Sparse Matrix

Gerado por Doxygen 1.9.4

Capítulo 1

Índice das estruturas de dados

1.1 Estruturas de dados

Lista das estruturas de dados com uma breve descrição:

Capítulo 2

Índice dos ficheiros

2.1 Lista de ficheiros

Lista de todos os ficheiros com uma breve descrição:

Capítulo 3

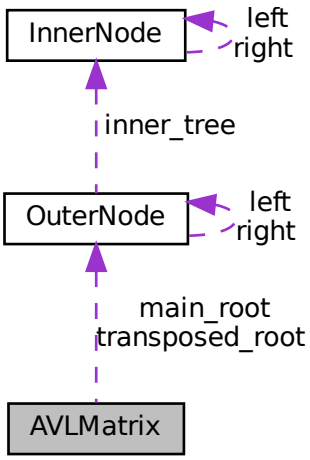
Documentação da estruturas de dados

3.1 Referência à estrutura AVLMatrix

Representação de uma matriz esparsa usando duas árvores AVL.

`#include <avl_matrix.h>`

Diagrama de colaboração para AVLMatrix:



Campos de Dados

- struct OuterNode ∗ main_root
- struct OuterNode ∗ transposed_root
- int k
- int n
- int m

3.1.1 Descrição detalhada

Representação de uma matriz esparsa usando duas árvores AVL.

A árvore main_root guarda os elementos no formato linha->coluna e a transposed_root armazena a matriz transposta para facilitar operações.

3.1.2 Documentação dos campos e atributos

3.1.2.1 k

```
int AVLMatrix::k
```

Quantidade de elementos não nulos.

3.1.2.2 m

```
int AVLMatrix::m
```

Quantidade de colunas de main_root.

3.1.2.3 main_root

```
struct OuterNode* AVLMatrix::main_root
```

Raiz (linhas) com árvores internas de colunas.

3.1.2.4 n

```
int AVLMatrix::n
```

Quantidade de linhas de main_root.

3.1.2.5 transposed_root

```
struct OuterNode* AVLMatrix::transposed_root
```

Raiz (colunas) com árvores internas de linhas.

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

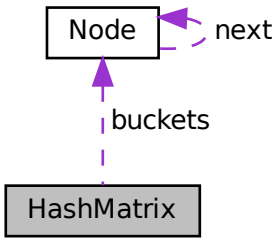
- avl_matrix.h

3.2 Referência à estrutura HashMatrix

Representação de uma matriz esparsa usando tabela de espalhamento (hash table).

`#include <hash_matrix.h>`

Diagrama de colaboração para HashMatrix:



Campos de Dados

- Node ** buckets
- int capacity
- int count
- int rows
- int columns
- bool is_transposed

3.2.1 Descrição detalhada

Representação de uma matriz esparsa usando tabela de espalhamento (hash table).

Armazena os buckets da tabela de espalhamento, capacidade total, número de elementos não nulos, dimensões da matriz (linhas e colunas) e flag de transposição.

3.2.2 Documentação dos campos e atributos

3.2.2.1 buckets

`Node** HashMatrix::buckets`

3.2.2.2 capacity

```
int HashMatrix::capacity
```

3.2.2.3 columns

```
int HashMatrix::columns
```

3.2.2.4 count

```
int HashMatrix::count
```

3.2.2.5 is_transposed

```
bool HashMatrix::is_transposed
```

3.2.2.6 rows

```
int HashMatrix::rows
```

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

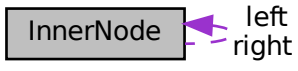
- hash_matrix.h

3.3 Referência à estrutura InnerNode

Nó da árvore interna (contém os elementos).

```
#include <avl_matrix.h>
```

Diagrama de colaboração para InnerNode:



Campos de Dados

- int key
- float data
- struct InnerNode * left
- struct InnerNode * right
- int height

3.3.1 Descrição detalhada

Nó da árvore interna (contém os elementos).

Armazena o índice da linha ou coluna, o valor do elemento não-nulo, os ponteiros dos filhos da árvore AVL interna e a altura relativa ao nó.

3.3.2 Documentação dos campos e atributos

3.3.2.1 data

```
float InnerNode::data
```

3.3.2.2 height

```
int InnerNode::height
```

3.3.2.3 key

```
int InnerNode::key
```

3.3.2.4 left

```
struct InnerNode* InnerNode::left
```

3.3.2.5 right

```
struct InnerNode* InnerNode::right
```

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

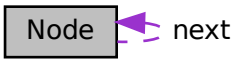
- avl_matrix.h

3.4 Referência à estrutura Node

Nó que contém os elementos em cada posição da tabela de espalhamento.

```
#include <hash_matrix.h>
```

Diagrama de colaboração para Node:



Campos de Dados

- int row
- int column
- float data
- struct Node * next

3.4.1 Descrição detalhada

Nó que contém os elementos em cada posição da tabela de espalhamento.

Armazena o índice da linha ou coluna, o valor do elemento não-nulo, e o ponteiro para o próximo nó na lista encadeada do elemento com hash colidido.

3.4.2 Documentação dos campos e atributos

3.4.2.1 column

```
int Node::column
```

3.4.2.2 data

```
float Node::data
```

3.4.2.3 next

```
struct Node* Node::next
```

3.4.2.4 row

```
int Node::row
```

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

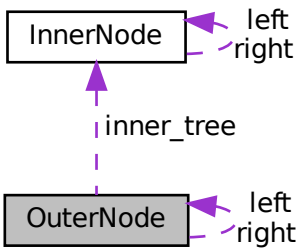
- hash_matrix.h

3.5 Referência à estrutura OuterNode

Nó da árvore externa (linha ou coluna da matriz).

```
#include <avl_matrix.h>
```

Diagrama de colaboração para OuterNode:



Campos de Dados

- int key
- struct InnerNode * inner_tree
- struct OuterNode * left
- struct OuterNode * right
- int height

3.5.1 Descrição detalhada

Nó da árvore externa (linha ou coluna da matriz).

Cada nó representa uma linha ou coluna da matriz e aponta para a árvore interna contendo os elementos não-nulos da linha ou coluna. Armazena o índice dessa linha ou coluna, os ponteiros dos filhos da árvore AVL externa e a altura relativa ao nó.

