

Matriz Esparsa

1.3

Gerado por Doxygen 1.9.8

1 Matriz Esparsa	1
1.0.1 Ajustes e melhorias	1
1.1 Pré-requisitos	1
1.2 Compilando a Matriz Esparsa	2
1.3 Executando a Matriz Esparsa	2
1.4 Contribuindo para Matriz Esparsa	2
1.5 Colaboradores	2
1.6 Licença	3
2 Índice dos Componentes	5
2.1 Lista de Classes	5
3 Índice dos Arquivos	7
3.1 Lista de Arquivos	7
4 Classes	9
4.1 Referência da Classe IteratorM	9
4.1.1 Descrição detalhada	10
4.1.2 Documentação das definições de tipos	10
4.1.2.1 difference_type	10
4.1.2.2 iterator_category	10
4.1.2.3 pointer	10
4.1.2.4 reference	10
4.1.2.5 value_type	10
4.1.3 Construtores e Destrutores	11
4.1.3.1 IteratorM() [1/2]	11
4.1.3.2 IteratorM() [2/2]	11
4.1.4 Documentação das funções	11
4.1.4.1 operator!=(())	11
4.1.4.2 operator*() [1/2]	12
4.1.4.3 operator*() [2/2]	12
4.1.4.4 operator++()	12
4.1.4.5 operator->() [1/2]	13
4.1.4.6 operator->() [2/2]	13
4.1.4.7 operator==(())	13
4.1.5 Documentação dos símbolos amigos e relacionados	14
4.1.5.1 Matriz	14
4.2 Referência da Classe Matriz	14
4.2.1 Descrição detalhada	15
4.2.2 Construtores e Destrutores	15
4.2.2.1 Matriz() [1/2]	15
4.2.2.2 Matriz() [2/2]	16
4.2.2.3 ~Matriz()	17

4.2.3 Documentação das funções	17
4.2.3.1 begin() [1/2]	17
4.2.3.2 begin() [2/2]	17
4.2.3.3 end() [1/2]	18
4.2.3.4 end() [2/2]	18
4.2.3.5 get() [1/2]	18
4.2.3.6 get() [2/2]	19
4.2.3.7 getColunas()	20
4.2.3.8 getLinhas()	20
4.2.3.9 insert()	20
4.2.3.10 limpar()	22
4.2.3.11 print()	22
4.3 Referência da Estrutura Node	23
4.3.1 Descrição detalhada	23
4.3.2 Construtores e Destrutores	23
4.3.2.1 Node()	23
4.3.3 Documentação das funções	24
4.3.3.1 atualizaValor()	24
4.3.4 Atributos	24
4.3.4.1 abaixo	24
4.3.4.2 coluna	24
4.3.4.3 direita	25
4.3.4.4 linha	25
4.3.4.5 valor	25
5 Arquivos	27
5.1 Referência do Arquivo include/IteratorM/IteratorM.hpp	27
5.2 IteratorM.hpp	28
5.3 Referência do Arquivo include/matriz/Matriz.hpp	29
5.4 Matriz.hpp	30
5.5 Referência do Arquivo include/node/Node.hpp	31
5.6 Node.hpp	31
5.7 Referência do Arquivo include/utils/utils.hpp	32
5.7.1 Funções	32
5.7.1.1 multiply()	32
5.7.1.2 sum()	33
5.8 utils.hpp	34
5.9 Referência do Arquivo README.md	35
5.10 Referência do Arquivo src/main/main.cpp	35
5.10.1 Descrição detalhada	36
5.10.2 Definições dos tipos	37
5.10.2.1 string	37

5.10.2.2 unordered_map	37
5.10.3 Enumerações	37
5.10.3.1 Opcoes	37
5.10.4 Funções	37
5.10.4.1 escolherMatrizes()	37
5.10.4.2 existeMatriz()	38
5.10.4.3 main()	38
5.10.4.4 printMatrizes()	40
5.10.4.5 readMatrix()	41
5.10.4.6 salvarMatriz()	42
5.11 main.cpp	42
5.12 Referência do Arquivo src/matriz/Matriz.cpp	46
5.13 Matriz.cpp	46
5.14 Referência do Arquivo tests/TestMatriz.cpp	49
5.14.1 Descrição detalhada	50
5.14.2 Funções	50
5.14.2.1 arquivoExiste()	50
5.14.2.2 leitura()	50
5.14.2.3 main()	51
5.14.2.4 testeInsercao()	51
5.14.2.5 testeMultiplicacao()	51
5.14.2.6 testePerformance()	52
5.14.2.7 testeSoma()	53
5.15 TestMatriz.cpp	54
Índice Remissivo	57

Capítulo 1

Matriz Esparsa

Este projeto implementa uma [Matriz](#) Esparsa eficiente, otimizando espaço ao armazenar somente valores diferentes de zero.

1.0.1 Ajustes e melhorias

As próximas atualizações para a [Matriz](#) Esparsa serão:

- ☒ Implementar inserção e remoção dinâmicas
- ☐ Otimizar a busca de elementos
- ☒ Adicionar testes automatizados
- ☐ Melhorar documentação de uso
- ☐ Suporte a diferentes tipos de dados

1.1 Pré-requisitos

Antes de começar, verifique se você atendeu aos seguintes requisitos:

- Git instalado
- Compilador C++ instalado
- Makefile instalado
- Sistema operacional Windows / Linux / Mac
- Consultou a [documentação](#) do projeto

1.2 Compilando a Matriz Esparsa

Para compilar a [Matriz](#) Esparsa, siga estas etapas:

Dentro do diretório do projeto, execute o seguinte comando:

- Linux e macOS:

```
make
```

- Windows:

```
mingw32-make
```

1.3 Executando a Matriz Esparsa

Para usar a [Matriz](#) Esparsa, execute o seguinte comando:

Dentro do diretório do projeto, execute o seguinte comando:

- Linux e macOS:

```
./bin/Matriz-Esparsa
```

- Windows:

```
bin/Matriz-Esparsa.exe
```

Personalize os exemplos conforme necessário.

1.4 Contribuindo para Matriz Esparsa

Para contribuir com a [Matriz](#) Esparsa:

1. Bifurque este repositório.
2. Crie um branch: `git checkout -b <nome_branch>`.
3. Faça suas alterações e confirme-as: `'git commit -m '<mensagem_commit>'`
4. Envie para o branch original: `git push origin Matriz-Esparsa/<local>`
5. Crie a solicitação de pull.

Consulte [como criar uma solicitação pull](#).

1.5 Colaboradores

Agradecimentos especiais aos seguintes colaboradores:

1.6 Licença

Este projeto está sob licença. Consulte [LICENÇA](LICENSE) para mais informações.

Capítulo 2

Índice dos Componentes

2.1 Lista de Classes

Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:

IteratorM	Iterador para percorrer uma matriz esparsa	9
Matriz	Classe que representa uma matriz esparsa	14
Node	Representa um nó em uma matriz esparsa	23

Capítulo 3

Índice dos Arquivos

3.1 Lista de Arquivos

Esta é a lista de todos os arquivos e suas respectivas descrições:

include/IteratorM/IteratorM.hpp	27
include/matriz/Matriz.hpp	29
include/node/Node.hpp	31
include/utils/utils.hpp	32
src/main/main.cpp	
Programa para manipulação de matrizes esparsas	35
src/matriz/Matriz.cpp	46
tests/TestMatriz.cpp	
Arquivo de teste para operações com matrizes esparsas	49

Capítulo 4

Classes

4.1 Referência da Classe IteratorM

Iterador para percorrer uma matriz esparsa.

```
#include <IteratorM.hpp>
```

Tipos Públicos

- `using iterator_category = std::forward_iterator_tag`
- `using difference_type = std::ptrdiff_t`
- `using value_type = double`
- `using pointer = double *`
- `using reference = double &`

Membros Públicos

- `IteratorM ()`
Construtor padrão.
- `IteratorM (Node *cabecalho, Node *current)`
Construtor com parâmetros.
- `reference operator* ()`
Operador de desreferenciação.
- `reference operator* () const`
Operador de desreferenciação (const).
- `pointer operator-> ()`
Operador de acesso a membro (const).
- `pointer operator-> () const`
Operador de acesso a membro (const).
- `IteratorM & operator++ ()`
Operador de incremento prefixado.
- `bool operator== (const IteratorM &it) const`
Operador de igualdade.
- `bool operator!= (const IteratorM &it) const`
Operador de desigualdade.

Amigos

- [class Matriz](#)

4.1.1 Descrição detalhada

Iterador para percorrer uma matriz esparsa.

A classe [IteratorM](#) fornece um iterador para percorrer os elementos de uma matriz esparsa.

@friend class [Matriz](#)

Definição na linha [16](#) do arquivo [IteratorM.hpp](#).

4.1.2 Documentação das definições de tipos

4.1.2.1 difference_type

```
using IteratorM::difference_type = std::ptrdiff_t
```

Definição na linha [26](#) do arquivo [IteratorM.hpp](#).

4.1.2.2 iterator_category

```
using IteratorM::iterator_category = std::forward_iterator_tag
```

Definição na linha [25](#) do arquivo [IteratorM.hpp](#).

4.1.2.3 pointer

```
using IteratorM::pointer = double *
```

Definição na linha [28](#) do arquivo [IteratorM.hpp](#).

4.1.2.4 reference

```
using IteratorM::reference = double &
```

Definição na linha [29](#) do arquivo [IteratorM.hpp](#).

4.1.2.5 value_type

```
using IteratorM::value_type = double
```

Definição na linha [27](#) do arquivo [IteratorM.hpp](#).

4.1.3 Construtores e Destrutores

4.1.3.1 IteratorM() [1/2]

```
IteratorM::IteratorM ( ) [inline]
```

Construtor padrão.

Inicializa o iterador com ponteiros nulos.

Definição na linha 36 do arquivo [IteratorM.hpp](#).

```
00036 : cabecalho(nullptr), current(nullptr) {}
```

4.1.3.2 IteratorM() [2/2]

```
IteratorM::IteratorM (
    Node * cabecalho,
    Node * current ) [inline]
```

Construtor com parâmetros.

Inicializa o iterador com o nó de cabeçalho e o nó atual.

Parâmetros

<i>cabecalho</i>	Ponteiro para o nó de cabeçalho.
<i>current</i>	Ponteiro para o nó atual (padrão é nullptr).

Definição na linha 46 do arquivo [IteratorM.hpp](#).

```
00046 : cabecalho(cabecalho), current(current)
00047 {
00048     while (current == cabecalho)
00049     {
00050         cabecalho = cabecalho->abaixo;
00051         current = current->abaixo->direita;
00052     }
00053 }
```

4.1.4 Documentação das funções

4.1.4.1 operator!=(())

```
bool IteratorM::operator!= (
    const IteratorM & it ) const [inline]
```

Operador de desigualdade.

Compara se dois iteradores são diferentes.

Parâmetros

<i>it</i>	Iterador a ser comparado.
-----------	---------------------------

Retorna

true se os iteradores são diferentes, false caso contrário.

Definição na linha 144 do arquivo [IteratorM.hpp](#).

```
00145 {  
00146     return cabecalho != it.cabecalho || current != it.current;  
00147 }
```

4.1.4.2 operator*() [1/2]

[reference](#) IteratorM::operator* () [inline]

Operador de desreferenciação.

Retorna uma referência ao valor do nó atual.

Retorna

Referência ao valor do nó atual.

Definição na linha 62 do arquivo [IteratorM.hpp](#).

```
00063 {  
00064     return current->valor;  
00065 }
```

4.1.4.3 operator*() [2/2]

[reference](#) IteratorM::operator* () const [inline]

Operador de desreferenciação (const).

Retorna uma referência constante ao valor do nó atual.

Retorna

Referência constante ao valor do nó atual.

Definição na linha 74 do arquivo [IteratorM.hpp](#).

```
00075 {  
00076     return current->valor;  
00077 }
```

4.1.4.4 operator++()

[IteratorM](#) & IteratorM::operator++ () [inline]

Operador de incremento prefixado.

Avança o iterador para o próximo elemento na matriz esparsa.

Retorna

Referência ao próprio iterador após o incremento.

Definição na linha 110 do arquivo [IteratorM.hpp](#).

```
00111 {  
00112     current = current->direita;  
00113  
00114     while (current == cabecalho)  
00115     {  
00116         cabecalho = cabecalho->abaixo;  
00117         current = current->abaixo->direita;  
00118     }  
00119  
00120     return *this;  
00121 }
```

4.1.4.5 operator->() [1/2]

```
pointer IteratorM::operator-> ( ) [inline]
```

Operador de acesso a membro (const).

Retorna um ponteiro constante para o valor do nó atual.

Retorna

Ponteiro constante para o valor do nó atual.

Definição na linha 86 do arquivo [IteratorM.hpp](#).

```
00087 {  
00088     return &current->valor;  
00089 }
```

4.1.4.6 operator->() [2/2]

```
pointer IteratorM::operator-> ( ) const [inline]
```

Operador de acesso a membro (const).

Retorna um ponteiro constante para o valor do nó atual.

Retorna

Ponteiro constante para o valor do nó atual.

Definição na linha 98 do arquivo [IteratorM.hpp](#).

```
00099 {  
00100     return &current->valor;  
00101 }
```

4.1.4.7 operator==()

```
bool IteratorM::operator==(  
    const IteratorM & it ) const [inline]
```

Operador de igualdade.

Compara se dois iteradores são iguais.

Parâmetros

<i>it</i>	Iterador a ser comparado.
-----------	---------------------------

Retorna

true se os iteradores são iguais, false caso contrário.

