Universidade Federal de São João Del-Rei - Brasil.
Ciências da Computação
Algorítmos e Estrutura de Dados III

# Trabalho Prático - Casamento de Padrões

**Pedro Garcia Nunes**
**Vitor Luiz Reis do Carmo**

pedrognunes139@gmail.com
vitorluiz.estudo@gmail.com

2023

## 1. Introdução;

O casamento de caracteres, também conhecido como busca de padrões, é uma tarefa fundamental na ciência da computação e processamento de texto. Envolve a busca de um padrão específico de caracteres dentro de uma sequência de texto maior, com o objetivo de encontrar todas as ocorrências desse padrão.

Essa operação desempenha um papel crucial em várias aplicações, como processamento de linguagem natural, análise de texto, pesquisa em bancos de dados, algoritmos de compressão e muito mais. O casamento de caracteres permite que os sistemas identifiquem palavras-chave, termos de busca, trechos relevantes de texto e até mesmo identifiquem erros ortográficos ou de digitação.

Dito isso, o problema deste trabalho é, dada uma sequência de símbolos e um padrão, verificar se esse padrão está nessa sequência de símbolos. Porém, como os símbolos estão em uma pedra redonda, o último símbolo é adjacente ao primeiro e o texto pode ser lido de maneira inversa.

Para execução do trabalho, basta compilar usando o utilitário make, e depois executar da seguinte forma: ./tp3 arquivo algoritmo. Sendo 1 o força bruta, 2 BMH e 3 shift-and.

## 2. Descrição das soluções e estruturas de dados utilizados;

Para resolver o problema foram utilizados 3 algoritmos de casamento exato de padrão: força bruta, BMH e Shift-And que serão discutidos à frente. Porém, existem duas características do problema em específico que requerem ajustes nesses algoritmos que são elas: A pedra é redonda, ou seja, o último caractere no texto é adjacente ao último, e além disso, o texto também pode ser lido de maneira inversa. Para lidar com a característica do texto circular, repetimos o texto no final do próprio texto, e, para padrões que são maiores que o texto, repetimos o texto até o texto seja maior que o padrão, assim, conseguimos emular essa característica, onde o último caractere se conecta com o primeiro. Para conseguir fazer o casamento de forma inversa, invertemos o padrão e executamos novamente o algoritmo de casamento pelo texto. Inverter o padrão é mais barato do que inverter o texto.

O programa inicialmente já foi pensado para fazer o uso de threads. Para isso, lemos os N testes no arquivo e armazenamos em uma estrutura “Tests”. Logo em seguida, criamos uma thread para cada um dos N testes ,ou seja, criamos N threads. Depois, esperamos todas as threads retornarem e escrevemos os resultados no arquivo. Todos os algoritmos que são executados pelas threads recebem a estrutura “FuncArgs” que contém o padrão e o respectivo texto.

### 2.1 Força Bruta:

O algoritmo força bruta para o casamento exato funciona comparando “deslizando” o padrão no texto checando letra a letra da esquerda para direita se houve casamento, caso alguma comparação falhe, move o padrão em 1 caractere e começa de novo. Caso todos os caracteres do padrão forem iguais ao texto naquela posição, houve um casamento e retorna a posição. Esse processo é feito duas vezes, uma para o padrão e outra para o padrão invertido.

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

No BMH, o funcionamento do algoritmo é bem parecido com o da força bruta, porém, ele consegue “deslizar” o padrão por mais de um caractere. Para isso, ele usa um pré-processamento no padrão para criar uma tabela de deslocamentos, essa tabela é usada para quando acontecer um “erro”, ou o caractere do padrão não corresponder ao do texto, o algoritmo consulta a tabela para saber quanto irá deslocar o padrão. O padrão é sempre analisado da esquerda para a direita que é essencial para realizar os deslocamentos.

O algoritmo procura a posição da última ocorrência do caractere incompatível no padrão e, se o caractere incompatível existir no padrão, deslocarmos o padrão para que fique alinhado ao caractere incompatível no texto T.


