UNIVERSIDADE FEDERAL DA BAHIA

IZAK ALVES GAMA
JEAN LOUI BERNARD SILVA DE JESUS
LUIZ GONZAGA SANTANA DOS SANTOS

BUSCA DE PADRÃO EM TEXTO

ESTUDO EXPERIMENTAL

SALVADOR - BA
2023

IZAK ALVES GAMA
JEAN LOUI BERNARD SILVA DE JESUS
LUIZ GONZAGA SANTANA DOS SANTOS

BUSCA DE PADRÃO EM TEXTO

ESTUDO EXPERIMENTAL

Trabalho apresentado ao curso de Estrutura de Dados e Algoritmos II como requisito para obtenção de nota.

Docente: George Marconi de Araujo Lima

SALVADOR - BA
2023

Sumário

1 Resumo

Neste trabalho, iremos descrever e avaliar alguns dos algoritmos mais famosos de busca por padrão: Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), Boyer-Moore-Horspool (BMH) — simplificação do algoritmo BM, descrita por Horspool — e Rabin-Karp (RK).

Para tal, realizaremos alguns experimentos, tomando como métrica o número de iterações necessárias para encontrar todas as ocorrências de um dado padrão aleatório em um dado texto aleatório, em diferentes tamanhos. Ao final, discutiremos a cerca dos resultados obtidos, comparando estes com os resultados teóricos esperados.

2 Algoritmos

2.1 Knuth-Morris-Pratt (KMP)

O algoritmo KMP — Knuth-Morris-Pratt: sobrenome dos criadores do algoritmo — é um algoritmo de busca em uma dada sequência S, tal que $|S| = N$, para encontrar ocorrências de um dado padrão P, tal que $|P| = M$. Ele consiste em pré-processar o padrão, calculando para cada subsequência $P[0, n]$ deste o tamanho $lps(n)$ do maior prefixo que é sufixo de $P[0, n]$, e então utilizar os valores encontrados no pré-processamento como deslocamento, quando a comparação de uma subsequência $S[x, y]$ falhar. [4].

Em termos de tempo de execução, a vantagem deste algoritmo está no fato de que, ao invés de comparar o padrão com uma subsequência de tamanho M em cada posição de S, como ocorre no algoritmo trivial — complexidade $O(N \times M)$ — processamos o padrão com custo $O(M)$ e então fazemos uma busca em S com custo $O(N)$ — ou seja, temos ao todo um custo $O(N + M)$ [4] — voltando o ponteiro apenas $lps(n)$ posições a partir do ponto em que a comparação falhou. Desta forma, quanto maior o tamanho do padrão e menores os valores de $lps(n)$, mais eficiente torna-se a busca.

Para obter os valores de $lps(n)$ em tempo $O(M)$, podemos utilizar o seguinte algoritmo [4]: Seja $L = [\ ]$ uma lista inicialmente vazia que guardará os resultados de $lps(n)$, percorreremos cada caractere $P[n]$, verificando se este é igual ao caractere $P[L[n-1]+1]$. Se sim, então $L[n] = L[n-1] + 1$. Se não, iremos procurar o próximo maior prefixo que é sufixo, utilizando o valor $L[L[n-1]]$ e realizando a comparação novamente. Em caso de falha, repetimos esse processo recursivamente, até chegar no início do padrão.

2.2 Boyer-Moore (BM) e Boyer-Moore-Horspool (BMH)

A primeira versão do algoritmo de Boyer-Moore foi publicada em 1977 por Robert S. Boyer e J. Strother Moore. A ideia básica por trás do seu funcionamento é realizar a pesquisa do padrão no sentido da direita para a esquerda, o que torna o algoritmo mais rápido. No algoritmo original, duas heurísticas são utilizadas para o pré-processamento do padrão a fim de

determinar o deslocamento seu deslocamento no texto: ocorrência e casamento, responsáveis pela criação das tabelas Δ_1 e Δ_2, respectivamente [1].

