Processamento de Cadeia de Caracteres - Projeto 2

Alunos:

Lucas Henrique Cavalcanti Santos - lhcs

Roberto Costa Fernandes - rcf6

1. Identificação

O projeto descrito no presente relatório foi implementado pelos alunos Lucas Henrique Cavalcanti Santos e Roberto Costa Fernandes. Lucas ficou responsável pela implementação do algoritmo de indexação de texto Suffix Array, enquanto Roberto ficou responsável pelos algoritmo de compressão LZ77.

Todo o código desenvolvido pela equipe pode se encontrado no Github: https://github.com/RC-Dynamics/string-processing.

2. Implementação

Algumas convenções serão utilizadas durante o relatório são elas:

- n é o tamanho do texto;
- LS é o tamanho do *search buffer;*
- LL é o tamanho do *lookahead buffer;*

2.1. Indexação e Busca de Texto - Suffix Array

Para realização de busca indexando o texto, foi utilizado o Suffix Array como algoritmo de indexação, este algoritmo indexa o texto e permite a busca de padrão através de duas buscas binárias. Portanto, a complexidade da indexação implementada ficou $O(n \log^2 n)$, pois a mesma só precisa ser realizada uma única vez para o mesmo texto. Porém com a utilização de vetor de sufixos, mais dois vetores de maior sufixo em comum, um para a esquerda e outro direita de cada item do vetor de sufixos, é possível buscar de forma otimizada em complexidade de $O(m + \log n)$. Porém para que tudo funcione, o index foi armazenado juntamente com o texto, ocupando mais espaço em disco para o index.

Outro fator importante da busca é a impressão da linha, pois como o Suffix Array apenas a posição do padrão no texto é conhecida. Portanto, para a impressão foi necessário encontrar o início e fim da linha linearmente.

2.2. Compressão de texto - LZ77

No algoritmo LZ77 é necessário definir um alfabeto a ser utilizado para a codificação, assim neste algoritmo o alfabeto é representado por uma string que contém os 128 primeiros caracteres da tabela ASCII.

O algoritmo LZ77 necessita que seja implementado uma função de compressão e outra de descompressão. Na função de compressão é necessário criar uma representação de uma máquina de estados finita para a busca do padrão no *lookahead buffer* na janela do *search buffer* + *lookahead buffer*. A estrutura escolhida para representar esse dicionário foi uma matrix (vetor de vetor), onde cada linha representa um estado e cada coluna uma transição, representada por uma letra do alfabeto de representação. Outras estruturas de C++ poderiam ser escolhidas, como por exemplo *unordered_map*, porém o uso da representação por matriz apresenta uma melhor performance em tempo. Também com a ideia de tornar mais eficiente toda para toda string manipulada, foi utilizado apenas os ponteiros e índices de início e fim.

Uma das limitações da implementação do LZ77 feita pela equipe é que tamanho máximo do *search buffer* e do *lookahead buffer* é 65535, que é o maior valor possível de se armazenar, sem sinal, em 16 bits. Esse valor poderia ser salvo em uma variável com mais bits, porém a equipe preferiu não alterar pois esse valor já foi suficiente para a realização dos testes.

Após a realização dos testes expostos na seção 3.1.3, foi possível escolher valores padrões para o *search buffer* e o *lookahead buffer*. Esses valores são:

- *lookahead buffer* = 4096
- *search buffer* = 256

3. Testes e Resultados

3.1. Realização dos Testes

Para realização dos testes foram escolhidos cinco arquivos com codificação de proteinas retirados do *Pizza&Chili* (http://pizzachili.dcc.uchile.cl/texts/protein/). Os arquivos utilizados no testes foram versões menores dos disponibilizados, por causa do tempo de execução. Os tamanhos dos arquivos utilizados foram de 5KB, 100KB, 1MB e 5MB

Todos os testes foram realizados na mesma máquina e, dentro do possível, nas mesma condições. A configuração da máquina de testes é:

- Ubuntu 18.04
- Processador i7-7600
- 8 GB RAM
- 240GB SSD

3.1.1. Tempo de Indexação

Os primeiros testes realizados foram a avaliação do tempo para criar e armazenar o index e texto. Para cada tamanho de arquivo especificado anteriormente foram realizados 10 testes, abaixo encontram-se o tempo médio para indexação de cada arquivo.