(imagen do site geeksforgeeks)

No exemplo acima, temos uma incompatibilidade na posição 3. Aqui, nosso caractere incompatível é “A”. Agora vamos procurar a última ocorrência de “A” no padrão. Obtivemos “A” na posição 1 no padrão (exibido em azul) e esta é a última ocorrência dele. Agora vamos mudar o padrão 2 vezes para que “A” no padrão fique alinhado com “A” no texto. As informações de deslocamentos ficam na tabela de deslocamento.

**2.3 Shift-And:**

No algoritmo Shift-And, é utilizado uma abordagem de processamento bit a bit para realizar a busca do padrão no texto. Para isso, inicialmente, o padrão é pré-processado e cada letra é transformada em uma representação binária, o que é chamado de "máscara" , que contém a posição que o caractere se encontra no padrão. Após o pré-processamento é inicializado a etapa de busca, onde um vetor de bits R com todos os bit definidos como 0, exceto o bit mais à esquerda que é definido como 1, realiza a operação AND com a máscara da letra do texto e o resultado dessa operação é colocado no vetor R’, após isso, ocorre o SHIFT no R, mantendo 1 no seu bit mais à esquerda, e o processo é realizado com as letras seguintes do texto. O algoritmo finaliza, quando é encontrado 1 no bit mais à direita do vetor R’ (ou seja, o padrão foi encontrado no texto) ou quando o texto termina e não é encontrado nenhum padrão nele. Exemplo:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| M[t] | 1 | 0 | 0 | 1 | 0 |
| M[e] | 0 | 1 | 0 | 0 | 1 |
| M[s] | 0 | 0 | 1 | 0 | 0 |

Acima observamos a representação binária, também chamada de máscara, da palavra teste. Podemos perceber que a cada ocorrência do caractere ele é marcado como 1 na representação binária, por exemplo, “T” está presente nas posições 1 e 4 do padrão, assim sua representação binária será 10010. Agora veremos como a busca funciona:

| Texto | $(R >> 1)\|10^{m-1}$ | | | | | $R'$ | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| o | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| s | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| t | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| e | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| s | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| t | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| e | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | *1* |
| s | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Acima, vemos que a primeira coluna é o texto e o R é representado pela segunda coluna, já o R’ pela terceira coluna. Assim podemos perceber que o R sempre possui o bit mais à esquerda como 1, e sempre que há um casamento do padrão com o texto, ele shifta 1 para a direita (lembrando de sempre manter o bit mais à esquerda como 1). No R’, é visto sempre o resultado da operação and, do R com as máscaras dos caracteres, podemos analisar também que quando R’ recebe 1 no seu bit mais à direita é porque houve um casamento do texto com o padrão (podemos analisar entre as linhas 4 e 8 da tabela). Nessa implementação do shift-and é necessário que o padrão caiba em uma palavra do computador, caso contrário, pode apresentar comportamento indesejado.

3. **Listagem de Rotinas;**
   **Tests :**
   - **createTests():** Cria a estrutura de testes contendo todos os testes passado no arquivo, usando um vetor de padrões e um de textos. Também contém o número de testes e o nome do arquivo onde os testes foram retirados.
   - **writeTests():** Escreve o resultado dos testes em um arquivo .out com o mesmo nome do arquivo de entrada.
   - **freeTests():** Libera a memória alocada para a estrutura de testes.
   - **execTests():** Executa os testes na estrutura de testes disparando uma thread para cada caso de teste com o algoritmo escolhido.

**StringMatching:**

- **bruteforce():** Executa o algoritmo de força bruta no no texto com o padrão.
- **preProcess():** Faz o pré processamento do padrão para criar a tabela de deslocamento para o algoritmo BMH.
- **BMHSearch():** Faz a busca usando o algoritmo BMH.
- **shif-and():** Faz a busca usando o algoritmo Shif-And.

## 4. Análise de complexidade;

Para a análise de complexidade, veremos separadamente como cada algoritmo se comporta.