A primeira tem como objetivo alinhar a posição no texto que divergiu do padrão com o primeiro caractere no padrão que casa com ele. Por outro lado, a segunda leva em consideração o sufixo do padrão anteriormente encontrado no texto para calcular o deslocamento. Assim, para realizar a busca no texto, sempre que necessário o algoritmo consulta os valores dos índices adequados de Δ_1 e Δ_2 e escolhe aquele que provoca o maior deslocamento no padrão.

Em 1980, porém, R. Nigel Horspool desenvolveu uma simplificação do algoritmo de BM ainda mais eficiente que o original, ficando esta conhecida como o algoritmo de Boyer-Moore-Horspool. Sua premissa é baseada na adaptação da heurística de ocorrência para criar uma tabela de deslocamentos com o número de entradas igual ao tamanho do alfabeto Σ utilizado [2]. Dessa forma, cada entrada $d[x]$ da tabela é definida da seguinte maneira: $d[x] := \min\{j : j = m \mid (1 \leq j < m \,\&\, P[m-j-1] = x)\}$.

Embora o algoritmo BMH apresente complexidade de execução igual a BM para o pré-processamento do padrão de $O(m + |\Sigma|)$, as constantes associadas em BMH são significativamente menores devido a não utilização de uma segunda tabela Δ_2. Contudo, em ambos o tempo de execução de busca médio esperado é de $O(n/m)$, um tempo sublinear [5].

2.3 Rabin-Karp (RK)

Criado por Richard M. Karp e Michael O. Rabin, este algoritmo de busca em texto consiste em representar o padrão de tamanho m como um hash bem mais curto, e então compará-lo com os hashs dos subtextos de tamanho m do texto [3]. Isso é vantajoso em termos de tempo de execução, pois em vez de comparar o padrão com um subtexto a cada iteração (comparações em tempo $O(m)$), compara-se os hashs de ambos (comparação em tempo $\Theta(1)$) na maior parte da execução.

O funcionamento do algoritmo ocorre de maneira semelhante à forma ingênua de buscar padrões no texto, mas comparando hashs em vez de comparar strings diretamente, com esse último ocorrendo apenas quando há colisão, pois pode ocorrer falsos positivos. Dado uma função hash $h(x)$, o algoritmo inicialmente calcula o hash do padrão P_m ($v_p = h(P_m)$) e dos primeiros m símbolos do texto ($v_t = h(t[0 \cdots m-1])$. Se $v_p = v_t$, então compara-se P_m com $t[0 \cdots m-1]$, e se forem iguais, então P_m foi encontrado no texto. Caso $v_p \neq v_t$ ou $P_m \neq t[0 \cdots m-1]$, atualiza-se v_t, que valerá $h(t[1 \cdots m])$. O processo se repete até que ou P_m seja encontrado, ou todo o texto foi percorrido sem encontrar P_m.

Em relação a complexidade do algoritmo, o pré-processamento de v_p e v_t ocorre em tempo de $\Theta(m)$, enquanto a atualização de v_t em cada iteração ocorre em tempo $\Theta(1)$. No pior caso, a probabilidade de colisão é 1, ou seja, o algoritmo degenera-se para a busca ingênua e portanto, o tempo de execução é $O(nm)$. No melhor caso, a probabilidade de colisão é 0, ou seja, $v_p = v_t$ só ocorre quando o padrão é encontrado e portanto, o tempo de execução é

$O(n)$. Já para o caso médio, o tempo de execução é $O(n+m)$ [3].

3 Experimento

A ideia deste trabalho é avaliar experimentalmente o comportamento de diferentes algoritmos de busca de padrão em texto. A métrica utilizada é o número de iterações necessárias para encontrar todas as ocorrências de um dado padrão P_m, de tamanho m, em um texto de tamanho n, com todos seus caracteres pertencentes à um alfabeto Σ_k, de tamanho k.
O experimento, que será aplicado à todos os algoritmos, seguirá os seguintes passos:

1. **Geração do alfabeto Σ_k:** Serão escolhidos k símbolos aleatórios para compor Σ_k, com $k \in \{2, 4, 16, 32\}$. Os passos a seguir serão repetidos para cada valor de k.

2. **Geração do texto:** O texto será gerado aleatoriamente a partir de $n = 1000$ símbolos retirados de Σ_k.

