Maximal-Assignment-C

Generated by Doxygen 1.10.0

Chapter 1

Maximal-Assignment-C

1.1 Overview

This repository houses a C library designed to solve assignment problems efficiently. Assignment problems involve finding the optimal assignment of tasks to resources with specific constraints. The library includes implementations of well-known algorithms like the Hungarian, Brute Force, and Backtrack algorithms. Developed initially for academic purposes, this library showcases algorithmic proficiency and serves as a useful reference for understanding and implementing assignment problem-solving algorithms.

1.2 Assignment Problem

In the context of this library, an assignment problem refers to the task of maximizing the sum of elements in a matrix by selecting one element from each row and column without repeating any row or column. The algorithms provided aim to find the optimal solution to such problems.

1.3 Usage

The library is provided as read-only and is intended for demonstration purposes. To utilize the algorithms in your own projects, include the necessary header files in your C project and link against the provided object files.

1.4 Algorithms Included

- **Hungarian Algorithm** : This algorithm efficiently solves assignment problems by finding the optimal assignment using a series of augmenting paths.
- **Greedy Algorithm** : While not the most effective, the Greedy algorithm makes the best local choices without reconsidering previous decisions, potentially resulting in a suboptimal solution.
- **Backtrack Algorithm** : This algorithm employs a systematic search strategy to explore the solution space, backtracking when a dead-end is reached and continuing until the optimal assignment is found.

1.5 How to Use

To use this library in your projects, follow these steps:

1. Clone the repository or download the source files.
2. Include the necessary header files (`hungarian.h`, `greedy.h`, `backtrack.h`) in your C project.
3. Link against the provided object files (`hungarian.o`, `brute_force.o`, `backtrack.o`).

1.6 Known Issues

In certain cases, the Hungarian Algorithm may encounter infinite loop errors, leading to difficulties in producing a solution. These errors typically arise when the algorithm struggles to converge on a solution due to complex matrix structures or ambiguous rules governing zero-covering.

1.7 Contributing

This project is read-only and not open for contributions. It was developed as part of academic coursework and is provided as a showcase of work done during university studies.

1.8 License

This project is not licensed for redistribution or modification. It is provided solely as a showcase of academic work and for educational purposes.

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 ExploreParams Struct Reference

Parameters passed to the recursive function of the "Backtrack" algorithm.

`#include <backtrack.h>`

Public Attributes

- Matrix * **matrix**
- int **currentRow**
- int **currentSum**
- int * **maxSum**
- int * **usedRows**
- int * **usedColumns**
- int * **selectionCount**
- SelectedElement * **selectionValues**

4.1.1 Detailed Description

Parameters passed to the recursive function of the "Backtrack" algorithm.

The documentation for this struct was generated from the following file:

- MatrixMatch/backtrack.h

4.2 Matrix Struct Reference

A matrix.

`#include <matrix_core.h>`

Public Attributes

- MatrixRowNode * **head**
- int **width**
- int **height**

4.2.1 Detailed Description

A matrix.

The matrix contains the `head`, which is a pointer to the first row of the matrix, and the size of the matrix.

The documentation for this struct was generated from the following file:

- MatrixMatch/matrix_core.h

4.3 MatrixElement Struct Reference

Represents an element of the matrix.

`#include <matrix_core.h>`

Public Attributes

- int **value**
- int **column**
- struct MatrixElement * **nextCol**

4.3.1 Detailed Description

Represents an element of the matrix.

Each element contains its integer value, its current column, and a pointer to the next element in the same row.

The documentation for this struct was generated from the following file:

- MatrixMatch/matrix_core.h

4.4 MatrixRowNode Struct Reference

Represents a row as a node of the matrix.

`#include <matrix_core.h>`

Public Attributes

- MatrixElement * **row**
- struct MatrixRowNode * **nextRow**

4.4.1 Detailed Description

Represents a row as a node of the matrix.

A row of the matrix containing a pointer to the first element of the row and a pointer to the next row.

The documentation for this struct was generated from the following file:

- MatrixMatch/matrix_core.h

4.5 SelectedElement Struct Reference

Structure to store elements chosen by the backtrack algorithm. Each element contains the current row and column in the matrix, and its integer value.

`#include <backtrack.h>`

Public Attributes

- int **row**
- int **col**
- int **value**

4.5.1 Detailed Description

Structure to store elements chosen by the backtrack algorithm. Each element contains the current row and column in the matrix, and its integer value.

The documentation for this struct was generated from the following file:

- MatrixMatch/backtrack.h

4.6 Zero Struct Reference

Structure representing a zero in the matrix.

`#include <hungarian.h>`

Public Attributes

- int **row**
- int **col**

4.6.1 Detailed Description

Structure representing a zero in the matrix.

The documentation for this struct was generated from the following file:

- MatrixMatch/hungarian.h

Chapter 5

File Documentation

5.1 MatrixMatch/backtrack.c File Reference

Implementation of the "Backtrack" algorithm.

```
#include "backtrack.h"
#include <stdio.h>
#include <stdlib.h>
#include "error_codes.h"
#include "matrix_core.h"
```

Functions

- int BacktrackAlgorithm (Matrix *matrix, int *maxSum, int *selectionCount, SelectedElement **selection↩ Values)

  "Backtrack" algorithm, a solution for calculating the maximum possible sum of integers from a matrix of integers with any dimensions, so that none of the selected integers share the same row or column.

