UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Departamento de Ciência da Computação
DCC605 -- Sistemas Operacionais
Trabalho Prático 2
Guilherme de Abreu Lima Buitrago Miranda - 2018054788
Victor Hugo Silva Moura - 2018054958

1 - Introdução

No segundo trabalho da disciplina de Sistemas Operacionais, é necessário que se implemente um simulador de memória virtual. Nele, após ser definido o tamanho da memória e o algoritmo de substituição de páginas a ser utilizado, deve-se simular o acesso à memória e atualizar bits de controle de cada quadro, detectar falhas de páginas e simular o processo de carga e substituição de páginas.

Assim, uma solução na linguagem C foi desenvolvida. Na seção imediatamente abaixo, encontram-se os detalhes da implementação, seguidos pela análise do desempenho, desafios enfrentados, conclusões e referências bibliográficas.

2 - Implementação

Para a implementação do trabalho, o primeiro ponto a ser trabalhado foi o tratamento dos argumentos de entrada para o programa. Sendo assim, inicialmente o número de parâmetros é conferido, para garantir que o usuário está entrando com o número correto de parâmetros para o programa, e duas funções adicionais foram feitas: uma para conferir se os valores do tamanho de página e tamanho da memória são inteiros positivos, e outra para conferir se o algoritmo de reposição está entre os algoritmos suportados pelo programa.

Feito isso, o próximo ponto trabalhado foi alocar as estruturas para armazenar a tabela de páginas e a memória física. Para a tabela de páginas, foi definido um struct que armazena os seguintes dados: a localização da página na memória, um valor que indica se a página está suja (ou seja, algo foi escrito nela), um valor que indica se a página possui segunda chance (utilizado no algoritmo de segunda chance) e um valor que guarda a última vez que a página foi acessada (utilizado no algoritmo LRU). Com isso, utilizando os valores passados via

argumentos, é feito o cálculo do tamanho da tabela de páginas e a tabela é armazenada na forma de um array composto dessas structs. Já para armazenar a memória física, foi calculado o seu tamanho, também utilizando os argumentos passados, e ela foi armazenada na forma de um array de valores unsigned, onde cada posição dela armazena a página que está contida naquela posição. Além dessas duas estruturas, uma lista foi criada para armazenar a ordem de acesso das páginas (utilizado no algoritmo FIFO).

Com as estruturas da tabela de páginas e da memória alocadas, o próximo passo foi fazer a leitura do arquivo e alocar as páginas para suas respectivas posições. Para esse passo, foram utilizados dois códigos fornecidos na especificação do trabalho, sendo eles o código de leitura do arquivo e o código para calcular o offset da página (sendo que esse cálculo é feito antes de entrar no loop de leitura do arquivo). Para cada página lida, sua posição na tabela de páginas é calculada, utilizando o valor de offset encontrado previamente e então uma sequência de operações é feita nessa página.

Primeiramente verificamos se a operação é uma operação de escrita e, caso seja, a página é marcada como suja. Ou seja, é uma página que precisará ser reescrita na tabela após sair da memória física. Após isso, o valor de último acesso desta página é atualizado e, finalmente, conferimos se essa página está presente na memória física por meio do campo que indica a posição dessa página na memória. Caso esse campo possua um valor válido de posição de memória, é considerado que houve um hit na memória, ou seja, a página está presente na memória, e com isso o bit de segunda chance dessa página é trocado para 1, indicando que aquela página terá uma segunda chance no algoritmo de mesmo nome. Porém, se o campo possuir um valor inválido de posição de memória, significa que a página não está presente na memória física, e nesse caso temos um page fault.

Após identificar a falha de página, inicialmente atribuímos o valor 0 para o bit de segunda chance, indicando que aquela página não terá uma segunda chance no algoritmo de mesmo nome (como é uma página que está entrando agora na memória, ela não tem motivos para receber uma segunda chance nesse momento). Após isso, é checado se ainda existem quadros disponíveis na memória. Caso existam, a página é atribuída para o próximo quadro disponível e a posição de memória desse quadro é guardada no campo da página responsável por armazenar a posição da memória física onde se encontra essa página. Além disso, caso o

algoritmo pedido pelo usuário seja o FIFO, a posição da memória física é inserida no fim da lista criada anteriormente. Porém, se a memória estiver completamente preenchida, um dos algoritmos a seguir será responsável por substituir uma das páginas já presentes na memória pela nova página:

- FIFO (First In, First Out): nesse algoritmo, a página mais antiga a entrar na memória é a primeira página a sair dela. Para a implementação desse algoritmo, é utilizada a lista de páginas, que contém a ordem de entrada das páginas na memória. A primeira página da lista é retirada da lista e da memória, e a nova página é adicionada na última posição da lista e na posição da página retirada na memória. Para esse algoritmo, temos uma complexidade *O(1)* para substituição, pois basta retirar a primeira página da lista e inserir a nova página no final, que são operações com custo constante.
- LRU (Least Recent Used): nesse algoritmo, a página usada há mais tempo é selecionada para sair da memória. A implementação desse algoritmo verifica o valor de último acesso de cada página presente na memória e seleciona a página com o menor valor, ou seja, a página acessada há mais tempo do que as outras. Após selecionar essa página, ela é removida da memória e a nova página é inserida no lugar dela. Para esse algoritmo, temos a complexidade *O(n)*, sendo n o número de frames na memória. Como é necessário conferir o último acesso de todas as posições para encontrar a página usada há mais tempo, isso justifica a complexidade.
- 2a (Segunda Chance): nesse algoritmo, a página recebe uma segunda chance de continuar na memória, caso tenha sido acessada após sua entrada na memória. Nesse caso, se uma página for selecionada e não tiver uma segunda chance, essa página é escolhida para ser substituída. Porém, se ela tiver uma segunda chance, ela não é retirada imediatamente. Caso ela seja selecionada novamente, sem que tenham ocorrido acessos a ela nesse período, ela então é escolhida para ser substituída. A implementação desse algoritmo verifica o valor de segunda chance de cada página presente na memória e seleciona a primeira página sem segunda chance encontrada. Caso a memória toda seja percorrida, e todas as páginas tenham segunda chance, uma nova varredura é feita para selecionar uma página (como uma varredura completa já foi feita, a próxima garantidamente terá uma página sem segunda chance). Além disso, essa varredura nem sempre começa da

primeira posição de memória. Um ponteiro é guardado para indicar a posição de onde terminou a última varredura e, dessa forma, a nova pesquisa se inicia da posição indicada por esse ponteiro. Para esse algoritmo, temos a complexidade *O(n)*, sendo n o número de frames na memória. No pior caso, é necessário conferir a segunda chance de todas as posições para encontrar uma página sem segunda chance, o que justifica a complexidade.

- Personalized (nosso algoritmo): nesse algoritmo, uma página aleatória é selecionada, porém é verificada se ela possui uma segunda chance. O resto do funcionamento segue o mesmo padrão do algoritmo acima. A implementação desse algoritmo seleciona uma posição aleatória da memória física e verifica se a página da memória possui uma segunda chance. Caso possua, essa segunda chance é removida e uma nova posição de memória é selecionada. Porém, caso a página não possua segunda chance, ela é selecionada para substituição, e a nova página é escrita no lugar dela. Para esse algoritmo, temos a complexidade *O(n)*, sendo n o número de frames na memória. No pior caso, todas as páginas terão segunda chance e o algoritmo seguirá uma ordem que não repete páginas, sendo necessário conferir a segunda chance de todas as posições para encontrar uma página sem segunda chance, o que justifica a complexidade.

Ao final da substituição, o campo que indica a localização da página na memória é invalidado, indicando que essa página não está mais na memória, e é verificado se a página estava suja. Caso estivesse, ela é limpa (trocando o bit de página suja) e a escrita dessa página é contabilizada.

Ao final da execução do algoritmo, os números de páginas lidas e escritas são impressos, bem como uma tabela indicando o estado final da memória.

3 - Análise do Desempenho

Para a análise de desempenho, realizou-se alguns experimentos, a fim de se observar o comportamento das páginas lidas e escritas, variando, dessa forma, o tamanho da memória e das páginas, assim como a técnica de reposição e o arquivo de entrada.

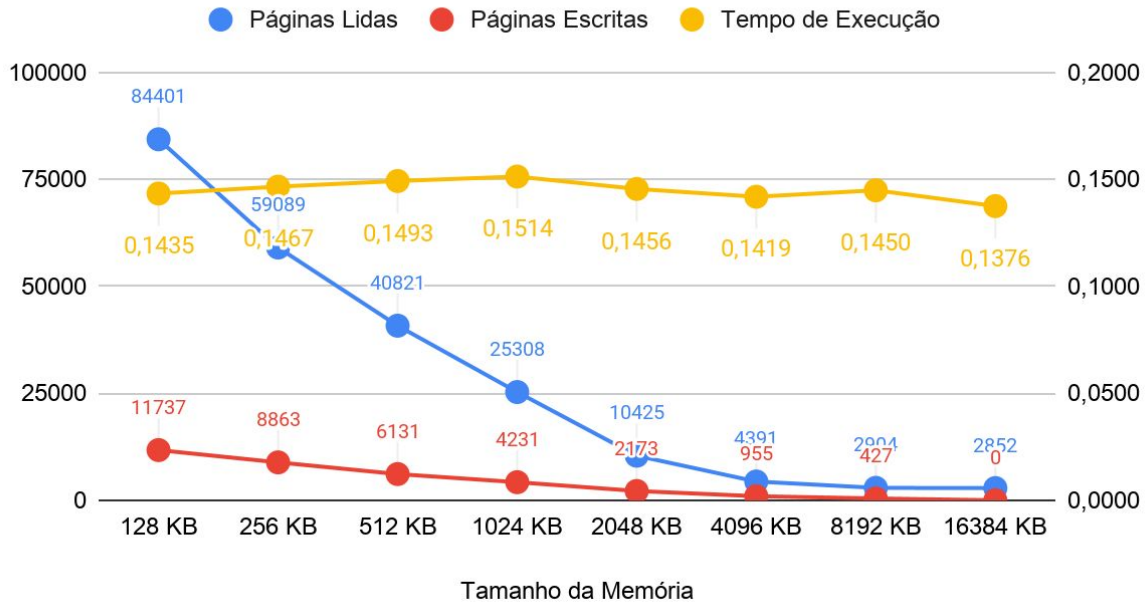
Assim, para cada um dos algoritmos (lru, 2a, fifo e o personalizado), mantendo-se o tamanho das páginas constantes a 4KB e variando o tamanho da

memória de 128 KB até 16348 KB, gerou-se um gráfico, que pode ser visualizado mais a frente no documento.