3. **Geração do padrão P_m:** Para cada texto gerado, serão gerados padrões P_m a serem buscados no texto, que serão os m últimos símbolos do texto gerado. Será gerado P_m para todos os valores de m, onde $m \in \{2, 4, 6, 8, 10, 12, 14\}$.

4. **Encontrar P_m no texto:** A busca do padrão P_m ocorrerá no texto até que todas as ocorrências sejam encontradas. Como P_m é formado pelos últimos símbolos do texto, então ao menos 1 padrão será encontrado. Serão contabilizados o número de iterações necessárias para pré-processar P_m e encontrar todas as ocorrências de P_m no texto. Para isso, basta contabilizar o número de iterações necessárias para encontrar a última ocorrência de P_m no texto.

5. Com o número de iterações para encontrar todas as ocorrências de P_m em mãos, dividiremos esse valor pelo tamanho do texto n, ou seja, por 1000. Isso resultará em uma média de iterações por símbolo no texto. Essa métrica será utilizada para analisarmos o desempenho dos algoritmos utilizados.

Os resultados encontrados serão a média de 10 repetições dos passos acima para cada valor de k e m. Os resultados serão apresentados em forma de gráficos bidimensionais, onde o eixo x será o tamanho do padrão (m) e y os resultados observados. Cada tamanho do alfabeto (k) será uma linha no gráfico. Cada algoritmo terá o seu próprio gráfico. Com os gráficos em mãos, serão discutidos os resultados observados para cada gráfico gerado, a fim de analisar o desempenho dos algoritmos.

4 Resultados e análises

4.1 Knuth-Morris-Pratt (KMP)

Segue abaixo o gráfico de desempenho do algoritmo de busca em texto KMP, gerado a partir dos dados coletados no experimento, assim como foi proposto na seção "Experimento"

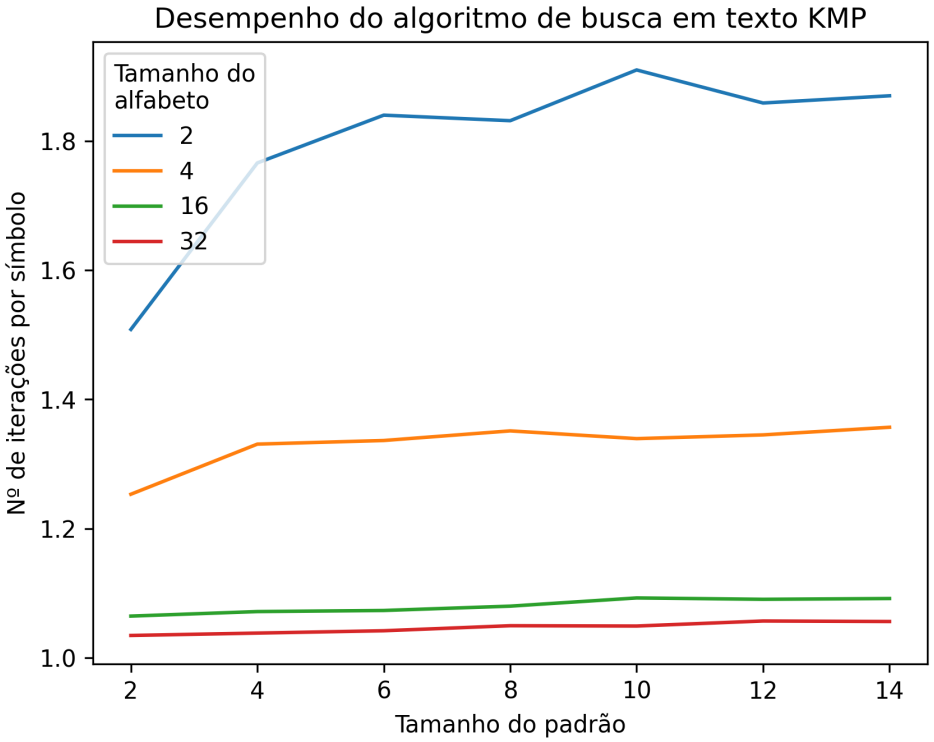

Figura 1: Gráfico de desempenho do algoritmo KMP

Analisando o gráfico, percebe-se que, para os tamanhos de alfabeto $k = 4$, $k = 16$ e $k = 32$, o número de interações por símbolo mantém-se mais ou menos constante ao longo da variação do tamanho do padrão. Por outro lado, para $k = 2$, o número de interações possui um grande crescimento ao longo do tamanho do padrão — possuindo inclusive os maiores números de interações registrados para cada k.

