Relatório Ferramenta ipmt

Mário Victor Gomes de Matos Bezerra

1. Implementação

A ferramenta **ipmt** desenvolvida neste projeto tem como objetivo de atender os seguintes requisitos: ser capaz de indexar um texto para que seja possível realizar busca exata veloz no texto, comprimir e descomprimir arquivos de texto.

Quanto a **indexação**, a ferramenta recebe um texto como entrada, o qual vai ser indexado usando o array de sufixos. Esse processo resultará em um array de sufixos, L_LCP, R_LCP e um vetor com a frequência de todos os caracteres do texto, os quais serão armazenados em um arquivo *arquivo.idx*.

A partir de um *arquivo.idx*, é possível realizar uma **busca exata** de um ou mais padrões no texto, sendo possível definir quais padrões vão ser utilizados a partir do comando ou de um arquivo *patterns.txt*. A ferramenta vai recriar o *arquivo* de texto a partir da frequência de caracteres armazenadas anteriormente e realizar uma busca binária no array de sufixos.

Além disso, é possível comprimir e descomprimir arquivos de texto com o algoritmo ***LZ77***. A ferramenta recebe um *arquivo.txt* como entrada e retorna um *arquivo.txt.idx* comprimido e vice-versa. A implementação tem uma limitação de apenas levar em conta Extended ASCII, apenas para facilitar na hora de criar a máquina de estados.

1.1 Otimizações

Com o intuito de otimizar as operações referentes a *string*, foi utilizado, onde possível, *string_view*, que nada mais é do que um ponteiro para o início de uma posição de memória e seu comprimento. Isso permite o uso do método *substring()* sem nenhuma perda de performance, já que não será necessário copiar a *substring* para um novo local na memória uma vez que a *string_view* apenas vai apontar para o início da *substring* e guardar o seu comprimento

1.2 Array de sufixos

A diferença mais notável é o uso do **lcp (longest common prefix)** para a construção do L_LCP e R_LCP. Além de sua construção ser O(n) a partir do *algoritmo de Kasai*, ele permite que consigamos o lcp entre dois sufixos consecutivos. Tendo isso em vista, é possível descobrir o lcp de dois sufixos não consecutivos A e B pegando o menor lcp dentro do intervalo [A,B-1], ou seja *min(lcp(A), lcp(A+1) … lcp(B-2), lcp(B-1))*.

