MAXIMUM SUBMATRIX SUM PROBLEM

****

Date: 2021-10-4

第一章 导言

1.1 问题背景

最大子数列和问题是一个经典的算法问题。其目标是在数列中找到一个子列，这个子列中全部元素的和是所有子列中最大的。而本次实验将问题拓展到二维，要求在给定的数组中寻找最大子矩阵和。

1.2 问题描述

给定一个$N \times N$的整数矩阵$(a_{ij})_{N\times N}$，找出对所有的$1 \le i \le m \le N$和$1 \le j \le n \le N$，表达式$\sum\limits_{k=i}^{m}\sum\limits_{l=j}^{n}a_{kl}$的最大值。方便起见，如果所有的整数都是负整数，则认为最大子矩阵和为0。

例如：对于矩阵

$$\begin{bmatrix} 0 & -2 & -7 & 0 \\ 9 & 2 & -6 & 2 \\ -4 & 1 & -4 & 1 \\ -1 & 8 & 0 & -2 \end{bmatrix},$$

最大子矩阵和为15，对应的子矩阵为

$$\begin{bmatrix} 9 & 2 \\ -4 & 1 \\ -1 & 8 \end{bmatrix}.$$

第二章 算法说明

2.1 $O(N^6)$算法

最直接的方法就是计算每一个子矩阵的和，从中找到最大值。

```
function algo0(matrix)
begin
    for x:=0 to N-1 do
    begin
        for y:=0 to N-1 do
        begin
            for l:=1 to N-x do
            begin
                for h:=1 to N-y do
                begin
                    thisSum=0
                    for i:=x to x+l-1 do
                    begin
                        for j:=y to y+h-1 do
                        begin
                            thisSum:=thisSum+matrix[j][i]
                        end
                    end
                    if thisSum>maxSum then maxSum = thisSum
                end
            end
        end
    end
end
```

- thisSum记录当前子矩阵所有元素之和，maxSum记录当前最大子矩阵和。
- (x, y)表示被求和的子矩阵左上角顶点的坐标（以原矩阵左上角为原点，横向为x轴，纵向为y轴）。
- l表示被求和子矩阵的宽度，即在x轴方向上的投影。
- h表示被求和子矩阵的高度，即在y轴方向上的投影。
- (i, j)表示要计入thisSum的元素坐标。

2.2 $O(N^4)$算法

与优化最大子数列和算法的方法一样，在$O(N^4)$算法中，通过利用已经计算过的和来减少计算量，从而达到降低算法时间复杂度的目的。

```
function algo1(matrix)
begin
    for x:=0 to N-1 do
    begin
        for y:=0 to N-1 do
        begin
            for l:=1 to N-x do
            begin
                for h:=1 to N-y do
                begin
                    sum[y+h][x+l]:=the sum of the submatrix determined by (x,y,h,l)
                    thisSum:=sum[y+h][x+l]
                    if thisSum>maxSum then maxSum = thisSum
                end
            end
        end
    end
end
```

- `sum[y+h][x+l]=sum[y+h][x+l]+sum[y+h-1][x+l]+sum[y+h][x+l-1]-sum[y+h-1][x+l-1]`，可以利用容斥的方法通过常数时间求得子矩阵和。这样就避免了重复计算，从而降低时间复杂度。

