**Relatório Trabalho Final Estrutura de Dados - 2024.2**

*Alunos: Henrique Rocha, Matheus Marques e Pablo Araújo*

*Professor: Pedro Moura*

*Objetivo:* Desenvolver uma implementação eficiente de Matriz Esparsa, que explore o conceito de Lista Encadeada Simples estudado ao longo da disciplina. A ideia consiste em aproveitar a alocação dinâmica decorrente do uso da Lista Encadeada, para não só consumir menos memória, mas também desenvolver métodos mais eficientes e objetivos para realizar as operações realizadas nas matrizes. Com isso, nós criamos duas classes, uma utilizando o conceito básico de matriz esparsa estática e outra utilizando a representação via lista encadeada, testamos alguns dos métodos propostos para serem desenvolvidos e analisamos seu desempenho em termos de tempo de cada operação testada.

*Matriz Esparsa (definição):* Uma matriz que contém uma quantidade considerável de elementos nulos ou que não constam na matriz.

*Observações:* Para desenvolver as implementações, nós criamos um repositório no github que constava das atualizações realizadas por cada membro, contendo as classes e a main em que se realizaram qualquer tipo de testagem referente a cada método. Além disso, a linguagem principal utilizada foi Java (versão: Java 8), através da IDE IntelliJ (versão 2024.1.1). Para realizarmos os testes, ainda utilizamos outras ferramentas adicionais, sendo elas: contourpy (versão1.3.1), cycler (versão 0.12.1), fonttools (versão 4.55.6), kiwisolver (versão 1.4.8), matplotlib (versão 3.10.0), numpy (versão 2.2.2), packaging (versão 24.2), pandas (versão 2.2.3), pillow (versão 11.1.0), pyparsing (versão 3.2.1), python-dateutil (versão 2.9.0), pytz (versão 2024.2), seaborn (versão 0.13.2), six (versão 1.17.0) e tzdata (2025.1).

*Estruturas de dados utilizadas:*

-Lista Encadeada: utilizamos o conceito de lista encadeada inserido na implementação de Matrizes Esparsas por meio de um vetor de Elos, inseridos dentro da própria classe referente a esse tipo de matrizes, sendo esses Elos usados em forma de vetor, onde cada posição do vetor aponta para uma linha e cada elemento da linha era o primeiro da lista encadeada.

-Matriz Esparsa Estática: Utilizamos uma implementação básica de vista em disciplinas anteriores como Algoritmos e Programação, onde é criado uma espécie de “vetor duplo”, onde o primeiro elemento seria a linha e o segundo a coluna.

-Matriz Esparsa Linked List: Como dito anteriormente, aproveitamos a ideia de lista para representar as matrizes esparsas, ganhando em desempenho e consumo de memória.

*Métodos criados*

1. Inserir elemento: Insere um elemento em uma determinada posição da matriz, que se tem por meio dos parâmetros de entrada da função;

* Matriz estática: insertElement(int linha, int coluna, int valor).
* Matriz com lista encadeada: insertElement(int linha, int coluna, int valor)
* O método com lista encadeada é mais complexo de ser implementado e tem complexidade maior, uma vez que precisa percorrer a lista até a posição de coluna desejado. Enquanto isso, o método de matriz estática apenas acessa a posição de linha e coluna em tempo constante e insere o valor desejado.

1. Remover um elemento: torna um elemento de determinada posição da matriz nulo, sendo essa posição também recebida por parâmetro.

* Matriz estática: removeElement(int linha, int coluna).
* Matriz com lista encadeada: removeElement(int linha, int coluna).
* Novamente, enquanto o método de matriz estática acessa a posição em tempo constante, o método utilizando listas encadeadas precisa percorrer a lista para encontrar a coluna correta.

1. Busca por um elemento específico: verifica se em determinada posição há ou não um elemento ocupando aquele espaço.

* Matriz estática: searchElement(int i, int j).
* Matriz com lista encadeada: searchElement(int linha, int coluna).
* Novamente, enquanto o método de matriz estática acessa a posição em tempo constante, o método utilizando listas encadeadas precisa percorrer a lista para encontrar a coluna correta.

