UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO)

Sistemas de Informação - Disciplina de Estrutura de dados

Implementação e testes de matrizes esparsas em Java

Trabalho final

Professor: Pedro Nuno

Alunos: Alexandre de Casado Lima Vidal

Camilo Lemos Lima

Rafael Tavares da Silva

1. INTRODUÇÃO

Este relatório apresenta o projeto desenvolvido como parte da disciplina de Estrutura de Dados, tendo como objetivo a implementação e análise de matrizes esparsas. O projeto envolve duas abordagens: a implementação de matrizes esparsas estáticas, armazenadas em vetores, e a implementação de matrizes esparsas dinâmicas, utilizando estruturas encadeadas por meio de Elos.

1.1 Implementação

Para estruturar a implementação, foram desenvolvidas duas classes principais: MatrizEstatica, que representa a matriz armazenada de forma estática em um vetor, e MListaEncadeada, que implementa a matriz de maneira dinâmica utilizando um vetor de listas encadeadas. Ambas as classes possuem os mesmos métodos, permitindo uma comparação direta entre as abordagens em termos de eficiência e desempenho.

Além dessas, foi criada uma terceira classe responsável por gerar números aleatórios e inseri-los em posições aleatórias da matriz, a classe: Produção, garantindo uma distribuição variada dos elementos e tornando os testes mais realistas. Nesta classe, é possível gerar números com determinado grau de esparsidade (quantidade de elementos nulos). Para este experimento, foi adotado um grau de esparsidade de 60%.

1.2 CÓDIGO

1.2.1 Classe Matriz Estática

A classe MatrizEstática funciona como uma matriz bidimensional de inteiros, organizada em linhas e colunas com capacidade fixa. Cada elemento da matriz pode ser acessado, modificado ou removido através de métodos específicos, e a estrutura interna é implementada com um array de duas dimensões. A classe oferece funcionalidades para inserir e remover elementos em posições específicas, além de verificar se a matriz

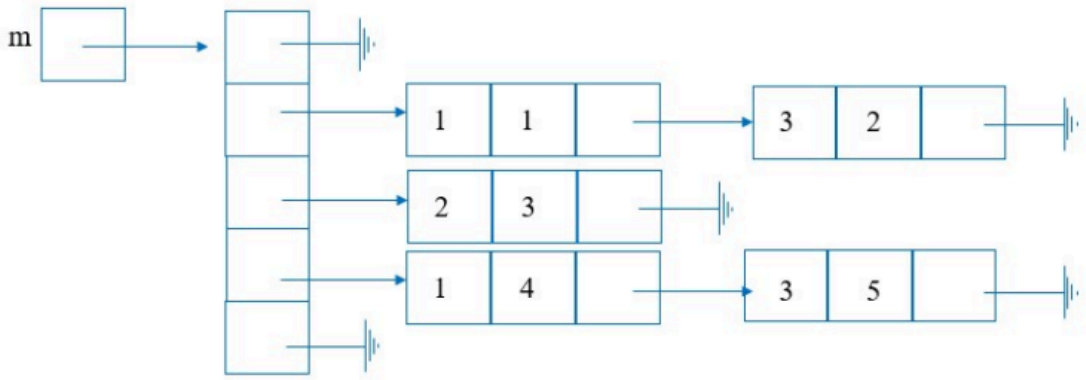
está vazia ou se apresenta determinadas características, como ser diagonal, triangular ou simétrica.

Ela inclui métodos como inserir, que adiciona valores na matriz, e remover, que redefine um elemento para zero. A classe também permite recuperar um elemento através do método recuperarElemento e buscar um valor específico com buscarElem.

Além disso, a classe possui funcionalidades para realizar operações matemáticas com matrizes, como soma (somarMatrizesEsparsas), multiplicação (multiplicarMatrizesEsparsas) e transposição (calcularTransposta). A matriz pode ser verificada quanto a condições como ser uma matriz triangular inferior, triangular superior, simétrica, ou se contém uma linha ou coluna cheia. A matriz pode ser representada visualmente com o método imprimirMatriz, e ainda oferece a capacidade de ser resetada para seu estado vazio com representarMatrizVazia.

1.2.2 Classe Matriz Lista Encadeada

A classe Matriz Lista Encadeada funciona como um vetor de Elos, onde cada posição do vetor representa uma linha da matriz e cada Elo representa uma coluna daquela linha. Nesta estrutura de dados, apenas elementos não nulos são representados. Na imagem abaixo é possível representar como essa matriz funciona conceitualmente:



O método de inserção na matriz foi adotado da seguinte forma: Em caso de inserção de zeros em determinada posição, aquele elemento é removido. Por outro lado, elementos não nulos são inseridos na lista

encadeada correspondente a sua linha (posição no vetor) de forma a manter a lista ordenada pelo atributo *colunaIndex,* tornando mais fácil percorrer a lista.

Já no método de remoção, é recebido uma posição (**linha, coluna**) para remover esse elemento e retorna *true* em caso de sucesso na remoção e *false* em caso de não encontrar o elemento.

1.2.3 Classe Produção

A classe Produção foi implementada de forma que não apresenta construtores nem atributos próprios, sendo desenvolvida exclusivamente para a criação dos métodos geradores de matrizes estáticas e dinâmicas.

Nesse contexto, foram desenvolvidos dois métodos estáticos na classe. O método GerarMatrizEstática , como o nome sugere, é responsável por retornar uma matriz estática com 60% de esparsidade nos elementos. Ou seja, a matriz gerada possui exatamente 60% de sua capacidade total preenchida com o valor 0.

Para que a matriz estática seja gerada, o método necessita de três parâmetros: a quantidade de linhas, a quantidade de colunas e o grau de esparsidade. No contexto deste trabalho, o grau de esparsidade foi definido como 60%.