5.1.1 Detailed Description

Implementation of the "Backtrack" algorithm.

This file contains the implementation of the "Backtrack" algorithm, a solution for calculating the maximum possible sum of integers from a matrix of integers with any dimensions, so that none of the selected integers share the same row or column.

Author

Enrique Rodrigues

Date

11.03.2024

Copyright

5.1.2 Function Documentation

5.1.2.1 BacktrackAlgorithm()

```
int BacktrackAlgorithm (
            Matrix * matrix,
            int * maxSum,
            int * selectionCount,
            SelectedElement ** selectionValues )
```

"Backtrack" algorithm, a solution for calculating the maximum possible sum of integers from a matrix of integers with any dimensions, so that none of the selected integers share the same row or column.

Parameters

matrix	- The matrix.
maxSum	- Maximum total sum possible.
selectionCount	- Number of elements chosen for the result.
maxSelection	- Array containing the chosen values.

Return values

`INVALID_MATRIX_OR_INDICES`	- The matrix or the provided indices are invalid.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure for the new matrix element.
`SUCCESS`	- Operation successful.

5.2 MatrixMatch/backtrack.h File Reference

Header file for the "Backtrack" algorithm.

```
#include "matrix_core.h"
```

Classes

- struct SelectedElement

  Structure to store elements chosen by the backtrack algorithm. Each element contains the current row and column in the matrix, and its integer value.
- struct ExploreParams

  Parameters passed to the recursive function of the "Backtrack" algorithm.

Typedefs

- typedef struct SelectedElement **SelectedElement**
- typedef struct ExploreParams **ExploreParams**

Functions

- __declspec (dllexport) int BacktrackAlgorithm(Matrix *matrix

  "Backtrack" algorithm, a solution for calculating the maximum possible sum of integers from a matrix of integers with any dimensions, so that none of the selected integers share the same row or column.

Variables

- int * **maxSum**
- int int * **selectionCount**
- int int SelectedElement ** **maxSelection**

5.2.1 Detailed Description

Header file for the "Backtrack" algorithm.

This header file contains function declarations and data structures related to the "Backtrack" algorithm.

Author

Enrique Rodrigues

Date

12.03.2024

Copyright

5.2.2 Function Documentation

5.2.2.1 __declspec()

```
__declspec (
            dllexport  )
```

"Backtrack" algorithm, a solution for calculating the maximum possible sum of integers from a matrix of integers with any dimensions, so that none of the selected integers share the same row or column.

Deletes a column from the matrix.

Deletes a row from the matrix.

Inserts a column at the end of a row in the matrix.

Inserts a row into the matrix.

Creates a matrix from data in a file.

Fills a matrix with data from a file.

Obtains the size of a matrix stored in a file.

Get an element in a specific column.

Get the row node at a specific index.

Free allocated memory of a matrix.

Create and add an element to the end of a row in the matrix.

Create a matrix of a given size filled with default values.

Initialize a row node of the matrix with a given size, filling it with default values.

Initialize a matrix row with default values.

Create a new element with a given value.

Parameters

matrix	- The matrix.
maxSum	- Maximum total sum possible.
selectionCount	- Number of elements chosen for the result.
maxSelection	- Array containing the chosen values.

Return values

`INVALID_MATRIX_OR_INDICES`	- The matrix or the provided indices are invalid.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure for the new matrix element.
`SUCCESS`	- Operation successful.

Parameters

value	- The value to be stored in the `MatrixElement`.

Return values

-	A pointer to the new element, or NULL in case of memory allocation error.

Parameters

width	- Row size (number of columns).

Return values

-	Pointer to the last element of the row, or NULL in case of memory allocation error.

Parameters

width	- Row size (number of columns).

Return values

-	A pointer to the new row node, or NULL in case of memory allocation error.

Parameters

width	- The number of columns of the matrix.
height	- The number of rows of the matrix.
matrix	- Matrix to hold the matrix data.

Return values

`INVALID_MATRIX_OR_INDICES`	- The matrix or indices provided are invalid.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure for the new matrix element.

Return values

`*SUCESS*`	- Operation successful.

Parameters

head	- The first element of the row.
value	- The value the element will have.

Return values

-	`NULL` in case of error or the pointer to the beginning of the row.

Parameters

matrix	- The matrix to be freed.
matrix	- The matrix.
rowIndex	- The index of the desired row.

Return values

-	The corresponding row node if the index is valid, NULL otherwise.

Parameters

rowNode	- The row node of the matrix.
colIndex	- The column index of the matrix.

Return values

-	The matrix element.

Parameters

filename	- The name of the file containing the matrix.
width	- A variable of type `int` that will hold the number of columns of the matrix.
height	- A variable of type `int` that will hold the number of rows of the matrix.

Return values

`*CANNOT_OPEN_FILE*`	- Failure to open the file.
`*SUCCESS*`	- Operation successful.

Parameters

filename	- The name of the file.
matrix	- The matrix to be filled with data from the file.

