A Busca Mágica pela Substring Perdida

Igor Cunha, Isabelle Matos, Lucas Rocha

8 de maio de 2025

1 Introdução

Em um reino encantado, dizia-se que uma poderosa substring estava perdida nas profundezas das palavras místicas. Os sábios do reino, temerosos dos perigos que essa substring poderia desencadear nas mãos erradas, convocaram os mais habilidosos buscadores para recuperá-la.

O problema, portanto, resume-se a encontrar a substring perdida dentro de uma série de palavras encantadas. Para isso, é dado um texto (string) na entrada, que será utilizado para encontrar a substring também dada na entrada, além de K consultas, cada uma contendo um intervalo no texto onde a substring será procurada. Se a substring estiver presente em uma palavra dentro do intervalo indicado, devemos revelar sua localização.

Dessa forma, percebe-se que o problema envolve encontrar correspondências de cadeias de caracteres ou casamento de padrões exatos em um texto. Esse problema é extremamente relevante e já foi amplamente discutido devido à sua natureza intrinsecamente útil. É possível encontrar diversas aplicações no nosso dia a dia, como: edição de texto, recuperação de informação, estudo de sequências de DNA em biologia computacional, processamento de linguagem natural, entre outras.

Portanto, o objetivo desta documentação é desenvolver e analisar diferentes abordagens para solucionar esse desafio. Os algoritmos implementados para discussão foram: Força Bruta, Knuth-Morris-Pratt (KMP), Boyer-Moore-Horspool (BMH) e Boyer-Moore-Horspool-Sunday (BMHS).

2 Fluxo de funcionamento

Para o funcionamento correto do programa, é necessário um planejamento do fluxo de funcionamento do programa. Ao executar o programa pela linha de comando, o usuário deve passar como parâmetro o número correspondente à estratégia desejada, sendo:

1. Knuth-Morris-Pratt `1`
2. Boyer-Moore-Horspool `2`
3. Boyer-Moore-Horspool-Sunday `3`
4. Força Bruta `4`

Também é necessário que o usuário passe o nome do arquivo de entrada que contém a configuração da entrada.

Portanto, o usuário deve digitar o seguinte comando dentro da raiz do projeto:

```
./tp3 <algoritmo> <arquivo_de_entrada>.txt
```

Após receber os dados da linha de comando, o programa armazena os valores obtidos no arquivo de entrada e chama a função correspondente à estratégia escolhida. Então o retorno da função é escrito no arquivo `saida.txt` na raiz do projeto.

Tendo em vista que a complexidade de tempo e memória são fundamentais para o estudo de algoritmos em geral, após escrever no arquivo `saida.txt` a maior pontuação possível para a sequência de inteiros dada, o programa escreve na saída padrão o tempo real, o tempo de CPU e a quantidade de memória gastas com o processamento.

3 Estruturas de Dados

Para uma maior escalabilidade do software e uma organização mais eficiente, foi implementada uma estrutura de dados destinada unicamente ao armazenamento dinâmico de um vetor de caracteres, possibilitando a leitura de um texto de qualquer tamanho, bem como de um padrão de qualquer tamanho.

```
typedef struct string
{
    char* str;
    long long capacidade;
    long long inicio;
    long long fim;
}string;
```

Uma vez que o problema lida com intervalos do texto onde o padrão deve ser encontrado, os inteiros `inicio` e `fim` permitem essa flexibilidade de modificar a string original sem a necessidade de criar uma nova.

Dentre as operações que envolvem essa estrutura de dados, formando um tipo abstrato de dados, está a realocação do texto ou do padrão. Como o tamanho desses elementos não é conhecido previamente, a realocação tornou-se a solução mais eficiente para lidar com essas informações. Outra operação importante é a desalocação dessa estrutura. Por fim, há a operação de leitura da string a partir de um arquivo dado.

```
string* lerString(FILE* input);
void desalocarString(string*s);
```

Dessa forma, esse tipo abstrato de dados permite uma melhor performance na implementação dos algoritmos, além de otimizar o processo de "corte"no texto dado ao procurar por um padrão em uma das K consultas. Essa abordagem não só melhora a eficiência na manipulação de grandes volumes de dados, como também facilita a gestão da memória, evitando a criação de cópias desnecessárias do texto original. Isso garante que o software possa lidar de maneira mais eficaz com diferentes tamanhos de entradas, mantendo a robustez e a escalabilidade da aplicação.

