Matriz Esparsa

2.2

Gerado por Doxygen 1.9.8

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:

- [x] Implementar inserção e remoção dinâmicas
- [ ] Otimizar a busca de elementos
- [x] 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

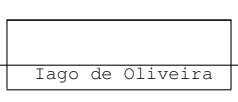
Para contribuir com a Matriz Esparsa:

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:

Capítulo 3

Índice dos Arquivos

3.1 Lista de Arquivos

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

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

cabecalho	Ponteiro para o nó de cabeçalho.
current	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

it	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

it	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 (const Matriz &outra)
  Construtor de cópia para a classe Matriz.
- ~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).
- Matriz operator= (Matriz matriz)
  Sobrecarga do operador de atribuição para a classe Matriz.
- int getLinhas () const
  Retorna a quantidade de linhas da matriz.
- int getColunas () const
  Retorna a quantidade de colunas da matriz.
- void limpar ()
  Limpa os dados armazenados na matriz esparsa.
- 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 em uma posição específica da matriz esparsa.
- 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() [1/3]

```
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() [2/3]

```
Matriz::Matriz (
            const int & ln,
            const int & cl )
```

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

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

Exceções

std::invalid_argument	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() [3/3]

```
Matriz::Matriz (
            const Matriz & outra )
```

Construtor de cópia para a classe Matriz.

Este construtor cria uma nova instância de Matriz a partir de outra instância existente, realizando uma cópia profunda dos elementos da matriz original. Inicialmente, ele chama o construtor principal da classe, passando como argumentos o número de linhas e colunas da matriz a ser copiada, garantindo que a nova matriz possua a mesma estrutura de armazenamento.

Após a alocação da estrutura adequada, o construtor itera por todos os elementos da matriz original ("outra") usando um iterador (IteratorM). Para cada elemento encontrado, o método insert é invocado para inserir o valor na nova matriz, mantendo a posição indicada pelos atributos "linha" e "coluna" presentes na estrutura apontada pelo iterador.

Parâmetros

outra	Referência para a instância da matriz que será copiada.

Definição na linha 42 do arquivo Matriz.cpp.

```
00042                                   : Matriz(outra.linhas, outra.colunas)
00043 {
00044     for (IteratorM it = outra.begin(); it != outra.end(); ++it)
00045         this->insert(it.current->linha, it.current->coluna, *it);
00046 }
```

4.2.2.4 ~Matriz()

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

Destrutor da classe Matriz.

Este destrutor é responsável por desalocar toda a memória alocada pela matriz esparsa, incluindo os nós de dados e os nós sentinela.

A operação é realizada em várias etapas para evitar vazamento de memória e acessos inválidos:

- Verifica se o nó cabeçalho é nulo. Se for, a matriz está vazia e não há nada a ser desalocado
- Inicia a partir do primeiro nó sentinela abaixo do cabeçalho e, enquanto não retorna para o cabeçalho, desaloca cada nó.
- Inicia a partir do primeiro nó sentinela à direita do cabeçalho e, seguindo até retornar ao cabeçalho, desaloca cada nó.
- Após remover todos os nós sentinela nas linhas e colunas, o cabeçalho é removido.
- O ponteiro do cabeçalho é então definido como nullptr para evitar acessos inválidos posteriores.

Observação

Este método garante que todos os nós foram corretamente liberados, evitando vazamentos de memória e a tentativa de desalocar a mesma memória mais de uma vez.

Definição na linha 88 do arquivo Matriz.cpp.

```
00089 {
00090     if (cabecalho == nullptr)
00091         return;
00092
00093     limpar();
00094
00095     Node *linhaAtual = cabecalho->abaixo;
00096     while (linhaAtual != cabecalho)
00097     {
00098         Node *proximoLinha = linhaAtual->abaixo;
00099         delete linhaAtual;
00100         linhaAtual = proximoLinha;
00101     }
00102
00103     Node *colunaAtual = cabecalho->direita;
00104     while (colunaAtual != cabecalho)
00105     {
00106         Node *proximoColuna = colunaAtual->direita;
00107         delete colunaAtual;
00108         colunaAtual = proximoColuna;
00109     }
00110
00111     delete cabecalho;
00112     cabecalho = nullptr;
00113 }
```

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 48 do arquivo Matriz.cpp.

```
00049 {
00050     return IteratorM(cabecalho->abaixo, cabecalho->abaixo->direita);
00051 }
```

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 58 do arquivo Matriz.cpp.

```
00059 {
00060     return IteratorM(cabecalho->abaixo, cabecalho->abaixo->direita);
00061 }
```

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 53 do arquivo Matriz.cpp.

```
00054 {
00055     return IteratorM(cabecalho, cabecalho->direita);
00056 }
```

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 63 do arquivo Matriz.cpp.

```
00064 {
00065     return IteratorM(cabecalho, cabecalho->direita);
00066 }
```

4.2.3.5 get() [1/2]

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

Retorna o valor armazenado em uma posição específica da matriz esparsa.

Esta função acessa o elemento na posição (posI, posJ) da matriz esparsa. Ela espera que os índices informados sigam a convenção onde o primeiro índice é 1 e o último corresponde ao número total de linhas ou colunas da matriz.

A função utiliza um iterador para percorrer a matriz, comparando as posições dos nós com os índices informados. Se o nó correspondente à posição (posI, posJ) for encontrado, a função retorna o valor armazenado nele. Caso contrário, a função retornará 0, indicando que não há valor armazenado na posição informada.

Parâmetros

posI	Constante que referencia o índice da linha desejada (deve estar no intervalo [1, linhas]).
posJ	Constante que referencia o índice da coluna desejada (deve estar no intervalo [1, colunas]).

Retorna

double O elemento contido na posição especificada, ou 0 caso não exista um nó correspondente.

Exceções

std::invalid_argument	Se posI ou posJ forem menores ou iguais a 0 ou excederem as dimensões da matriz.

Definição na linha 200 do arquivo Matriz.cpp.

```
00201 {
00202     if (posI <= 0 || posI > linhas || posJ <= 0 || posJ > colunas)
00203         throw std::invalid_argument("Erro: Local de acesso inválido");
00204
00205     IteratorM it = begin();
00206
00207     // Avança enquanto o nó atual estiver "antes" da posição desejada.
00208     while (it != end() &&
00209            (it.current->linha < posI ||
00210             (it.current->linha == posI && it.current->coluna < posJ)))
00211     {
00212         ++it;
00213     }
00214
00215     // Se o nó atual corresponde exatamente à posição, retorna o valor.
00216     if (it != end() && it.current->linha == posI && it.current->coluna == posJ)
00217         return *it;
00218
00219     return 0;
00220 }
```

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

posI	Índice da linha solicitada.
posJ	Índice da coluna solicitada.

Retorna

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

Exceções

std::invalid_argument	Se `posI` ou `posJ` forem menores ou iguais a 0 ou excederem as dimensões da matriz.

Definição na linha 222 do arquivo Matriz.cpp.

```
00223 {
00224     if (posI <= 0 || posI > linhas || posJ <= 0 || posJ > colunas)
00225         throw std::invalid_argument("Erro: Local de acesso inválido");
00226
00227     IteratorM it = begin();
00228
00229     // Avança enquanto o nó atual estiver "antes" da posição desejada.
00230     while (it != end() &&
00231            (it.current->linha < posI ||
00232             (it.current->linha == posI && it.current->coluna < posJ)))
00233     {
00234         ++it;
00235     }
00236
00237     // Se o nó atual corresponde exatamente à posição, retorna o valor.
00238     if (it != end() && it.current->linha == posI && it.current->coluna == posJ)
00239         return *it;
00240
00241     return 0;
00242 }
```

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 83 do arquivo Matriz.cpp.

```
00084 {
00085     return colunas;
00086 }
```

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 78 do arquivo Matriz.cpp.

```
00079 {
00080     return linhas;
00081 }
```

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

posI	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.
posJ	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.
value	Valor a ser armazenado na nova célula. Valores iguais a zero não são inseridos na matriz.

Exceções

std::invalid_argument	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 153 do arquivo Matriz.cpp.