As assinaturas dos métodos são as seguintes:

```
public static MatrizEstática gerarMatrizEstática(int lin, int col, float grauEsparcidade);

public static MListaEncadeada gerarMatrizListaEncadeada(int lin, int col, float grauEsparcidade);
```

Em ambos os métodos, o funcionamento e a lógica aplicados são semelhantes. Primeiramente, realiza-se um cálculo para determinar a quantidade de elementos que a matriz deve ter. Com esse valor, é

executado um loop com um contador, durante o qual são geradas aleatoriamente posições na matriz para inserir os elementos. Há uma verificação em cada iteração para garantir que a posição selecionada ainda não tenha sido preenchida.

Ao gerar o número aleatório para a inserção na matriz, também é necessário garantir que o número gerado não seja zero, pois, caso contrário, isso prejudicaria o objetivo de manter a esparsidade definida, já que o valor 0 não deve ser inserido nas posições que não são esparsas.

2. MÉTODOS IMPLEMENTADOS

2.1. Busca por um elemento específico

Matriz Estática - public int recuperarElemento(int lin, int col);

O método recebe a posição do elemento na matriz e retorna imediatamente o elemento naquela posição do vetor.

Complexidade: *O(1)*.

Matriz Encadeada - public int getElemento(int l, int c);

O método recebe a posição do elemento na matriz e percorre a partir do primeiro Elo da linha que ele pertence, até encontrar a coluna correspondente.

Complexidade: *O(n)* sendo n a quantidade de elementos (colunas) não nulos naquela linha.

2.2. Impressão da matriz

Matriz Estática - public void imprimirMatriz();

O método percorre por meio de dois *loops* para imprimir.

Complexidade: *O(L x C)*. sendo L a quantidade de linhas e C a quantidade de colunas.

Matriz Encadeada - public void imprime();

O método percorre a matriz por meio de dois *loops*, um itera pelo vetor de linhas e o outro por cada Elo na linha corrente.

Complexidade: *O(L x C)* sendo L a quantidade de linhas e C a quantidade de colunas não nulas.

2.3.Representar uma matriz vazia

Matriz Estática - public void representarMatrizVazia();

O método percorre a matriz à procura de elementos diferentes de 0, e caso ele os encontre, são substituídos por zero, transformando-a em uma matriz vazia.

Complexidade: *o(L x C)* sendo L a quantidade de linhas e C a quantidade de colunas.

Matriz Encadeada - public void representarVazia();

O método elimina todos os elementos, transformando em uma matriz vazia.

Complexidade: *O(L)* sendo L a quantidade de linhas.

2.4. Verificar se é uma matriz vazia

Matriz Estática - public boolean isMatrizVazia();

O método percorre toda a matriz em busca de elementos diferentes de zero. Caso encontre algum, ele retorna false. Se nenhum elemento diferente de zero for encontrado, ou seja, se todos os elementos forem iguais a zero, o método retorna true.

Complexidade: *O(L*C)* sendo L a quantidade de linhas e C a quantidade de colunas.

Matriz Encadeada - public boolean vazia();

O método percorre pelo vetor de linhas para verificar se existe algum elemento inserido.

Complexidade: *O(L)* sendo L a quantidade de linhas.

2.5. Verificar se é uma matriz diagonal

Matriz Estática - public boolean isMatrizDiagonal();

O método verifica se uma matriz é diagonal, ou seja, se todos os elementos fora da diagonal principal são iguais a zero. Ele percorre cada elemento da matriz usando dois loops aninhados: o primeiro itera sobre as linhas (i), e o segundo sobre as colunas (j). Para cada elemento, o método verifica se o índice da linha (i) é diferente do índice da coluna (j) e se o valor do elemento (matriz[i][j]) é diferente de zero. Caso essa condição seja verdadeira, o método retorna false, indicando que a matriz não é diagonal. Se o método percorrer toda a matriz sem encontrar elementos fora da diagonal principal diferentes de zero, ele retorna true, confirmando que a matriz é diagonal.

Complexidade: *O(L*C)* sendo L as linhas e C as colunas.

Matriz Encadeada - public boolean matrizDiagonal();

O método percorre a matriz para verificar se existem elementos fora da diagonal principal e retorna *false* no caso de ter.

Complexidade: *O(n)* sendo n elementos não nulos.

2.6. Verificar se é uma matriz linha

Matriz Estática - public boolean isMatrizLinha();

O método verifica se a matriz contém exatamente uma linha onde todos os elementos são diferentes de zero (uma 'linha cheia'). Ele utiliza uma variável para contar quantas linhas atendem ao critério. O método percorre a matriz com dois loops aninhados: o primeiro itera sobre as linhas (i), e o segundo sobre as colunas (j). Para cada linha, uma variável

booleana é inicializada como true. Se algum elemento da linha for igual a zero, ela é definida como false, e o loop interno é interrompido. Caso a linha seja considerada 'cheia' (ou seja, nenhum elemento é zero), um contador é incrementado. Se mais de uma linha cheia for encontrada, o método retorna false imediatamente, pois a matriz não atende ao critério de ter exatamente uma linha cheia. Ao final, o método retorna true se exatamente uma linha cheia foi encontrada; caso contrário, retorna false.

Complexidade: *O(L * C)* sendo L a quantidade de linhas e C a quantidade de colunas.

Matriz Encadeada - public boolean matrizLinha();

O método percorre o vetor das linhas verificando se apenas uma linhas está preenchida e se essa linha está completa.

Complexidade: *O(L + C)* sendo L a quantidade de linhas e C a quantidade de colunas.

