



**UNIVERSIDADE FEDERAL DO CEARÁ**  
**CIÊNCIA DA COMPUTAÇÃO**

Victor Emanuel de Sousa Costa - 535718

Kaynan Pereira de Sousa - 540864

**TRABALHO DE ESTRUTURA DE DADOS**

Matrizes esparsas

Quixadá/CE

2022

## 1. DESCRIÇÃO COMPLETA E SUAS APLICAÇÕES

A matriz esparsa de maneira geral é uma matriz onde os valores 0 estão em maior quantidade, se comparados aos outros valores, esses valores zerados, vistos da perspectiva computacional, não serão alocados, tornando assim a matriz mais otimizada e funcional ao seu propósito, visto também no campo da engenharia e da matemática, como por exemplo:

- método das malhas para resolução de circuitos elétricos ou sistemas de equações lineares.
- Microsoft Excel
- Machine Learning
- Análise de dados
- Entre outros exemplos de aplicações

## 2. DECISÕES TOMADAS E FUNÇÕES AUXILIARES

Primordialmente, no nosso projeto de matriz esparsa criamos todas as funções que estão submetidas no trabalho. Além disso, criamos mais duas funções auxiliares que foram:

- Função *getColuna()*;  
retorna o número de colunas da matriz esparsa.
- Função *getLinha()*;  
retorna o número de linhas da matriz esparsa.

Funções estas, que facilitaram a implementação das funções Soma e Multiplicação da matriz.

Decidimos implementar uma Main interativa, na qual o usuário pode realizar operações com matrizes já pré-definidas do arquivo ou pode criar sua própria matriz de tamanho e valores de sua preferência.

Dividimos nossa main em 3(três) menus, sendo esses:

```
+=====+
| (1) - Operacoes com Matriz do Arquivo |
| (2) - Operacoes com Matriz do Usuario |
| (3) - Sair                             |
+=====+
Opcao: █
```

Menu principal.

```
+=====+
| (1) - Ler Matriz                               |
| (2) - Retorna o valor da celula da Matriz      |
| (3) - Soma                                      |
| (4) - Multiplicacao                            |
| (5) - Imprime                                  |
| (6) - Voltar                                   |
+=====+
Opcao: █
```

Menu para Operações com Matriz do arquivo, no qual são utilizadas as matrizes que colocamos no arquivo.

```
+=====+
| (1) - Criar Matriz                             |
| (2) - Insere o valor na Matriz                  |
| (3) - Soma                                      |
| (4) - Multiplicacao                            |
| (5) - Imprime                                  |
| (6) - Voltar                                   |
+=====+
Opcao: █
```

Menu para Operações com Matriz do Usuário, no qual são utilizadas matrizes que o usuário pode criar, podendo escolher seu tamanho e os valores a serem inseridos.

```
+=====+
|              DESENVOLVIDO                       |
|              POR:                                |
|=====|
|              - KAYNAN PEREIRA                    |
|              - VICTOR EMANUEL                    |
|=====|
|      CIENCIA DA COMPUTACAO 2022.2                |
|=====|
|      UFC - CAMPUS DE QUIXADA                     |
|=====|
|      TRABALHO DE                                |
|      ESTRUTURA DE DADOS                         |
|=====|
|      PROFESSOR:                                  |
|      ATILIO GOMES                               |
+=====+
```

Por fim, temos também uma tela de saída do programa.

### 3. DIVISÃO DO PROJETO

Não houve uma divisão exata em relação ao trabalho, nos reunimos diversas vezes, tanto para planejarmos nosso próximo passo, como também para realmente pôr a mão na massa e programar, até nesse aspecto da programação em si, utilizamos um site que nos auxiliou bastante, chamado *REPLIT*, o qual é um site de desenvolvimento integrado online, que nos permitiu a criação do projeto e da sua implementação em tempo real.

#### 4. DIFICULDADES

De início, nos reunimos e planejamos todos os passos que deveríamos percorrer nesse projeto, como os principais na formação da matriz, por exemplo o construtor, o destrutor, e a função insert. No nosso ponto de vista, o construtor e a insert foram as mais complicadas de implementar.