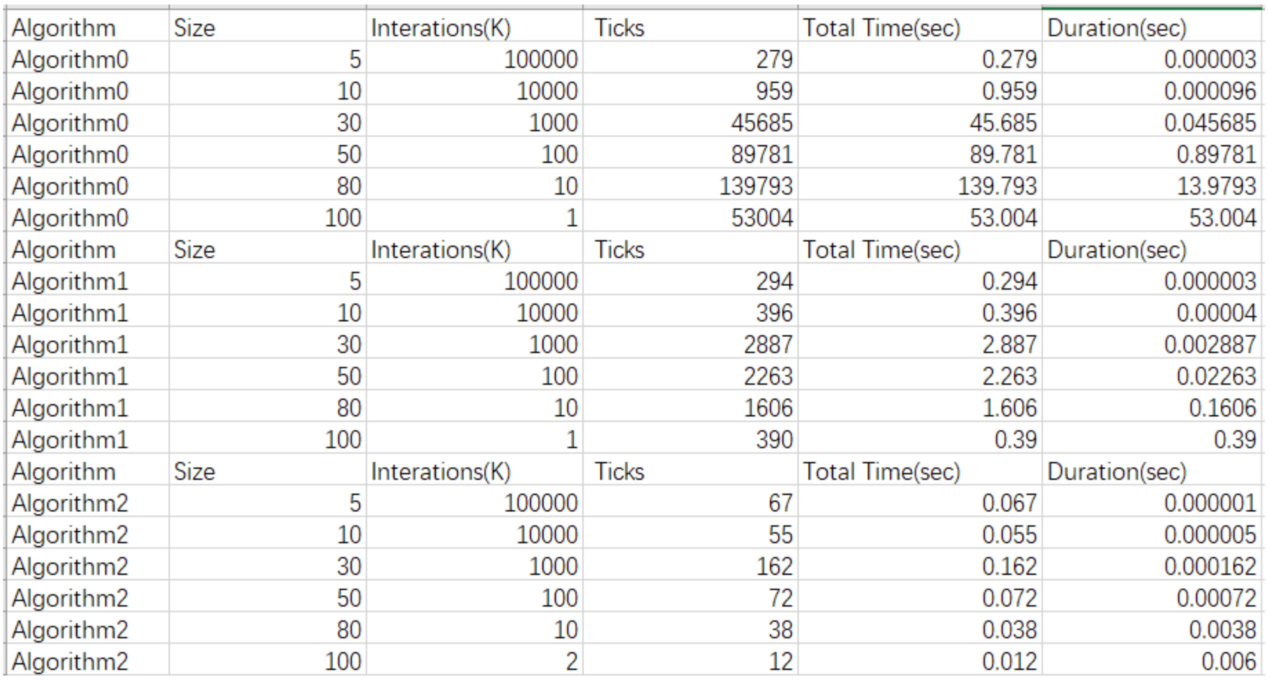
2.3 $O(N^3)$算法

确定子矩阵左上角顶点的横坐标，并将对应的子矩阵按照长度分类。对于固定了左上角顶点和长度的子矩阵，可以将其每一行的元素加起来，形成一个一维的数列。从而可以利用线性时间求出最大子数列和，即这一类子矩阵中最大的矩阵和，进而可以得出所有子矩阵中最大的矩阵和。

```
 1  function algo2(matrix)
 2  begin
 3      for x:=0 to N-1 do
 4      begin
 5          sum[][]:=matrix[][]
 6          for j:=0 to N-1 do
 7          begin
 8              for i:=x to N-1 do
 9              begin
10                  sum[j][i]:=sum[j][i]+sum[j][i-1]
11              end
12          end
13          for i:=x to N-1 do
14          begin
15              thisSum:=the maximum subsequence sum
16              if thisSum>maxSum then maxSum = thisSum
17          end
18      end
19  end
```