2.7. Verificar se é uma matriz coluna

Matriz Estática - public boolean isMatrizColuna();

O método verifica se a matriz contém exatamente uma coluna onde todos os elementos são diferentes de zero (uma 'coluna cheia'). Ele utiliza uma variável para contar quantas colunas atendem a esse critério. O método percorre a matriz com dois loops aninhados: o primeiro itera sobre as colunas (j), e o segundo sobre as linhas (i). Para cada coluna, uma variável booleana é inicializada como true. Se algum elemento da coluna for igual a zero, ela é definida como false, e o loop interno é interrompido (break). Caso a coluna seja considerada 'cheia' (ou seja, nenhum elemento é zero), o contador é incrementado. Se mais de uma coluna cheia for encontrada, o método retorna false imediatamente, pois a matriz não atende ao critério de ter exatamente uma coluna cheia. Ao final, o método

retorna true se exatamente uma coluna cheia foi encontrada; caso contrário, retorna false.

Complexidade: *O(L*C)* sendo L a quantidade de linhas e C a quantidade de colunas.

Matriz Encadeada - public boolean matrizColuna();

O método percorre pelo vetor linhas para verificar se todas tem exatamente um elemento e se esse elemento em todas as linhas pertence a mesma coluna.

Complexidade: *O(L)* sendo L a quantidade de linhas.

2.8. Verificar se é uma matriz triangular inferior

Matriz Estática - public boolean isMatrizTriangularInferior();

O método percorre a matriz verificando os elementos acima da diagonal principal. Caso pelo menos um desses elementos seja diferente de 0, retorna falso. Caso contrário - se todos os elementos acima da diagonal principal forem 0 - retorna verdadeiro.

Complexidade: *O(n)* sendo n a quantidade de elementos não nulos na matriz.

Matriz Encadeada - public boolean isTriangularInferior();

O método percorre elo por elo e verifica se cada elemento acima da diagonal principal não é nulo. Depois verifica se o elemento é diferente de 0. Caso seja, retorna falso, caso contrário vai para o próximo elo. Caso todos os elementos acima da diagonal principal sejam 0, retorna verdadeiro;

Complexidade: *O(n)* sendo n a quantidade de elementos não nulos na matriz.

2.9. Verificar se é uma matriz triangular superior

Matriz Estática - public boolean isMatrizTriangularSuperior();

O método percorre a matriz verificando os elementos abaixo da diagonal principal. Caso pelo menos um desses elementos seja diferente de 0, retorna falso. Caso contrário - se todos os elementos acima da diagonal principal forem 0 - retorna verdadeiro.

Complexidade: *O(L*C)* sendo L a quantidade de linhas e C a quantidade de colunas

Matriz Encadeada - public boolean isTriangularSuperior();

O método percorre elo por elo e verifica se cada elemento abaixo da diagonal principal não é nulo. Depois verifica se o elemento é diferente de 0. Caso seja, retorna falso, caso contrário vai para o próximo elo. Caso todos os elementos acima da diagonal principal sejam 0, retorna verdadeiro;

Complexidade: *O(L)* sendo L a quantidade de linhas com elementos não nulos.

2.10. Verificar se a matriz é simétrica

Matriz Estática - public boolean isMatrizSimetrica();

O método verifica se a matriz é quadrada, se não for retorna falso. Caso contrário, percorre os elementos acima da diagonal principal da matriz verificando se os itens em relação, por exemplo, A[i][j] e A[j][i] têm o mesmo valor, se não forem, retorna false. Caso todos os elementos forem simétricos retornará true.

Complexidade: *O(L*C)* onde L é a quantidade de linhas e C a quantidade de colunas.

Matriz Encadeada - public boolean isSimetrica();

O método verifica se o número de linhas é o mesmo de colunas (matriz quadrada), caso não seja, retorna falso. Caso contrário, o método percorre a matriz. para cada elemento da matriz ele salva o valor da linha (i) e da coluna (j), além de salvar o valor do dado do elemento. Então chama a função usando como parâmetros o j como linha, o i como coluna e o valor do dado do elemento. A função verifica se naquela posição é o valor indicado. Caso seja, retorna verdadeiro, caso não seja retorna falso.
Complexidade: $O(n^2)$ sendo "n" o número de elementos não nulos na matriz.

2.11. Somar duas matrizes esparsas

Matriz Estática -
public MatrizEstática somarMatrizesEsparsas(MatrizEstática matrizA, MatrizEstática matrizB)

O método verifica se as dimensões são compatíveis para a soma. Depois cria a matriz resultado. Então percorre todos os elementos das matrizes e soma os respectivos elementos. No fim, retorna a matriz resultado.
Complexidade: $O(n \times m)$ sendo n o número de elementos da matriz A e m o número de elementos da matriz B.

Matriz Encadeada -
public MListaEncadeada somarMatrizesEsparsas(MListaEncadeada matrizA, MListaEncadeada matrizB)

O método verifica se as dimensões são compatíveis para a soma. Então cria a matriz resultado. Depois percorre as matrizes somando seus respectivos elementos e inserindo na matriz resultado. No fim, retorna a matriz resultado.

Complexidade: *O(n x m)* sendo n o número de elementos não nulos da matriz A e m o número de elementos não nulos da matriz B.

2.12. Multiplicar duas matrizes esparsas

Matriz Estática -

public MatrizEstática multiplicarMatrizesEsparsas(MatrizEstática matrizA, MatrizEstática matrizB);

O método verifica se as dimensões são compatíveis para multiplicação. Depois cria a matriz que será o produto da multiplicação. Então percorre os elementos da matriz A (ignorando os elementos nulos), depois percorre os elementos das colunas da matriz B (ignorando os elementos nulos), depois multiplica o elemento atual das duas matrizes e coloca na matriz resultado. No fim, retorna a matriz resultado.

