar o trabalho necessário ao longo da busca.

Na adaptação para buscas dentro de intervalos específicos, após encontrar todas as ocorrências do padrão, o algoritmo verifica se as posições correspondentes estão dentro dos limites dos intervalos predefinidos. Embora essa verificação adicione uma pequena sobrecarga, proporcional ao número de intervalos e correspondências, ela é administrada dentro do escopo de um algoritmo linear, mantendo a eficiência característica do KMP em buscas segmentadas.

O fluxograma na figura 4 ilustra o funcionamento detalhado do algoritmo, incluindo a adaptação para verificação de intervalos específicos.

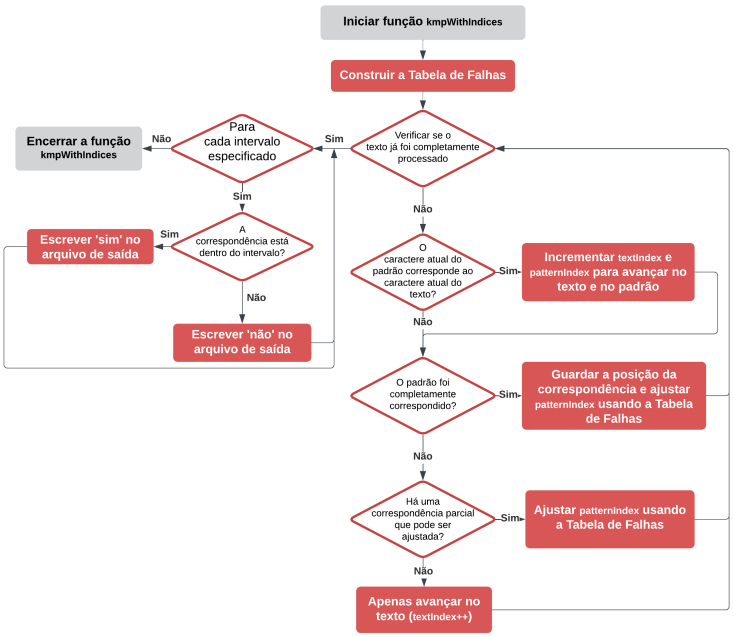

Figure 4: Fluxograma do funcionamento do algoritmo KMP

4.2 Boyer-Moore-Horspool (BMH)

O algoritmo Boyer-Moore-Horspool (BMH) é uma variação simplificada e eficiente do algoritmo Boyer-Moore, desenvolvido por Nigel Horspool em 1980. Ele é amplamente utilizado em tarefas de busca de padrões em textos devido à sua simplicidade e eficácia prática.

O BMH se destaca por sua abordagem baseada em deslocamentos, que permite otimizar a busca de padrões ao realizar grandes saltos durante a comparação entre o texto e o padrão. Esse comportamento é possibilitado por uma fase de pré-processamento, onde é construída uma tabela de deslocamento. Esta tabela atribui a cada caractere do alfabeto um valor que indica o número de posições que o padrão pode ser movido adiante caso uma falha de correspondência ocorra naquele caractere específico.

Durante o pré-processamento, todos os caracteres do alfabeto são inicialmente atribuídos com um deslocamento igual ao tamanho do padrão. Para os primeiros $m - 1$ caracteres do padrão, onde m é o tamanho do padrão, o deslocamento é atualizado para o valor $m - i$, sendo i a posição do caractere no padrão. Com isso,

a tabela de deslocamento possibilita que, ao encontrar um caractere no texto que não corresponda ao caractere do padrão, o algoritmo possa pular diretamente para a próxima posição potencialmente válida, sem revisitar partes já processadas do texto.

Ao contrário de outros algoritmos, como o Knuth-Morris-Pratt (KMP), que começam a comparação a partir do início do padrão, o BMH inicia a comparação pelo final do padrão em relação ao texto, prosseguindo de trás para frente. Essa técnica permite que, ao detectar uma falha, o algoritmo possa ignorar uma parte significativa do texto, movendo o padrão diretamente para a próxima posição onde uma correspondência válida pode ser possível. Isso torna o BMH particularmente eficiente em textos que contêm muitos caracteres que não aparecem no padrão, permitindo saltos frequentes e consideráveis.