Return values

`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure.
`CANNOT_OPEN_FILE`	- Failure to open the file.
`FILE_READ_ERROR`	- Error reading data from the file.
`SUCCESS`	- Operation successful.

Parameters

filename	- The name of the file.
matrix	- Matrix that will contain the data of the matrix.

Return values

`CANNOT_OPEN_FILE`	- Failure to open the file.
`INVALID_MATRIX_OR_INDICES`	- The matrix or the provided indices are invalid.
`MEMORY_ALLOCATION_FAILURE`	- Failure to allocate memory for the new matrix element.
`FILE_READ_ERROR`	- Error reading data from the file.
`SUCCESS`	- Operation successful.

Parameters

matrix	- The matrix where we will insert a new row.
newRow	- Array with the values of the new row.
sizeArray	- Size of the array.

Return values

`OUT_OF_BOUNDS`	- Size of the array is different from the size of the matrix.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure.
`SUCCESS`	- Operation successful.

Parameters

matrix	- The matrix where we will insert the new column.
newColumn	- Array with the values of the new column.
sizeArray	- Size of the array.

Return values

`OUT_OF_BOUNDS`	- Size of the array is different from the size of the matrix.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure.
`SUCCESS`	- Operation successful.

Parameters

matrix	- The matrix.

Parameters

rowIndex	- The row of the matrix to be deleted.

Return values

`OUT_OF_BOUNDS`	- Position does not exist.
`SUCCESS`	- Operation successful.

Parameters

matrix	- The matrix.
colIndex	- The column to be deleted.

Return values

`OUT_OF_BOUNDS`	- Position does not exist.
`SUCCESS`	- Operation successful.

5.3 backtrack.h

Go to the documentation of this file.

```
00001
00010 #ifndef BACKTRACK_H
00011 #define BACKTRACK_H
00012
00013 #include "matrix_core.h"
00014
00021 typedef struct SelectedElement {
00022   int row;    // Row of the element
00023   int col;    // Column of the element
00024   int value;  // Value of the element
00025 } SelectedElement;
00026
00032 typedef struct ExploreParams {
00033   Matrix* matrix;                    // The matrix
00034   int currentRow;                    // Current row
00035   int currentSum;                    // Current sum
00036   int* maxSum;                       // Total maximum sum
00037   int* usedRows;                     // Used rows
00038   int* usedColumns;                  // Used columns
00039   int* selectionCount;               // Number of selected elements
00040   SelectedElement* selectionValues;  // Chosen values
00041 } ExploreParams;
00042
00058 __declspec(dllexport) int BacktrackAlgorithm(Matrix* matrix, int* maxSum,
00059                                              int* selectionCount,
00060                                              SelectedElement** maxSelection);
00061
00062 #endif  // !BACKTRACK_H
```

5.4 constants.h

```
00001
00010 #ifndef CONSTANTS_H
00011 #define CONSTANTS_H
00012
00013 // ANSI Codes
00014 #define ANSI_RESET "\033[0m"
00015 #define ANSI_BOLD "\033[1m"
00016 #define ANSI_UNDERLINE "\033[4m"
```

```
00017 #define ANSI_RED "\033[31m"
00018 #define ANSI_GREEN "\033[32m"
00019 #define ANSI_YELLOW "\033[33m"
00020
00021 // Message Strings
00022 #define MSG_CRITICAL ANSI_BOLD ANSI_RED "CRITICO: " ANSI_RESET
00023 #define MSG_ERROR ANSI_BOLD ANSI_RED "ERRO: " ANSI_RESET
00024 #define MSG_WARNING ANSI_BOLD ANSI_YELLOW "AVISO: " ANSI_RESET
00025 #define MSG_INFO ANSI_BOLD ANSI_GREEN "INFO: " ANSI_RESET
00026
00027 // Text File Constants
00028 #define ELEMENT_SEPARATOR ";"
00029 #define MAX_LINE_SIZE 500
00030
00031 // Default Matrix Values
00032 #define DEFAULT_MATRIX_VALUE 0
00033
00034 #endif  // !CONSTANTS_H
```

5.5 MatrixMatch/error_codes.h File Reference

Defines error codes used in the program.

Macros

- #define **SUCCESS** 0
- #define **MEMORY_ALLOCATION_FAILURE** -1
- #define **UNKNOWN_ARGUMENT** -2
- #define **NO_FILENAME_PROVIDED** -3
- #define **INVALID_MATRIX_OR_INDICES** -4
- #define **CANNOT_OPEN_FILE** -5
- #define **FILE_READ_ERROR** -6
- #define **UNABLE_REPLACE_VALUE** -7
- #define **OUT_OF_BOUNDS** -8
- #define **NULL_POINTER** -9

5.5.1 Detailed Description

Defines error codes used in the program.

This header file contains definitions of error codes that may occur during program execution. Each error code is defined with a unique value to facilitate error identification and handling.