Definição na linha 131 do arquivo [IteratorM.hpp](#).

```

00132     {
00133         return cabecalho == it.cabecalho && current == it.current;
00134     }

```

4.1.5 Documentação dos símbolos amigos e relacionados

4.1.5.1 Matriz

```
friend class Matriz [friend]
```

Definição na linha 18 do arquivo [IteratorM.hpp](#).

A documentação para essa classe foi gerada a partir do seguinte arquivo:

- [include/IteratorM/IteratorM.hpp](#)

4.2 Referência da Classe Matriz

Classe que representa uma matriz esparsa.

```
#include <Matriz.hpp>
```

Membros Públicos

- [Matriz \(\)](#)
Construtor padrão da classe [Matriz](#).
- [Matriz \(const int &ln, const int &cl\)](#)
Construtor da classe [Matriz](#) que inicializa uma matriz esparsa com linhas e colunas especificadas.
- [~Matriz \(\)](#)
Destrutor da classe [Matriz](#).
- [IteratorM begin \(\)](#)
Inicializa um iterador que aponta para o primeiro elemento significativo da matriz.
- [IteratorM end \(\)](#)
Iterador que aponta para o final da matriz.
- [IteratorM begin \(\) const](#)
Iterador que aponta para o primeiro elemento significativo da matriz (versão const).
- [IteratorM end \(\) const](#)
Iterador que aponta para o final da matriz (versão const).
- [int getLinhas \(\) const](#)
Retorna a quantidade de linhas da matriz.
- [int getColunas \(\) const](#)
Retorna a quantidade de colunas da matriz.
- [void limpar \(\)](#)
Limpa a matriz esparsa, removendo todos os nós.
- [void insert \(const int &posI, const int &posJ, const double &value\)](#)
Insere um valor em uma posição específica da matriz esparsa.
- [double get \(const int &posI, const int &posJ\)](#)
Retorna o valor armazenado na matriz em uma posição específica.
- [double get \(const int &posI, const int &posJ\) const](#)
Retorna o valor armazenado na matriz em uma posição específica (versão const).
- [void print \(\)](#)
Imprime a matriz no console.

4.2.1 Descrição detalhada

Classe que representa uma matriz esparsa.

A classe [Matriz](#) implementa uma estrutura de dados para armazenar matrizes esparsas, onde a maioria dos elementos são zeros. Utiliza uma lista encadeada de nós para armazenar apenas os elementos não-zero, economizando memória e permitindo operações eficientes.

A matriz é representada por um nó-cabeçalho que atua como sentinela para a estrutura interna. Este nó-cabeçalho aponta para si mesmo em ambas as direções (direita e abaixo) quando a matriz está vazia. A classe fornece métodos para inserir, acessar e remover elementos, além de obter o número de linhas e colunas e imprimir a matriz.

Observação

- A matriz é inicializada com um nó-cabeçalho que aponta para si mesmo.
- Os índices das linhas e colunas começam em 1.
- A inserção de elementos é feita apenas em posições válidas (dentro dos limites da matriz).

Aviso

- A tentativa de acessar ou inserir elementos em posições inválidas (fora dos limites da matriz) resultará em uma exceção `std::invalid_argument`.

Definição na linha [32](#) do arquivo [Matriz.hpp](#).

4.2.2 Construtores e Destrutores

4.2.2.1 Matriz() [1/2]

```
Matriz::Matriz ( )
```

Construtor padrão da classe [Matriz](#).

Este construtor inicializa os valores de linhas e colunas com zero e cria o nó-cabeçalho. Este nó-cabeçalho atua como sentinela para a estrutura interna da matriz, permitindo operações de inserção, remoção e acesso eficientes. Inicialmente, o nó-cabeçalho aponta para si mesmo em ambas as direções (direita e abaixo), indicando que a matriz ainda não possui elementos de dados.

Definição na linha [4](#) do arquivo [Matriz.cpp](#).

```
00004         : cabecalho(new Node(0, 0, 0)), linhas(0), colunas(0)
00005 {
00006     cabecalho->direita = cabecalho->abaixo = cabecalho;
00007 }
```

4.2.2.2 Matriz() [2/2]

```
Matriz::Matriz (
    const int & lin,
    const int & col )
```

Construtor da classe [Matriz](#) que inicializa uma matriz esparsa com linhas e colunas especificadas.

Este construtor verifica primeiro se os valores passados para o número de linhas (lin) e colunas (col) são maiores que zero. Caso contrário, é lançada uma exceção `std::invalid_argument`, garantindo que somente matrizes com dimensões válidas sejam criadas.

Em seguida, são inicializadas:

- As variáveis internas para armazenar a quantidade de linhas e colunas.
- Um nó-cabeçalho que serve como referência principal da estrutura, mantendo laços para si mesmo tanto à direita quanto abaixo.
- Um conjunto de nós auxiliares, linkados entre si de maneira circular:
 - Nós responsáveis pelas linhas (um nó para cada linha),
 - Nós responsáveis pelas colunas (um nó para cada coluna).

Cada nó de linha aponta para si mesmo à direita, enquanto cada nó de coluna aponta para si mesmo abaixo, formando estruturas circulares independentes para linhas e colunas, todas centralizadas no nó-cabeçalho.

Parâmetros

<i>lin</i>	Quantidade de linhas da matriz (deve ser um valor maior que zero).
<i>col</i>	Quantidade de colunas da matriz (deve ser um valor maior que zero).

Exceções

<code>std::invalid_argument</code>	Exceção lançada quando lin ou col são menores ou iguais a zero.
------------------------------------	---

Definição na linha 9 do arquivo [Matriz.cpp](#).

```
00011 {
00012     if (lin <= 0 || col <= 0)
00013         throw std::invalid_argument("Erro: Tamanho de matriz inválido, insira valores maiores que 0");
00014
00015     linhas = lin;
00016     colunas = col;
00017
00018     cabecalho = new Node(0, 0, 0);
00019     cabecalho->direita = cabecalho->abaixo = cabecalho;
00020
00021     Node *auxLinha = cabecalho;
00022     for (int i = 1; i <= lin; i++)
00023     {
00024         Node *novo = new Node(i, 0, 0);
00025         auxLinha->abaixo = novo;
00026         novo->direita = novo;
00027         auxLinha = novo;
00028     }
00029     auxLinha->abaixo = cabecalho;
00030
00031     Node *auxColuna = cabecalho;
00032     for (int j = 1; j <= col; j++)
00033     {
00034         Node *novo = new Node(0, j, 0);
00035         auxColuna->direita = novo;
00036         novo->abaixo = novo;
```

```

00037         auxColuna = novo;
00038     }
00039     auxColuna->direita = cabecalho;
00040 }

```

4.2.2.3 ~Matriz()

```
Matriz::~~Matriz ( )
```

Destrutor da classe [Matriz](#).

Este destrutor é responsável por liberar os recursos alocados pela instância da classe [Matriz](#). Ele chama a função [limpar\(\)](#) para garantir que toda a memória dinâmica utilizada pela matriz seja devidamente desalocada, evitando vazamentos de memória.

Definição na linha 72 do arquivo [Matriz.cpp](#).

```

00073 {
00074     limpar();
00075 }

```

4.2.3 Documentação das funções

4.2.3.1 begin() [1/2]

```
IteratorM Matriz::begin ( )
```

Inicializa um iterador que aponta para o primeiro elemento significativo da matriz.

Esta função permite obter um iterador para o primeiro nó principal da matriz esparsa (localizado logo abaixo do cabeçalho), facilitando o acesso aos elementos e a manipulação da estrutura de dados. O iterador retornado aponta para o primeiro nó relevante da linha principal, permitindo assim percorrer as colunas de forma apropriada.

Retorna

Retorna um objeto de iterador ([IteratorM](#)) posicionado no início da matriz esparsa.

Observação

Esse método presume que a matriz está devidamente inicializada e que o cabeçalho aponta para posicionamentos válidos da estrutura.

Definição na linha 42 do arquivo [Matriz.cpp](#).

```

00043 {
00044     return IteratorM(cabecalho->abaixo, cabecalho->abaixo->direita);
00045 }

```

4.2.3.2 begin() [2/2]

```
IteratorM Matriz::begin ( ) const
```

Iterador que aponta para o primeiro elemento significativo da matriz (versão const).

Esta função é uma sobrecarga da função [begin\(\)](#) que permite obter um iterador para o primeiro nó principal da matriz esparsa, sem permitir alterações nos valores da matriz. O iterador retornado aponta para o primeiro nó relevante

Retorna

[IteratorM](#) Objeto iterador apontando para o início da matriz.

Definição na linha 52 do arquivo [Matriz.cpp](#).

```

00053 {
00054     return IteratorM(cabecalho->abaixo, cabecalho->abaixo->direita);
00055 }

```

4.2.3.3 end() [1/2]

```
IteratorM Matriz::end ( )
```

Iterador que aponta para o final da matriz.

Esta função provê um objeto `IteratorM` associado à lista principal de nós da matriz esparsa, indicando uma posição que corresponde ao final da estrutura de dados.

Retorna

`IteratorM` Objeto iterador apontando para o final da matriz.

Definição na linha 47 do arquivo `Matriz.cpp`.

```
00048 {
00049     return IteratorM(cabecalho, cabecalho->direita);
00050 }
```

4.2.3.4 end() [2/2]

```
IteratorM Matriz::end ( ) const
```

Iterador que aponta para o final da matriz (versão const).

Esta função é uma sobrecarga da função `end()` que permite obter um iterador para o final da matriz esparsa, sem permitir alterações nos valores da matriz. O iterador retornado aponta para o final da estrutura de dados.

Retorna

`IteratorM` Objeto iterador apontando para o final da matriz.

Definição na linha 57 do arquivo `Matriz.cpp`.

```
00058 {
00059     return IteratorM(cabecalho, cabecalho->direita);
00060 }
```

4.2.3.5 get() [1/2]

```
double Matriz::get (
    const int & posI,
    const int & posJ )
```

Retorna o valor armazenado na matriz em uma posição específica.

Esta função busca o valor localizado na linha e coluna indicadas por `posI` e `posJ`. Se não houver nenhum valor armazenado nessa posição (ou seja, se o nó correspondente não existir), a função retornará 0. Caso a posição seja inválida, será lançada uma exceção do tipo `std::invalid_argument`.

Parâmetros

<i>posI</i>	Índice da linha solicitada.
<i>posJ</i>	Índice da coluna solicitada.

Retorna

O valor do tipo double encontrado na posição indicada, ou 0 caso não exista nenhum valor armazenado.

Exceções

<code>std::invalid_argument</code>	Se <code>posI</code> ou <code>posJ</code> forem menores ou iguais a 0 ou excederem as dimensões da matriz.
------------------------------------	--

Definição na linha 154 do arquivo [Matriz.cpp](#).

```

00155 {
00156     if (posI <= 0 || posI > linhas || posJ <= 0 || posJ > colunas)
00157         throw std::invalid_argument("Erro: Local de acesso inválido");
00158
00159     IteratorM it = begin();
00160
00161     while (it != end())
00162     {
00163         if (it.current->linha == posI && it.current->coluna == posJ)
00164             return *it;
00165
00166         if (it.current->linha > posI || (it.current->linha == posI && it.current->coluna > posJ))
00167             return 0;
00168
00169         ++it;
00170     }
00171
00172     return 0;
00173 }
```

4.2.3.6 get() [2/2]

```

double Matriz::get (
    const int & posI,
    const int & posJ ) const
```

Retorna o valor armazenado na matriz em uma posição específica (versão const).

Mesma funcionalidade da versão não-const, mas permite consultar valores em matrizes constantes, sem modificá-las.

Esta função é uma sobrecarga da função [get\(\)](#) que permite consultar valores em matrizes constantes, sem alterar seu conteúdo. A diferença é que esta versão não permite alterar o conteúdo da matriz, garantindo a integridade dos dados.

Parâmetros

<code>posI</code>	Índice da linha solicitada.
<code>posJ</code>	Índice da coluna solicitada.

Retorna

O valor do tipo double encontrado na posição indicada, ou 0

Exceções

<code>std::invalid_argument</code>	Se <code>posI</code> ou <code>posJ</code> forem menores ou iguais a 0 ou excederem as dimensões da matriz.
------------------------------------	--

Definição na linha 175 do arquivo [Matriz.cpp](#).

```

00176 {
00177     if (posI <= 0 || posI > linhas || posJ <= 0 || posJ > colunas)
00178         throw std::invalid_argument("Erro: Local de acesso inválido");
00179     IteratorM it = begin();
00180     while (it != end())
00181     {
00182         if (it.current->linha == posI && it.current->coluna == posJ)
00183             return *it;
00184         if (it.current->linha > posI || (it.current->linha == posI && it.current->coluna > posJ))
00185             return 0;
00186         ++it;
00187     }
00188     return 0;
00189 }
00190
00191
00192
00193
00194 }

```

4.2.3.7 getColunas()

```
int Matriz::getColunas ( ) const
```

Retorna a quantidade de colunas da matriz.

Retorna

Número de colunas da matriz.

Semelhante a [getLinhas\(\)](#), utilizada para consultar o total de colunas da estrutura.

Definição na linha 67 do arquivo [Matriz.cpp](#).

```

00068 {
00069     return colunas;
00070 }

```

4.2.3.8 getLinhas()

```
int Matriz::getLinhas ( ) const
```

Retorna a quantidade de linhas da matriz.

Retorna

Número de linhas da matriz.

Esta função permite consultar o total de linhas para verificação de limites ou para iterações relacionadas ao tamanho da matriz.