3.5.2 Documentação dos campos e atributos

3.5.2.1 height

```
int OuterNode::height
```

3.5.2.2 inner_tree

```
struct InnerNode* OuterNode::inner_tree
```

3.5.2.3 key

```
int OuterNode::key
```

3.5.2.4 left

```
struct OuterNode* OuterNode::left
```

3.5.2.5 right

```
struct OuterNode* OuterNode::right
```

A documentação para esta estrutura foi gerada a partir do seguinte ficheiro:

- avl_matrix.h

Capítulo 4

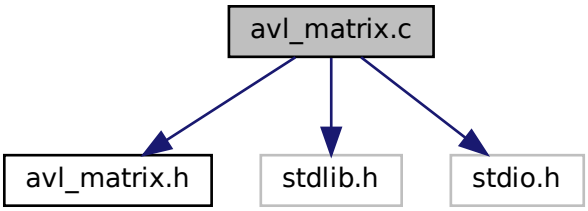
Documentação do ficheiro

4.1 Referência ao ficheiro avl_matrix.c

Implementação de matriz esparsa com duas árvores AVL (linhas e colunas).

```
#include "avl_matrix.h"
#include <stdlib.h>
#include <stdio.h>
```

Diagrama de dependências de inclusão para avl_matrix.c:



Funções

- static void _allocation_fail ()

  Encerramento imediato em caso de falha de alocação.
- const char * avl_status_string (AVLStatus status)

  Converte um código AVLStatus em mensagem textual.
- static AVLStatus _validate_matrix (AVLMatrix *matrix)

  Valida ponteiro da matriz e dimensões armazenadas.
- static AVLStatus _validate_indexes (AVLMatrix *matrix, int i, int j)

  Verifica se índices i,j estão dentro dos limites da matriz.
- static AVLStatus _validate_same_dimensions (AVLMatrix *A, AVLMatrix *B)

  Garante que duas matrizes têm dimensões compatíveis (mesma ordem).

- static int _height_i (InnerNode *tree)

  Retorna altura de um nó da árvore interna (0 se nulo).
- static int _height_o (OuterNode *tree)

  Retorna altura de um nó da árvore externa (0 se nulo).
- static int _balance_factor_i (InnerNode *tree)

  Calcula fator de balanceamento (altura esquerda - altura direita) da árvore interna.
- static int _balance_factor_o (OuterNode *tree)

  Calcula fator de balanceamento (altura esquerda - altura direita) da árvore externa.
- static int _max (int a, int b)

  Retorna o maior entre dois inteiros.
- static InnerNode * _left_rotate_i (InnerNode *tree)

  Rotação à esquerda na árvore interna.
- static InnerNode * _right_rotate_i (InnerNode *tree)

  Rotação à direita na árvore interna.
- static OuterNode * _left_rotate_o (OuterNode *tree)

  Rotação à esquerda na árvore externa.
- static OuterNode * _right_rotate_o (OuterNode *tree)

  Rotação à direita na árvore externa.
- static InnerNode * _find_node_i (InnerNode *tree, int search_key)

  Busca nó na árvore interna pelo índice.
- static OuterNode * _find_node_o (OuterNode *tree, int search_key)

  Busca nó na árvore externa pelo índice.
- static InnerNode * _insert_i (InnerNode *tree, int insert_key, float value, int *already_existed)

  Insere ou atualiza um valor na árvore interna mantendo balanceamento AVL.
- static OuterNode * _insert_o (OuterNode *tree, int insert_key, InnerNode *inner_tree)

  Insere ou atualiza um valor na árvore externa mantendo balanceamento AVL.
- static InnerNode * _find_max_i (InnerNode *tree)

  Encontra o maior nó (mais à direita) em uma árvore interna.
- static OuterNode * _find_max_o (OuterNode *tree)

  Encontra o maior nó (mais à direita) em uma árvore externa.
- static InnerNode * _remove_i (InnerNode *tree, int remove_key)

  Remove chave da árvore interna e rebalanceia.
- static OuterNode * _remove_o (OuterNode *tree, int remove_key)

  Remove fileira e rebalanceia a árvore externa.
- static void _free_i_tree (InnerNode *tree)

  Libera recursivamente uma árvore interna.
- static void _free_o_tree (OuterNode *tree)

  Libera recursivamente árvore externa e todas as internas associadas.
- static InnerNode * _clone_i_tree (InnerNode *tree)

  Clona profundamente uma árvore interna.
- static OuterNode * _clone_o_tree (OuterNode *tree)

  Clona profundamente a árvore externa e cada árvore interna.
- static AVLStatus _copy_matrix (AVLMatrix *source, AVLMatrix *dest)

  Copia conteúdo de uma matriz para outra, recriando ambas as árvores.
- static void _scalar_multiply_i_tree (InnerNode *tree, float a)

  Multiplica todos os nós de uma árvore interna por um escalar.
- static void _scalar_multiply_o_tree (OuterNode *tree, float a)

  Multiplica todos os valores armazenados na árvore externa (todas as fileiras) por um escalar.
- static void _copy_i (InnerNode *inner_tree, int *I, int *J, float *Data, int *position, int i)

  Copia pares (i,j,valor) de uma árvore interna para vetores auxiliares.
- static void _copy_o (OuterNode *outer_tree, int *I, int *J, float *Data, int *position)

Copia toda a matriz percorrendo a árvore externa e delegando para _copy_i.

- static AVLStatus _matmul_i_accumulate (InnerNode *inner_tree, int row, float A_value, AVLMatrix *C)

  *Acumula parcial de multiplicação de matrizes: atualiza linha de C com valores de A * B.*
- AVLStatus get_element_avl (AVLMatrix *matrix, int i, int j, float *out_value)

  Obtém o valor de um elemento da matriz.
- AVLStatus insert_element_avl (AVLMatrix *matrix, float value, int i, int j)

  Insere ou atualiza um elemento na matriz.
- AVLStatus delete_element_avl (AVLMatrix *matrix, int i, int j)

  Remove um elemento da matriz.
- AVLStatus transpose_avl (AVLMatrix *matrix)

  Transpõe a matriz.
- AVLStatus scalar_mul_avl (AVLMatrix *A, AVLMatrix *B, float a)

  *Calcula B = a * A (possivelmente in-place).*
- AVLStatus sum_avl (AVLMatrix *A, AVLMatrix *B, AVLMatrix *C)

  Calcula C = A + B.
- AVLStatus matrix_mul_avl (AVLMatrix *A, AVLMatrix *B, AVLMatrix *C)

  *Calcula C = A * B (sem suporte a in-place).*
- AVLMatrix * create_matrix_avl (int n, int m)

  Cria uma matriz vazia de dimensões n x m.
- void free_matrix_avl (AVLMatrix *matrix)

  Libera a memória associada à uma matriz AVL.

4.1.1 Descrição detalhada

Implementação de matriz esparsa com duas árvores AVL (linhas e colunas).