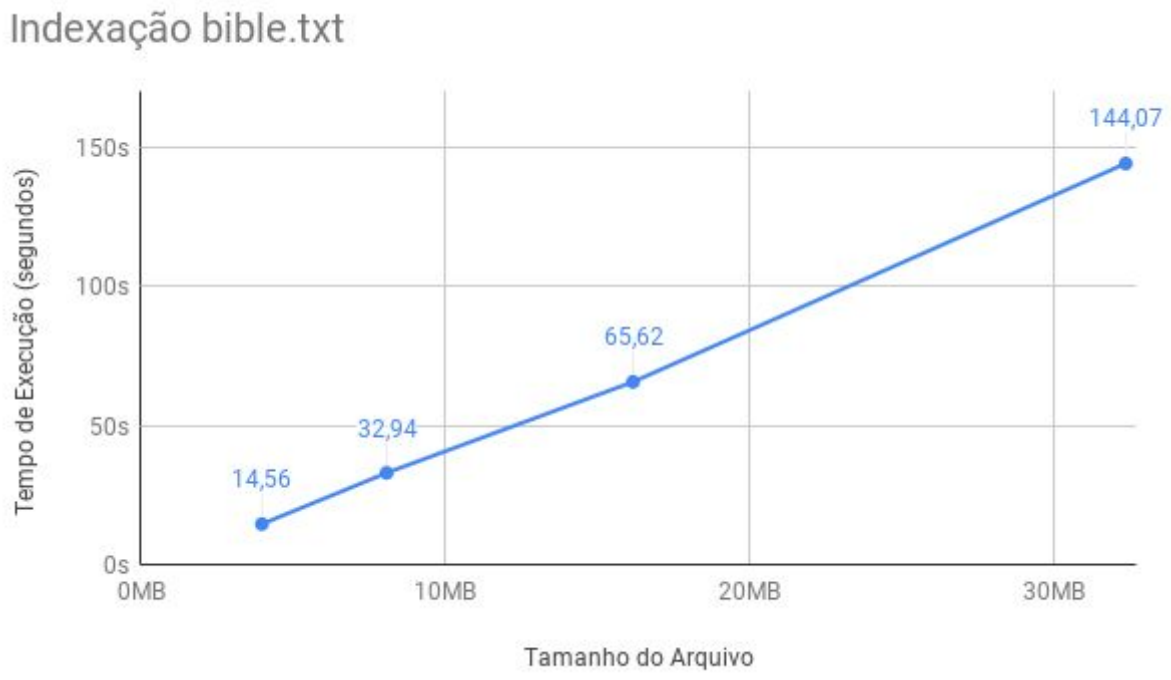
1.3 LZ77

O algoritmo de compressão faz uso de uma janela de busca de tamanho 512 e *look-ahead* de tamanho 128, para facilitar o armazenamento e otimizar o tempo de execução, já que quanto maior a janela de busca, mais lento. Além disso, é mais fácil escrever na memória utilizando bytes, o que reforçou ainda mais encontrar um balanço tempo e eficiência que atendesse a essa premissa.

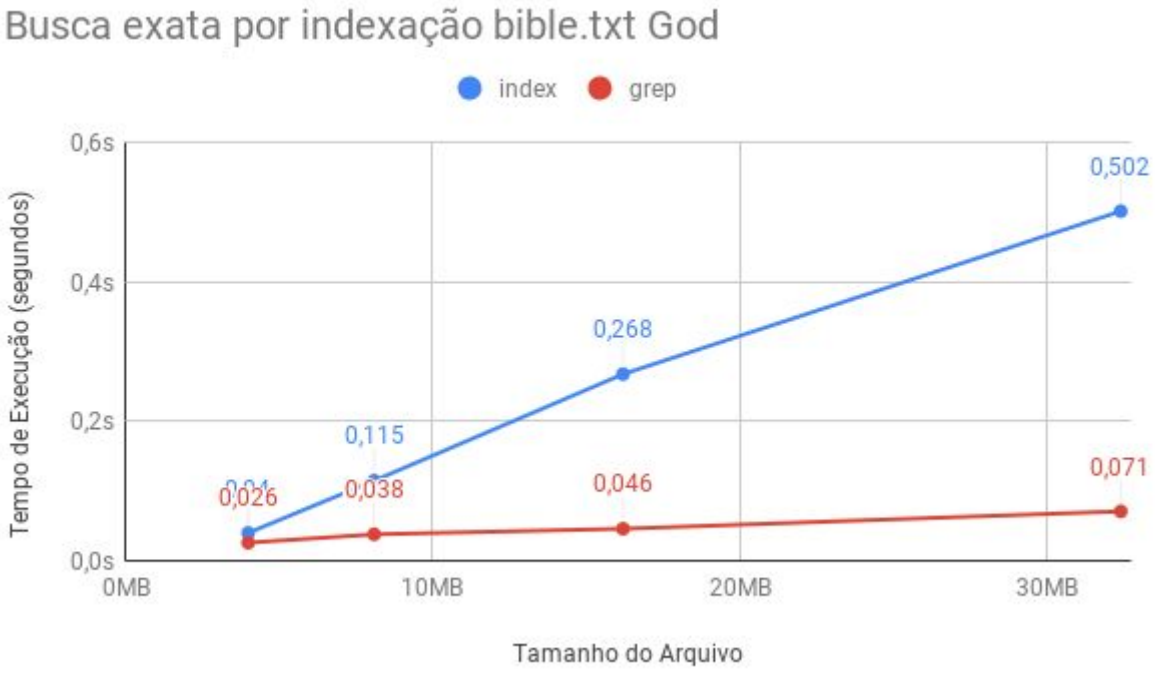
Para armazenar o par posição e comprimento foi necessário comprimí-los dentro de apenas um *size_t* (de tamanho 32 bits ou 64 bits a depender do sistema operacional), onde 9 bits foram usados para armazenar a posição (2 << 9 == 512) e 7 bits para armazenar o comprimento (2 << 7 == 128). Além disso, foi necessário armazenar o caractere, que toma um espaço de 8 bits, totalizando 24 bits (3 bytes) para cada tripla (posição, comprimento, caractere).