```
00154 {
00155     if (value == 0)
```

```
00156         return;
00157
00158     if (posI <= 0 || posI > linhas || posJ <= 0 || posJ > colunas)
00159         throw std::invalid_argument("Erro: Local de inserção inválido");
00160
00161     Node *linhaAtual = cabecalho;
00162     while (linhaAtual->linha < posI)
00163     {
00164         linhaAtual = linhaAtual->abaixo;
00165     }
00166
00167     Node *aux = linhaAtual;
00168     while (aux->direita != linhaAtual && aux->direita->coluna < posJ)
00169     {
00170         aux = aux->direita;
00171     }
00172
00173     if (aux->direita->coluna == posJ)
00174     {
00175         aux->direita->atualizaValor(value);
00176         return;
00177     }
00178
00179     Node *novo = new Node(posI, posJ, value);
00180
00181     novo->direita = aux->direita;
00182     aux->direita = novo;
00183
00184     Node *colunaAtual = cabecalho;
00185     while (colunaAtual->coluna < posJ)
00186     {
00187         colunaAtual = colunaAtual->direita;
00188     }
00189
00190     aux = colunaAtual;
00191     while (aux->abaixo != colunaAtual && aux->abaixo->linha < posI)
00192     {
00193         aux = aux->abaixo;
00194     }
00195
00196     novo->abaixo = aux->abaixo;
00197     aux->abaixo = novo;
00198 }
```

4.2.3.10 limpar()

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

Limpa os dados armazenados na matriz esparsa.

Esta função remove todos os nós de dados presentes na matriz esparsa, liberando a memória alocada para estes nós.

Procedimento:

1. Verifica se o nó cabeçalho da matriz é nulo. Se for, a função retorna imediatamente, pois não há estrutura alocada para ser limpa.
2. Obtém o primeiro sentinela de linha a partir do cabeçalho (cabecalho->abaixo). Caso esse sentinela seja o próprio cabeçalho, a matriz está vazia, e a função retorna sem realizar nenhuma ação.
3. Quebra a circularidade vertical:
   - Percorre a lista de sentinelas de linha até encontrar a última linha (um nó cujo ponteiro "abaixo" aponta para o cabeçalho).
   - Ajusta o ponteiro "abaixo" do último nó para nullptr, transformando a lista circular em uma lista linear, o que facilita a iteração e remoção dos nós.
4. Para cada linha (sentinela) na lista linear:
   - Percorre a lista horizontal de nós de dados. Essa lista é circular, com os dados localizados entre o ponteiro "direita" do sentinela e o próprio sentinela.

- Exclui cada nó de dado da linha, armazenando o nó seguinte antes de chamar delete, de forma a não perder a referência.
- Após excluir os nós de dados, restaura o ponteiro "direita" do sentinela para que ele aponte para ele mesmo.

5. Restaura a circularidade vertical:
   - Após a limpeza, percorre novamente a lista linear de linhas até o último nó.
   - Reconfigura o ponteiro "abaixo" do último nó para apontar novamente ao cabeçalho da matriz.

Observação

- A função somente remove os nós de dados e restaura os ponteiros dos sentinelas, mantendo o nó cabeçalho intacto.
- A estrutura original da matriz esparsa é preservada, permitindo que a mesma seja reutilizada posteriormente.

Definição na linha 115 do arquivo Matriz.cpp.

```
00116 {
00117     if (cabecalho == nullptr)
00118         return;
00119
00120     Node *LinhaAtual = cabecalho->abaixo;
00121     if (LinhaAtual == cabecalho)
00122         return;
00123
00124     Node *ColunaAtual = LinhaAtual;
00125     while (ColunaAtual->abaixo != cabecalho)
00126     {
00127         ColunaAtual = ColunaAtual->abaixo;
00128     }
00129
00130     ColunaAtual->abaixo = nullptr;
00131
00132     for (Node *linha = LinhaAtual; linha != nullptr; linha = linha->abaixo)
00133     {
00134
00135         Node *atual = linha->direita;
00136         while (atual != linha)
00137         {
00138             Node *proximo = atual->direita;
00139             delete atual;
00140             atual = proximo;
00141         }
00142         linha->direita = linha;
00143     }
00144
00145     ColunaAtual = LinhaAtual;
00146     while (ColunaAtual->abaixo != nullptr)
00147     {
00148         ColunaAtual = ColunaAtual->abaixo;
00149     }
00150     ColunaAtual->abaixo = cabecalho;
00151 }
```

4.2.3.11 operator=()

```
Matriz Matriz::operator= (
            Matriz matriz )
```

Sobrecarga do operador de atribuição para a classe Matriz.

Esta função utiliza a técnica de cópia e troca (copy-and-swap) para implementar o operador de atribuição de forma segura e eficiente. Ao receber o objeto 'matriz' por valor, ele garante que todos os recursos sejam copiados e que, em seguida, uma troca (std::swap) seja realizada entre os atributos do objeto atual e os do objeto recebido. Assim, os dados antigos do objeto atual serão automaticamente liberados quando o objeto 'matriz' passado por valor for destruído, proporcionando uma forte garantia de exceção.

Parâmetros

matriz	Objeto do tipo Matriz que contém os novos dados a serem atribuídos. Por ser passado por valor, uma cópia dos dados é criada, permitindo que a operação de troca seja efetuada sem riscos de perda de recursos em caso de erros.

Retorna

Retorna o próprio objeto (∗this) já contendo os dados do objeto 'matriz'. O retorno por valor, nesse contexto, assegura que o objeto resultante possui os recursos de forma consistente.

- std::swap: São trocados os atributos 'cabecalho', 'linhas' e 'colunas' entre o objeto atual e o objeto recebido, o que permite que o estado do objeto atual seja atualizado com o novo conteúdo.
- Ao final da função, o objeto local 'matriz' é destruído, liberando os recursos que anteriormente pertenciam ao objeto atual, evitando assim possíveis vazamentos de memória.

Definição na linha 68 do arquivo Matriz.cpp.

```
00069 {
00070     // Troca os dados do objeto atual com os dados de 'matriz'
00071     std::swap(cabecalho, matriz.cabecalho);
00072     std::swap(linhas, matriz.linhas);
00073     std::swap(colunas, matriz.colunas);
00074     // 'matriz' é destruída, liberando os recursos antigos
00075     return *this;
00076 }
```

4.2.3.12 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 244 do arquivo Matriz.cpp.

```
00245 {
00246     IteratorM it = begin();
00247
00248     for (int i = 1; i <= linhas; i++)
00249     {
00250         for (int j = 1; j <= colunas; j++)
00251         {
00252             if (it.current->linha == i && it.current->coluna == j)
00253             {
00254                 std::cout << std::fixed << std::setprecision(1) << *it;
00255                 ++it;
00256             }
00257             else
00258             {
00259                 std::cout << "0.0";
00260             }
00261             std::cout << " ";
00262         }
00263         std::cout << std::endl;
00264     }
00265 }
```

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

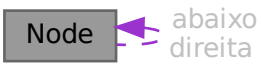
- include/matriz/Matriz.hpp
- src/matriz/Matriz.cpp

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

linha	Referência constante para o número da linha.
coluna	Referência constante para o número da coluna.
valor	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

novoValor	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

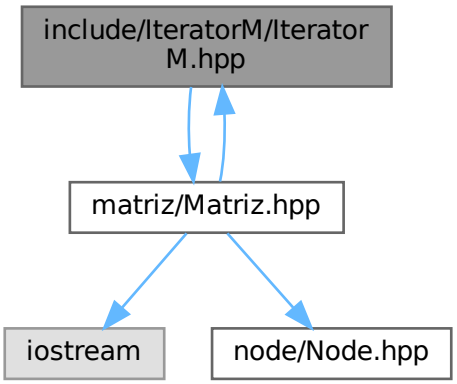
Capítulo 5

Arquivos

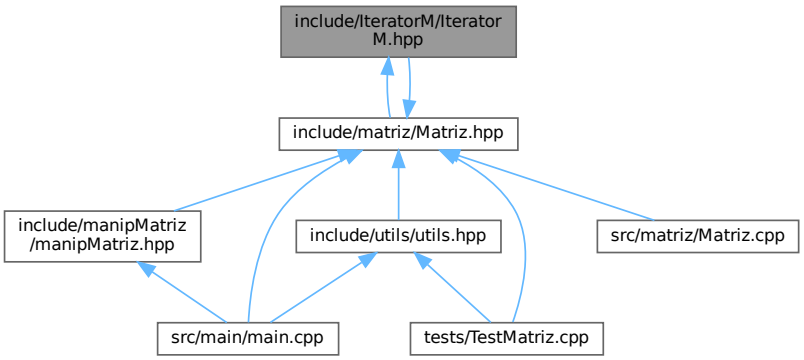
5.1 Referência do Arquivo include/IteratorM/IteratorM.hpp

```
#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.