Para resolver a problemática proposta, é necessário adaptar o BMH para realizar buscas dentro de intervalos específicos. Após a conclusão da busca principal, o algoritmo verifica se as posições das correspondências encontradas estão dentro dos intervalos especificados. Embora essa adaptação adicione uma pequena sobrecarga, ela é bem administrada dentro do escopo do BMH, mantendo a eficiência característica do algoritmo.

A Figura 5 apresenta o fluxograma do algoritmo BMH, detalhando cada etapa do processo. O pseudocódigo a seguir ilustra o funcionamento detalhado do algoritmo, incluindo a adaptação para verificação de intervalos específicos.

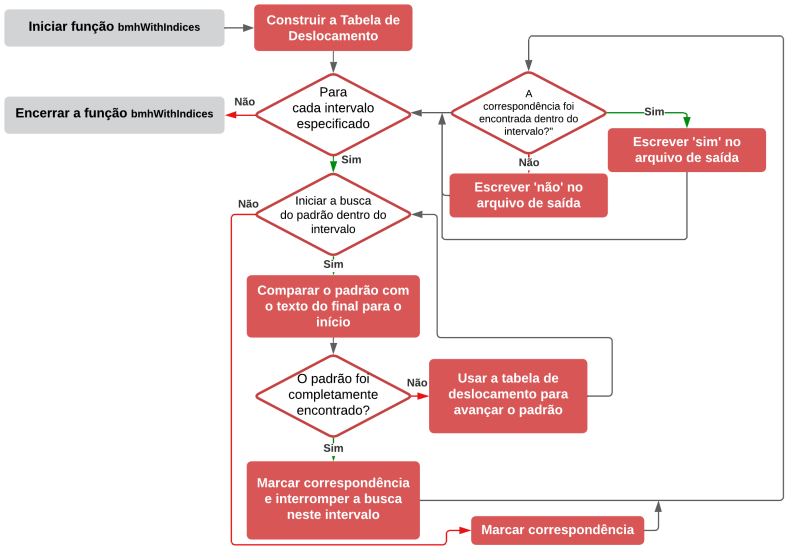

Figure 5: Fluxograma do funcionamento do algoritmo BMH

5 Análise matemática

Os dois algoritmos apresentados na seção anterior possuem funcionamentos notavelmente diferentes, o que resulta em tempos de execução distintos. Por isso, é importante realizar uma análise matemática detalhada do funcionamento de ambos.

5.1 Análise do algoritmo Knuth-Morris-Pratt (KMP)

A eficiência do algoritmo Knuth-Morris-Pratt (KMP) é diretamente ligada à sua complexidade linear, que se desdobra em duas fases principais: o pré-processamento do padrão e a busca no texto. Na fase de pré-processamento, a construção da "tabela de falhas" ocorre em tempo $O(m)$, onde m é o tamanho do padrão. Essa tabela permite ao algoritmo determinar rapidamente a próxima posição a ser comparada após uma falha, o que evita a necessidade de retrocessos no texto durante a busca. Como o padrão é processado uma única vez e cada operação de construção da tabela é realizada em tempo constante, a complexidade dessa fase permanece linear.

Durante a execução do algoritmo, na fase de busca, o KMP percorre o texto de tamanho n em tempo $O(n)$. Graças à tabela de falhas pré-processada, o algoritmo consegue evitar comparações desnecessárias, garantindo que cada caractere do texto seja examinado no máximo uma vez. Isso elimina a possibilidade de retrocessos no texto, o que é uma melhoria significativa em relação a algoritmos de busca de padrões menos eficientes. A combinação dessas duas fases—com a fase de pré-processamento contribuindo com $O(m)$ e a fase de busca com $O(n)$—resulta na complexidade total do algoritmo sendo $O(m + n)$.