1. Impressão da matriz: percorre a matriz inteira e exibe cada elemento que nela consta, além de sua respectiva posição junto.

* Matriz estática: printMatriz().
* Matriz com lista encadeada: printMatriz().
* Nesse caso, os dois métodos são bem parecidos pois precisam necessariamente percorrer todas as posições.

1. Representar uma matriz vazia: torna todos os elementos de uma matriz nulos, percorrendo posição por posição.

* Matriz estática: printMatrizVazia ().
* Matriz com lista encadeada: EmptyMatriz().
* O método de matriz com lista encadeada é mais eficiente pois só precisa fazer um vetor sem elos enquanto o método de matriz estática percorre todas suas posições para colocar valor 0 em cada uma delas.

1. Verificar se é uma matriz vazia: Se todos os elementos de uma matriz são nulos, percorrendo posição por posição.

* Matriz estática: isVazia().
* Matriz com lista encadeada: isVazia().
* Enquanto o método de matrizes estáticas é O(n^2) pois precisa checar posição por posição, o método utilizando listas encadeadas percorre apenas as posições do vetor para checar se seus elos são nulos O(n).

1. Verificar se é uma matriz diagonal: Verifica se somente na diagonal principal há elementos não nulos.

* Matriz estática: isDiagonal().
* Matriz com lista encadeada: isDiagonal().
* Os dois métodos percorrem a matriz até encontrar um elemento não nulo fora da diagonal principal

1. Verificar se é uma matriz linha: Se em apenas uma linha, há elementos não nulos.

* Matriz estática: isLinha().
* Matriz com lista encadeada: isLinha().
* Os dois métodos percorrem a matriz inteira para checar se apenas uma linha tem elementos

1. Verificar se é uma matriz coluna: Se em apenas uma coluna, há elementos não nulos.

* Matriz estática: isColuna().
* Matriz com lista encadeada: isColuna().
* Os dois métodos percorrem a matriz inteira para checar se apenas uma linha tem elementos.

1. Verificar se é uma matriz triangular inferior: Só há elementos da diagonal principal para baixo.

* Matriz estática: isTriangularInferior().
* Matriz com lista encadeada: isTriangularInferior().
* Os dois métodos percorrem da mesma maneira as linhas e colunas, porém o método utilizando listas encadeadas checa apenas se existe alguma coluna numa posição que não satisfaz a propriedade, enquanto o método de matriz estática precisa passar por essas posições e checar se não há elementos nulos nas posições,

1. Verificar se é uma matriz triangular superior: Só há elementos da diagonal principal para cima.

* Matriz estática: isTriangularSuperior().
* Matriz com lista encadeada: isTriangularSuperior().
* Utiliza a mesma lógica da última função mas, dessa vez, verifica as posições que i > j onde A[i,j].

1. Verificar se a matriz é simétrica: A[i,j] = A[j,i] para todo i, j.

* Matriz estática: isSimetrica().
* Matriz com lista encadeada: isSimetrica() + temValorNaPosicao(int linha, int coluna, int valor)
* O método de matriz estática tem complexidade quadrática enquanto o método utilizando listas encadeadas tem complexidade cúbica, pois além de percorrer a matriz, precisa percorrer novamente para encontrar o valor da posição simétrica e compará-los. Nesse caso foi utilizado um método auxiliar para realizar o teste de simetria.

1. Somar duas matrizes esparsas.

* Matriz estática: somarMatrizes(MatrizEsparsaEstatica matriz2).
* Matriz com lista encadeada: somarMatrizes(MatrizEsparsaLinkedList outra).
* O método de matriz estática tem complexidade quadrática enquanto o método utilizando listas encadeadas tem complexidade cúbica, isso se deve ao fato do método de inserir elemento na matriz com listas ser O(n).Além disso, a classe de matriz com lista encadeada deve se atentar às somas em que alguma das matrizes tem um elemento nulo em uma posição.

1. Multiplicar duas matrizes esparsas.

* Matriz estática: multiplicaMatrizes(MatrizEsparsaEstatica matriz2): Complexidade O(n^3)
* Matriz com lista encadeada: multiplicaMatrizes(MatrizEsparsaLinkedList outra): Complexidade O(n^4) pois a multiplicação custa n^3 e para cada multiplicação, são utilizados os métodos de buscar elemento O(n) e inserir elemento O(n).