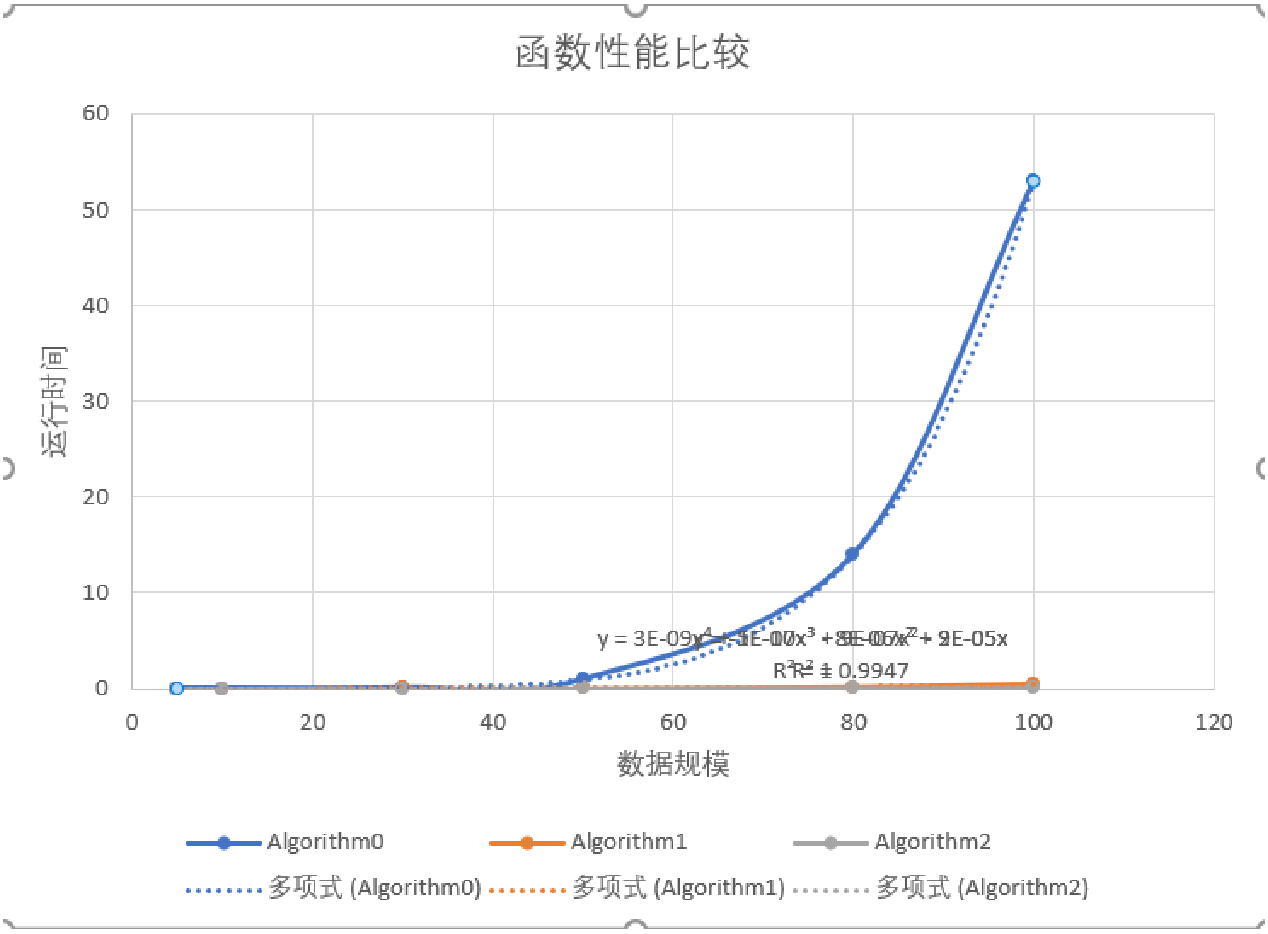
- 第3行：将矩阵复制给`sum`，时间复杂度$O(N^2)$
- 第8行：使每一个位置的值为从子矩阵行首到此处的元素和。
- 第13行：将这一列最大子数列和赋值给`thisSum`，时间复杂度$O(N^2)$