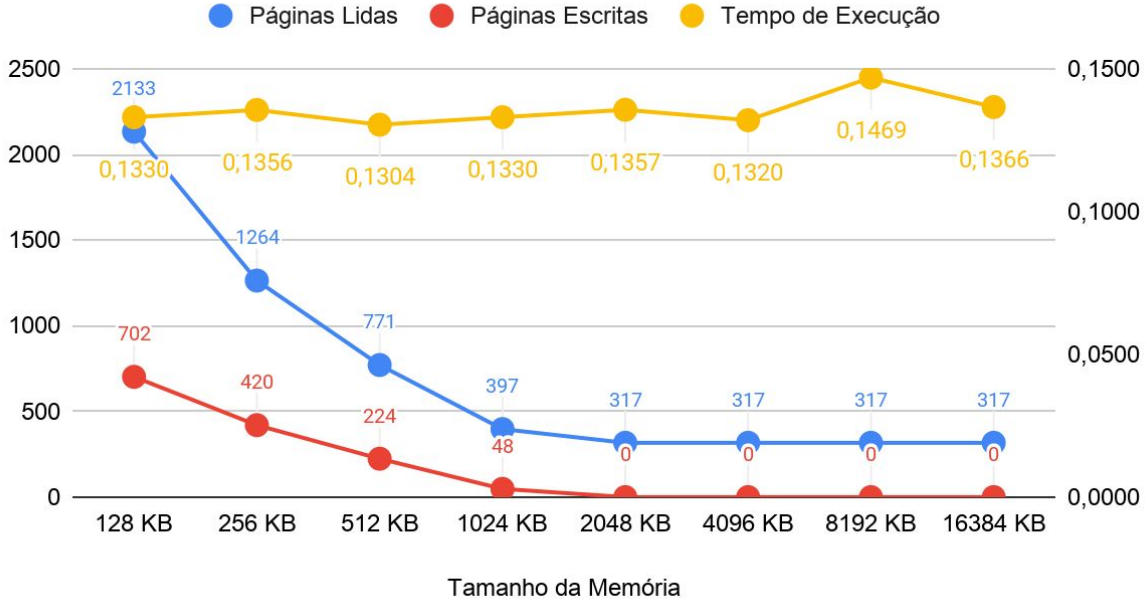
Com eles, podemos observar que, de maneira geral, quanto mais memória disponível para a alocação das páginas, menor o número de substituições necessárias. No caso do simulador desenvolvido, o tempo não é afetado, mas em um sistema real, tais operações são extremamente custosas, o que poderia impactar severamente (ou até mesmo inviabilizar) o desempenho.

Ademais, para um sistema com a quantidade de memória constante, um tamanho de páginas menor é mais vantajoso, justamente pelo motivo supracitado. É também interessante mencionar que o arquivo do compressor tem menores valores de leitura e escrita, o que indica um baixo uso de páginas da memória.

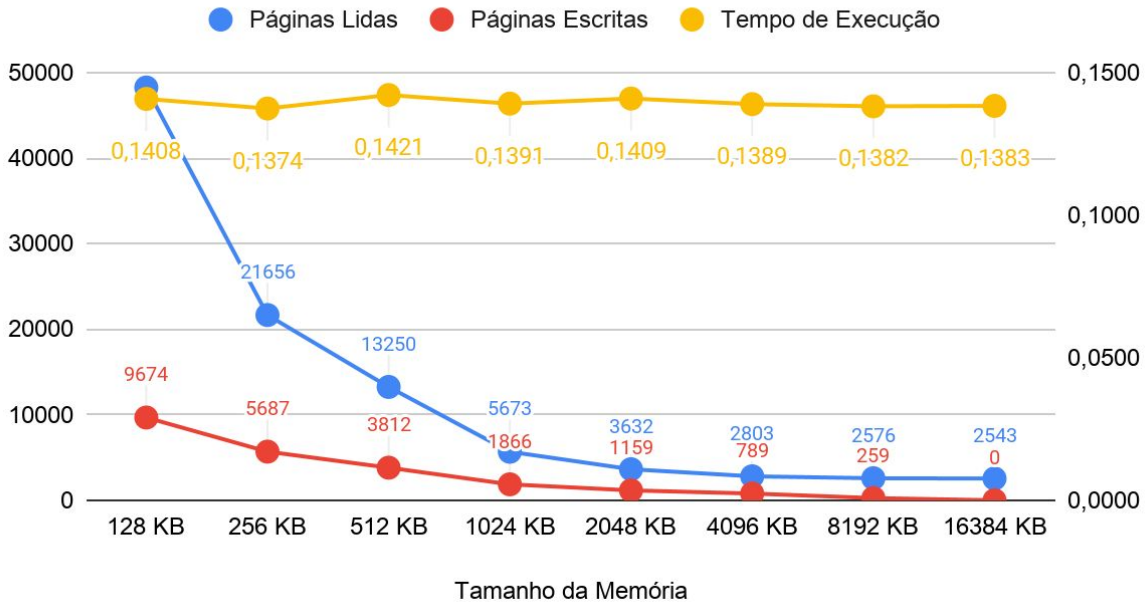
LRU - Tamanho da Página Constante - Compilador



