ipmt

1. Identificação

Gabriel de Albuquerque Souza Meireles - gasm@cin.ufpe.br
Daniel de Azevedo Pacheco - dap5@cin.ufpe.br

2. Implementação

a. Suffix array

O algoritmo é executado linha por linha. Ao realizar a operação index, um arquivo de extensão .idx é gerado contendo toda a informação necessária para reconstruir cada linha do texto (o array de sufixo e um array de frequências). Ao executar o comando search, cada linha do texto é reconstruída e então busca-se cada padrão fornecido nessa linha.

b. Huffman

O algoritmo é executado a cada uma certa quantidade fixa de caracteres no texto. Assim, um mesmo arquivo pode ser comprimido em vários blocos. Ao usar o comando zip, é criado um arquivo de mesmo nome acrescido da extensão ".myz". O arquivo comprimido possui um ou mais blocos que seguem o formato **header + tamanho em bytes do texto + texto codificado convertido para char + número de *trailing zeros***. Já o header segue o formato de tamanho do header seguido de **letra + tamanho do código + código da letra convertido para char.** Ao realizar o comando unzip, é recriado um arquivo sem a extensão ".myz". É feita a decodificação do header, recuperando o código de cada letra. Assim é possível reconstruir a árvore de Huffman que é então usada para decodificar o texto. Considere o exemplo abaixo. Nele usamos a notação {} para representar a conversão de uma cadeia binária para o seu respectivo char:
Texto: acccccccbbbbaab
Códigos: (a: 00), (b: 11), (c: 0)
Texto codificado: 10000000011111111101011
Header: 3a2{"**00**000000"}b2{"**11**000000"}c1{"**0**0000000"}
Texto codificado em char: 3{"**10000000**"}c{"**11111111**"}{"**10101**00"}2

Foi pensado, mas não implementado, uma segunda abordagem para a formatação do header que melhoraria o desempenho para textos pequenos. Nessa segunda abordagem, seria escrito primeiro as letras presentes juntamente com os tamanhos dos seus códigos e só depois seria escrito todos os códigos em ordem. No mesmo exemplo usado acima:
Header ideal: 3**a2b2c1**{"**00110**0000"}

3. Testes e resultados

Para a realização dos testes para se fazer a análise de desempenho dos algoritmos implementados na ferramenta foram desenvolvido scripts em python que realiza chamadas da ferramenta junto com a ferramenta “time” da biblioteca do gnu que foi utilizada com a opção “-f %e” para que o output seja apenas o tempo real de execução da ferramenta, e também foi utilizada as opções “-o” e “-a” para salvar esses tempos de execução em um arquivo. Para cada teste realizado também foi passado para a ferramenta a opção “-c” para que as saídas sejam apenas o número de ocorrências do padrão, de forma que não seja gasto tempo exibindo cada ocorrência na tela.

Os testes foram realizados em um computador com sistema operacional Linux mint 20.3 com kernel 5.4.0-107-generic, processador AMD Ryzen 5 3500U, 13.7 Gb de memória Ram DDR 4 com velocidade de 2400 MT/s. todos os testes foram realizados com o notebook na tomada e sem acesso a rede.

Para fazer uma visualização gráfica dos resultados foi utilizada a ferramenta gnuplot e a ferramenta de gráficos do google sheets, e para cada gráfico gerado foi utilizado a média do tempo de execução resultante de 10 execuções do teste realizados em sequência.

Ao longo dos testes foram utilizados 3 tipos diferentes de arquivos para se realizar as buscas e compactação, cada tipo contendo um arquivo de 100 megabytes, 500 megabytes e 1 gigabyte. A primeira categoria de arquivos são referentes a sequencias de proteínas (disponibilizadas em http://pizzachili.dcc.uchile.cl/texts.html) codificadas por 20 diferentes letras maiúsculas. O segundo tipo de arquivo são arquivcos de texto em que seu conteúdo é o texto da “The King James version of the bible” (disponibilizado em https://corpus.canterbury.ac.nz/descriptions/large/bible.html) repetido até o arquivo ter o tamanho desejado. Por fim, também foram usados arquivos de texto de nossa autoria que consistem apenas do mesmo caractere “A” repetido a fim do arquivo ter o volume certo.

