# Performance Measurement (MSS)

Date: 2022-9.23

FYA: **If you want to run my codes, please check `Readme.md` first!.**

# Chapter 1: Introduction

## 1 Description

- The problem ask us to find the max sum of sub matrix of a $N * N$（$N \leq 100$） matrix with three methods with the time complexity of O($N^3$), O($N^4$) and O($N^6$). It can be also described like the following words.

$$Find\ \ i_{lt}, j_{lt}, i_{rb}, j_{rb}\ \ that:$$
$$\forall i_1, i_2, j_1, j_2, \sum_{i \in [i_{lt}, i_{rb}), j \in [j_{lt}, j_{rb})} M_{i,j} > \sum_{i \in [i_1, i_2), j \in [j_1, j_2)} M_{i,j}$$

- Also, we are required to design a module to measure the performance of the three methods.

# Chapter 2: Algorithm Specification

## 1 $N^6$

- For $N^6$ version, I just use 6 for loop to iterate the left-top key point ($N^2$), the right-bottom key point ($N^2$), and calculate the sum of the sub matrix chosen by the two key points ($N^2$).
- Only one matrix with $N^2$ size is used to store the input data.
- The idea is quite easy, and I write enough comments, so maybe check `src/N6.c` for details. Everything is done in `calMaxSubMatrixSum()`.

```
func calMaxSubMatrixSum(mat){
    // Iterate the left-top key point.
    for(i, j){
        // Iterate the right-bottom key point.
        for(k, l){
            // Calculate the sum in region from [k][l] to [i+1][j+1].
            for(x, y){
                tmp += mat[x][y];
            }
         }
```

```
            // Update the answer if we get larger sum.
            ans = max(ans, tmp);
        }
    }
    return ans;
}
```

## 2 $N^4$

- $N^4$ version has something improved based on the $N^6$ one. I generate a brand new matrix to calculate the prefix sum of the original matrix. That means, suppose the old matrix is M, and the new one is M', than $M'_{I,J} = \sum_{i,j}^{I,J} M_{i,j}$, so that, we can calculate the sum of any sub matrix indexed by $(i_{lt}, j_{lt}), (i_{rb}, j_{rb})$ in O(1) by $M^{sub}_{i_{lt},j_{lt},i_{rb},j_{rb}} = M'_{i_{rb},j_{rb}} + M'_{i_{lt}-1,j_{lt}-1} - M'_{i_{rb},j_{lt}-1} - M'_{i_{lt}-1,j_{rb}}$ according to the Inclusion and Exclusion Principle.

```
// a matrix that has the following feature:
// sumM[I][J] = ${ \sum^{I,J}_{i,j} mat[i][j] }$
func initSumMatrix(sumM, mat){
    for(j){
        sumM[0][j] = 0;
    }
    for(i){
        sumM[i][0] = 0;
    }
    // sumM[I][J] = ${ \sum^{I,J}_{i,j} mat[i][j] }$
    //            = sumM[I-1][J] + sumM[I-1][J-1] - sumM[I-1][J-1] + mat[I][J]
    for(i, j){
        sumM[i][j] = sumM[i-1][j] + sumM[i-1][j-1] - sumM[i-1][j-1] + mat[i][j]
    }
}
```

- So I use $N^2$ memory to deal with the prefix sum matrix, and $N^4$ memory to iterate the left-top key point ($N^2$ memory) and the right-bottom key point ($N^2$ memory). Check `src/N4.c` for details.

```
// The N2 for loop in N6 method is replaced by the following things.
                    int tmp = 0;
                    // Calculate the sum in region from [k][l] to [i+1][j+1].
                    tmp = sumM[k][l] + sumM[i][j] - sumM[k][j] - sumM[i][l];
                    // Update the answer if we get larger sum.
                    ans = max(ans, tmp);
```

- One matrix is used to store the input data, another is to store the prefix sum.