LRU - Tamanho da Página Constante - Compressor

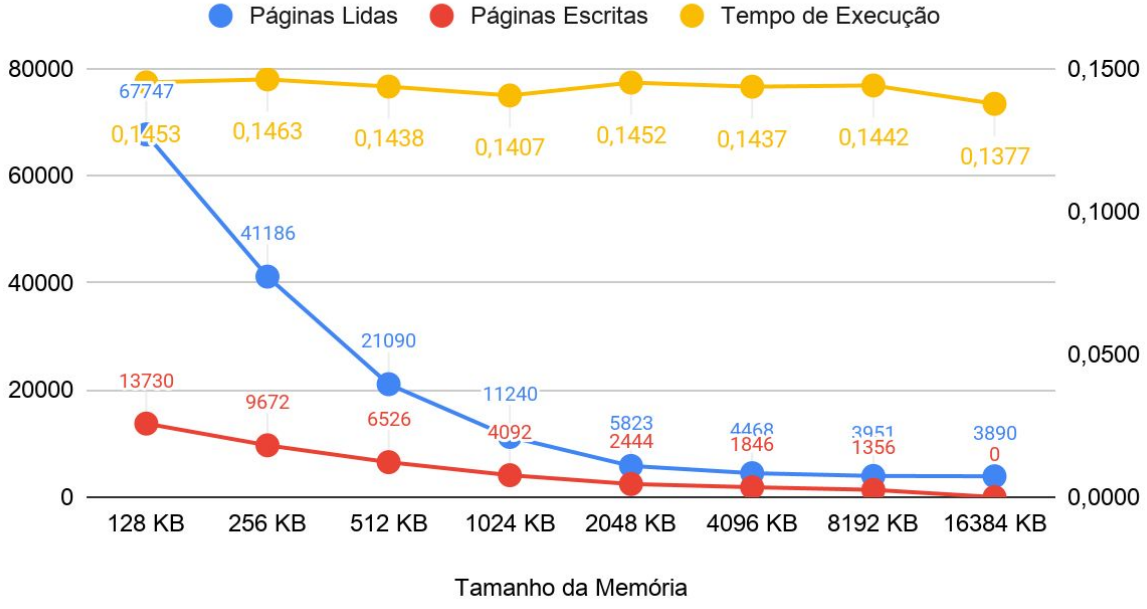


LRU - Tamanho da Página Constante - Matriz

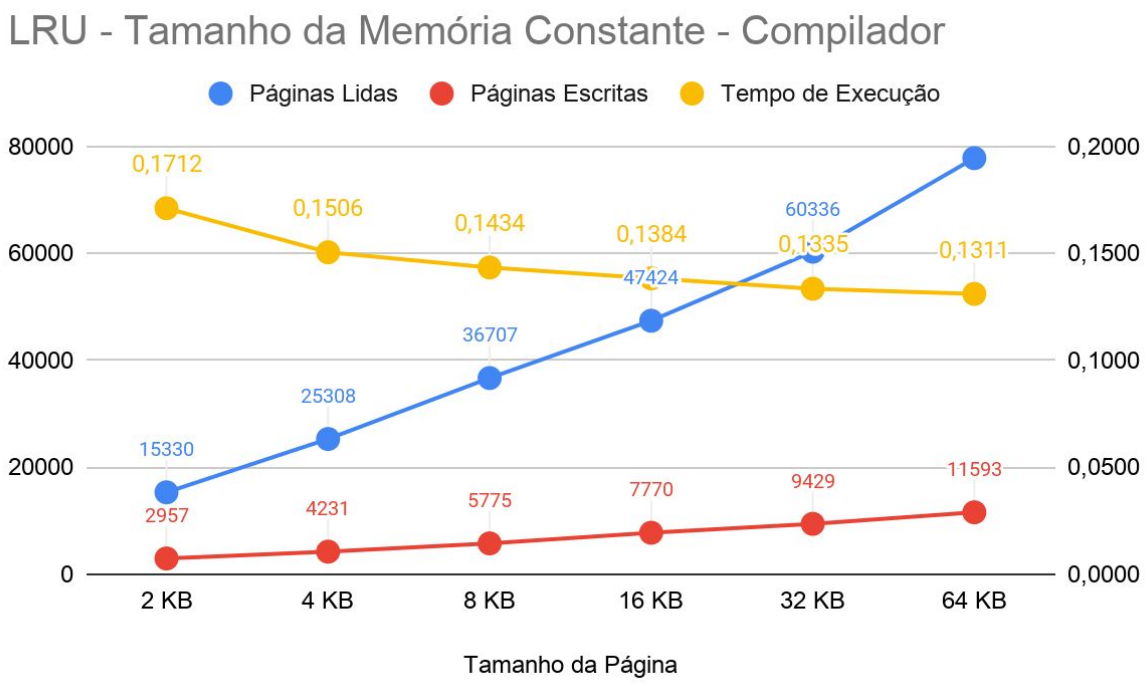


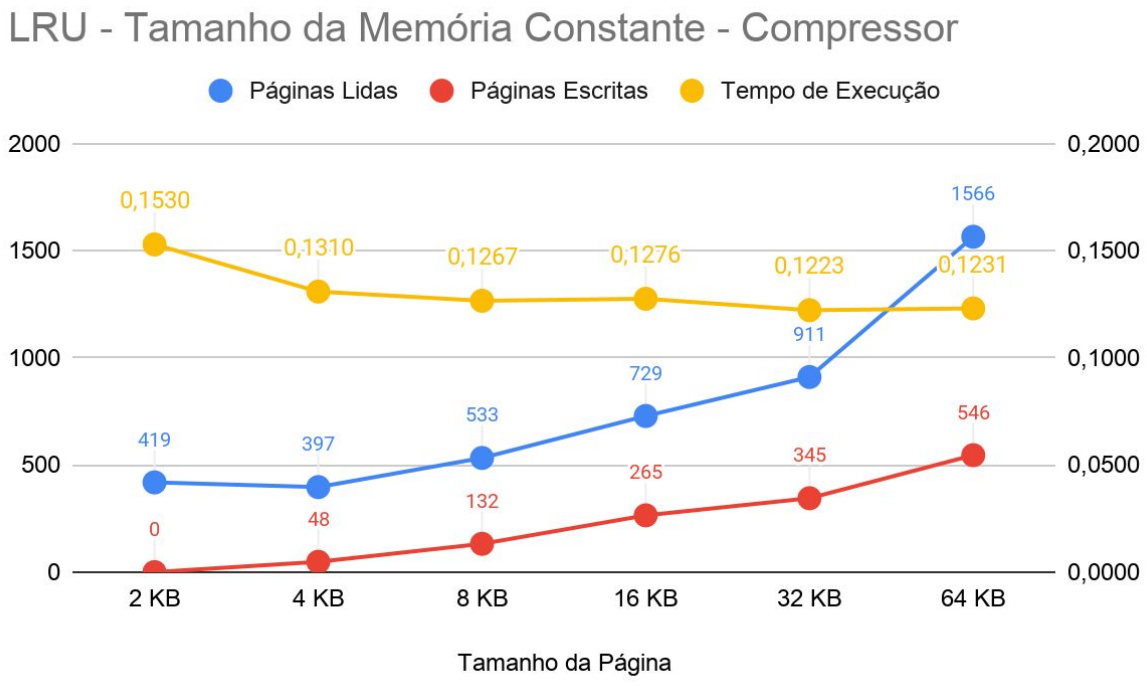
Tamanho da Memória

LRU - Tamanho da Página Constante - Simulador

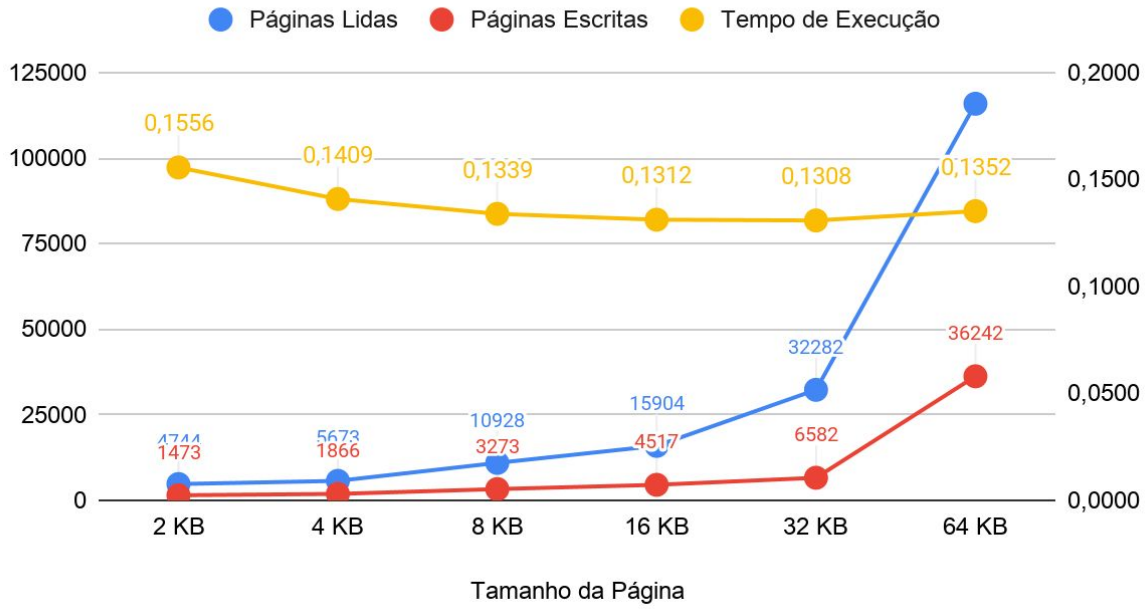


Tamanho da Memória

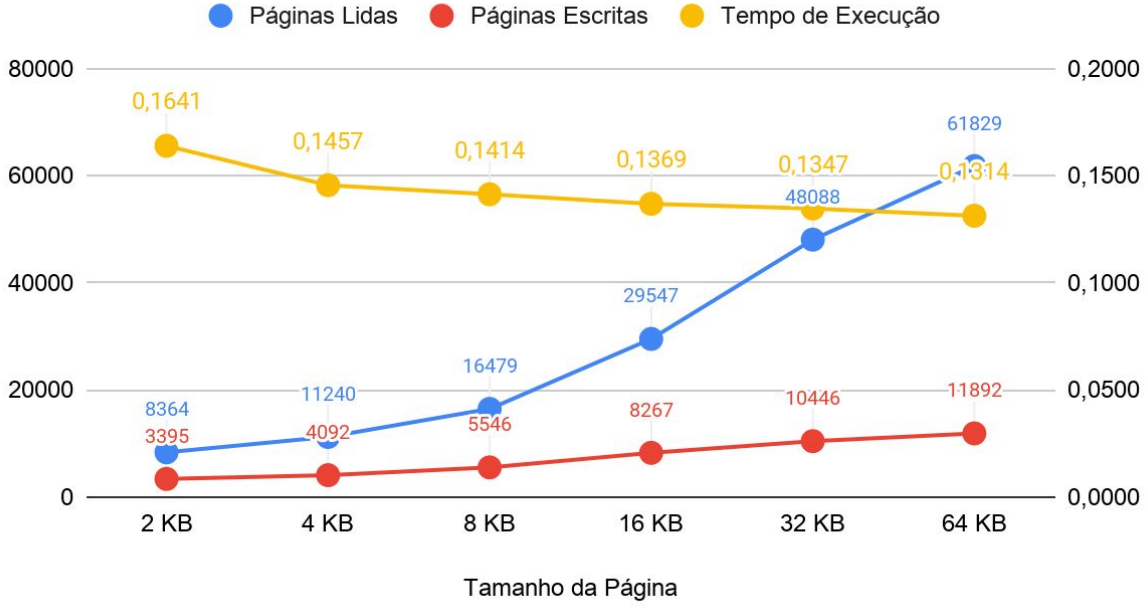

LRU - Tamanho da Memória Constante - Compilador
Páginas Lidas
Páginas Escritas
Tempo de Execução
80000
60000
40000
20000
0
0,2000
0,1500
0,1000
0,0500
0,0000
0,1712
0,1506
0,1434
0,1384
0,1335
0,1311
15330
25308
36707
47424
60336
2957
4231
5775
7770
9429
11593
2 KB
4 KB
8 KB
16 KB
32 KB
64 KB
Tamanho da Página


LRU - Tamanho da Memória Constante - Compressor
Páginas Lidas
Páginas Escritas
Tempo de Execução
2000
1500
1000
500
0
0,2000
0,1500
0,1000
0,0500
0,0000
0,1530
0,1310
0,1267
0,1276
0,1223
0,1231
419
397
533
729
911
1566
0
48
132
265
345
546
2 KB
4 KB
8 KB
16 KB
32 KB
64 KB
Tamanho da Página

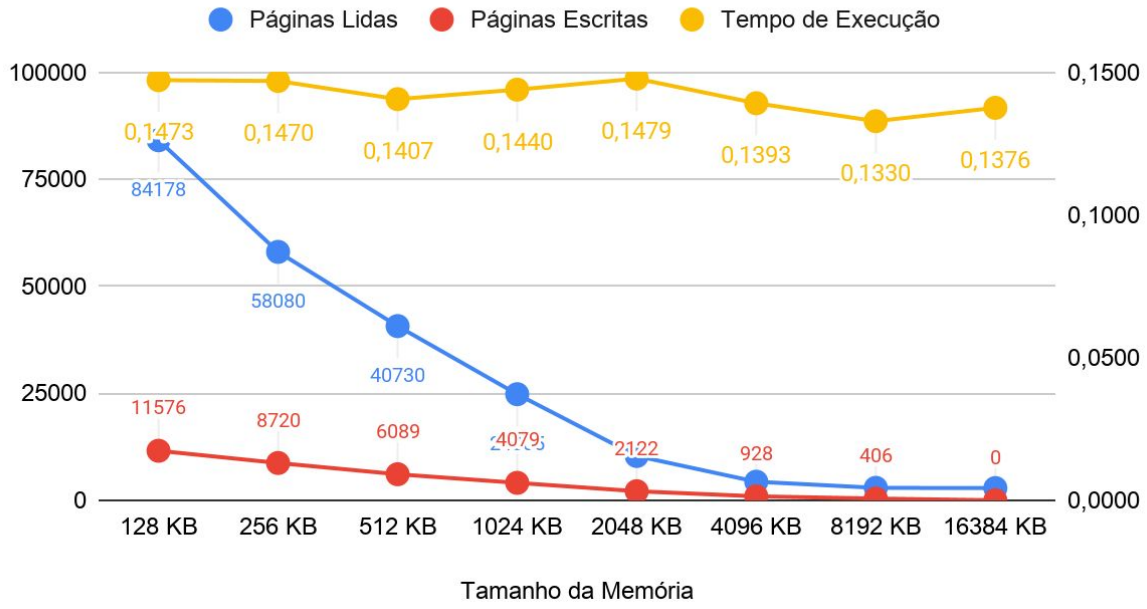
LRU - Tamanho da Memória Constante - Matriz



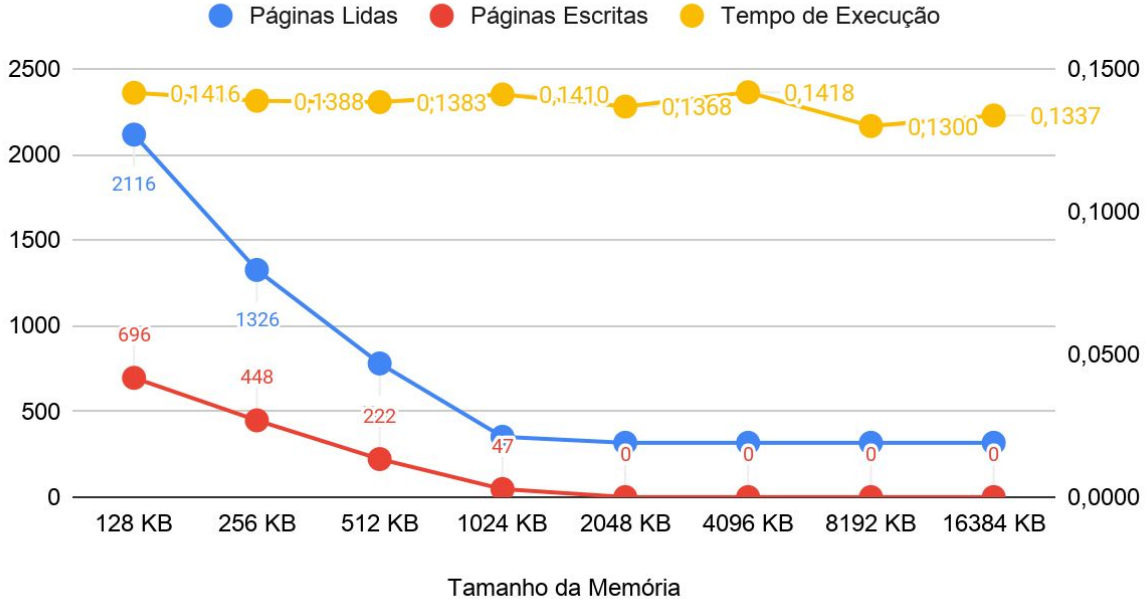
LRU - Tamanho da Memória Constante - Simulador