Ferramenta de indexação

Primeiramente, antes de realizar os testes do modo de busca foi realizada a indexação dos arquivos onde as buscas serão realizadas.

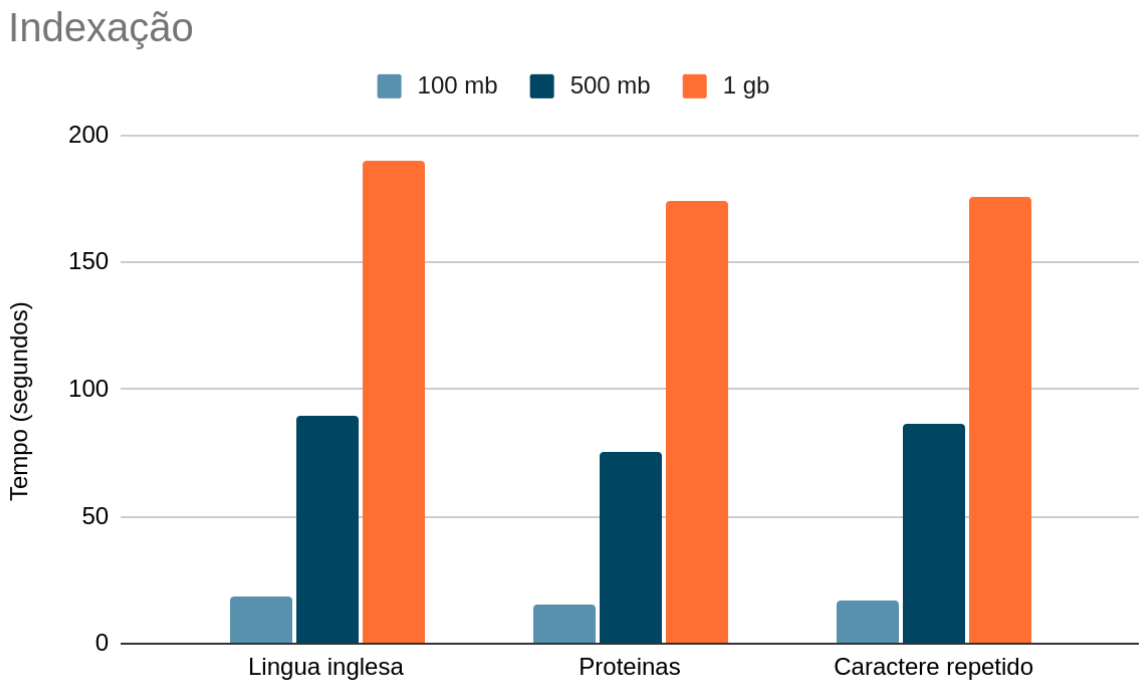

Gráfico 1 - Tempos de indexação

Ao analisar os tempos de execução das indexações no gráfico acima podemos observar que o tempo de execução do algoritmo está majoritariamente ligada ao tamanho do arquivo a ser processado, o que está dentro do esperado já que a complexidade do algoritmo é de **O(N log N)** onde N é o tamanho do texto a ser indexado.

Ferramenta de busca

Nos primeiros testes realizados para analisar o desempenho da ferramenta de busca indexada foram realizadas buscas para cada tamanho dos diferentes tipos de arquivos descritos acima, aumentando o padrão a ser procurado indo de uma string de tamanho 10 até uma string de tamanho 100. As strings realizas para a busca fora strings retiradas previamente do texto de forma aleatória.