4 Estratégias de resolução

Como discutido anteriormente, as quatro estratégias implementadas foram: a força bruta, que testa todas as possíveis correspondências entre padrões e textos; o algoritmo Knuth-Morris-Pratt, que utiliza um pré-processamento como abordagem para melhorar a busca; o algoritmo Boyer-Moore-Horspool, que emprega heurísticas que provocam o maior deslocamento do padrao; e o algoritmo Boyer-Moore-Horspool-Sundayl, uma variante do Boyer-Moore-Horspool que permite deslocamentos mais longos, levando a saltos relativamente maiores para padrões curtos.

4.1 Força bruta

Explorar todas as possíveis posições para alinhar o padrão com o texto permite avaliar cada uma e identificar a correspondência exata. No caso do algoritmo de força bruta para casamento de caracteres, essa abordagem examina cada possível posição no texto onde o padrão pode se alinhar e compara os caracteres um a um para verificar a correspondência exata.

Com isso, o fluxo de funcionamento para o texto `abcabab abababaabac` e o padrão `ababaaba` é a seguinte:

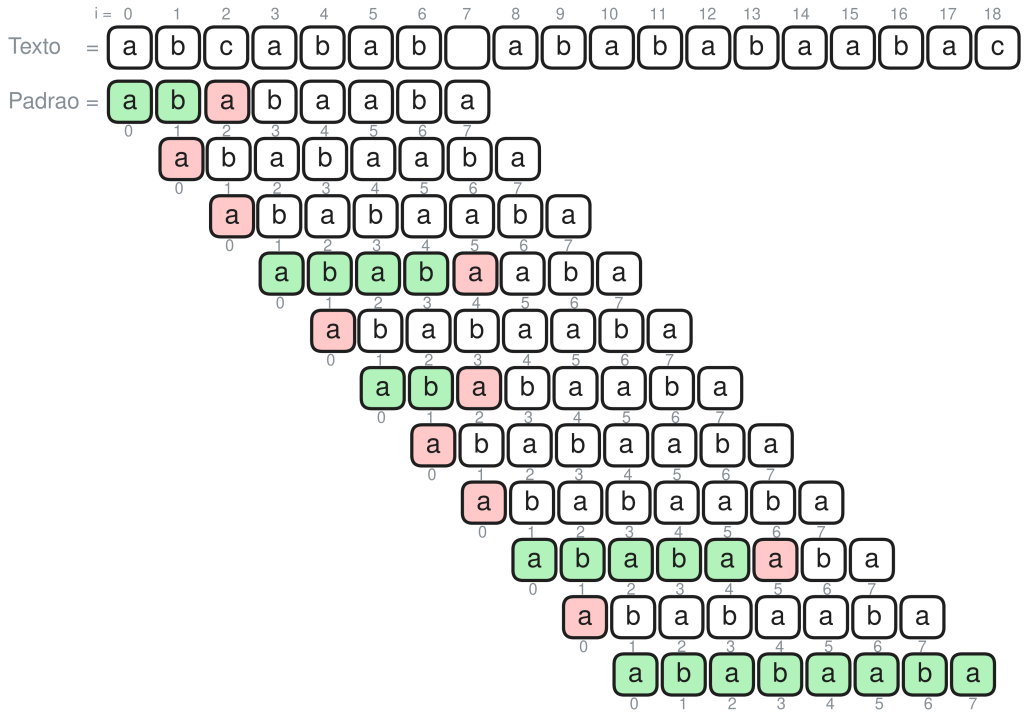

Figura 1: Diagrama de funcionamento da função de força bruta.

A Figura 1 mostra o funcionamento do algoritmo de força bruta. Para o trabalho, a implementação desse algoritmo percorre o texto verificando cada possível posição para a substring até encontrar uma correspondência. Quando uma substring é encontrada nos intervalos especificados, o algoritmo utiliza uma variável de controle para indicar que a substring foi encontrada e a busca é interrompida.