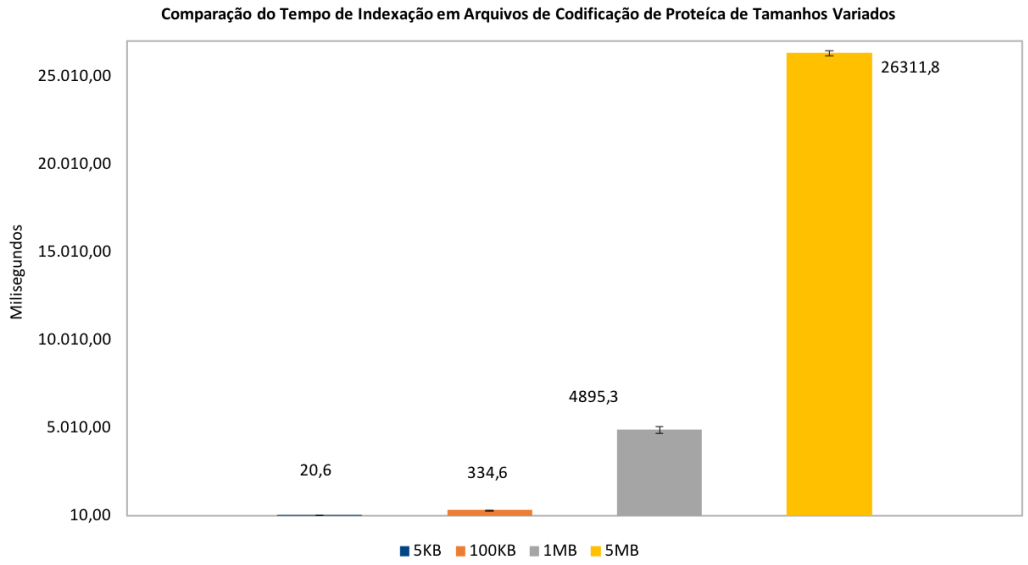

Figura 1 - Tempo médio de 10 indexação dos arquivos de 5KB, 100KB, 1MB e 5MB.

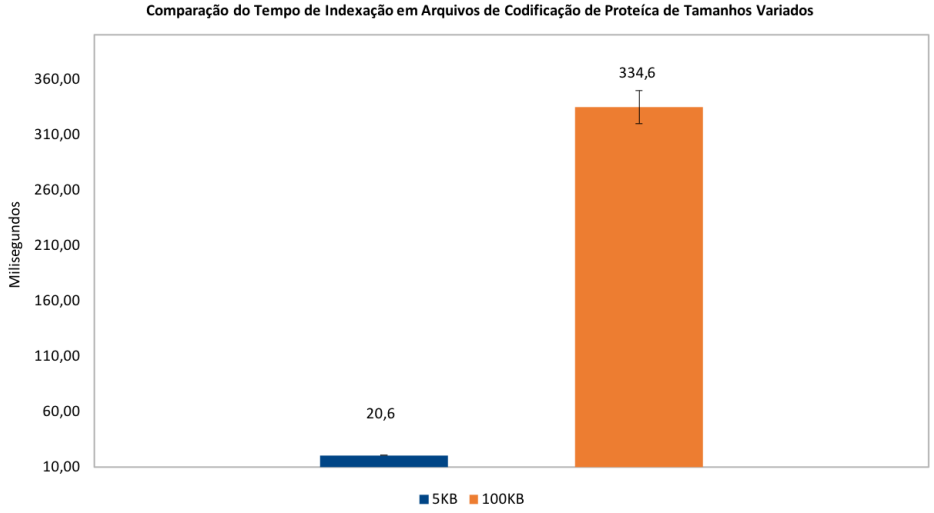

Figura 2 - Tempo médio de 10 indexação dos arquivos de 5KB e 100KB.

Através da Figura 1 e 2, podemos observar o crescimento exponencial no tempo da indexação, de acordo com a complexidade do algoritmo. Os dados também demonstram uma pequena variação entre as avaliações, obtendo um desvio padrão baixo.

3.1.2. Tempo de Busca

Com o arquivo indexado foram realizadas leituras e buscas, de diferentes padrões em todos tamanhos de arquivos. Um dos padrões utilizados foi escolhido aleatoriamente do texto e conta com 10 caracteres, o segundo padrão não existe no texto e possui também tamanho de 10 caracteres. Por fim, foi avaliado a busca por um padrão de apenas 1 caractere para avaliar a busca com diversas ocorrências.