Programamos o construtor, com o intuito de criar o nó Head e os nós sentinelas, estes que seriam as bases das linhas e das colunas de nossa matriz. Para descobrir se estávamos criando corretamente o nosso Head e seus ponteiros, imprimimos no terminal os endereços de memória alocados pelos nós fazendo um:

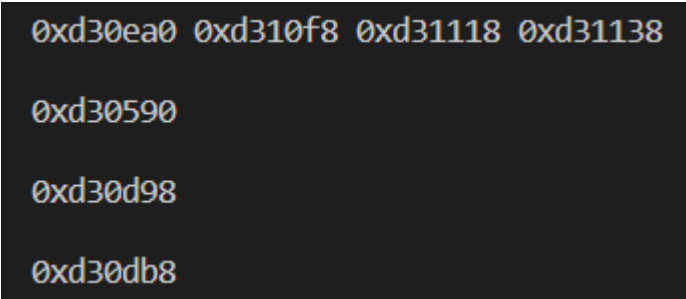
```
cout << m_head
      << m_head->direita
      << m_head->direita->direita
      << m_head->direita->direita->direita,
```

para os sentinelas das colunas, e:

```
cout << m_head
      << m_head->abaixo
      << m_head->abaixo->abaixo
      << m_head->abaixo->abaixo->abaixo,
```

para as linhas de uma Matriz 3x3.

Tivemos dificuldade em criar os nós sentinelas, pois cada vez que fazíamos a impressão no terminal, só saía o mesmo endereço de memória, ou seja, só era impresso o endereço do próprio Head, mas conseguimos resolver esse problema após desenhar e tentar mudar o código várias vezes. Caso esses endereços fossem diferentes uns dos outros, seria um sinal de que os nós sentinelas das linhas e das colunas estavam sendo criados corretamente.



```
0xd30ea0 0xd310f8 0xd31118 0xd31138

0xd30590

0xd30d98

0xd30db8
```

Além disso, tivemos outro problema muito importante na implementação do nosso código, que foi a perda contínua de ponteiros na inserção de um novo Node na matriz. Utilizando o mesmo exemplo da matriz 3 x 3, caso entrássemos com um valor X na linha 3 e coluna 2 e com outro valor Y na mesma linha porém na coluna 1, ao printarmos toda matriz, estávamos perdendo esse primeiro valor. Mas depois de analisar os Slides do conteúdo de Listas encadeadas, ver diversos vídeos no YouTube sobre o tema e tendo vários testes mal sucedidos, conseguimos um norte para nos estabilizar novamente no trabalho e, com isso, conseguimos terminar a nossa insert().

Outro problema que nos trouxe dor de cabeça foi o método print(), porque chamamos o método getValor() para facilitar a impressão da matriz no terminal, porém a matriz sempre saía com algum erro, o que nos levava a colocar toda culpa na nossa insert(). Depois de testar várias vezes a insert e não resolver os erros, percebemos que o problema poderia não estar lá, mas sim na getValor(). Decidimos implementar este método novamente, desconsiderando nossa última análise de como deveria ser feito, e após isso, o código passou a funcionar corretamente.

## 5. TESTES EXECUTADOS

Como nos baseamos em um esquema de Main interativa, os nossos testes são realizados pelo próprio usuário, deixando-o assim, livre para realizar as operações propostas sobre a matriz esparsa criada pelo mesmo.

Diante dessa perspectiva, iremos realizar os casos de teste baseado nos requerimentos feitos. Faremos um pequeno “guia”, para demonstrarmos o funcionamento da nossa Main.

- **Para Operações com Matrizes dos arquivos:**

Ao selecionarmos a primeira opção do menu inicial, devemos, primeiramente lermos as matrizes do arquivo, para assim, realizarmos as demais operações

```
+=====+
| (1) - Ler Matriz
| (2) - Retorna o valor da celula da Matriz
| (3) - Soma
| (4) - Multiplicacao
| (5) - Imprime
| (6) - Voltar
+=====+
Opcao: 1

Digite o numero do arquivo (1 ou 2): 1

Matriz 1 lida com sucesso!
```

Como no exemplo acima, lemos a matriz 1, podemos agora imprimir essa matriz:

```
Opcao: 5

Digite qual matriz deseja imprimir: 1
50 0 0 0
10 0 20 0
0 0 0 0
-30 0 -60 -5
```

Caso tente, por exemplo, somar ou multiplicar as duas matrizes e alguma delas não tiver sido lida anteriormente, imprimirá uma mensagem de erro. Assim como em diversos casos de saída da lógica padrão estabelecida.

**\*Ou seja, leia as duas matrizes caso queira realizar uma dessas operações!**

```
Opcao: 3

ERROR: Nao foi possivel realizar a soma, nao possui as duas matrizes necessarias para a operacao.
```

Mas, vamos imaginar que o usuário leu uma das duas matrizes corretamente, e deseja, retornar um valor de uma célula específica de uma das matrizes lidas. Caso o usuário tente retornar o valor de uma matriz que ainda não foi lida, será lançada uma mensagem de erro, e essas mensagens de erro funcionam para qualquer tentativa de realizar uma operação com uma matriz não lida.

```
Opcao: 2

Digite qual matriz deseja verificar: 1
Digite a linha e a coluna da matriz: 1 1
Valor: 50
```

```
4 4
1 1 50.0
2 1 10.0
2 3 20.0
4 1 -30.0
4 3 -60.0
4 4 -5.0
```

(Arquivo da matriz 1, exemplificada acima).