4.2 Knuth-Morris-Pratt (KMP)

Explorar todas as possíveis posições para a busca de padrões no texto pode ser muito custoso, especialmente quando lidamos com textos e padrões com tamanhos grandes, na seção 5 a complexidade de cada algoritmo será analisada mais detalhadamente. Com isso, utilizar algoritmos que realizam o pré-processamento das entradas é uma alternativa eficiente, pois, ao preparar os dados antes da busca, reduzem significamente o número de comparações necessárias durante a execução.

Nesse sentido, o KMP cria uma tabela que armazena informações sobre o padrão para otimizar a busca. Essa tabela, chamada de $Tabela$, para cada posição j do padrão, $Tabela[j]$ indica o comprimento do maior sufixo do padrão $p[0..j]$ que também é um prefixo. Caso haja uma incompatibilidade entre os caracteres durante a busca, a tabela $Tabela$ ajuda a retomar a comparação a partir de um prefixo que já foi validado.

A figura abaixo mostra o fluxo de funcionamento do KMP para a texto `abcabab abababaabac` e o padrão `ababaaba` :

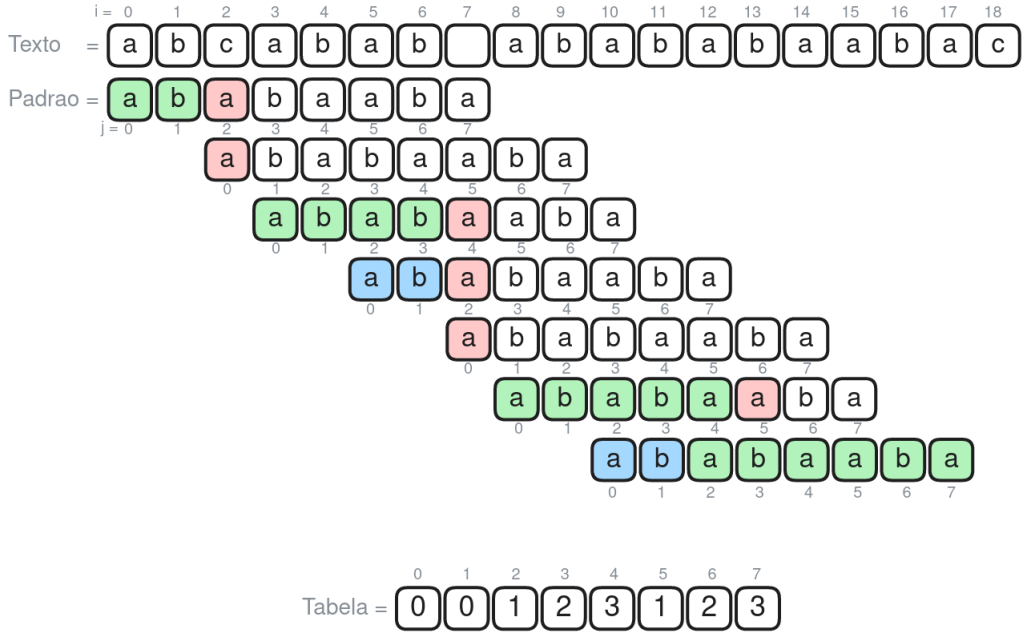

Figura 2: Diagrama de funcionamento da função KMP.

A Figura 2 ilustra o funcionamento do algoritmo KMP. No entanto, como mencionado anteriormente, a implementação do algoritmo para este trabalho percorre o texto até encontrar uma correspondência com o padrão. Para isso, utiliza-se uma variável de controle que indica quando o padrão foi encontrado, resultando na interrupção da busca.

4.3 Boyer-Moore-Horspool (BMH)

Entre os algoritmos que aumentam a eficiência das buscas ao reduzir o número de comparações, o algoritmo Boyer-Moore-Horspool se destaca. Sua principal ideia é realizar a busca do padrão da direita para a esquerda. Além disso, o algoritmo examina o padrão em uma janela que desliza ao longo do texto. Para cada posição dessa janela, ele verifica se um sufixo da janela corresponde a um sufixo do padrão, realizando comparações da direita para a esquerda. Se não houver desigualdade, uma ocorrência do padrão no texto é confirmada. Caso contrário, o algoritmo calcula um deslocamento para a direita, reposicionando o padrão antes de tentar uma nova correspondência.