# 3 $N^3$

- This method has very difference between the previous two. It's more like the O(N) approach of the largest sub segment sum. Our idea is to change the matrix into a vector, which means we can directly use the method of the largest sub-terminal sum, which contributes to the complexity of O(N).
- So the idea is to iterate the upper bound (l2) and the lower bound (l1) on one axis of the matrix, and just flatten this matrix. That means, we will have a new $1 \times m$ matrix (actually a vector) $M^{fold}_{l1,l2,1,j} = \sum_{l1 \le i < l2} M_{i,j}$. Then, we just need to do the max sub segment sum on $M^{fold}$.

```
// This function will fold the sumM and make foldM to
// a matrix that has the following feature:
// sumM[J] = { \sum_{l1<=i<l2} mat[i][J] }
func calFoldMatrix(foldM, sumM, l1, l2){
    // Using the feature of prefix sum algorithm.
    for(j){
        // Explaination for this line will be given in next code block.
        foldM[j] = sumM[l2-1][j] - sumM[l1-1][j];
    }
    return;
}
```

- You can calculate $M^{fold}$ in $O(N^3)$ or $O(N^2)$, and I use $O(N^2)$ here using the prefix sum's idea (just like what I do in $N^4$ method, but only one dimension here, you can see details in code comments).

```
// This function will preprocess the sumM and make it to
// a matrix that has the following feature:
// sumM[I][J] = ${ \sum^{I}_{i} mat[i][J] }$
func initSumMatrix(sumM, mat){
    for(j = 1; j <= mat->m; ++j){
        sumM[0][j] = 0;
    }
    // sumM[I][J] = ${ \sum^{I}_{i} mat[i][J] }$
    //            = sumM[I-1][J] + mat[I][J]
    for(i, j){
        sumM[i][j] = sumM[i-1][j] + mat[i][j];
        // That is, if we move the elements, we will get
        // M_ELE(foldM, 1, j) = M_ELE(sumM, l2-1, j) - M_ELE(sumM, l1-1, j);
        // in the code block above.
    }
    return;
}
```

- Now we can see, to calculate the answer, we should do following things.

```
// This function will calculate the max sum of the sub-Matrix
// in mat.
func calMaxSubMatrixSum(mat){
    // Iterate the upper and lower bounds, then change the problem
    // into max sub-segment sum.
    int l1, l2;
    for(l1, l2){
        calFoldMatrix(foldM, sumM, l1, l2);
        tmp = calMaxSubSegmentSum(foldM);
        ret = max(ret, tmp);
    }
    return ret;
}
```

- I use $N^2$ memory to store original matrix, the prefix sum matrix, and N memory for fold matrix.So the time complexity should be $O(N^3)$ and the space complexity should be $O(N^2)$.

## 4 Specifications of main data structures

- I defined a matrix struct here to contain the matrix through the solution. Here is the defination of the struct. (You can see it in `matrix_utils.h`)

```
typedef struct MatrixStruct{
    int n, m;
    int * elePtr;
} Matrix;
```

- Actually, it should be initialized as a struct that has a pointer that points to a space of memory of $(n+1)*(m+1)$.
- Also, in order to manipulate the struct, I defined several macro functions. And I had already written enough code comments over them, so check `matrix_utils.h` if you want to know the details.

# Chapter 3: Testing Results

## 1 Methods

- In order to ensure the correctness of my algorithm, I made a data generator and a compare script.
- Specifically, the script will give the generator the size of the output, and the generator will construct random data (here are matrix with random elements).

```
# in _setup.sh
for SIZEN in 5 10 15 20 25 30 35 40 50 60 80 100
do
    ./data_maker.exe $SIZEN $SIZEN
    mv test_data.txt ./data_maker/$SIZEN.in
done
```

```
// in data_maker.cpp
int main(int argc, char * argv[]){
    srand((unsigned)time(NULL));
    int n = 4, m = 4;
    // Get size from arguments.
    if(argc == 3){
        n = atoi(argv[1]), m = atoi(argv[2]);
    }
    ofstream of;
    of.open("test_data.txt",ios::out | ios::trunc);
    int cnt = 10;
    // States the number of test cases.
    of << cnt << "\n";
    for(int cc = 0; cc < cnt; ++cc){
        of << n << " " << m << "\n";
        for(int i = 1; i <= n; ++i){
            for(int j = 1;j <= m;++j){
                // The MAX is a macro variable.
                of << rand()%MAX - MAX/2 << " ";
            }
        of << "\n";
        }
    }
    return 0;
}
```

- After that, I also write script to run the codes and compare the output of each methods.

```
# in _setup.sh
for SIZEN in 5 10 15 20 25 30 35 40 50 60 80 100
do
    for METHOD in N3 N4 N6
    do
        # Redirect the input and the output, store the output particularly.
        ./$METHOD.exe < ./data_maker/$SIZEN.in >>
./data_maker/$SIZEN.$METHOD.out
    done

    echo "Check out the answer while N = $SIZEN!"
```

```
    # Compare the three output and give the judgment.
    if diff ./data_maker/$SIZEN.N3.out ./data_maker/$SIZEN.N4.out; then
        if diff ./data_maker/$SIZEN.N3.out ./data_maker/$SIZEN.N6.out; then
            echo "Accept!"
        else
            echo "Wrong Answer!"
            exit 0
        fi
    else
        echo "Wrong Answer!"
        exit 0
    fi
done
```