Há de se observar que a quantidade de interações diminui conforme aumenta-se o tamanho do alfabeto. Isto já é esperado pois, quando aumenta-se o tamanho do alfabeto, diminuí-se a probabilidade de se obter valores de $lps(n)$ grandes, tornando assim a busca mais eficiente, pois se reduz o deslocamento à esquerda no texto, significando um menor número de comparações.

Podemos também notar que existe uma grande diferença de interações entre $k = 2$ e $k = 4$. Entre $k = 4$ e $k = 16$ a diferença já é um pouco menor, e entre $k = 16$ e $k = 32$ temos as linhas do gráfico bem próximas. Por esse viés, tendo em mente o funcionamento do algoritmo descrito na seção 2.1, podemos concluir que aumentando o tamanho do alfabeto e o do padrão à infinito, o número de comparações por símbolo irá convergir para 1.

4.2 Boyer-Moore (BM) e Boyer-Moore-Horspool (BMH)

Os resultados encontrados para o algoritmo BM estão apresentados na figura 2. O primeiro ponto que chama atenção no gráfico é disposição das curvas em três patamares distintos, em que o número de iterações atinge os maiores valores na curva $|\Sigma| = 2$ e os menores nas curvas $|\Sigma| = 16$ e $|\Sigma| = 32$ para todo m. Esse comportamento é explicado pelo fato de que quanto menor o alfabeto, maior a probabilidade de que um dado padrão seja casado no texto. Em decorrência disso, o algoritmo passa a ter que realizar mais comparações e produzir deslocamentos menores ao longo do texto, aumentando o número de iterações. Outrossim, $m < |\Sigma|$ introduz a possibilidade de símbolos do texto não estarem no padrão, provocando deslocamento maiores. No entanto, é válido apontar que esse efeito não ocorre de forma significativa entre $|\Sigma| = 16$ e $|\Sigma| = 32$. Uma possível explicação para tal é que o número de combinações para geração do texto passa a ser tão grande que a probabilidade de ocorrer um casamento além do padrão ao final do texto é muito pequena, tornando as curvas praticamente idênticas.

