Documentação Trabalho Prático 3 - AEDS III

Adélson O. C. Júnior, Renan Rodrigues Sousa

[1]Universidade Federal de São João del Rei - UFSJ

1. Introdução

Em lugar da terra onde a magia era verdadeira, os habitantes viviam em harmonia com uma variedade de poderes, provenientes de pedras mágicas. Cada pedra possuía inscrições de símbolos e cada sequência de símbolos concedia um poder distinto. No entanto, devido as inscrições serem geradas aleatoriamente, algumas pedras não conferiam nenhum poder.

Com o passar do tempo, os segredos da magia acabaram se perdendo. No entanto, um grupo de pesquisadores descobriu um antigo templo do reino mágico. Após explorarem o local, eles fizeram uma descoberta: um dicionário que relacionava cada conjunto de símbolos a um poder específico. O problema é que as pedras podem ter milhares de símbolos, e verificar manualmente se a pedra possui alguma habilidade é um processo custoso e muito sujeito a erros humanos.

As inscrições de cada pedra são compostas por letras minúsculas, de a até z. Cada sequência de poder possui entre 1 e 10^2 caracteres, enquanto a sequência da pedra possui entre 1 e 10^4 caracteres. As pedras têm formato esférico, e o símbolo representado pelo último caractere é adjacente ao primeiro.

O problema descrito é um desafio fictício que simboliza uma série de problemas reais relacionados ao processamento de caracteres e ao casamento de padrões. Os conceitos abordados neste problema possuem diversas aplicações em diferentes áreas, tais como processamento de texto, detecção de padrões em dados, busca de strings em texto, entre outras. Este trabalho fornece três abordagens para a resolução de problemas de casamento de padrões: o algoritmo de força bruta, o algoritmo de Boyer-Moore-Horspool e o algoritmo Shift-And.

2. Força Bruta

O algoritmo de força bruta é o método mais simples e intuitivo para realizar o casamento de padrão. Essa abordagem consiste em testar todos os elementos do padrão na palavra observada. O primeiro caractere do padrão é comparado com o primeiro caractere da palavra. Se houver uma correspondência, os ponteiros em ambas as sequências avançam. Caso contrário, o ponteiro da palavra é incrementado em um e o do padrão é reiniciado. Esse processo é chamado de shift e é repetido até que ocorra um casamento exato.

No entanto, a simplicidade desse algoritmo vem acompanhada de uma baixa performance, pois a abordagem de força bruta requer testar todas as combinações possíveis. Por isso, em muitos casos, essa abordagem pode não ser a mais adequada.

3. Boyer-Moore-Horspool (BMH)

O algoritmo Boyer-Moore foi introduzido em 1977 com o propósito de melhorar o processo de casamento exato de padrões. Ao contrário da abordagem de força bruta, esse algoritmo realiza a busca da direita para a esquerda. Além disso, o algoritmo utiliza

duas heurísticas para realizar o deslocamento do padrão, tornando o algoritmo bem mais rapido que o força bruta. Posteriormente, em 1980, Horspool propôs modificações no algoritmo Boyer-Moore com o objetivo de simplificá-lo, resultando no algoritmo Boyer-Moore-Horspool.

A ideia fundamental por trás dessa melhoria criada por Horspool era a implementação de uma tabela de valores de deslocamento para cada caractere do padrão. Quando ocorre uma falha no casamento em uma determinada posição, o padrão é movido de acordo com o valor de deslocamento especificado para o caractere correspondente na tabela de deslocamentos. Os deslocamentos são maiores durante a busca, o que contribuí para o aumento da eficiência do algoritmo.

4. Shift-And

O algoritmo Shift-And é uma abordagem mais eficiente em comparação com o método de força bruta. Ele utiliza o conceito de paralelismo de bits para criar uma máscara de bits para o padrão, o que resulta em uma redução do número de operações necessárias. Esse pré-processamento do padrão é realizado para construir a máscara de bits. As operações de manipulação de bits permitem realizar o casamento exato de padrões de forma mais rápida e eficiente.