O arquivo contém funções internas de balanceamento e validação, além das operações públicas declaradas em avl_matrix.h.

4.1.2 Documentação das funções

4.1.2.1 _allocation_fail()

```
static void _allocation_fail ( )  [static]
```

Encerramento imediato em caso de falha de alocação.

Imprime uma mensagem de erro em stderr e aborta o processo usando EXIT_FAILURE.

4.1.2.2 _balance_factor_i()

```
static int _balance_factor_i (
            InnerNode * tree )  [static]
```

Calcula fator de balanceamento (altura esquerda - altura direita) da árvore interna.

Parâmetros

tree	raiz da subárvore interna.

4.1.2.3 _balance_factor_o()

```
static int _balance_factor_o (
            OuterNode * tree )  [static]
```

Calcula fator de balanceamento (altura esquerda - altura direita) da árvore externa.

Parâmetros

tree	raiz da subárvore externa.

4.1.2.4 _clone_i_tree()

```
static InnerNode * _clone_i_tree (
            InnerNode * tree )  [static]
```

Clona profundamente uma árvore interna.

Parâmetros

tree	raiz a copiar.

Retorna

Ponteiro para a nova raiz clonada.

4.1.2.5 _clone_o_tree()

```
static OuterNode * _clone_o_tree (
            OuterNode * tree )  [static]
```

Clona profundamente a árvore externa e cada árvore interna.

Parâmetros

tree	raiz a copiar.

Retorna

Ponteiro para a nova raiz clonada.

4.1.2.6 _copy_i()

```
static void _copy_i (
            InnerNode * inner_tree,
            int * I,
            int * J,
            float * Data,
            int * position,
            int i )  [static]
```

Copia pares (i,j,valor) de uma árvore interna para vetores auxiliares.

Parâmetros

inner_tree	árvore interna.
I	vetor de linhas.
J	vetor de colunas.
Data	vetor de valores.
position	ponteiro com posição inicial dos vetores.
i	índice de linha fixado.

4.1.2.7 _copy_matrix()

```
static AVLStatus _copy_matrix (
            AVLMatrix * source,
            AVLMatrix * dest )  [static]
```

Copia conteúdo de uma matriz para outra, recriando ambas as árvores.

Parâmetros

source	origem (já validada).
dest	destino, que terá árvores antigas liberadas.

4.1.2.8 _copy_o()

```
static void _copy_o (
            OuterNode * outer_tree,
            int * I,
```

```
            int * J,
            float * Data,
            int * position )  [static]
```

Copia toda a matriz percorrendo a árvore externa e delegando para _copy_i.

Resulta em vetores paralelos I,J,Data de tamanho k com todas as entradas não nulas.

Parâmetros

outer_tree	raiz da árvore externa.
I	vetor de linhas.
J	vetor de colunas.
Data	vetor de valores.
position	ponteiro com posição inicial dos vetores.

4.1.2.9 _find_max_i()

```
static InnerNode * _find_max_i (
            InnerNode * tree )  [static]
```

Encontra o maior nó (mais à direita) em uma árvore interna.

Parâmetros

tree	raiz da árvore interna.

Retorna

Ponteiro para o nó máximo ou NULL se a árvore estiver vazia.

4.1.2.10 _find_max_o()

```
static OuterNode * _find_max_o (
            OuterNode * tree )  [static]
```

Encontra o maior nó (mais à direita) em uma árvore externa.

Parâmetros

tree	raiz da árvore externa.

Retorna

Ponteiro para o nó máximo ou NULL se a árvore estiver vazia.

4.1.2.11 _find_node_i()

```
static InnerNode * _find_node_i (
            InnerNode * tree,
            int search_key )  [static]
```

Busca nó na árvore interna pelo índice.

Parâmetros

tree	raiz da árvore interna.
search_key	fileira procurada.

Retorna

Ponteiro para o nó ou NULL se não existir.

4.1.2.12 _find_node_o()

```
static OuterNode * _find_node_o (
            OuterNode * tree,
            int search_key )  [static]
```

Busca nó na árvore externa pelo índice.

Parâmetros

tree	raiz da árvore externa.
search_key	fileira procurada.

Retorna

Ponteiro para o nó ou NULL se não existir.

4.1.2.13 _free_i_tree()

```
static void _free_i_tree (
            InnerNode * tree )  [static]
```

Libera recursivamente uma árvore interna.

Parâmetros

tree	raiz da árvore interna a ser desalocada (ou NULL).

4.1.2.14 _free_o_tree()

```
static void _free_o_tree (
            OuterNode * tree )  [static]
```

Libera recursivamente árvore externa e todas as internas associadas.

Parâmetros

tree	raiz da árvore externa a ser desalocada (ou NULL).

4.1.2.15 _height_i()

```
static int _height_i (
            InnerNode * tree )  [static]
```

Retorna altura de um nó da árvore interna (0 se nulo).

Parâmetros

tree	nó cuja altura é consultada.

4.1.2.16 _height_o()

```
static int _height_o (
            OuterNode * tree )  [static]
```

Retorna altura de um nó da árvore externa (0 se nulo).

Parâmetros

tree	nó cuja altura é consultada.

4.1.2.17 _insert_i()

```
static InnerNode * _insert_i (
            InnerNode * tree,
            int insert_key,
            float value,
            int * already_existed )  [static]
```

Insere ou atualiza um valor na árvore interna mantendo balanceamento AVL.

Parâmetros

tree	raiz da árvore interna.
insert_key	índice do elemento a inserir.
value	valor a armazenar.
already_existed	flag de saída: 1 se chave já existia.

Retorna

Nova raiz da subárvore após inserção/balanceamento.

4.1.2.18 _insert_o()

```
static OuterNode * _insert_o (
            OuterNode * tree,
            int insert_key,
            InnerNode * inner_tree )  [static]
```

Insere ou atualiza um valor na árvore externa mantendo balanceamento AVL.

Parâmetros

tree	raiz da árvore externa.
insert_key	fileira a inserir.
inner_tree	ponteiro para a árvore interna à ser inserida no nó.

Retorna

Nova raiz da subárvore após inserção/balanceamento.

4.1.2.19 _left_rotate_i()

```
static InnerNode * _left_rotate_i (
            InnerNode * tree )  [static]
```

Rotação à esquerda na árvore interna.

Rotação usada para o rebalanceamento da árvore AVL interna.

Parâmetros

tree	raiz desbalanceada.

Retorna

Nova raiz após rotação.

4.1.2.20 _left_rotate_o()

```
static OuterNode * _left_rotate_o (
            OuterNode * tree )  [static]
```

Rotação à esquerda na árvore externa.