Definição na linha 62 do arquivo [Matriz.cpp](#).

```

00063 {
00064     return linhas;
00065 }

```

4.2.3.9 insert()

```

void Matriz::insert (
    const int & posI,
    const int & posJ,
    const double & value )

```

Insere um valor em uma posição específica da matriz esparsa.

Esta função permite inserir uma nova célula com valor diferente de zero em uma matriz esparsa, levando em conta sua organização em listas duplamente encadeadas na horizontal e vertical. Qualquer valor igual a zero é ignorado, pois não se armazena valores nulos na estrutura. Caso a posição informada não seja válida, uma exceção de argumento inválido é lançada.

Parâmetros

<i>posI</i>	Índice da linha na qual a célula será inserida. Deve ser um valor positivo e menor ou igual ao número total de linhas da matriz.
<i>posJ</i>	Índice da coluna na qual a célula será inserida. Deve ser um valor positivo e menor ou igual ao número total de colunas da matriz.
<i>value</i>	Valor a ser armazenado na nova célula. Valores iguais a zero não são inseridos na matriz.

Exceções

<i>std::invalid_argument</i>	Lançada quando (posI, posJ) excede os limites de linhas ou colunas definidos para a matriz.
------------------------------	---

A função percorre primeiro a lista horizontal (linha) correspondente para localizar a posição adequada. Caso já exista um nó na mesma coluna, o valor é atualizado. Se não existir, cria-se um novo nó para armazenar o valor na posição indicada. Após isso, a função também atualiza a referência vertical (coluna), posicionando o novo nó de forma adequada na estrutura de dados.

Definição na linha 97 do arquivo [Matriz.cpp](#).

```

00098 {
00099     // Não armazena valores iguais a zero
00100     if (value == 0)
00101         return;
00102
00103     // Validação dos índices
00104     if (posI <= 0 || posI > linhas || posJ <= 0 || posJ > colunas)
00105         throw std::invalid_argument("Erro: Local de inserção inválido");
00106
00107     // Busca na lista horizontal (linha)
00108     Node *linhaAtual = cabecalho;
00109     while (linhaAtual->linha < posI)
00110     {
00111         linhaAtual = linhaAtual->abaixo;
00112     }
00113
00114     // Procura pela posição correta na linha
00115     Node *aux = linhaAtual;
00116     while (aux->direita != linhaAtual && aux->direita->coluna < posJ)
00117     {
00118         aux = aux->direita;
00119     }
00120
00121     // Verifica se o nó já existe
00122     if (aux->direita->coluna == posJ)
00123     {
00124         aux->direita->atualizaValor(value); // Atualiza o valor
00125         return;
00126     }
00127
00128     // Cria um novo nó
00129     Node *novo = new Node(posI, posJ, value);
00130
00131     // Insere o nó na lista horizontal
00132     novo->direita = aux->direita;
00133     aux->direita = novo;
00134
00135     // Busca na lista vertical (coluna)
00136     Node *colunaAtual = cabecalho;
00137     while (colunaAtual->coluna < posJ)
00138     {
00139         colunaAtual = colunaAtual->direita;
00140     }
00141
00142     // Procura pela posição correta na coluna
00143     aux = colunaAtual;
00144     while (aux->abaixo != colunaAtual && aux->abaixo->linha < posI)
00145     {
00146         aux = aux->abaixo;
00147     }
00148
00149     // Insere o nó na lista vertical
00150     novo->abaixo = aux->abaixo;
00151     aux->abaixo = novo;
00152 }
```

4.2.3.10 limpar()

```
void Matriz::limpar ( )
```

Limpa a matriz esparsa, removendo todos os nós.

Esta função percorre todos os nós da matriz esparsa, começando pelo nó imediatamente abaixo do cabeçalho e, em seguida, removendo todos os nós em cada linha e coluna. A função utiliza dois loops aninhados: o loop externo percorre cada linha da matriz, enquanto o loop interno percorre cada nó em uma linha específica. Cada nó é deletado para liberar a memória alocada dinamicamente.

Variáveis:

- linhaAtual: Ponteiro para o nó atual na linha que está sendo processada.
- colunaAtual: Ponteiro para o nó atual na coluna que está sendo processada.
- aux: Ponteiro auxiliar que aponta para o nó imediatamente abaixo do cabeçalho.

O loop externo continua até que todas as linhas tenham sido processadas, e o loop interno continua até que todos os nós em uma linha específica tenham sido deletados.

Definição na linha 77 do arquivo [Matriz.cpp](#).

```
00078 {
00079     Node *linhaAtual = cabecalho->abaixo;
00080     Node *colunaAtual = linhaAtual->direita;
00081     Node *aux = cabecalho->abaixo;
00082
00083     while (linhaAtual != aux)
00084     {
00085         while (colunaAtual != linhaAtual)
00086         {
00087             Node *temp = colunaAtual;
00088             colunaAtual = colunaAtual->direita;
00089             delete temp;
00090         }
00091         Node *temp = linhaAtual;
00092         linhaAtual = linhaAtual->abaixo;
00093         delete temp;
00094     }
00095 }
```

4.2.3.11 print()

```
void Matriz::print ( )
```

Imprime a matriz no console.

Exibe todas as linhas e colunas da matriz, mostrando os valores armazenados. Para posições onde não há valor armazenado (na forma esparsa), é imprimido o número 0.

Definição na linha 196 do arquivo [Matriz.cpp](#).

```
00197 {
00198     IteratorM it = begin();
00199
00200     for (int i = 1; i <= linhas; i++)
00201     {
00202         for (int j = 1; j <= colunas; j++)
00203         {
00204             if (it.current->linha == i && it.current->coluna == j)
00205             {
00206                 std::cout << std::fixed << std::setprecision(1) << *it;
00207                 ++it;
00208             }
00209             else
00210             {
00211                 std::cout << "0.0";
00212             }
00213             std::cout << " ";
00214         }
00215     }
00216     std::cout << std::endl;
00217 }
00218 }
```

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- `include/matriz/`[Matriz.hpp](#)
- `src/matriz/`[Matriz.cpp](#)

4.3 Referência da Estrutura Node

Representa um nó em uma matriz esparsa.

```
#include <Node.hpp>
```

Diagrama de colaboração para Node:



Membros Públicos

- [Node](#) (`const int &linha, const int &coluna, const double &valor`)
Construtor da classe [Node](#).
- `void atualizaValor` (`const double &novoValor`)
Atualiza o valor do nó.

Atributos Públicos

- `Node * direita`
- `Node * abaixo`
- `int linha`
- `int coluna`
- `double valor`

4.3.1 Descrição detalhada

Representa um nó em uma matriz esparsa.

Esta struct é usada para representar um nó em uma matriz esparsa, que contém ponteiros para os nós à direita e abaixo, bem como a linha, coluna e valor do elemento.

Definição na linha 11 do arquivo [Node.hpp](#).

4.3.2 Construtores e Destrutores

4.3.2.1 Node()

```
Node::Node (
    const int & linha,
    const int & coluna,
    const double & valor ) [inline]
```

Construtor da classe [Node](#).

Este construtor inicializa um objeto [Node](#) com os valores fornecidos para linha, coluna e valor.

Parâmetros

<i>linha</i>	Referência constante para o número da linha.
<i>coluna</i>	Referência constante para o número da coluna.
<i>valor</i>	Referência constante para o valor armazenado no nó.

Definição na linha 28 do arquivo [Node.hpp](#).

```
00028      : linha(linha), coluna(coluna),
      valor(valor)
00029      {
00030          direita = nullptr;
00031          abaixo = nullptr;
00032      }
```

4.3.3 Documentação das funções**4.3.3.1 atualizaValor()**

```
void Node::atualizaValor (
    const double & novoValor ) [inline]
```

Atualiza o valor do nó.

Esta função atualiza o valor armazenado no nó com um novo valor fornecido.

Parâmetros

<i>novoValor</i>	O novo valor que substituirá o valor atual do nó.
------------------	---

Definição na linha 41 do arquivo [Node.hpp](#).

```
00042      {
00043          valor = novoValor;
00044      }
```

4.3.4 Atributos**4.3.4.1 abaixo**

```
Node* Node::abaixo
```

Ponteiro para o próximo nó na mesma coluna.

Definição na linha 14 do arquivo [Node.hpp](#).

4.3.4.2 coluna

```
int Node::coluna
```

Número da coluna onde o nó está localizado.

Definição na linha 16 do arquivo [Node.hpp](#).

4.3.4.3 direita

```
Node* Node::direita
```

Ponteiro para o próximo nó na mesma linha.

Definição na linha 13 do arquivo [Node.hpp](#).

4.3.4.4 linha

```
int Node::linha
```

Número da linha onde o nó está localizado.

Definição na linha 15 do arquivo [Node.hpp](#).

4.3.4.5 valor

```
double Node::valor
```

Valor armazenado no nó.

Definição na linha 17 do arquivo [Node.hpp](#).

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- [include/node/Node.hpp](#)

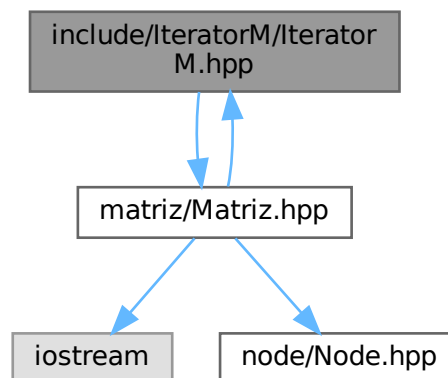
Capítulo 5

Arquivos

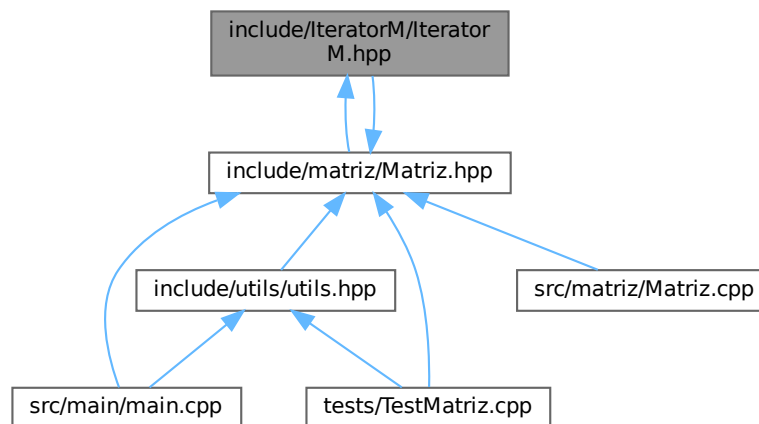
5.1 Referência do Arquivo include/IteratorM/IteratorM.hpp

```
#include "matriz/Matriz.hpp"
```

Gráfico de dependência de inclusões para IteratorM.hpp:



Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:



Componentes

- class [IteratorM](#)

Iterador para percorrer uma matriz esparsa.

5.2 IteratorM.hpp

[Ir para a documentação desse arquivo.](#)

```

00001 #ifndef ITERATORM_HPP
00002 #define ITERATORM_HPP
00003
00004 #include "matriz/Matriz.hpp"
00005
00006 class Matriz;
00007
00016 class IteratorM
00017 {
00018     friend class Matriz;
00019
00020 private:
00021     Node *cabecalho;
00022     Node *current;
00024 public:
00025     using iterator_category = std::forward_iterator_tag;
00026     using difference_type = std::ptrdiff_t;
00027     using value_type = double;
00028     using pointer = double *;
00029     using reference = double &;
00030
00036     IteratorM() : cabecalho(nullptr), current(nullptr) {}
00037
00046     IteratorM(Node *cabecalho, Node *current) : cabecalho(cabecalho), current(current)
00047     {
00048         while (current == cabecalho)
00049         {
00050             cabecalho = cabecalho->abaixo;
00051             current = current->abaixo->direita;
00052         }
00053     }
00054
00062     reference operator*()
00063     {
00064         return current->valor;
00065     }
  
```

```

00066
00074     reference operator*() const
00075     {
00076         return current->valor;
00077     }
00078
00086     pointer operator->()
00087     {
00088         return &current->valor;
00089     }
00090
00098     pointer operator->() const
00099     {
00100         return &current->valor;
00101     }
00102
00110     IteratorM &operator++()
00111     {
00112         current = current->direita;
00113
00114         while (current == cabecalho)
00115         {
00116             cabecalho = cabecalho->abaixo;
00117             current = current->abaixo->direita;
00118         }
00119
00120         return *this;
00121     }
00122
00131     bool operator==(const IteratorM &it) const
00132     {
00133         return cabecalho == it.cabecalho && current == it.current;
00134     }
00135
00144     bool operator!=(const IteratorM &it) const
00145     {
00146         return cabecalho != it.cabecalho || current != it.current;
00147     }
00148 };
00149
00150 #endif

```

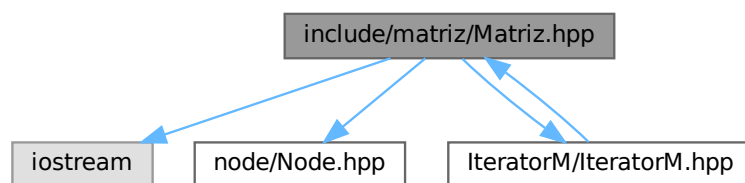
5.3 Referência do Arquivo include/matriz/Matriz.hpp

```

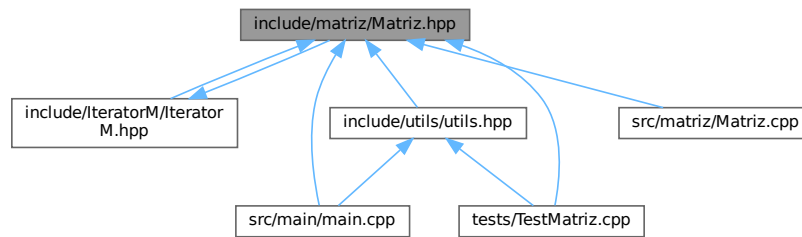
#include <iostream>
#include "node/Node.hpp"
#include "IteratorM/IteratorM.hpp"

```

Gráfico de dependência de inclusões para Matriz.hpp:



Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:



Componentes

- class [Matriz](#)

Classe que representa uma matriz esparsa.