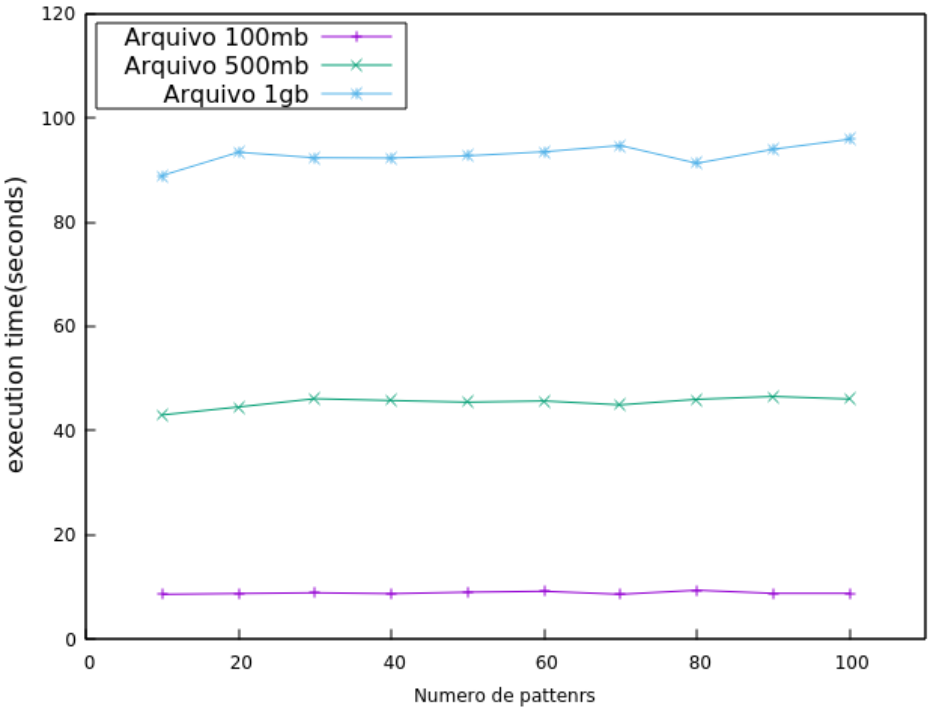

Gráfico 2 - Busca em arquivos de texto em língua inglesa

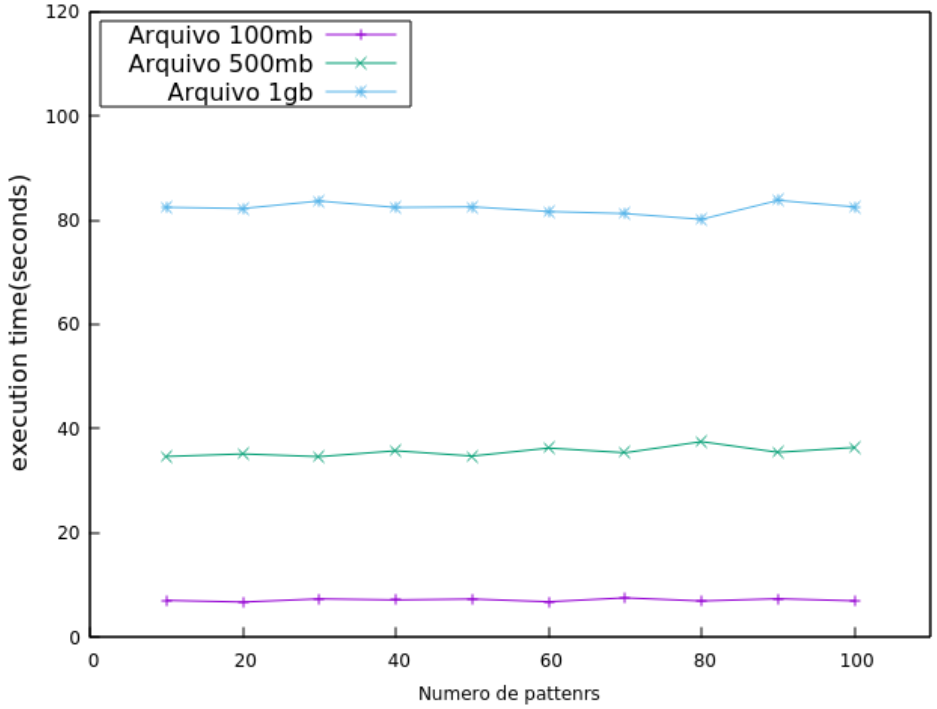

Gráfico 3 - Busca em arquivos de proteínas

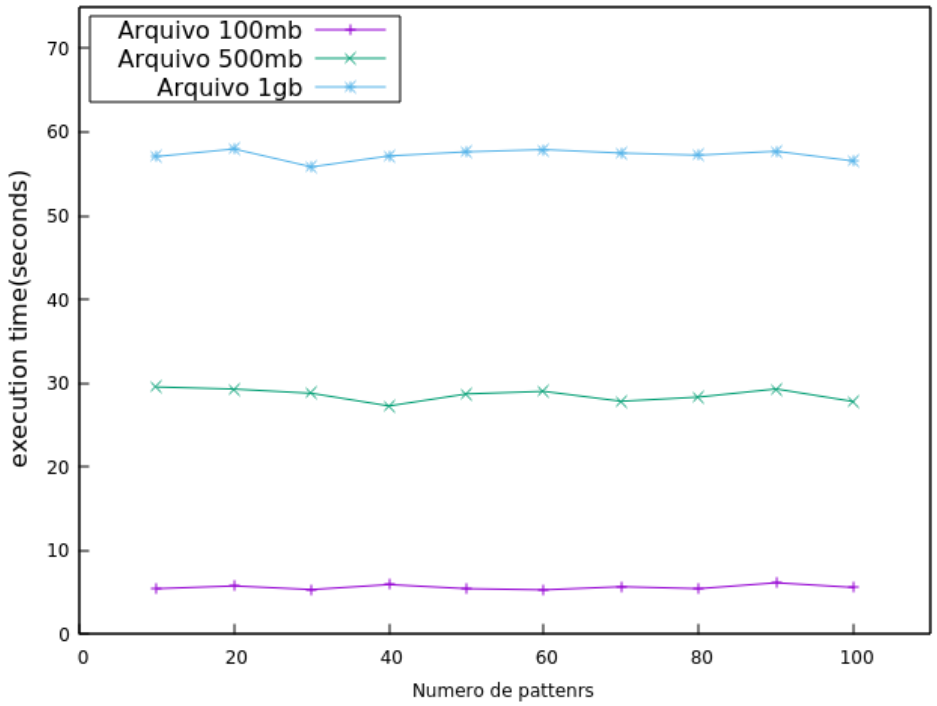

Gráfico 4 - Busca em arquivos de caractere repetido

Como é possível observar nos gráficos acima podemos observar que o tempo de execução vem recebendo um impacto desprezível com o aumento do padrão. Podemos atribuir esse comportamento ao fato de que nos casos observados o tamanho do texto é maior que o padrão em muitas magnitudes.

Pattern File

Outra funcionalidade do nosso IPMT que desejamos testar é o uso do pattern file para buscar mais de um padrão por chamada do programa. para realizar esses testes foi feito um script em python que selecionou 100 substrings de tamanho 100 distintas e que só apareçam uma única vez no arquivo em que a busca irá ser realizada e separe elas em arquivos contendo de 10 a 100 dessas substrings. com esses arquivos realizamos as buscas no arquivo de cadeias de proteínas.