5. Listagem de Rotinas

bmh.h

- **Tabela de deslocamento - int* shift_table(char* text, char* pattern, int m):** Constroi a tabela de deslocamento para o padrão.
- **Método BMH - int bmh(char* text, int n, char* pattern, int m):** Utiliza o método Boyer-Moore-Horspool para realizar o casamento exato de um padrão.

bruteForce.h

- **Força Bruta - int bruteForce(char* text, int n, char* pattern, int m):** Utiliza o método da Força Bruta para realizar o casamento exato de um padrão.

io.h (Input Output)

- **Criar palavras - char* createString(File *input, int* size, int m, int max):** Extrai o padrão e o texto do arquivo e armazena em um vetor.
-

shiftAnd.h

- **Máscara de bits - int* create_mask(char* pattern, int m, int begin):** É a função responsável por construir a máscara de bits para um determinado padrão.
- **Método Shift And - int shift_and(char* text, char* pattern, int m):** Utiliza o método Shift-And para realizar o casamento exato de um padrão

6. Soluções do Problema

Para solucionar o problema de casamento de padrão, foram implementadas três estratégias em linguagem C. Primeiramente, optamos por utilizar o algoritmo de força bruta devido à sua simplicidade e com o intuito de compará-lo com outras estratégias mais eficientes.

Além disso, escolhemos os algoritmos Boyer-Moore-Horspool e Shift-And como outras estratégias.

Antes de executar qualquer uma das opções, é necessário primeiro ler o arquivo de entrada e extrair as strings de padrão e texto. O arquivo de entrada apresenta a seguinte estrutura: na primeira linha há um número inteiro que representa a quantidade de estradas serão testados. A linha seguinte contém as strings de padrão e texto, nessa ordem. Todos os caracteres são letras minúsculas e as strings estão separadas por um espaço.

Primeiro, o programa realiza a leitura do padrão e chama o Algoritmo 1 com o valor de entrada m igual a zero. Em seguida, o programa lê os caracteres até encontrar um espaço vazio. Como o valor de m é zero, não ocorre a circularidade para o padrão. Em seguida, o texto é lido e o Algoritmo 1 é chamado novamente com o valor de entrada m igual ao tamanho do padrão. O programa lê os caracteres do texto até encontrar uma quebra de linha. Nesse caso, como o valor de m é diferente de zero, o texto também recebe uma parte de tamanho $m - 1$ do início da string, simulando uma string circular.

Algoritmo 1 Algoritmo Criar Palavra

$char : c$
$char : fim$
$int : i \leftarrow 0$
$palavra \leftarrow ALOCAR(Maximo)$
se m igual a 0 **então**
    $fim \leftarrow espaço$
senão
    $fim \leftarrow quebra\ de\ linha$
enquanto $c \leftarrow LER(Arquivo)\ \ e\ \ c \neq fim$ **faça**
    $palavra[i] \leftarrow c$
    $i \leftarrow i + 1$
para $k = 0$ até $(m - 1)$ **faça**
    $palavra[i] \leftarrow palavra[k]$
    $i \leftarrow i + 1$
$tamanho \leftarrow i$
$palavra \leftarrow REALOCAR(tamanho)$

6.1. Solução Força Bruta

Como mencionado previamente, a força bruta trata-se de um método mais intuitivo para realizar o casamento de padrão, sendo este comparar cada elemento da palavra com o padrão.

Para isso, é necessário utilizar um loop 'for' para percorrer o vetor de acordo com o tamanho da palavra. Em seguida, são utilizadas duas variáveis auxiliares, k e j. A primeira guarda a posição atual da palavra no loop de repetição, enquanto a segunda armazena a posição atual do padrão.