Quando o KMP é adaptado para realizar buscas em intervalos específicos do texto, a complexidade da verificação das correspondências encontradas é influenciada pelo número de intervalos consultados. Se houver k intervalos definidos, a verificação de cada ocorrência do padrão para determinar se está dentro de um dos intervalos resulta em uma sobrecarga adicional de $O(k)$. Assim, para t ocorrências do padrão encontradas, a complexidade adicional para a verificação em intervalos específicos é $O(k \cdot t)$. Isso significa que, embora a busca no texto e o pré-processamento do padrão permaneçam lineares, a complexidade final da tarefa pode aumentar em função do número de intervalos consultados e do número de correspondências encontradas, tornando-se $O(m + n + k \cdot t)$.

5.2 Análise do algoritmo Boyer-Moore-Horspool (BMH)

O algoritmo Boyer-Moore-Horspool (BMH) é uma otimização simplificada do algoritmo Boyer-Moore, conhecido por sua eficiência em busca de padrões graças à

utilização de uma tabela de deslocamento. A fase de pré-processamento do BMH envolve a construção dessa tabela, onde cada caractere do alfabeto é mapeado para um valor que indica o número de posições que o padrão pode ser deslocado em caso de falha de correspondência. Essa fase ocorre em tempo $O(m)$, onde m é o tamanho do padrão. O tamanho do alfabeto σ pode ser considerado constante para alfabetos fixos e pequenos, como o ASCII, e portanto, não influencia significativamente a complexidade.

Durante a busca, o BMH percorre o texto de tamanho n comparando o padrão com o texto de trás para frente, começando pelo último caractere do padrão. Cada vez que uma falha de correspondência é detectada, o algoritmo utiliza a tabela de deslocamento para pular uma parte do texto, o que pode reduzir significativamente o número de comparações realizadas. No melhor caso, quando o padrão e o texto têm poucas correspondências, o algoritmo pode realizar grandes saltos, resultando em um desempenho superior. No entanto, no pior caso, onde o texto contém muitos caracteres que coincidem com o final do padrão, a complexidade pode aproximar-se de $O(m \cdot n)$, pois o algoritmo pode acabar verificando múltiplas vezes o mesmo conjunto de caracteres no texto.

Ao adaptar o BMH para buscas dentro de intervalos específicos, a complexidade da verificação em cada intervalo depende do tamanho do intervalo e do número de deslocamentos realizados. Se houver k intervalos e o tamanho médio dos intervalos for n_i, a complexidade da busca dentro de cada intervalo será $O(m \cdot n_i)$ no pior caso. Portanto, a complexidade total para buscar em todos os intervalos é $O(k \cdot m \cdot n_i)$, onde n_i é o comprimento do maior intervalo. A fase de pré-processamento permanece $O(m)$, resultando em uma complexidade total de $O(m + k \cdot m \cdot n_i)$.

6 Testes

Para monitorar o tempo de execução total de cada algoritmo na busca de soluções, foi utilizada a biblioteca getrusage.h, que permite acompanhar especificamente o tempo de CPU, evitando assim inconsistências decorrentes do uso do tempo físico como parâmetro de medição.

Os testes estiveram presentes em todas as etapas do desenvolvimento do software, desde o funcionamento das funções mais elementares até a medição do tempo de execução da função para encontrar a solução do problema, garantindo uma aplicação performática e de fácil manutenção. Os dados de tempo obtidos foram utilizados para criar gráficos informativos sobre a eficiência das duas estratégias, proporcionando uma análise mais detalhada das informações coletadas.

6.1 Monitoramento do tempo de execução

Nos testes realizados para avaliar qualitativamente os algoritmos implementados, considerou-se a eficiência dos tempos de execução em diferentes tamanhos de padrões e textos, focando nos cenários de pior caso, onde a máxima quantidade de operações é necessária.

Para uniformizar os testes, foram utilizados textos 10 vezes maiores que seus respectivos padrões. O gráfico da Figura 6 traz os resultados obtidos.

