

# Relatório Projeto 1 - Matriz Esparsa

Dupla: Luís Gustavo Rabelo de Oliveira - 540974 e  
Kauan Oliveira Perdigão Lopes - 514867  
Professor: Atílio Gomes Luiz

Semestre 2023.1

## 1 Introdução do TAD Matriz Esparsa:

### 1.1 Descrição inicial

A estrutura de dados implementada é uma Matriz Esparsa, representada pela classe `SparseMatrix`. É uma estrutura adequada para lidar com uma grande quantidade de elementos nulos, ou seja, elementos iguais a zero, mas ainda faz-se necessário acessar os elementos não nulos.

### 1.2 Sobre a implementação

A implementação utiliza uma lista encadeada de nós onde cada elemento da matriz é um nó criado de acordo com a struct `Node`. Cada nó possui informações a cerca da posição de cada elemento na Matriz Esparsa (linha e coluna) e o valor do elemento. Além disso, são utilizados nós sentinelas para indicar o início de cada linha e coluna da Matriz.

### 1.3 Motivação

Essa representação permite economizar espaço de armazenamento, pois apenas os elementos não nulos são explicitamente representados nessa estrutura de dados. Além disso, essa estrutura permite o acesso aos elementos da Matriz, uma vez que é possível percorrer apenas os elementos não nulos em uma linha e/ou coluna especificado.

### 1.4 Aplicações da Matriz Esparsa

A Matriz Esparsa tem diversas aplicações em áreas como matemática, física, engenharia, computação, etc. Alguns exemplos de aplicação incluem: Processamento de imagens, modelagem de redes e etc.

## 2 Detalhes da especificação da implementação

Deu-se o início com a criação dos nós sentinelas para cada linha e coluna da Matriz Esparsa partindo do nó cabeça (`m_head`). Após essa explicação, as funções-membro foram implementadas utilizando os nós sentinelas que são os marcadores do endereço de memória. Cada elemento está conectado ao próximo elemento à direita e abaixo dele. O último elemento de cada linha tem seu ponteiro **direita** apontando para o sentinela da mesma linha. O último elemento de cada coluna tem seu ponteiro **abaixo** apontando para o sentinela da mesma coluna. Com essa abordagem, a matriz torna-se circularmente encadeada.

Durante a implementação, notou-se a necessidade de implementar duas novas funções auxiliares do tipo inteiras:

- `int getLinhas()`
- `int getColunas()`

### 3 Divisão do trabalho:

Inicialmente, a dupla se organizou para discutir as ideias de como começar a implementação partindo do arquivo de descrição do projeto. Os integrantes se reuniram na faculdade para começar a programar juntos, e dessa forma, a maioria do código foi implementado. Quando não foi possível ocorrer os encontros presenciais, a dupla se reuniu via Discord onde compartilharam as telas e programaram as funções juntos. Sobre o relatório, a dupla se reuniu via Discord com compartilhamento de telas para escrever cada tópico solicitado na descrição do projeto.

Para o desenvolvimento do projeto e implementação, foi utilizado o Visual Studio Code, pois suas ferramentas e seu ambiente dão um excelente suporte ao desenvolvedor.