5.4 Matriz.hpp

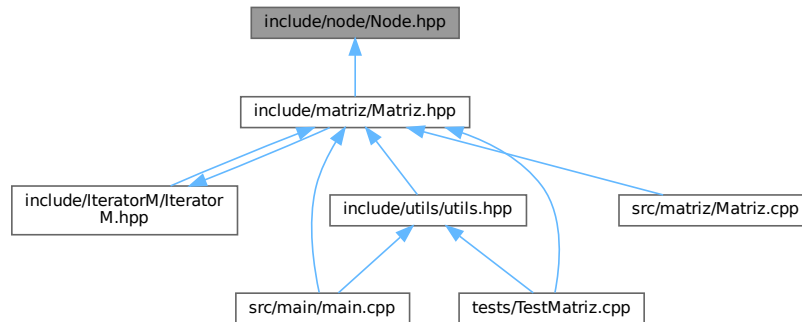
[Ir para a documentação desse arquivo.](#)

```

00001 #ifndef MATRIZ_HPP
00002 #define MATRIZ_HPP
00003
00004 #include <iostream>
00005 #include "node/Node.hpp"
00006 #include "IteratorM/IteratorM.hpp"
00007
00032 class Matriz
00033 {
00034 private:
00035     Node *cabecalho;
00036     int linhas;
00037     int colunas;
00039 public:
00050     Matriz();
00051
00077     Matriz(const int &ln, const int &cl);
00078
00086     ~Matriz();
00087
00102     IteratorM begin();
00103
00113     IteratorM end();
00114
00125     IteratorM begin() const;
00126
00136     IteratorM end() const;
00137
00146     int getLinhas() const;
00147
00156     int getColunas() const;
00157
00178     void limpar();
00179
00210     void insert(const int &posI, const int &posJ, const double &value);
00211
00230     double get(const int &posI, const int &posJ);
00231
00251     double get(const int &posI, const int &posJ) const;
00252
00260     void print();
00261 };
00262
00263 #endif
  
```

5.5 Referência do Arquivo include/node/Node.hpp

Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:



Componentes

- struct [Node](#)

Representa um nó em uma matriz esparsa.

5.6 Node.hpp

[Ir para a documentação desse arquivo.](#)

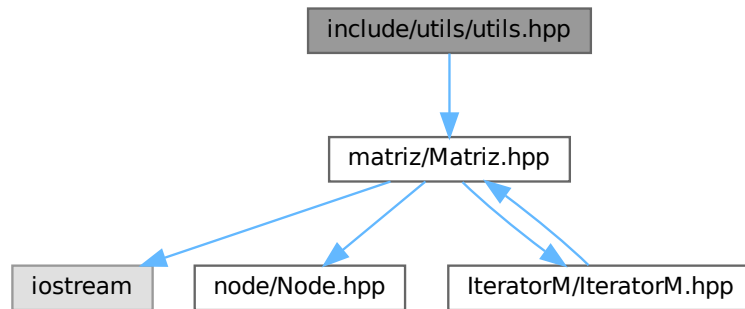
```

00001 #ifndef NODE_HPP
00002 #define NODE_HPP
00003
00011 struct Node
00012 {
00013     Node *direita;
00014     Node *abaixo;
00015     int linha;
00016     int coluna;
00017     double valor;
00028     Node(const int &linha, const int &coluna, const double &valor) : linha(linha), coluna(coluna),
    valor(valor)
00029     {
00030         direita = nullptr;
00031         abaixo = nullptr;
00032     }
00033
00041     void atualizaValor(const double &novoValor)
00042     {
00043         valor = novoValor;
00044     }
00045 };
00046
00047 #endif
  
```

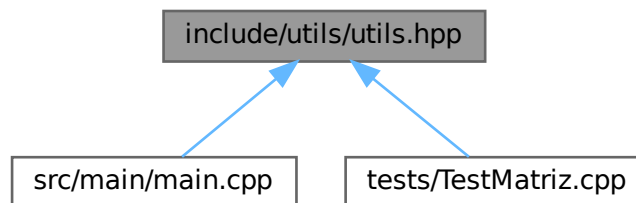
5.7 Referência do Arquivo include/utils/utils.hpp

```
#include "matriz/Matriz.hpp"
```

Gráfico de dependência de inclusões para utils.hpp:



Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:



Funções

- **Matriz sum** (`const Matriz &matrizA, const Matriz &matrizB`)
Soma duas matrizes de mesmo tamanho.
- **Matriz multiply** (`const Matriz &matrizA, const Matriz &matrizB`)
Multiplica duas matrizes e retorna a matriz resultante.

5.7.1 Funções

5.7.1.1 multiply()

```
Matriz multiply (  
    const Matriz & matrizA,  
    const Matriz & matrizB )
```

Multiplica duas matrizes e retorna a matriz resultante.

Esta função realiza a multiplicação de duas matrizes, `matrizA` e `matrizB`, e retorna a matriz resultante. A multiplicação de matrizes é possível apenas se o número de colunas de `matrizA` for igual ao número de linhas de `matrizB`. Caso contrário, uma exceção `std::invalid_argument` será lançada.

Parâmetros

<code>matrizA</code>	A primeira matriz a ser multiplicada.
<code>matrizB</code>	A segunda matriz a ser multiplicada.

Retorna

[Matriz](#) A matriz resultante da multiplicação de `matrizA` e `matrizB`.

Exceções

<code>std::invalid_argument</code>	Se o número de colunas de <code>matrizA</code> for diferente do número de linhas de <code>matrizB</code> .
------------------------------------	--

A multiplicação de matrizes é realizada da seguinte forma:

- Para cada linha *i* de `matrizA` e cada coluna *j* de `matrizB`, calcula-se o produto escalar entre a linha *i* de `matrizA` e a coluna *j* de `matrizB`.
- O valor resultante é inserido na posição (*i*, *j*) da matriz resultante.

Definição na linha 60 do arquivo `utils.hpp`.

```
00061 {
00062     // Verificação se a multiplicação é possível
00063     if (matrizA.getColunas() != matrizB.getLinhas())
00064     {
00065         throw std::invalid_argument("Erro: As matrizes precisam ter o mesmo número de colunas e
linhas");
00066     }
00067
00068     // Criando a matriz resultante (C)
00069     Matriz matriz(matrizA.getLinhas(), matrizB.getColunas());
00070
00071     // Percorrendo as linhas de matrizA e as colunas de matrizB
00072     for (int i = 1, linha = matrizA.getLinhas(); i <= linha; i++)
00073     {
00074         for (int j = 1, coluna = matrizB.getColunas(); j <= coluna; j++)
00075         {
00076             double valor = 0;
00077
00078             // Calculando o produto escalar entre a linha i de A e a coluna j de B
00079             for (int k = 1; k <= matrizA.getColunas(); k++)
00080             {
00081                 valor += matrizA.get(i, k) * matrizB.get(k, j);
00082             }
00083
00084             matriz.insert(i, j, valor);
00085         }
00086     }
00087
00088     return matriz; // Retorna a matriz resultante
00089 }
```

5.7.1.2 sum()

```
Matriz sum (
    const Matriz & matrizA,
    const Matriz & matrizB )
```

Soma duas matrizes de mesmo tamanho.

Esta função realiza a soma elemento a elemento das matrizes passadas por parâmetro, retornando uma nova matriz com o resultado.

Parâmetros

<i>matrixA</i>	Primeira matriz de entrada, cujas dimensões (linhas e colunas) devem ser iguais às de <i>matrizB</i> .
<i>matrizB</i>	Segunda matriz de entrada, com dimensões compatíveis com <i>matrixA</i> , para que a soma seja realizada corretamente.

Exceções

<i>std::invalid_argument</i>	Exceção lançada caso as matrizes fornecidas não possuam as mesmas dimensões, impossibilitando a operação de soma.
------------------------------	---

Retorna

Uma nova matriz que representa o resultado da soma elemento a elemento de *matrixA* e *matrizB*, mantendo as mesmas dimensões das matrizes de entrada.

Definição na linha 23 do arquivo *utils.hpp*.

```
00024 {
00025     if (matrixA.getLinhas() != matrizB.getLinhas() || matrixA.getColunas() != matrizB.getColunas())
00026         throw std::invalid_argument("Erro: As matrizes não possuem o mesmo tamanho");
00027
00028     Matriz matriz(matrixA.getLinhas(), matrixA.getColunas());
00029
00030     for (int i = 1, linha = matrixA.getLinhas(); i <= linha; i++)
00031     {
00032         for (int j = 1, coluna = matrixA.getColunas(); j <= coluna; j++)
00033         {
00034             double valor = matrixA.get(i, j) + matrizB.get(i, j);
00035             matriz.insert(i, j, valor);
00036         }
00037     }
00038 }
00039
00040 return matriz;
00041 }
```

5.8 utils.hpp

Ir para a documentação desse arquivo.

```
00001 #ifndef UTILS_HPP
00002 #define UTILS_HPP
00003
00004 #include "matriz/Matriz.hpp"
00005
00023 Matriz sum(const Matriz &matrixA, const Matriz &matrizB)
00024 {
00025     if (matrixA.getLinhas() != matrizB.getLinhas() || matrixA.getColunas() != matrizB.getColunas())
00026         throw std::invalid_argument("Erro: As matrizes não possuem o mesmo tamanho");
00027
00028     Matriz matriz(matrixA.getLinhas(), matrixA.getColunas());
00029
00030     for (int i = 1, linha = matrixA.getLinhas(); i <= linha; i++)
00031     {
00032         for (int j = 1, coluna = matrixA.getColunas(); j <= coluna; j++)
00033         {
00034             double valor = matrixA.get(i, j) + matrizB.get(i, j);
00035             matriz.insert(i, j, valor);
00036         }
00037     }
00038 }
00039
```



```

00040     return matriz;
00041 }
00042
00060 Matriz multiply(const Matriz &matrizA, const Matriz &matrizB)
00061 {
00062     // Verificação se a multiplicação é possível
00063     if (matrizA.getColunas() != matrizB.getLinhas())
00064     {
00065         throw std::invalid_argument("Erro: As matrizes precisam ter o mesmo número de colunas e
linhas");
00066     }
00067
00068     // Criando a matriz resultante (C)
00069     Matriz matriz(matrizA.getLinhas(), matrizB.getColunas());
00070
00071     // Percorrendo as linhas de matrizA e as colunas de matrizB
00072     for (int i = 1, linha = matrizA.getLinhas(); i <= linha; i++)
00073     {
00074         for (int j = 1, coluna = matrizB.getColunas(); j <= coluna; j++)
00075         {
00076             double valor = 0;
00077
00078             // Calculando o produto escalar entre a linha i de A e a coluna j de B
00079             for (int k = 1; k <= matrizA.getColunas(); k++)
00080             {
00081                 valor += matrizA.get(i, k) * matrizB.get(k, j);
00082             }
00083
00084             matriz.insert(i, j, valor);
00085         }
00086     }
00087
00088     return matriz; // Retorna a matriz resultante
00089 }
00090
00091 #endif

```

5.9 Referência do Arquivo README.md

5.10 Referência do Arquivo src/main/main.cpp

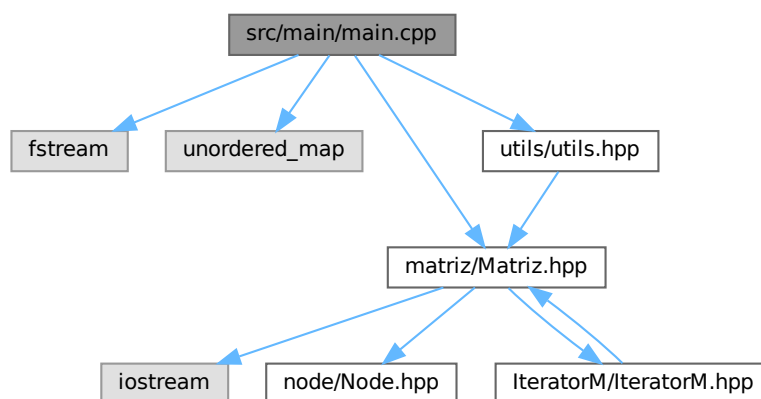
Programa para manipulação de matrizes esparsas.

```

#include <fstream>
#include <unordered_map>
#include "matriz/Matriz.hpp"
#include "utils/utils.hpp"

```

Gráfico de dependência de inclusões para main.cpp:



Definições de Tipos

- `using string = std::string`
- `using unordered_map = std::unordered_map< string, Matriz >`

Enumerações

- `enum Opcoes {
 LER_MATRIZ = 1 , IMPRIMIR_MATRIZ , SOMAR_MATRIZES , MULTIPLICAR_MATRIZES ,
 SAIR }`

Funções

- `void readMatrix (Matriz &matriz, const std::string filename)`
Lê dados de um arquivo e insere em uma matriz esparsa.
- `bool existeMatriz (const std::string filename, const unordered_map &matrizes)`
Verifica se existe uma matriz previamente armazenada em um `unordered_map`.
- `void salvarMatriz (const Matriz &matriz, unordered_map &matrizes)`
Salva uma matriz em um mapa associativo de matrizes, permitindo que seja recuperada posteriormente.
- `void escolherMatrizes (string &filename, string &filename2)`
Solicita ao usuário o nome de duas matrizes a serem processadas.
- `void printMatrizes (const unordered_map &matrizes)`
Exibe informações sobre as matrizes armazenadas em um `std::unordered_map`.
- `int main ()`

5.10.1 Descrição detalhada

Programa para manipulação de matrizes esparsas.