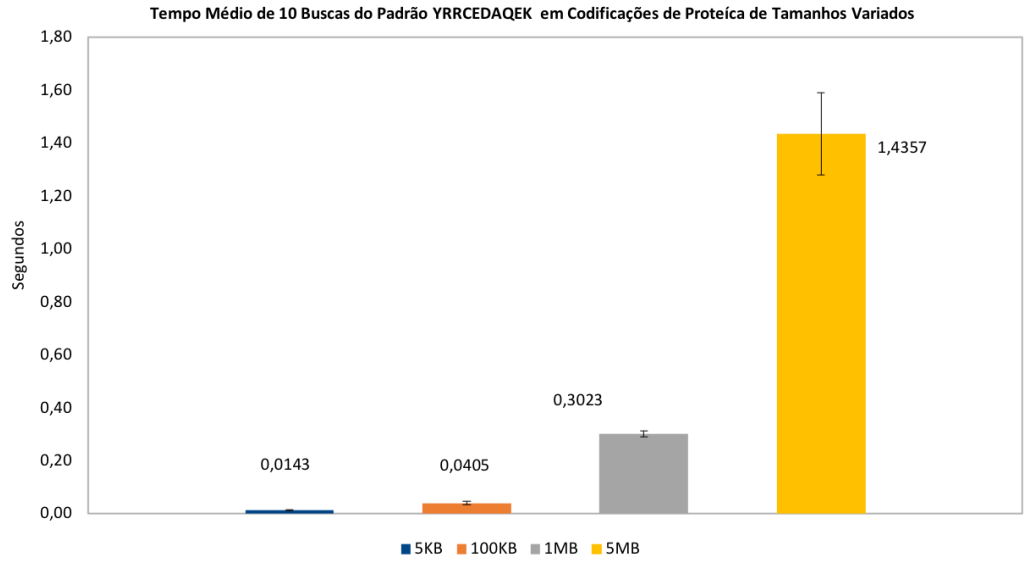

Figura 3 - Tempo médio de 10 buscas do padrão YRRCEDAQEK nos arquivos de 5KB, 100KB, 1MB e 5MB já indexados pelo Suffix Array.

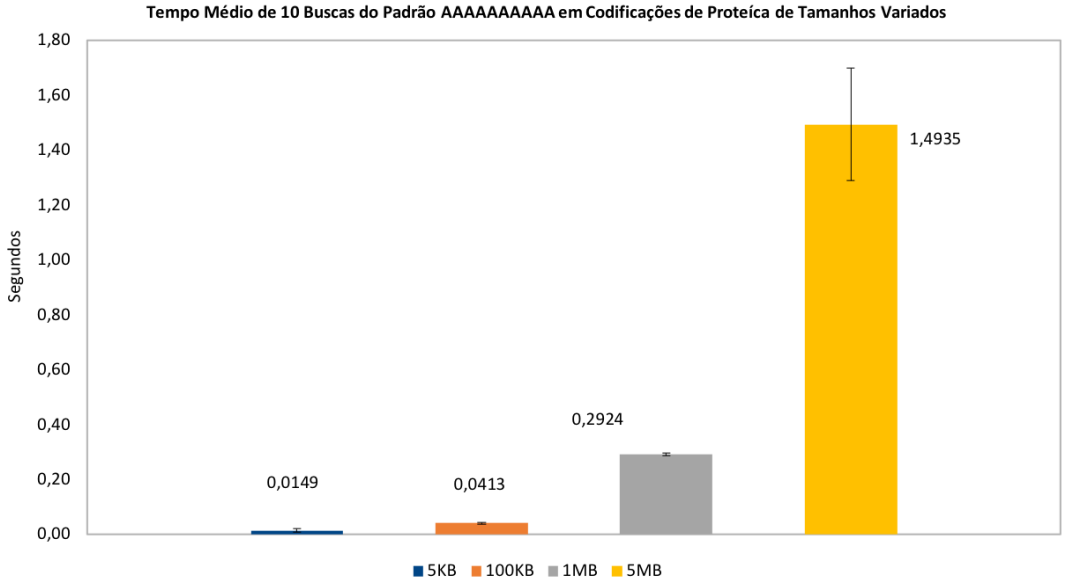

Figura 4 - Tempo médio de 10 buscas do padrão AAAAAAAAAA nos arquivos de 5KB, 100KB, 1MB e 5MB já indexados pelo Suffix Array.