# 2 Sample Cases

## 2.1 Random 5*5 Cases (Pass)

- Random test case generated by the `data_maker.exe`.

```
input:                    |  output:
--------------------------|--------------------------
1                         |  N3: 1486
5 5                       |  N4: 1486
336 -459 -178 408 -218    |  N6: 1486
423 -172 -337 -88 -239    |
-119 251 -186 -273 263    |
380 -494 -36 -117 -437    |
466 88 41 -386 -429       |
```

- As you can see, all my three methods give the same answer. The strategy is `m[1][1]` to `m[5][1]` (i.e. 336+423-119+380+466=1486).

# 3 How to ensure the correctness?

- You can run the `_setup.sh` for any times as you want, and in my choice, I get the following result.

```
......
Start testing codes! Don't edit output.txt!
Check out the answer while N = 5!
Accept!
Check out the answer while N = 10!
Accept!
Check out the answer while N = 15!
Accept!
Check out the answer while N = 20!
Accept!
Check out the answer while N = 25!
```

```
Accept!
Check out the answer while N = 30!
Accept!
Check out the answer while N = 35!
Accept!
Check out the answer while N = 40!
Accept!
Check out the answer while N = 50!
Accept!
Check out the answer while N = 60!
Accept!
Check out the answer while N = 80!
Accept!
Check out the answer while N = 100!
Accept!
Start drawing figure!
......
```

# Chapter 4: Analysis and Comments

## 1 Analysis

- $N^6$: The time complexity should be $O(N^6)$ and the space complexity should be $O(N^2)$.
- $N^4$: The time complexity should be $O(N^4)$ and the space complexity should be $O(N^2)$.
- $N^3$: The time complexity should be $O(N^3)$ and the space complexity should be $O(N^2)$.