Para pré-computar o padrão, a tabela de deslocamentos é inicialmente preenchida com o valor m, onde m é o comprimento do padrão. Em seguida, apenas os primeiros m - 1 caracteres do padrão são utilizados para ajustar os valores na tabela de deslocamentos.

O funcionamento do algoritmo BMH para o texto `abcabab abababaabac` e o padrão `ababaaba` é apresentado abaixo:

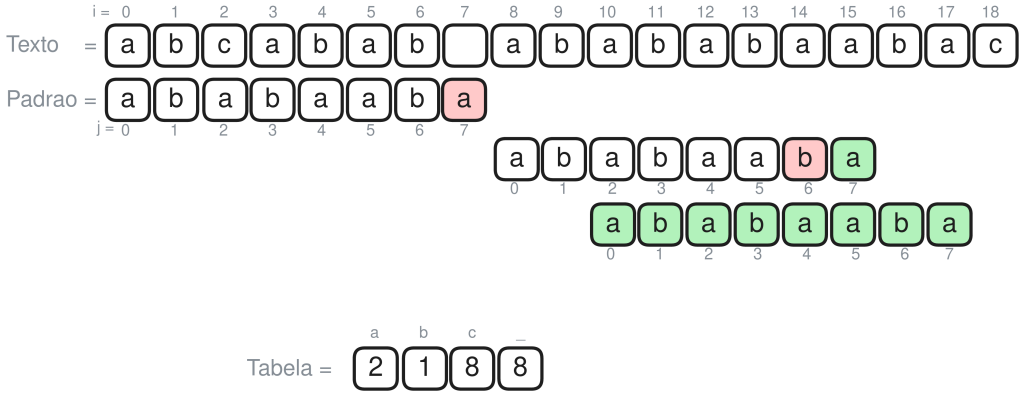

Figura 3: Diagrama de funcionamento da função BMH.

A Figura 5 e a figura ilustram o funcionamento do algoritmo BMH tanto na implementação padrão quanto na implementação específica para o trabalho.

4.4 Boyer-Moore-Horspool-Sunday (BMHS)

O algoritmo Boyer-Moore-Horspool-Sunday (BMHS) é uma variante do algoritmo BMH mencionado anteriormente. Uma das características do BMHS é modificar a forma como a tabela de deslocamentos é endereçada. Em vez de deslocar o padrão utilizando o último caractere do padrão como referência, como ocorre no algoritmo Boyer-Moore-Horspool, essa variante endereça a tabela com o caractere no texto correspondente ao próximo caractere após o último caractere do padrão.

Além disso, a tabela de deslocamentos do BMHS é pre-computada com uma configuração inicial onde o valor de todas as entradas é definido como $m + 1$, sendo m o comprimento do padrão que se deseja buscar. Essa abordagem simplifica o processo de busca e pode reduzir significativamente o número de comparações necessárias, tornando o algoritmo mais rápido e eficaz na localização do padrão dentro do texto.

Dessa forma, para o texto `abcabab abababaabac` e o padrão `ababaaba` , o funcionamento do BMHS é o seguinte:

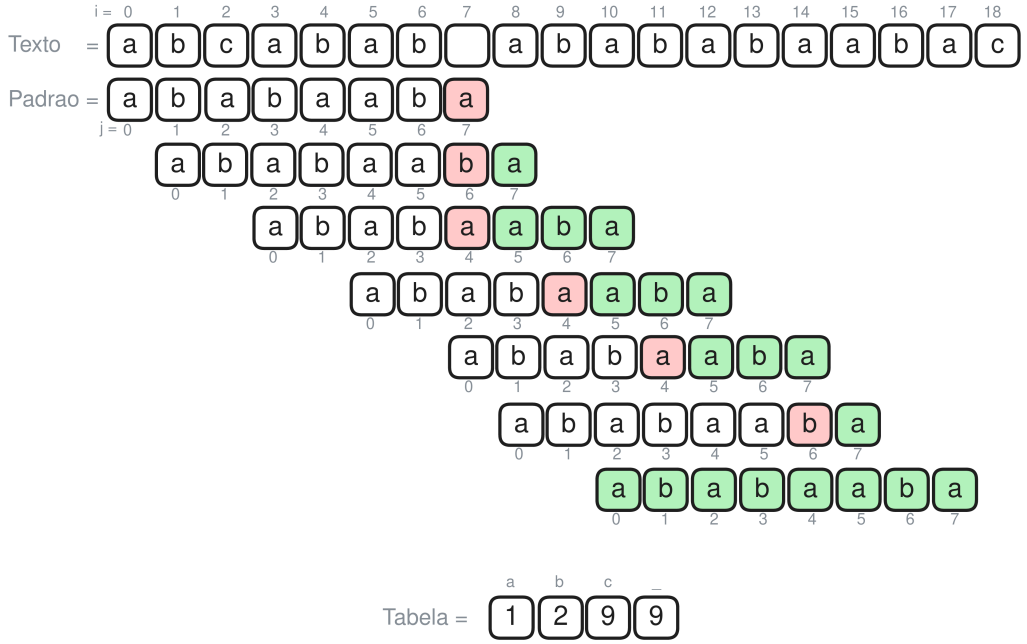

Figura 4: Diagrama de funcionamento da função BMHS.

5 Análise de complexidade

5.1 Força bruta

Como visto na Seção 4.1, o algoritmo mais simples para casamento de cadeias é o algoritmo de força bruta. A ideia, como discutido anteriormente, é testar todas as possíveis posições de casamento entre o texto e o padrão.

Portanto, seja T o texto dado na entrada, de tamanho n, e P o padrão, também fornecido na entrada, de tamanho m. No pior caso, para cada caractere do texto T, comparamos todos os caracteres do padrão P. Assim, a complexidade é $O(nm)$.

Exemplo do pior caso. Considere o padrão $P = \{a, a, b\}$ e o texto $T = \{a, a, a, a, a, a\}$.

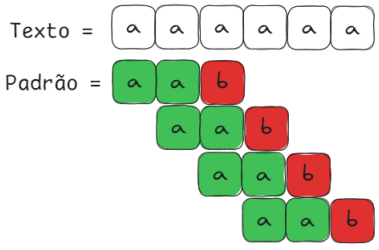

Figura 5: Diagrama de funcionamento do algoritmo Força Bruta no pior caso.

5.2 Knuth-Morris-Pratt (KMP)

Como visto na Seção 4.2, o algoritmo faz, primeiramente, o pré-processamento de P. Logo em seguida, computa o sufixo mais longo no texto que também é o prefixo de P até que ocorra um casamento, caso ocorra. Sendo assim, KMP permite que nenhum caractere seja reexaminado e, por conseguinte, o apontador para o texto nunca é decrementado. Sua complexidade final, então, é $O(n)$.

5.3 Boyer-Moore-Horspool

Como visto na Seção 4.3, o BMH parte do fato de que qualquer caractere já lido do texto a partir do último deslocamento pode ser usado para endereçar a tabela de deslocamentos. Dessa forma, ele endereça a tabela com o caractere no texto correspondente ao último caractere do padrão. Assim, o custo para a criação da tabela de deslocamento é $O(c)$, onde c é o tamanho do alfabeto Σ utilizado. Já na fase de pesquisa, sendo n o tamanho do texto e m o tamanho do padrão, temos que o pior caso do algoritmo é $O(nm)$.

Para exemplificar o motivo de o pior caso ser $O(nm)$, considere o seguinte exemplo:

Seja o alfabeto $\Sigma = \{a, b\}$, o padrão P de tamanho m e o texto T de tamanho n, definidos como $P = ba^{m-1}$ e $T = a^n$. Nesse caso, como o único caractere que difere o padrão do texto é o primeiro caractere b, e ele sempre causa uma colisão com um caractere a, o padrão será deslocado apenas uma posição à direita a cada comparação.