2 Testes

Para comparação com o algoritmo de busca exata, utilizou-se o *grep*, já que um dos objetivos da indexação é permitir uma busca rápida e para o algoritmo de compressão/descompressão utilizou-se o *gzip* que também faz uso do LZ77.



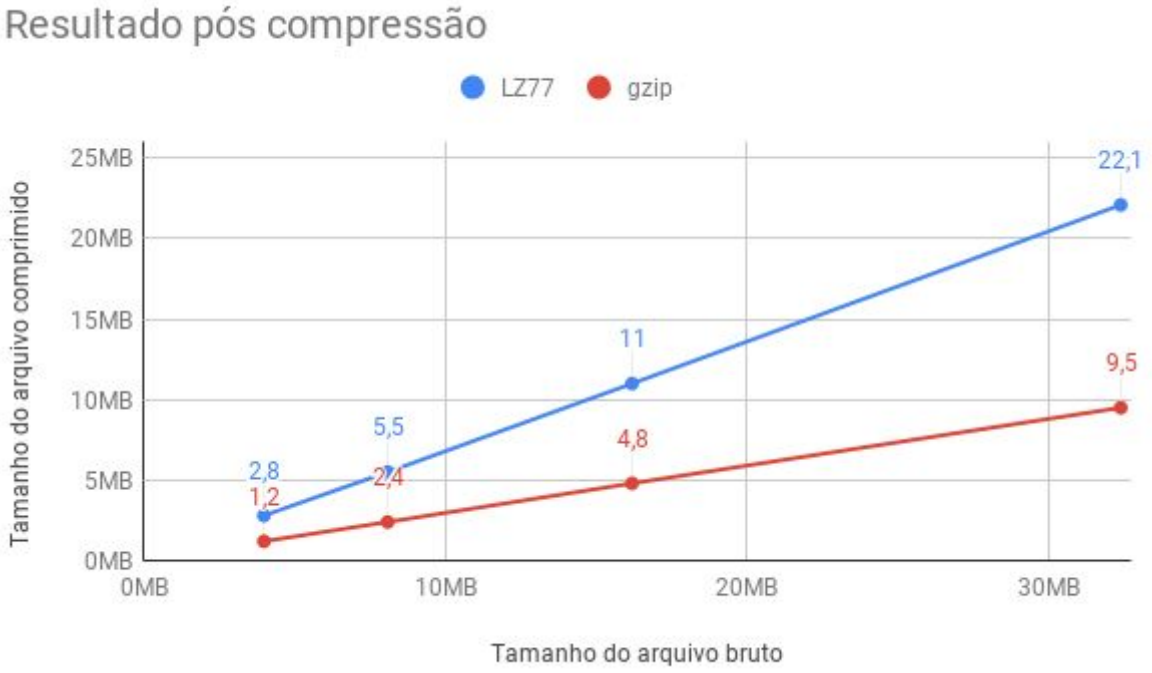
Aqui é possível notar que o tempo de execução é diretamente proporcional ao tamanho do arquivo, sempre duplicando juntamente com a quantidade de megabytes.