Author

Enrique Rodrigues

Date

2.03.2024

Copyright

5.6 error_codes.h

Go to the documentation of this file.

```
00001
00012 #ifndef ERROR_CODES_H
00013 #define ERROR_CODES_H
00014
00015 #define SUCCESS 0  // Operation Success
00016
00017 #define MEMORY_ALLOCATION_FAILURE -1  // Memory allocation failure
00018 #define UNKNOWN_ARGUMENT -2           // Unknown argument
00019 #define NO_FILENAME_PROVIDED -3       // No filename provided
00020 #define INVALID_MATRIX_OR_INDICES -4  // Invalid matrix or matrix indices
00021 #define CANNOT_OPEN_FILE -5           // Unable to open the file
00022 #define FILE_READ_ERROR -6            // File read error
00023 #define UNABLE_REPLACE_VALUE -7  // Unable to replace the value of an element
00024 #define OUT_OF_BOUNDS -8         // Position is out of bounds of the matrix
00025 #define NULL_POINTER -9          // Pointer is NULL
00026
00027 #endif  // !ERROR_CODES_H
```

5.7 MatrixMatch/greedy.c File Reference

Implementation of the "Greedy" algorithm.

```
#include "greedy.h"
#include <stdio.h>
#include <stdlib.h>
#include "error_codes.h"
#include "matrix_core.h"
```

Functions

- int GreedyAlgorithm (Matrix *matrix, int *maxSum, int *maxSelection, int *currentSelectionSize)

  Solve the problem with a "Greedy" algorithm.

5.7.1 Detailed Description

Implementation of the "Greedy" algorithm.

This file contains the implementation of functions related to the "Greedy" algorithm.

Author

Enrique Rodrigues

Date

10.03.2024

Copyright

5.7.2 Function Documentation

5.7.2.1 GreedyAlgorithm()

```
int GreedyAlgorithm (
            Matrix * matrix,
            int * maxSum,
            int * maxSelection,
            int * currentSelectionSize )
```

Solve the problem with a "Greedy" algorithm.

Parameters

matrix	- The matrix.
maxSum	- Pointer to store the maximum sum.
maxSelection	- Pointer to store the selected numbers.
currentSelectionSize	- Pointer to store the selected elements.

Return values

`INVALID_MATRIX_OR_INDICES`	- The matrix or the provided indices are invalid.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure for the new matrix element.
`SUCCESS`	- Operation successful.

5.8 MatrixMatch/greedy.h File Reference

Header file for the "Greedy" algorithm.

```
#include "matrix_core.h"
```

Functions

- __declspec (dllexport) int GreedyAlgorithm(Matrix *matrix

  Solve the problem with a "Greedy" algorithm.

Variables

- int * **maxSum**
- int int * **maxSelection**
- int int int * **currentSelectionSize**

5.8.1 Detailed Description

Header file for the "Greedy" algorithm.

This header file contains function declarations and data structures related to the "Greedy" algorithm.

Author

Enrique Rodrigues

Date

10.03.2024

Copyright

5.8.2 Function Documentation

5.8.2.1 __declspec()

```
__declspec (
            dllexport  )
```

Solve the problem with a "Greedy" algorithm.

Parameters

matrix	- The matrix.
maxSum	- Pointer to store the maximum sum.
maxSelection	- Pointer to store the number of selected elements.
currentSelectionSize	- Pointer to store the selected elements.

Return values

`*INVALID_MATRIX_OR_INDICES*`	- The matrix or the provided indices are invalid.
`*MEMORY_ALLOCATION_FAILURE*`	- Memory allocation failure for the new matrix element.
`*SUCCESS*`	- Operation successful.

5.9 greedy.h

Go to the documentation of this file.

```
00001
00010 #ifndef GREEDY_H
00011 #define GREEDY_H
00012
00013 #include "matrix_core.h"
```

```
00014
00028 __declspec(dllexport) int GreedyAlgorithm(Matrix* matrix, int* maxSum,
00029                                           int* maxSelection,
00030                                           int* currentSelectionSize);
00031
00032 #endif  // !GREEDY_H
```

5.10 MatrixMatch/hungarian.c File Reference

Implementation of the Hungarian Algorithm to solve the problem.

```
#include "hungarian.h"
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "matrix_core.h"
#include "matrix_io.h"
```

Functions

- int HungarianAlgorithm (Matrix ∗matrix, int ∗∗chosenElements, int ∗result)

  Implements the Hungarian algorithm to find the solution to the problem.

5.10.1 Detailed Description

Implementation of the Hungarian Algorithm to solve the problem.

This file contains the necessary functions for the implementation of the Hungarian Algorithm.

Author

Enrique Rodrigues

Date

15.03.2024

Copyright

5.10.2 Function Documentation

5.10.2.1 HungarianAlgorithm()

```
int HungarianAlgorithm (
            Matrix * matrix,
            int ** chosenElements,
            int * result )
```

Implements the Hungarian algorithm to find the solution to the problem.

Parameters

matrix	- Pointer to the input matrix.
chosenElements	- Pointer to a pointer of integers, which will be allocated and filled with the chosen elements.
result	- Pointer to an integer, which will be filled with the result of the sum of the chosen elements.

Return values

`INVALID_MATRIX_OR_INDICES`	- The matrix or the provided indices are invalid.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure for the new matrix element.
`SUCCESS`	- Operation successful.

5.11 MatrixMatch/hungarian.h File Reference

Header file for the Hungarian algorithm.

```
#include "matrix_core.h"
```

Classes

- struct Zero

  Structure representing a zero in the matrix.