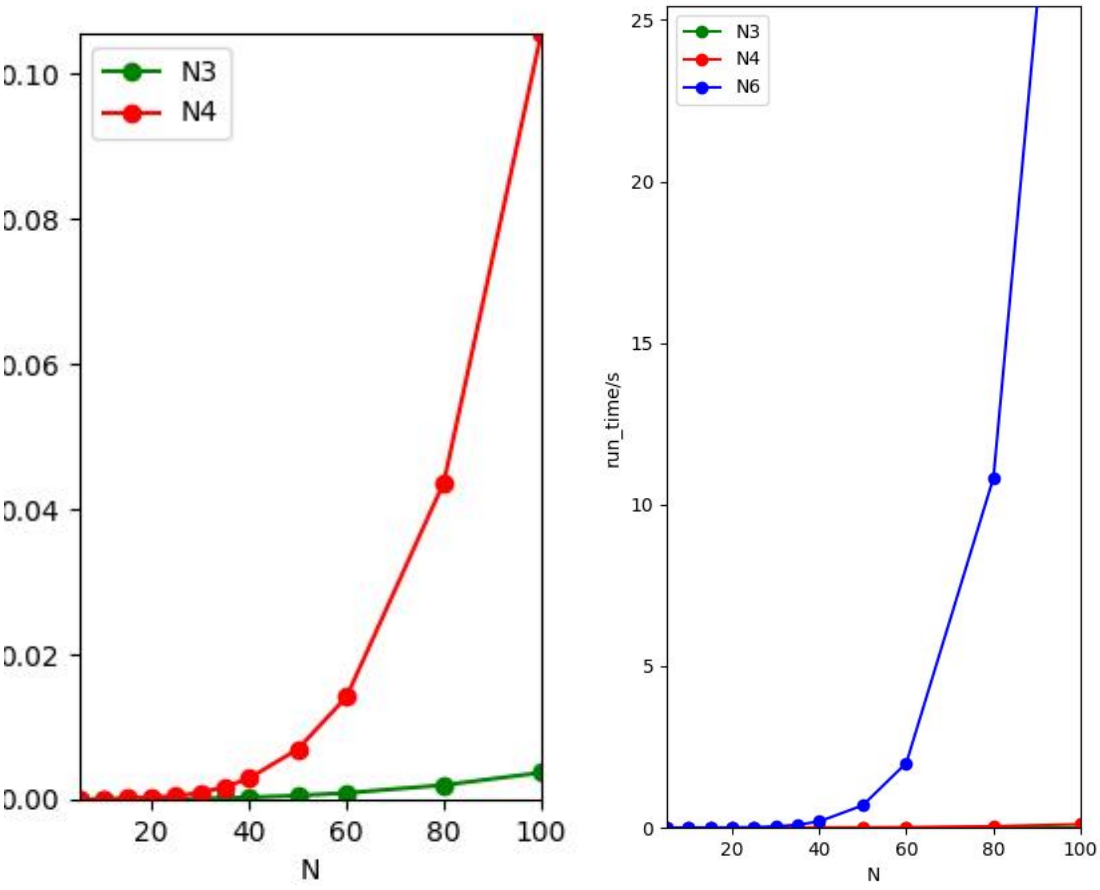
### 1.1 Table

- Data following is gotten on Mac(M1).
- In order to make the data more reasonable and observable, I will generate 10 ~~different~~ input data. All the methods will run with the same 10 different input.

| 0() | opt | 5 | 10 | 30 | 50 | 80 | 100 |
|---|---|---|---|---|---|---|---|
| O($N^6$) | Iterations(K) | 10 | 10 | 10 | 10 | 10 | 10 |
| | Ticks | totTime*1000 | totTime*1000 | totTime*1000 | totTime*1000 | totTime*1000 | totTime*1000 |
| | Total Time(sec) | 0.000126 | 0.002742 | 0.367200 | 6.892734 | 108.260617 | 400.305024 |
| | Duration(sec) | 0.000013 | 0.000022 | 0.036720 | 0.689273 | 10.826062 | 40.030502 |
| O($N^4$) | Iterations(K) | 10 | 10 | 10 | 10 | 10 | 10 |
| | Ticks | totTime*1000 | totTime*1000 | totTime*1000 | totTime*1000 | totTime*1000 | totTime*1000 |
| | Total Time(sec) | 0.000065 | 0.000439 | 0.009535 | 0.069383 | 0.436988 | 1.055402 |
| | Duration(sec) | 0.000006 | 0.000044 | 0.000954 | 0.006938 | 0.043699 | 0.105540 |
| O($N^3$) | Iterations(K) | 10 | 10 | 10 | 10 | 10 | 10 |
| | Ticks | totTime*1000 | totTime*1000 | totTime*1000 | totTime*1000 | totTime*1000 | totTime*1000 |
| | Total Time(sec) | 0.000039 | 0.000218 | 0.001245 | 0.005501 | 0.019971 | 0.037084 |
| | Duration(sec) | 0.000004 | 0.000022 | 0.000124 | 0.000550 | 0.001997 | 0.003708 |

## 1.2 Figure

- $O(N^3)$ vs $O(N^4)$ and $O(N^3)$ vs $O(N^4)$ vs $O(N^6)$



## 2 Comments

- We can see that, the best method given to solve the max sub-matrix is based on max sub-segment. That means, we reduce the dimension of the data and perform the O(N) method for one-dimensional case. In another words, if we want to calculate the max sub-cube or cases in even higher dimension, we could try to reduce the dimension of the data just like what we do here, although it seems horrible that each dimension should cost $O(N^2)$ to reduce.
- But the option to calculate the prefix sum will be quite slow when we are in a high dimension. So maybe we can use some technology such as parallel computing to accelerate it.

# Appendix

## 1 Index

**All source codes are in the current folder.**

- main algorithm codes: `./src`
  - entrance: `./src/main.c`
  - matrix utils (macro functions): `./src/matrix_utils.h`
  - logger helper: `./src/helper/*` (hidden because of the rule)
  - methods implementation: `./src/N?.c` `./src/N?.h` ($? \in 3, 4, 6$)
- test data generator: `./data_maker/maker.cpp`
- figure drawing tool: `./analyzer/*`
- setup shell script: `./_setup.sh`
- clear shell script: `./_clear.sh`
- instructions: `./Readme.md`

## 2 main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// #include "helper/CNewbieHelper.h"
#include "matrix_utils.h"
#include "N3.h"
#include "N4.h"
#include "N6.h"

// Do one round.
double foo(int round){
    Matrix matrixObj, *mat = &matrixObj;
    M_INIT(mat);
```