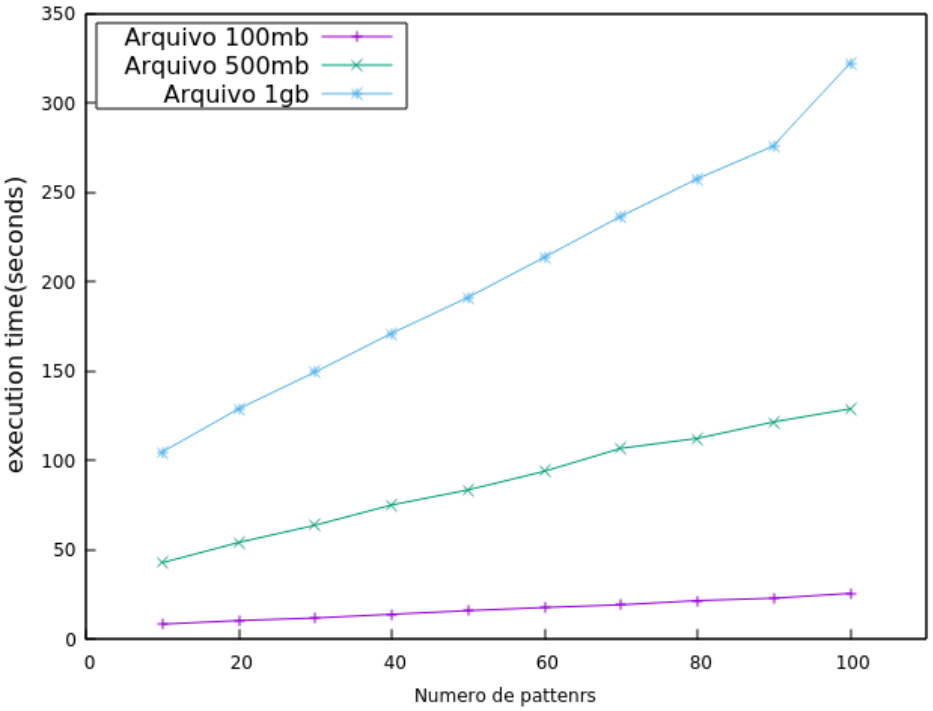

Gráfico 5 - Busca com patternfile

Com os dados coletados podemos observar que a execução da busca com o pattern file aumenta o tempo de execução com o aumento do número de padrões a serem procurados.

Ferramenta de compactação e descompactação

Para avaliarmos o desempenho do modo de compactação da ferramenta rodamos o seu modo de compactação para os arquivos de texto de língua inglesa, proteínas e de um caráter repetido nos tamanhos de 100 megabytes, 500 megabytes e 1 gigabyte com objetivo de inspecionar o tempo de execução do algoritmo e quão bem ele consegue diminuir os arquivos compactados. para fim de comparação também realizamos compactação dos mesmos arquivos utilizando o comando zip no terminal linux. por fim para garantir a integridade dos arquivos após a descompactação foi utilizando o comando “diff” entre o arquivo descompactado e uma cópia do arquivo orginal.