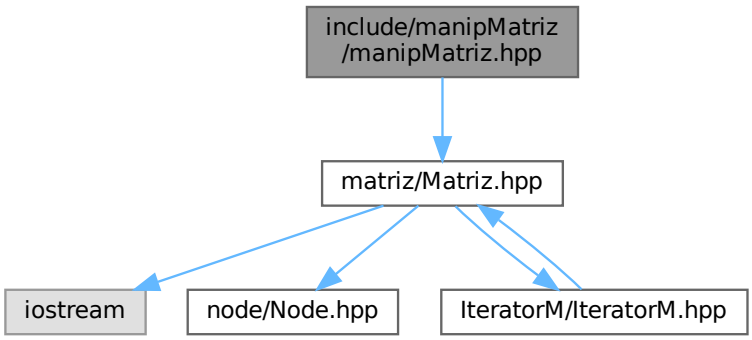
```
00001 #ifndef ITERATORM_HPP
00002 #define ITERATORM_HPP
00003
00004 #include "matriz/Matriz.hpp"
00005
00006 class Matriz;
00007
00016 class IteratorM
00017 {
00018     friend class Matriz;
00019
00020 private:
00021     Node *cabecalho;
00022     Node *current;
00024 public:
00025     using iterator_category = 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
```

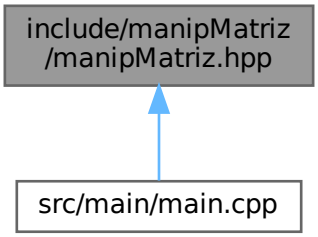
5.3 Referência do Arquivo include/manipMatriz/manipMatriz.hpp

Menu para as funções para manipulação de matrizes.

`#include "matriz/Matriz.hpp"`
Gráfico de dependência de inclusões para manipMatriz.hpp:



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



Funções

- void manipMatrix (Matriz &matriz, const std::string &nomeMatriz)

5.3.1 Descrição detalhada

Menu para as funções para manipulação de matrizes.

Parâmetros

matriz	Matriz a ser manipulada
nomeMatriz	Nome da matriz a ser manipulada

Definição no arquivo manipMatriz.hpp.

5.3.2 Funções

5.3.2.1 manipMatrix()

```
void manipMatrix (
            Matriz & matriz,
            const std::string & nomeMatriz )
```

Definição na linha 13 do arquivo manipMatriz.hpp.

```
00014 {
00015     while (true)
00016     {
00017         std::cout « "Matriz selecionada: " « nomeMatriz « std::endl;
00018         matriz.print();
00019         std::cout « "Escolha uma opção:" « std::endl;
00020         std::cout « "[1] - Inserir Valor" « std::endl;
00021         std::cout « "[2] - Limpar Matriz" « std::endl;
00022         std::cout « "[3] - Voltar" « std::endl;
00023 
00024         int opcao;
00025         std::cin » opcao;
00026         std::cin.ignore();
00027 
```

```
00028         switch (opcao)
00029         {
00030         case 1:
00031         {
00032             int i, j;
00033             double valor;
00034 
00035             std::cout « "Digite a linha: ";
00036             std::cin » i;
00037             std::cin.ignore();
00038 
00039             std::cout « "Digite a coluna: ";
00040             std::cin » j;
00041             std::cin.ignore();
00042 
00043             std::cout « "Digite o valor: ";
00044             std::cin » valor;
00045             std::cin.ignore();
00046 
00047             try
00048             {
00049                 matriz.insert(i, j, valor);
00050             }
00051             catch (const std::exception &e)
00052             {
00053                 std::cerr « e.what() « '\n';
00054             }
00055             break;
00056         }
00057 
00058         case 2:
00059         {
00060             std::cout « "Tem certeza que deseja limpar a matriz? [s/n]" « std::endl;
00061             char confirmacao;
00062             std::cin » confirmacao;
00063 
00064             switch (confirmacao)
00065             {
00066             case 's':
00067             case 'S':
00068                 matriz.limpar();
00069                 std::cout « "Matriz limpa" « std::endl;
00070                 break;
00071 
00072             case 'n':
00073             case 'N':
00074                 std::cout « "Operação cancelada" « std::endl;
00075                 break;
00076             }
00077             break;
00078         }
00079 
00080         case 3:
00081         {
00082             std::cout « "Voltando..." « std::endl;
00083             return;
00084         }
00085 
00086         default:
00087         {
00088             std::cout « "Opção inválida" « std::endl;
00089             break;
00090         }
00091         }
00092     }
00093 }
```

5.4 manipMatriz.hpp

Ir para a documentação desse arquivo.

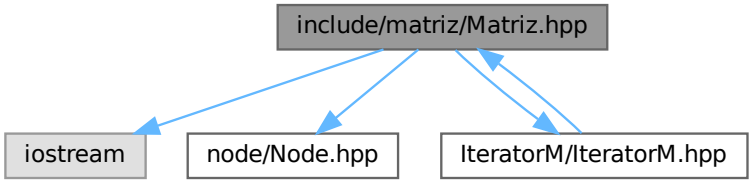
```
00001 #pragma once
00002 
00003 #include "matriz/Matriz.hpp"
00004 
00013 void manipMatrix(Matriz &matriz, const std::string &nomeMatriz)
00014 {
00015     while (true)
00016     {
00017         std::cout « "Matriz selecionada: " « nomeMatriz « std::endl;
00018         matriz.print();
00019         std::cout « "Escolha uma opção:" « std::endl;
```

```
00020         std::cout « "[1] - Inserir Valor" « std::endl;
00021         std::cout « "[2] - Limpar Matriz" « std::endl;
00022         std::cout « "[3] - Voltar" « std::endl;
00023 
00024         int opcao;
00025         std::cin » opcao;
00026         std::cin.ignore();
00027 
00028         switch (opcao)
00029         {
00030         case 1:
00031         {
00032             int i, j;
00033             double valor;
00034 
00035             std::cout « "Digite a linha: ";
00036             std::cin » i;
00037             std::cin.ignore();
00038 
00039             std::cout « "Digite a coluna: ";
00040             std::cin » j;
00041             std::cin.ignore();
00042 
00043             std::cout « "Digite o valor: ";
00044             std::cin » valor;
00045             std::cin.ignore();
00046 
00047             try
00048             {
00049                 matriz.insert(i, j, valor);
00050             }
00051             catch (const std::exception &e)
00052             {
00053                 std::cerr « e.what() « '\n';
00054             }
00055             break;
00056         }
00057 
00058         case 2:
00059         {
00060             std::cout « "Tem certeza que deseja limpar a matriz? [s/n]" « std::endl;
00061             char confirmacao;
00062             std::cin » confirmacao;
00063 
00064             switch (confirmacao)
00065             {
00066             case 's':
00067             case 'S':
00068                 matriz.limpar();
00069                 std::cout « "Matriz limpa" « std::endl;
00070                 break;
00071 
00072             case 'n':
00073             case 'N':
00074                 std::cout « "Operação cancelada" « std::endl;
00075                 break;
00076             }
00077             break;
00078         }
00079 
00080         case 3:
00081         {
00082             std::cout « "Voltando..." « std::endl;
00083             return;
00084         }
00085 
00086         default:
00087         {
00088             std::cout « "Opção inválida" « std::endl;
00089             break;
00090         }
00091         }
00092     }
00093 }
```

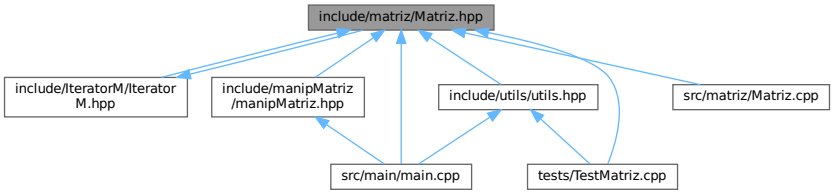
5.5 Referência do Arquivo include/matriz/Matriz.hpp

```
#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.

5.6 Matriz.hpp

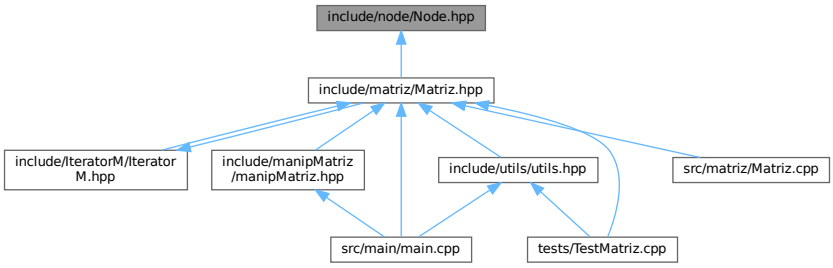
Ir para a documentação desse arquivo.