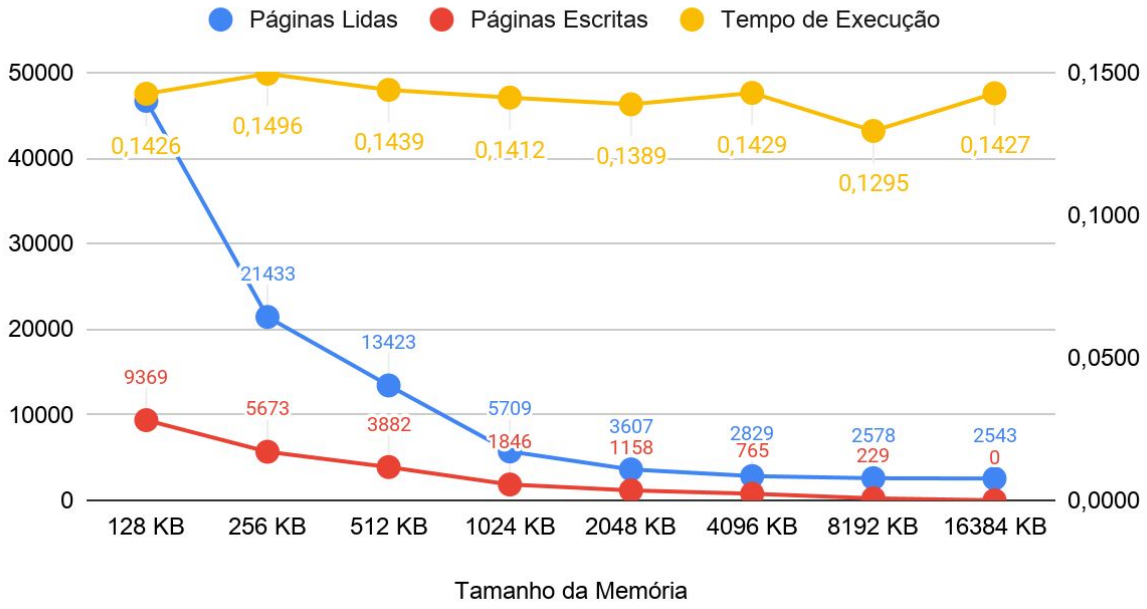
2a - Tamanho da Página Constante - Compilador



2a - Tamanho da Página Constante - Compressor

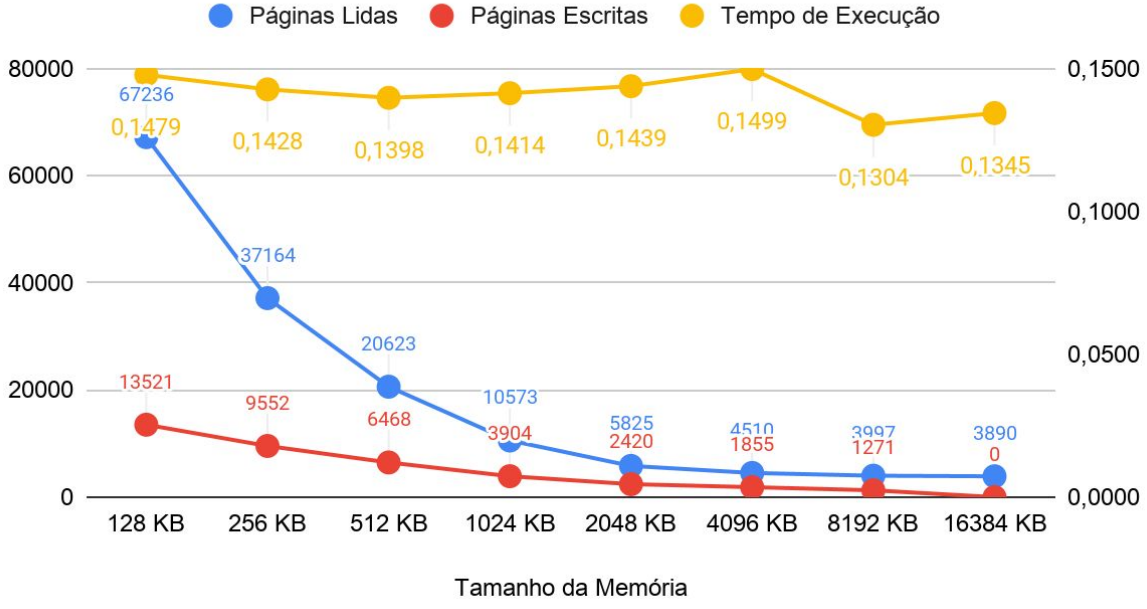


2a - Tamanho da Página Constante - Matriz



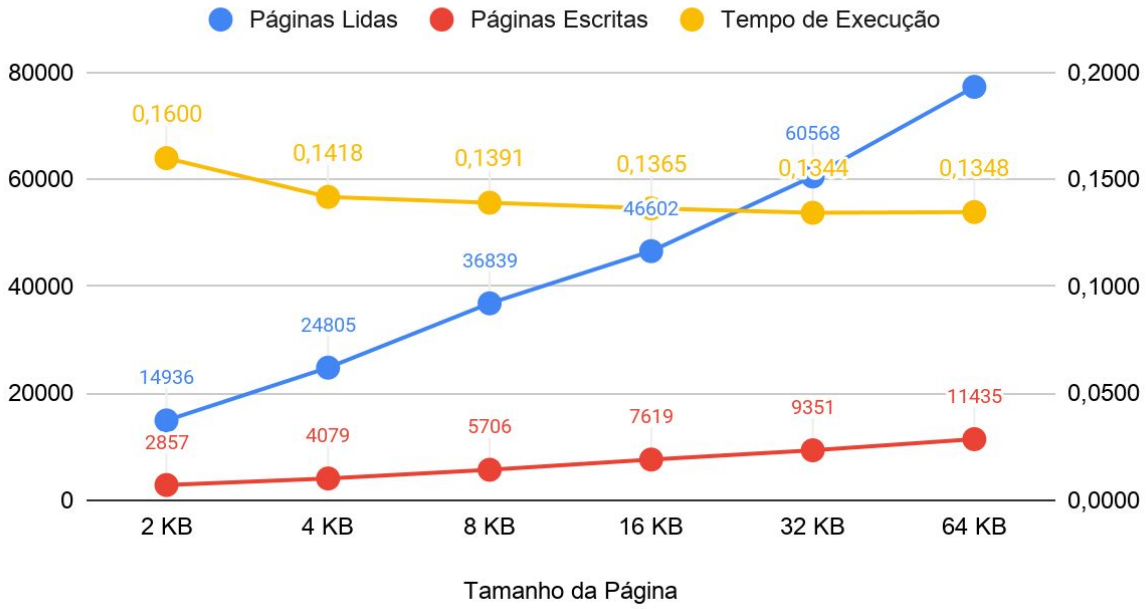
Tamanho da Memória

2a - Tamanho da Página Constante - Simulador

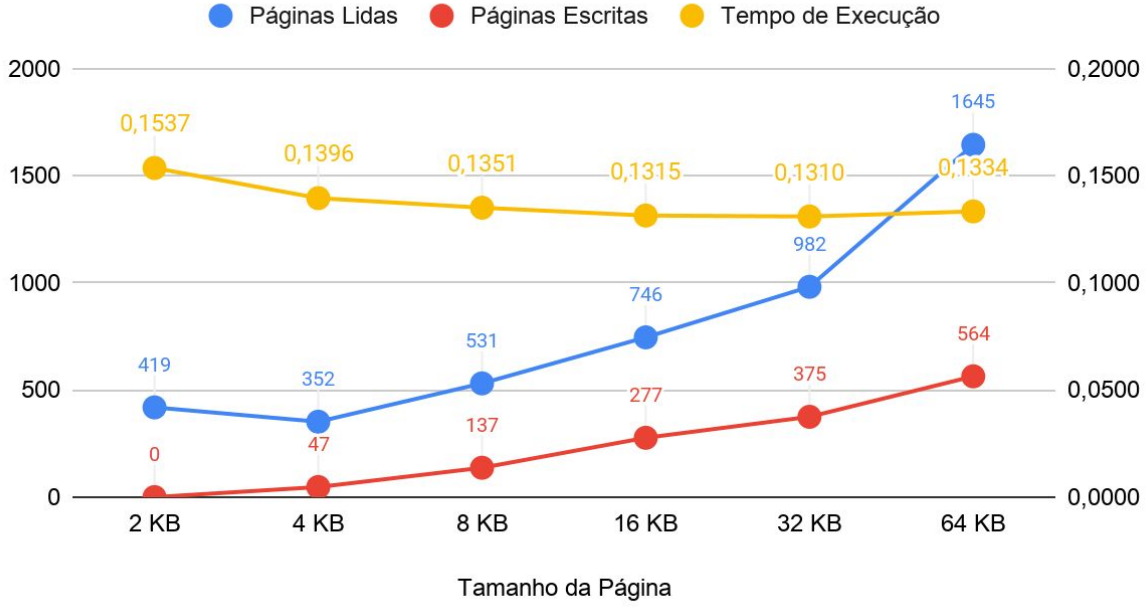


Tamanho da Memória

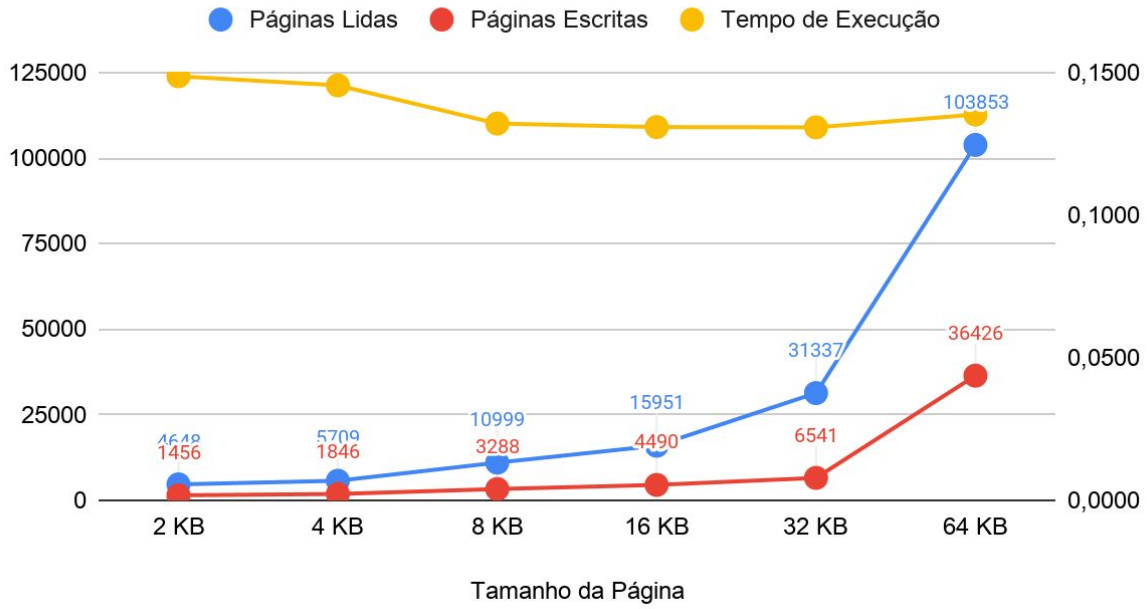
2a - Tamanho da Memória Constante - Compilador