第三章 测试结果

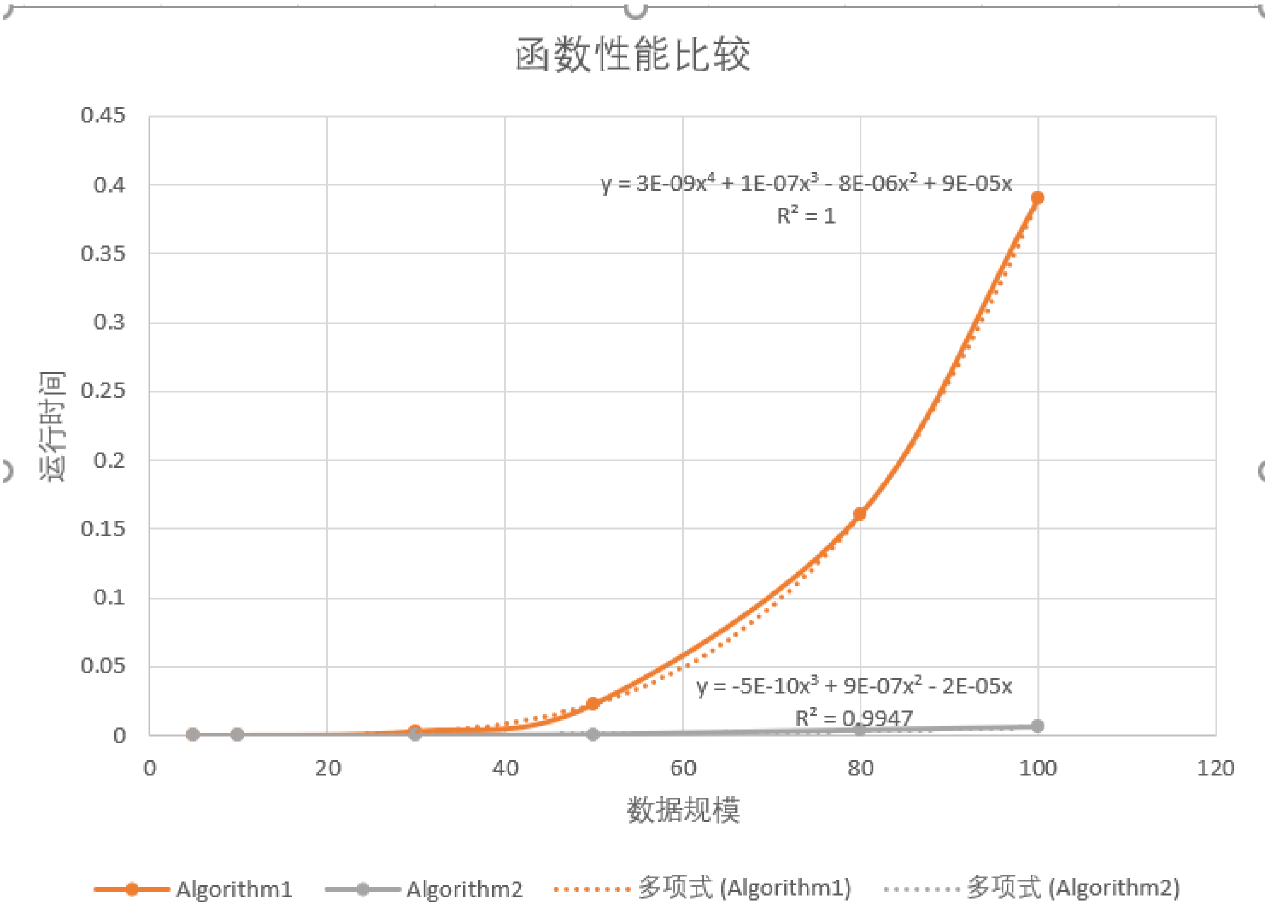


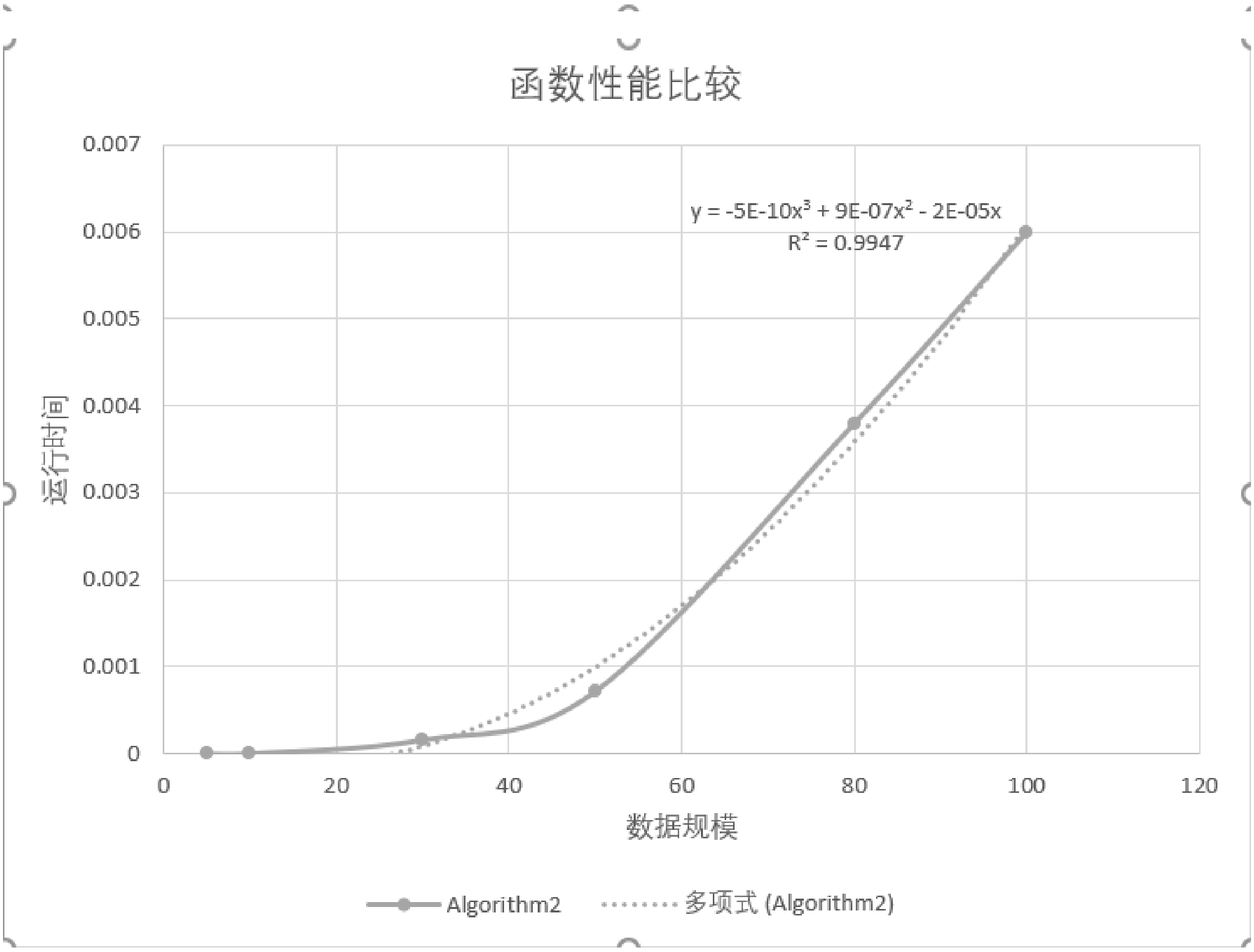
Algorithm	Size	Interations(K)	Ticks	Total Time(sec)	Duration(sec)
Algorithm0	5	100000	279	0.279	0.000003
Algorithm0	10	10000	959	0.959	0.000096
Algorithm0	30	1000	45685	45.685	0.045685
Algorithm0	50	100	89781	89.781	0.89781
Algorithm0	80	10	139793	139.793	13.9793
Algorithm0	100	1	53004	53.004	53.004
Algorithm	Size	Interations(K)	Ticks	Total Time(sec)	Duration(sec)
Algorithm1	5	100000	294	0.294	0.000003
Algorithm1	10	10000	396	0.396	0.00004
Algorithm1	30	1000	2887	2.887	0.002887
Algorithm1	50	100	2263	2.263	0.02263
Algorithm1	80	10	1606	1.606	0.1606
Algorithm1	100	1	390	0.39	0.39
Algorithm	Size	Interations(K)	Ticks	Total Time(sec)	Duration(sec)