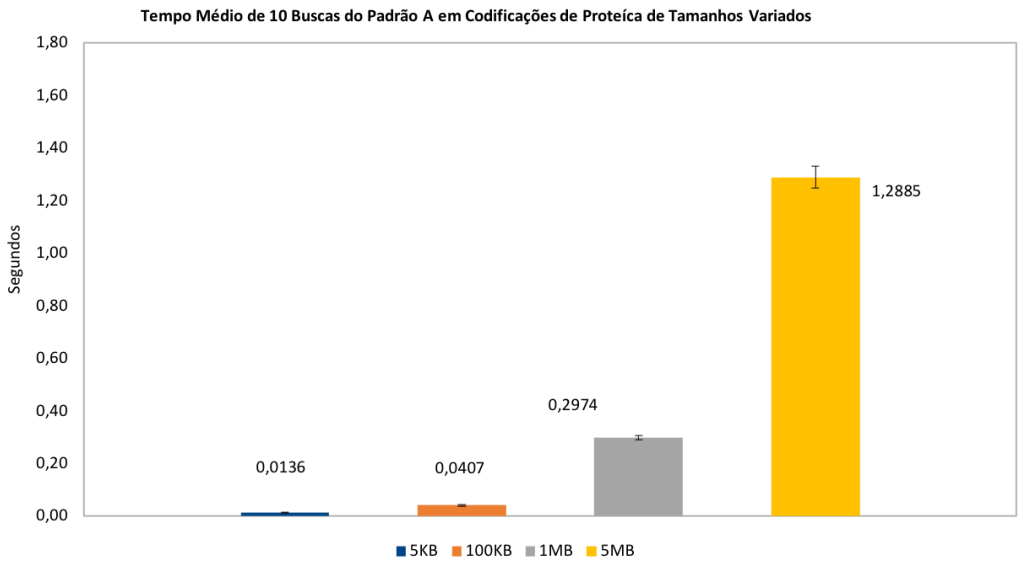

Figura 5 - Tempo médio de 10 buscas do padrão A nos arquivos de 5KB, 100KB, 1MB e 5MB já indexados pelo Suffix Array.

Após observar as Figuras 3, 4 e 5 é possível ver uma semelhança entre elas no tempo para buscar diferentes padrões, com números de ocorrências diferentes. Também é importante destacar que o pior caso para a busca binária, Figura 4, não possui diferença significativa para o caso médio Figura 5.

3.1.3. Tamanho dos Buffers do LZ77

O primeiro teste realizado pela equipe foi para identificar o tamanho ideal para ser utilizado nos buffers do LZ77, de acordo com o tempo de execução e com o tamanho do arquivo comprimido. Esse teste foi realizado apenas apenas com a compressão sem a indexação do teto. Foram realizadas 10 execuções para cada par de tamanho de buffers e cada tamanho de arquivo, e com isso foi tirado a média do tempo de execução.

Os valores escolhidos a serem testados para LS e LL foram escolhidos estrategicamente para caber em um byte, e os valores testados foram:

- LS = 65535 e LL = 255;
- LS = 512 e LL = 128;
- LS = 4096 e LL = 256;