É implementado um segundo loop para verificar o número máximo de casamentos a partir daquela posição. Isso é feito incrementando simultaneamente as variáveis k e j. O

loop continua até que ocorra uma falha no casamento, o que pode ser devido a diferenças entre os elementos ou para indicar que o padrão atingiu seu limite.

Por fim, é verificado se o contador j é igual ao tamanho do padrão. Se isso for verdade, a função retorna a posição inicial do casamento acrescida de 1, que corresponde à posição real do caractere. O processo descrito pode ser visualizado no código mostrado no Algoritmo 2.

Algoritmo 2 Algoritmo Força Bruta

```
j, k
para i = 0 até (n − m − 1) faça
    k ← i
    j ← 0
    enquanto Texto[k] igual Padrão[j] faça
        j++
        k++
    se j igual a m então
        Retorna posição
```

Para ilustrar o método da força bruta, vamos considerar uma palavra e um padrão: ”forcabruta”e ”orca”, respectivamente. O objetivo é determinar se há um casamento exato do padrão dentro da palavra.

Na Figura 1, é possível observar o processo de iteração, em que, já na primeira iteração, ocorre uma falha no casamento entre o primeiro caractere do padrão “o” e o primeiro caractere da palavra “f”. No entanto, na segunda iteração, há um casamento entre o primeiro caractere do padrão “o” e o segundo caractere da palavra “o”. A partir desse ponto, o restante do padrão é comparado com os caracteres subsequentes da palavra até que o padrão chegue ao seu fim, havendo casamento para todos os caracteres seguintes.

Portanto, conclui-se que o casamento exato ocorre na posição 2, já que, ao comparar o tamanho do padrão com o número de casamentos, percebe-se que eles são iguais. Assim, é retornada à posição inicial desse casamento.

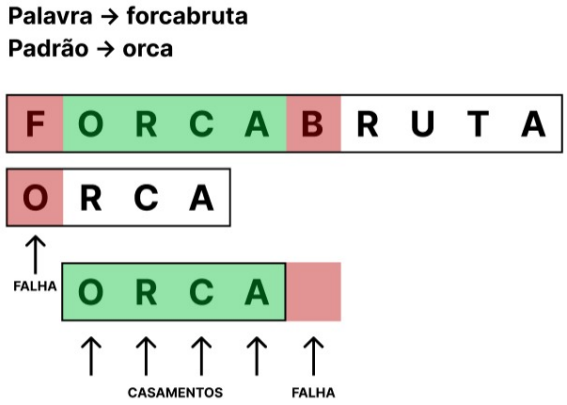

Figura 1. Exemplo de funcionamento do método Força Bruta

Além disso, é importante considerar que a palavra pode estar ao contrário. Desse modo, o método da força bruta deve ser repetido para verificar o casamento inverso do padrão. Isso pode ser ilustrado na Figura 2. Em razão disso, ao aplicar o método da força bruta tanto para o padrão original quanto para o padrão invertido, é possível encontrar possíveis casamentos exatos na palavra.

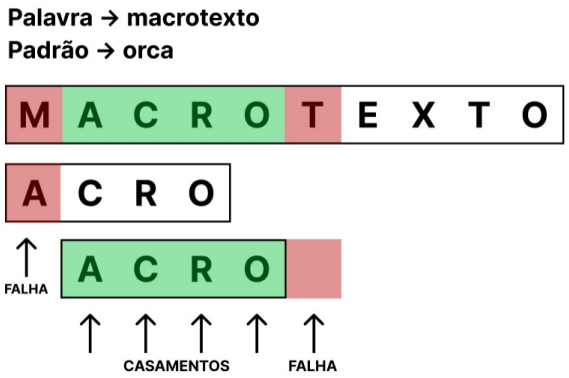

Figura 2. Força Bruta para padrão invertido

6.2. Solução Boyer-Moore-Horspool

Como já mencionado anteriormente, o algoritmo BMH é mais eficiente do que a abordagem de força bruta. A primeira parte do algoritmo consiste na criação da tabela de deslocamento, que armazena os valores de deslocamento (shift) que o padrão deve realizar quando não ocorre uma correspondência de caracteres.