```
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
00092     Matriz(const Matriz &outra);
00093
00117     ~Matriz();
00118
00133     IteratorM begin();
00134
```

```
00144     IteratorM end();
00145
00156     IteratorM begin() const;
00157
00167     IteratorM end() const;
00168
00190     Matriz operator=(Matriz matriz);
00191
00200     int getLinhas() const;
00201
00210     int getColunas() const;
00211
00240     void limpar();
00241
00272     void insert(const int &posI, const int &posJ, const double &value);
00273
00292     double get(const int &posI, const int &posJ);
00293
00313     double get(const int &posI, const int &posJ) const;
00314
00322     void print();
00323 };
00324
00325 #endif
```

5.7 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.

5.8 Node.hpp

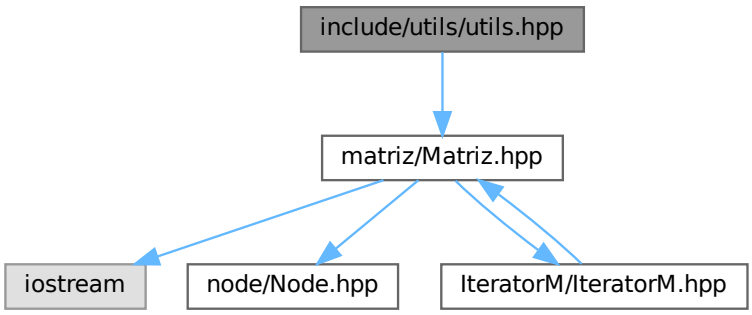
Ir para a documentação desse arquivo.

```
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     {
```

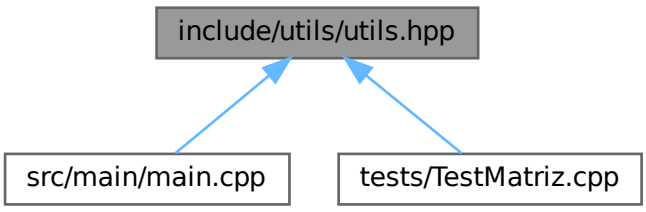
```
00030         direita = nullptr;
00031         abaixo = nullptr;
00032     }
00033
00041     void atualizaValor(const double &novoValor)
00042     {
00043         valor = novoValor;
00044     }
00045 };
00046
00047 #endif
```

5.9 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 &matrixA, 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.

5.9.1 Funções

5.9.1.1 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

matrizA	A primeira matriz a ser multiplicada.
matrizB	A segunda matriz a ser multiplicada.

Retorna

Matriz A matriz resultante da multiplicação de matrizA e matrizB.

Exceções

std::invalid_argument	Se o número de colunas de matrizA for diferente do número de linhas de matrizB.

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: A matriz A precisa possui o número de colunas iguais ao
      número de 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 }
```

5.9.1.2 sum()

```
Matriz sum (
              const Matriz & matrixA,
              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

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

Exceções

std::invalid_argument	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
00036             matriz.insert(i, j, valor);
00037         }
00038     }
00039
00040     return matriz;
00041 }
```

5.10 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 
00036             matriz.insert(i, j, valor);
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: A matriz A precisa possui o número de colunas iguais ao
      número de 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
```

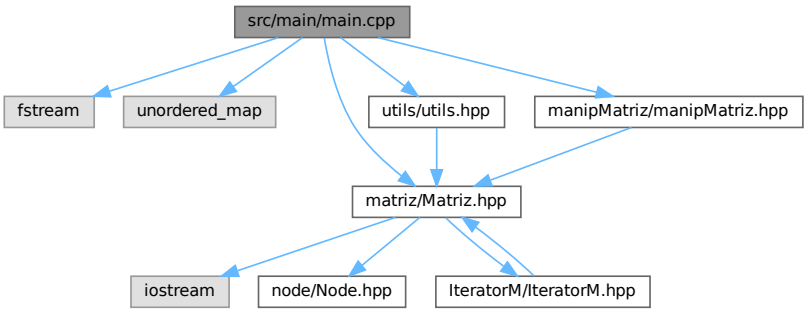
5.11 Referência do Arquivo README.md

5.12 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"
#include "manipMatriz/manipMatriz.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 , MANIPULAR_MATRIZ , 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 ()

5.12.1 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.

5.12.2 Definições dos tipos

5.12.2.1 string

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

Autores

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

Definição na linha 13 do arquivo main.cpp.

5.12.2.2 unordered_map

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

Definição na linha 14 do arquivo main.cpp.

5.12.3 Enumerações

5.12.3.1 Opcoes

```
enum Opcoes
```

Enumeradores

LER_MATRIZ	
MANIPULAR_MATRIZ	
IMPRIMIR_MATRIZ	
SOMAR_MATRIZES	
MULTIPLICAR_MATRIZES	
SAIR	

Definição na linha 16 do arquivo main.cpp.

```
00017 {
00018     LER_MATRIZ = 1,
```

```
00019     MANIPULAR_MATRIZ,
00020     IMPRIMIR_MATRIZ,
00021     SOMAR_MATRIZES,
00022     MULTIPLICAR_MATRIZES,
00023     SAIR
00024 };
```

5.12.4 Funções

5.12.4.1 escolherMatrizes()

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

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

Parâmetros

filename	Referência para a string que armazenará o nome da primeira matriz.
filename2	Referência para a string que armazenará o nome da segunda matriz.

Definição na linha 383 do arquivo main.cpp.

```
00384 {
00385     std::cout « "Coloque o nome da primeira matriz: ";
00386     std::getline(std::cin, filename);
00387
00388     std::cout « "Coloque o nome da segunda matriz: ";
00389     std::getline(std::cin, filename2);
00390 }
```

5.12.4.2 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

filename	O nome do arquivo cujo registro de matriz deve ser verificado.
matrizes	O `unordered_map` 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 338 do arquivo main.cpp.

```
00339 {
00340     return matrizes.find(filename) != matrizes.end();
00341 }
```

5.12.4.3 main()

```
int main ( )
```

Definição na linha 117 do arquivo main.cpp.

```
00118 {
00119     // Configura a localização para suportar caracteres especiais em português
00120     setlocale(LC_ALL, "pt_BR.UTF-8");
00121 
00122     std::cout « "Bem-vindo ao programa de manipulação de matrizes esparsas" « std::endl;
00123     std::cout « "-------------------------------------------------------------" « std::endl;
00124 
00125     // Mapa associativo para armazenar as matrizes
00126     unordered_map matrizes;
00127 
00128     while (true)
00129     {
00130         // Exibe o menu de opções
00131         std::cout « "Escolha uma opção:" « std::endl;
00132         std::cout « "[1] - Ler Matriz" « std::endl;
00133         std::cout « "[2] - Manipular Matriz" « std::endl;
00134         std::cout « "[3] - Imprimir Matriz" « std::endl;
00135         std::cout « "[4] - Somar Matrizes" « std::endl;
00136         std::cout « "[5] - Multiplicar Matrizes" « std::endl;
00137         std::cout « "[6] - Sair" « std::endl;
00138         int opcao;
00139         std::cin » opcao;
00140         std::cin.ignore();
00141 
00142         switch (opcao)
00143         {
00144         case LER_MATRIZ:
00145         {
00146             // Lê uma matriz a partir de um arquivo
00147             string filename;
00148             std::cout « "Digite o nome do arquivo: ";
00149             std::getline(std::cin, filename);
00150 
00151             Matriz matriz;
00152 
00153             try
00154             {
00155                 readMatrix(matriz, filename);
00156             }
00157             catch (const std::exception &e)
00158             {
00159                 std::cerr « e.what() « '\n';
00160                 break;
00161             }
00162 
00163             // Armazena a matriz no mapa associativo
00164             matrizes.insert(std::make_pair(filename, matriz));
00165             break;
00166         }
00167 
00168         case MANIPULAR_MATRIZ:
00169         {
00170             if (matrizes.empty())
00171             {
00172                 std::cout « "Não há matrizes armazenadas" « std::endl;
00173                 break;
00174             }
00175 
00176             // Manipula uma matriz armazenada
00177             printMatrizes(matrizes);
00178 
00179             std::cout « "Qual matriz deseja manipular?" « std::endl;
00180             string filename;
00181             std::getline(std::cin, filename);
```