1. Obter a matriz transposta: Trocar as linhas pelas colunas de uma matriz.

* Matriz estática: obterTransposta().
* Matriz com lista encadeada: obterTransposta().
* O método utilizando lista encadeada paga mais complexidade pois o método de inserir não é em tempo constante como na classe de matriz estática.

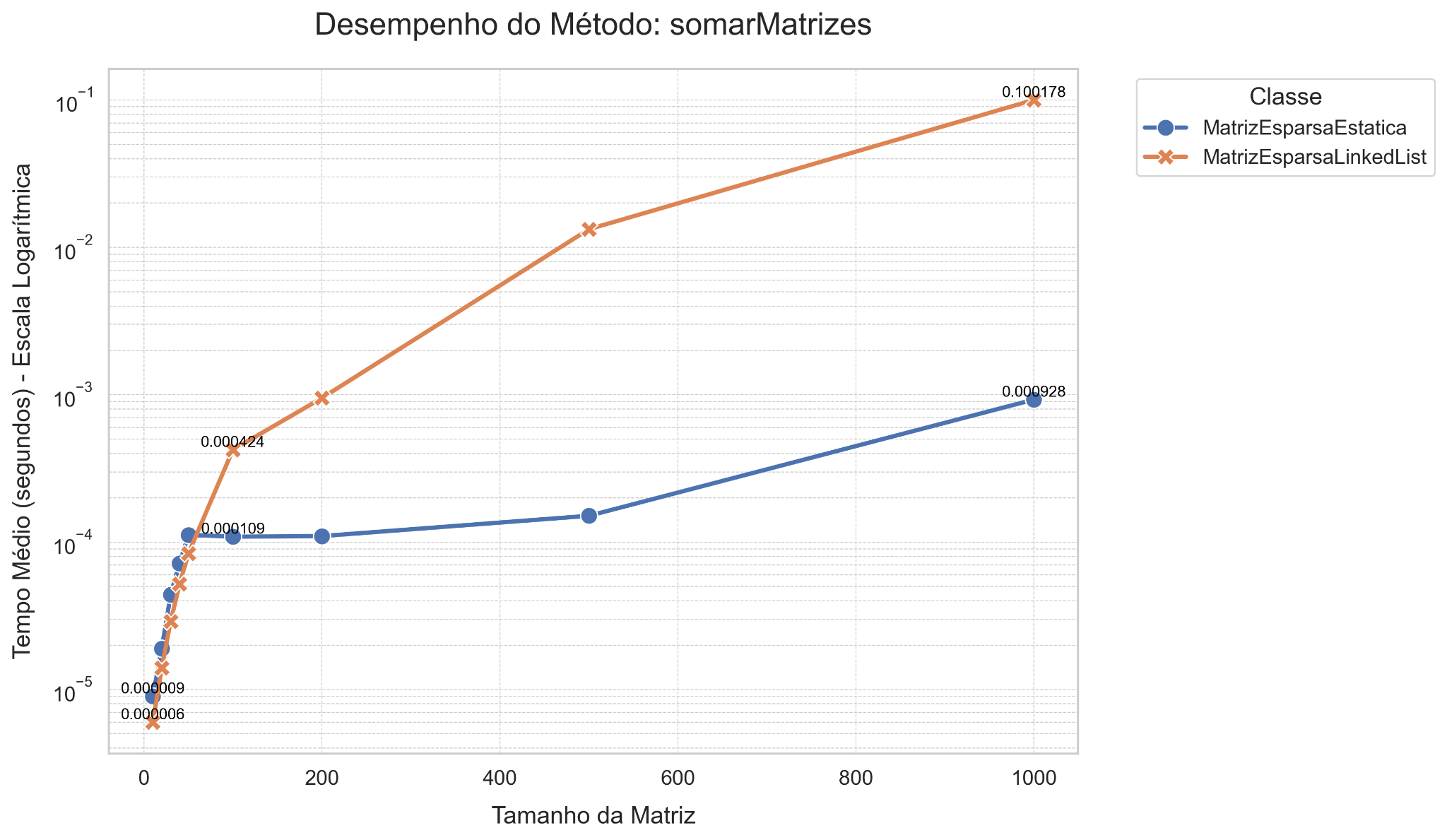
*Testes*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