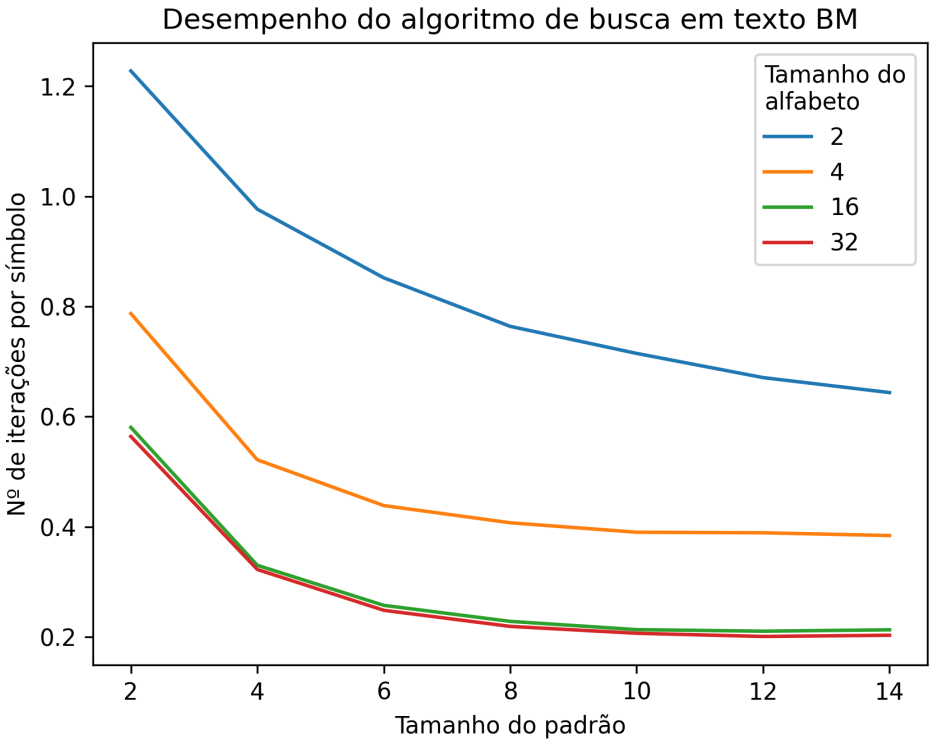

Figura 2: Gráfico de desempenho do algoritmo BM

Ademais, é notório que as 4 curvas exibem comportamentos semelhantes à medida que o tamanho do padrão cresce. Isso se deve à diminuição da probabilidade do padrão ser encontrado no texto ao passo que seu tamanho aumenta, gerando maiores deslocamentos. Esse resultado também justifica o impacto não muito significativo do aumento de iterações para o pré-processamento do padrão à medida que o seu tamanho e do alfabeto cresce. Outro ponto importante é que no geral o número de iterações manteve-se abaixo de 1, o que demonstra o

desempenho sublinear do algoritmo, conforme descrito na seção 2.3. Além disso, os resultados obtidos estão em conformidade com as análises de Boyer e Moore [1].

Em relação ao algoritmo de BMH, os resultados encontrados apresentam diversas semelhanças com o seu antecessor, como exibe a figura 3. No entanto, não é possível deixar de notar uma grande diferença: a curva para $|\Sigma| = 2$ mantém-se aparentemente constante para $m \geq 6$, não decrescendo em momento algum. Isso acontece devido a tabela de deslocamentos do algoritmo só considerar a última ocorrência de cada caractere no padrão (desconsiderando a última posição). Como nesse caso só há no máximo 2, então os deslocamentos tendem a ser muito pequenos, tornando-se um fator ainda mais relevante para valores maiores de m. Logo, como discutido anteriormente, deslocamentos menores geram a necessidade de mais iterações para varrer o texto.