Rotação usada para o rebalanceamento da árvore AVL externa.

Parâmetros

tree	raiz desbalanceada.

Retorna

Nova raiz após rotação.

4.1.2.21 _matmul_i_accumulate()

```
static AVLStatus _matmul_i_accumulate (
            InnerNode * inner_tree,
            int row,
            float A_value,
            AVLMatrix * C )  [static]
```

Acumula parcial de multiplicação de matrizes: atualiza linha de C com valores de A $*$ B.

Percorre a árvore interna correspondente à linha j de B (elementos B[j, c]) e, para cada coluna c ali presente, soma no resultado C a contribuição A[i, j] $*$ B[j, c]. O efeito é acumular na linha i de C todas as parcelas referentes a um valor específico A[i, j], de modo que, com chamadas que englobam todas as posições necessárias, a multiplicação de matriz ocorre corretamente.

Parâmetros

inner_tree	árvore interna da linha j de B (colunas não nulas).
row	linha alvo em C.
A_value	valor A[i, j] correspondente.
C	matriz resultado.

4.1.2.22 _max()

```
static int _max (
            int a,
            int b )  [static]
```

Retorna o maior entre dois inteiros.

Parâmetros

a	primeiro inteiro.
b	segundo inteiro.

4.1.2.23 _remove_i()

```
static InnerNode * _remove_i (
            InnerNode * tree,
            int remove_key )  [static]
```

Remove chave da árvore interna e rebalanceia.

Parâmetros

tree	raiz da árvore interna.
remove_key	coluna a remover.

Retorna

Nova raiz da subárvore após remoção.

4.1.2.24 _remove_o()

```
static OuterNode * _remove_o (
            OuterNode * tree,
            int remove_key )  [static]
```

Remove fileira e rebalanceia a árvore externa.

Só deve ser chamado para a remoção "estrutural" do nó, não limpa o conteúdo interno dele. Planejado para a remoção de um nó já vazio.

Parâmetros

tree	raiz da árvore externa.
remove_key	fileira a remover.

Retorna

Nova raiz da subárvore após remoção.

4.1.2.25 _right_rotate_i()

```
static InnerNode * _right_rotate_i (
            InnerNode * tree )  [static]
```

Rotação à direita na árvore interna.

Rotação usada para o rebalanceamento da árvore AVL interna.

Parâmetros

tree	raiz desbalanceada.

Retorna

Nova raiz após rotação.

4.1.2.26 _right_rotate_o()

```
static OuterNode * _right_rotate_o (
            OuterNode * tree )  [static]
```

Rotação à direita na árvore externa.

Rotação usada para o rebalanceamento da árvore AVL externa.

Parâmetros

tree	raiz desbalanceada.

Retorna

Nova raiz após rotação.

4.1.2.27 _scalar_multiply_i_tree()

```
static void _scalar_multiply_i_tree (
            InnerNode * tree,
            float a )  [static]
```

Multiplica todos os nós de uma árvore interna por um escalar.

Parâmetros

tree	raiz da árvore interna.
a	fator escalar.

4.1.2.28 _scalar_multiply_o_tree()

```
static void _scalar_multiply_o_tree (
            OuterNode * tree,
            float a )  [static]
```

Multiplica todos os valores armazenados na árvore externa (todas as fileiras) por um escalar.

Parâmetros

tree	raiz da árvore externa.
a	fator escalar.

4.1.2.29 _validate_indexes()

```
static AVLStatus _validate_indexes (
            AVLMatrix * matrix,
            int i,
            int j )  [static]
```

Verifica se índices i,j estão dentro dos limites da matriz.

Parâmetros

matrix	matriz onde a posição será utilizada.
i	índice de linha.
j	índice de coluna.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.1.2.30 _validate_matrix()

```
static AVLStatus _validate_matrix (
            AVLMatrix * matrix )  [static]
```

Valida ponteiro da matriz e dimensões armazenadas.

Parâmetros

matrix	ponteiro para a matriz a ser checada.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.1.2.31 _validate_same_dimensions()

```
static AVLStatus _validate_same_dimensions (
            AVLMatrix * A,
            AVLMatrix * B )  [static]
```

Garante que duas matrizes têm dimensões compatíveis (mesma ordem).

Parâmetros

A	primeira matriz.
B	segunda matriz.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.1.2.32 avl_status_string()

```
const char * avl_status_string (
            AVLStatus status )
```

Converte um código AVLStatus em mensagem textual.

Parâmetros

status	código de retorno.

Retorna

String com a mensagem.

4.1.2.33 create_matrix_avl()

```
AVLMatrix * create_matrix_avl (
            int n,
            int m )
```

Cria uma matriz vazia de dimensões n x m.

Parâmetros

n	número de linhas (não negativo).
m	número de colunas (não negativo).

Retorna

Ponteiro para nova matriz ou NULL em caso de parâmetros inválidos.

4.1.2.34 delete_element_avl()

```
AVLStatus delete_element_avl (
            AVLMatrix * matrix,
            int i,
            int j )
```

Remove um elemento da matriz.

Parâmetros

matrix	ponteiro para a matriz AVL.
i	índice da linha.
j	índice da coluna.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.1.2.35 free_matrix_avl()

```
void free_matrix_avl (
            AVLMatrix * matrix )
```

Libera a memória associada à uma matriz AVL.

Parâmetros

matrix	ponteiro para a matriz a ser destruída (ignorado se NULL).

4.1.2.36 get_element_avl()

```
AVLStatus get_element_avl (
            AVLMatrix * matrix,
            int i,
            int j,
            float * out_value )
```

Obtém o valor de um elemento da matriz.

Parâmetros

matrix	ponteiro para a matriz AVL.
i	índice da linha.
j	índice da coluna.
out_value	ponteiro onde o valor será escrito (0.0 se ausente).

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.1.2.37 insert_element_avl()

```
AVLStatus insert_element_avl (
            AVLMatrix * matrix,
            float value,
            int i,
            int j )
```

Insere ou atualiza um elemento na matriz.

Parâmetros

matrix	ponteiro para a matriz AVL.
value	valor a ser armazenado.
i	índice da linha.
j	índice da coluna.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.1.2.38 matrix_mul_avl()

```
AVLStatus matrix_mul_avl (
            AVLMatrix * A,
            AVLMatrix * B,
            AVLMatrix * C )
```

Calcula C = A ∗ B (sem suporte a in-place).