Complexidade: *O(n x m)* sendo n o número de elementos não nulos da matriz A e m o número de elementos não nulos da matriz B.

Matriz Encadeada -

public MListaEncadeada multiplicarMatrizesEsparsas(MListaEncadeada matrizA, MListaEncadeada matrizB);

O método verifica se multiplicação é válida (com base nos tamanhos das matrizes). Depois cria uma nova matriz que será a matriz resultado. Então percorre os elementos das linhas da matriz A, e pega os valores desses elementos (ignorando os elementos nulos), depois faz o mesmo com os elementos da coluna da matriz B. Então multiplica os respectivos elementos. No fim, retorna a matriz resultado.

Complexidade: *O(n x m)* sendo n o número de elementos não nulos da matriz A e m o número de elementos não nulos da matriz B

2.13. Obter a matriz transposta

Matriz Estática - public MatrizEstática calcularTransposta(MatrizEstática matriz)

O método cria uma nova matriz, depois percorre ela e em cada elemento percorrido, insere um elemento na posição transposta, ou seja, se o elemento está na posição [i][j] na matriz original, na transposta ele estará na posição [j][i]. No fim o método retorna a matriz transposta gerada.

Complexidade: *O(L*C)* sendo L o número de linhas e C o número de colunas da matriz.

Matriz Encadeada -

public MListaEncadeada calcularTransposta(MListaEncadeada matriz)

O método cria uma nova matriz, depois percorre elo por elo da matriz original. Depois inverte a posição do i (linha) e do j (coluna), ou seja, o que é linha vira coluna e o que é coluna vira linha, e no fim retorna a matriz transposta gerada.

Complexidade: $O(n^2)$ sendo n o número de elementos da matriz

3. EXPERIMENTOS REALIZADOS

3.1 MÉTODO ADOTADO

Com a finalidade de comparar o desempenho das duas estruturas de dados, foram realizados três experimentos com diferentes métodos, medindo seu tempo médio de execução de cada estrutura para que seja possível analisar o desempenho. Para isso, foram escolhidos os métodos de soma de matriz, multiplicação de matriz, cálculo da transposta,verificar se é simétrica e, por fim, o método de representação vazia. Cada método foi realizado 10 vezes em cada estrutura, tomando-se o tempo médio de execução para evitar ruídos na medição.

3.2 AMBIENTE COMPUTACIONAL

O testes foram realizados no computador pessoal de um dos estudantes do grupo, com as seguintes configurações:

Processador	Intel(R) Core(TM) i5-3570K CPU @ 3.40GHz 3.40 GHz
RAM instalada	8,00 GB
Tipo de sistema	Sistema operacional de 64 bits, processador baseado em x64
Sistema instalado	Windows 10 PRO versão 22H2
Placa de vídeo	NVIDIA GEFORCE GTX 1050 TI

O código foi escrito e os testes foram realizados a partir da IDE (ambiente de desenvolvimento integrado) IntelliJ IDEA, com as seguintes configurações:

IntelliJ IDEA 2023.3.6 (Community Edition)
Build #IC-233.15026.9, built on March 21, 2024

Runtime version: 17.0.10+1-b1087.23 amd64
VM: OpenJDK 64-Bit Server VM by JetBrains s.r.o.
Windows 10.0
GC: G1 Young Generation, G1 Old Generation
Memory: 2048M
Cores: 4
Registry: ide.experimental.ui=true
Kotlin: 233.15026.9-IJ

A linguagem de programação adotada foi o Java na versão:

java version "1.8.0_401"
Java(TM) SE Runtime Environment (build 1.8.0_401-b10)
Java HotSpot(TM) 64-Bit Server VM (build 25.401-b10, mixed mode)

3.2.1 LIMITAÇÕES

Matriz de tamanho 10.000 x 10.000

Inicialmente, o experimento tinha como objetivo testar o desempenho das estruturas de dados para os seguintes tamanhos de matriz como entrada: 10 x 10, 20 x 20, 30 x 30, 40 x 40, 50 x 50, 100 x 100, 200 x 200, 500 x 500, 1000 x 1000, 10.000 x 10.000, 20.000 x 20.000, 50.000 x 50.000 e 100.000 x 100.000. No entanto, a partir do tamanho 10.000 x 10.000, o tempo de execução na Matriz Lista Encadeada **ultrapassa 24 horas** sem concluir.

Foram feitas três tentativas de completar o experimento, todas sem sucesso devido ao tempo excessivo de execução no método de **multiplicação de matrizes**. Na primeira tentativa, o tempo de execução já passava de 20 horas quando, por problemas na máquina utilizada, a mesma desligou sozinha. Posteriormente, o experimento foi novamente testado e chegou-se a 24 horas de execução sem sucesso. Assim, foi decidido encerrar o teste para dar prosseguimento no restante do trabalho.

Por último, foi feita uma mudança na *Main* do projeto para que fosse impresso no terminal em qual execução o algoritmo está (das 10 execuções que ocorrem por experimento) e então o código foi rodado por 11 horas para verificar quantas execuções são feitas nesse intervalo de tempo. Como resultado, foi constatado que em 11 horas e 10 minutos o experimento não foi capaz de executar a operação de multiplicação de matriz nenhuma vez.

Portanto, este experimento se limita a entradas de, no máximo, tamanho 1000 x 1000, o que ainda permite uma avaliação detalhada do desempenho de cada estrutura.

4. RESULTADOS

O tempo contabilizado em cada experimento foi colocado em uma planilha, o que permitiu a geração de gráficos para comparar os resultados. Os gráficos e as planilhas podem ser encontrados na outra pasta deste projeto. Além disso, também estão disponíveis PDFs que contém o console resultado de cada um dos experimentos.