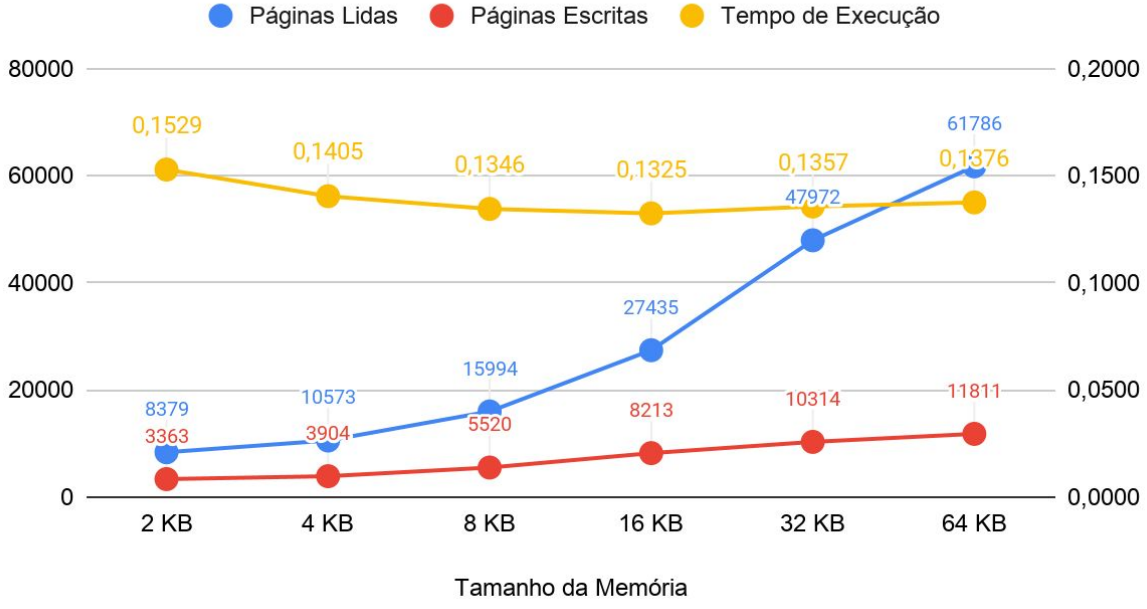
2a - Tamanho da Memória Constante - Compressor

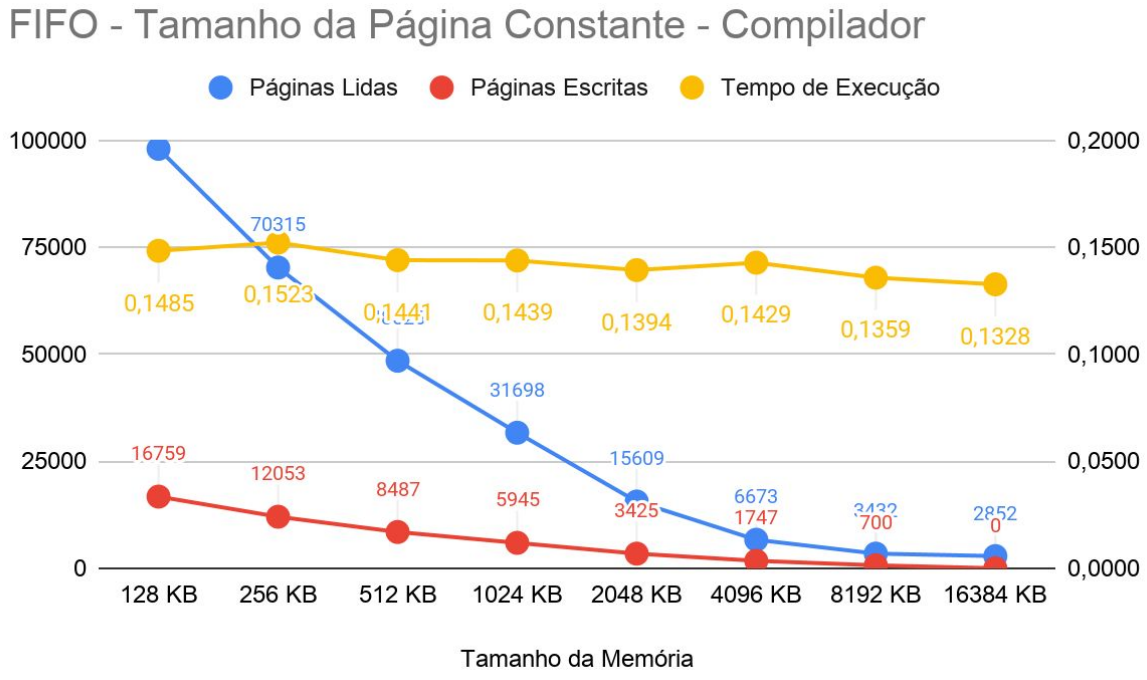


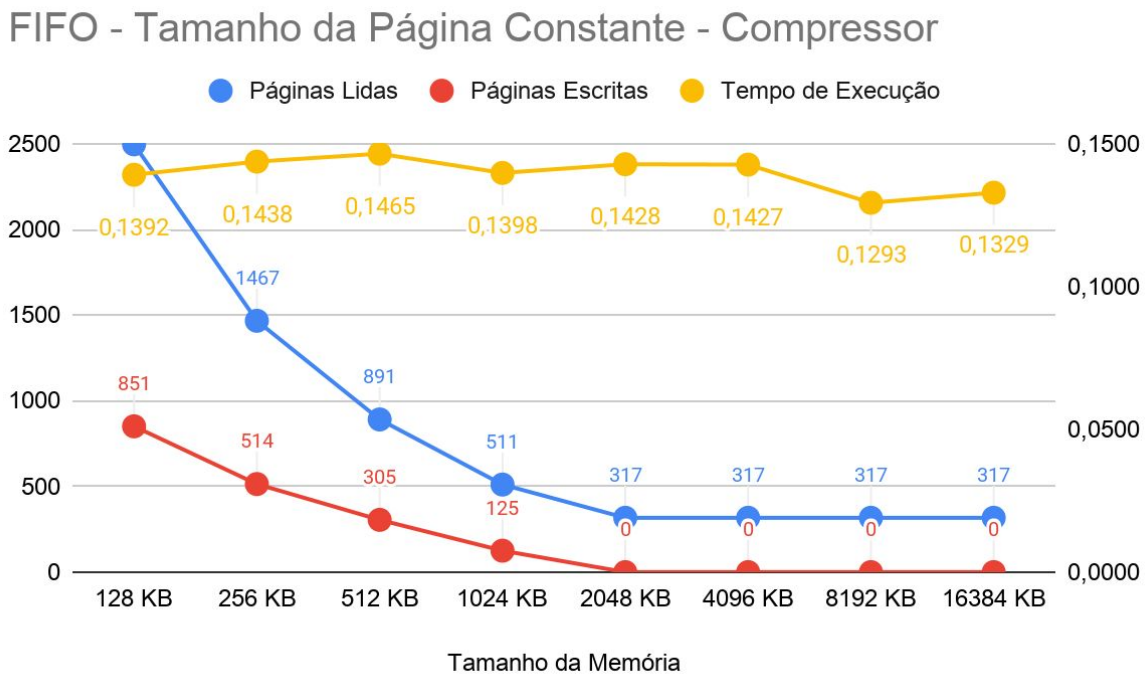
2a - Tamanho da Página Constante - Matriz



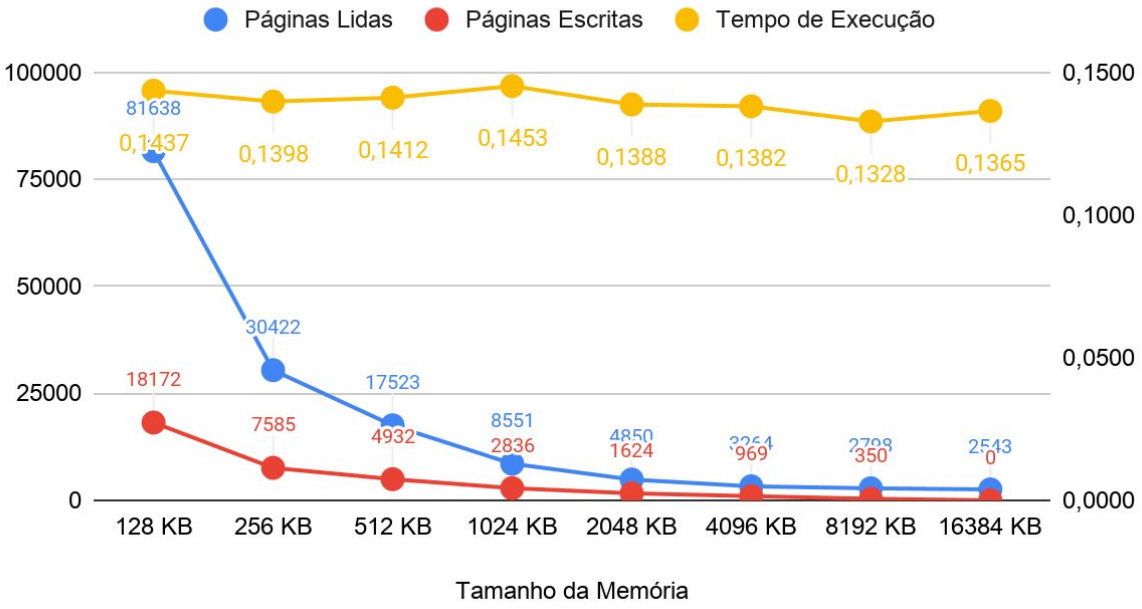
2a - Tamanho da Memória Constante - Simulador