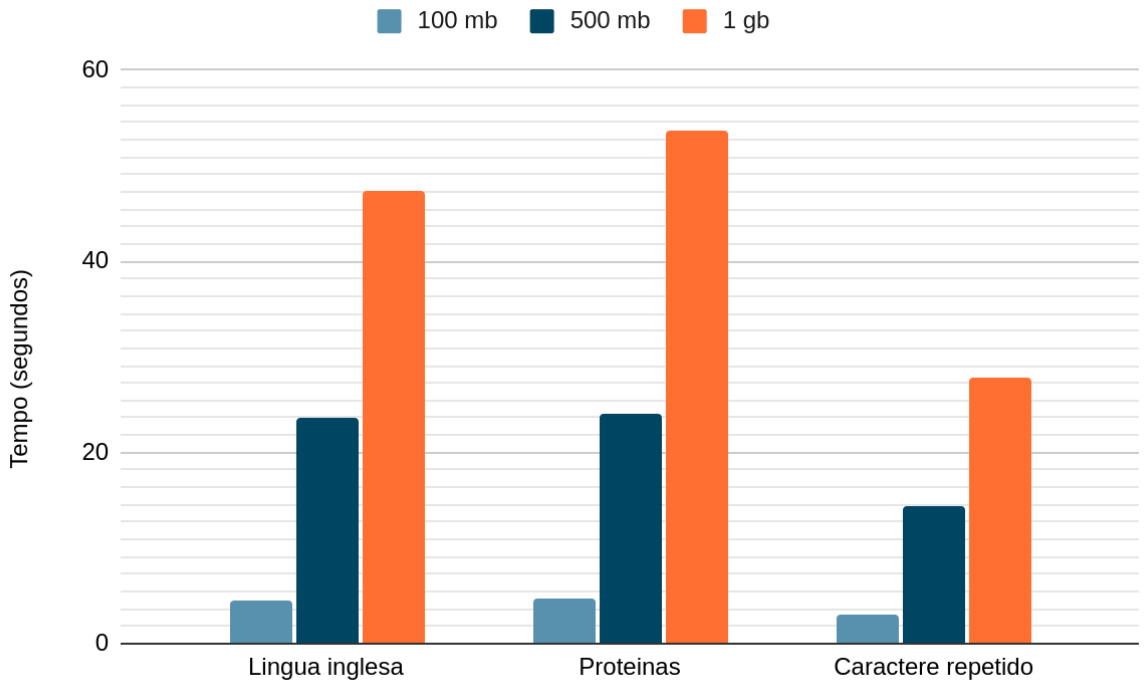

Gráfico 6 - Tempos de compactação IPMT

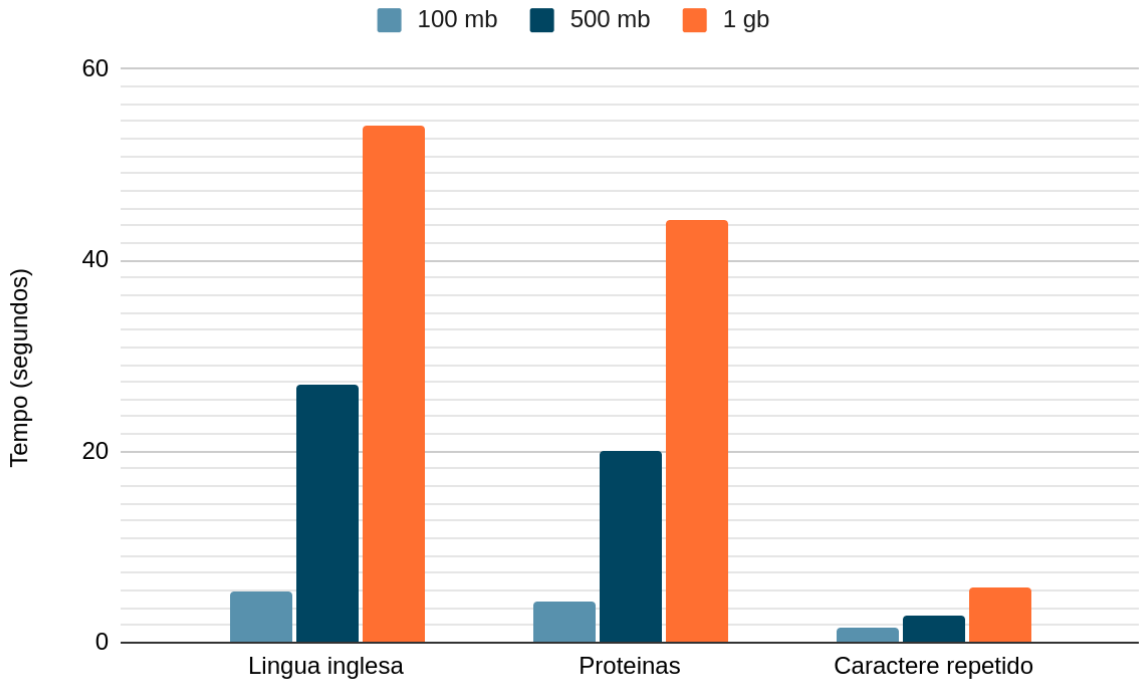

Gráfico 7 - Tempos de compactação ZIP

Nos gráficos acima podemos observar os desempenhos dos processos de compactação. Como é possível perceber ao analisarmos o gráfico 6 e 7 temos que o

algoritmo de huffman utilizado no ipmt demonstrou uma eficiência de tempo comparável com o da ferramenta zip para os arquivos de proteínas e de língua inglesa tendo uma média de tempo de execução muito parecida. já ao analisarmos o tempo de compactação dos arquivos compostos por um único caractere repetido podemos ver o zip consegue ser muito mais eficiente realizando o processo em cerca de ⅙ do tempo.