Uma tabela é criada como um vetor do tamanho do alfabeto do problema, neste caso, 26 caracteres. Todos os espaços dessa tabela são inicializados com o valor do tamanho do padrão, representado por m. Em seguida, um contador percorre a tabela, alterando o valor nas posições correspondentes aos caracteres do padrão. Os novos valores são calculados usando a fórmula $m-1-j$, onde m representa o tamanho do padrão e j representa o índice do caractere lido. Após o cálculo dos novos valores, a tabela criada é então retornada. O processo de criação da tabela de deslocamento pode ser melhor visualizado no Algoritmo 3.

Algoritmo 3 Algoritmo Tabela de Deslocamento

$tabela \leftarrow Alocar\ \ Tabela$
para $i = 0$ até $(Alfabeto)$ **faça**
    $tabela[i] \leftarrow m$
para $j = 0$ até $(m - 1)$ **faça**
    $indice \leftarrow indice\ \ do\ \ caractere$
    $tabela[j] \leftarrow m - j - 1$

Com a tabela de deslocamento criada, o algoritmo BMH para casamento exato é realizado. A comparação de caracteres entre o texto e o padrão é feita da direita para a esquerda, usando os valores de deslocamento da tabela para quando não ocorre uma correspondência entre os caracteres. Nessa situação, o valor considerado para o deslocamento é o valor associado ao primeiro caractere do texto que está sendo comparado com

o padrão. O funcionamento do algoritmo pode ser visualizado de forma mais clara na figura 3.

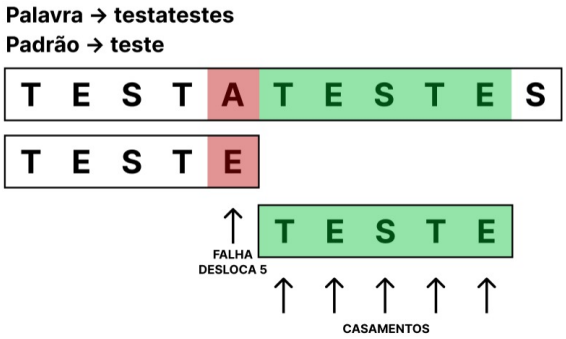

Figura 3. Exemplo de busca no BMH

Enquanto há um casamento entre os caracteres, os índices tanto do texto quanto do padrão diminuem. Caso o valor do índice do padrão, representado por j, chegue a -1, significa que todo o padrão foi percorrido e ocorreu um casamento exato do padrão no texto. A função retorna a posição no texto onde o casamento começa.

No caso de não ocorrer um casamento exato, o algoritmo deve também verificar a possibilidade de casamento com o padrão invertido. Uma lógica semelhante é aplicada nesse teste, utilizando a mesma tabela de deslocamento. A diferença no algoritmo é que o primeiro caractere do padrão é comparado com o último caractere do texto. A medida que o ponteiro do texto diminui, o ponteiro do padrão avança quando há uma correspondência de caracteres. A figura 4 ilustra como é feita essa comparação.