Parâmetros

A	matriz esquerda.
B	matriz direita.
C	matriz resultado pré-alocada com dimensões corretas.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.1.2.39 scalar_mul_avl()

```
AVLStatus scalar_mul_avl (
            AVLMatrix * A,
            AVLMatrix * B,
            float a )
```

Calcula B = a ∗ A (possivelmente in-place).

Parâmetros

A	matriz de entrada.
B	matriz de saída (pode ser a mesma que A).
a	fator escalar.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.1.2.40 sum_avl()

```
AVLStatus sum_avl (
            AVLMatrix * A,
            AVLMatrix * B,
            AVLMatrix * C )
```

Calcula C = A + B.

Parâmetros

A	primeira matriz de entrada.
B	segunda matriz de entrada.
C	matriz resultado.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.1.2.41 transpose_avl()

```
AVLStatus transpose_avl (
            AVLMatrix * matrix )
```

Transpõe a matriz.

Parâmetros

matrix	ponteiro para a matriz AVL.

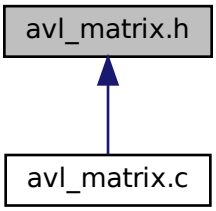
Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.2 Referência ao ficheiro avl_matrix.h

Estruturas e operações para matrizes esparsas baseadas em árvores AVL.

Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Estruturas de Dados

- struct InnerNode

  Nó da árvore interna (contém os elementos).
- struct OuterNode

  Nó da árvore externa (linha ou coluna da matriz).
- struct AVLMatrix

  Representação de uma matriz esparsa usando duas árvores AVL.

Definições de tipos

- typedef struct InnerNode InnerNode

  Nó da árvore interna (contém os elementos).
- typedef struct OuterNode OuterNode

  Nó da árvore externa (linha ou coluna da matriz).
- typedef struct AVLMatrix AVLMatrix

  Representação de uma matriz esparsa usando duas árvores AVL.

Enumerações

- enum AVLStatus {
  AVL_STATUS_OK = 0 , AVL_STATUS_NOT_FOUND = 1 , AVL_ERROR_NULL_MATRIX = -1 ,
  AVL_ERROR_OUT_OF_BOUNDS = -2 ,
  AVL_ERROR_DIMENSION_MISMATCH = -3 , AVL_ERROR_INVALID_ARGUMENT = -4 , AVL_ERROR_NOT_IMPLEMENTED
  = -5 }

  Códigos de retorno das operações na matriz AVL.

Funções

- AVLStatus get_element_avl (AVLMatrix ∗matrix, int i, int j, float ∗out_value)

  Obtém o valor de um elemento da matriz.
- AVLStatus insert_element_avl (AVLMatrix ∗matrix, float value, int i, int j)

  Insere ou atualiza um elemento na matriz.
- AVLStatus delete_element_avl (AVLMatrix ∗matrix, int i, int j)

  Remove um elemento da matriz.
- AVLStatus transpose_avl (AVLMatrix ∗matrix)

  Transpõe a matriz.
- AVLStatus scalar_mul_avl (AVLMatrix ∗A, AVLMatrix ∗B, float a)

  Calcula B = a ∗ A (possivelmente in-place).
- AVLStatus sum_avl (AVLMatrix ∗A, AVLMatrix ∗B, AVLMatrix ∗C)

  Calcula C = A + B.
- AVLStatus matrix_mul_avl (AVLMatrix ∗A, AVLMatrix ∗B, AVLMatrix ∗C)

  Calcula C = A ∗ B (sem suporte a in-place).
- const char ∗ avl_status_string (AVLStatus status)

  Converte um código AVLStatus em mensagem textual.
- AVLMatrix ∗ create_matrix_avl (int n, int m)

  Cria uma matriz vazia de dimensões n x m.
- void free_matrix_avl (AVLMatrix ∗matrix)

  Liberta a memória associada à uma matriz AVL.

4.2.1 Descrição detalhada

Estruturas e operações para matrizes esparsas baseadas em árvores AVL.

4.2.2 Documentação dos tipos

4.2.2.1 AVLMatrix

```
typedef struct AVLMatrix AVLMatrix
```

Representação de uma matriz esparsa usando duas árvores AVL.

A árvore main_root guarda os elementos no formato linha->coluna e a transposed_root armazena a matriz transposta para facilitar operações.

4.2.2.2 InnerNode

```
typedef struct InnerNode InnerNode
```

Nó da árvore interna (contém os elementos).

Armazena o índice da linha ou coluna, o valor do elemento não-nulo, os ponteiros dos filhos da árvore AVL interna e a altura relativa ao nó.

4.2.2.3 OuterNode

```
typedef struct OuterNode OuterNode
```

Nó da árvore externa (linha ou coluna da matriz).

Cada nó representa uma linha ou coluna da matriz e aponta para a árvore interna contendo os elementos não-nulos da linha ou coluna. Armazena o índice dessa linha ou coluna, os ponteiros dos filhos da árvore AVL externa e a altura relativa ao nó.

4.2.3 Documentação dos valores da enumeração

4.2.3.1 AVLStatus

```
enum AVLStatus
```

Códigos de retorno das operações na matriz AVL.

Valores de enumerações

AVL_STATUS_OK	Operação concluída com sucesso.
AVL_STATUS_NOT_FOUND	Elemento solicitado não existe.
AVL_ERROR_NULL_MATRIX	Ponteiro de matriz nulo.
AVL_ERROR_OUT_OF_BOUNDS	Índices fora dos limites da matriz.
AVL_ERROR_DIMENSION_MISMATCH	Incompatibilidade de dimensões entre matrizes.
AVL_ERROR_INVALID_ARGUMENT	Parâmetro inválido.
AVL_ERROR_NOT_IMPLEMENTED	Funcionalidade ainda não implementada.

4.2.4 Documentação das funções

4.2.4.1 avl_status_string()

```
const char * avl_status_string (
            AVLStatus status )
```

Converte um código AVLStatus em mensagem textual.

Parâmetros

status	código de retorno.

Retorna

String com a mensagem.

4.2.4.2 create_matrix_avl()

```
AVLMatrix * create_matrix_avl (
            int n,
            int m )
```

Cria uma matriz vazia de dimensões n x m.

Parâmetros

n	número de linhas (não negativo).
m	número de colunas (não negativo).

Retorna

Ponteiro para nova matriz ou NULL em caso de parâmetros inválidos.

4.2.4.3 delete_element_avl()

```
AVLStatus delete_element_avl (
            AVLMatrix * matrix,
            int i,
            int j )
```

Remove um elemento da matriz.