```
00182
00183             if (!existeMatriz(filename, matrizes))
00184             {
00185                 std::cout « "Matriz não encontrada" « std::endl;
00186                 break;
00187             }
00188
00189             std::system("cls||clear");
00190             manipMatrix(matrizes[filename], filename);
00191
00192             break;
00193         }
00194
00195         case IMPRIMIR_MATRIZ:
00196         {
00197             if (matrizes.empty())
00198             {
00199                 std::cout « "Não há matrizes armazenadas" « std::endl;
00200                 break;
00201             }
00202
00203             // Imprime uma matriz armazenada
00204             std::cout « "Qual matriz deseja imprimir?" « std::endl;
00205             printMatrizes(matrizes);
00206             std::cout « "Coloque o nome do arquivo que deseja imprimir: ";
00207             string filename;
00208             std::getline(std::cin, filename);
00209
00210             if (existeMatriz(filename, matrizes))
00211                 matrizes[filename].print();
00212             else
00213                 std::cout « "Matriz não encontrada" « std::endl;
00214
00215             break;
00216         }
00217
00218         case SOMAR_MATRIZES:
00219         {
00220             if (matrizes.empty())
00221             {
00222                 std::cout « "Não há matrizes armazenadas" « std::endl;
00223                 break;
00224             }
00225
00226             // Soma duas matrizes armazenadas
00227             std::cout « "Quais as matrizes irá usar para somar" « std::endl;
00228             printMatrizes(matrizes);
00229             std::string filename, filename2;
00230             escolherMatrizes(filename, filename2);
00231
00232             if (!existeMatriz(filename, matrizes) || !existeMatriz(filename2, matrizes))
00233             {
00234                 std::cout « "Alguma matriz não foi encontrada" « std::endl;
00235                 break;
00236             }
00237
00238             Matriz matriz;
00239
00240             try
00241             {
00242                 matriz = sum(matrizes[filename], matrizes[filename2]);
00243             }
00244             catch (const std::exception &e)
00245             {
00246                 std::cerr « e.what() « '\n';
00247                 break;
00248             }
00249
00250             matriz.print();
00251             salvarMatriz(matriz, matrizes);
00252             break;
00253         }
00254
00255         case MULTIPLICAR_MATRIZES:
00256         {
00257             if (matrizes.empty())
00258             {
00259                 std::cout « "Não há matrizes armazenadas" « std::endl;
00260                 break;
00261             }
00262
00263             // Multiplica duas matrizes armazenadas
00264             std::cout « "Quais as matrizes irá usar para multiplicar" « std::endl;
00265             printMatrizes(matrizes);
00266             std::string filename, filename2;
00267             escolherMatrizes(filename, filename2);
00268
```

```
00269             if (!existeMatriz(filename, matrizes) || !existeMatriz(filename2, matrizes))
00270             {
00271                 std::cout « "Alguma matriz não foi encontrada" « std::endl;
00272                 break;
00273             }
00274
00275             Matriz matriz;
00276
00277             try
00278             {
00279                 matriz = multiply(matrizes[filename], matrizes[filename2]);
00280             }
00281             catch (const std::exception &e)
00282             {
00283                 std::cerr « e.what() « '\n';
00284                 break;
00285             }
00286
00287             matriz.print();
00288             salvarMatriz(matriz, matrizes);
00289             break;
00290         }
00291
00292         case SAIR:
00293         {
00294             // Encerra o programa
00295             std::cout « "Saindo..." « std::endl;
00296             return 0;
00297         }
00298
00299         default:
00300         {
00301             // Opção inválida
00302             std::cout « "Opção inválida" « std::endl;
00303             std::cin.clear();
00304             std::cin.ignore();
00305             break;
00306         }
00307         }
00308
00309         // Pausa e limpa a tela
00310         std::system("pause || read -p 'Pressione enter para continuar...' var");
00311         std::system("cls||clear");
00312     }
00313 }
```

5.12.4.4 printMatrizes()

```
void printMatrizes (
            const unordered_map & matrizes )
```

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

Parâmetros

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

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 392 do arquivo main.cpp.

```
00393 {
00394     for (const auto &par : matrizes)
00395     {
00396         std::cout « "------------------" « std::endl;
00397         std::cout « par.first « " |" « par.second.getLinhas() « " x " « par.second.getColunas() « "|"
      « std::endl;
00398         std::cout « "------------------" « std::endl;
00399     }
00400 }
```

5.12.4.5 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

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

Exceções

std::runtime_error	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 315 do arquivo main.cpp.

```
00316 {
00317     std::ifstream file("src/arquivos/" + filename);
00318
00319     if (!file || !file.is_open())
00320         throw std::runtime_error("Erro ao abrir o arquivo");
00321
00322     int linhas{0}, colunas{0};
00323     file » linhas » colunas;
00324
00325     matriz = Matriz(linhas, colunas);
00326
00327     int i{0}, j{0};
00328     double valor{0.0f};
00329
00330     while (file » i » j » valor)
00331     {
00332         matriz.insert(i, j, valor);
00333     }
00334
00335     file.close();
00336 }
```

5.12.4.6 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

matriz	Objeto do tipo Matriz que será salvo.
matrizes	Estrutura (unordered_map) 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 343 do arquivo main.cpp.

