Indexed Pattern Matching Tool (IPMT)

Diogo Oliveira Rodrigues, Gabriela Araújo Britto

10/12/2017

1 Identificação

A equipe é formada pelos alunos Diogo Oliveira Rodrigues e Gabriela Araújo Britto. Diogo e Gabriela implementaram o Suffix Array. Gabriela também implementou o algoritmo de compressão. Diogo construiu a CLI e integrou os algoritmos ao programa. Os dois foram responsáveis por consertar bugs e pelos testes.

2 Implementação

A ferramenta possui dois modos: Indexação e Busca.

No primeiro modo, o programa recebe um arquivo .txt, constrói os vetores necessários do algoritmo Suffix Array e cria um arquivo .idx que contém tais vetores, necessários para o segundo modo. Além disso, tal arquivo passa por um processo de compressão com base no algoritmo LZ77.

No segundo modo, o programa recebe um padrão (ou um arquivo que contém uma lista de padrões) e um arquivo .idx, previamente criado pelo primeiro modo. O arquivo é descomprimido e as informações necessárias para fazer a busca dos padrões no texto indexado são obtidas. O algoritmo do Suffix Array então procede para executá-la.

2.1 Detalhes de implementação

2.1.1 Complexidade de tempo da construção do Suffix Array

Como será visto na seção de testes, a construção do índice é feita significativamente mais rápida que a compressão dele. Desta forma, foi decidido que uma complexidade de tempo $O(n \log^2 n)$ era suficiente e que otimizá-la para $O(n \log n)$ ou $O(n)$ não traria benefícios à ferramenta. Tal processo devolve três vetores de inteiros: a posição de cada sufixo ordenado, o Llcp e o Rlcp. Estes dois últimos correspondem ao tamanho do maior prefixo em comum entre o sufixo da extremidade da esquerda e o da extremidade da direita em relação ao sufixo do meio, informação útil para a busca binária da etapa de busca. Uma representação de tais vetores pode ser vista na Figura 1.

2.1.2 LZ77

Na implementação do LZ77 foi fixado um valor para o tamanho da janela de busca (ls) e da janela de lookahead (la). Os valores escolhidos foram 4096 e

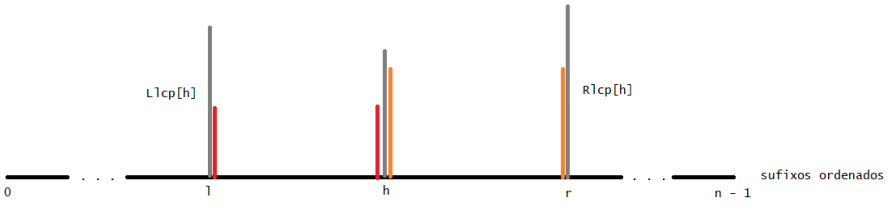

Figura 1: Representação dos vetores de sufixo, Llcp e Rlcp.

32 caracteres, respectivamente. Tais valores foram escolhidos, pois, como será visto na seção de testes, valores maiores da janela de lookahead não melhoram a taxa de compressão, principalmente para textos sem muitas repetições, em que só se precisa de 1 byte para codificar a maioria dos tamanhos de repetição. Além disso, foi escolhido 32 caracteres ao invés de 256 (o equivalente a 1 byte completo), já que a taxa de compressão não é significativamente alterada para textos sem muitas repetições, mas melhora significativamente o tempo da compressão. Todos os testes de tempo de execução que serão apresentados foram feitos com estes valores. Devido à complexidade de tempo da compressão de $O((ls + la * \sigma) * n)$, a ferramenta encontra muitas dificuldades para indexar arquivos maiores que 10^6 bytes, podendo levar mais do que 1 minuto, já que a quantidade de operações pode ultrapassar 10^9.

2.1.3 Representação do texto no arquivo índice

Ao invés de armazenar o texto no índice, foi armazenado um vetor de frequências. Por meio do Suffix Array, pode-se reconstruir todo o texto com esse vetor. Por exemplo, se o primeiro caractere do alfabeto se repete x vezes, sabe-se que nas posições txt[SA[0..x]] tem-se este caractere. Logo, ocupa espaço $O(\sigma \log n)$.