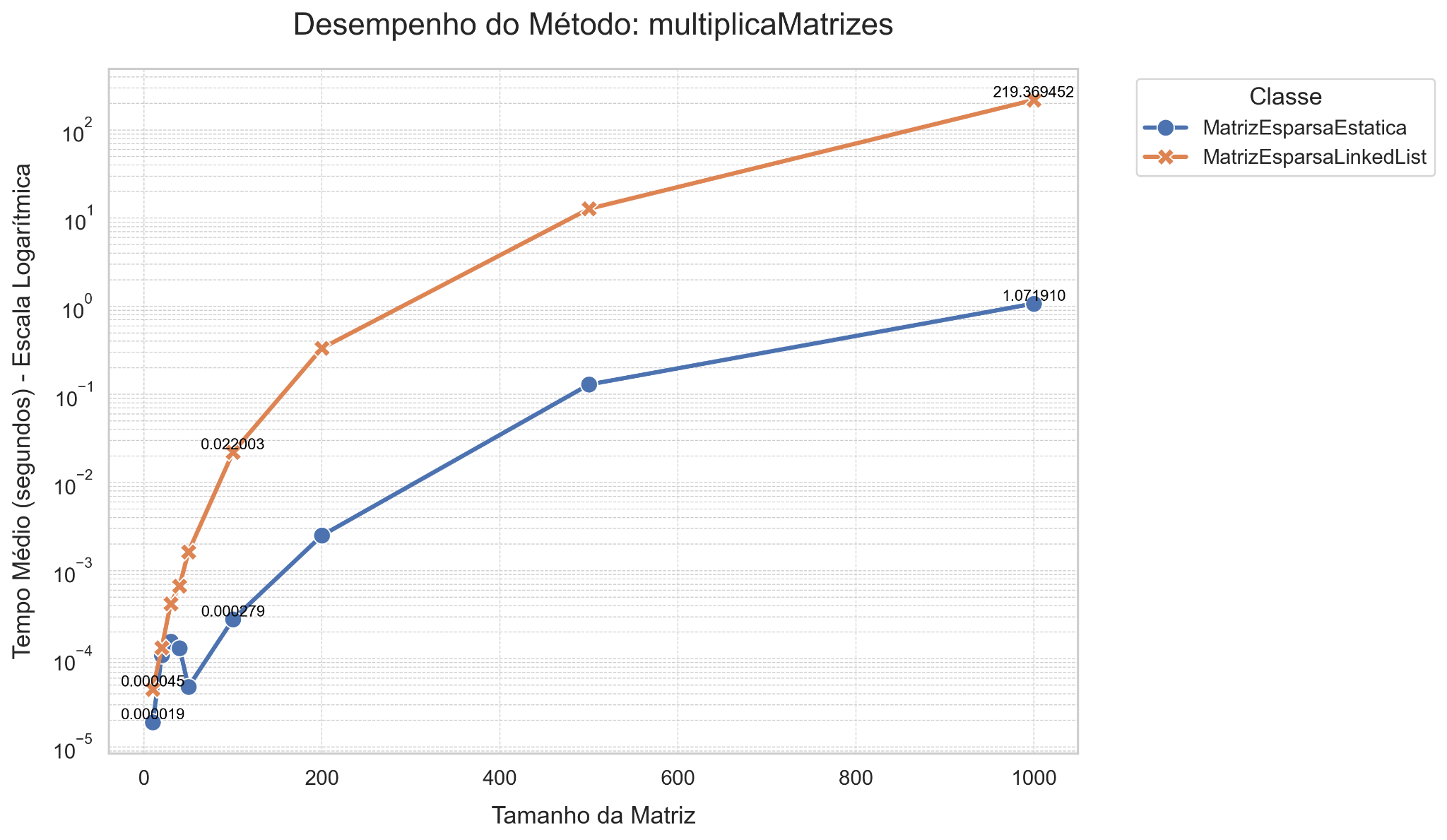
Para a execução dos testes, colocamos os tamanhos em um vetor, e fizemos um for para iterar sobre os tamanhos, e para cada tamanho geramos a matriz correspondente. Dentro do loop de tamanhos iteramos sobre um vetor de métodos que queríamos testar, e dentro desse loop efetivamente executamos os métodos e era feita a anotação do tempo de execução. Dessa forma era executado os 3 métodos da matriz estática e os 3 métodos da matriz com lista encadeada, e cada tempo médio individualmente.  
  