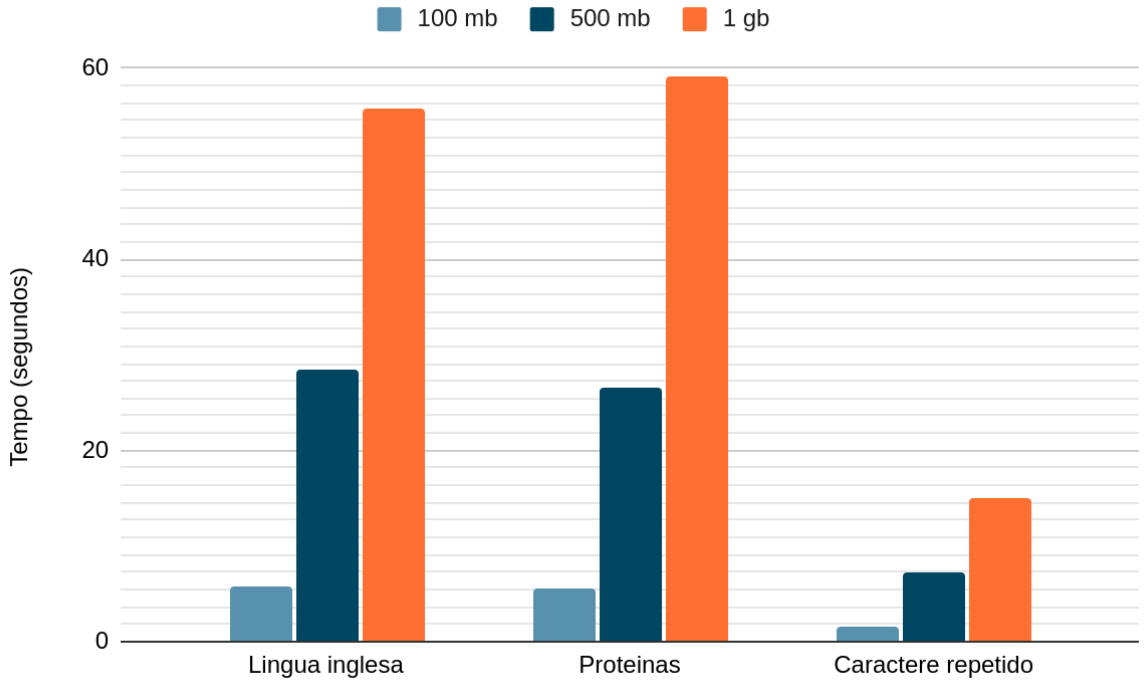

Gráfico 8 - Tempos de descompactação IPMT

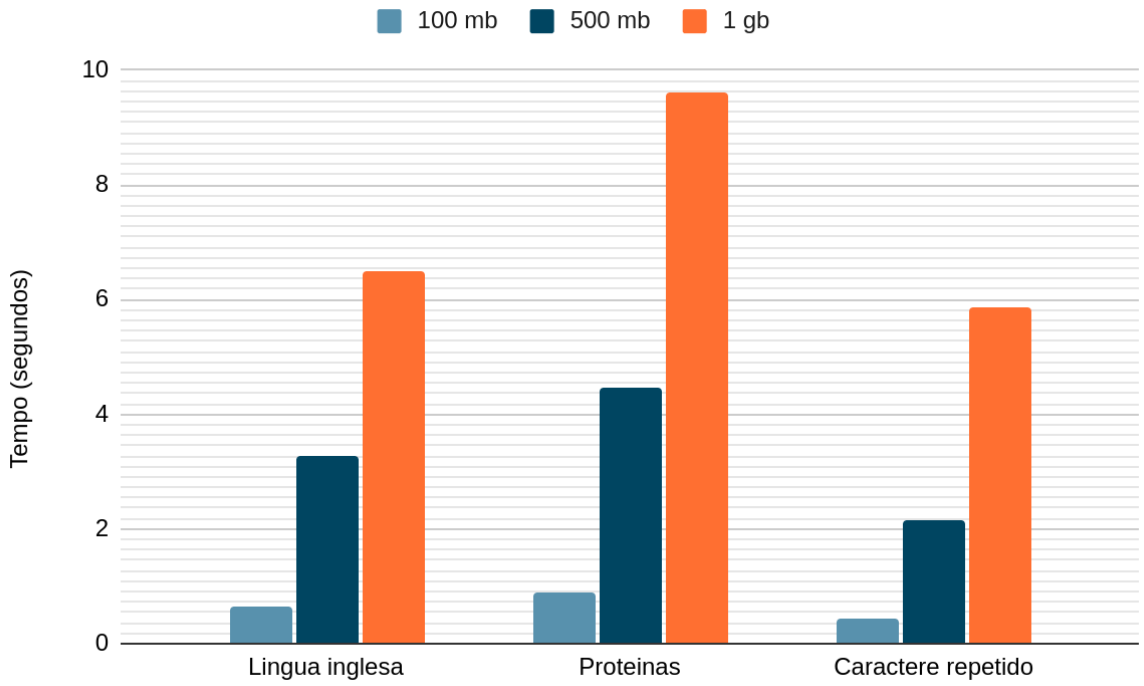

Gráfico 9 - Tempos de descompactação ZIP

Já em relação aos gráficos 8 e 9 referentes aos tempos de descompressão dos arquivos podemos primeiramente perceber que para o ipmt os tempos de descompactação estão muito próximos dos tempos de compactação, que é o comportamento esperado do algoritmo de huffman que para refazer o arquivo original tem que realizar o processo inverso da compactação. Ao comparar os tempos do ipmt com a ferramenta zip é possível perceber que a descompressão do algoritmo utilizado pelo "zip" se mostra muito mais eficiente, tomando cerca de ⅕ do tempo do ipmt para os arquivos de proteínas e língua inglesa. esse comportamento só não é observado para a descompressão dos arquivos de um caractere repetido, onde o ipmt teve resultados muito parecidos com o zip.

	100 MB	500 MB	1 GB
Língua inglesa	55%	51%	55%
Caractere repetido	12.5%	12.4%	12.5%
Proteínas	55%	51%	59%

Tabela 1 - Volume dos arquivos compactados pelo IPMT (em relação ao original)

	100 MB	500 MB	1 GB
Língua inglesa	41%	30%	28.9%
Caractere repetido	0.27%	0.8%	0.3%
Proteínas	51.7%	50%	52.9%

Tabela 2 - Volume dos arquivos compactados pelo ZIP (em relação ao original)

Para podermos entender melhor como está o desempenho efetivo nos tamanhos dos arquivos compactados podemos observar as tabelas 1 e 2.

Primeiramente podemos observar que a compactação do algoritmo huffman do ipmt obteve um desempenho quase idêntico tanto para os arquivos em língua inglesa quanto para os arquivos de proteínas, reduzindo eles em média para um tamanho de 54% do arquivo original. também podemos observar como o huffman comprimir para um percentual menor os arquivos de caracteres repetidos, já que por só ter um caractere no alfabeto só é necessário utilizar um bit para cada caractere de forma que a compactação é a mais eficiente possível para esse algoritmo.

Ao compararmos os resultados do ipmt na tabela 1 com os do zip na tabela 2 podemos ver que no arquivo de proteínas onde os texto é muito aleatório e com menos repetições o desempenho do ipmt chegou próximo do zip, que gerou arquivos em em média com volume de 51,4% do tamanho original. Já para os arquivos de língua inglesa e de caracteres repetidos o zip teve resultados consideravelmente melhores, com uma média de 33% do volume original para arquivos da língua inglesa e 0.45% para os arquivos de caracteres repetidos.