```
00344 {
00345     while (true)
00346     {
00347         std::cout « "Deseja salvar a matriz? [S/N]: ";
00348         char resposta;
00349         std::cin » resposta;
00350         std::cin.ignore();
00351 
00352         switch (tolower(resposta))
00353         {
00354         case 'n':
00355             return;
00356 
00357         case 's':
00358         {
00359             std::cout « "Digite o nome que deseja salvar a matriz: ";
00360             string filename;
00361 
00362             std::getline(std::cin, filename);
00363 
00364             if (existeMatriz(filename, matrizes))
00365             {
00366                 std::cout « "Já existe uma matriz com esse nome, tente outro nome para salvar" «
    std::endl;
00367                 break;
00368             }
00369 
00370             matrizes.insert(std::make_pair(filename, matriz));
00371 
00372             std::cout « "Matriz salva com sucesso" « std::endl;
00373             return;
00374         }
00375 
00376         default:
00377             std::cout « "Opção inválida" « std::endl;
00378             break;
00379         }
00380     }
00381 }
```

5.13 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 #include "manipMatriz/manipMatriz.hpp"
00012 
00013 using string = std::string;
```

```
00014 using unordered_map = std::unordered_map<string, Matriz>;
00015
00016 enum Opcoes // Enumeração para as opções do menu
00017 {
00018     LER_MATRIZ = 1,
00019     MANIPULAR_MATRIZ,
00020     IMPRIMIR_MATRIZ,
00021     SOMAR_MATRIZES,
00022     MULTIPLICAR_MATRIZES,
00023     SAIR
00024 };
00025
00048 void readMatrix(Matriz &matriz, const std::string filename);
00049
00063 bool existeMatriz(const std::string filename, const unordered_map &matrizes);
00064
00080 void salvarMatriz(const Matriz &matriz, unordered_map &matrizes);
00081
00088 void escolherMatrizes(string &filename, string &filename2);
00089
00099 void printMatrizes(const unordered_map &matrizes);
00100
00117 int main()
00118 {
00119     // Configura a localização para suportar caracteres especiais em português
00120     setlocale(LC_ALL, "pt_BR.UTF-8");
00121
00122     std::cout « "Bem-vindo ao programa de manipulação de matrizes esparsas" « std::endl;
00123     std::cout « "-------------------------------------------------------------" « std::endl;
00124
00125     // Mapa associativo para armazenar as matrizes
00126     unordered_map matrizes;
00127
00128     while (true)
00129     {
00130         // Exibe o menu de opções
00131         std::cout « "Escolha uma opção:" « std::endl;
00132         std::cout « "[1] - Ler Matriz" « std::endl;
00133         std::cout « "[2] - Manipular Matriz" « std::endl;
00134         std::cout « "[3] - Imprimir Matriz" « std::endl;
00135         std::cout « "[4] - Somar Matrizes" « std::endl;
00136         std::cout « "[5] - Multiplicar Matrizes" « std::endl;
00137         std::cout « "[6] - Sair" « std::endl;
00138         int opcao;
00139         std::cin » opcao;
00140         std::cin.ignore();
00141
00142         switch (opcao)
00143         {
00144         case LER_MATRIZ:
00145         {
00146             // Lê uma matriz a partir de um arquivo
00147             string filename;
00148             std::cout « "Digite o nome do arquivo: ";
00149             std::getline(std::cin, filename);
00150
00151             Matriz matriz;
00152
00153             try
00154             {
00155                 readMatrix(matriz, filename);
00156             }
00157             catch (const std::exception &e)
00158             {
00159                 std::cerr « e.what() « '\n';
00160                 break;
00161             }
00162
00163             // Armazena a matriz no mapa associativo
00164             matrizes.insert(std::make_pair(filename, matriz));
00165             break;
00166         }
00167
00168         case MANIPULAR_MATRIZ:
00169         {
00170             if (matrizes.empty())
00171             {
00172                 std::cout « "Não há matrizes armazenadas" « std::endl;
00173                 break;
00174             }
00175
00176             // Manipula uma matriz armazenada
00177             printMatrizes(matrizes);
00178
00179             std::cout « "Qual matriz deseja manipular?" « std::endl;
00180             string filename;
00181             std::getline(std::cin, filename);
```

```
00182
00183             if (!existeMatriz(filename, matrizes))
00184             {
00185                 std::cout « "Matriz não encontrada" « std::endl;
00186                 break;
00187             }
00188
00189             std::system("cls||clear");
00190             manipMatrix(matrizes[filename], filename);
00191
00192             break;
00193         }
00194
00195         case IMPRIMIR_MATRIZ:
00196         {
00197             if (matrizes.empty())
00198             {
00199                 std::cout « "Não há matrizes armazenadas" « std::endl;
00200                 break;
00201             }
00202
00203             // Imprime uma matriz armazenada
00204             std::cout « "Qual matriz deseja imprimir?" « std::endl;
00205             printMatrizes(matrizes);
00206             std::cout « "Coloque o nome do arquivo que deseja imprimir: ";
00207             string filename;
00208             std::getline(std::cin, filename);
00209
00210             if (existeMatriz(filename, matrizes))
00211                 matrizes[filename].print();
00212             else
00213                 std::cout « "Matriz não encontrada" « std::endl;
00214
00215             break;
00216         }
00217
00218         case SOMAR_MATRIZES:
00219         {
00220             if (matrizes.empty())
00221             {
00222                 std::cout « "Não há matrizes armazenadas" « std::endl;
00223                 break;
00224             }
00225
00226             // Soma duas matrizes armazenadas
00227             std::cout « "Quais as matrizes irá usar para somar" « std::endl;
00228             printMatrizes(matrizes);
00229             std::string filename, filename2;
00230             escolherMatrizes(filename, filename2);
00231
00232             if (!existeMatriz(filename, matrizes) || !existeMatriz(filename2, matrizes))
00233             {
00234                 std::cout « "Alguma matriz não foi encontrada" « std::endl;
00235                 break;
00236             }
00237
00238             Matriz matriz;
00239
00240             try
00241             {
00242                 matriz = sum(matrizes[filename], matrizes[filename2]);
00243             }
00244             catch (const std::exception &e)
00245             {
00246                 std::cerr « e.what() « '\n';
00247                 break;
00248             }
00249
00250             matriz.print();
00251             salvarMatriz(matriz, matrizes);
00252             break;
00253         }
00254
00255         case MULTIPLICAR_MATRIZES:
00256         {
00257             if (matrizes.empty())
00258             {
00259                 std::cout « "Não há matrizes armazenadas" « std::endl;
00260                 break;
00261             }
00262
00263             // Multiplica duas matrizes armazenadas
00264             std::cout « "Quais as matrizes irá usar para multiplicar" « std::endl;
00265             printMatrizes(matrizes);
00266             std::string filename, filename2;
00267             escolherMatrizes(filename, filename2);
00268
```

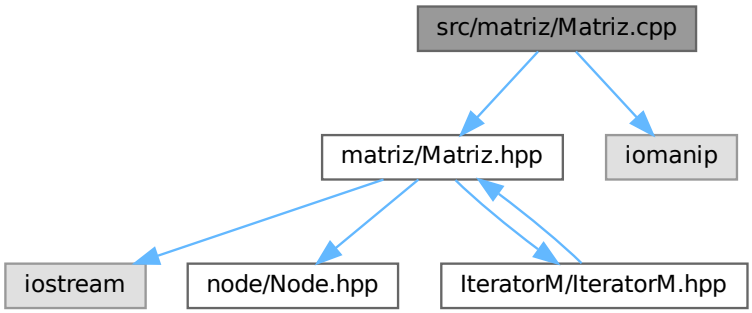
```
00269             if (!existeMatriz(filename, matrizes) || !existeMatriz(filename2, matrizes))
00270             {
00271                 std::cout « "Alguma matriz não foi encontrada" « std::endl;
00272                 break;
00273             }
00274
00275             Matriz matriz;
00276
00277             try
00278             {
00279                 matriz = multiply(matrizes[filename], matrizes[filename2]);
00280             }
00281             catch (const std::exception &e)
00282             {
00283                 std::cerr « e.what() « '\n';
00284                 break;
00285             }
00286
00287             matriz.print();
00288             salvarMatriz(matriz, matrizes);
00289             break;
00290         }
00291
00292         case SAIR:
00293         {
00294             // Encerra o programa
00295             std::cout « "Saindo..." « std::endl;
00296             return 0;
00297         }
00298
00299         default:
00300         {
00301             // Opção inválida
00302             std::cout « "Opção inválida" « std::endl;
00303             std::cin.clear();
00304             std::cin.ignore();
00305             break;
00306         }
00307         }
00308
00309         // Pausa e limpa a tela
00310         std::system("pause || read -p 'Pressione enter para continuar...' var");
00311         std::system("cls||clear");
00312     }
00313 }
00314
00315 void readMatrix(Matriz &matriz, const std::string filename)
00316 {
00317     std::ifstream file("src/arquivos/" + filename);
00318
00319     if (!file || !file.is_open())
00320         throw std::runtime_error("Erro ao abrir o arquivo");
00321
00322     int linhas{0}, colunas{0};
00323     file » linhas » colunas;
00324
00325     matriz = Matriz(linhas, colunas);
00326
00327     int i{0}, j{0};
00328     double valor{0.0f};
00329
00330     while (file » i » j » valor)
00331     {
00332         matriz.insert(i, j, valor);
00333     }
00334
00335     file.close();
00336 }
00337
00338 bool existeMatriz(const std::string filename, const unordered_map &matrizes)
00339 {
00340     return matrizes.find(filename) != matrizes.end();
00341 }
00342
00343 void salvarMatriz(const Matriz &matriz, unordered_map &matrizes)
00344 {
00345     while (true)
00346     {
00347         std::cout « "Deseja salvar a matriz? [S/N]: ";
00348         char resposta;
00349         std::cin » resposta;
00350         std::cin.ignore();
00351
00352         switch (tolower(resposta))
00353         {
00354         case 'n':
00355             return;
```

```
00356
00357         case 's':
00358         {
00359             std::cout « "Digite o nome que deseja salvar a matriz: ";
00360             string filename;
00361
00362             std::getline(std::cin, filename);
00363
00364             if (existeMatriz(filename, matrizes))
00365             {
00366                 std::cout « "Já existe uma matriz com esse nome, tente outro nome para salvar" «
      std::endl;
00367                 break;
00368             }
00369
00370             matrizes.insert(std::make_pair(filename, matriz));
00371
00372             std::cout « "Matriz salva com sucesso" « std::endl;
00373             return;
00374         }
00375
00376         default:
00377             std::cout « "Opção inválida" « std::endl;
00378             break;
00379         }
00380     }
00381 }
00382
00383 void escolherMatrizes(string &filename, string &filename2)
00384 {
00385     std::cout « "Coloque o nome da primeira matriz: ";
00386     std::getline(std::cin, filename);
00387
00388     std::cout « "Coloque o nome da segunda matriz: ";
00389     std::getline(std::cin, filename2);
00390 }
00391
00392 void printMatrizes(const unordered_map &matrizes)
00393 {
00394     for (const auto &par : matrizes)
00395     {
00396         std::cout « "------------------" « std::endl;
00397         std::cout « par.first « " |" « par.second.getLinhas() « " x " « par.second.getColunas() « "|"
      « std::endl;
00398         std::cout « "------------------" « std::endl;
00399     }
00400 }
```