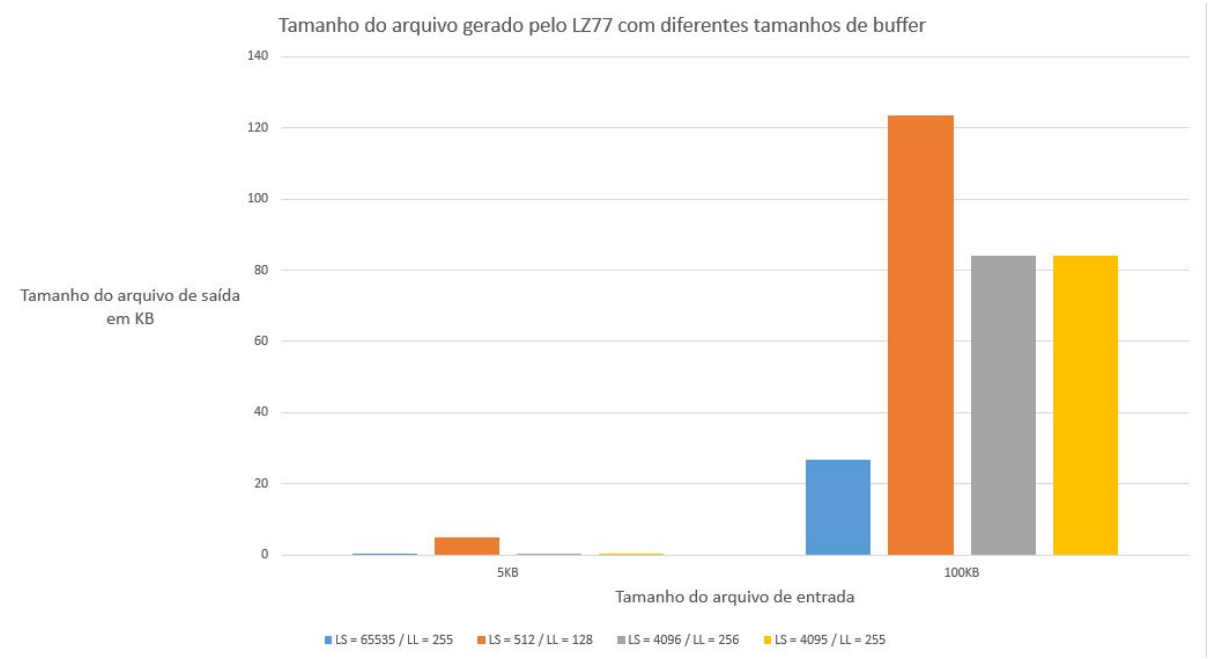

Figura 6 - Tamanho do arquivo comprimido para arquivos de 5KB e 100KB

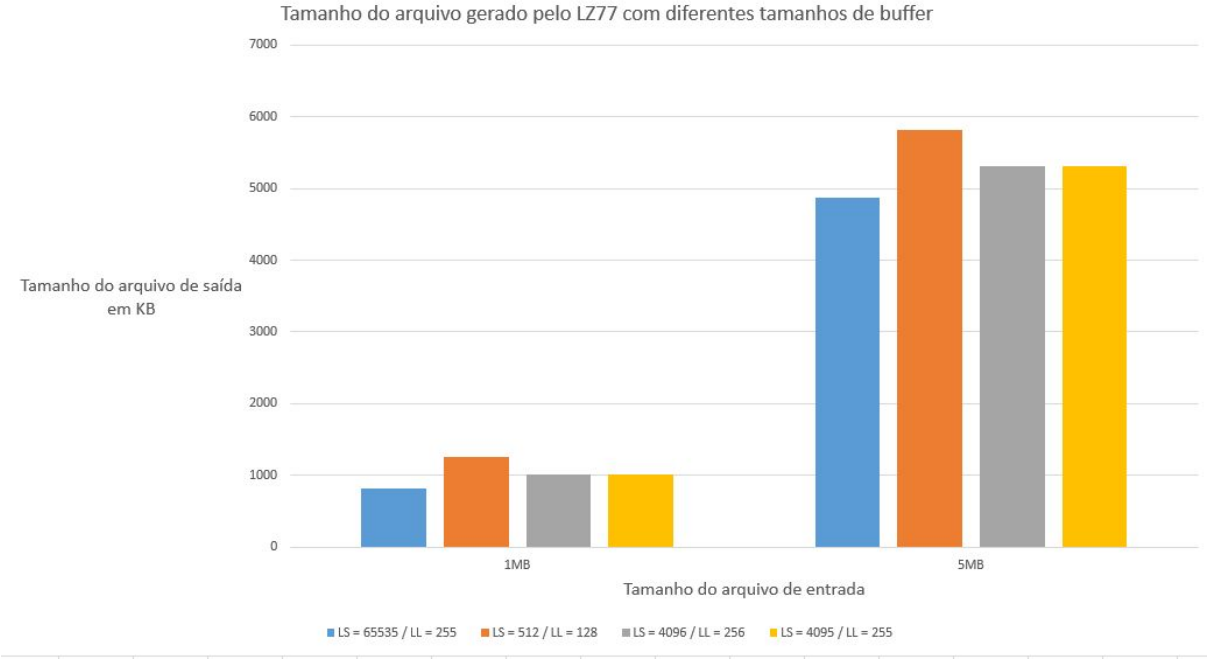

Figura 7 - Tamanho do arquivo comprimido para arquivos de 1MB e 5MB

As figuras 6 e 7 mostram a comparação do tamanho de arquivo gerado após a compressão de vários arquivos com diferentes tamanhos. As figuras 8 e 9 mostram a comparação de tempo de execução para o mesmo testes realizados para gerar a figura 6. Com base nos resultados obtidos, a equipe escolheu os valores de LS = 4096 e LL = 256, para não gerar um arquivo muito grande e a geração desse arquivo não durar muito tempo.