2.1.4 Representação em caracteres dos três vetores de inteiros

Para fornecer o índice e o vetor de frequência para o LZ77 comprimí-los, devido à forma como ele foi implementado, necessita-se que tudo esteja representado como um vetor de caracteres. Em C++, inteiros possuem 4 bytes e, portanto, necessitariam de 4 caracteres para sua representação. Entretanto, o maior valor desses 4 vetores é no máximo $n-1$, em que n é o tamanho do texto. Portanto, ao transformar os inteiros em caracteres, eles são codificados com apenas $\lceil \frac{\lceil \log_2 n \rceil}{8} \rceil$ bytes cada.

2.1.5 Composição do arquivo índice

Por fim, o arquivo índice é composto resumidamente por:

- n = tamanho do texto que origina o índice
- la = tamanho da janela de busca
- ls = tamanho da janela de lookahead
- array = todos os 4 arrays citados previamente, após o processo de compressão do algoritmo LZ77.

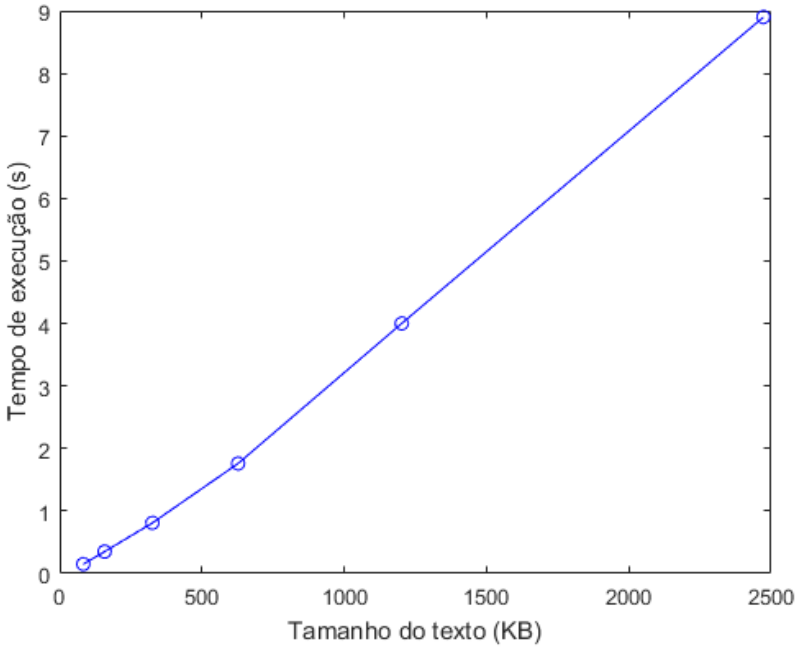

Figura 2: Construção do Suffix Array

3 Testes e Resultados

Os testes foram feitos em um computador com sistema operacional macOS High Sierra v10.13.2, processador 2.4 GHz Intel Core i5, memória 8GB 1600MHz DDR3 e Flash Storage. O compilador utilizado foi o GNU GCC v7.2.0. Para realizar a comparação em relação à taxa de compressão, a ferramenta nativa Archive Utility do macOS foi utilizada.

3.1 Tempo de execução vs. Tamanho da entrada

Para analisar o tempo de execução em relação ao tamanho da entrada, foram selecionados vários arquivos com textos em inglês de tamanhos variados.

3.1.1 Construção do Suffix Array

Através da Figura 2, pode-se notar como o tamanho do texto afeta o tempo de execução. Pode-se observar que o tempo de execução cresce e que o comportamento do gráfico obedece ao gráfico da complexidade assintótica de tempo do algoritmo de $O(n \log^2 n)$. Uma curva com inclinação quase constante, porém com uma inclinação que cresce levemente, devido ao fator polilog.