### 4.1 Força Bruta;

O algoritmo força bruta como já vimos, funciona deslizando o padrão pelo texto e quando acontece um “não casamento” desliza o padrão em uma posição e começa novamente o processo comparando letra por letra. O pior caso aconteceria se ele fizer a comparação de todo o padrão a cada deslizamento como por exemplo:

PADRÃO: AAAAAAAAAAAAAAAA

TEXTO : AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Nesta entrada específica, ele faria a comparação de todo o padrão a cada deslizamento. Então, sendo M o tamanho do padrão e M o do texto ele faria NxM comparações no pior caso. Logo sua complexidade é O(NxM).

### 4.2 Boyer-Moore-Horspool;

O BMH é bem parecido com o Força bruta em seu pior caso, com a mesma entrada, pois ele consegue deslizar o padrão apenas uma vez pois sempre a próxima letra que causará o casamento será a próxima já que, todas as letras são iguais tanto no padrão como no texto ,logo, não conseguirá fazer nenhum salto e sua complexidade será igual a do força bruta. No pior caso, a complexidade de tempo do BMH também é O(m * n), onde m é o tamanho do padrão e n é o tamanho do texto de entrada.

No entanto, em casos típicos, o BMH tem uma complexidade média muito melhor do que isso. Isso ocorre porque o BMH usa heurística de salto de tabela de caracteres.

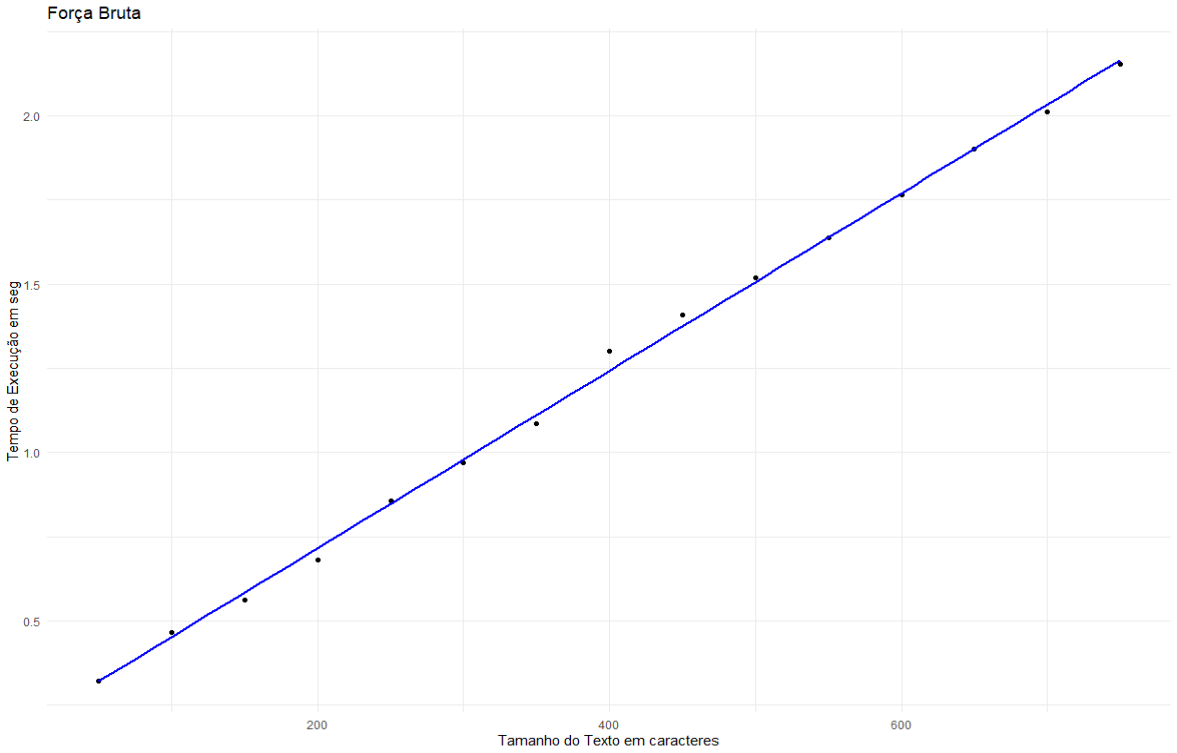
### 4.3 Shift-And;