Este programa permite ao usuário ler, imprimir, somar e multiplicar matrizes esparsas. As matrizes são armazenadas em um mapa associativo, onde a chave é o nome do arquivo e o valor é a matriz correspondente.

O programa apresenta um menu interativo com as seguintes opções:

- Ler `Matriz`: Lê uma matriz a partir de um arquivo.
- Imprimir `Matriz`: Imprime uma matriz armazenada.
- Somar Matrizes: Soma duas matrizes armazenadas.
- Multiplicar Matrizes: Multiplica duas matrizes armazenadas.
- Sair: Encerra o programa.

Definição no arquivo `main.cpp`.

5.10.2 Definições dos tipos

5.10.2.1 string

```
using string = std::string
```

Autores

- Antonio Willian Silva Oliveira - 567294 (
- Iago de Oliveira Lo - 565321 (

Definição na linha 12 do arquivo [main.cpp](#).

5.10.2.2 unordered_map

```
using unordered_map = std::unordered_map<string, Matriz>
```

Definição na linha 13 do arquivo [main.cpp](#).

5.10.3 Enumerações

5.10.3.1 Opcoes

```
enum Opcoes
```

Enumeradores

LER_MATRIZ	
IMPRIMIR_MATRIZ	
SOMAR_MATRIZES	
MULTIPLICAR_MATRIZES	
SAIR	

Definição na linha 15 do arquivo [main.cpp](#).

```
00016 {  
00017     LER_MATRIZ = 1,  
00018     IMPRIMIR_MATRIZ,  
00019     SOMAR_MATRIZES,  
00020     MULTIPLICAR_MATRIZES,  
00021     SAIR  
00022 };
```

5.10.4 Funções

5.10.4.1 escolherMatrizes()

```
void escolherMatrizes (  
    string & filename,  
    string & filename2 )
```

Solicita ao usuário o nome de duas matrizes a serem processadas.

Parâmetros

<i>filename</i>	Referência para a string que armazenará o nome da primeira matriz.
<i>filename2</i>	Referência para a string que armazenará o nome da segunda matriz.

Definição na linha 335 do arquivo [main.cpp](#).

```
00336 {
00337     std::cout << "Coloque o nome da primeira matriz: ";
00338     std::getline(std::cin, filename);
00339
00340     std::cout << "Coloque o nome da segunda matriz: ";
00341     std::getline(std::cin, filename2);
00342 }
```

5.10.4.2 existeMatriz()

```
bool existeMatriz (
    const std::string filename,
    const unordered_map & matrizes )
```

Verifica se existe uma matriz previamente armazenada em um `unordered_map`.

Esta função realiza uma busca no `unordered_map` pelo nome do arquivo (`filename`) para determinar se já existe uma matriz associada a ele.

Parâmetros

<i>filename</i>	O nome do arquivo cujo registro de matriz deve ser verificado.
<i>matrizes</i>	O <code>unordered_map</code> que mantém o mapeamento entre nomes de arquivos e matrizes.

Retorna

Retorna `true` se existir a matriz correspondente ao nome do arquivo; caso contrário, `false`.

Observação

O uso do `unordered_map` permite que a busca ocorra de maneira eficiente, pois a busca possui complexidade média $O(1)$.

Definição na linha 290 do arquivo [main.cpp](#).

```
00291 {
00292     return matrizes.find(filename) != matrizes.end();
00293 }
```

5.10.4.3 main()

```
int main ( )
```

Definição na linha 115 do arquivo [main.cpp](#).

```
00116 {
00117     // Configura a localização para suportar caracteres especiais em português
00118     setlocale(LC_ALL, "pt_BR.UTF-8");
00119
00120     std::cout << "Bem-vindo ao programa de manipulação de matrizes esparsas" << std::endl;
00121     std::cout << "-----" << std::endl;
```

```

00122
00123 // Mapa associativo para armazenar as matrizes
00124 unordered_map matrizes;
00125
00126 while (true)
00127 {
00128     // Exibe o menu de opções
00129     std::cout << "Escolha uma opção:" << std::endl;
00130     std::cout << "[1] - Ler Matriz" << std::endl;
00131     std::cout << "[2] - Imprimir Matriz" << std::endl;
00132     std::cout << "[3] - Somar Matrizes" << std::endl;
00133     std::cout << "[4] - Multiplicar Matrizes" << std::endl;
00134     std::cout << "[5] - Sair" << std::endl;
00135     int opcao;
00136     std::cin >> opcao;
00137     std::cin.ignore();
00138
00139     switch (opcao)
00140     {
00141     case LER_MATRIZ:
00142     {
00143         // Lê uma matriz a partir de um arquivo
00144         string filename;
00145         std::cout << "Digite o nome do arquivo: ";
00146         std::getline(std::cin, filename);
00147
00148         Matriz matriz;
00149
00150         try
00151         {
00152             readMatrix(matriz, filename);
00153         }
00154         catch (const std::exception &e)
00155         {
00156             std::cerr << e.what() << '\n';
00157             break;
00158         }
00159
00160         // Armazena a matriz no mapa associativo
00161         matrizes.insert(std::make_pair(filename, matriz));
00162         break;
00163     }
00164
00165     case IMPRIMIR_MATRIZ:
00166     {
00167         // Imprime uma matriz armazenada
00168         std::cout << "Qual matriz deseja imprimir?" << std::endl;
00169         printMatrizes(matrizes);
00170         std::cout << "Coloque o nome do arquivo que deseja imprimir: ";
00171         string filename;
00172         std::getline(std::cin, filename);
00173
00174         if (existeMatriz(filename, matrizes))
00175             matrizes[filename].print();
00176         else
00177             std::cout << "Matriz não encontrada" << std::endl;
00178
00179         break;
00180     }
00181
00182     case SOMAR_MATRIZES:
00183     {
00184         // Soma duas matrizes armazenadas
00185         std::cout << "Quais as matrizes irá usar para somar" << std::endl;
00186         printMatrizes(matrizes);
00187         std::string filename, filename2;
00188         escolherMatrizes(filename, filename2);
00189
00190         if (!existeMatriz(filename, matrizes) || !existeMatriz(filename2, matrizes))
00191         {
00192             std::cout << "Alguma matriz não foi encontrada" << std::endl;
00193             break;
00194         }
00195
00196         Matriz matriz;
00197
00198         try
00199         {
00200             matriz = sum(matrizes[filename], matrizes[filename2]);
00201         }
00202         catch (const std::exception &e)
00203         {
00204             std::cerr << e.what() << '\n';
00205             break;
00206         }
00207
00208         matriz.print();

```

```

00209         salvarMatriz(matriz, matrizes);
00210         break;
00211     }
00212
00213     case MULTIPLICAR_MATRIZES:
00214     {
00215         // Multiplica duas matrizes armazenadas
00216         std::cout << "Quais as matrizes irá usar para multiplicar" << std::endl;
00217         printMatrizes(matrizes);
00218         std::string filename, filename2;
00219         escolherMatrizes(filename, filename2);
00220
00221         if (!existeMatriz(filename, matrizes) || !existeMatriz(filename2, matrizes))
00222         {
00223             std::cout << "Alguma matriz não foi encontrada" << std::endl;
00224             break;
00225         }
00226
00227         Matriz matriz;
00228
00229         try
00230         {
00231             matriz = multiply(matrizes[filename], matrizes[filename2]);
00232         }
00233         catch (const std::exception &e)
00234         {
00235             std::cerr << e.what() << '\n';
00236             break;
00237         }
00238
00239         matriz.print();
00240         salvarMatriz(matriz, matrizes);
00241         break;
00242     }
00243
00244     case SAIR:
00245     {
00246         // Encerra o programa
00247         std::cout << "Saindo..." << std::endl;
00248         return 0;
00249     }
00250
00251     default:
00252     {
00253         // Opção inválida
00254         std::cout << "Opção inválida" << std::endl;
00255         std::cin.clear();
00256         std::cin.ignore();
00257         break;
00258     }
00259 }
00260
00261 // Pausa e limpa a tela
00262 std::system("pause || read -p 'Pressione enter para continuar...' var");
00263 std::system("cls||clear");
00264 }
00265 }

```

5.10.4.4 printMatrizes()

```

void printMatrizes (
    const unordered_map & matrizes )

```

Exibe informações sobre as matrizes armazenadas em um `std::unordered_map`.

Parâmetros

<i>matrizes</i>	Um <code>std::unordered_map</code> que mapeia uma chave (nome da matriz) para uma instância de uma classe que fornece métodos <code>getLinhas()</code> e <code>getColunas()</code> .
-----------------	--

As dimensões exibidas são obtidas diretamente da instância armazenada em cada valor do map. A exibição é formatada para melhor leitura e compreensão dos dados de cada matriz.

Definição na linha 344 do arquivo `main.cpp`.

```

00345 {
00346     for (const auto &par : matrizes)
00347     {
00348         std::cout << "-----" << std::endl;
00349         std::cout << par.first << " |" << par.second.getLinhas() << " x " << par.second.getColunas() << "|"
<< std::endl;
00350         std::cout << "-----" << std::endl;
00351     }
00352 }

```

5.10.4.5 readMatrix()

```

void readMatrix (
    Matriz & matriz,
    const std::string filename )

```

Lê dados de um arquivo e insere em uma matriz esparsa.

Esta função abre um arquivo contendo o número de linhas e colunas de uma matriz, seguido por múltiplas linhas que descrevem itens não nulos da matriz. Cada linha deve conter índices (i, j) e o valor correspondente.

Parâmetros

<i>matriz</i>	Referência para o objeto Matriz que será inicializado e populado.
<i>filename</i>	Nome do arquivo a ser lido (sem o caminho completo).

Exceções

<i>std::runtime_error</i>	Quando não é possível acessar ou abrir o arquivo.
---------------------------	---

- O arquivo é aberto a partir do diretório "src/arquivos/" concatenado ao nome do arquivo passado em `filename`.
- Os primeiros valores lidos do arquivo correspondem ao número de linhas (`linhas`) e de colunas (`colunas`) para inicializar corretamente a matriz.
- Em seguida, cada conjunto de três valores (índice de linha, índice de coluna e valor) é lido e inserido na matriz usando `matriz.insert()`.
- Caso o arquivo não seja encontrado ou ocorra algum outro problema, é gerada uma exceção do tipo `std::runtime_error`.

Definição na linha [267](#) do arquivo [main.cpp](#).

```

00268 {
00269     std::ifstream file("src/arquivos/" + filename);
00270
00271     if (!file || !file.is_open())
00272         throw std::runtime_error("Erro ao abrir o arquivo");
00273
00274     int linhas{0}, colunas{0};
00275     file >> linhas >> colunas;
00276
00277     matriz = Matriz(linhas, colunas);
00278
00279     int i{0}, j{0};
00280     double valor{0.0f};
00281
00282     while (file >> i >> j >> valor)
00283     {
00284         matriz.insert(i, j, valor);
00285     }
00286
00287     file.close();
00288 }

```

5.10.4.6 salvarMatriz()

```
void salvarMatriz (
    const Matriz & matriz,
    unordered_map & matrizes )
```

Salva uma matriz em um mapa associativo de matrizes, permitindo que seja recuperada posteriormente.

A função solicita ao usuário se deseja salvar a matriz atual. Caso a resposta seja afirmativa, é solicitado um nome para identificar a nova matriz no mapa. Se esse nome já existir, o usuário é informado para tentar novamente com outro nome. Em caso de sucesso, a matriz é inserida no mapa com a chave fornecida pelo usuário, e é exibida uma mensagem de confirmação.

Parâmetros

<i>matriz</i>	Objeto do tipo Matriz que será salvo.
<i>matrizes</i>	Estrutura (<code>unordered_map</code>) onde a matriz será armazenada, associada a um nome (string).

Observação

Essa função não retorna valores. É importante que o usuário insira corretamente as opções (s ou n) para prosseguir ou cancelar o salvamento, e que forneça um nome válido quando optar por salvar a matriz.