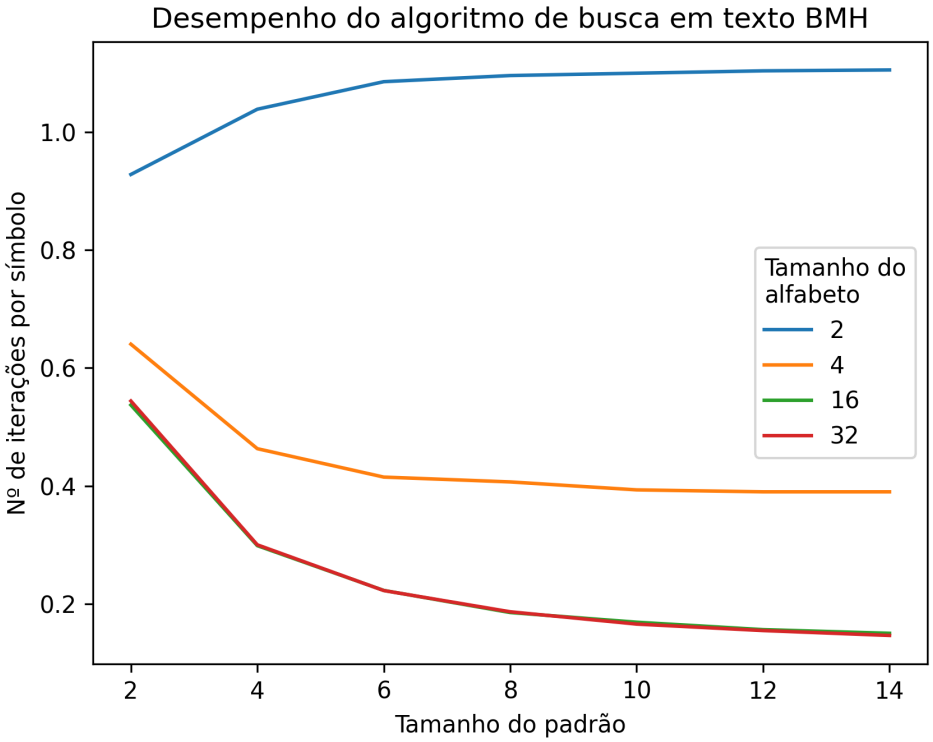

Figura 3: Gráfico de desempenho do algoritmo BMH

Apesar disso, o algoritmo apresenta resultados excelentes para tamanhos de alfabeto maiores, com destaque para a dupla $|\Sigma| = 16$ e $|\Sigma| = 32$ para $m \geq 12$. Dado que o pré-processamento do padrão é simplificado em relação ao algoritmo de BM, para esses cenários o algoritmo de BMH ostenta números de iterações por símbolo ainda menores que o seu predecessor, reforçando ainda mais o caráter sublinear da busca em texto do algoritmo.

4.3 Rabin-Karp (RK)

Segue abaixo o gráfico de desempenho do algoritmo de busca em texto Rabin-Karp, gerado a partir dos dados coletados no experimento, assim como foi proposto na seção "Experimento"

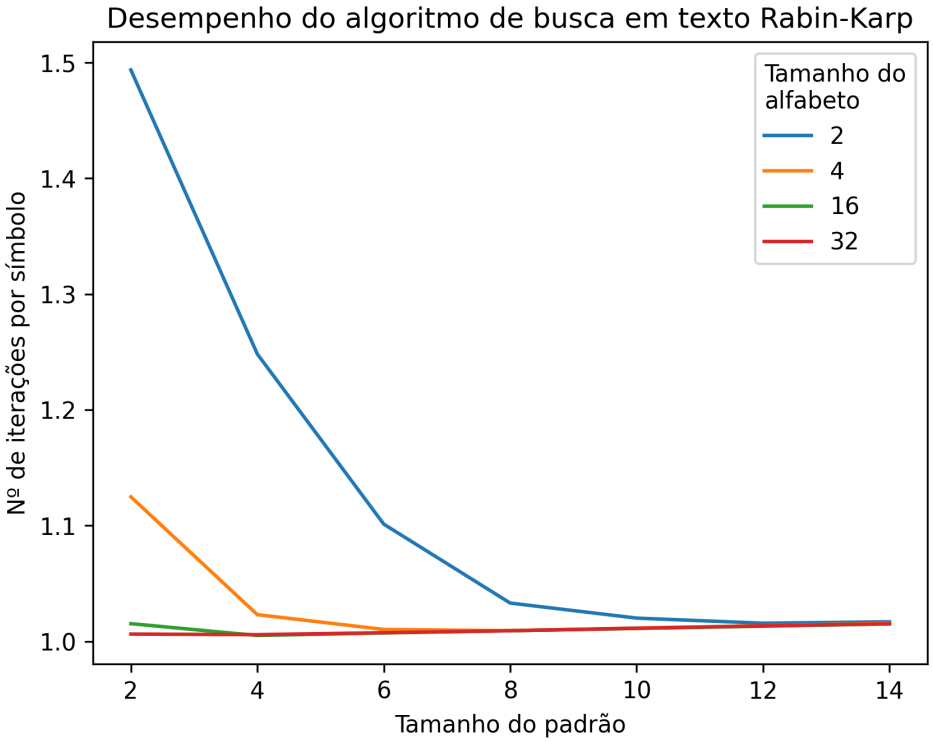

Figura 4: Grafico de desempenho do algoritmo Rabin-Karp

Analisando o gráfico, percebe-se que para padrões de tamanho 2 (m), o número de iterações por símbolo costuma ser os maiores registrados para cada tamanho de alfabeto (k), principalmente para $k = 2$ e $k = 4$, especialmente $k = 2$, que possui uma diferença relativamente grande se comparado com $k = 4$. Isso se deve justamente aos tamanhos do padrão e principalmente do alfabeto, pois devido a baixa variedade de símbolos e o pequeno tamanho do padrão, a probabilidade do padrão formado por símbolos aleatórios se repetir várias vezes em um texto construído aleatoriamente é alta, o que significa maior número de comparações entre padrão e subtexto do texto e consequentemente, mais iterações. O aumento do tamanho do alfabeto corrobora com esse fato, pois para $k = 4$ quando $m = 2$, o número médio de iterações é relativamente muito menor do que para $k = 2$, pois a probabilidade do padrão escolhido se repetir no texto diminui, justamente pela variedade de símbolos no alfabeto. Comparando $k = 4$ com k=16, a diferença de número de iterações é considerável, mas se compararmos $k = 16$ com $k = 32$, a diferença é pequena, indicando que o número de iterações para $m = 2$ e $m = 4$ diminui logaritmicamente a medida que o padrão cresce.