```
15
16      clock_t timerI = clock();
17      int ans = calMaxSubMatrixSum(mat);
18      clock_t timerF = clock();
19      double delTime = (double)(timerF-timerI)/CLOCKS_PER_SEC;
20
21      printf("%d\n", ans);
22      // printf("Program Finished in %.4lf seconds!", (double)(timerF-
    timerI)/CLK_TCK);
23
24      FILE * fp = fopen("output_details.txt", "a");
25      fprintf(fp, "Method: N%d,  Size: %d*%d,  Round: %d\n  The ans is: %d \n
    The delta time is: %.6lf\n", getMethod(),  mat->n, mat->m, round, ans,
    delTime);
26      fclose(fp);
27
28      M_DEL(mat);
29      return delTime;
30  }
31
32  int main(){
33      // Logger util.
34      // SET_CNH_SHOW(1);
35      // SET_CNH_BRIEF_MODE(1);
36      // LOG("Program Started!");
37
38      int cnt;
39      scanf("%d", &cnt);
40
41      clock_t timerI = clock();
42
43      double tot = 0;
44      int i;
45      for(i = 0;i < cnt;++i){
46          double delTime = foo(i);
47          // tot += delTime;
48      }
49
50      clock_t timerF = clock();
51
52      tot = (double)(timerF-timerI)/CLOCKS_PER_SEC;
53      FILE * fp = fopen("output.txt", "a");
54      fprintf(fp, "%.6lf %.6lf \n", tot, tot/cnt);
55      fclose(fp);
56      return 0;
57  }
```

## 3 matrix_utils.h

```
#ifndef __MATRIX_UTILS__
#define __MATRIX_UTILS__

#include "stdio.h"
#include "stdlib.h"

/*****************************************************
 * Matrix Utils Lib 1.0                              *
 * ------------------------------------------------- *
 * Last Edit: 2022.9.22                              *
 *****************************************************/

// Macro Function:
// - The function returns the left value of the elements in the
// - matrix indexed by the "pos_x" and "pos_y".
//
// Usage:
// - Just use M_ELE(mat_ptr, pos_x, pos_y) as a right value or
// - left value.
// - eg: [ M_ELE(mat_ptr, pos_x, pos_y) = ... ]
// - "mat_ptr" here is the a pointer to a Matrix type variable.
// - "pos_x" and "pos_y" here is the position of the target elements.
#define M_ELE(MATRIX_P, POS_N, POS_M) *(MATRIX_P->elePtr + MATRIX_P->m *
(POS_N) + (POS_M))

// Macro Function:
// - This function is used to allocate the memory of the
// - (only) elements of Matrix type variable.
//
// Usage:
// - [ M_NEW(mat_ptr); ]
// - "mat_ptr" here is the a pointer to a Matrix type variable.
#define M_NEW(MATRIX_P) do{
                \
    MATRIX_P->elePtr = (int * )malloc( (MATRIX_P->n+1) * (MATRIX_P->m+1) *
sizeof(int) );   \
}while(0);

// Macro Function:
// - This function is used to deal with the memory of the
// - (only) elements of Matrix type variable.
// - That means, you should deal with the memory pointed by
// - the "mat_ptr" directly!
//
// Usage:
```

```
// - [ M_DEL(mat_ptr); ]
// - "mat_ptr" here is the a pointer to a Matrix type variable.
#define M_DEL(MATRIX_P) do{ \
    free(MATRIX_P->elePtr); \
}while(0);

// Macro Function:
// - This function is used to initialize a matrix with data
// - given from console.
// - First it reads two integers, n and m, which represent
// - the shape of the matrix.
// - Then it reads the elements of the matrix, which are
// - organized as n rows and m columns.
// - It will also allocate memory for it.
//
// Usage:
// - [ M_INIT(mat_ptr); ]
// - "mat_ptr" here is the a pointer to a Matrix type variable.
#define M_INIT(mat) do{ \
    /* Read in the shape of matrix. */ \
    scanf("%d %d", &(mat->n), &(mat->m)); \
    /* Allocate the memory. */ \
    M_NEW(mat); \
    /* Read in the elements one by one. */ \
    int read_mat_it1, read_mat_it2; \
    for(read_mat_it1 = 1; read_mat_it1 <= mat->n; ++read_mat_it1){ \
        for(read_mat_it2 = 1; read_mat_it2 <= mat->m; ++read_mat_it2){ \
            scanf("%d", &M_ELE(mat, read_mat_it1, read_mat_it2)); \
        } \
    } \
}while(0)\

#define M_SHOW(mat) do{ \
    int read_mat_it1, read_mat_it2; \
    printf("n: %d, m: %d;\n[\n", mat->n, mat->m); \
    for(read_mat_it1 = 1; read_mat_it1 <= mat->n; ++read_mat_it1){ \
        for(read_mat_it2 = 1; read_mat_it2 <= mat->m; ++read_mat_it2){ \
            printf(" %d", M_ELE(mat, read_mat_it1, read_mat_it2)); \
        } \
        printf("\n"); \
    } \
    printf("]\n"); \
}while(0)