Definição na linha [295](#) do arquivo [main.cpp](#).

```
00296 {
00297     while (true)
00298     {
00299         std::cout << "Deseja salvar a matriz? [S/N]: ";
00300         char resposta;
00301         std::cin >> resposta;
00302         std::cin.ignore();
00303
00304         switch (tolower(resposta))
00305         {
00306             case 'n':
00307                 return;
00308
00309             case 's':
00310             {
00311                 std::cout << "Digite o nome que deseja salvar a matriz: ";
00312                 string filename;
00313
00314                 std::getline(std::cin, filename);
00315
00316                 if (existeMatriz(filename, matrizes))
00317                 {
00318                     std::cout << "Já existe uma matriz com esse nome, tente outro nome para salvar" <<
std::endl;
00319                     break;
00320                 }
00321
00322                 matrizes.insert(std::make_pair(filename, matriz));
00323
00324                 std::cout << "Matriz salva com sucesso" << std::endl;
00325                 return;
00326             }
00327
00328             default:
00329                 std::cout << "Opção inválida" << std::endl;
00330                 break;
00331         }
00332     }
00333 }
```

5.11 main.cpp

[Ir para a documentação desse arquivo.](#)


```

00001
00007 #include <fstream>
00008 #include <unordered_map>
00009 #include "matriz/Matriz.hpp"
00010 #include "utils/utils.hpp"
00011
00012 using string = std::string;
00013 using unordered_map = std::unordered_map<string, Matriz>;
00014
00015 enum Opcoes // Enumeração para as opções do menu
00016 {
00017     LER_MATRIZ = 1,
00018     IMPRIMIR_MATRIZ,
00019     SOMAR_MATRIZES,
00020     MULTIPLICAR_MATRIZES,
00021     SAIR
00022 };
00023
00046 void readMatrix(Matriz &matriz, const std::string filename);
00047
00061 bool existeMatriz(const std::string filename, const unordered_map &matrizes);
00062
00078 void salvarMatriz(const Matriz &matriz, unordered_map &matrizes);
00079
00086 void escolherMatrizes(string &filename, string &filename2);
00087
00097 void printMatrizes(const unordered_map &matrizes);
00098
00115 int main()
00116 {
00117     // Configura a localização para suportar caracteres especiais em português
00118     setlocale(LC_ALL, "pt_BR.UTF-8");
00119
00120     std::cout << "Bem-vindo ao programa de manipulação de matrizes esparsas" << std::endl;
00121     std::cout << "-----" << std::endl;
00122
00123     // Mapa associativo para armazenar as matrizes
00124     unordered_map matrizes;
00125
00126     while (true)
00127     {
00128         // Exibe o menu de opções
00129         std::cout << "Escolha uma opção:" << std::endl;
00130         std::cout << "[1] - Ler Matriz" << std::endl;
00131         std::cout << "[2] - Imprimir Matriz" << std::endl;
00132         std::cout << "[3] - Somar Matrizes" << std::endl;
00133         std::cout << "[4] - Multiplicar Matrizes" << std::endl;
00134         std::cout << "[5] - Sair" << std::endl;
00135         int opcao;
00136         std::cin >> opcao;
00137         std::cin.ignore();
00138
00139         switch (opcao)
00140         {
00141             case LER_MATRIZ:
00142             {
00143                 // Lê uma matriz a partir de um arquivo
00144                 string filename;
00145                 std::cout << "Digite o nome do arquivo: ";
00146                 std::getline(std::cin, filename);
00147
00148                 Matriz matriz;
00149
00150                 try
00151                 {
00152                     readMatrix(matriz, filename);
00153                 }
00154                 catch (const std::exception &e)
00155                 {
00156                     std::cerr << e.what() << '\n';
00157                     break;
00158                 }
00159
00160                 // Armazena a matriz no mapa associativo
00161                 matrizes.insert(std::make_pair(filename, matriz));
00162                 break;
00163             }
00164
00165             case IMPRIMIR_MATRIZ:
00166             {
00167                 // Imprime uma matriz armazenada
00168                 std::cout << "Qual matriz deseja imprimir?" << std::endl;
00169                 printMatrizes(matrizes);
00170                 std::cout << "Coloque o nome do arquivo que deseja imprimir: ";
00171                 string filename;
00172                 std::getline(std::cin, filename);
00173

```

```

00174         if (existeMatriz(filename, matrizes))
00175             matrizes[filename].print();
00176         else
00177             std::cout << "Matriz não encontrada" << std::endl;
00178
00179         break;
00180     }
00181
00182     case SOMAR_MATRIZES:
00183     {
00184         // Soma duas matrizes armazenadas
00185         std::cout << "Quais as matrizes irá usar para somar" << std::endl;
00186         printMatrizes(matrizes);
00187         std::string filename, filename2;
00188         escolherMatrizes(filename, filename2);
00189
00190         if (!existeMatriz(filename, matrizes) || !existeMatriz(filename2, matrizes))
00191         {
00192             std::cout << "Alguma matriz não foi encontrada" << std::endl;
00193             break;
00194         }
00195
00196         Matriz matriz;
00197
00198         try
00199         {
00200             matriz = sum(matrizes[filename], matrizes[filename2]);
00201         }
00202         catch (const std::exception &e)
00203         {
00204             std::cerr << e.what() << '\n';
00205             break;
00206         }
00207
00208         matriz.print();
00209         salvarMatriz(matriz, matrizes);
00210         break;
00211     }
00212
00213     case MULTIPLICAR_MATRIZES:
00214     {
00215         // Multiplica duas matrizes armazenadas
00216         std::cout << "Quais as matrizes irá usar para multiplicar" << std::endl;
00217         printMatrizes(matrizes);
00218         std::string filename, filename2;
00219         escolherMatrizes(filename, filename2);
00220
00221         if (!existeMatriz(filename, matrizes) || !existeMatriz(filename2, matrizes))
00222         {
00223             std::cout << "Alguma matriz não foi encontrada" << std::endl;
00224             break;
00225         }
00226
00227         Matriz matriz;
00228
00229         try
00230         {
00231             matriz = multiply(matrizes[filename], matrizes[filename2]);
00232         }
00233         catch (const std::exception &e)
00234         {
00235             std::cerr << e.what() << '\n';
00236             break;
00237         }
00238
00239         matriz.print();
00240         salvarMatriz(matriz, matrizes);
00241         break;
00242     }
00243
00244     case SAIR:
00245     {
00246         // Encerra o programa
00247         std::cout << "Saindo..." << std::endl;
00248         return 0;
00249     }
00250
00251     default:
00252     {
00253         // Opção inválida
00254         std::cout << "Opção inválida" << std::endl;
00255         std::cin.clear();
00256         std::cin.ignore();
00257         break;
00258     }
00259 }
00260

```

```

00261         // Pausa e limpa a tela
00262         std::system("pause || read -p 'Pressione enter para continuar...' var");
00263         std::system("cls||clear");
00264     }
00265 }
00266
00267 void readMatrix(Matriz &matriz, const std::string filename)
00268 {
00269     std::ifstream file("src/arquivos/" + filename);
00270
00271     if (!file || !file.is_open())
00272         throw std::runtime_error("Erro ao abrir o arquivo");
00273
00274     int linhas{0}, colunas{0};
00275     file » linhas » colunas;
00276
00277     matriz = Matriz(linhas, colunas);
00278
00279     int i{0}, j{0};
00280     double valor{0.0f};
00281
00282     while (file » i » j » valor)
00283     {
00284         matriz.insert(i, j, valor);
00285     }
00286
00287     file.close();
00288 }
00289
00290 bool existeMatriz(const std::string filename, const unordered_map &matrizes)
00291 {
00292     return matrizes.find(filename) != matrizes.end();
00293 }
00294
00295 void salvarMatriz(const Matriz &matriz, unordered_map &matrizes)
00296 {
00297     while (true)
00298     {
00299         std::cout << "Deseja salvar a matriz? [S/N]: ";
00300         char resposta;
00301         std::cin » resposta;
00302         std::cin.ignore();
00303
00304         switch (tolower(resposta))
00305         {
00306             case 'n':
00307                 return;
00308
00309             case 's':
00310             {
00311                 std::cout << "Digite o nome que deseja salvar a matriz: ";
00312                 string filename;
00313
00314                 std::getline(std::cin, filename);
00315
00316                 if (existeMatriz(filename, matrizes))
00317                 {
00318                     std::cout << "Já existe uma matriz com esse nome, tente outro nome para salvar" <<
std::endl;
00319                     break;
00320                 }
00321
00322                 matrizes.insert(std::make_pair(filename, matriz));
00323
00324                 std::cout << "Matriz salva com sucesso" << std::endl;
00325                 return;
00326             }
00327
00328             default:
00329                 std::cout << "Opção inválida" << std::endl;
00330                 break;
00331         }
00332     }
00333 }
00334
00335 void escolherMatrizes(string &filename, string &filename2)
00336 {
00337     std::cout << "Coloque o nome da primeira matriz: ";
00338     std::getline(std::cin, filename);
00339
00340     std::cout << "Coloque o nome da segunda matriz: ";
00341     std::getline(std::cin, filename2);
00342 }
00343
00344 void printMatrizes(const unordered_map &matrizes)
00345 {
00346     for (const auto &par : matrizes)

```

```

00347     {
00348         std::cout << "-----" << std::endl;
00349         std::cout << par.first << " |" << par.second.getLinhas() << " x " << par.second.getColunas() << "|"
00350         << std::endl;
00351         std::cout << "-----" << std::endl;
00352     }

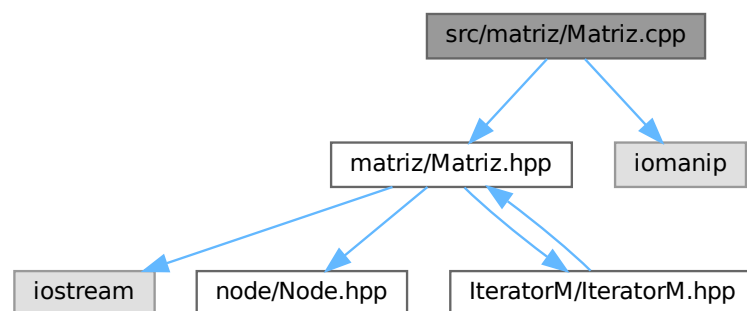
```

5.12 Referência do Arquivo src/matriz/Matriz.cpp

```
#include "matriz/Matriz.hpp"
```

```
#include <iomanip>
```

Gráfico de dependência de inclusões para Matriz.cpp:



5.13 Matriz.cpp

[Ir para a documentação desse arquivo.](#)

```

00001 #include "matriz/Matriz.hpp"
00002 #include <iomanip>
00003
00004 Matriz::Matriz() : cabecalho(new Node(0, 0, 0)), linhas(0), colunas(0)
00005 {
00006     cabecalho->direita = cabecalho->abaixo = cabecalho;
00007 }
00008
00009 Matriz::Matriz(const int &lin, const int &col)
00010 {
00011     if (lin <= 0 || col <= 0)
00012         throw std::invalid_argument("Erro: Tamanho de matriz inválido, insira valores maiores que 0");
00013     linhas = lin;
00014     colunas = col;
00015
00016     cabecalho = new Node(0, 0, 0);
00017     cabecalho->direita = cabecalho->abaixo = cabecalho;
00018
00019     Node *auxLinha = cabecalho;
00020     for (int i = 1; i <= lin; i++)
00021     {
00022         Node *novo = new Node(i, 0, 0);
00023         auxLinha->abaixo = novo;
00024         novo->direita = novo;
00025         auxLinha = novo;
00026     }
00027     auxLinha->abaixo = cabecalho;
00028
00029     Node *auxColuna = cabecalho;

```

```

00032     for (int j = 1; j <= col; j++)
00033     {
00034         Node *novo = new Node(0, j, 0);
00035         auxColuna->direita = novo;
00036         novo->abaixo = novo;
00037         auxColuna = novo;
00038     }
00039     auxColuna->direita = cabecalho;
00040 }
00041
00042 IteratorM Matriz::begin()
00043 {
00044     return IteratorM(cabecalho->abaixo, cabecalho->abaixo->direita);
00045 }
00046
00047 IteratorM Matriz::end()
00048 {
00049     return IteratorM(cabecalho, cabecalho->direita);
00050 }
00051
00052 IteratorM Matriz::begin() const
00053 {
00054     return IteratorM(cabecalho->abaixo, cabecalho->abaixo->direita);
00055 }
00056
00057 IteratorM Matriz::end() const
00058 {
00059     return IteratorM(cabecalho, cabecalho->direita);
00060 }
00061
00062 int Matriz::getLinhas() const
00063 {
00064     return linhas;
00065 }
00066
00067 int Matriz::getColunas() const
00068 {
00069     return colunas;
00070 }
00071
00072 Matriz::~Matriz()
00073 {
00074     limpar();
00075 }
00076
00077 void Matriz::limpar()
00078 {
00079     Node *linhaAtual = cabecalho->abaixo;
00080     Node *colunaAtual = linhaAtual->direita;
00081     Node *aux = cabecalho->abaixo;
00082
00083     while (linhaAtual != aux)
00084     {
00085         while (colunaAtual != linhaAtual)
00086         {
00087             Node *temp = colunaAtual;
00088             colunaAtual = colunaAtual->direita;
00089             delete temp;
00090         }
00091         Node *temp = linhaAtual;
00092         linhaAtual = linhaAtual->abaixo;
00093         delete temp;
00094     }
00095 }
00096
00097 void Matriz::insert(const int &posI, const int &posJ, const double &value)
00098 {
00099     // Não armazena valores iguais a zero
00100     if (value == 0)
00101         return;
00102
00103     // Validação dos índices
00104     if (posI <= 0 || posI > linhas || posJ <= 0 || posJ > colunas)
00105         throw std::invalid_argument("Erro: Local de inserção inválido");
00106
00107     // Busca na lista horizontal (linha)
00108     Node *linhaAtual = cabecalho;
00109     while (linhaAtual->linha < posI)
00110     {
00111         linhaAtual = linhaAtual->abaixo;
00112     }
00113
00114     // Procura pela posição correta na linha
00115     Node *aux = linhaAtual;
00116     while (aux->direita != linhaAtual && aux->direita->coluna < posJ)
00117     {
00118         aux = aux->direita;