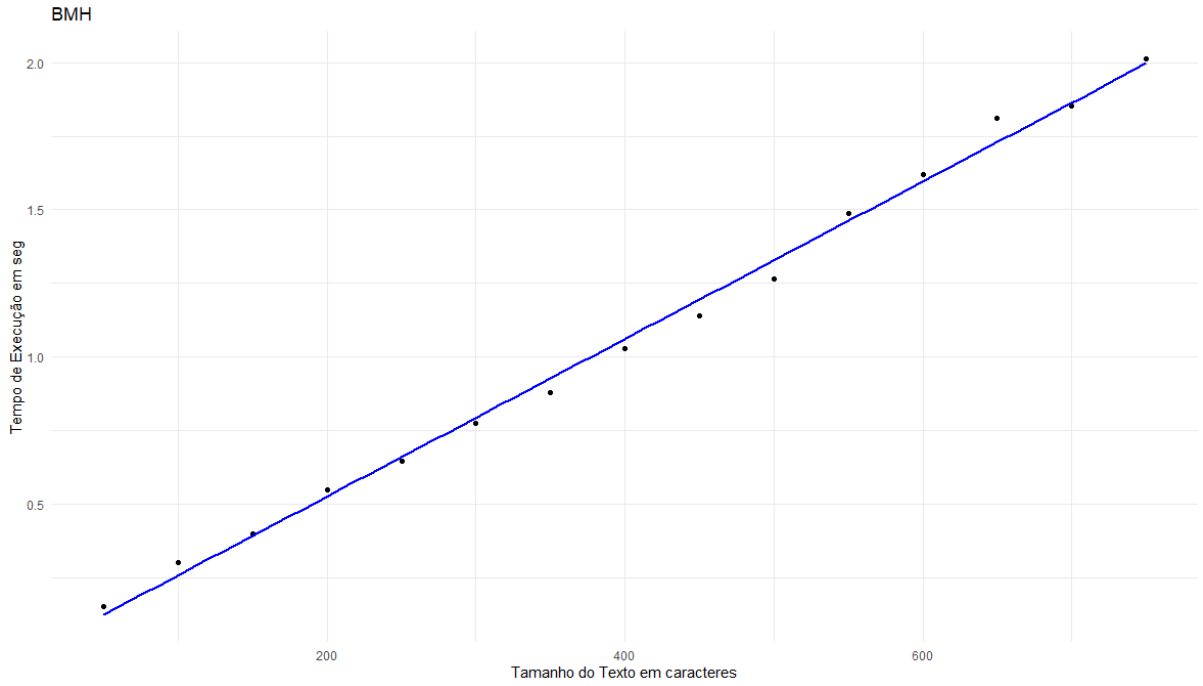
Para a análise de complexidade do algoritmo Shift-And, temos que levar em consideração dois aspectos principais que são o pré-processamento e a etapa de busca. Assim, temos que no pré-processamento o algoritmo requer a construção de um vetor de máscara de tamanho igual ao número de caracteres distintos no padrão, então, temos que o tempo de construção é linear em relação ao tamanho do padrão, ou seja, O(n), onde n é o tamanho do padrão. Na etapa de busca, o algoritmo realiza a operação bit a bit para cada caractere do texto, resultando, também, em uma complexidade de tempo linear O(n), onde n é a quantidade de caracteres do texto. Por fim, podemos concluir que a complexidade do Shift-And é O(n), pois possui complexidade de tempo e espaço linear ao tamanho do texto e do padrão, respectivamente.