A mensagem de erro também aparecerá se o usuário tentar imprimir uma matriz que ainda não foi lida, exemplo:

```
Opcao: 5

Digite qual matriz deseja imprimir: 2
ERROR: Matriz nao encontrada!
```

(Se a matriz desejada não foi lida).

```
Opcao: 5

ERROR: Nenhuma matriz foi lida!
```

(Se nenhuma das matrizes foi lida).

- **Para Operações com Matrizes criadas pelo Usuário:**

Ao selecionarmos a segunda opção do menu inicial, primeiramente o usuário deverá criar a matriz. Caso queira efetuar as operações de soma e multiplicação, terá que criar as duas matrizes. Porém, se o mesmo desejar apenas criar uma matriz, inserir valores nela ou imprimir a matriz, essas operações podem ser realizadas, desde que sejam submetidas de maneira correta.

```
Opcao: 1

Crie a Matriz 1 ou a Matriz 2: 1
Insira o tamanho da matriz: 3 3

Matriz 1 criada com sucesso!
```

(criamos a matriz 1, com o tamanho 3x3)

```
Opcao: 2
```

```
Deseja inserir o valor na Matriz 1 ou 2? 1  
Quantos valores voce quer inserir? 2  
Digite a linha, a coluna e o valor: 1 1 3  
Digite a linha, a coluna e o valor: 2 2 4  
Valores inseridos com sucesso!
```

(inserimos 2(dois) valores na matriz 1)

```
Opcao: 5
```

```
Digite qual matriz deseja imprimir: 1
```

```
3 0 0  
0 4 0  
0 0 0
```

Ao selecionarmos a opção 5(cinco), imprimimos nossa matriz 1

Agora, iremos criar e inserir valores na segunda matriz para, somente assim, realizarmos as operações de soma e multiplicação das matrizes.

```
Opcao: 2
```

```
Deseja inserir o valor na Matriz 1 ou 2? 2  
Quantos valores voce quer inserir? 2  
Digite a linha, a coluna e o valor: 1 3 5  
Digite a linha, a coluna e o valor: 2 2 6  
Valores inseridos com sucesso!
```

(inserimos 2(dois) valores na matriz 2)

```
Opcao: 5
```

```
Digite qual matriz deseja imprimir: 2
```

```
0 0 5  
0 6 0  
0 0 0
```

(Impressão da matriz 2)



Opcao: 3

3	0	5
0	10	0
0	0	0

Opcao: 4

0	0	15
0	24	0
0	0	0

(Soma e multiplicação das matrizes criadas acima, respectivamente)

## 6. ANÁLISE DE COMPLEXIDADE

Nessas análises, poderíamos focar apenas nas complexidades não constantes, como a  $O(n)$  ou  $O(n^2)$ , pois as complexidades constantes não influenciam diretamente no resultado final.

- **Análise da função insert().**

```
void SparseMatrix::insert(int i, int j, double value) {
    if (i > 0 && i <= m_linha && j > 0 && j <= m_coluna) { // C1

        Node * prev = {nullptr};
        Node * aux;

        linha = coluna = m_head; // C2

        for (int k = 1; k <= i; k++) {
            linha = linha -> abaixo; // C3 * (n)
        }
        aux = linha;

        for (int k = 1; k <= j; k++) {
            coluna = coluna -> direita; // C4 * (n)
        }

        prev = aux; // C5
        aux = aux -> direita; // C6

        while (aux != linha && aux -> coluna < j) {
            prev = aux; // C7 * (n)
            aux = aux -> direita; // C8 * (n)
        }

        if (aux != linha && aux -> coluna == j && aux -> linha == i) { // C9

            if (value == 0) { // C10
                if (prev == linha) { // C11
                    linha -> direita = aux -> direita; // C12
                } else {
                    prev -> direita = aux -> direita;
                    coluna -> abaixo = aux -> abaixo;
                }
            }
            delete aux; // C13
        }
    }
}
```

```

/*
 *  Analise de Pior Caso dessa funcao:
 *  C1 + C2 + C3 * (n) + C4 * (n) + C5 + C6 + C7 * (n) + C8 * (n) + C9 + C10 + C11 + C12 + C13
 *  C1 + C2 + C3n + C4n + C5 + C6 + C7n + C8n + C9 + C10 + C11 + C12 + C13
 *  C1 + C2 + C5 + C6 + C9 + C10 + C11 + C12 + C13 + C3n + C4n + C7n + C8n
 *  C1 + C2 + C5 + C6 + C9 + C10 + C11 + C12 + C13 = A
 *  A + C3n + C4n + C7n + C8n
 *  A + n(C3 + C4 + C7 + C8)
 *  C3 + C4 + C7 + C8 = B
 *  A + nB
 *  A + nB <= An + Bn
 *  A + nB <= n(A + B)
 *  A + B = C
 *  A + nB <= n(C)
 *  A + nB <= O(n)
 */