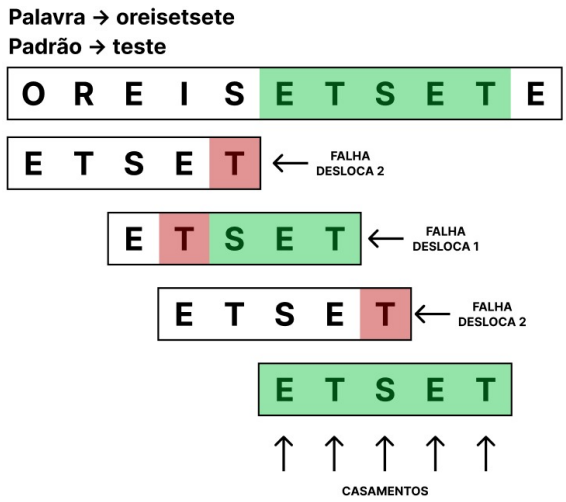

Figura 4. Exemplo de busca invertida no BMH

Se o índice do padrão, ”j”, chegar a ”m”, isso significa que todo o padrão foi percorrido e ocorreu um casamento exato do padrão no texto. A função retorna a posição no texto onde o casamento começa, que corresponde à posição no do texto onde ocorreu o primeiro casamento. Caso não ocorra casamento novamente a função retorna o valor -1 significando que não houve casamento entre o padrão e o texto. O funcionamento desse algoritmo pode ser visualizado no Algoritmo 4.

Algoritmo 4 Algoritmo BMH

Criar Tabela
enquanto $i <= n - 1$ **faça**
    $k \leftarrow i$
    $j \leftarrow m - 1$
    enquanto $j >= 0$ and (Texto[k] igual Padrão[j]) **faça**
        j- -
        k- -
    se j igual a -1 **então**
        Retorna posição
    $i \leftarrow i+$ tabela[índice caractere]

6.3. Solução Shift-And

O algoritmo Shift-And é um algoritmo eficiente para o casamento de padrões que utiliza um conceito de paralelismo de bit. O funcionamento do algoritmo Shift-And ocorre em duas fases. A primeira fase do algoritmo é chamada e pré-processamento.

Durante essa fase, o padrão de entrada é processado para criar uma estrutura chamada de máscara de bits. A máscara de é uma representação do padrão em cada caractere do padrão é mapeado para um bit na máscara. Se o caractere do padrão estiver presente no alfabeto do problema a máscara recebe 1 na posição deste caractere, caso contrário, recebe 0. Um exemplo de máscara pode ser visto na figura 5.

	1	2	3	4	5
M[S]	1	0	0	0	0
M[H]	0	1	0	0	0
M[I]	0	0	1	0	0
M[F]	0	0	0	1	0
M[T]	0	0	0	0	1

Figura 5. Exemplo de máscara

No caso da solução implementada nesse trabalho a máscara pode ser criada tanto para o padrão normal quanto para o padrão invertido baseado em um valor de entrada chamado begin. Se begin for igual a 0 então a máscara deve ser criada a partir da primeira posição do vetor, caso contrario a máscara começa a partir da última posição. O Algoritmo 5 representa a criação dessa máscara.

Algoritmo 5 Algoritmo Criar Máscara de Bits

Alocar espaço para a máscara
para $k = 0$ até $(Alfabeto)$ **faça**
    Inicializa máscara igual 0
para $j = 0$ até m **faça**
    Calcula índice para o caractere
    $mascara[indice] \leftarrow SHIFT$

A próxima fase é a busca do padrão no texto, nessa fase o texto é processado caractere por caractere. A máscara de bits é usada para verificar se o padrão ocorre na posição atual do texto. Para isso é feito uma operação and bit a bit entre a máscara de bits e o valor em bits deslocado que representa o caractere atual do texto. Se a o resultado dessa operação for igual a máscara de bits então houve uma correspondência entre os caracteres do padrão e do texto. A posição onde isso acontece é registrada. Um exemplo desse processo pode ser visualizada na figura 6.