// Macro Function:
// - This function is used to copy the shape of matrix
// - from "fromMat" to "toMat". And also allocate the
```

```
// - memory.
//
// Usage:
// - [ M_CP_SHAPE(toMat, fromMat); ]
// - "toMat" here is the a pointer to the Matrix type variable
// - to be modified, while "fromMat" provides the shape.
#define M_CP_SHAPE(toMat, fromMat) do{ \
    toMat->m = fromMat->m, toMat->n = fromMat->n; \
    M_NEW(toMat); \
}while(0)


typedef struct MatrixStruct{
    int n, m;
    int * elePtr;
} Matrix;

#endif
```

## 4 N3.h

```
#ifndef __MAX_SUB_MATRIX_SUM_N3__
#define __MAX_SUB_MATRIX_SUM_N3__

#include "stdlib.h"
#include "matrix_utils.h"
// #include "helper/CNewbieHelper.h"

int getMethod();
int calMaxSubMatrixSum(Matrix * mat);

#endif
```

## 5 N3.c

```
#include "N3.h"

// This function will preprocess the sumM and make it to
// a matrix that has the following feature:
// sumM[I][J] = ${ \sum^{I}_{i} mat[i][J] }$
void initSumMatrix(Matrix * sumM, Matrix * mat){
    M_CP_SHAPE(sumM, mat);
    int i, j;
    for(j = 1; j <= mat->m; ++j){
        M_ELE(sumM, 0, j) = 0;
```

```
    }
    // sumM[I][J] = ${ \sum^{I}_{i} mat[i][J] }$
    //            = sumM[I-1][J] + mat[I][J]
    for(i = 1; i <= mat->n; ++i){
        for(j = 1; j <= mat->m; ++j){
            M_ELE(sumM, i, j) = M_ELE(sumM, i-1, j) + M_ELE(mat, i, j);
        }
    }
    return;
}

// This function will fold the sumM and make foldM to
// a matrix that has the following feature:
// sumM[J] = { \sum_{l1<=i<l2} mat[i][J] }
void calFoldMatrix(Matrix * foldM, Matrix * sumM, int l1, int l2){
    int j;
    // Using the feature of prefix sum algorithm.
    for(j = 1; j <= sumM->m; ++j){
        M_ELE(foldM, 1, j) = M_ELE(sumM, l2-1, j) - M_ELE(sumM, l1-1, j);
    }
    return;
}

// This function will calculate the max sum of the sub-segment
// in vec.
int calMaxSubSegmentSum(Matrix * vec){
    int j, curSum = 0, maxSum = 0;
    for(j = 1; j <= vec->m; ++j){
        curSum += M_ELE(vec, 1, j);
        curSum = curSum < 0 ? 0 : curSum;           // curSum = max(curSum,
0);
        maxSum = curSum > maxSum ? curSum : maxSum; // maxSum = max(maxSum,
curSum);
    }
    return maxSum;
}

// This function will calculate the max sum of the sub-Matrix
// in mat.
int calMaxSubMatrixSum(Matrix * mat){
    // Logger util.
    // SET_CNH_SHOW(0);
    // SET_CNH_BRIEF_MODE(1);

    int ret = -0x3FFFFFFF;

    // This matrix is to store the 2d-prefix sum on the n-axis.
```

```
    // That is, sumM[I][J] = ${ \sum^{I}_{i} mat[i][J] }$
    Matrix sumMObj, * sumM = &sumMObj;
    initSumMatrix(sumM, mat);
    // M_SHOW(sumM); // For debug.

    // This matrix is to store the segment sum of mat on the
    // n-axis, calculated from sumM.
    // That is, foldM(l1,l2)[J] = ${ \sum_{l1<=i<l2} mat[i][J] }$
    Matrix foldMObj, * foldM = &foldMObj;
    foldM->n = 1, foldM->m = mat->m;
    M_NEW(foldM);

    // Iterate the upper and lower bounds, then change the problem
    // into max sub-segment sum.
    int l1, l2;
    for(l1 = 1; l1 <= mat->n; ++l1){
        for(l2 = l1+1; l2 <= mat->n+1; ++l2){
            calFoldMatrix(foldM, sumM, l1, l2);
            int tmp = calMaxSubSegmentSum(foldM);
            ret = tmp > ret ? tmp : ret;    // ret = max(ret, tmp);
        }
    }

    return ret;
}

int getMethod(){
    return 3;
}
```

# 6 N4.h

```
#ifndef __MAX_SUB_MATRIX_SUM_N4__
#define __MAX_SUB_MATRIX_SUM_N4__

#include "stdlib.h"
#include "matrix_utils.h"

int getMethod();
int calMaxSubMatrixSum(Matrix * mat);