Exemplo com m = 5 e n = 8:

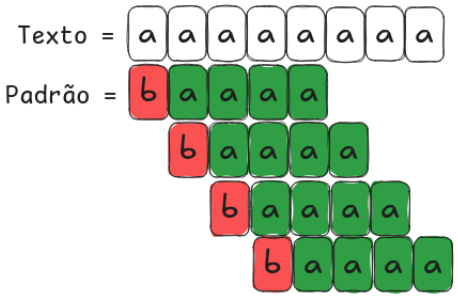

Figura 6: Diagrama de funcionamento do algoritmo BMH no pior caso.

Já o melhor caso é $O(n/m)$, e o caso esperado é $O(n/m)$, se c não é pequeno e m não é muito grande.

5.4 Boyer-Moore-Horspool-Sunday

Como visto na Seção 4.4, O BMHS é uma variante do BMH, com uma alteração muito sutil. Ao invés de considerar o caractere do texto que causou a colisão, ele considera o caractere seguinte a ele. Sendo assim, seu comportamento assintótico é igual ao do algoritmo BMH.

Porém, seu diferencial é que seus deslocamentos são mais longos, podendo, inclusive, ser iguais a $m + 1$, criando saltos relativamente maiores para padrões curtos.

Um bom exemplo seria: dado um padrão P de tamanho m, onde $m = 1$, o seu deslocamento é igual a $2m$ quando não há casamento.

5.5 Testes de execução

Para testar os algoritmos implementados, foi criada uma entrada onde o texto é composto por vários caracteres 'a' e o padrão também consiste em vários caracteres 'a', seguidos por um 'b'. Em outras palavras, eles são idênticos em todos os caracteres, exceto no último do padrão. Com isso, foram obtidos os seguintes tempos de execução, detalhados para cada algoritmo:

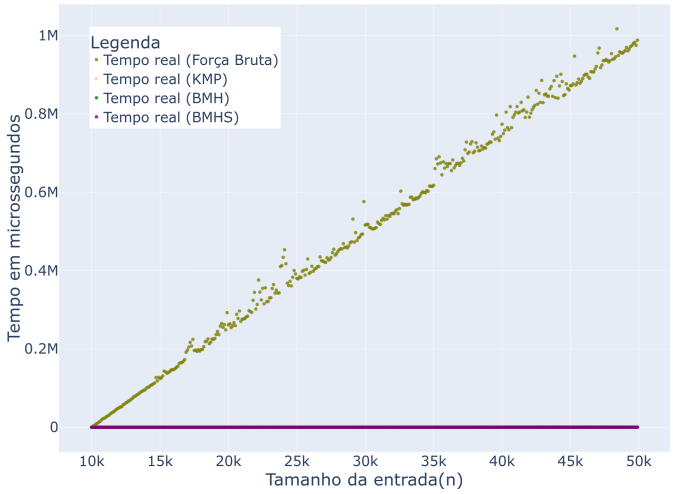

Figura 7: Tempos de todos os algoritmos.

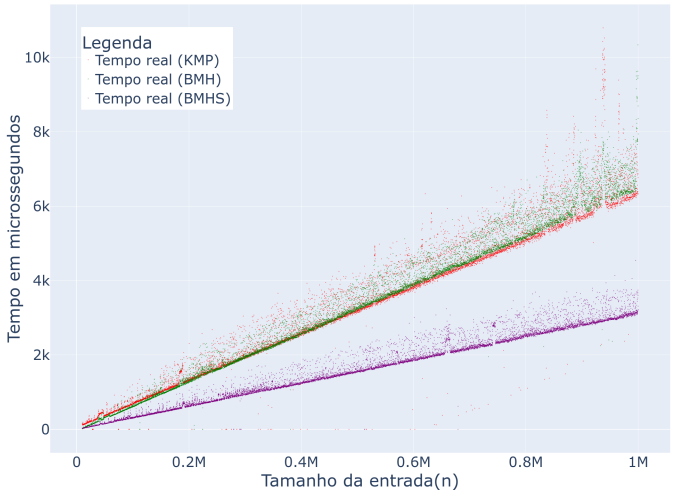

Figura 8: Tempos dos algoritmos sem o força bruta.