5.14 Referência do Arquivo src/matriz/Matriz.cpp

```
#include "matriz/Matriz.hpp"
#include <iomanip>
```

Gráfico de dependência de inclusões para Matriz.cpp:



5.15 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 {
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 }
00041
00042 Matriz::Matriz(const Matriz &outra) : Matriz(outra.linhas, outra.colunas)
00043 {
00044     for (IteratorM it = outra.begin(); it != outra.end(); ++it)
00045         this->insert(it.current->linha, it.current->coluna, *it);
00046 }
00047
00048 IteratorM Matriz::begin()
00049 {
00050     return IteratorM(cabecalho->abaixo, cabecalho->abaixo->direita);
00051 }
00052
00053 IteratorM Matriz::end()
00054 {
00055     return IteratorM(cabecalho, cabecalho->direita);
00056 }
00057
00058 IteratorM Matriz::begin() const
00059 {
00060     return IteratorM(cabecalho->abaixo, cabecalho->abaixo->direita);
00061 }
00062
00063 IteratorM Matriz::end() const
00064 {
00065     return IteratorM(cabecalho, cabecalho->direita);
00066 }
00067
00068 Matriz Matriz::operator=(Matriz matriz)
00069 {
00070     // Troca os dados do objeto atual com os dados de 'matriz'
00071     std::swap(cabecalho, matriz.cabecalho);
00072     std::swap(linhas, matriz.linhas);
00073     std::swap(colunas, matriz.colunas);
00074     // 'matriz' é destruída, liberando os recursos antigos
00075     return *this;
00076 }
00077
00078 int Matriz::getLinhas() const
00079 {
00080     return linhas;
00081 }
00082
```

```
00083 int Matriz::getColunas() const
00084 {
00085     return colunas;
00086 }
00087
00088 Matriz::~Matriz()
00089 {
00090     if (cabecalho == nullptr)
00091         return;
00092
00093     limpar();
00094
00095     Node *linhaAtual = cabecalho->abaixo;
00096     while (linhaAtual != cabecalho)
00097     {
00098         Node *proximoLinha = linhaAtual->abaixo;
00099         delete linhaAtual;
00100         linhaAtual = proximoLinha;
00101     }
00102
00103     Node *colunaAtual = cabecalho->direita;
00104     while (colunaAtual != cabecalho)
00105     {
00106         Node *proximoColuna = colunaAtual->direita;
00107         delete colunaAtual;
00108         colunaAtual = proximoColuna;
00109     }
00110
00111     delete cabecalho;
00112     cabecalho = nullptr;
00113 }
00114
00115 void Matriz::limpar()
00116 {
00117     if (cabecalho == nullptr)
00118         return;
00119
00120     Node *LinhaAtual = cabecalho->abaixo;
00121     if (LinhaAtual == cabecalho)
00122         return;
00123
00124     Node *ColunaAtual = LinhaAtual;
00125     while (ColunaAtual->abaixo != cabecalho)
00126     {
00127         ColunaAtual = ColunaAtual->abaixo;
00128     }
00129
00130     ColunaAtual->abaixo = nullptr;
00131
00132     for (Node *linha = LinhaAtual; linha != nullptr; linha = linha->abaixo)
00133     {
00134
00135         Node *atual = linha->direita;
00136         while (atual != linha)
00137         {
00138             Node *proximo = atual->direita;
00139             delete atual;
00140             atual = proximo;
00141         }
00142         linha->direita = linha;
00143     }
00144
00145     ColunaAtual = LinhaAtual;
00146     while (ColunaAtual->abaixo != nullptr)
00147     {
00148         ColunaAtual = ColunaAtual->abaixo;
00149     }
00150     ColunaAtual->abaixo = cabecalho;
00151 }
00152
00153 void Matriz::insert(const int &posI, const int &posJ, const double &value)
00154 {
00155     if (value == 0)
00156         return;
00157
00158     if (posI <= 0 || posI > linhas || posJ <= 0 || posJ > colunas)
00159         throw std::invalid_argument("Erro: Local de inserção inválido");
00160
00161     Node *linhaAtual = cabecalho;
00162     while (linhaAtual->linha < posI)
00163     {
00164         linhaAtual = linhaAtual->abaixo;
00165     }
00166
00167     Node *aux = linhaAtual;
00168     while (aux->direita != linhaAtual && aux->direita->coluna < posJ)
00169     {
```

```
00170         aux = aux->direita;
00171     }
00172
00173     if (aux->direita->coluna == posJ)
00174     {
00175         aux->direita->atualizaValor(value);
00176         return;
00177     }
00178
00179     Node *novo = new Node(posI, posJ, value);
00180
00181     novo->direita = aux->direita;
00182     aux->direita = novo;
00183
00184     Node *colunaAtual = cabecalho;
00185     while (colunaAtual->coluna < posJ)
00186     {
00187         colunaAtual = colunaAtual->direita;
00188     }
00189
00190     aux = colunaAtual;
00191     while (aux->abaixo != colunaAtual && aux->abaixo->linha < posI)
00192     {
00193         aux = aux->abaixo;
00194     }
00195
00196     novo->abaixo = aux->abaixo;
00197     aux->abaixo = novo;
00198 }
00199
00200 double Matriz::get(const int &posI, const int &posJ)
00201 {
00202     if (posI <= 0 || posI > linhas || posJ <= 0 || posJ > colunas)
00203         throw std::invalid_argument("Erro: Local de acesso inválido");
00204
00205     IteratorM it = begin();
00206
00207     // Avança enquanto o nó atual estiver "antes" da posição desejada.
00208     while (it != end() &&
00209            (it.current->linha < posI ||
00210             (it.current->linha == posI && it.current->coluna < posJ)))
00211     {
00212         ++it;
00213     }
00214
00215     // Se o nó atual corresponde exatamente à posição, retorna o valor.
00216     if (it != end() && it.current->linha == posI && it.current->coluna == posJ)
00217         return *it;
00218
00219     return 0;
00220 }
00221
00222 double Matriz::get(const int &posI, const int &posJ) const
00223 {
00224     if (posI <= 0 || posI > linhas || posJ <= 0 || posJ > colunas)
00225         throw std::invalid_argument("Erro: Local de acesso inválido");
00226
00227     IteratorM it = begin();
00228
00229     // Avança enquanto o nó atual estiver "antes" da posição desejada.
00230     while (it != end() &&
00231            (it.current->linha < posI ||
00232             (it.current->linha == posI && it.current->coluna < posJ)))
00233     {
00234         ++it;
00235     }
00236
00237     // Se o nó atual corresponde exatamente à posição, retorna o valor.
00238     if (it != end() && it.current->linha == posI && it.current->coluna == posJ)
00239         return *it;
00240
00241     return 0;
00242 }
00243
00244 void Matriz::print()
00245 {
00246     IteratorM it = begin();
00247
00248     for (int i = 1; i <= linhas; i++)
00249     {
00250         for (int j = 1; j <= colunas; j++)
00251         {
00252             if (it.current->linha == i && it.current->coluna == j)
00253             {
00254                 std::cout « std::fixed « std::setprecision(1) « *it;
00255                 ++it;
00256             }
```

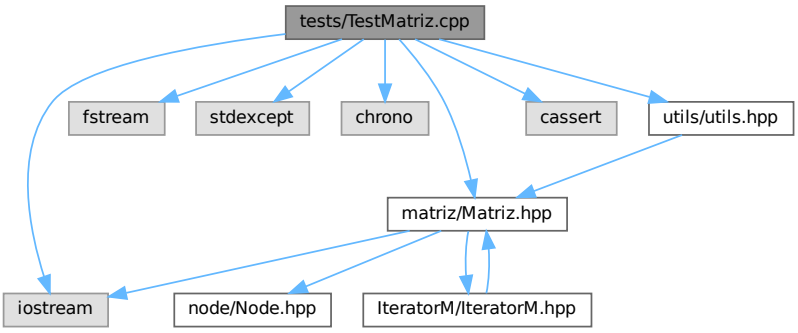
```
00257             else
00258             {
00259                 std::cout « "0.0";
00260             }
00261             std::cout « " ";
00262         }
00263         std::cout « std::endl;
00264     }
00265 }
```

5.16 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 &multi)
- 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 ()

5.16.1 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.

5.16.2 Funções

5.16.2.1 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 }
```