#endif
```

# 7 N4.c

```c
#include "N4.h"

// This function will preprocess the sumM and make it to
// a matrix that has the following feature:
// sumM[I][J] = ${ \sum^{I,J}_{i,j} mat[i][j] }$
void initSumMatrix(Matrix * sumM, Matrix * mat){
    M_CP_SHAPE(sumM, mat);
    int i, j;
    for(j = 1; j <= mat->m; ++j){
        M_ELE(sumM, 0, j) = 0;
    }
    for(i = 1; i <= mat->n; ++i){
        M_ELE(sumM, i, 0) = 0;
    }
    // sumM[I][J] = ${ \sum^{I,J}_{i,j} mat[i][j] }$
    //            = sumM[I-1][J] + sumM[I-1][J-1] - sumM[I-1][J-1] + mat[I][J]
    for(i = 1; i <= mat->n; ++i){
        for(j = 1; j <= mat->m; ++j){
            M_ELE(sumM, i, j) = M_ELE(sumM, i-1, j) + M_ELE(sumM, i, j-1) - M_ELE(sumM, i-1, j-1) + M_ELE(mat, i, j);
        }
    }
}

// This function will calculate the max sum of the sub-Matrix
// in mat.
int calMaxSubMatrixSum(Matrix * mat){
    // This matrix is to store the 2d-prefix sum on the both
    // n-axis and m-axis.
    // That is, sumM[I][J] = ${ \sum^{I,J}_{i,j} mat[i][j] }$
    Matrix sumMObj, * sumM = &sumMObj;
    initSumMatrix(sumM, mat);
    // M_SHOW(sumM); // For debug.

    int i, j, k, l, x, y, ans = -0x3F3F3F3F;
    // Iterate the left-top key point.
    for(i = 0; i < mat->n; ++i){
        for(j = 0; j < mat->m; ++j){
            // Iterate the right-bottom key point.
            for(k = i+1; k <= mat->n; ++k){
                for(l = j+1; l <= mat->m; ++l){
                    int tmp = 0;
                    // Calculate the sum in region from [k][l] to [i+1][j+1].
```

```
                    tmp = M_ELE(sumM, k, l) + M_ELE(sumM, i, j) -
M_ELE(sumM, k, j) - M_ELE(sumM, i, l);
                    // Update the answer if we get larger sum.
                    ans = tmp > ans ? tmp : ans;
                }
            }
        }
    }
    return ans;
}

int getMethod(){
    return 4;
}
```

## 8 N6.h

```
#ifndef __MAX_SUB_MATRIX_SUM_N6__
#define __MAX_SUB_MATRIX_SUM_N6__

#include "stdlib.h"
#include "matrix_utils.h"

int getMethod();
int calMaxSubMatrixSum(Matrix * mat);

#endif
```

## 9 N6.c

```
#include "N6.h"

// This function will calculate the max sum of the sub-Matrix
// in mat.
int calMaxSubMatrixSum(Matrix * mat){
    int i, j, k, l, x, y, ans = -0x3F3F3F3F;
    // Iterate the left-top key point.
    for(i = 0; i < mat->n; ++i){
        for(j = 0; j < mat->m; ++j){
            // Iterate the right-bottom key point.
            for(k = i+1; k <= mat->n; ++k){
                for(l = j+1; l <= mat->m; ++l){
                    int tmp = 0;
                    // Calculate the sum in region from [k][l] to [i+1]
[j+1].
                    for(x = i+1; x <= k; ++x){
                        for(y = j+1; y <= l; ++y){
```

```
                            tmp += M_ELE(mat, x, y);
                        }
                    }
                    // Update the answer if we get larger sum.
                    ans = tmp > ans ? tmp : ans;
                }
            }
        }
    }
    return ans;
}

int getMethod(){
    return 6;
}
```

## 10 maker.cpp

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <random>

using namespace std;

#define MAX 1024

int main(int argc, char * argv[]){
    srand((unsigned)time(NULL));
    int n = 4, m = 4;
    if(argc == 3){
        n = atoi(argv[1]);
        m = atoi(argv[2]);
    }
    ofstream of;
    of.open("test_data.txt",ios::out | ios::trunc);
    int cnt = n <= 50 ? 100 : 10;
    of << cnt << "\n";
    for(int cc = 0; cc < cnt; ++cc){
        of << n << " " << m << "\n";
        for(int i = 1; i <= n; ++i){
            for(int j = 1;j <= m;++j){
                of << rand()%MAX - MAX/2 << " ";
            }
        of << "\n";
        }
```

```
    }

}
```