### 4 Dificuldades encontradas:

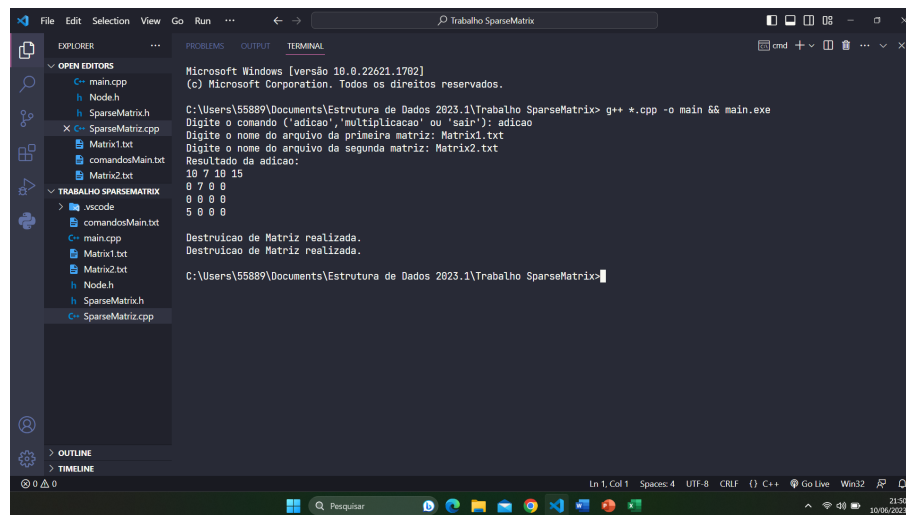
Dentre as dificuldades encontradas pela dupla, pode-se destacar entre as principais:

- A implementação da função **insert**, no sentido da atualização de valores dentro das posições especificadas da Matriz Esparsa, e ainda a criação de um novo nó afim de armazenar um novo valor para cada posição que não foi especificada.
- A implementação da função **get**, onde pelo primeiro raciocínio seguido pela dupla, que foi o de criar uma lista auxiliar para armazenar os valores originais da Matriz, a função não estava retornando o valor de cada linha e coluna desejado. Depois de consultar os endereços de memória e atestar que a função não estava correta, a dupla resolveu trabalhar apenas com a função original e lançar uma "std::out\_of\_range" caso a posição fosse inválida.

### 5 Testes executados:

A Estrutura de Dados implementada contém os seguintes comandos:

- Adição: Faz a soma das matrizes



```
Microsoft Windows [versão 10.0.22621.1702]
(c) Microsoft Corporation. Todos os direitos reservados.

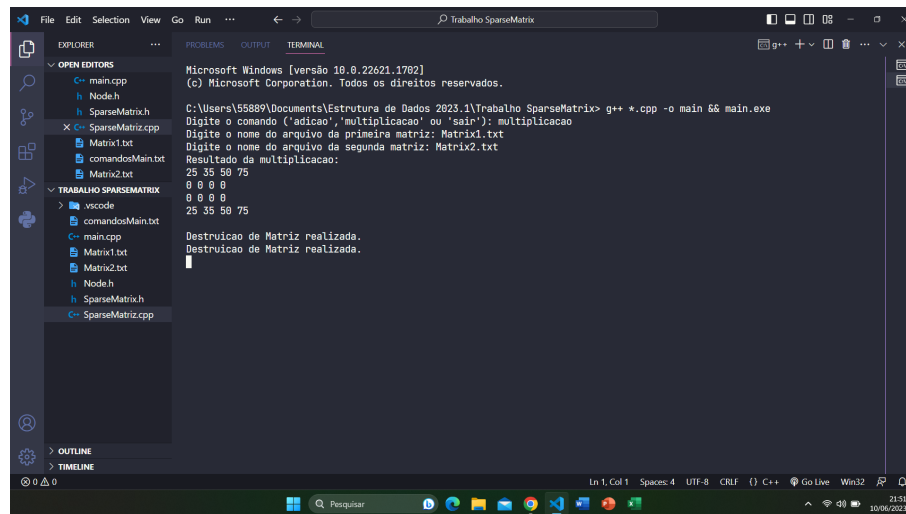
C:\Users\55889\Documents\Estrutura de Dados 2023.1\Trabalho SparseMatrix> g++ *.cpp -o main && main.exe
Digite o comando ('adicao', 'multiplicacao' ou 'sair'): adicao
Digite o nome do arquivo da primeira matriz: Matrix1.txt
Digite o nome do arquivo da segunda matriz: Matrix2.txt
Resultado da adicao:
10 7 10 15
0 7 0 0
0 0 0 0
5 0 0 0

Destruido de Matriz realizada.
Destruido de Matriz realizada.

C:\Users\55889\Documents\Estrutura de Dados 2023.1\Trabalho SparseMatrix>
```