Texto	(R >> 1)|10 ^ m − 1					R′				
H	1	1	0	0	0	0	1	0	0	0
I	1	0	1	0	0	0	0	1	0	0
S	1	0	0	0	0	1	0	0	0	0
H	1	1	0	0	0	0	1	0	0	0
I	1	0	1	0	0	0	0	1	0	0
F	1	0	0	1	0	0	0	0	1	0
T	1	0	0	0	1	0	0	0	0	1
A	1	0	0	0	0	0	0	0	0	0
N	1	0	0	0	0	0	0	0	0	0
D	1	0	0	0	0	0	0	0	0	0

Figura 6. Processo Shift-And

Conforme a sequência de texto é percorrida a máscara de bits é deslocada, um movimento chamado shift. Caso o bit 1 desloque até a última posição do R linha, o valor que é calculado a cada caractere, isso significa que houve um casamento exato de padrão. Caso contrário, não há casamento. O algoritmo pode ser observado no Algoritmo 6

Algoritmo 6 Algoritmo Shift End

Cria a máscara
$R \leftarrow 0$
$R' \leftarrow 0$
para $i = 0$ até n **faça**
    Deslocar R e colocar 1 na esquerda
    $R' \leftarrow R \,\&Mascara[indicecaractere]$
    se R' & 1 **então**
        Retorna posição

7. Análise de Complexidade

Nesta seção, será realizada a análise de complexidade de tempo das soluções propostas neste trabalho. Será considerado o tempo de execução das operações mais significativas, como as comparações, e o pior caso de cada função, a fim de determinar o comportamento assintótico. As variáveis a serem consideradas serão o tamanho do texto (n) e o tamanho do padrão (m).

Inicialmente, abordaremos as funções responsáveis por ler a entrada e escrever a saída em um arquivo:

- **Ler e alocar uma string -** $O(n)$

Nesta função, existem dois laços de repetição consecutivos: um laço while e um laço for. O primeiro varia de acordo com o tamanho da palavra, enquanto o segundo ocorre de 0 até $m-1$, o tamanho do padrão. Sendo assim, o comportamento assintótico do algoritmo é $O(max(n,m))$, que pode ser simplificado para $O(n)$, considerando que n será maior que m.

- **Escreve a saída no arquivo -** $O(1)$

Como nenhuma estrutura de repetição foi empregada, o comportamento assintótico do algoritmo é $O(1)$.

7.1. Força Bruta

No método da força bruta, é utilizada apenas uma função, e como mencionado anteriormente, existem dois loops *for*. O primeiro loop realiza o casamento do padrão normalmente, enquanto o segundo loop faz o mesmo percorrendo o padrão invertido. Para cada um desses loops, há ainda outro loop *while*, cuja complexidade é $O(m)$. Ambos os loops *for* são executados n vezes, resultando em uma complexidade geral de $O(2(nm))$, que pode ser simplificada para $O(nm)$. Portanto, o comportamento assintótico do algoritmo de força bruta é $O(nm)$.

7.2. Boyer Moore Horspool

É necessário analisar separadamente as duas funções desse método.

- **Constrói a tabela de deslocamento -** $O(m)$

No método em questão, são utilizados dois loops *for* consecutivos. O primeiro laço de repetição tem um valor fixo equivalente ao tamanho do alfabeto, o que resulta em uma complexidade $O(26)$, correspondente às 26 letras do alfabeto. Já o segundo loop varia de acordo com o tamanho do padrão e possui uma complexidade $O(m)$, onde m representa o tamanho do padrão.

- **Método BMH -** $O(nm)$

Como primeiro comando, a função chama a função responsável por construir a tabela de deslocamento, conforme analisado anteriormente, com uma complexidade de O(m). Em seguida, há um loop *while* que contém uma declaração *if* e outro loop *while* interno. Considerando que a repetição mais interna ocorre de $m-1$ até 0, e a repetição mais externa ocorre n vezes, temos as respectivas complexidades: $O(m)$ e $O(n)$. Portanto, como o loop externo envolve a repetição do interno, basta multiplicar as complexidades, resultando em $O(nm)$.

Em conclusão, levando em consideração a chamada da função e que os laços de repetição são executados duas vezes, uma vez para o padrão normal e outra para o padrão invertido, obtemos uma complexidade de $O(m, 2nm)$, que pode ser simplificada para $O(nm)$.