## 11 analyze.py

```python
import matplotlib.pyplot as plt
import numpy as np

def foo():
    # read in the data
    data = ""
    with open("output.txt", "r") as f:
        data = f.read()

    data = data.split(' \n')
    arrN = [5, 10, 15, 20, 25, 30, 35, 40, 50, 60, 80, 100]
    arrT = np.zeros((3, len(arrN)), dtype=np.float)
    arrTT = np.zeros((3, len(arrN)), dtype=np.float)

    for itr in range(len(arrN)):
        for met in range(3):
            s = data[3*itr+met].split(' ')
            print(s)
            arrTT[met][itr] = float(s[0])
            arrT[met][itr] = float(s[1])

    # draw picture of N3 vs N4
    plt.figure(figsize=(3, 5))
    plt.xlim(5, 100)
    plt.xlabel("N")
    plt.ylim(0, arrT[1][len(arrN)-1])
    plt.ylabel("run_time/s")
    plt.plot(arrN, arrT[0], color='g', marker='o', label='N3')
    plt.plot(arrN, arrT[1], color='r', marker='o', label='N4')
    plt.legend()
    plt.savefig("N3vsN4.jpg")

    # draw picture of N3 vs N4 vs N6
    plt.figure(figsize=(4, 8))
    plt.xlim(5, 100)
    plt.xlabel("N")
    plt.ylim(0, 0.5*(arrT[2][len(arrN)-1]+arrT[2][len(arrN)-2]))
    plt.ylabel("run_time/s")
    plt.plot(arrN, arrT[0], color='g', marker='o', label='N3')
    plt.plot(arrN, arrT[1], color='r', marker='o', label='N4')
    plt.plot(arrN, arrT[2], color='b', marker='o', label='N6')
```

```
    plt.legend()
    plt.savefig("N3vsN4vsN6.jpg")

    plt.show()

foo()
```

## 12 _setup.sh

```
echo "You are supposed to run this scrips with bash!"

# ============================================================
# Compile and link the C codes.

echo "Start compiling algorithm codes!"

gcc -c ./src/main.c -o ./src/main.o

for METHOD in N3 N4 N6
do
    gcc -c ./src/$METHOD.c -o ./src/$METHOD.o
    gcc -o $METHOD.exe ./src/main.o ./src/$METHOD.o
done

echo "Done!"

# ============================================================
# Compile the data maker and generate the test data.

echo "Start generating test data!"

g++ ./data_maker/maker.cpp -o data_maker.exe -std=c++11

for SIZEN in 5 10 15 20 25 30 35 40 50 60 80 100
do
    ./data_maker.exe $SIZEN $SIZEN
    mv test_data.txt ./data_maker/$SIZEN.in
done

echo "Done!"

# ============================================================
# Test the codes and store the necessary data.

echo "Start testing codes! Don't edit output.txt!"

rm output.txt || true
```

```
touch output.txt
rm output_details.txt || true
touch output_details.txt
rm ./data_maker/*.out || true

for SIZEN in 5 10 15 20 25 30 35 40 50 60 80 100
do
    for METHOD in N3 N4 N6
    do
        # Redirect the input and the output, store the output particularly.
        ./$METHOD.exe < ./data_maker/$SIZEN.in >>
./data_maker/$SIZEN.$METHOD.out
    done

    echo "Check out the answer while N = $SIZEN!"
    # Compare the three output and give the judgment.
    if diff ./data_maker/$SIZEN.N3.out ./data_maker/$SIZEN.N4.out; then
        if diff ./data_maker/$SIZEN.N3.out ./data_maker/$SIZEN.N6.out; then
            echo "Accept!"
        else
            echo "Wrong Answer!"
            exit 0
        fi
    else
        echo "Wrong Answer!"
        exit 0
    fi
done

# ============================================================
# Draw the figure.

echo "Start drawing figure!"

python ./analyzer/analyze.py

# ============================================================

echo "Every thing done!"
```

## 13 _clear.sh

```
rm ./data_maker/*.in || true
rm ./data_maker/*.out || true
rm ./*.exe || true
rm ./src/*.o || true
```

# Declaration

I hereby declare that all the work done in this project titled "Performance Measurement" is of my independent effort.