Figura 1: Demonstração da soma dos arquivos **Matrix1.txt** e **Matrix2.txt**

- Multiplicação: Faz a multiplicação das matrizes



The screenshot shows the VS Code interface with the 'TERMINAL' tab active. The terminal output is as follows:

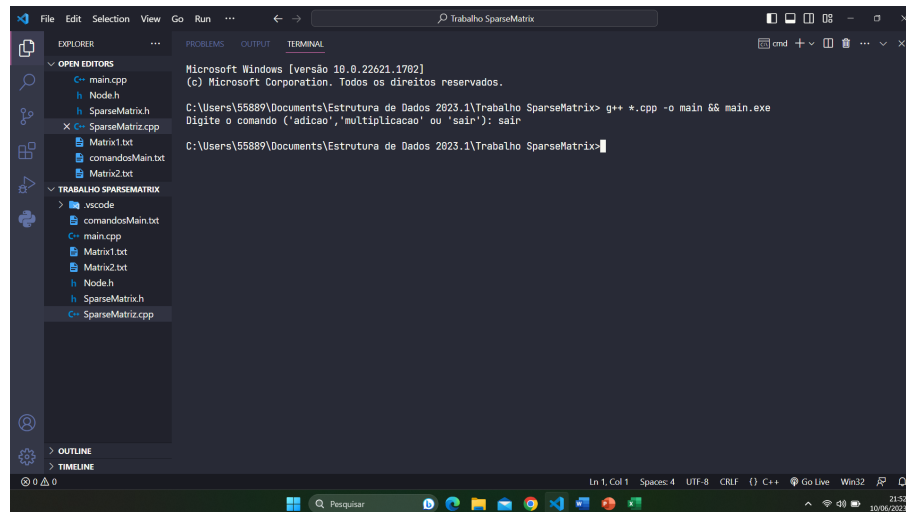
```
Microsoft Windows [versão 10.0.22621.1702]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\55889\Documents\Estrutura de Dados 2023.1\Trabalho SparseMatrix> g++ *.cpp -o main && main.exe
Digite o comando ('adicao','multiplicacao' ou 'sair'): multiplicacao
Digite o nome do arquivo da primeira matriz: Matrix1.txt
Digite o nome do arquivo da segunda matriz: Matrix2.txt
Resultado da multiplicacao:
25 35 50 75
0 0 0 0
0 0 0 0
25 35 50 75

Destruicao de Matriz realizada.
Destruicao de Matriz realizada.
```

Figura 2: Demonstração da multiplicação dos arquivos **Matrix1.txt** e **Matrix2.txt**

- Sair: Sai do programa



The screenshot shows the VS Code interface with the 'TERMINAL' tab active. The terminal output is as follows:

```
Microsoft Windows [versão 10.0.22621.1702]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\55889\Documents\Estrutura de Dados 2023.1\Trabalho SparseMatrix> g++ *.cpp -o main && main.exe
Digite o comando ('adicao','multiplicacao' ou 'sair'): sair
C:\Users\55889\Documents\Estrutura de Dados 2023.1\Trabalho SparseMatrix>
```

Figura 3: Demonstração do comando de saída do programa

A Estrutura de Dados foi aprovada nos testes dos comandos acima, onde os arquivos **Matrix1.txt** e **Matrix2.txt**, que estão dentro da pasta do projeto, contêm os valores das matrizes de teste.

## 6 Análise Assintótica

Nesta sessão será mostrado a análise da complexidade de pior caso das funções **insert**, **get** e **sum**.

- Complexidade da função **insert**:

Busca do nó correspondente à linha: O código percorre uma linha até encontrar o nó correspondente à linha desejada. O pior caso ocorre quando a linha desejada é a última, fazendo o loop iterar  $n$  vezes. Portanto, a complexidade dessa parte é  $O(n)$ .