Cada gráfico foi gerado de dois modos diferentes: Um incluindo todos os tamanhos testados e o segundo até o tamanho 200 x 200, para facilitar a visualização de valores muito pequenos.

4.1 MÉTODO SOMAR MATRIZES

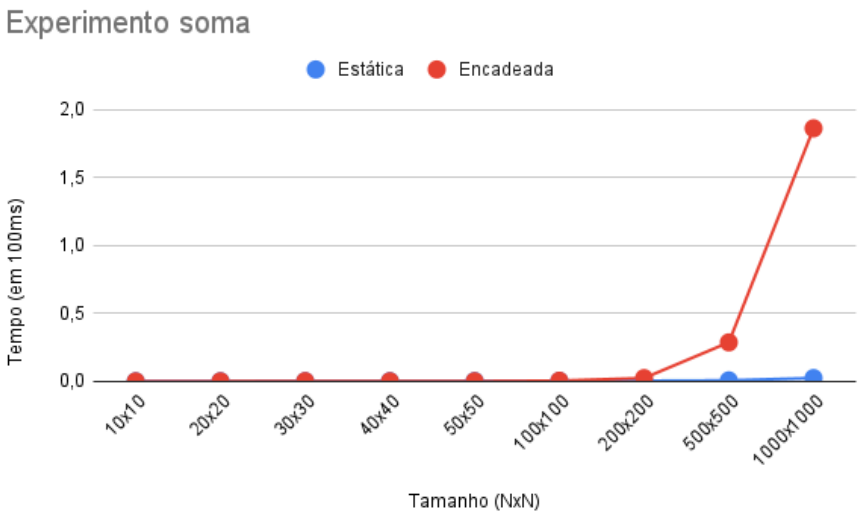


Gráfico 1

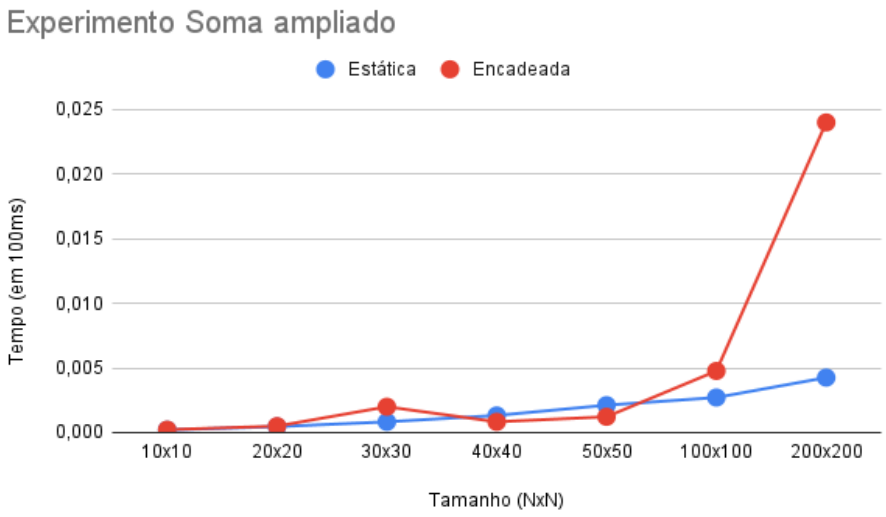


Gráfico 2

A partir dos gráficos, é possível observar que o desempenho das matrizes para entradas de até 100 x 100 é bem parecido. No entanto, para entradas maiores, a Matriz Encadeada tem um aumento considerável, enquanto a Estática cresce muito pouco seu tempo de execução.