## 5. Análise dos resultados;

Para a análise dos algoritmos iremos primeiro simular o pior caso dos algoritmos BMH e Força Bruta. Para isso vamos fixar o tamanho do padrão e variar o tamanho do

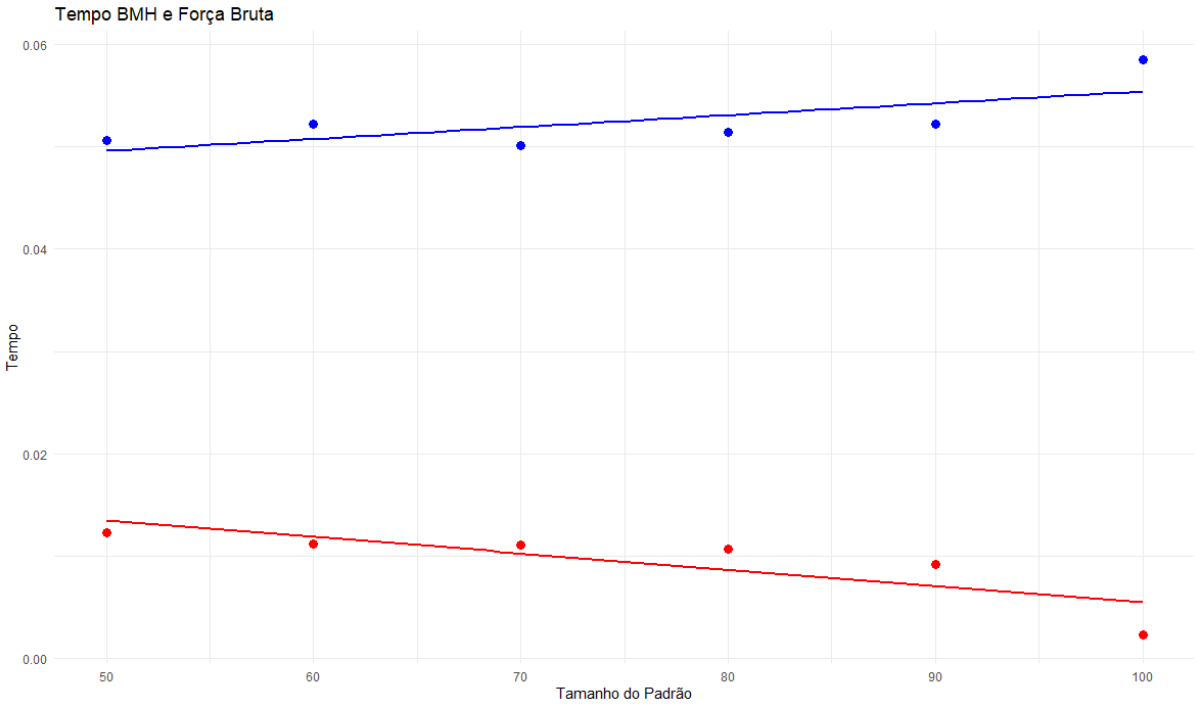
texto. Como os tempos são muito pequenos, adicionamos um pequeno sleep a cada casamento no padrão para melhor visualização.




Podemos perceber que no pior caso os dois algoritmos têm comportamentos iguais, o que já era esperado pela análise de complexidade feita. Em geral, o BMH executa melhor quando o padrão a ser encontrado é maior, pois o algoritmo se beneficia de heurísticas de salto para evitar comparações desnecessárias. Para demonstrar isso, vamos fixar o