Busca do nó correspondente à coluna: O código percorre uma coluna até encontrar o nó correspondente à coluna desejada. O pior caso ocorre quando a coluna desejada é a última, fazendo o loop iterar  $n$  vezes. Portanto, a complexidade dessa parte é  $O(x)$ .

Inserção do valor na matriz: Nessa parte, existem duas situações:

Primeira: Se o valor for diferente de zero: O código faz uma busca para verificar se já existe um nó na posição  $(i, j)$ . Se existir, só o valor desse nó é alterado. Se não, é criado um novo nó e inserido na posição correta. O loop percorre a Matriz até encontrar a posição correta, e o pior caso ocorre quando a posição correta é a última, fazendo o loop iterar  $x$  vezes. Portanto, a complexidade dessa parte é  $O(x)$ .

Segunda: Se o valor for zero: O código faz uma busca para verificar se já existe um nó na posição  $(i, j)$ . Se existir, o nó é removido da linha e da coluna. O loop percorre a Matriz até encontrar o nó a ser removido, e o pior caso ocorre quando o nó a ser removido está na última posição da lista, fazendo o loop iterar  $x$  vezes. Portanto, a complexidade dessa parte é  $O(x)$ .

Portanto, a complexidade de pior caso da função **insert** é  $O(n + x)$ , onde  $n$  é o número de linhas e  $x$  é o número de colunas na Matriz Esparsa.

- Complexidade da função **get**:

O loop itera até a linha desejada: O loop for itera de 1 até  $i-1$ , o que necessita de  $i-1$  iterações no pior caso. Portanto, a complexidade dessa parte é  $O(i)$ .

O loop itera até a coluna desejada: O loop while itera até que o aux seja igual a  $\text{noLinhas}$ , enquanto forem diferentes o loop vai percorrer. No pior caso, esse loop vai ter que percorrer todos os elementos da linha. Portanto, a complexidade dessa parte é  $O(n)$ , onde ' $n$ ' é o número de elementos da linha.

Se o número de elementos da linha for proporcional ao número de elementos na matriz, podemos considerar ' $n$ ' como uma constante ' $k$ '. Portanto, a complexidade do pior caso dessa função é  $O(i + k)$ , sendo ' $i$ ' o número da linha e ' $k$ ' uma constante.

Mas, se o número de elementos na linha não for uma constante, mas que seja proporcional ao número de elementos na matriz, a complexidade seria  $O(i + n)$ , sendo ' $n$ ' o número total de elementos na matriz.

- Complexidade da função **sum**:

For interno e externo: Os dois loops 'for' percorrem as linhas e colunas das matrizes A e B. Cada iteração desses loops faz uma soma e uma insere valores na matriz resultante. Como os loop's percorrem todas as posições das matrizes A e B, no pior caso, a complexidade desses loop's é  $O(n^2)$ .

## 7 Referências utilizadas pela dupla como suporte ao desenvolvimento do projeto:

### Referências

Harvey M Deitel and Paul J Deitel. *C++. Fondamenti di programmazione*. Maggioli Editore, 2014.

Deitel and Deitel [2014]

<https://cplusplus.com/reference/fstream/fstream/>

<https://www.geeksforgeeks.org/stringstream-c-applications/>

[https://www.youtube.com/watch?v=C\\_ePgrEbLs0](https://www.youtube.com/watch?v=C_ePgrEbLs0)

<https://www.youtube.com/watch?v=S8rZj0Vbtdk>

<https://www.youtube.com/watch?v=hi74K3ibq-w>

<https://youtube.com/playlist?list=PLjbaYg8NJsoNZY87IDxKzZ6x2XeXtICim>

[https://youtube.com/playlist?list=PLa\\_2246N48\\_p9ndUH10255uvKtSR8mshE](https://youtube.com/playlist?list=PLa_2246N48_p9ndUH10255uvKtSR8mshE)

<https://homepages.dcc.ufmg.br/~nivio/cursos/aed2/roteiro/>

<https://www.youtube.com/watch?v=EA0H2snfCXs>

<http://www.decom.ufop.br/menotti/aedI082/tps/tp2.pdf>