5.16.2.2 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
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     {                                          // Lê os valores da matriz
00132         matriz.insert(linha, coluna, valor); // Insere na matriz
00133     }
00134
00135     file.close(); // Fecha o arquivo
00136     return matriz;
00137 }
```

5.16.2.3 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/MatrixSoma1.txt"))
00230         {
00231             throw std::runtime_error("Arquivo MatrixSoma1.txt não encontrado.");
00232         }
00233         if (!arquivoExiste("tests/arquivosTestes/MatrixMulti1.txt"))
00234         {
00235             throw std::runtime_error("Arquivo MatrizMulti.txt não encontrado.");
00236         }
00237         if (!arquivoExiste("tests/arquivosTestes/Matrix3.txt"))
```

```
00238         {
00239             throw std::runtime_error("Arquivo Matriz3.txt não encontrado.");
00240         }
00241         if (!arquivoExiste("tests/arquivosTestes/MatrixSoma2.txt"))
00242         {
00243             throw std::runtime_error("Arquivo MatrixSoma1.txt não encontrado.");
00244         }
00245         if (!arquivoExiste("tests/arquivosTestes/MatrixMulti2.txt"))
00246         {
00247             throw std::runtime_error("Arquivo MatrizMulti.txt não encontrado.");
00248         }
00249 
00250         // Lê as matrizes de arquivos
00251         Matriz A = leitura("Matrix1.txt");
00252         Matriz B = leitura("Matrix2.txt");
00253         std::cout « "Matrizes lidas com sucesso" « std::endl;
00254 
00255         std::cout « "Teste 1 usando matrizes com o mesmo tamanho 3x3" « std::endl;
00256         Matriz soma = leitura("MatrixSoma1.txt");
00257         Matriz multi = leitura("MatrixMulti1.txt");
00258         // Executa os testes
00259         testeSoma(A, B, soma);
00260         testeMultiplicacao(A, B, multi);
00261 
00262         std::cout « "Teste 2 usando matrizes com tamanho diferentes, sendo uma 3x3 e outra 4x4 " «
      std::endl;
00263         Matriz C = leitura("Matrix3.txt");
00264         soma = leitura("MatrixSoma2.txt");
00265         multi = leitura("MatrixMulti2.txt");
00266         std::cout « "Matrizes lidas com sucesso" « std::endl;
00267 
00268         // Executa os testes
00269         try
00270         {
00271 
00272             testeSoma(A, C, soma);
00273             testeMultiplicacao(A, C, multi);
00274 
00275         }
00276         catch(std::exception& e) {
00277             std::cerr « e.what() « "\n";
00278         }
00279 
00280         try
00281         {
00282             testeMultiplicacao(A, C, multi);
00283 
00284         }
00285         catch(std::exception& e) {
00286             std::cerr « e.what() « "\n";
00287         }
00288 
00289         std::cout « "Testes de inserção e de Performance, sendo esta com uma martriz 100x100" «
      std::endl;
00290         testeInsercao();    // Teste básico de inserção
00291         testePerformance(); // Teste de performance para matrizes grandes
00292 
00293     }
00294     catch (const std::runtime_error &e)
00295     {
00296         std::cerr « "Erro main: " « e.what() « std::endl;
00297     }
00298 
00299     return 0;
00300 }
```

5.16.2.4 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 }
```

5.16.2.5 testeMultiplicacao()

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

Definição na linha 82 do arquivo TestMatriz.cpp.

```
00083 {
00084     Matriz D = multiply(A, B);
00085
00086     // Verificando as dimensões
00087     if (D.getColunas() != multi.getColunas() || D.getLinhas() != multi.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 <= multi.getLinhas(); i++)
00094     {
00095         for (int j = 1; j <= multi.getColunas(); j++)
00096         {
00097             if (D.get(i, j) != multi.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(multi.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 }
```

5.16.2.6 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 Multiplição: " « duracao2.count() « "ms" « std::endl;
00205 }
```

5.16.2.7 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: " +
    std::to_string(D.get(i, j)));
00063             }
00064         }
00065     }
00066 
00067     std::cout « "Teste de soma passou" « std::endl;
00068 }
```

5.17 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
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: " +
    std::to_string(D.get(i, j)));
00063             }
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
    exceção.
00076  *
00077  *   @param A Matriz A.
00078  *  @param B Matriz B.
00079  *
```

```
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 &multi)
00083 {
00084     Matriz D = multiply(A, B);
00085 
00086     // Verificando as dimensões
00087     if (D.getColunas() != multi.getColunas() || D.getLinhas() != multi.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 <= multi.getLinhas(); i++)
00094     {
00095         for (int j = 1; j <= multi.getColunas(); j++)
00096         {
00097             if (D.get(i, j) != multi.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(multi.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     {                                           // Lê os valores da matriz
00132         matriz.insert(linha, coluna, valor); // Insere na matriz
00133     }
00134 
00135     file.close(); // Fecha o arquivo
00136     return matriz;
00137 }
00138 
00139 /*
00140  * @brief função para verificar se um arquivo existe
00141  *
00142  * @param caminho Caminho do arquivo a ser verificado.
00143  * @return bool Verdadeiro se o arquivo existir, falso caso contrário.
00144  */
00145 bool arquivoExiste(const std::string &caminho)
00146 {
00147     std::ifstream file(caminho);
00148     return file.good();
00149 }
00150 
00174 void testePerformance()
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 Multiplição: " « 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/MatrixSoma1.txt"))
00230         {
00231             throw std::runtime_error("Arquivo MatrixSoma1.txt não encontrado.");
00232         }
00233         if (!arquivoExiste("tests/arquivosTestes/MatrixMulti1.txt"))
00234         {
00235             throw std::runtime_error("Arquivo MatrizMulti.txt não encontrado.");
00236         }
00237         if (!arquivoExiste("tests/arquivosTestes/Matrix3.txt"))
00238         {
00239             throw std::runtime_error("Arquivo Matriz3.txt não encontrado.");
00240         }
00241         if (!arquivoExiste("tests/arquivosTestes/MatrixSoma2.txt"))
00242         {
00243             throw std::runtime_error("Arquivo MatrixSoma1.txt não encontrado.");
00244         }
00245         if (!arquivoExiste("tests/arquivosTestes/MatrixMulti2.txt"))
00246         {
00247             throw std::runtime_error("Arquivo MatrizMulti.txt não encontrado.");
00248         }
00249 
00250         // Lê as matrizes de arquivos
00251         Matriz A = leitura("Matrix1.txt");
00252         Matriz B = leitura("Matrix2.txt");
00253         std::cout « "Matrizes lidas com sucesso" « std::endl;
00254 
00255         std::cout « "Teste 1 usando matrizes com o mesmo tamanho 3x3" « std::endl;
00256         Matriz soma = leitura("MatrixSoma1.txt");
00257         Matriz multi = leitura("MatrixMulti1.txt");
00258         // Executa os testes
00259         testeSoma(A, B, soma);
00260         testeMultiplicacao(A, B, multi);
00261 
00262         std::cout « "Teste 2 usando matrizes com tamanho diferentes, sendo uma 3x3 e outra 4x4 " «
      std::endl;
00263         Matriz C = leitura("Matrix3.txt");
00264         soma = leitura("MatrixSoma2.txt");
00265         multi = leitura("MatrixMulti2.txt");
00266         std::cout « "Matrizes lidas com sucesso" « std::endl;
00267 
00268         // Executa os testes
00269         try
00270         {
00271 
00272             testeSoma(A, C, soma);
00273             testeMultiplicacao(A, C, multi);
00274 
00275         }
00276         catch(std::exception& e) {
00277             std::cerr « e.what() « "\n";
00278         }
```

```
00279
00280         try
00281         {
00282             testeMultiplicacao(A, C, multi);
00283
00284         }
00285         catch(std::exception& e) {
00286             std::cerr « e.what() « "\n";
00287         }
00288
00289         std::cout « "Testes de inserção e de Performance, sendo esta com uma martriz 100x100" «
      std::endl;
00290         testeInsercao();    // Teste básico de inserção
00291         testePerformance(); // Teste de performance para matrizes grandes
00292
00293     }
00294     catch (const std::runtime_error &e)
00295     {
00296         std::cerr « "Erro main: " « e.what() « std::endl;
00297     }
00298
00299     return 0;
00300 }
```

Índice Remissivo