Após a execução, tudo que foi gerado era escrito em um arquivo csv, e por meio de um código auxiliar utilizando python esse arquivo era lido e de acordo com as informações contidas no arquivo os gráficos eram plotados.

Foram 3 métodos escolhidos para serem testados com base nos seus tempos de execução e seus resultados estão representados abaixo:

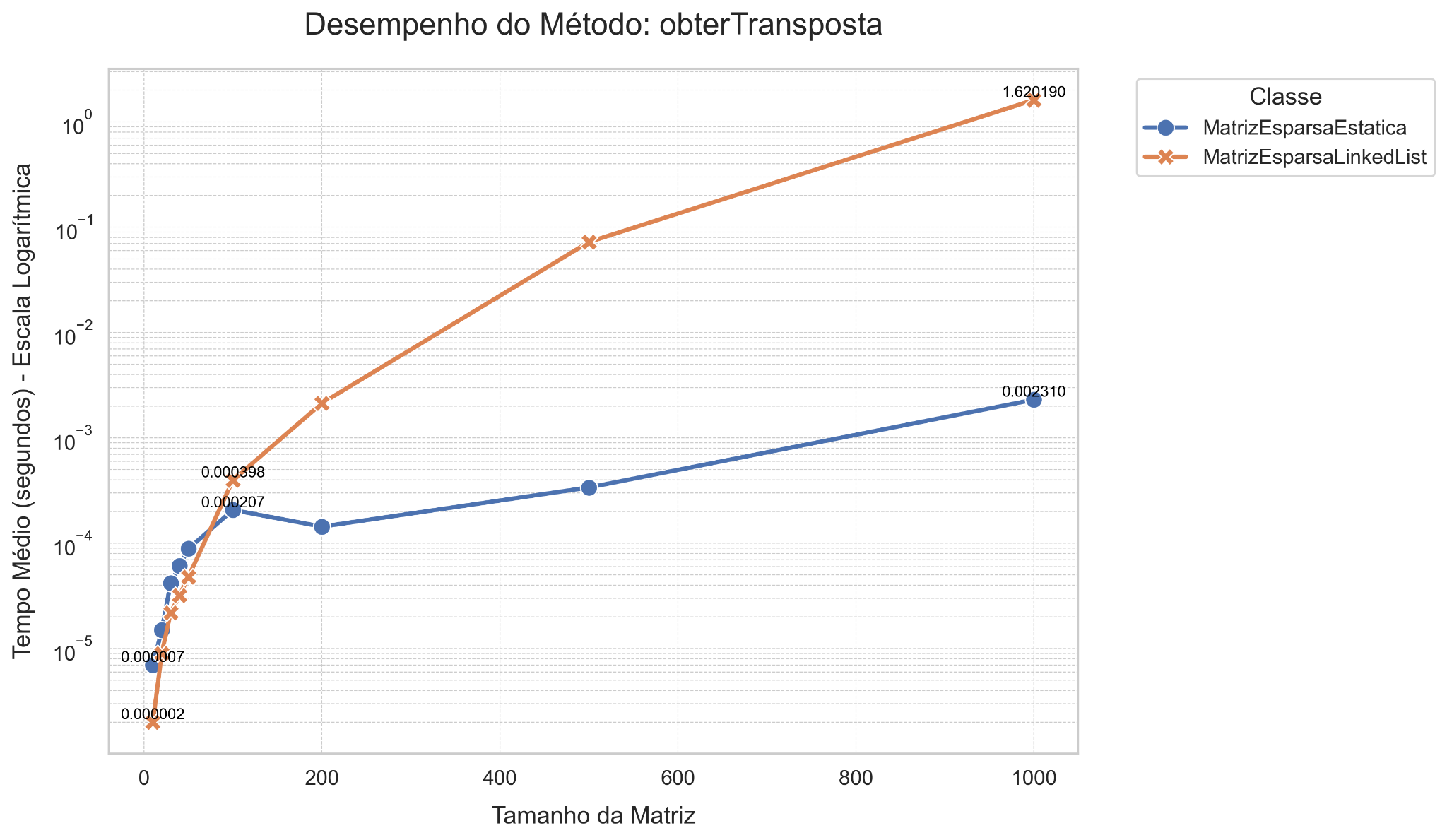
1. Soma de matrizes:



1. Multiplicação de matrizes:



1. Obter matriz transposta:



*Observações:*

Por limitações de poder computacional, foi impossível realizar os testes nos métodos de matrizes com listas encadeadas com matrizes de tamanho 10.000 x 10.000 ou mais, por isso os resultados estão limitados até matrizes de 1.000 x 1.000.

"main" #1 [22972] prio=5 os\_prio=0 cpu=33423484.38ms elapsed=106056.38s tid=0x000001fa0cf145e0 nid=22972 runnable [0x0000003c5e9fe000]

java.lang.Thread.State: RUNNABLE

at MatrizEsparsaLinkedList.searchElement(MatrizEsparsaLinkedList.java:123)

at MatrizEsparsaLinkedList.multiplicaMatrizes(MatrizEsparsaLinkedList.java:330)

at Main.main(Main.java:53)

Aproximadamente 30 horas de execução. No dia seguinte ficou acumulado em 45 horas de execução e estava longe de terminar.  
  
Inicialmente utilizamos uma abordagem de usar arraylist para guardar as posições dos elementos não nulos, de primeira impressão funcionou bem, porém ao chegar no tamanho 20.000 x 20.000 o arraylist tinha um estouro de memória, portanto não foi uma solução eficiente. Após isso chegamos em outra abordagem, que é a atual que utilizamos, que basicamente consiste em dividir a matriz em blocos de 100, dessa forma reduzimos o número de colisões e o tempo de execução.

*Conclusão:*

Para os métodos selecionados, é possível perceber um padrão de comportamento. Com matrizes de tamanho menor as duas implementações parecem bem parelhas em relação ao tempo de execução, com alguns pontos em que a implementação utilizando listas encadeadas consegue ser mais eficiente em relação a implementação tradicional de matriz. Conforme o tamanho das matrizes aumenta, a diferença passa a ser mais notável e a implementação utilizando listas perde muito em relação à implementação tradicional. O grupo acredita que a desvantagem mostrada nos gráficos não está ligada diretamente aos métodos responsáveis por essas operações, mas sim os métodos auxiliares utilizados dentro das operações como os métodos de inserção e busca. A diferença de complexidade desses métodos auxiliares implica diretamente no desempenho dessas operações uma vez que eles são chamados inúmeras vezes em loop dentro das operações.

Um exemplo presente em uma das operações testadas, mas que se repete nas outras duas de maneira semelhante é o método de obter a matriz transposta: Nas duas diferentes implementações é necessário percorrer as posições das matrizes e isso é O(n^2) e também é utilizado o método de inserir para retornar a matriz transposta porém, justamente pela diferença de complexidade do método de inserir (Matriz estática = O(1) | Matriz lista encadeada = O(n)) a complexidade da operação na implementação utilizando lista encadeada aumenta consideravelmente.

Outro fator que o grupo considerou importante para o resultado dos testes foi o fato da classe de matriz estática aproveitar a memória cache enquanto a outra forma de implementação, não. A alocação de memória para vetores é contígua, diferentemente da alocação de uma lista encadeada, dessa forma, a memória cache é muito menos suscetível a miss utilizando a implementação de matriz estática, o que deve poupar muito tempo, principalmente em matrizes de tamanho maior, em relação a implementação com listas encadeadas. Além disso, para matrizes esparsas com grau de esparsidade maior( 90%), a matriz utilizando lista encadeada teria um desempenho melhor, visto que, essa implementação só percorre os elementos não nulos, e com 90% de esparsidade teríamos apenas 10% para percorrer, enquanto na estática teria que percorrer tudo.

Com base na implementação utilizada é plausível considerar esses fatores como motivos para os resultados apresentados nos testes.