Parâmetros

matrix	ponteiro para a matriz AVL.
i	índice da linha.
j	índice da coluna.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.2.4.4 free_matrix_avl()

```
void free_matrix_avl (
            AVLMatrix * matrix )
```

Libera a memória associada à uma matriz AVL.

Parâmetros

matrix	ponteiro para a matriz a ser destruída (ignorado se NULL).

4.2.4.5 get_element_avl()

```
AVLStatus get_element_avl (
            AVLMatrix * matrix,
            int i,
            int j,
            float * out_value )
```

Obtém o valor de um elemento da matriz.

Parâmetros

matrix	ponteiro para a matriz AVL.
i	índice da linha.
j	índice da coluna.
out_value	ponteiro onde o valor será escrito (0.0 se ausente).

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.2.4.6 insert_element_avl()

```
AVLStatus insert_element_avl (
            AVLMatrix * matrix,
            float value,
            int i,
            int j )
```

Insere ou atualiza um elemento na matriz.

Parâmetros

matrix	ponteiro para a matriz AVL.
value	valor a ser armazenado.
i	índice da linha.
j	índice da coluna.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.2.4.7 matrix_mul_avl()

```
AVLStatus matrix_mul_avl (
            AVLMatrix * A,
            AVLMatrix * B,
            AVLMatrix * C )
```

Calcula C = A ∗ B (sem suporte a in-place).

Parâmetros

A	matriz esquerda.
B	matriz direita.
C	matriz resultado pré-alocada com dimensões corretas.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.2.4.8 scalar_mul_avl()

```
AVLStatus scalar_mul_avl (
            AVLMatrix * A,
            AVLMatrix * B,
            float a )
```

Calcula B = a ∗ A (possivelmente in-place).

Parâmetros

A	matriz de entrada.
B	matriz de saída (pode ser a mesma que A).
a	fator escalar.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.2.4.9 sum_avl()

```
AVLStatus sum_avl (
            AVLMatrix * A,
            AVLMatrix * B,
            AVLMatrix * C )
```

Calcula C = A + B.

Parâmetros

A	primeira matriz de entrada.
B	segunda matriz de entrada.
C	matriz resultado.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.2.4.10 transpose_avl()

```
AVLStatus transpose_avl (
            AVLMatrix * matrix )
```

Transpõe a matriz.

Parâmetros

matrix	ponteiro para a matriz AVL.

Retorna

Código AVLStatus indicando sucesso ou motivo da falha.

4.3 avl_matrix.h

Ir para a documentação deste ficheiro.

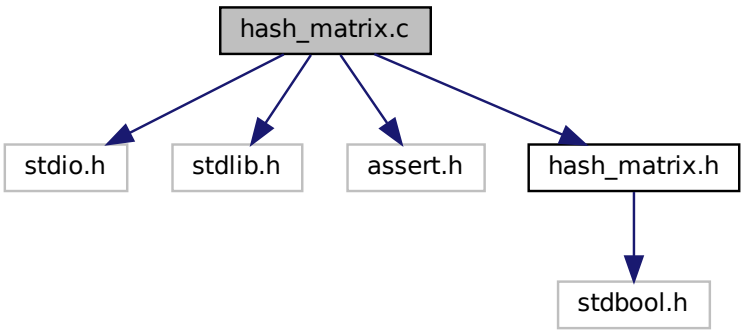
```
1 #pragma once
2
15 typedef struct InnerNode{
16     int key;
17     float data;
18     struct InnerNode* left;
19     struct InnerNode* right;
20     int height;
21 } InnerNode;
22
32 typedef struct OuterNode{
33     int key;
34     struct InnerNode* inner_tree;
35     struct OuterNode* left;
36     struct OuterNode* right;
```

```
37     int height;
38 } OuterNode;
39
43 typedef enum {
44     AVL_STATUS_OK = 0,
45     AVL_STATUS_NOT_FOUND = 1,
46     AVL_ERROR_NULL_MATRIX = -1,
47     AVL_ERROR_OUT_OF_BOUNDS = -2,
48     AVL_ERROR_DIMENSION_MISMATCH = -3,
49     AVL_ERROR_INVALID_ARGUMENT = -4,
50     AVL_ERROR_NOT_IMPLEMENTED = -5
51 } AVLStatus;
52
59 typedef struct AVLMatrix{
60     struct OuterNode* main_root;
61     struct OuterNode* transposed_root;
62     int k;
63     int n;
64     int m;
65 } AVLMatrix;
66
76 AVLStatus get_element_avl(AVLMatrix* matrix, int i, int j, float* out_value);
77
87 AVLStatus insert_element_avl(AVLMatrix* matrix, float value, int i, int j);
88
97 AVLStatus delete_element_avl(AVLMatrix* matrix, int i, int j);
98
105 AVLStatus transpose_avl(AVLMatrix* matrix);
106
115 AVLStatus scalar_mul_avl(AVLMatrix* A, AVLMatrix* B, float a);
116
125 AVLStatus sum_avl(AVLMatrix* A, AVLMatrix* B, AVLMatrix* C);
126
135 AVLStatus matrix_mul_avl(AVLMatrix* A, AVLMatrix* B, AVLMatrix* C);
136
143 const char* avl_status_string(AVLStatus status);
144
152 AVLMatrix* create_matrix_avl(int n, int m);
153
159 void free_matrix_avl(AVLMatrix* matrix);
```

4.4 Referência ao ficheiro hash_matrix.c

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "hash_matrix.h"
```

Diagrama de dependências de inclusão para hash_matrix.c:



Macros

- #define INITIAL_CAPACITY 16
- #define LOAD_FACTOR_UPPER 0.75
- #define LOAD_FACTOR_LOWER 0.25

Funções

- static void _allocation_fail ()

  Encerramento imediato em caso de falha de alocação.
- unsigned int hash (int row, int column, int capacity)

  retorna um hash dados inteiros de linha, coluna, e capacidade.
- HashStatus resize (HashMatrix *matrix)

  Redimensiona a tabela de espalhamento da matriz hash.
- HashMatrix * create_hash_matrix (int rows, int columns)

  Cria uma nova matriz hash com dimensões especificadas.
- float get_element_hash (HashMatrix *matrix, int row, int column)

  Obtém o valor de um elemento da matriz hash.
- HashStatus set_element_hash (HashMatrix *matrix, int row, int column, float data)

  Define o valor de um elemento na matriz hash.
- HashMatrix * matrix_multiplication_hash (HashMatrix *A, HashMatrix *B)

  Multiplica duas matrizes hash.
- HashMatrix * matrix_addition_hash (HashMatrix *A, HashMatrix *B)

  Soma duas matrizes hash.
- HashMatrix * matrix_scalar_multiplication_hash (HashMatrix *A, float scalar)

  Multiplica uma matriz hash por um escalar.
- HashStatus transpose_hash (HashMatrix *matrix)

  Transpõe a matriz hash.
- HashStatus free_hash_matrix (HashMatrix *matrix)

  Liberta a memória alocada para a matriz hash.

4.4.1 Documentação das macros

4.4.1.1 INITIAL_CAPACITY