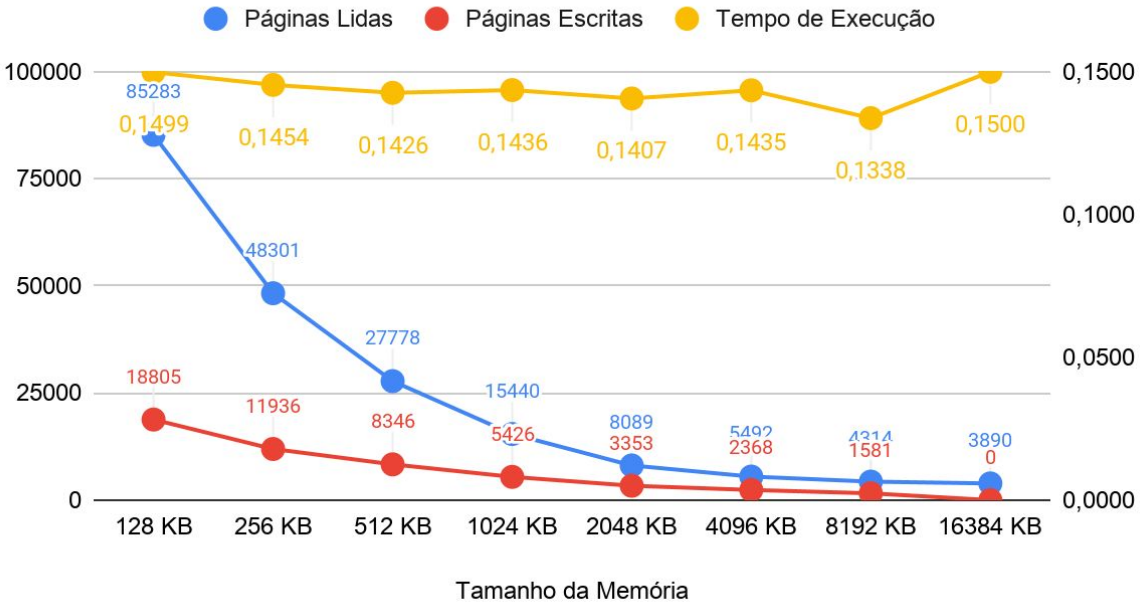

FIFO - Tamanho da Página Constante - Compilador
Páginas Lidas
Páginas Escritas
Tempo de Execução
100000
75000
50000
25000
0
0,2000
0,1500
0,1000
0,0500
0,0000
70315
31698
15609
6673
3432
2852
16759
12053
8487
5945
3425
1747
700
0
0,1485
0,1523
0,1441
0,1439
0,1394
0,1429
0,1359
0,1328
128 KB
256 KB
512 KB
1024 KB
2048 KB
4096 KB
8192 KB
16384 KB
Tamanho da Memória


FIFO - Tamanho da Página Constante - Compressor
Páginas Lidas
Páginas Escritas
Tempo de Execução
2500
2000
1500
1000
500
0
0,1500
0,1000
0,0500
0,0000
1467
891
511
317
317
317
317
851
514
305
125
0
0
0
0
0,1392
0,1438
0,1465
0,1398
0,1428
0,1427
0,1293
0,1329
128 KB
256 KB
512 KB
1024 KB
2048 KB
4096 KB
8192 KB
16384 KB
Tamanho da Memória

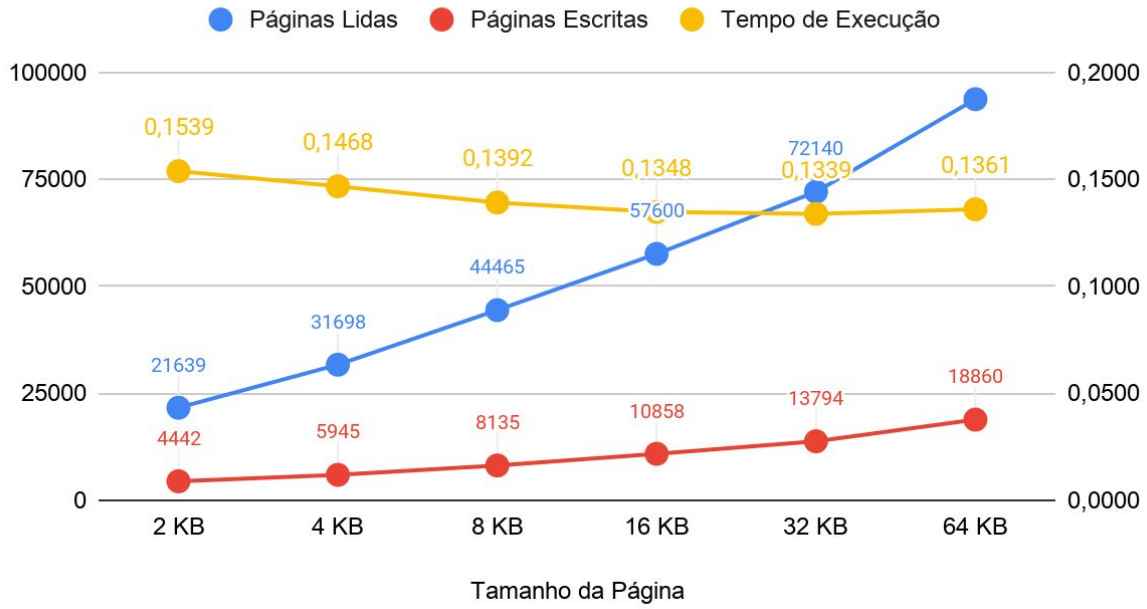
FIFO - Tamanho da Página Constante - Matriz



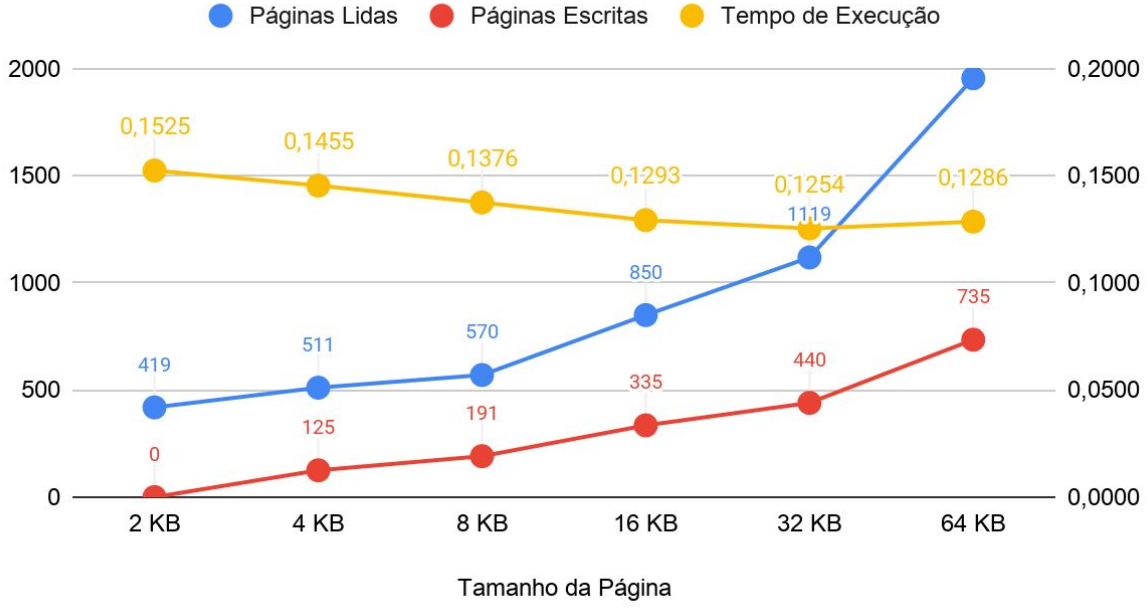
FIFO - Tamanho da Página Constante - Simulador



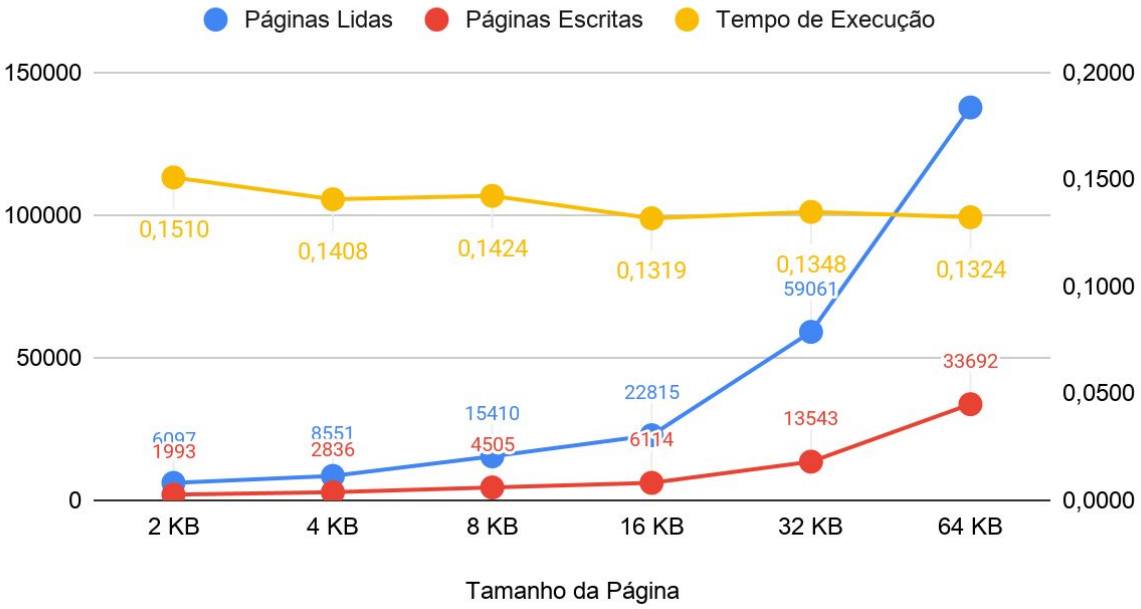
FIFO - Tamanho da Memória Constante - Compilador



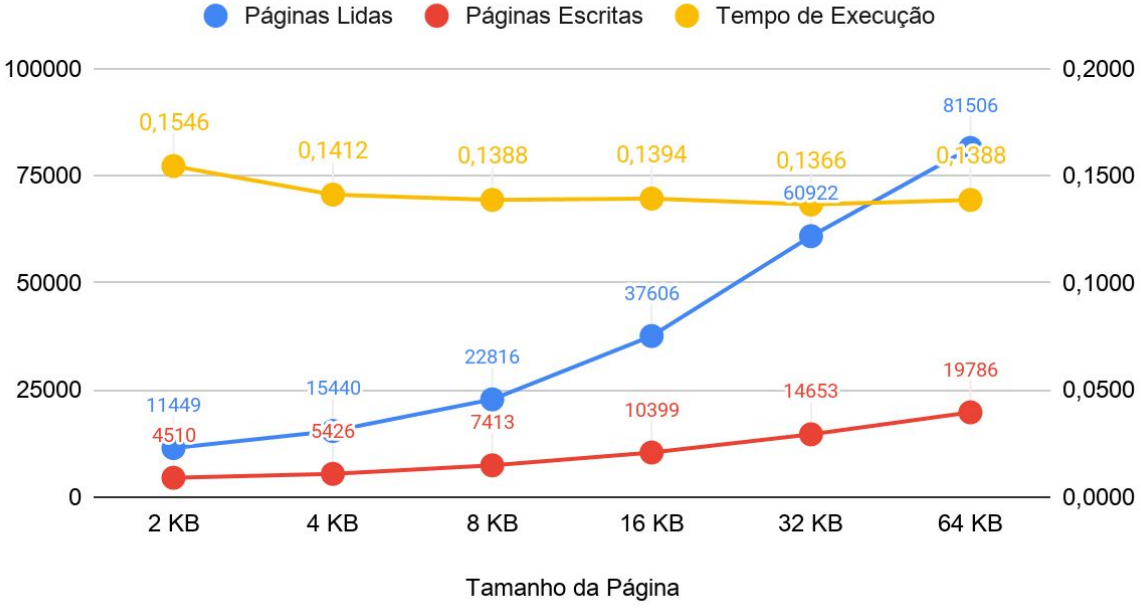
FIFO - Tamanho da Memória Constante - Compressor