4.2 MÉTODO MULTIPLICAR MATRIZES

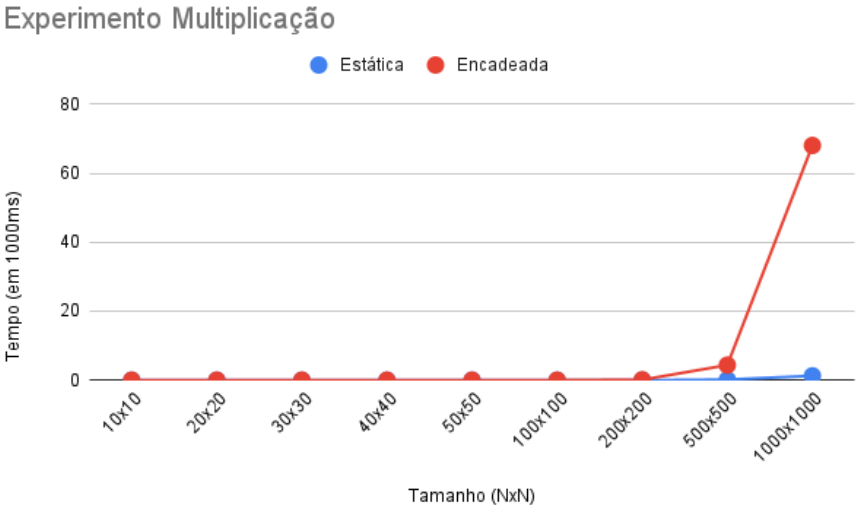


Gráfico 1

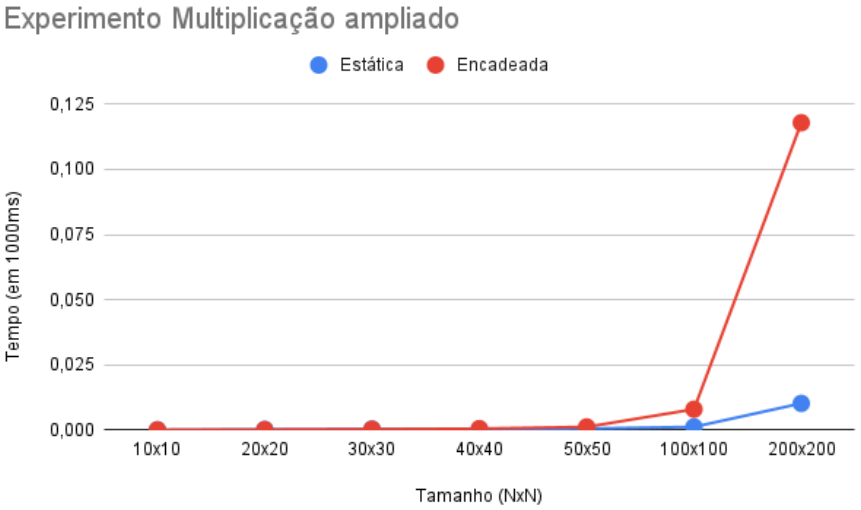


Gráfico 2

A partir dos gráficos, é possível observar que o desempenho das matrizes se repete conforme o experimento anterior, com um aumento muito maior na Matriz Lista Encadeada.

4.2.1 MÉTODO CALCULAR TRANSPOSTA

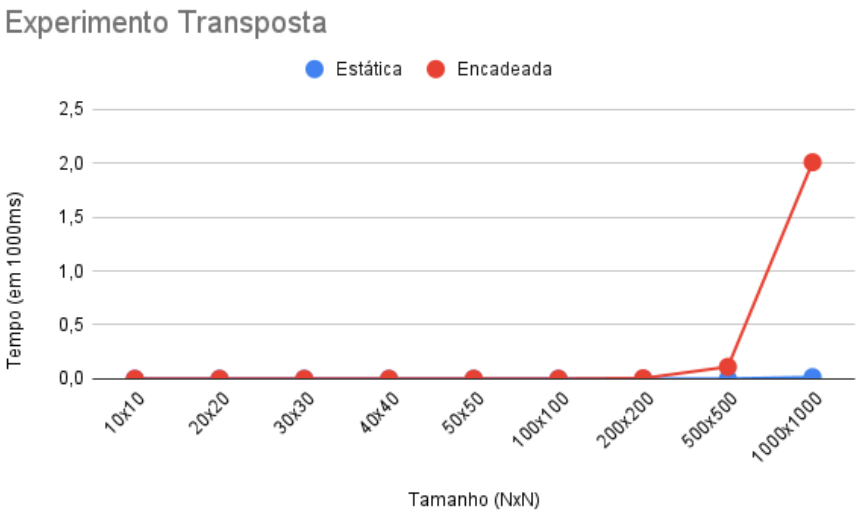


Gráfico 1

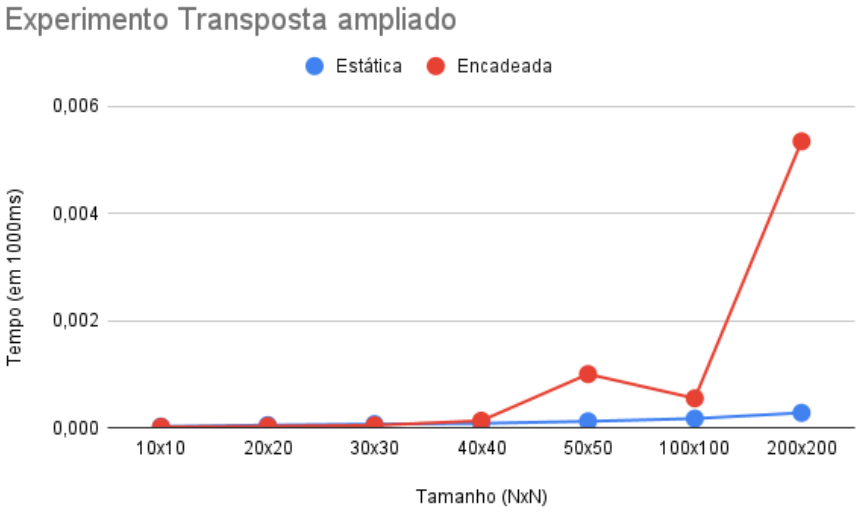


Gráfico 2

Novamente, é perceptível um aumento muito maior no tempo gasto pela Matriz Lista Encadeada do que na Matriz Estática.