```

```

00119     }
00120
00121     // Verifica se o nó já existe
00122     if (aux->direita->coluna == posJ)
00123     {
00124         aux->direita->atualizaValor(value); // Atualiza o valor
00125         return;
00126     }
00127
00128     // Cria um novo nó
00129     Node *novo = new Node(posI, posJ, value);
00130
00131     // Insere o nó na lista horizontal
00132     novo->direita = aux->direita;
00133     aux->direita = novo;
00134
00135     // Busca na lista vertical (coluna)
00136     Node *colunaAtual = cabecalho;
00137     while (colunaAtual->coluna < posJ)
00138     {
00139         colunaAtual = colunaAtual->direita;
00140     }
00141
00142     // Procura pela posição correta na coluna
00143     aux = colunaAtual;
00144     while (aux->abaixo != colunaAtual && aux->abaixo->linha < posI)
00145     {
00146         aux = aux->abaixo;
00147     }
00148
00149     // Insere o nó na lista vertical
00150     novo->abaixo = aux->abaixo;
00151     aux->abaixo = novo;
00152 }
00153
00154 double Matriz::get(const int &posI, const int &posJ)
00155 {
00156     if (posI <= 0 || posI > linhas || posJ <= 0 || posJ > colunas)
00157         throw std::invalid_argument("Erro: Local de acesso inválido");
00158
00159     IteratorM it = begin();
00160
00161     while (it != end())
00162     {
00163         if (it.current->linha == posI && it.current->coluna == posJ)
00164             return *it;
00165
00166         if (it.current->linha > posI || (it.current->linha == posI && it.current->coluna > posJ))
00167             return 0;
00168
00169         ++it;
00170     }
00171
00172     return 0;
00173 }
00174
00175 double Matriz::get(const int &posI, const int &posJ) const
00176 {
00177     if (posI <= 0 || posI > linhas || posJ <= 0 || posJ > colunas)
00178         throw std::invalid_argument("Erro: Local de acesso inválido");
00179
00180     IteratorM it = begin();
00181
00182     while (it != end())
00183     {
00184         if (it.current->linha == posI && it.current->coluna == posJ)
00185             return *it;
00186
00187         if (it.current->linha > posI || (it.current->linha == posI && it.current->coluna > posJ))
00188             return 0;
00189
00190         ++it;
00191     }
00192
00193     return 0;
00194 }
00195
00196 void Matriz::print()
00197 {
00198     IteratorM it = begin();
00199
00200     for (int i = 1; i <= linhas; i++)
00201     {
00202         for (int j = 1; j <= colunas; j++)
00203         {
00204             if (it.current->linha == i && it.current->coluna == j)
00205                 {

```

```

00206         std::cout << std::fixed << std::setprecision(1) << *it;
00207         ++it;
00208     }
00209     else
00210     {
00211         std::cout << "0.0";
00212     }
00213     std::cout << " ";
00214 }
00215 }
00216     std::cout << std::endl;
00217 }
00218 }

```

5.14 Referência do Arquivo tests/TestMatriz.cpp

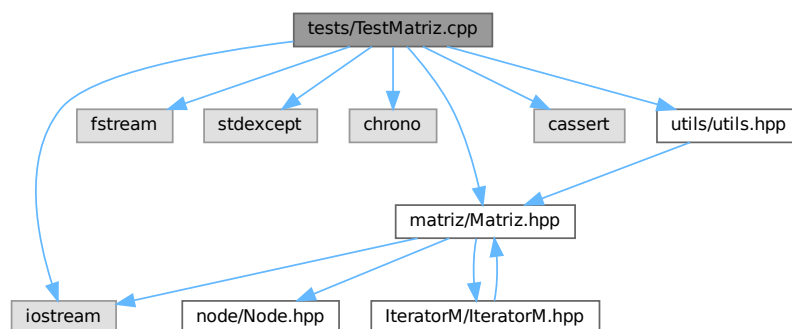
Arquivo de teste para operações com matrizes esparsas.

```

#include <iostream>
#include <fstream>
#include <stdexcept>
#include <chrono>
#include "matriz/Matriz.hpp"
#include <cassert>
#include "utils/utils.hpp"

```

Gráfico de dependência de inclusões para TestMatriz.cpp:



Funções

- `void testeInsercao ()`
- `void testeSoma (const Matriz &A, const Matriz &B, const Matriz &soma)`
- `void testeMultiplicacao (const Matriz &A, const Matriz &B, const Matriz &soma)`
- `Matriz leitura (const std::string &arquivo)`
- `bool arquivoExiste (const std::string &caminho)`
- `void testePerformance ()`
Testa a performance das operações de soma e multiplicação de matrizes.
- `int main ()`

5.14.1 Descrição detalhada

Arquivo de teste para operações com matrizes esparsas.

Este arquivo contém a função principal que realiza testes de operações com matrizes esparsas, incluindo leitura de arquivos, soma, multiplicação, inserção e performance.

Definição no arquivo [TestMatriz.cpp](#).

5.14.2 Funções

5.14.2.1 arquivoExiste()

```
bool arquivoExiste (
    const std::string & caminho )
```

Definição na linha 145 do arquivo [TestMatriz.cpp](#).

```
00146 {
00147     std::ifstream file(caminho);
00148     return file.good();
00149 }
```

5.14.2.2 leitura()

```
Matriz leitura (
    const std::string & arquivo )
```

Definição na linha 115 do arquivo [TestMatriz.cpp](#).

```
00116 {
00117     std::ifstream file("tests/arquivosTestes/" + arquivo); // Abre o arquivo
00118     if (!file.is_open())
00119     {
00120         throw std::runtime_error("Erro ao abrir o arquivo: " + arquivo);
00121     }
00122     int linhas, colunas;
00123     file >> linhas >> colunas;
00124     Matriz matriz(linhas, colunas);
00125     int linha, coluna;
00126     double valor;
00127     while (file >> linha >> coluna >> valor)
00128     {
00129         // Lê os valores da matriz
00130         matriz.insert(linha, coluna, valor); // Insere na matriz
00131     }
00132     file.close(); // Fecha o arquivo
00133     return matriz;
00134 }
00135
00136
00137 }
```


5.14.2.3 main()

```
int main ( )
```

Definição na linha 214 do arquivo [TestMatriz.cpp](#).

```
00215 {
00216     setlocale(LC_ALL, "pt_BR.UTF-8");
00217
00218     try
00219     {
00220         // Verificando se os arquivos existem
00221         if (!arquivoExiste("tests/arquivosTestes/Matrix1.txt"))
00222         {
00223             throw std::runtime_error("Arquivo Matrix1.txt não encontrado.");
00224         }
00225         if (!arquivoExiste("tests/arquivosTestes/Matrix2.txt"))
00226         {
00227             throw std::runtime_error("Arquivo Matrix2.txt não encontrado.");
00228         }
00229         if (!arquivoExiste("tests/arquivosTestes/MatrixResult.txt"))
00230         {
00231             throw std::runtime_error("Arquivo MatrixResult.txt não encontrado.");
00232         }
00233
00234         // Lê as matrizes de arquivos
00235         Matriz A = leitura("Matrix1.txt");
00236         Matriz B = leitura("Matrix2.txt");
00237         Matriz soma = leitura("MatrixResult.txt");
00238         Matriz multi = leitura("MatrixMulti.txt");
00239         std::cout << "Matrizes lidas com sucesso" << std::endl;
00240         // Executa os testes
00241         testeSoma(A, B, soma);
00242         testeMultiplicacao(A, B, multi);
00243         testeInsercao(); // Teste básico de inserção
00244         testePerformance(); // Teste de performance para matrizes grandes
00245     }
00246     catch (const std::runtime_error &e)
00247     {
00248         std::cerr << "Erro main: " << e.what() << std::endl;
00249     }
00250
00251     return 0;
00252 }
```

5.14.2.4 testeInsercao()

```
void testeInsercao ( )
```

Definição na linha 18 do arquivo [TestMatriz.cpp](#).

```
00019 {
00020     Matriz matriz(5, 5);
00021     matriz.insert(1, 1, 1);
00022     assert(matriz.get(1, 1) == 1); // Garantir que o valor foi inserido corretamente
00023     matriz.insert(5, 5, 2);
00024     assert(matriz.get(5, 5) == 2); // Garantir que o valor foi inserido corretamente
00025     matriz.insert(3, 2, 2);
00026     assert(matriz.get(3, 2) == 2); // Garantir que o valor foi inserido corretamente
00027     std::cout << "Teste de inserção passou" << std::endl;
00028 }
```

5.14.2.5 testeMultiplicacao()

```
void testeMultiplicacao (
    const Matriz & A,
    const Matriz & B,
    const Matriz & soma )
```

Definição na linha 82 do arquivo [TestMatriz.cpp](#).

```
00083 {
00084     Matriz D = multiply(A, B);
00085
00086     // Verificando as dimensões
```

```

00087     if (D.getColunas() != soma.getColunas() || D.getLinhas() != soma.getLinhas())
00088     {
00089         throw std::runtime_error("Erro: Dimensões incorretas na multiplicação das matrizes.");
00090     }
00091
00092     // Verificando cada valor da matriz resultante
00093     for (int i = 1; i <= soma.getLinhas(); i++)
00094     {
00095         for (int j = 1; j <= soma.getColunas(); j++)
00096         {
00097             if (D.get(i, j) != soma.get(i, j))
00098             {
00099                 throw std::runtime_error("Erro na multiplicação das matrizes: Valor incorreto na
posição (" +
00100                                     std::to_string(i) + ", " + std::to_string(j) + "). Esperado:
" +
00101                                     std::to_string(soma.get(i, j)) + ", Obtido: " +
00102                                     std::to_string(D.get(i, j)));
00103             }
00104         }
00105     }
00106     std::cout << "Teste de multiplicação passou" << std::endl;
00107 }

```

5.14.2.6 testePerformance()

```
void testePerformance ( )
```

Testa a performance das operações de soma e multiplicação de matrizes.

Esta função cria duas matrizes A e B de tamanho 100x100, preenche-as com valores específicos, e então mede o tempo necessário para realizar a soma e a multiplicação dessas matrizes.

A matriz A é preenchida com valores onde cada elemento é a soma dos índices de linha e coluna ($i + j$). A matriz B é preenchida com valores onde cada elemento é a diferença dos índices de linha e coluna ($i - j$).

A função utiliza a biblioteca `<chrono>` para medir o tempo de execução das operações de soma e multiplicação.

Passos executados pela função:

1. Cria duas matrizes A e B de tamanho 100x100.
2. Preenche as matrizes A e B com valores específicos.
3. Exibe uma mensagem indicando que as matrizes foram preenchidas com sucesso.
4. Mede o tempo necessário para somar as matrizes A e B.
5. Exibe o tempo de execução da soma.
6. Mede o tempo necessário para multiplicar as matrizes A e B.
7. Exibe o tempo de execução da multiplicação.

Retorna

void

Definição na linha 174 do arquivo TestMatriz.cpp.

```

00175 {
00176     Matriz A(100, 100);
00177     Matriz B(100, 100);
00178
00179     // Preenche as matrizes com valores
00180     for (int i = 1; i <= A.getLinhas(); ++i)
00181     {
00182         for (int j = 1; j <= A.getColunas(); ++j)
00183         {
00184             A.insert(i, j, i + j);
00185             B.insert(i, j, i - j);
00186         }
00187     }
00188     std::cout << "Matrizes preenchidas com sucesso" << std::endl;
00189
00190     std::cout << "Iniciando teste de performance" << std::endl;
00191
00192     auto inicio = std::chrono::high_resolution_clock::now();
00193     Matriz soma = sum(A, B); // Soma as matrizes
00194     auto fim = std::chrono::high_resolution_clock::now();
00195
00196     auto duracao = std::chrono::duration_cast<std::chrono::milliseconds>(fim - inicio);
00197     std::cout << "Tempo para Soma: " << duracao.count() << "ms" << std::endl;
00198
00199     auto inicio2 = std::chrono::high_resolution_clock::now();
00200     Matriz multi = multiply(A, B); // Soma as matrizes
00201     auto fim2 = std::chrono::high_resolution_clock::now();
00202
00203     auto duracao2 = std::chrono::duration_cast<std::chrono::milliseconds>(fim2 - inicio2);
00204     std::cout << "Tempo para Multiplicação: " << duracao2.count() << "ms" << std::endl;
00205 }

```

5.14.2.7 testeSoma()

```

void testeSoma (
    const Matriz & A,
    const Matriz & B,
    const Matriz & soma )

```

Definição na linha 42 do arquivo TestMatriz.cpp.

```

00043 {
00044
00045     Matriz D = sum(A, B);
00046
00047     // Verificando as dimensões
00048     if (D.getColunas() != soma.getColunas() || D.getLinhas() != soma.getLinhas())
00049     {
00050         throw std::runtime_error("Erro: Dimensões incorretas na soma das matrizes.");
00051     }
00052
00053     // Verificando cada valor da matriz resultante
00054     for (int i = 1; i <= soma.getLinhas(); i++)
00055     {
00056         for (int j = 1; j <= soma.getColunas(); j++)
00057         {
00058             if (D.get(i, j) != soma.get(i, j))
00059             {
00060                 throw std::runtime_error("Erro na soma das matrizes: Valor incorreto na posição (" +
00061                     std::to_string(i) + ", " + std::to_string(j) + "). Esperado:
00062                     " + std::to_string(soma.get(i, j)) + ", Obtido: " +
00063                     std::to_string(D.get(i, j)));
00064             }
00065         }
00066     }
00067     std::cout << "Teste de soma passou" << std::endl;
00068 }