Typedefs

- typedef struct Zero **Zero**

Functions

- __declspec (dllexport) int HungarianAlgorithm(Matrix ∗matrix

  Implements the Hungarian algorithm to find the solution to the problem.

Variables

- int ∗∗ **chosenElements**
- int int ∗ **result**

5.11.1 Detailed Description

Header file for the Hungarian algorithm.

This header file contains function declarations and data structures related to the Hungarian algorithm.

Author

Enrique Rodrigues

Date

14.03.2024

Copyright

5.11.2 Function Documentation

5.11.2.1 __declspec()

```
__declspec (
            dllexport  )
```

Implements the Hungarian algorithm to find the solution to the problem.

Parameters

matrix	- Pointer to the input matrix.
chosenElements	- Pointer to a pointer of integers, which will be allocated and filled with the chosen elements.
result	- Pointer to an integer, which will be filled with the result of the sum of the chosen elements.

Return values

`*INVALID_MATRIX_OR_INDICES*`	- The matrix or the provided indices are invalid.
`*MEMORY_ALLOCATION_FAILURE*`	- Memory allocation failure for the new matrix element.
`*SUCCESS*`	- Operation successful.

5.12 hungarian.h

Go to the documentation of this file.

```
00001
00010 #ifndef HUNGARIAN_ALGORITHM
00011 #define HUNGARIAN_ALGORITHM
00012
00013 #include "matrix_core.h"
00014
00019 typedef struct Zero {
```

```
00020   int row;  // Row where the zero is
00021   int col;  // Column where the zero is
00022 } Zero;
00023
00038 __declspec(dllexport) int HungarianAlgorithm(Matrix* matrix,
00039                                               int** chosenElements, int* result);
00040
00041 #endif  // !HUNGARIAN_ALGORITHM
```

5.13 MatrixMatch/matrix_core.c File Reference

Implementations of the main functions for matrix operations.

```
#include "matrix_core.h"
#include <stdio.h>
#include <stdlib.h>
#include "constants.h"
#include "error_codes.h"
```

Functions

- int ReplaceValueAtPosition (Matrix *matrix, int row, int col, int value)

  Swap the value of an element at a given position in the matrix.
- MatrixElement * CreateMatrixElement (int value, int column)

  Create a new element with a given value.
- MatrixElement * InitializeRow (int width)

  Initialize a matrix row with default values.
- MatrixRowNode * InitializeRowNode (int width)

  Initialize a row node of the matrix with a given size, filling it with default values.
- int CreateMatrix (int width, int height, Matrix **matrix)

  Create a matrix of a given size filled with default values.
- MatrixElement * AddElementToRow (MatrixElement *head, int value, int column)

  Create and add an element to the end of a row in the matrix.
- void FreeMatrix (Matrix *matrix)

  Free allocated memory of a matrix.
- MatrixRowNode * GetRowNode (Matrix *matrix, int rowIndex)

  Get the row node at a specific index.
- MatrixElement * GetElementCol (MatrixRowNode *rowNode, int colIndex)

  Get an element in a specific column.

5.13.1 Detailed Description

Implementations of the main functions for matrix operations.

This file contains the implementations of the main functions defined in the header file matrix_core.h. It includes functions for creating matrices, filling matrices with data, freeing allocated memory for matrices, and other related operations.

Author

Enrique Rodrigues

Date

1.03.2024

Copyright

5.13.2 Function Documentation

5.13.2.1 AddElementToRow()

```
MatrixElement * AddElementToRow (
            MatrixElement * head,
            int value,
            int column )
```

Create and add an element to the end of a row in the matrix.

Parameters

head	- The first element of the row.
value	- The value the element will have.

Return values

-	NULL in case of error or the pointer to the beginning of the row.

5.13.2.2 CreateMatrix()

```
int CreateMatrix (
            int width,
            int height,
            Matrix ** matrix )
```

Create a matrix of a given size filled with default values.

Parameters

width	- The number of columns of the matrix.
height	- The number of rows of the matrix.
matrix	- Matrix to hold the matrix data.

Return values

`INVALID_MATRIX_OR_INDICES`	- The matrix or indices provided are invalid.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure for the new matrix element.
`SUCESS`	- Operation successful.

5.13.2.3 CreateMatrixElement()

```
MatrixElement * CreateMatrixElement (
            int value,
            int column )
```

Create a new element with a given value.

Parameters

value	- The value to be stored in the MatrixElement.

Return values

-	A pointer to the new element, or NULL in case of memory allocation error.

5.13.2.4 FreeMatrix()

```
void FreeMatrix (
            Matrix * matrix )
```

Free allocated memory of a matrix.

Parameters

matrix	- The matrix to be freed.

5.13.2.5 GetElementCol()

```
MatrixElement * GetElementCol (
            MatrixRowNode * rowNode,
            int colIndex )
```

Get an element in a specific column.

Parameters

rowNode	- The row node of the matrix.
colIndex	- The column index of the matrix.

Return values

-	The matrix element.

5.13.2.6 GetRowNode()

```
MatrixRowNode * GetRowNode (
```

```
            Matrix * matrix,
            int rowIndex )
```

Get the row node at a specific index.

Parameters

matrix	- The matrix.
rowIndex	- The index of the desired row.