4.2.2 MÉTODO VERIFICAR SE É SIMÉTRICA

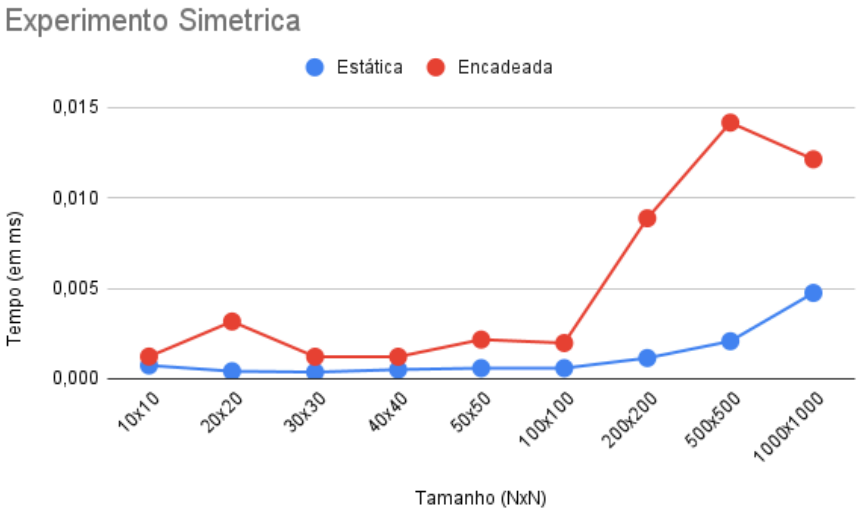


Gráfico 1

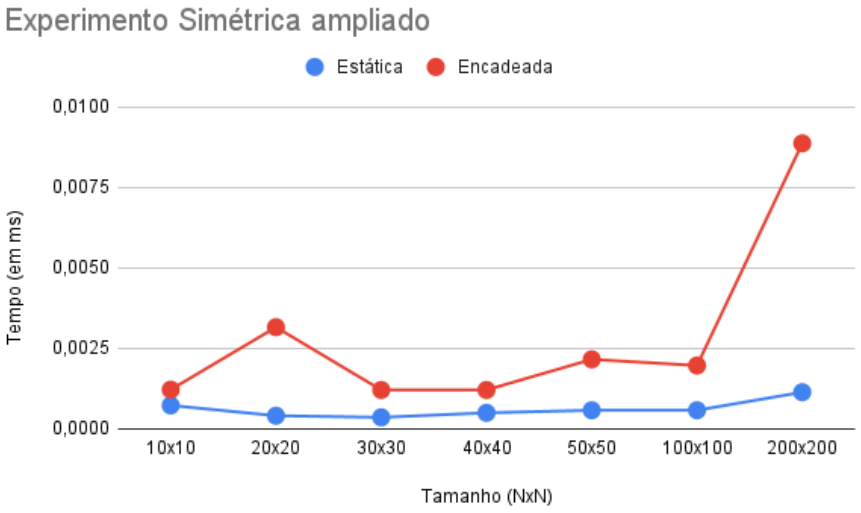


Gráfico 2

Neste caso, é possível observar uma oscilação maior para cada aumento de entrada mas, ainda assim, o comportamento de ambas se repete.

4.2.3 MÉTODO REPRESENTAÇÃO VAZIA

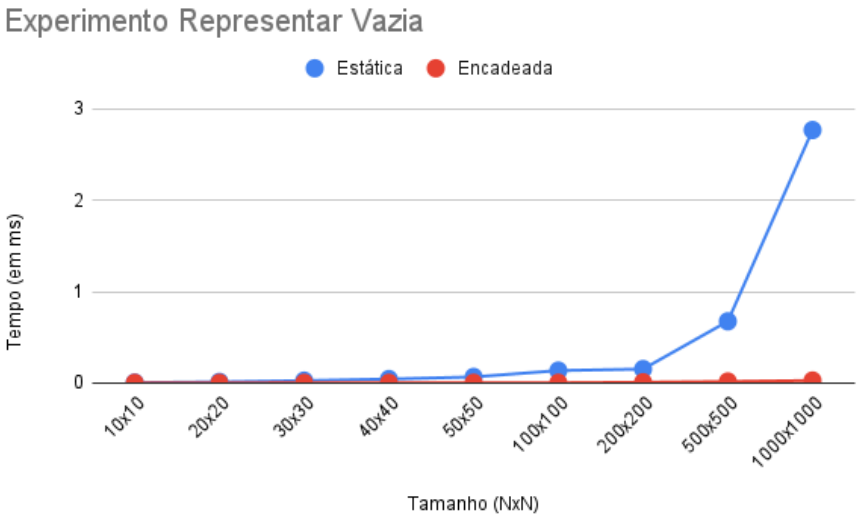


Gráfico 1

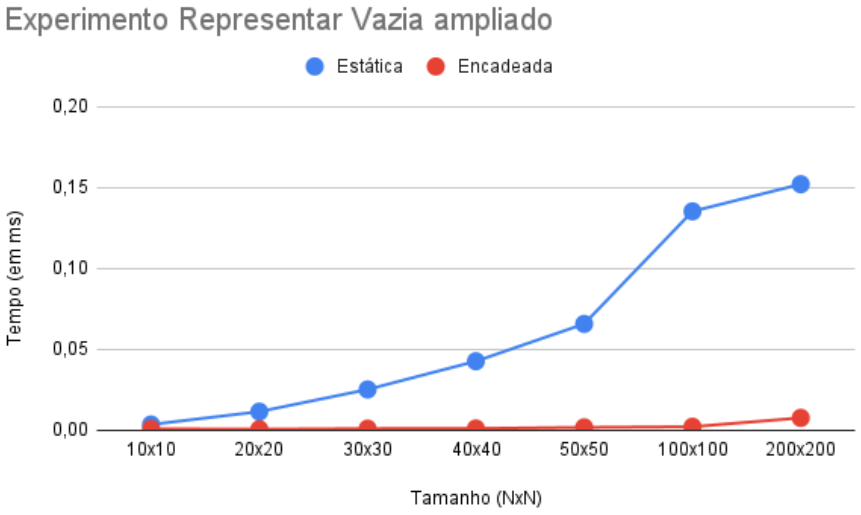


Gráfico 2

Nesse experimento, é notória a vantagem que a matriz lista encadeada leva em relação a Matriz Estática. Isso se deve,

principalmente, pelas suas complexidades que são, respectivamente, *O(L x C)* e *O(L)*.

5. CONCLUSÃO

No início do experimento, esperava-se que, por conta de sua complexidade menor, a MLE (Matriz Lista Encadeada) fosse mais rápida do que a ME (Matriz Estática) para matrizes esparsas pois, com muitos elementos nulos na matriz, ela consegue executar o método com menos operações, ou seja, é esperado um resultado mais rápido para a abordagem encadeada. No entanto, ao observar os resultados de cada um dos experimentos, nota-se que o comportamento não se repete. Para entradas de até 200 de tamanho, ambas obtêm resultados semelhantes mas, ainda assim, a MLE leva mais tempo na maioria dos casos. Observa-se também que para entradas acima de 200 de tamanho, a MLE tem uma queda de desempenho considerável, podendo até triplicar o tempo de execução, como no método de somar matrizes. Já a Matriz Estática, apesar de perder desempenho para entradas maiores, o que é esperado, mantém-se regular ainda com entradas grandes, como 1000 x 1000.