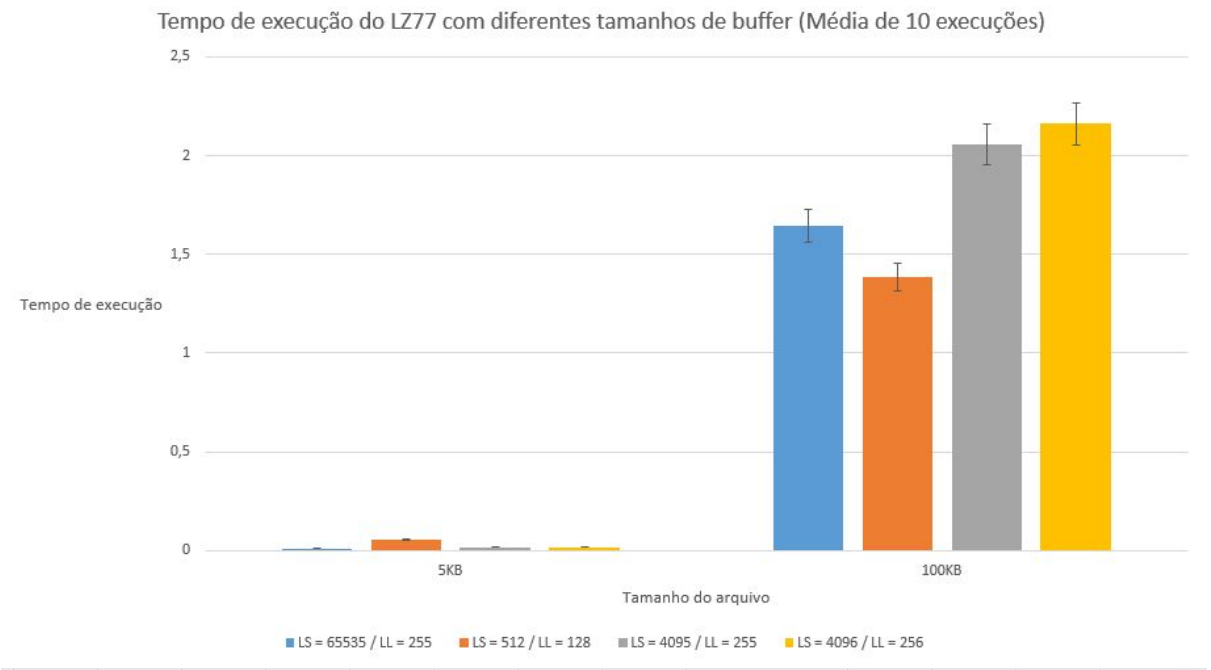

Figura 8 - Tempo de compressão para arquivos de 5KB e 100KB

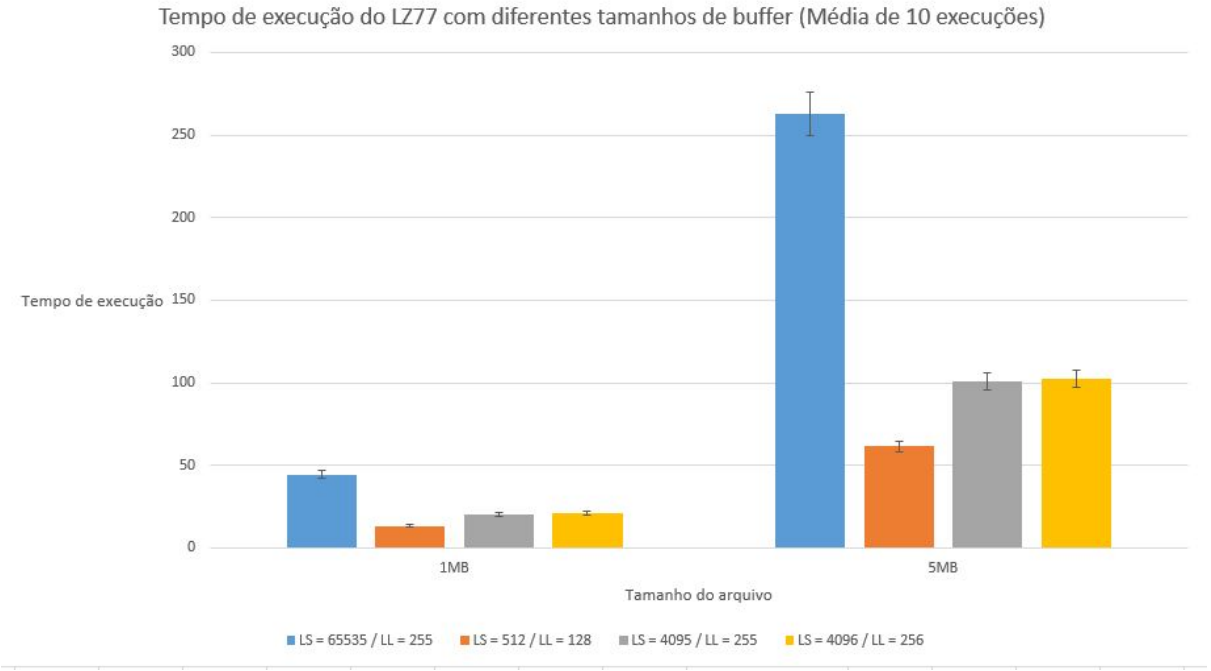

Figura 9 - Tempo de compressão para arquivos de 1MB e 5MB

3.1.4. Tempo de Indexação e Compressão

Nestes testes depois de realizar a indexação, o os dados foram comprimidos antes de salvos. Neste, realizamos apenas nos arquivos de 5KB, 100KB e 1MB, pois arquivos maiores estavam levando muito tempo para serem comprimidos.