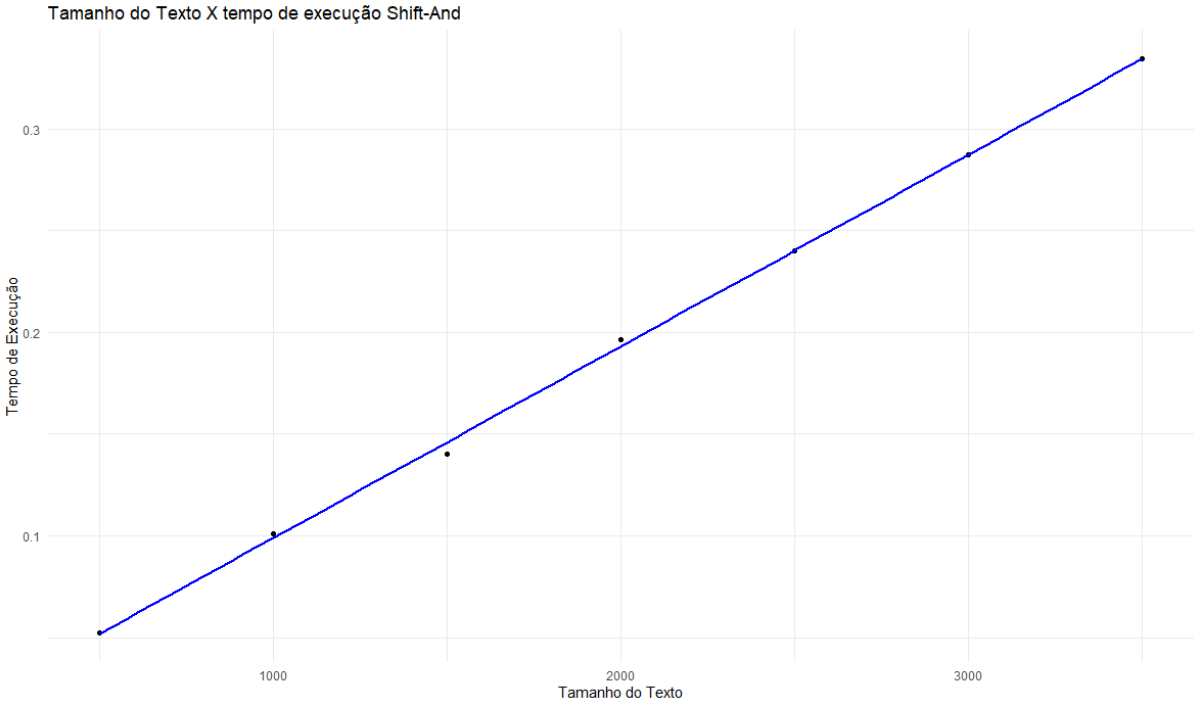
tamanho do texto e variar o tamanho do padrão. O texto e o padrão são quaisquer desde que não seja o pior caso.




BMH Força Bruta

Como esperado, o BMH tende a ser melhor à medida que o padrão cresce pois consegue fazer saltos cada vez maiores, já a força bruta tende a ser pior pois terá que fazer mais comparações já que desliza apenas de um em um.

Já o Shift-And é um algoritmo que se beneficia do paralelismo de bits para fazer operações em tempo O(1). Logo não existe um pior caso, desde que as operações possam ser realizadas em O(1) e o padrão caiba em umas poucas palavras do computador. Então o seu tempo é diretamente ligado ao tamanho do texto.



Podemos perceber o seu comportamento linear O(n) de acordo com o tamanho do texto. No geral, o algoritmo força bruta não é muito útil na prática pois os dois outros algoritmos são melhores em casos arbitrários. Entre o BMH e o Shift-And, O BMH é preferível em casos que o padrão é muito grande, assim ele consegue dar saltos maiores além de que se o padrão precisar de muitas palavras do computador, o shift-and não consegue tirar tanto proveito do paralelismo de bits. Porém, para padrões não tão grandes, O shift-and pode ser ideal, pois seu comportamento é bem previsível e constante, nunca sendo pior nem melhor que O(n).

## 6. Conclusão;

Em conclusão, os algoritmos de casamento de caracteres desempenham um papel fundamental em várias aplicações que envolvem processamento de texto e busca de padrões. Esses algoritmos são projetados para comparar duas sequências de caracteres e determinar sua similaridade ou correspondência. Eles têm ampla utilidade em campos como a análise de texto, a recuperação de informações, a comparação de DNA e até mesmo a detecção de plágio.

Esses algoritmos utilizam diferentes técnicas e abordagens para realizar o casamento de caracteres, como os algoritmos apresentados, o de Levenshtein, algoritmos baseados em árvores, algoritmos de busca aproximada e outros. Cada um desses métodos tem suas vantagens e limitações, o que torna importante escolher a técnica correta para uma determinada aplicação.

Os algoritmos de casamento de caracteres têm um impacto significativo em nossa vida cotidiana. Eles são usados em motores de busca para fornecer resultados relevantes e precisos para consultas de pesquisa. Além disso, são essenciais para sistemas de correção automática de texto em dispositivos móveis e processadores de texto, ajudando a melhorar a precisão e a eficiência da digitação.

## 7. Referências;

https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/