É importante salientar que, embora a complexidade encontrada corresponda ao pior caso, na prática, o cenário mais provável é o caso médio, uma vez que é muito mais difícil atingir a situação de pior desempenho. Assim, a complexidade do algoritmo é $O(\frac{n}{m})$ para o caso esperado.

7.3. Shift-And

Novamente, este é um método que implementa duas funções, em que ambas serão analisadas separadamente.

- **Cria uma máscara para o padrão -** $O(m)$

Inicialmente, é utilizado um loop *for* com um valor fixo correspondente ao tamanho do alfabeto, resultando em uma complexidade de $O(26)$. Em seguida, há uma comparação *if* para determinar a construção da máscara. Independentemente do resultado dessa comparação, há um loop *for* aninhado com uma complexidade assintótica de $O(m)$, onde m é o tamanho do padrão. Considerando isso, é possível somar todas as complexidades, resultando em $O(max(26, 1, m))$, ou simplesmente $O(m)$.

- **Método Shift-And -** $O(n)$

Nessa função, é feita uma chamada para outra função que cria a máscara, como já analisado previamente, com complexidade $O(m)$. Em seguida, há um loop 'for' que varia de 0 a n, com complexidade $O(n)$. Esses dois processos são repetidos para o padrão invertido. Portanto, tem-se: $O(2max(m, n))$, que pode ser simplificado para $O(max(m, n))$ ou $O(n)$, considerando que o n é maior que o m.

8. Resultados

Para avaliar o desempenho do programa, foram obtidos o tempo de usuário e de sistema por meio da função 'getrusage', utilizando as funções contidas no arquivo 'tempo.c'. Esses tempos foram utilizados para ilustrar o comportamento assintótico dos métodos empregados.

Devido à eficiência dos métodos para entradas muito pequenas, o tempo de sistema não é significativo. Dessa forma, os gráficos foram construídos apenas com base no tempo de usuário. As entradas utilizadas possuíam tamanho fixo para o padrão e os seguintes tamanhos para a palavra: 20, 50, 100, 150, 200, 250, 500, 1000, 2500 e 5000. No entanto, apenas o tamanho da palavra foi representado nos gráficos.

Na figura 7, é perceptível que o gráfico do método Força Bruta exibe um comportamento linear. Isso ocorre devido ao uso de um tamanho fixo para o padrão, enquanto apenas o tamanho da palavra varia. Apesar disso, ele reafirma a complexidade O(n*m) proposta inicialmente.

Da mesma forma, o gráfico correspondente ao método BMH, figura 8, também é linear, como mencionado anteriormente, pois fixa-se um tamanho para o padrão, variando apenas o tamanho da palavra. É notável que o método BMH apresenta uma melhoria

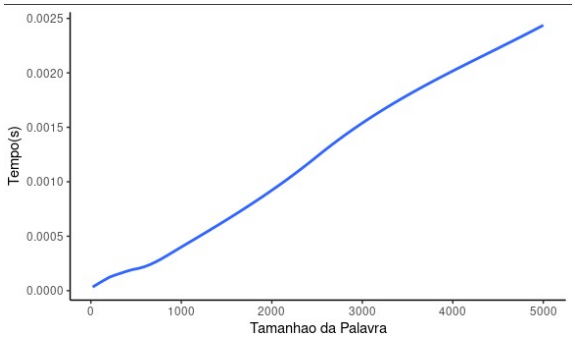

Figura 7. Gráfico do método Força Bruta

em relação ao Força Bruta, uma vez que, para uma mesma entrada, é cerca de 60 por cento mais eficiente (valor aproximado calculado com base nos valores máximos de ambos). Isso ilustra claramente que o método BMH é uma opção muito mais viável em comparação com o Força Bruta.