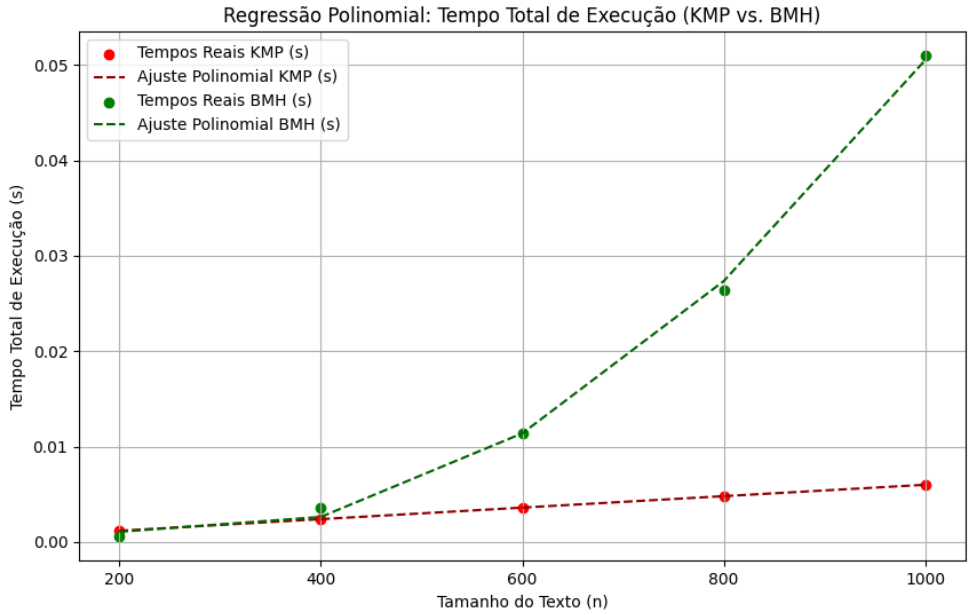

Figure 6: Gráfico do teste de execução dos algoritmos

Tomando por base a informação de que n=10m, a análise do gráfico permite constatar que o algoritmo BMH apresenta uma complexidade quadrática para solucionar o problema, evidenciada pelo aumento substancial no tempo de execução em cenários onde o padrão possui um caractere distinto em sua última posição e o texto é composto por uma sequência repetitiva. Esse comportamento se distancia do algoritmo KMP, cujos tempos se mantiveram lineares e consistentemente menores durante a bateria de testes, devido à sua capacidade de pré-processar o padrão e evitar retrocessos desnecessários durante a busca.

7 Conclusão

O desenvolvimento deste projeto evidenciou a importância de analisar diferentes algoritmos para a busca de padrões em textos, especificamente o Knuth-Morris-Pratt (KMP) e o Boyer-Moore-Horspool (BMH). Durante os testes, o BMH demonstrou eficiência em muitos cenários práticos, mas também apresentou desafios em situações específicas que comprometeram seu desempenho. Por outro lado, o KMP mostrou-se consistente e confiável, mantendo um bom desempenho em uma ampla gama de condições.

A comparação entre os dois algoritmos destacou como as características do problema podem influenciar significativamente o comportamento dos algoritmos. Enquanto o BMH é útil em contextos onde o padrão é relativamente pequeno, o KMP provou ser mais versátil, lidando bem com diferentes tipos de entradas e condições de execução. Essa análise reforça a necessidade de uma avaliação cuidadosa ao escolher a estratégia de busca mais adequada.

É importante notar que, assim como em muitos problemas computacionais, não existe um algoritmo que seja superior em todos os casos. Cada algoritmo traz consigo vantagens e desvantagens que devem ser consideradas com base no contexto específico do problema. A escolha do algoritmo ideal depende de fatores como o tamanho do texto e do padrão, a necessidade de eficiência em termos de tempo e espaço, e a complexidade da implementação.

Em conclusão, o estudo de algoritmos de casamento de padrões, como o KMP e o BMH, destaca a importância de se avaliar cuidadosamente as características do problema antes de selecionar uma solução. O entendimento detalhado desses algoritmos permite otimizar o desempenho em diversas aplicações, contribuindo para o desenvolvimento de sistemas computacionais mais eficientes e precisos.

8 Referências

1. Thomas H. Cormen. *Algoritmos: Teoria e prática.* LTC, 2012.
2. Nivio Ziviani. Projetos de Algoritmos em C e Pascal. 2. ed. São Paulo: Cengage Learning, 2006.