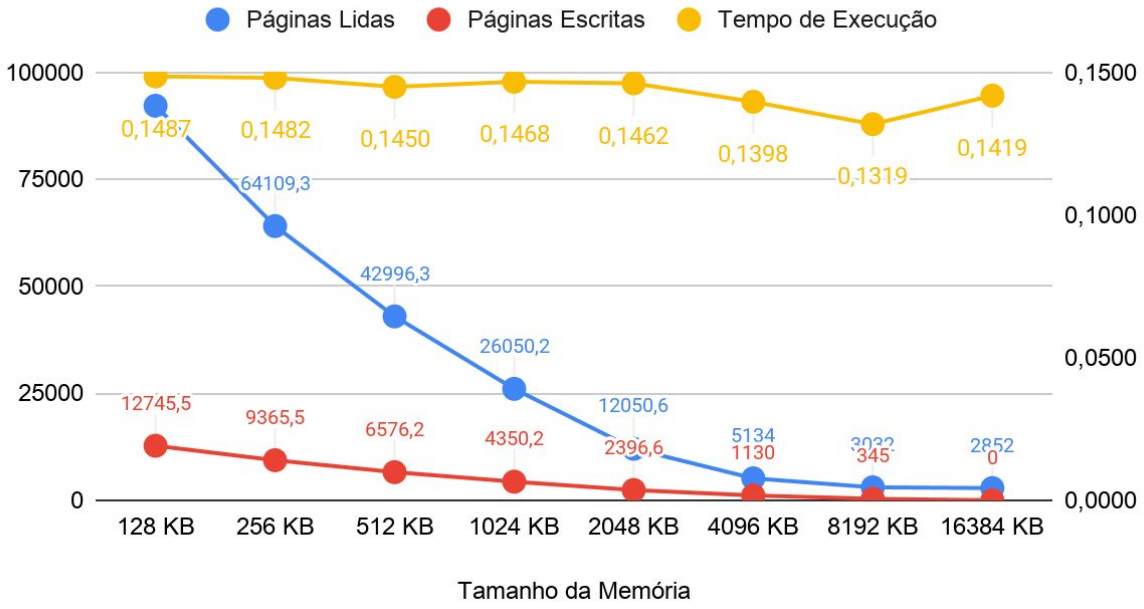
FIFO - Tamanho da Página Constante - Matriz



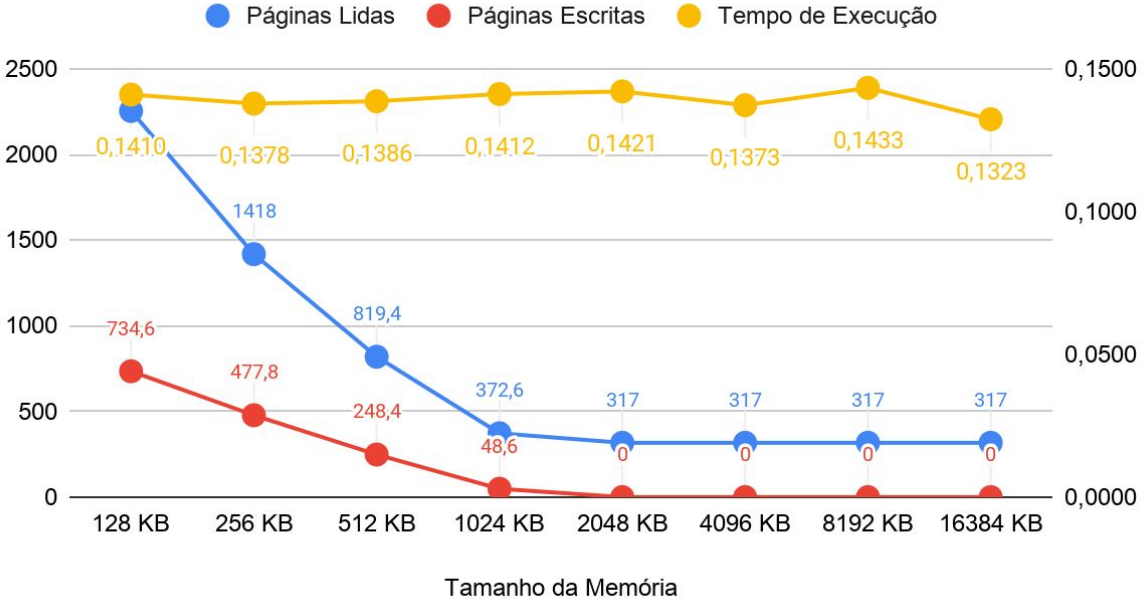
FIFO - Tamanho da Memória Constante - Simulador



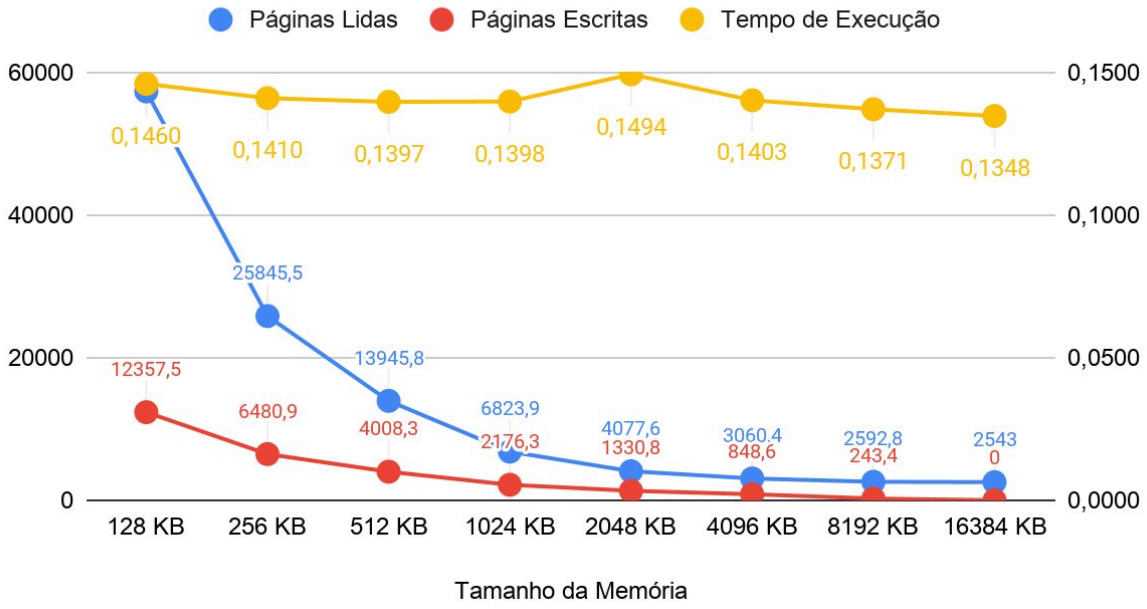
Personalizado - Tamanho da Página Constante - Compilador



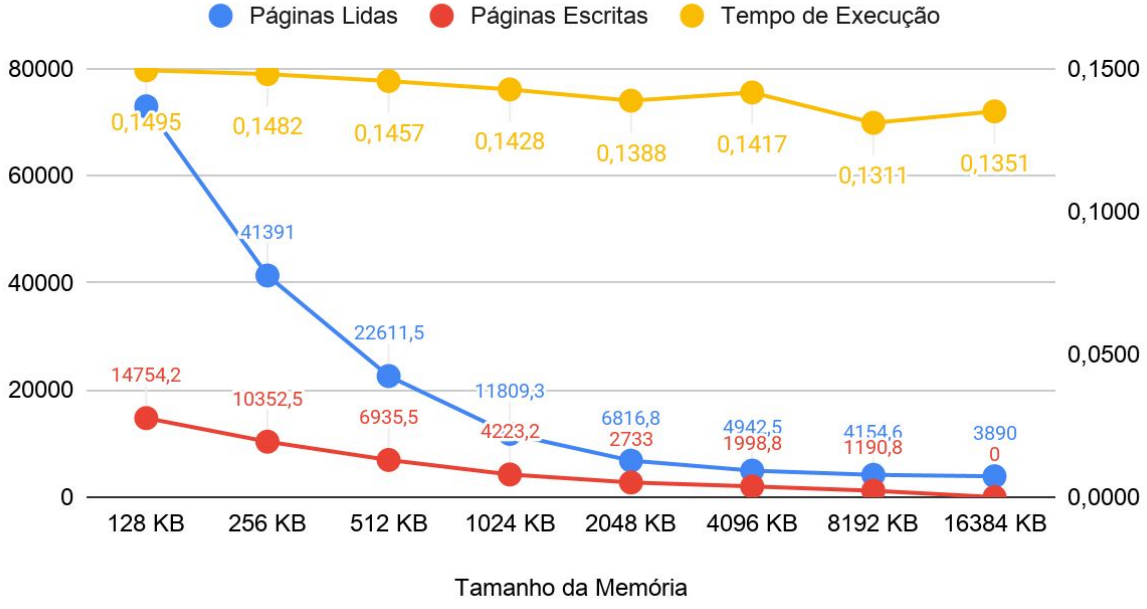
Personalizado - Tamanho da Página Constante - Compressor