Após analisar os resultados e questionar o porquê do comportamento esperado não se cumprir, mais um experimento foi realizado. Desta vez, o intuito era verificar se para um grau de esparsidade maior que 60% (utilizado anteriormente), a MLE finalmente conseguiria melhores resultados. Então, o experimento de multiplicar matrizes foi novamente realizado, porém, com grau de esparsidade de 90%. Os resultados obtidos foram os seguintes:

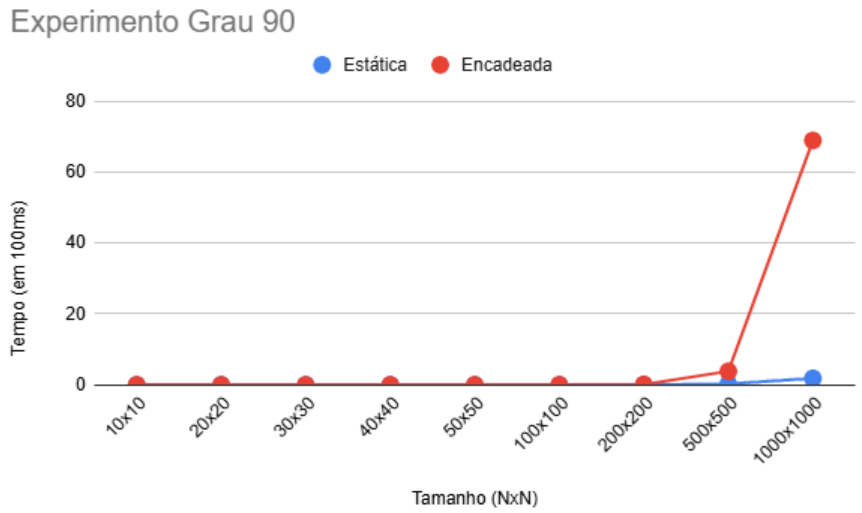


Gráfico 1

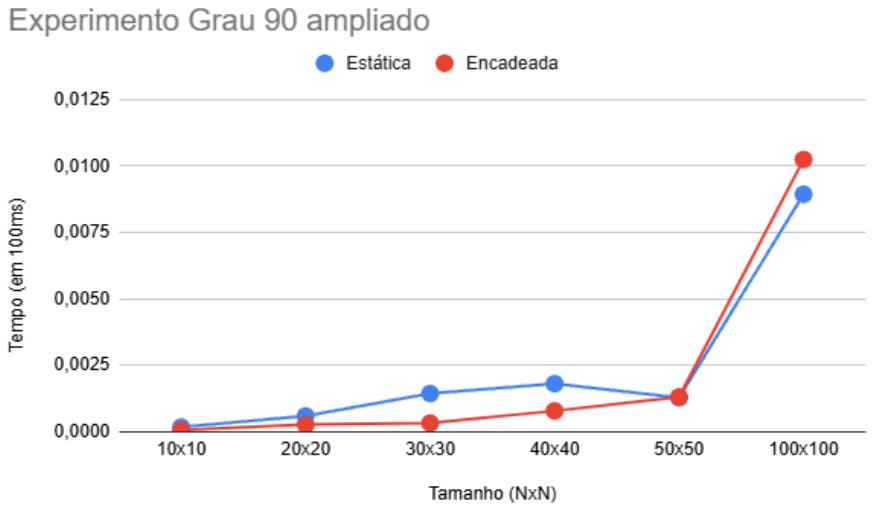


Gráfico 2

Novamente, o comportamento esperado não se repete no experimental. Na imagem ampliada, é possível observar que com entradas pequenas, a Matriz Encadeada acaba por obter resultados levemente melhores. Depois, para tamanhos médios, ambos possuem resultados bem parecidos (50x50 e 100x100, 200x200 e 500x500). E, por fim, para a maior entrada do teste, a MLE tem um aumento exponencial, enquanto a Matriz Estática se mantém num aumento constante. Provando assim que, ainda com um grau de esparsidade muito maior, o desempenho da MLE para grandes entradas é insatisfatório.

Portanto, pode-se concluir que para este experimento, o que importa para ditar o desempenho das estruturas de dados vistas não é necessariamente sua

complexidade para cada método. Para explicar o imprevisto comportamento da Matriz Lista Encadeada, uma hipótese seria que o grau de esparsidade teria que ser ainda maior, com mais de 90% de elementos nulos. Talvez, assim, ela obtivesse um resultado mais condizente. Além disso, os resultados díspares com o esperado podem ser explicados por um grande consumo de memória que a MLE consume, uma vez que cada Elo guarda uma tupla de valores inteiros, além de possuir um vetor inteiro só para guardar as referências das linhas. Desta forma, com entradas maiores é preciso alocar muita memória para resolver suas operações, o que faz com que o tempo para executar aumente consideravelmente para entradas muito grandes, o que é observado nos gráficos.

6. REFERÊNCIAS

- Conversor online:
https://calculator.name/scientific-notation-to-decimal/7.1e-4

- Como converter de nanossegundos para milissegundos:
https://convertlive.com/pt/u/converter/nanossegundos/em/milissegundos#1

- Classe MatrizListaEncadeada inspirada na classe Lista encadeda na disciplina Estruturas de Dados.

- Minuto Java #3 - Tempo de execução de um algoritmo:
Minuto Java #3 - Tempo de execução de um algoritmo