```
#define INITIAL_CAPACITY 16
```

4.4.1.2 LOAD_FACTOR_LOWER

```
#define LOAD_FACTOR_LOWER 0.25
```

4.4.1.3 LOAD_FACTOR_UPPER

```
#define LOAD_FACTOR_UPPER 0.75
```

4.4.2 Documentação das funções

4.4.2.1 _allocation_fail()

```
static void _allocation_fail ( )  [static]
```

Encerramento imediato em caso de falha de alocação.

Imprime uma mensagem de erro em stderr e aborta o processo usando EXIT_FAILURE.

4.4.2.2 create_hash_matrix()

```
HashMatrix * create_hash_matrix (
            int rows,
            int columns )
```

Cria uma nova matriz hash com dimensões especificadas.

Parâmetros

rows	número de linhas.
columns	número de colunas.

Retorna

Ponteiro para a nova matriz hash.

4.4.2.3 free_hash_matrix()

```
HashStatus free_hash_matrix (
            HashMatrix * matrix )
```

Libera a memória alocada para a matriz hash.

Parâmetros

matrix	ponteiro para a matriz hash a ser liberada.

Retorna

Código HashStatus indicando sucesso ou motivo da falha.

4.4.2.4 get_element_hash()

```
float get_element_hash (
            HashMatrix * matrix,
            int row,
            int column )
```

Obtém o valor de um elemento da matriz hash.

Parâmetros

matrix	ponteiro para a matriz hash.
row	índice de linha.
column	índice de coluna.

Retorna

valor do elemento na posição (row, column) ou código de erro.

4.4.2.5 hash()

```
unsigned int hash (
            int row,
            int column,
            int capacity )
```

retorna um hash dados inteiros de linha, coluna, e capacidade.

Parâmetros

row	índice de linha.
column	índice de coluna.
capacity	capacidade total da tabela de espalhamento.

Retorna

o hash calculado.

4.4.2.6 matrix_addition_hash()

```
HashMatrix * matrix_addition_hash (
             HashMatrix * A,
             HashMatrix * B )
```

Soma duas matrizes hash.

Parâmetros

A	ponteiro para a matriz hash A.
B	ponteiro para a matriz hash B.

Retorna

Ponteiro para a matriz resultante C.

4.4.2.7 matrix_multiplication_hash()

```
HashMatrix * matrix_multiplication_hash (
             HashMatrix * A,
             HashMatrix * B )
```

Multiplica duas matrizes hash.

Parâmetros

A	ponteiro para a matriz hash A.
B	ponteiro para a matriz hash B.

Retorna

Ponteiro para a matriz resultante C.

4.4.2.8 matrix_scalar_multiplication_hash()

```
HashMatrix * matrix_scalar_multiplication_hash (
             HashMatrix * A,
             float scalar )
```

Multiplica uma matriz hash por um escalar.

Parâmetros

A	ponteiro para a matriz hash A.
scalar	valor escalar.

Retorna

Ponteiro para a nova matriz resultante B.

4.4.2.9 resize()

```
HashStatus resize (
            HashMatrix * matrix )
```

Redimensiona a tabela de espalhamento da matriz hash.

Parâmetros

matrix	ponteiro para a matriz hash a ser redimensionada.

Retorna

Código HashStatus indicando sucesso ou motivo da falha.

4.4.2.10 set_element_hash()

```
HashStatus set_element_hash (
            HashMatrix * matrix,
            int row,
            int column,
            float data )
```

Define o valor de um elemento na matriz hash.

Parâmetros

matrix	ponteiro para a matriz hash.
row	índice de linha.
column	índice de coluna.
data	valor a ser definido.

Retorna

Código HashStatus indicando sucesso ou motivo da falha.

4.4.2.11 transpose_hash()

```
HashStatus transpose_hash (
            HashMatrix * matrix )
```

Transpõe a matriz hash.

Parâmetros

matrix	ponteiro para a matriz hash.

Retorna

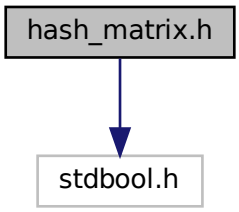
Código HashStatus indicando sucesso ou motivo da falha.

4.5 Referência ao ficheiro hash_matrix.h

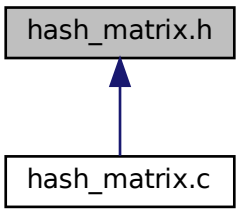
Estruturas e operações para matrizes esparsas baseadas em hash.

```
#include <stdbool.h>
```

Diagrama de dependências de inclusão para hash_matrix.h:



Este grafo mostra quais são os ficheiros que incluem directamente ou indirectamente este ficheiro:



Estruturas de Dados

- struct Node

  Nó que contém os elementos em cada posição da tabela de espalhamento.
- struct HashMatrix

  Representação de uma matriz esparsa usando tabela de espalhamento (hash table).

Definições de tipos

- typedef struct Node Node

  Nó que contém os elementos em cada posição da tabela de espalhamento.
- typedef struct HashMatrix HashMatrix

  Representação de uma matriz esparsa usando tabela de espalhamento (hash table).

Enumerações

- enum HashStatus {
  HASH_STATUS_OK = 0 , HASH_STATUS_NOT_FOUND = 1 , HASH_ERROR_NULL_MATRIX = -1 ,
  HASH_ERROR_OUT_OF_BOUNDS = -2 ,
  HASH_ERROR_DIMENSION_MISMATCH = -3 , HASH_ERROR_INVALID_ARGUMENT = -4 , HASH_ERROR_NOT_IMPLEM
  = -5 }

  Códigos de retorno das operações na matriz hash.

Funções

- HashMatrix ∗ create_hash_matrix (int rows, int columns)

  Cria uma nova matriz hash com dimensões especificadas.
- float get_element_hash (HashMatrix ∗matrix, int row, int column)

  Obtém o valor de um elemento da matriz hash.
- HashStatus set_element_hash (HashMatrix ∗matrix, int row, int column, float data)

  Define o valor de um elemento na matriz hash.
- HashMatrix ∗ matrix_multiplication_hash (HashMatrix ∗A, HashMatrix ∗B)

  Multiplica duas matrizes hash.
- HashMatrix ∗ matrix_addition_hash (HashMatrix ∗A, HashMatrix ∗B)

  Soma duas matrizes hash.
- HashMatrix ∗ matrix_scalar_multiplication_hash (HashMatrix ∗A, float scalar)

  Multiplica uma matriz hash por um escalar.
- HashStatus transpose_hash (HashMatrix ∗matrix)

  Transpõe a matriz hash.
- HashStatus free_hash_matrix (HashMatrix ∗matrix)

  Liberta a memória alocada para a matriz hash.

4.5.1 Descrição detalhada

Estruturas e operações para matrizes esparsas baseadas em hash.

4.5.2 Documentação dos tipos

4.5.2.1 HashMatrix