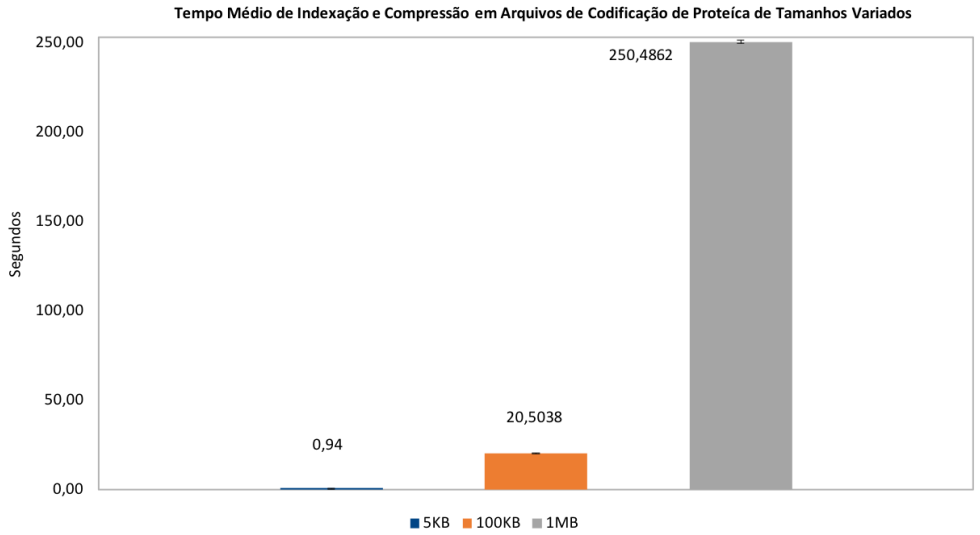

Figura 10 - Tempo médio de 5 indexações e compressão para arquivos de 5KB, 100KB e 5MB.

Comparando a Figura 1 com a Figura 10 podemos observar um aumento de aproximadamente 60 vezes no tempo execução, com a adição da compressão de arquivo, porém observando a tabela abaixo é observada uma grande diminuição no consumo espaço em disco.

Tabela 1 - Comparando os tamanhos da codificação original, com o tamanho do arquivo apenas indexado, e com o arquivo indexado e comprimido.

Codificação de Proteína	Sem Compressão	Com Compressão
7.364 bytes	72.561 bytes	8.083 bytes
129.482 bytes	1.443.553 bytes	2.528 bytes
1.283.058 bytes	16.016.928 bytes	30.203 bytes

3.1.5. Tempo de Descompressão e Busca

Agora, toda busca necessita antes descomprimir o arquivo. Portanto foram gerados os mesmos testes de busca que os anteriores, com os mesmos padrões, porém excluindo o arquivo de 5MB, como explicado na seção anterior.

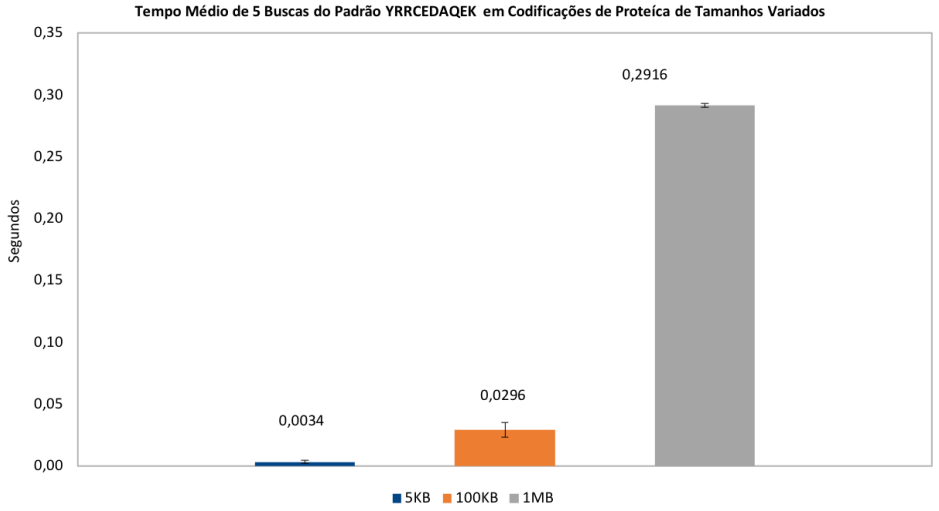

Figura 11 - Tempo médio de 5 buscas do padrão YRRCEDAQEK nos arquivos de 5KB, 100KB e 1MB indexados e comprimidos.