```

Portanto, na função insert() temos uma complexidade  $O(n)$

- **Análise da função getValor().**

```

double SparseMatrix::getValor(int i, int j) {

    if (i > 0 && j > 0 && i <= m_linha && j <= m_coluna) { // C1

        linha = m_head -> abaixo; // C2

        for (int k = 1; k < i; k++) {
            linha = linha -> abaixo; // C3 * (n - 1)
        }

        Node * aux;
        aux = linha -> direita; // C4

        while (aux != linha) {

            if (aux -> coluna == j && aux -> linha == i) { // C5 * (n)
                return aux -> valor;
            }
            aux = aux -> direita; // C6 * (n)
        }
    } else {
        try {
            throw "Erro: Indices invalidos!";
        } catch (const char * msg) {
            cerr << msg << endl;
        }
    }

    return 0;
}

```

```

/*
 *  Analise de pior caso:
 *  C1 + C2 + C3 * (n - 1) + C4 + C5 * (n) + C6 * (n)
 *  C1 + C2 + C4 + C3n - C3 + C5n + C6n
 *  C1 + C2 + C4 - C3 + C3n + C5n + C6n
 *  C1 + C2 + C4 - C3 = A
 *  A + C3n + C5n + C6n
 *  A + n(C3 + C5 + C6)
 *  C3 + C5 + C6 = B
 *  A + nB
 *  A + nB <= An + Bn
 *  A + nB <= n(A + B)
 *  A + B = C
 *  A + nB <= n(C)
 *  A + nB <= O(n)
 */

```

Portanto, na função `getValor()` temos uma complexidade  $O(n)$

- **Análise da função `Soma()`.**

```
SparseMatrix * Soma(SparseMatrix * a, SparseMatrix * b) {  
  
    if (a -> getLinha() == b -> getLinha() && a -> getColuna() == b -> getColuna()) { // C1  
  
        SparseMatrix * c = new SparseMatrix(a -> getLinha(), a -> getColuna());  
  
        for (int i = 1; i <= a -> getLinha(); i++) {  
            for (int j = 1; j <= a -> getColuna(); j++) {  
                c -> insert(i, j, a -> getValor(i, j) + b -> getValor(i, j)); // é chamado o metodo getValor que tem complexidade O(n)  
                // dentro do metodo insert que tem complexidade O(n) logo a complexidade fica O(n^2). Esses metodos são chamados dentro de um  
                // for que tem complexidade O(n), então a complexidade fica O(n^3). Porém, esse for está dentro de outro for que tem  
                // complexidade O(n), então a complexidade dessa função fica O(n^4).  
            }  
        }  
        return c;  
    } else {  
        try {  
            throw "ERROR: Matrizes de tamanhos diferentes!";  
        } catch (const char * msg) {  
            cerr << msg << endl;  
        }  
        return 0;  
    }  
}
```

Portanto, na função `soma()` temos uma complexidade  $O(n^4)$ .

## 7. REFERÊNCIAS BIBLIOGRÁFICAS

1. NASCIMENTO, G. **Como limpar a tela em C++?** Disponível em: <<https://pt.stackoverflow.com/questions/290972/como-limpar-a-tela-em-c>>. Acesso em: 9 nov. 2022.
2. CAMPOS, B. F. **Curso de C++ #51 - Operações com arquivos (ifstream) - Parte 2.** Disponível em: <<https://www.youtube.com/watch?v=Tczymt0OkYo>>. Acesso em: 9 nov. 2022.
3. DIGIAMPIETRI, L. **Estrutura de Dados - Aula 14 - Matriz esparsa.** Disponível em: <[https://www.youtube.com/watch?v=C\\_ePgrEbLs0](https://www.youtube.com/watch?v=C_ePgrEbLs0)>. Acesso em: 9 nov. 2022.
4. MANZATO, M. **ALG1 - 09 - Resolvendo exercício sobre Matrizes Esparsas com Listas Cruzadas (live).** Disponível em: <<https://www.youtube.com/watch?v=CnIuR5sqgiA&t=1083s>>. Acesso em: 9 nov. 2022.