Return values

-	The corresponding row node if the index is valid, NULL otherwise.

5.13.2.7 InitializeRow()

```
MatrixElement * InitializeRow (
            int width )
```

Initialize a matrix row with default values.

Parameters

width	- Row size (number of columns).

Return values

-	Pointer to the last element of the row, or NULL in case of memory allocation error.

5.13.2.8 InitializeRowNode()

```
MatrixRowNode * InitializeRowNode (
            int width )
```

Initialize a row node of the matrix with a given size, filling it with default values.

Parameters

width	- Row size (number of columns).

Return values

-	A pointer to the new row node, or NULL in case of memory allocation error.

5.13.2.9 ReplaceValueAtPosition()

```
int ReplaceValueAtPosition (
            Matrix * matrix,
            int row,
            int col,
            int value )
```

Swap the value of an element at a given position in the matrix.

Parameters

matrix	- The matrix containing the element.
row	- The row of the matrix where the element is.
col	- The column of the matrix where the element is.
value	- The new value to place in the element.

Return values

`OUT_OF_BOUNDS`	- Invalid matrix position.
`SUCCESS`	- Operation successful.

5.14 MatrixMatch/matrix_core.h File Reference

Header file containing the definitions of structures and main functions for matrix operations.

```
#include "constants.h"
#include "error_codes.h"
```

Classes

- struct MatrixElement

  Represents an element of the matrix.
- struct MatrixRowNode

  Represents a row as a node of the matrix.
- struct Matrix

  A matrix.

Typedefs

- typedef struct MatrixElement **MatrixElement**
- typedef struct MatrixRowNode **MatrixRowNode**
- typedef struct Matrix **Matrix**

Functions

- __declspec (dllexport) int ReplaceValueAtPosition(Matrix *matrix

  Swap the value of an element at a given position in the matrix.

Variables

- int **row**
- int int **col**
- int int int **value**
- int **column**
- int **height**
- int Matrix ** **matrix**
- int **rowIndex**
- int **colIndex**

5.14.1 Detailed Description

Header file containing the definitions of structures and main functions for matrix operations.

This file contains the main structures and functions related to matrix manipulation, such as creation and population of matrices, memory deallocation, etc.

Author

Enrique Rodrigues

Date

1.03.2024

Copyright

5.14.2 Function Documentation

5.14.2.1 __declspec()

```
__declspec (
            dllexport  )
```

Swap the value of an element at a given position in the matrix.

Get an element in a specific column.

Get the row node at a specific index.

Free allocated memory of a matrix.

Create and add an element to the end of a row in the matrix.

Create a matrix of a given size filled with default values.

Initialize a row node of the matrix with a given size, filling it with default values.

Initialize a matrix row with default values.

Create a new element with a given value.

Parameters

matrix	- The matrix containing the element.
row	- The row of the matrix where the element is.
col	- The column of the matrix where the element is.
value	- The new value to place in the element.

Return values

`OUT_OF_BOUNDS`	- Invalid matrix position.
`SUCCESS`	- Operation successful.

Parameters

value	- The value to be stored in the `MatrixElement`.

Return values

-	A pointer to the new element, or NULL in case of memory allocation error.

Parameters

width	- Row size (number of columns).

Return values

-	Pointer to the last element of the row, or NULL in case of memory allocation error.

Parameters

width	- Row size (number of columns).

Return values

-	A pointer to the new row node, or NULL in case of memory allocation error.

Parameters

width	- The number of columns of the matrix.
height	- The number of rows of the matrix.
matrix	- Matrix to hold the matrix data.

Return values

`INVALID_MATRIX_OR_INDICES`	- The matrix or indices provided are invalid.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure for the new matrix element.
`SUCESS`	- Operation successful.

Parameters

head	- The first element of the row.
value	- The value the element will have.

Return values

-	NULL in case of error or the pointer to the beginning of the row.

Parameters

matrix	- The matrix to be freed.
matrix	- The matrix.
rowIndex	- The index of the desired row.

Return values

-	The corresponding row node if the index is valid, NULL otherwise.

Parameters

rowNode	- The row node of the matrix.
colIndex	- The column index of the matrix.

Return values

-	The matrix element.

5.15 matrix_core.h

Go to the documentation of this file.

```
00001
00012 #ifndef MATRIX_H
00013 #define MATRIX_H
00014
00015 #include "constants.h"
00016 #include "error_codes.h"
00017
00025 typedef struct MatrixElement {
00026   int value;                      // Element value as integer
00027   int column;                     // Current column
00028   struct MatrixElement* nextCol;  // Next element in the line (next col)
00029 } MatrixElement;
00030
00038 typedef struct MatrixRowNode {
00039   MatrixElement* row;             // Pointer to the first element of the row
00040   struct MatrixRowNode* nextRow;  // Pointer to the next row node
00041 } MatrixRowNode;
00042
00050 typedef struct Matrix {
00051   MatrixRowNode* head;  // Pointer to first row of matrix
00052   int width;            // Matrix width
00053   int height;           // Matrix height
00054 } Matrix;
00055
00065 __declspec(dllexport) int ReplaceValueAtPosition(Matrix* matrix, int row,
00066                                                  int col, int value);
```

```
00067
00074 __declspec(dllexport) MatrixElement* CreateMatrixElement(int value, int column);
00075
00082 __declspec(dllexport) MatrixElement* InitializeRow(int width);
00083
00091 __declspec(dllexport) MatrixRowNode* InitializeRowNode(int width);
00092
00104 __declspec(dllexport) int CreateMatrix(int width, int height, Matrix** matrix);
00105
00113 __declspec(dllexport) MatrixElement* AddElementToRow(MatrixElement* head,
00114                                                      int value, int column);
00115
00120 __declspec(dllexport) void FreeMatrix(Matrix* matrix);
00121
00129 __declspec(dllexport) MatrixRowNode* GetRowNode(Matrix* matrix, int rowIndex);
00130
00137 __declspec(dllexport) MatrixElement* GetElementCol(MatrixRowNode* rowNode,
00138                                                    int colIndex);
00139
00140 #endif  // !MATRIX_H
```

