Maximum Submatrix Sum Project

Generated by Doxygen 1.9.4

1 Maximum Submatrix Sum Project

1.1 Introduction

The goal of this project is to find the maximum submatrix sum of a given matrix.

1.2 Data File Format

The data file stores the matrix data. The first line is the number of rows and cols of the matrix. The following lines are the elements of the matrix. For example:

```
3 3
1 2 3
4 5 6
7 8 9
```

1.3 Report File Format

This program will append the result to the report file "report.csv". The format is:

datasize	algorithm	iteration	ticks	total_time	duration
100	1	10	123456	0.123456	0.012345

2 Module Index

2.1 Modules

Here is a list of all modules:

3 Class Index

3.1 Class List

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

4 File Index

4.1 File List

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

5 Module Documentation

5.1 Maximum Submatrix Sum

Maximum Submatrix Sum.

Functions

- Matrix * MaxSubmatrixN6 (Matrix *m)

  The naive version of the maximum submatrix sum algorithm.
- Matrix * MaxSubmatrixN4 (Matrix *m)

  This function implements the N4 version of the maximum submatrix sum algorithm.
- Matrix * MaxSubmatrix (Matrix *m)

  This function implements my version of the maximum submatrix sum algorithm.

5.1.1 Detailed Description

Maximum Submatrix Sum.

This module contains the functions for the maximum submatrix sum problem. Their APIs are unified to make it easy to test them.

All the functions in this module take a matrix as input and return another matrix as output, even if the result is the same as the input (to prevent double free problem). The input matrix is not modified. There is no constraint on the input matrix. The output matrix is also a general matrix.

Parameters

m	Pointer to the matrix.

Returns

Matrix* Pointer to the result matrix.

5.1.2 Function Documentation

5.1.2.1 MaxSubmatrix() `Matrix * MaxSubmatrix ( Matrix * m )`

This function implements my version of the maximum submatrix sum algorithm.

It substitutes the N4 version of finding the maximum subarray sum with Kadane's algorithm. This reduces the time complexity from O(n^4) to O(n^3).

5.1.2.2 MaxSubmatrixN4() `Matrix * MaxSubmatrixN4 ( Matrix * m )`

This function implements the N4 version of the maximum submatrix sum algorithm.

Compared to the N6 version, this algorithm reduces the time complexity from O(n^6) to O(n^4). It sums each row of the submatrix and stores the sum in an array. Then it uses Kadane's algorithm to find the maximum subarray sum of the array.

5.1.2.3 MaxSubmatrixN6() `Matrix * MaxSubmatrixN6 ( Matrix * m )`

The naive version of the maximum submatrix sum algorithm.

This algorithm simply enumerates all possible submatrices and finds the one with the maximum sum. The variable i, j, k, l means the submatrix is from row i to row k and from column j to column l. For each combination of i, j, k, l, it traverses all elements in the submatrix and calculates the sum. If the sum is larger than the maximum sum, it updates the maximum sum and the maximum submatrix. After traversing all possible submatrices, it returns the maximum submatrix.

6 Class Documentation

6.1 Matrix Struct Reference

Matrix structure.

`#include <mss.h>`

Public Attributes

- int **rows**
- int **cols**
- int * **data**

6.1.1 Detailed Description

Matrix structure.

This structure is simply a wrapper of the matrix elements and its number of rows and columns.

Though the problem in PTA only requires the input matrix to be a square matrix, I still use a general matrix structure to make the program more flexible. The result matrix is also a general matrix.

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

- mss.h

7 File Documentation

7.1 main.c File Reference

Driver program for the maximum submatrix sum problem.

```
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <stdlib.h>
#include "mss.h"
```

Functions

- int **main** (int argc, char ∗argv[ ])

7.1.1 Detailed Description

Driver program for the maximum submatrix sum problem.

This program runs test for the project and append the result to the report file "report.csv".

7.1.2 Usage

./mss <datafile> <algorithm> [iteration]

- datafile: The name of the data file.
- algorithm: The algorithm to be tested. 1 means the N6 version, 2 means the N4 version, 3 means my version.
- iteration: The number of iterations to run the algorithm. If not specified, the program will run the algorithm at least once until the total time is more than 5 second.

This program will print the result matrix to the standard output.

7.2 mss.c File Reference

This file contains function implementations for the problem.

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

Functions

- Matrix * CreateMatrix (const int rows, const int cols)

  Create a Matrix object.
- Matrix * CopyMatrix (Matrix *m)

  Copy a Matrix object.
- void ReadMatrix (Matrix *m, FILE *fp)

  Read Matrix Elements from File.
- void PrintMatrix (Matrix *m, FILE *fp)

  Print Matrix to File.
- void FreeMatrix (Matrix *m)

  Free the memory allocated for the matrix.
- Matrix * MaxSubmatrixN6 (Matrix *m)

  The naive version of the maximum submatrix sum algorithm.
- Matrix * MaxSubmatrixN4 (Matrix *m)

  This function implements the N4 version of the maximum submatrix sum algorithm.
- Matrix * MaxSubmatrix (Matrix *m)

  This function implements my version of the maximum submatrix sum algorithm.

7.2.1 Detailed Description

This file contains function implementations for the problem.

7.2.2 Function Documentation

7.2.2.1 CopyMatrix() `Matrix * CopyMatrix ( Matrix * m )`

Copy a Matrix object.