```
typedef struct HashMatrix HashMatrix
```

Representação de uma matriz esparsa usando tabela de espalhamento (hash table).

Armazena os buckets da tabela de espalhamento, capacidade total, número de elementos não nulos, dimensões da matriz (linhas e colunas) e flag de transposição.

4.5.2.2 Node

```
typedef struct Node Node
```

Nó que contém os elementos em cada posição da tabela de espalhamento.

Armazena o índice da linha ou coluna, o valor do elemento não-nulo, e o ponteiro para o próximo nó na lista encadeada do elemento com hash colidido.

4.5.3 Documentação dos valores da enumeração

4.5.3.1 HashStatus

```
enum HashStatus
```

Códigos de retorno das operações na matriz hash.

Valores de enumerações

HASH_STATUS_OK	Operação concluída com sucesso.
HASH_STATUS_NOT_FOUND	Elemento solicitado não existe.
HASH_ERROR_NULL_MATRIX	Ponteiro de matriz nulo.
HASH_ERROR_OUT_OF_BOUNDS	Índices fora dos limites da matriz.
HASH_ERROR_DIMENSION_MISMATCH	Incompatibilidade de dimensões entre matrizes.
HASH_ERROR_INVALID_ARGUMENT	Parâmetro inválido.
HASH_ERROR_NOT_IMPLEMENTED	Funcionalidade ainda não implementada.

4.5.4 Documentação das funções

4.5.4.1 create_hash_matrix()

```
HashMatrix * create_hash_matrix (
            int rows,
            int columns )
```

Cria uma nova matriz hash com dimensões especificadas.

Parâmetros

rows	número de linhas.
columns	número de colunas.

Retorna

Ponteiro para a nova matriz hash.

4.5.4.2 free_hash_matrix()

```
HashStatus free_hash_matrix (
            HashMatrix * matrix )
```

Libera a memória alocada para a matriz hash.

Parâmetros

matrix	ponteiro para a matriz hash a ser liberada.

Retorna

Código HashStatus indicando sucesso ou motivo da falha.

4.5.4.3 get_element_hash()

```
float get_element_hash (
            HashMatrix * matrix,
            int row,
            int column )
```

Obtém o valor de um elemento da matriz hash.

Parâmetros

matrix	ponteiro para a matriz hash.
row	índice de linha.
column	índice de coluna.

Retorna

valor do elemento na posição (row, column) ou código de erro.

4.5.4.4 matrix_addition_hash()

```
HashMatrix * matrix_addition_hash (
            HashMatrix * A,
            HashMatrix * B )
```

Soma duas matrizes hash.

Parâmetros

A	ponteiro para a matriz hash A.
B	ponteiro para a matriz hash B.

Retorna

Ponteiro para a matriz resultante C.

4.5.4.5 matrix_multiplication_hash()

```
HashMatrix * matrix_multiplication_hash (
             HashMatrix * A,
             HashMatrix * B )
```

Multiplica duas matrizes hash.

Parâmetros

A	ponteiro para a matriz hash A.
B	ponteiro para a matriz hash B.

Retorna

Ponteiro para a matriz resultante C.

4.5.4.6 matrix_scalar_multiplication_hash()

```
HashMatrix * matrix_scalar_multiplication_hash (
             HashMatrix * A,
             float scalar )
```

Multiplica uma matriz hash por um escalar.

Parâmetros

A	ponteiro para a matriz hash A.
scalar	valor escalar.

Retorna

Ponteiro para a nova matriz resultante B.

4.5.4.7 set_element_hash()

```
HashStatus set_element_hash (
            HashMatrix * matrix,
            int row,
            int column,
            float data )
```

Define o valor de um elemento na matriz hash.

Parâmetros

matrix	ponteiro para a matriz hash.
row	índice de linha.
column	índice de coluna.
data	valor a ser definido.

Retorna

Código HashStatus indicando sucesso ou motivo da falha.

4.5.4.8 transpose_hash()

```
HashStatus transpose_hash (
            HashMatrix * matrix )
```

Transpõe a matriz hash.

Parâmetros

matrix	ponteiro para a matriz hash.

Retorna

Código HashStatus indicando sucesso ou motivo da falha.

4.6 hash_matrix.h

Ir para a documentação deste ficheiro.

```
1 #pragma once
2 #include <stdbool.h>
3
15 typedef struct Node {
16     int row, column;
17     float data;
18     struct Node* next;
19 }Node;
20
27 typedef struct HashMatrix{
28     Node **buckets;
29     int capacity, count, rows, columns; //capacity = tamanho total do hash, count = num de elementos não
       nulos
```

```
30     bool is_transposed;
31 }HashMatrix;
32
36 typedef enum {
37     HASH_STATUS_OK = 0,
38     HASH_STATUS_NOT_FOUND = 1,
39     HASH_ERROR_NULL_MATRIX = -1,
40     HASH_ERROR_OUT_OF_BOUNDS = -2,
41     HASH_ERROR_DIMENSION_MISMATCH = -3,
42     HASH_ERROR_INVALID_ARGUMENT = -4,
43     HASH_ERROR_NOT_IMPLEMENTED = -5
44 } HashStatus;
45
53 HashMatrix* create_hash_matrix(int rows, int columns);
54
63 float get_element_hash(HashMatrix* matrix, int row, int column);
64
74 HashStatus set_element_hash(HashMatrix* matrix, int row, int column, float data);
75
83 HashMatrix* matrix_multiplication_hash(HashMatrix* A, HashMatrix* B);
84
92 HashMatrix* matrix_addition_hash(HashMatrix* A, HashMatrix* B);
93
101 HashMatrix* matrix_scalar_multiplication_hash(HashMatrix* A, float scalar);
102
109 HashStatus transpose_hash(HashMatrix* matrix);
110
117 HashStatus free_hash_matrix(HashMatrix* matrix);
```

Índice