5.16 MatrixMatch/matrix_io.c File Reference

Input/output operations for matrices.

```
#include "matrix_io.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "constants.h"
#include "error_codes.h"
#include "matrix_core.h"
```

Functions

- void PrintMatrix (const Matrix *matrix)

  Displays a matrix on the screen.
- int GetMatrixSizeFromFile (const char *filename, int *width, int *height)

  Obtains the size of a matrix stored in a file.
- int PopulateMatrixFromFile (const char *filename, Matrix **matrix)

  Fills a matrix with data from a file.
- int CreateMatrixFromFile (const char *filename, Matrix **matrix)

  Creates a matrix from data in a file.
- int InsertRow (Matrix *matrix, const int *newRow, const int sizeArray)

  Inserts a row into the matrix.
- int InsertColumn (Matrix *matrix, const int *newColumn, const int sizeArray)

  Inserts a column at the end of a row in the matrix.
- int DeleteRow (Matrix *matrix, int rowIndex)

  Deletes a row from the matrix.
- int DeleteColumn (Matrix *matrix, int colIndex)

  Deletes a column from the matrix.

5.16.1 Detailed Description

Input/output operations for matrices.

This file contains functions for input/output manipulation of matrices. Specifically, it allows populating a matrix with data from a file, printing a matrix, etc.

Author

Enrique Rodrigues

Date

1.03.2024

Copyright

5.16.2 Function Documentation

5.16.2.1 CreateMatrixFromFile()

```
int CreateMatrixFromFile (
            const char * filename,
            Matrix ** matrix )
```

Creates a matrix from data in a file.

Parameters

filename	- The name of the file.
matrix	- Matrix that will contain the data of the matrix.

Return values

`CANNOT_OPEN_FILE`	- Failure to open the file.
`INVALID_MATRIX_OR_INDICES`	- The matrix or the provided indices are invalid.
`MEMORY_ALLOCATION_FAILURE`	- Failure to allocate memory for the new matrix element.
`FILE_READ_ERROR`	- Error reading data from the file.
`SUCCESS`	- Operation successful.

5.16.2.2 DeleteColumn()

```
int DeleteColumn (
            Matrix * matrix,
            int colIndex )
```

Deletes a column from the matrix.

Parameters

matrix	- The matrix.
colIndex	- The column to be deleted.

Return values

`OUT_OF_BOUNDS`	- Position does not exist.
`SUCCESS`	- Operation successful.

5.16.2.3 DeleteRow()

```
int DeleteRow (
            Matrix * matrix,
            int rowIndex )
```

Deletes a row from the matrix.

Parameters

matrix	- The matrix.
rowIndex	- The row of the matrix to be deleted.

Return values

`OUT_OF_BOUNDS`	- Position does not exist.
`SUCCESS`	- Operation successful.

5.16.2.4 GetMatrixSizeFromFile()

```
int GetMatrixSizeFromFile (
            const char * filename,
            int * width,
            int * height )
```

Obtains the size of a matrix stored in a file.

Parameters

filename	- The name of the file containing the matrix.
width	- A variable of type `int` that will hold the number of columns of the matrix.
height	- A variable of type `int` that will hold the number of rows of the matrix.

Return values

`CANNOT_OPEN_FILE`	- Failure to open the file.
`SUCCESS`	- Operation successful.

5.16.2.5 InsertColumn()

```
int InsertColumn (
            Matrix * matrix,
            const int * newColumn,
            const int sizeArray )
```

Inserts a column at the end of a row in the matrix.

Parameters

matrix	- The matrix where we will insert the new column.
newColumn	- Array with the values of the new column.
sizeArray	- Size of the array.

Return values

`OUT_OF_BOUNDS`	- Size of the array is different from the size of the matrix.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure.
`SUCCESS`	- Operation successful.

5.16.2.6 InsertRow()

```
int InsertRow (
            Matrix * matrix,
            const int * newRow,
            const int sizeArray )
```

Inserts a row into the matrix.

Parameters

matrix	- The matrix where we will insert a new row.
newRow	- Array with the values of the new row.
sizeArray	- Size of the array.

Return values

`OUT_OF_BOUNDS`	- Size of the array is different from the size of the matrix.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure.
`SUCCESS`	- Operation successful.