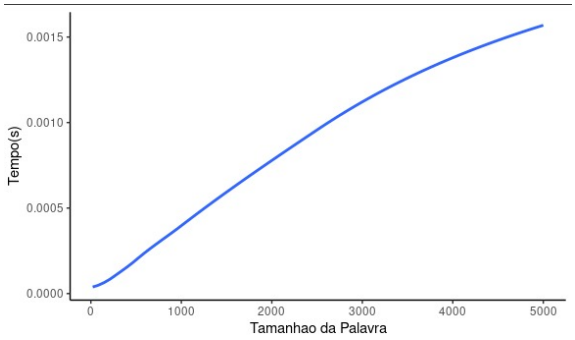

Figura 8. Gráfico do método BMH

Quanto ao método Shift-And, o gráfico evidencia um comportamento linear, Figura 9, apesar de algumas irregularidades, confirmando a complexidade assintótica proposta de O(n). Além disso, em comparação com os outros métodos, os valores de tempo são mais favoráveis, indicando uma execução mais rápida. Com base nisso, pode-se afirmar que o Shift-And é, de maneira geral, o método mais eficiente dentre os três.

Vale destacar que o método Shift-And foi o único que não foi afetado pelo fixamento do tamanho do padrão. Isso se deve ao fato de que, na análise de complexidade, concluiu-se que sua eficiência é linear e depende exclusivamente do tamanho da palavra. Portanto, o método Shift-And demonstra uma vantagem adicional, pois mantém seu desempenho ótimo independentemente do tamanho do padrão.

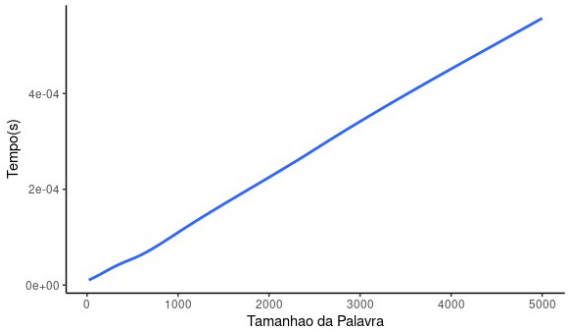

Figura 9. Gráfico do método Shift-And

9. Conclusão

Após avaliar os resultados obtidos e realizar análises de complexidade, podemos chegar a algumas conclusões. Como demonstrado na seção 8, em relação à eficiência geral, a ordem dos métodos é a seguinte: Força Bruta ¡ BMH ¡ Shift-And. A eficácia do método Shift-And é suportada pelo fato de que ele realiza operações separadas com um custo constante de O(1). No entanto, é importante ressaltar que, embora alguns métodos possuam uma complexidade relativamente alta no pior caso, o caso esperado difere significativamente, evidenciando que a eficiência está sujeita à entrada de dados. No entanto, vale destacar que a ordem dos métodos permanece constante, independentemente disso.

É válido destacar que, ao manter o tamanho do padrão fixo, os métodos de Força Bruta e BMH exibiram um comportamento linear, uma vez que apenas o tamanho do texto variava. No entanto, mesmo nessas circunstâncias, o método BMH mostrou-se mais eficiente do que o método de Força Bruta. Embora ambos os métodos possam ter desempenho igual no pior caso, é no caso esperado que se destacam as diferenças significativas entre eles, conforme mencionado anteriormente.

As implementações realizadas neste trabalho proporcionaram uma compreensão mais clara do processamento de caracteres. Os resultados obtidos demonstram um significativo ganho em termos de otimização quando as operações são executadas de forma separada, seja por meio da utilização de uma tabela ou através do uso de uma máscara de bits. Esses resultados reforçam a eficácia dos métodos BMH e Shift-And, evidenciando a importância de utilizar essas abordagens, em comparação com o método Força Bruta.

10. Referências

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein “ALGORITMOS teoria e prática”, 3ª edição. São Paulo GEN LTC 2012

Ziviani, N. Projeto de Algoritmos com Implementações em Pascal e C, São Paulo, Brazil, Cengage Learning, ISBN 13 978-85-221-1050-6, 2010, third edition reviewed and extended, 659 pages (in Portuguese).