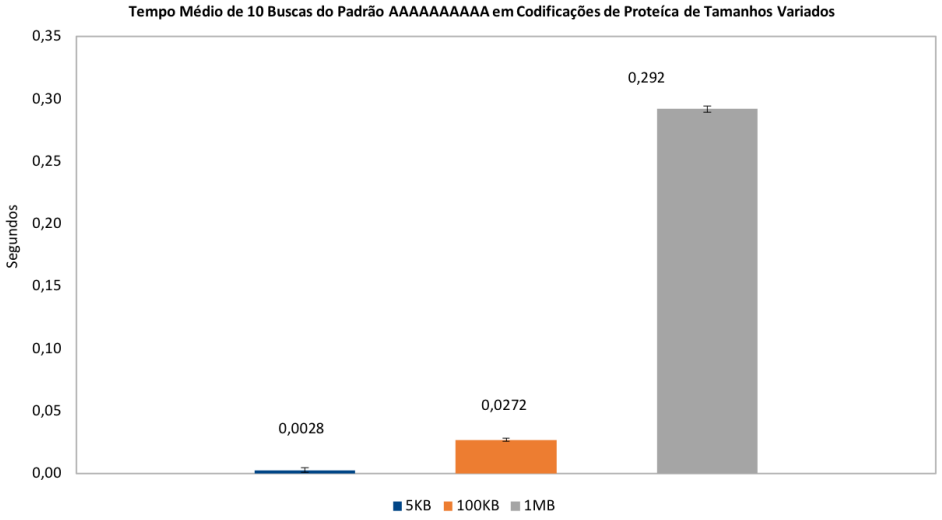

Figura 12 - Tempo médio de 5 buscas do padrão AAAAAAAAAA nos arquivos de 5KB, 100KB e 1MB indexados e comprimidos.

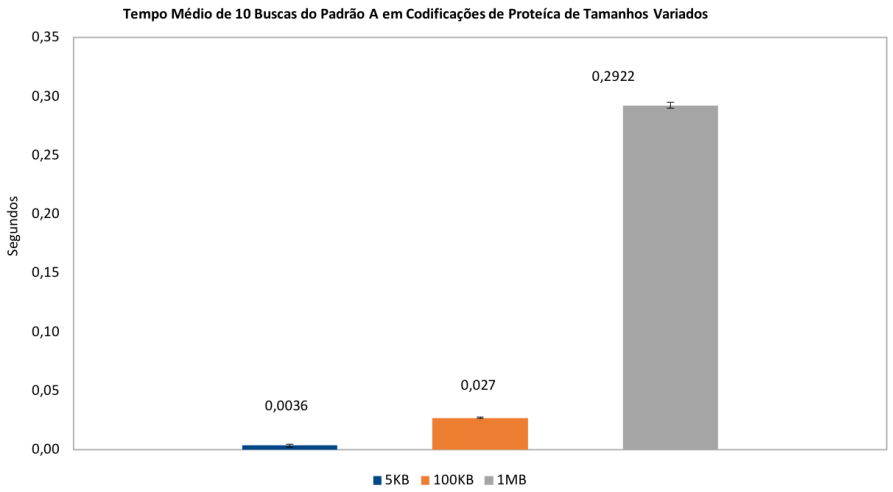

Figura 13 - Tempo médio de 5 buscas do padrão A nos arquivos de 5KB, 100KB e 1MB indexados e comprimidos.

Comparando as Figuras 3, 4, e 5 com as Figuras 11, 12 e 13, é possível observar que a descompressão não afeta o tempo de busca, mas traz a vantagem de reduzir o espaço utilizado em disco.

4. Conclusões e Trabalhos Futuros

Com os resultados obtidos foi possível obter algumas conclusões:

- O Suffix Array e o texto junto, mesmo quando comprimidos ocupam bastante espaço em memória, mais que o texto não comprimido.
- Apesar da indexação ser custosa, a busca fica otimizada a ponto que padrões de tamanhos diferentes e ocorrências diferentes possuem tempo de busca similar.
- A compressão de arquivo varia sua eficiência para LS e LL diferentes.
- A compressão se demonstrou lenta, portanto, indexar o arquivo é lento.
- A descompressão não afetou o tempo de busca, portanto o LZ77 traz a vantagem de reduzir o espaço utilizado em disco.

Como trabalhos futuros e melhorias para o projeto apresentado fica como ideias:

- Utilizar outras bases de dados e outros tamanhos de arquivo para a escolha dos melhores tamanhos das janelas no LZ77.
- Guardar o Suffix Array e a frequência dos sufixos para reconstrução do texto, e assim reduzir o espaço ocupado em memória.
- Procurar outras estratégias mais ágeis de compressão.