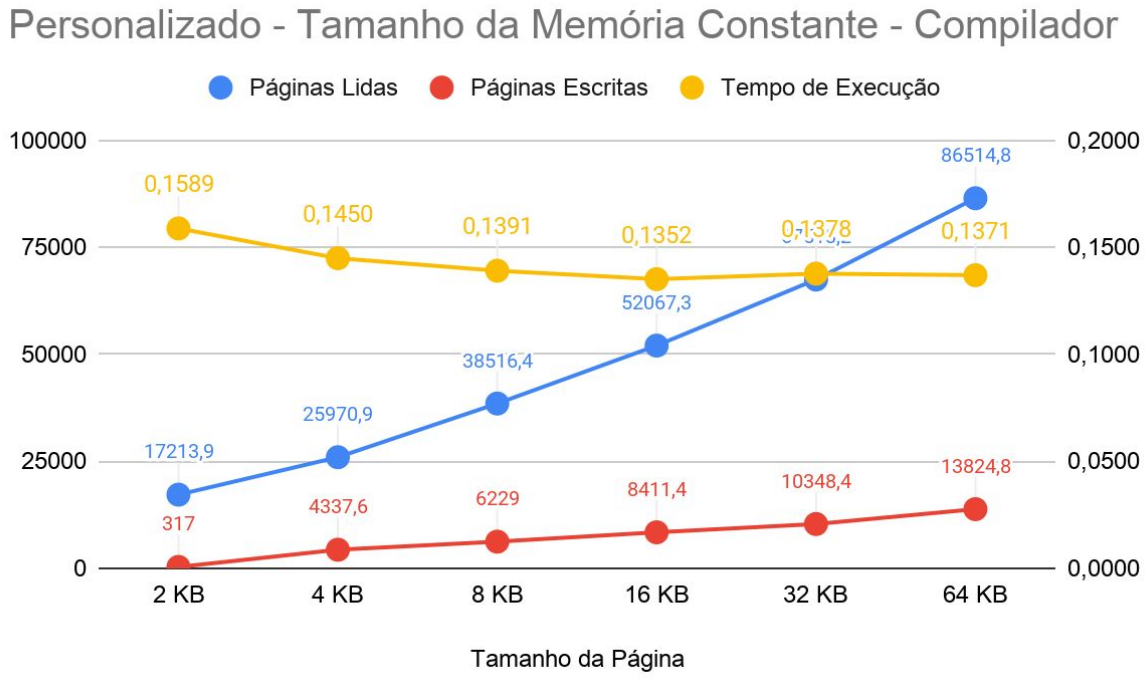


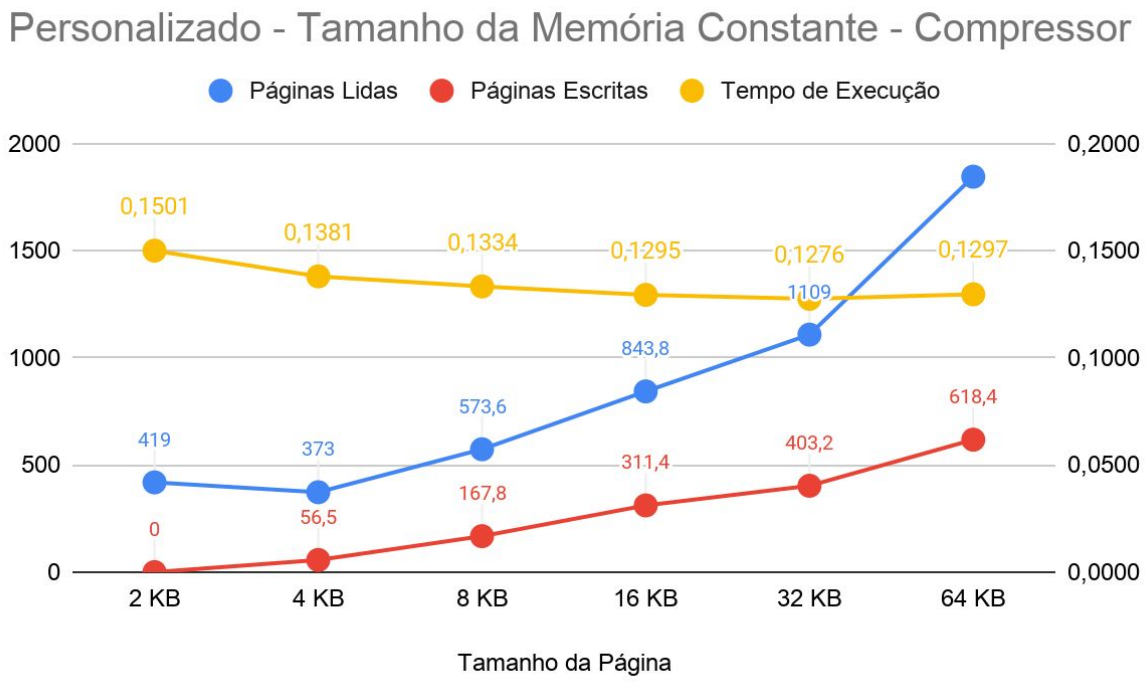
Personalizado - Tamanho da Página Constante - Matriz

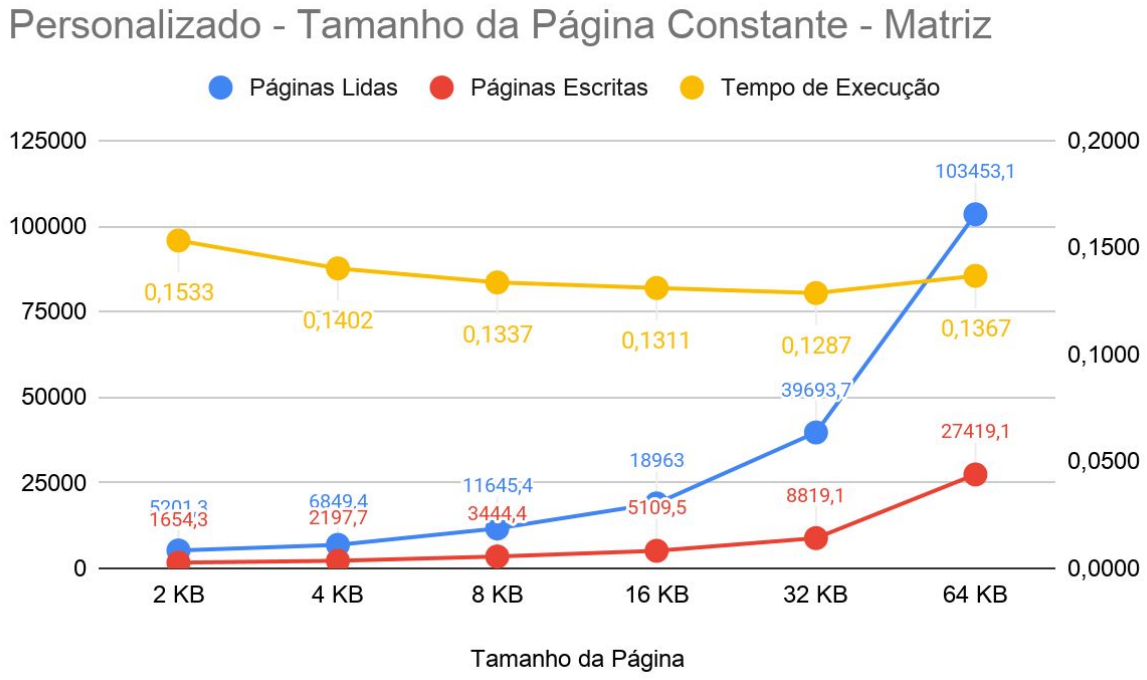


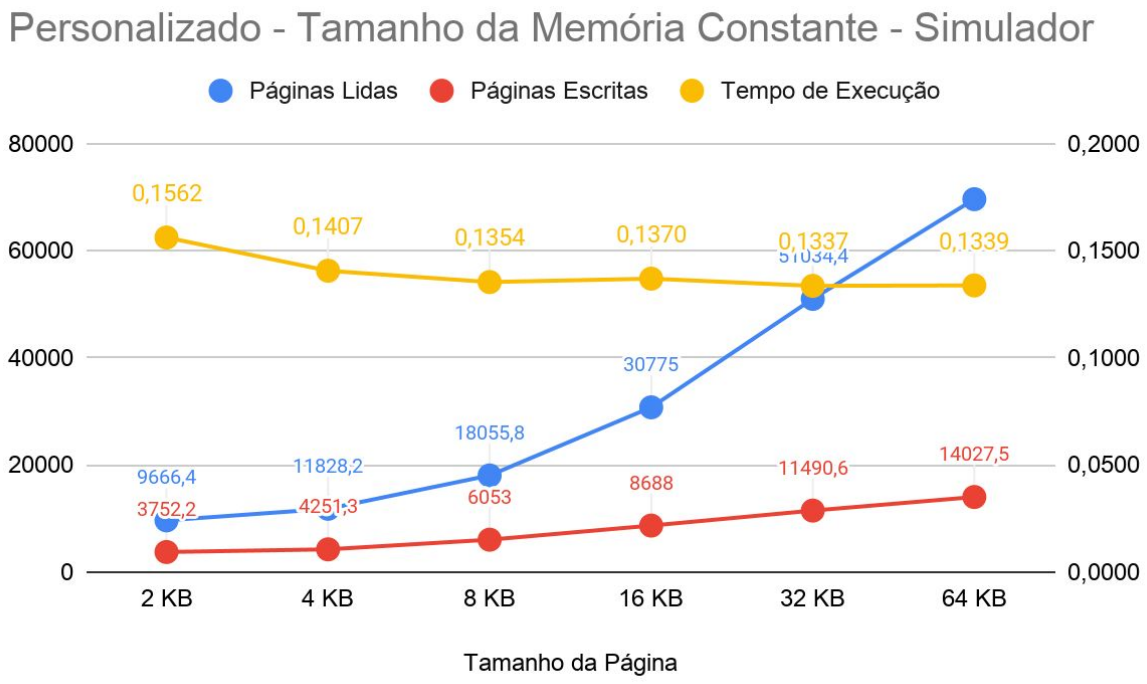
Personalizado - Tamanho da Página Constante - Simulador




Personalizado - Tamanho da Memória Constante - Compilador
Páginas Lidas
Páginas Escritas
Tempo de Execução
100000
75000
50000
25000
0
0,2000
0,1500
0,1000
0,0500
0,0000
0,1589
0,1450
0,1391
0,1352
0,1371
17213,9
25970,9
38516,4
52067,3
86514,8
317
4337,6
6229
8411,4
10348,4
13824,8
2 KB
4 KB
8 KB
16 KB
32 KB
64 KB
Tamanho da Página


Personalizado - Tamanho da Memória Constante - Compressor
Páginas Lidas
Páginas Escritas
Tempo de Execução
2000
1500
1000
500
0
0,2000
0,1500
0,1000
0,0500
0,0000
0,1501
0,1381
0,1334
0,1295
0,1276
0,1297
419
373
573,6
843,8
1109
0
56,5
167,8
311,4
403,2
618,4
2 KB
4 KB
8 KB
16 KB
32 KB
64 KB
Tamanho da Página





4 - Desafios

Dentre os principais desafios enfrentados durante o trabalho, destacam-se os aspectos relacionados ao gerenciamento de leitura e escrita de páginas na memória. Apesar dos algoritmos parecerem simples na teoria, a implementação dos

mesmos é desafiadora, principalmente considerando todas as condições que devem ser verificadas para que eles funcionem corretamente.

5 - Conclusões

Para além da prática com a linguagem C, a implementação do trabalho foi bastante proveitosa para compreender efetivamente os conceitos vistos até o presente momento na disciplina. O aprendizado se deu, sobretudo, no desafio citado na seção anterior, mas também com outras dúvidas dos colegas apresentadas no grupo do WhatsApp. Com isso, considera-se que o trabalho foi de grande proveito para o processo de aprendizagem.

6 - Referências Bibliográficas

- Second Chance (or Clock) Page Replacement Policy - https://www.geeksforgeeks.org/second-chance-or-clock-page-replacement-policy/