```

5.15 TestMatriz.cpp

[Ir para a documentação desse arquivo.](#)

```
00001 #include <iostream>
00002 #include <fstream>
00003 #include <stdexcept>
00004 #include <chrono>
00005 #include "matriz/Matriz.hpp"
00006 #include <cassert>
00007 #include "utils/utils.hpp"
00008
00009 /*
00010 *   @brief Função de teste de inserção de valores na matriz.
00011 *
00012 *   Esta função cria uma matriz 5x5 e insere valores em posições específicas.
00013 *   Em seguida, a função verifica se os valores foram inseridos corretamente.
00014 *   Se algum valor não for inserido corretamente, a função lança uma exceção.
00015 *
00016 *   @throw std::runtime_error Se algum valor não for inserido corretamente.
00017 */
00018 void testeInsercao()
00019 {
00020     Matriz matriz(5, 5);
00021     matriz.insert(1, 1, 1);
00022     assert(matriz.get(1, 1) == 1); // Garantir que o valor foi inserido corretamente
00023     matriz.insert(5, 5, 2);
00024     assert(matriz.get(5, 5) == 2); // Garantir que o valor foi inserido corretamente
00025     matriz.insert(3, 2, 2);
00026     assert(matriz.get(3, 2) == 2); // Garantir que o valor foi inserido corretamente
00027     std::cout << "Teste de inserção passou" << std::endl;
00028 }
00029
00030 /*
00031 *   @brief Função de teste de soma de matrizes.
00032 *
00033 *   Esta função recebe duas matrizes A e B e a matriz resultante da soma dessas matrizes.
00034 *   A função calcula a soma das matrizes A e B e compara com a matriz resultante esperada.
00035 *   Se a matriz resultante da soma for diferente da matriz esperada, a função lança uma exceção.
00036 *
00037 *   @param A Matriz A.
00038 *   @param B Matriz B.
00039 *
00040 *   @throw std::runtime_error Se a matriz resultante da soma for diferente da matriz esperada.
00041 */
00042 void testeSoma(const Matriz &A, const Matriz &B, const Matriz &soma)
00043 {
00044     Matriz D = sum(A, B);
00045
00046     // Verificando as dimensões
00047     if (D.getColunas() != soma.getColunas() || D.getLinhas() != soma.getLinhas())
00048     {
00049         throw std::runtime_error("Erro: Dimensões incorretas na soma das matrizes.");
00050     }
00051
00052     // Verificando cada valor da matriz resultante
00053     for (int i = 1; i <= soma.getLinhas(); i++)
00054     {
00055         for (int j = 1; j <= soma.getColunas(); j++)
00056         {
00057             if (D.get(i, j) != soma.get(i, j))
00058             {
00059                 throw std::runtime_error("Erro na soma das matrizes: Valor incorreto na posição (" +
00060                     std::to_string(i) + ", " + std::to_string(j) + "). Esperado: " +
00061                     " +
00062                     std::to_string(soma.get(i, j)) + ", Obtido: " +
00063                     std::to_string(D.get(i, j)));
00064             }
00065         }
00066     }
00067     std::cout << "Teste de soma passou" << std::endl;
00068 }
00069
00070 /*
00071 *   @brief Função de teste de multiplicação de matrizes.
00072 *
00073 *   Esta função recebe duas matrizes A e B e a matriz resultante da multiplicação dessas matrizes.
00074 *   A função calcula a multiplicação das matrizes A e B e compara com a matriz resultante esperada.
00075 *   Se a matriz resultante da multiplicação for diferente da matriz esperada, a função lança uma
00076 *   exceção.
00077 *
00078 *   @param A Matriz A.
00079 *   @param B Matriz B.
00080 */
```

```

00080 * @throw std::runtime_error Se a matriz resultante da multiplicação for diferente da matriz
      esperada.
00081 */
00082 void testeMultiplicacao(const Matriz &A, const Matriz &B, const Matriz &soma)
00083 {
00084     Matriz D = multiply(A, B);
00085
00086     // Verificando as dimensões
00087     if (D.getColunas() != soma.getColunas() || D.getLinhas() != soma.getLinhas())
00088     {
00089         throw std::runtime_error("Erro: Dimensões incorretas na multiplicação das matrizes.");
00090     }
00091
00092     // Verificando cada valor da matriz resultante
00093     for (int i = 1; i <= soma.getLinhas(); i++)
00094     {
00095         for (int j = 1; j <= soma.getColunas(); j++)
00096         {
00097             if (D.get(i, j) != soma.get(i, j))
00098             {
00099                 throw std::runtime_error("Erro na multiplicação das matrizes: Valor incorreto na
posição (" +
00100                                     std::to_string(i) + ", " + std::to_string(j) + "). Esperado:
" +
00101                                     std::to_string(soma.get(i, j)) + ", Obtido: " +
std::to_string(D.get(i, j)));
00102             }
00103         }
00104     }
00105
00106     std::cout << "Teste de multiplicação passou" << std::endl;
00107 }
00108
00109 /*
00110 * @brief Função para ler uma matriz de um arquivo.
00111 *
00112 * @param arquivo Nome do arquivo a ser lido.
00113 * @return Matriz Matriz lida do arquivo.
00114 */
00115 Matriz leitura(const std::string &arquivo)
00116 {
00117     std::ifstream file("tests/arquivosTestes/" + arquivo); // Abre o arquivo
00118     if (!file.is_open())
00119     {
00120         throw std::runtime_error("Erro ao abrir o arquivo: " + arquivo);
00121     }
00122
00123     int linhas, colunas;
00124     file >> linhas >> colunas;
00125
00126     Matriz matriz(linhas, colunas);
00127
00128     int linha, coluna;
00129     double valor;
00130     while (file >> linha >> coluna >> valor)
00131     {
00132         // Lê os valores da matriz
00133         matriz.insert(linha, coluna, valor); // Insere na matriz
00134     }
00135
00136     file.close(); // Fecha o arquivo
00137     return matriz;
00138 }
00139
00140 /*
00141 * @brief função para verificar se um arquivo existe
00142 *
00143 * @param caminho Caminho do arquivo a ser verificado.
00144 * @return bool Verdadeiro se o arquivo existir, falso caso contrário.
00145 */
00146 bool arquivoExiste(const std::string &caminho)
00147 {
00148     std::ifstream file(caminho);
00149     return file.good();
00150 }
00151
00152 void testePerformance()
00153 {
00154     Matriz A(100, 100);
00155     Matriz B(100, 100);
00156
00157     // Preenche as matrizes com valores
00158     for (int i = 1; i <= A.getLinhas(); ++i)
00159     {
00160         for (int j = 1; j <= A.getColunas(); ++j)
00161         {
00162             A.insert(i, j, i + j);
00163             B.insert(i, j, i - j);
00164         }
00165     }

```

```

00186     }
00187 }
00188 std::cout << "Matrizes preenchidas com sucesso" << std::endl;
00189
00190 std::cout << "Iniciando teste de performance" << std::endl;
00191
00192 auto inicio = std::chrono::high_resolution_clock::now();
00193 Matriz soma = sum(A, B); // Soma as matrizes
00194 auto fim = std::chrono::high_resolution_clock::now();
00195
00196 auto duracao = std::chrono::duration_cast<std::chrono::milliseconds>(fim - inicio);
00197 std::cout << "Tempo para Soma: " << duracao.count() << "ms" << std::endl;
00198
00199 auto inicio2 = std::chrono::high_resolution_clock::now();
00200 Matriz multi = multiply(A, B); // Soma as matrizes
00201 auto fim2 = std::chrono::high_resolution_clock::now();
00202
00203 auto duracao2 = std::chrono::duration_cast<std::chrono::milliseconds>(fim2 - inicio2);
00204 std::cout << "Tempo para Multiplicação: " << duracao2.count() << "ms" << std::endl;
00205 }
00206
00214 int main()
00215 {
00216     setlocale(LC_ALL, "pt_BR.UTF-8");
00217
00218     try
00219     {
00220         // Verificando se os arquivos existem
00221         if (!arquivoExiste("tests/arquivosTestes/Matrix1.txt"))
00222         {
00223             throw std::runtime_error("Arquivo Matrix1.txt não encontrado.");
00224         }
00225         if (!arquivoExiste("tests/arquivosTestes/Matrix2.txt"))
00226         {
00227             throw std::runtime_error("Arquivo Matrix2.txt não encontrado.");
00228         }
00229         if (!arquivoExiste("tests/arquivosTestes/MatrixResult.txt"))
00230         {
00231             throw std::runtime_error("Arquivo MatrixResult.txt não encontrado.");
00232         }
00233
00234         // Lê as matrizes de arquivos
00235         Matriz A = leitura("Matrix1.txt");
00236         Matriz B = leitura("Matrix2.txt");
00237         Matriz soma = leitura("MatrixResult.txt");
00238         Matriz multi = leitura("MatrixMulti.txt");
00239         std::cout << "Matrizes lidas com sucesso" << std::endl;
00240         // Executa os testes
00241         testeSoma(A, B, soma);
00242         testeMultiplicacao(A, B, multi);
00243         testeInsercao(); // Teste básico de inserção
00244         testePerformance(); // Teste de performance para matrizes grandes
00245     }
00246     catch (const std::runtime_error &e)
00247     {
00248         std::cerr << "Erro main: " << e.what() << std::endl;
00249     }
00250
00251     return 0;
00252 }

```

Índice Remissivo

- ~Matriz
 - Matriz, [17](#)
- abaixo
 - Node, [24](#)
- arquivoExiste
 - TestMatriz.cpp, [50](#)
- atualizaValor
 - Node, [24](#)
- begin
 - Matriz, [17](#)
- coluna
 - Node, [24](#)
- difference_type
 - IteratorM, [10](#)
- direita
 - Node, [24](#)
- end
 - Matriz, [17](#), [18](#)
- escolherMatrizes
 - main.cpp, [37](#)
- existeMatriz
 - main.cpp, [38](#)
- get
 - Matriz, [18](#), [19](#)
- getColunas
 - Matriz, [20](#)
- getLinhas
 - Matriz, [20](#)
- IMPRIMIR_MATRIZ
 - main.cpp, [37](#)
- include/IteratorM/IteratorM.hpp, [27](#), [28](#)
- include/matriz/Matriz.hpp, [29](#), [30](#)
- include/node/Node.hpp, [31](#)
- include/Utils/Utils.hpp, [32](#), [34](#)
- insert
 - Matriz, [20](#)
- iterator_category
 - IteratorM, [10](#)
- IteratorM, [9](#)
 - difference_type, [10](#)
 - iterator_category, [10](#)
 - IteratorM, [11](#)
 - Matriz, [14](#)
- operator!=, [11](#)
- operator++, [12](#)
- operator->, [12](#), [13](#)
- operator==, [13](#)
- operator*, [12](#)
- pointer, [10](#)
- reference, [10](#)
- value_type, [10](#)
- leitura
 - TestMatriz.cpp, [50](#)
- LER_MATRIZ
 - main.cpp, [37](#)
- limpar
 - Matriz, [21](#)
- linha
 - Node, [25](#)
- main
 - main.cpp, [38](#)
 - TestMatriz.cpp, [50](#)
- main.cpp
 - escolherMatrizes, [37](#)
 - existeMatriz, [38](#)
 - IMPRIMIR_MATRIZ, [37](#)
 - LER_MATRIZ, [37](#)
 - main, [38](#)
 - MULTIPLICAR_MATRIZES, [37](#)
 - Opcoes, [37](#)
 - printMatrizes, [40](#)
 - readMatrix, [41](#)
 - SAIR, [37](#)
 - salvarMatriz, [41](#)
 - SOMAR_MATRIZES, [37](#)
 - string, [37](#)
 - unordered_map, [37](#)
- Matriz, [14](#)
 - ~Matriz, [17](#)
 - begin, [17](#)
 - end, [17](#), [18](#)
 - get, [18](#), [19](#)
 - getColunas, [20](#)
 - getLinhas, [20](#)
 - insert, [20](#)
 - IteratorM, [14](#)
 - limpar, [21](#)
 - Matriz, [15](#)
 - print, [22](#)
- Matriz Esparsa, [1](#)
- MULTIPLICAR_MATRIZES

- main.cpp, 37
- multiply
 - utils.hpp, 32
- Node, 23
 - abaixo, 24
 - atualizaValor, 24
 - coluna, 24
 - direita, 24
 - linha, 25
 - Node, 23
 - valor, 25
- Opcoes
 - main.cpp, 37
- operator!=
 - IteratorM, 11
- operator++
 - IteratorM, 12
- operator->
 - IteratorM, 12, 13
- operator==
 - IteratorM, 13
- operator*
 - IteratorM, 12
- pointer
 - IteratorM, 10
- print
 - Matriz, 22
- printMatrizes
 - main.cpp, 40
- readMatrix
 - main.cpp, 41
- README.md, 35
- reference
 - IteratorM, 10
- SAIR
 - main.cpp, 37
- salvarMatriz
 - main.cpp, 41
- SOMAR_MATRIZES
 - main.cpp, 37
- src/main/main.cpp, 35, 42
- src/matriz/Matriz.cpp, 46
- string
 - main.cpp, 37
- sum
 - utils.hpp, 33
- testeInsercao
 - TestMatriz.cpp, 51
- testeMultiplicacao
 - TestMatriz.cpp, 51
- testePerformance
 - TestMatriz.cpp, 52
- testeSoma
 - TestMatriz.cpp, 53
- TestMatriz.cpp
 - arquivoExiste, 50
 - leitura, 50
 - main, 50
 - testeInsercao, 51
 - testeMultiplicacao, 51
 - testePerformance, 52
 - testeSoma, 53
- tests/TestMatriz.cpp, 49, 54
- unordered_map
 - main.cpp, 37
- utils.hpp
 - multiply, 32
 - sum, 33
- valor
 - Node, 25
- value_type
 - IteratorM, 10