Já quando o tamanho do padrão aumenta, isto é, quando $m = 12$ e $m = 14$, percebe-se que independentemente de k, o número de iterações converge para para um valor ligeiramente acima de 1.0. Para $k = 2$, o número de iterações cai bruscamente, enquanto para $k = 4$ ocorre uma queda menos brusca em relação à $k = 2$. Já para $k = 16$ e $k = 32$ não há grande diferença em relação aos m's menores. Na verdade, a partir de $m = 6$ para $k = 16$ e $k = 32$ e $m = 8$ para $k = 4$, há um leve aumento nas iterações, mas nada muito acima de 1.0. Isso se deve ao fato de que com o aumento do tamanho do padrão, toda vez que há colisão, o número de iterações necessárias para comparar o padrão com uma subtexto do texto aumenta levemente, pois a probabilidade de colisão, é reduzida.

O fato é que a quantidade de iterações por caractere converge para um valor ligeiramente acima de 1.0 com o aumento do tamanho do padrão, devido a diminuição da probabilidade de colisão, pois com padrões aleatórios maiores, a probabilidade de encontrá-los no texto torna-se cada vez menor, ou seja, torna-se necessário, em média, ligeiramente mais do que 1 iteração por símbolo, com o algoritmo percorrendo o texto realizando muito poucas comparações entre padrão e subtexto, aproximando-se do melhor caso, como descrito na seção 2.3.

5 Conclusão

Neste relatório foram avaliados 4 dos algoritmos clássicos na computação para busca de padrão em texto. A análise do algoritmo KMP mostrou que o número de interações diminui ao aumentar o tamanho do alfabeto. Ainda, notou-se que ao aumentar o tamanho do alfabeto e o do padrão ao infinito, o número de iterações por símbolo tende a 1.

Quando a análise dos algoritmos de BM e BMH, comportamento semelhante para o tamanho de alfabeto foi encontrado, com a diferença que esses algoritmos apresentaram número de iterações consideravelmente inferiores. Nesse sentido, constatou-se que o de BMH tem o desempenho melhor em relação ao de BM, com exceção para $k = 2$. Além disso, para um mesmo k, o número de iterações de ambos decresce com o aumento do tamanho do padrão, diferentemente do KMP.

No exame do algoritmo de RK, observou-se que a quantidade de iterações por símbolo tende para um valor próximo a 1 para $m \geq 10$, independente do tamanho do alfabeto. Por outro lado, de maneira semelhante ao que acontece nos algoritmos de BM e BMH, essa quantidade também diminui à medida que m aumenta para $m < 10$, ocorrendo de forma brusca para valores menores de k.

Assim, os resultados dos experimentos realizados demonstraram que o algoritmo no geral com o melhor desempenho foi o de BMH, com resultados fantásticos para tamanhos de alfabeto e padrão maiores, mas para o caso específico de $k = 2$, o algoritmo de BM exibiu o melhor desempenho. Este relatório, portanto, alcançou seu objetivo de avaliar e discutir os resultados experimentais desses 4 algoritmos para busca de padrão em texto.

Referências

[1] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Association for Computing Machinery*, 20(10):762–772, oct 1977.

[2] R. Nigel Horspool. Practical fast searching in strings. *Software: Practice and Experience*, 10, 1980.

[3] Richard M. Karp and Michael O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[4] Donald Ervin Knuth, James H. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6:323–350, 1977.

[5] Nivio Ziviani. *Projeto e Analise de Algoritmos*. Cengage Learnin, São Paulo, Brasil, 2006.