This function creates a new matrix and copies the elements from the original matrix to the new matrix.

7.2.2.2 CreateMatrix() `Matrix * CreateMatrix ( const int rows, const int cols )`

Create a Matrix object.

This function uses malloc to allocate memory for the matrix structure and the matrix elements. So you need not to allocate memory for the matrix before calling this function.

Because there is pointer in the matrix structure, you need to use FreeMatrix() function to free the memory allocated for the matrix.

7.2.2.3 FreeMatrix() void FreeMatrix (Matrix * m)

Free the memory allocated for the matrix.

This function frees the memory allocated for the matrix structure and the matrix elements.

There is pointer in the matrix structure, so you need to use this function to free the memory allocated for the matrix.

Matrix pointer should be set to NULL if not been initialized or was freed.

This function will check if the pointer is NULL and prevent double free, so you need not to check it before calling this function.

7.2.2.4 PrintMatrix() void PrintMatrix (Matrix * m, FILE * fp)

Print Matrix to File.

This function prints matrix elements to the file following row-major order.

7.2.2.5 ReadMatrix() void ReadMatrix (Matrix * m, FILE * fp)

Read Matrix Elements from File.

This function reads matrix elements from the file following row-major order.

Rows and cols should be read from the file before calling this function. Numbers of rows and cols of the matrix should already exist in the matrix structure. This function only reads the matrix elements.

7.3 mss.h File Reference

Matrix interface for the maximum submatrix sum problem.

Classes

- struct Matrix

  Matrix structure.

Typedefs

- typedef struct Matrix **Matrix**

Functions

- Matrix * CreateMatrix (const int rows, const int cols)

  Create a Matrix object.
- Matrix * CopyMatrix (Matrix *m)

  Copy a Matrix object.
- void ReadMatrix (Matrix *m, FILE *fp)

  Read Matrix Elements from File.
- void PrintMatrix (Matrix *m, FILE *fp)

  Print Matrix to File.
- void FreeMatrix (Matrix *m)

  Free the memory allocated for the matrix.
- Matrix * MaxSubmatrixN6 (Matrix *m)

  The naive version of the maximum submatrix sum algorithm.
- Matrix * MaxSubmatrixN4 (Matrix *m)

  This function implements the N4 version of the maximum submatrix sum algorithm.
- Matrix * MaxSubmatrix (Matrix *m)

  This function implements my version of the maximum submatrix sum algorithm.

7.3.1 Detailed Description

Matrix interface for the maximum submatrix sum problem.

This file contains the definition of the matrix structure and interface for the maximum submatrix sum problem.

7.3.2 Function Documentation

7.3.2.1 CopyMatrix() `Matrix * CopyMatrix ( Matrix * m )`

Copy a Matrix object.

Parameters

m	Pointer to the matrix.

Returns

Matrix* Pointer to the new matrix.

This function creates a new matrix and copies the elements from the original matrix to the new matrix.

7.3.2.2 CreateMatrix() `Matrix * CreateMatrix ( const int rows, const int cols )`

Create a Matrix object.

Parameters

rows	Rows of the matrix.
cols	Columns of the matrix.

Returns

Matrix* Pointer to the matrix.

This function uses malloc to allocate memory for the matrix structure and the matrix elements. So you need not to allocate memory for the matrix before calling this function.

Because there is pointer in the matrix structure, you need to use FreeMatrix() function to free the memory allocated for the matrix.

7.3.2.3 FreeMatrix()

```
void FreeMatrix (
        Matrix * m )
```

Free the memory allocated for the matrix.

Parameters

m	Pointer to the matrix.

This function frees the memory allocated for the matrix structure and the matrix elements.

There is pointer in the matrix structure, so you need to use this function to free the memory allocated for the matrix.

Matrix pointer should be set to NULL if not been initialized or was freed.

This function will check if the pointer is NULL and prevent double free, so you need not to check it before calling this function.

7.3.2.4 PrintMatrix()

```
void PrintMatrix (
        Matrix * m,
        FILE * fp )
```

Print Matrix to File.

Parameters

m	Pointer to the matrix.
fp	Pointer to the file to be written.

This function prints matrix elements to the file following row-major order.

7.3.2.5 ReadMatrix()

```
void ReadMatrix (
        Matrix * m,
        FILE * fp )
```

Read Matrix Elements from File.

Parameters

m	Pointer to the matrix.
fp	Pointer to the file to be read.

This function reads matrix elements from the file following row-major order.

Rows and cols should be read from the file before calling this function. Numbers of rows and cols of the matrix should already exist in the matrix structure. This function only reads the matrix elements.

7.4 mss.h

Go to the documentation of this file.

```
1
7 #ifndef _MSS_H_
8 #define _MSS_H_
9
20 struct Matrix
21 {
22     int rows;
23     int cols;
24     int *data;
25 };
26 typedef struct Matrix Matrix;
27
35 Matrix* CreateMatrix(const int rows, const int cols);
36
43 Matrix* CopyMatrix(Matrix *m);
44
51 void ReadMatrix(Matrix *m, FILE *fp);
52
59 void PrintMatrix(Matrix *m, FILE *fp);
60
66 void FreeMatrix(Matrix *m);
67
84 Matrix* MaxSubmatrixN6(Matrix *m);
85 Matrix* MaxSubmatrixN4(Matrix *m);
86 Matrix* MaxSubmatrix(Matrix *m); // end of mss
88
89 #endif
90
```

Index