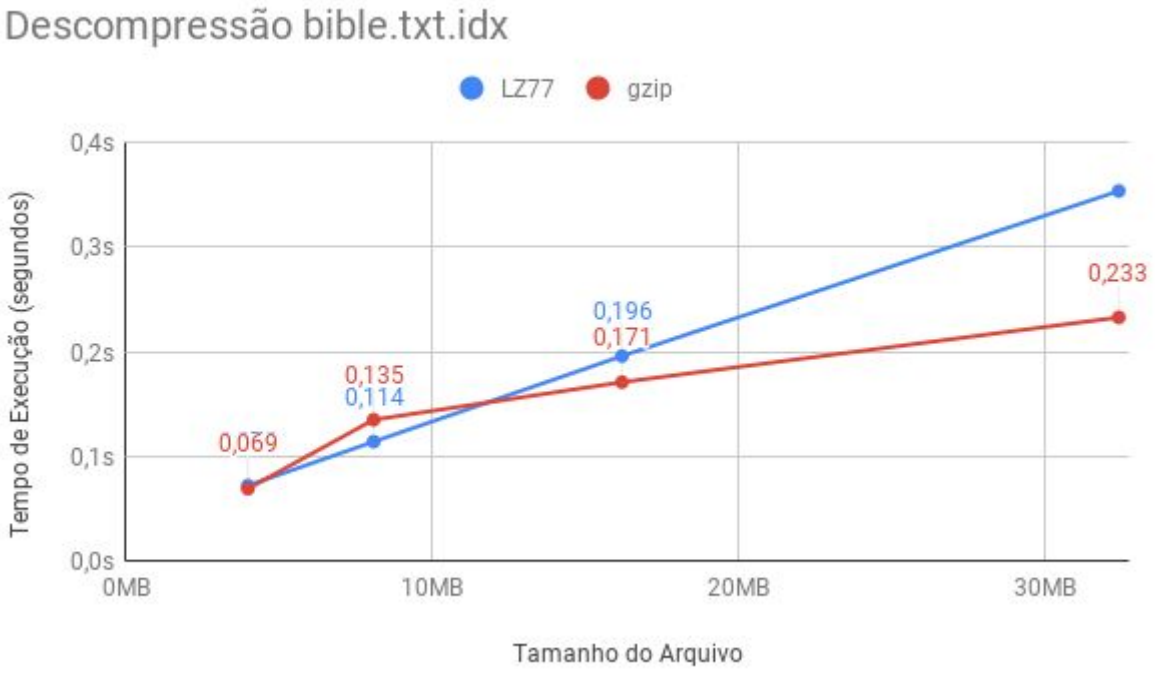
Um dos objetivos do algoritmo de indexação era permitir uma busca exata eficiente no texto, mas ainda com o pré-processamento necessário para criar o arquivo de indexação, o *grep* ainda apresenta um melhor desempenho, além de mostrar um ângulo de crescimento muito menor.

Pode-se observar que a implementação do algoritmo LZ77 do *gzip* é extremamente mais eficiente, fazendo com que a comparação do escopo de implementação do LZ77 deste

projeto não chegue nem perto de seu tempo de execução, provavelmente devido ao tempo necessário para criar as máquinas de estado da janela de busca.



Além de um tempo de execução melhor na compressão de arquivos, o gzip apresenta uma melhor taxa de compressão, resultando em arquivos mais compactos. Note que o ângulo das duas curvas é bem parecido, logo, esse comportamento deve se manter proporcional mesmo com arquivos de tamanho maior.



O tempo de descompressão do gzip e do algoritmo implementado neste projeto apresentam um tempo de execução similar, mas, provavelmente, com arquivos de tamanho maior o gzip seja mais eficiente, dado o ângulo de crescimento de sua reta. Talvez isso se explique pelo fato de que o gzip resulte em arquivos comprimidos de tamanho menor.

3 Conclusão

Com base nos resultados dos testes é possível observar que, mesmo com as otimizações usadas neste projeto, o espaço para mais otimizações ainda é bem expressivo. Quanto a compressão e descompressão, o *gzip* apresentou arquivos menores em um tempo de execução muito menor e, quanto à busca exata, mesmo com o tempo usado para criação do arquivo de indexação, teve um tempo de execução inferior ao *grep.*