3.1.2 Compressão do índice

Através da Figura 3, pode-se notar como o tamanho do texto afeta o tempo de execução. Pode-se observar que o tempo de execução cresce e que o comporta-

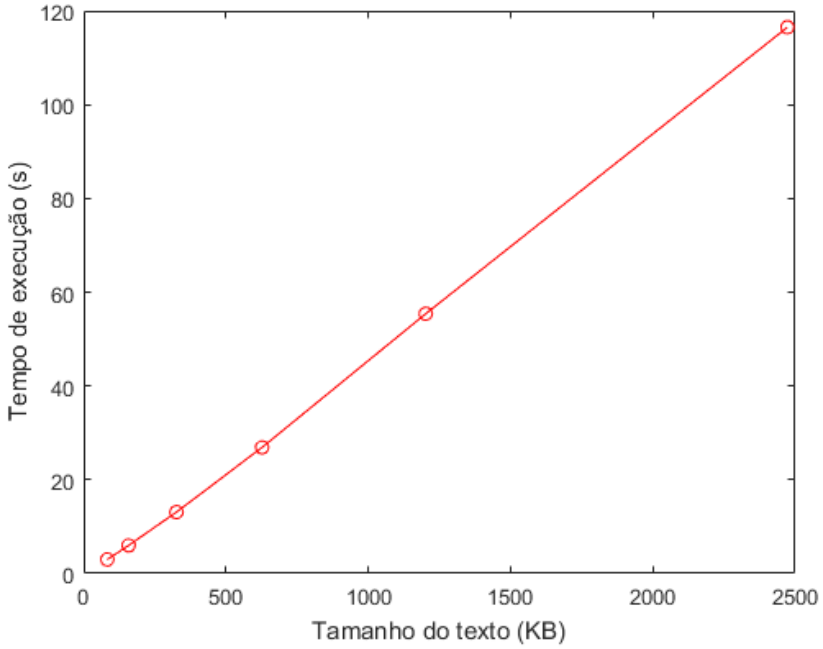

Figura 3: Comportamento da compressão do LZ77

mento do gráfico se assemelha a uma reta, indicando uma linearidade com tal parâmetro. Este comportamento é esperado, dada a complexidade assintótica do algoritmo. Outra coisa que pode ser observada é que o tempo de compressão é muito mais elevado que o tempo de construção. Isto porque além do tamanho do texto, o tempo de execução é afetado pela necessidade da construção da FSM para fazer o casamento da janela de lookahead com a janela de busca e do casamento em si, para cada deslocamento da janela através do algoritmo Aho-Corasick.

3.1.3 Descompressão do índice

Através da Figura 4, pode-se ver que a mesma análise sobre o comportamento do gráfico da compressão pode ser feita aqui. Além disso, podemos notar que esta etapa é feita em muito menos tempo do que a etapa de compressão. Isto se deve ao fato de que aqui não é necessário a construção de FSM's nem de casamentos.

3.1.4 Busca através do Suffix Array

Para identificar como o tamanho do texto afeta o algoritmo, precisa-se lembrar sua complexidade assintótica. Ela é $O(m + \log n)$, em que m é o tamanho do padrão e n é o tamanho do texto. Para evidenciar a parcela que depende do tamanho do texto, o padrão buscado foi pequeno, de tamanho 6. Na verdade, um arquivo contendo 50000 padrões com tamanho médio 6 foi passado como argumento à ferramenta. Isto foi necessário pois a busca de padrões pequenos