5.16.2.7 PopulateMatrixFromFile()

```
int PopulateMatrixFromFile (
            const char * filename,
            Matrix ** matrix )
```

Fills a matrix with data from a file.

Parameters

filename	- The name of the file.
matrix	- The matrix to be filled with data from the file.

Return values

`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure.
`CANNOT_OPEN_FILE`	- Failure to open the file.
`FILE_READ_ERROR`	- Error reading data from the file.
`SUCCESS`	- Operation successful.

5.16.2.8 PrintMatrix()

```
void PrintMatrix (
            const Matrix * matrix )
```

Displays a matrix on the screen.

Parameters

matrix	- The matrix to be displayed.

5.17 MatrixMatch/matrix_io.h File Reference

Header file that defines input/output operations for matrices.

```
#include "matrix_core.h"
```

Functions

- __declspec (dllexport) void PrintMatrix(const Matrix *matrix)

  Displays a matrix on the screen.

Variables

- int * **width**
- int int * **height**
- Matrix ** **matrix**
- const int * **newRow**
- const int const int **sizeArray**
- const int * **newColumn**
- int **rowIndex**
- int **colIndex**

5.17.1 Detailed Description

Header file that defines input/output operations for matrices.

This file contains the definitions of functions for manipulating matrices through input/output. Specifically, it allows populating a matrix with data from a file, printing a matrix, etc.

Author

Enrique Rodrigues

Date

1.03.2024

Copyright

5.17.2 Function Documentation

5.17.2.1 __declspec()

```
__declspec (
            dllexport  ) const
```

Displays a matrix on the screen.

Deletes a column from the matrix.

Deletes a row from the matrix.

Inserts a column at the end of a row in the matrix.

Inserts a row into the matrix.

Creates a matrix from data in a file.

Fills a matrix with data from a file.

Obtains the size of a matrix stored in a file.

Parameters

matrix	- The matrix to be displayed.
filename	- The name of the file containing the matrix.
width	- A variable of type `int` that will hold the number of columns of the matrix.
height	- A variable of type `int` that will hold the number of rows of the matrix.

Return values

`CANNOT_OPEN_FILE`	- Failure to open the file.
`SUCCESS`	- Operation successful.

Parameters

filename	- The name of the file.
matrix	- The matrix to be filled with data from the file.

Return values

`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure.
`CANNOT_OPEN_FILE`	- Failure to open the file.
`FILE_READ_ERROR`	- Error reading data from the file.
`SUCCESS`	- Operation successful.

Parameters

filename	- The name of the file.
matrix	- Matrix that will contain the data of the matrix.

Return values

`CANNOT_OPEN_FILE`	- Failure to open the file.
`INVALID_MATRIX_OR_INDICES`	- The matrix or the provided indices are invalid.
`MEMORY_ALLOCATION_FAILURE`	- Failure to allocate memory for the new matrix element.
`FILE_READ_ERROR`	- Error reading data from the file.
`SUCCESS`	- Operation successful.

Parameters

matrix	- The matrix where we will insert a new row.
newRow	- Array with the values of the new row.
sizeArray	- Size of the array.

Return values

`OUT_OF_BOUNDS`	- Size of the array is different from the size of the matrix.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure.
`SUCCESS`	- Operation successful.

Parameters

matrix	- The matrix where we will insert the new column.
newColumn	- Array with the values of the new column.
sizeArray	- Size of the array.

Return values

`OUT_OF_BOUNDS`	- Size of the array is different from the size of the matrix.
`MEMORY_ALLOCATION_FAILURE`	- Memory allocation failure.
`SUCCESS`	- Operation successful.

Parameters

matrix	- The matrix.
rowIndex	- The row of the matrix to be deleted.

Return values

`OUT_OF_BOUNDS`	- Position does not exist.
`SUCCESS`	- Operation successful.

Parameters

matrix	- The matrix.
colIndex	- The column to be deleted.

Return values

`OUT_OF_BOUNDS`	- Position does not exist.
`SUCCESS`	- Operation successful.

5.18 matrix_io.h

Go to the documentation of this file.

```
00001
00011 #ifndef MATRIX_IO_H
00012 #define MATRIX_IO_H
00013
00014 #include "matrix_core.h"
00015
00020 __declspec(dllexport) void PrintMatrix(const Matrix* matrix);
00021
00032 __declspec(dllexport) int GetMatrixSizeFromFile(const char* filename,
00033                                                 int* width, int* height);
00034
00044 __declspec(dllexport) int PopulateMatrixFromFile(const char* filename,
00045                                                  Matrix** matrix);
00046
00059 __declspec(dllexport) int CreateMatrixFromFile(const char* filename,
00060                                                Matrix** matrix);
00061
00072 __declspec(dllexport) int InsertRow(Matrix* matrix, const int* newRow,
00073                                     const int sizeArray);
00074
00085 __declspec(dllexport) int InsertColumn(Matrix* matrix, const int* newColumn,
00086                                        const int sizeArray);
00087
00095 __declspec(dllexport) int DeleteRow(Matrix* matrix, int rowIndex);
00096
00104 __declspec(dllexport) int DeleteColumn(Matrix* matrix, int colIndex);
00105
00106 #endif  // !MATRIX_IO_H
```

Index