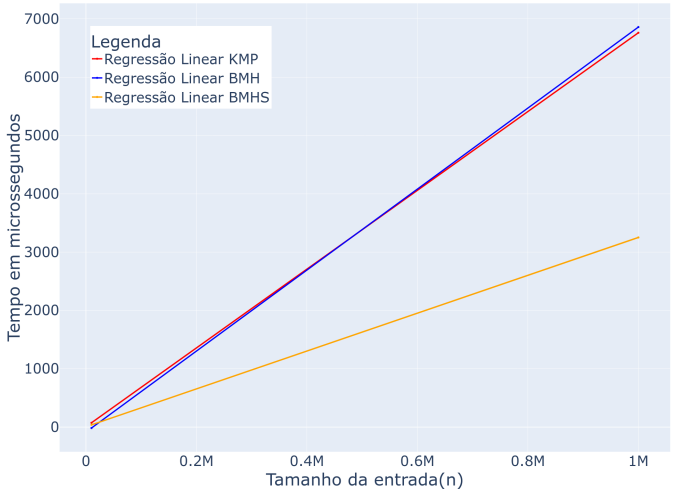

Figura 9: Regressão linear dos algoritmos sem o força bruta.

Em todos os testes realizados, o tamanho do padrão foi de 10^4, enquanto que o tamanho do texto variou. Na Figura 7, é possível observar que os testes realizados com entradas maiores que 50.000 ficariam muito grandes, e o algoritmo de força bruta começaria a se tornar inviável. Enquanto os outros algoritmos mantêm um tempo de execução próximo de $0ms$, o de força bruta atinge cerca de $106ms$. Portanto, foram realizados novos testes com entradas de até 10^6 com os outros algoritmos, excluindo o de força bruta.

Observando as Figuras 8 e 9, é possível perceber que, para os testes realizados, o KMP e o BMH tiveram tempos muito semelhantes, enquanto o BMHS se mostrou superior. No entanto, mesmo com essas diferenças de tempo entre os algoritmos, é importante destacar que todos eles têm complexidade linear, e é possível criar situações nas quais um algoritmo se comporte melhor do que os outros.

6 Conclusão

A partir da exposição do funcionamento de cada algoritmo e das análises feitas sobre eles, é possível identificar tanto aspectos positivos quanto negativos.

Começando pelo de abordagem mais simples, o força bruta, notou-se que, embora seja o menos complexo de implementar, ele rapidamente se torna o mais ineficiente, mesmo com uma pequena entrada.

Por outro lado, foi apresentado o algoritmo KMP (Knuth-Morris-Pratt), que melhorou o limite inferior do problema de casamento exato de cadeias de caracteres, tornando-o $O(n)$. Ele se destaca pela eficiência no pior caso, graças ao seu pré-processamento do padrão. No entanto, sua implementação é mais complexa e requer espaço adicional para armazenar a tabela de prefixos de cada caractere do padrão.

Assim como o KMP, o BMH (Boyer-Moore-Horspool) e o BMHS (Boyer-Moore-Horspool-Sunday) também realiza saltos com base no pré-processamento do padrão. No entanto, suas implementações são menos complexas, e, no caso esperado, o BMH e o BMHS também são mais eficientes. Contudo, no pior caso, eles se tornam mais lento 5.3, tornando-os uma opção menos adequada para circunstâncias em que grandes variações de tempo são intoleráveis.

Conclui-se, portanto, que não há uma decisão definitiva sobre a melhor abordagem; em vez disso, é necessário estudar o problema dado para escolher a estratégia que melhor se adapta à situação. Cada algoritmo possui suas vantagens e desvantagens, e seu desempenho varia dependendo das características do problema e do conjunto de dados envolvidos.

Dessa forma, é evidente a importância de discutir e estudar o problema do processamento de cadeias de caracteres, dada sua vasta aplicação no dia a dia. A evolução dessa área permite avanços significativos em diversos setores tecnológicos, tornando indispensável o aprofundamento nos estudos sobre o tema.

Referências

[1] Nivio Ziviani. *Projeto de algoritmos: com implementações em Pascal e C*. Cengage Learning Edições Ltda., São Paulo, 3ª edição revista e ampliada, 2011.