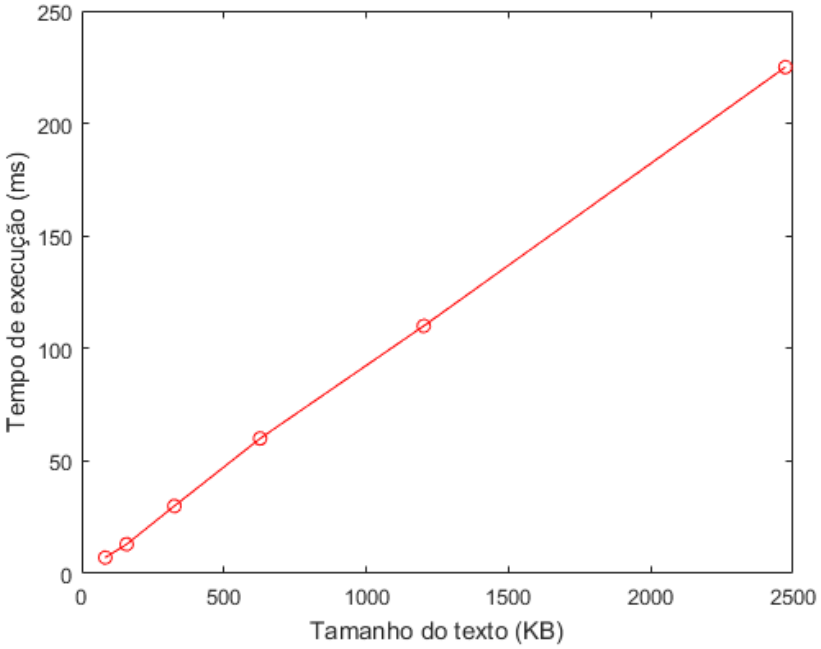

Figura 4: Comportamento da descompressão do LZ77

ocorre em ordem de grandeza menor que 10^{-5} segundo nos textos de teste, o que acarreta em grandes flutuações e instabilidade na medição de tempo. Portanto, ao aumentar em 50000 a quantidade de padrões, uma estabilidade maior foi adquirida e os resultados estão presentes na Figura 5. Fica claro o comportamento logarítmico deste estágio da ferramenta, como esperado ao fixar o tamanho do padrão e aumentar o tamanho do texto.

3.2 Tamanho do arquivo vs. Tamanho do índice vs. Tamanho do índice comprimido

Neste teste, é feita a relação entre tamanho do índice e de seu tamanho comprimido através do algoritmo LZ77 implementado, com as janelas fixadas nos valores apresentados na seção 2.1.2. Pode-se ver, na Tabela 1, os resultados. Além disso, pode-se ver a taxa de compressão da ferramenta nativa do macOS. É observável que a compressão do LZ77 não é extremamente ruim para arquivos pequenos, mas conforme o tamanho aumenta, seu desempenho fica horrível. Tal queda no desempenho pode ser atribuída a alguns fatores como a diminuição na quantidade de repetições, dado que ao aumentar o arquivo, aumenta a quantidade de possíveis elementos nos vetores (0 à $n-1$), diminuindo então a chance de repetições dentro de uma janela de tamanho fixo. Entretanto, tal fator pode não ser tão claro e talvez uma análise mais aprofundada e complexa sobre o conteúdo do arquivo do índice seja necessária.

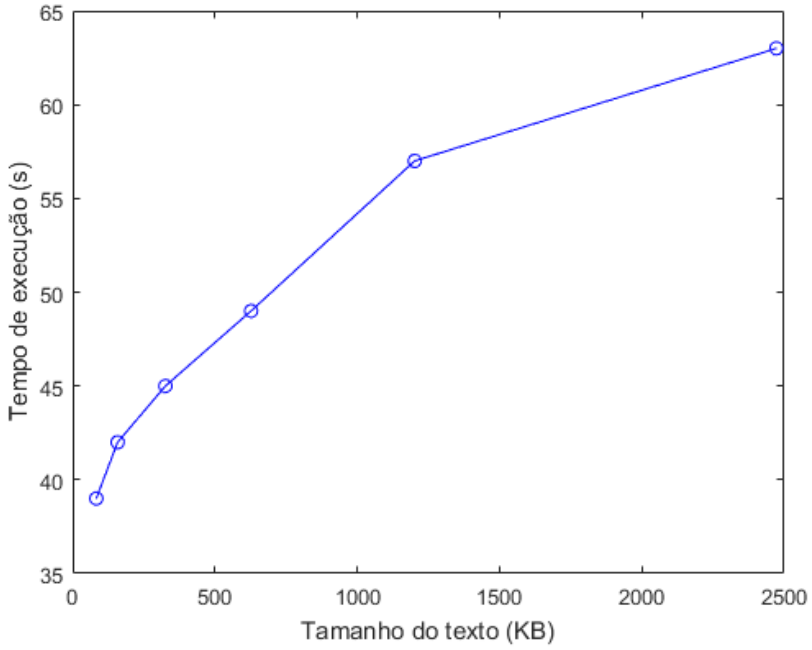

Figura 5: Comportamento da busca do Suffix Array

Texto	Índice	Tam. relativo (LZ77)	Tam. relativo (Utility Archive)
83 KB	730 KB	67 %	42 %
158 KB	1388 KB	70 %	44 %
326 KB	2867 KB	75 %	45 %
627 KB	5508 KB	80 %	47 %
1202 KB	10507 KB	84 %	48 %
2473 KB	21739 KB	86 %	49 %

Tabela 1: Informações sobre a taxa de compressão

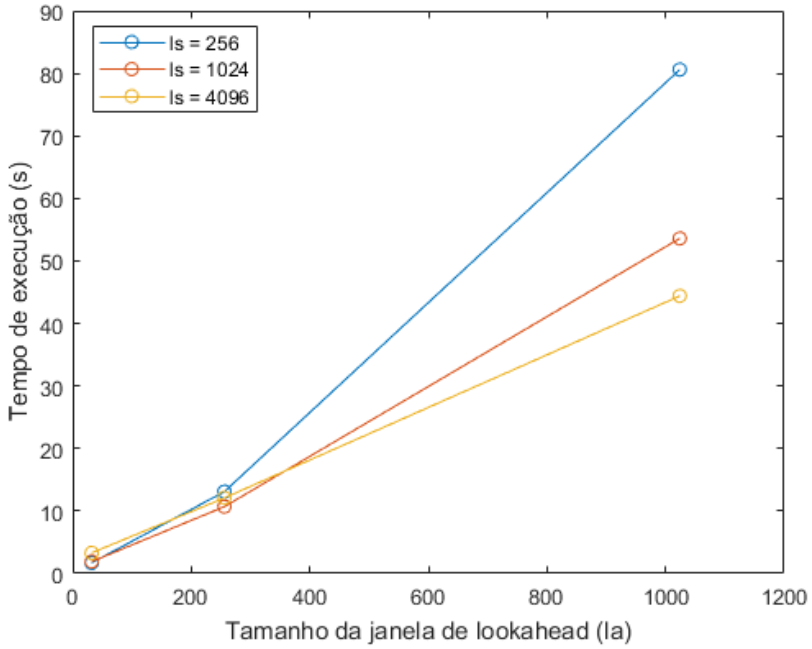

Figura 6: Comportamento da codificação de acordo com a janela

la	32	256	1024
$ls = 256$	618 KB	617 KB	823 KB
$ls = 1024$	638 KB	636 KB	796 KB
$ls = 4096$	531 KB	527 KB	658 KB

Tabela 2: Tamanho do índice comprimido de acordo com a janela

3.3 Tempo de execução vs. Tamanho da janela

Neste teste, um arquivo de texto de proteína de 100 KB foi utilizado. O objetivo é analisar a relação do tempo de execução conforme aumenta-se a janela tanto de busca quanto de lookahead do algoritmo de compressão. Pode-se notar, na Figura 6, que para tamanhos pequenos da janela de lookahead, alterar o tamanho da janela de busca não altera o tempo de execução. Este fato, aliado ao que será apresentado na próxima subseção, irá servir como base para a escolha apresentada na seção 2.1.2.

3.4 Tamanho do índice comprimido vs. Tamanho da janela

Ao executar a construção do Suffix Array, o índice construído possui tamanho 905 KB. O resultado da compressão para vários tamanhos de janelas estão apresentados na Tabela 2. Com esta tabela, nota-se que as melhores taxas de compressão são observadas para uma janela de lookahead pequena e uma janela de busca grande.

4 Conclusões

De posse de todos estes resultados, pode-se concluir que a ferramenta teve seu desempenho bastante limitado pelo tempo de compressão do LZ77. Com algoritmos de compressão melhores, o Suffix Array demonstra ser uma ótima opção para indexação de textos e busca de padrões. Foi visto que os algoritmos obedecem suas complexidades assintóticas e que o Suffix Array é capaz de indexar e procurar em arquivos bem maiores do que os utilizados nos testes, em um tempo razoável. Entretanto, o LZ77 tem um desempenho bastante rudimentar e não é capaz de acompanhar o desempenho do Suffix Array.