Algorithm2	5	100000	67	0.067	0.000001
Algorithm2	10	10000	55	0.055	0.000005
Algorithm2	30	1000	162	0.162	0.000162
Algorithm2	50	100	72	0.072	0.00072
Algorithm2	80	10	38	0.038	0.0038
Algorithm2	100	2	12	0.012	0.006

- 所有测试全部通过。



由于不同算法之间性能差异较大，导致运行速度较快的算法性能特点不能很好地展示，特给出以下两个图象，使算法的性能特点更好地展示出来。


函数性能比较
y = 3E-09x⁴ + 1E-07x³ - 8E-06x² + 9E-05x
R² = 1
y = -5E-10x³ + 9E-07x² - 2E-05x
R² = 0.9947
运行时间
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
20
40
60
80
100
120
数据规模
Algorithm1
Algorithm2
多项式 (Algorithm1)
多项式 (Algorithm2)


函数性能比较
y = -5E-10x³ + 9E-07x² - 2E-05x
R² = 0.9947
运行时间
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0
0
20
40
60
80
100
120
数据规模
Algorithm2
多项式 (Algorithm2)

第四章 分析和意见

4.1 算法分析

4.1.1 $O(N^6)$算法

- 时间复杂度：$O(N^6)$。使用了六层的嵌套循环。
- 空间复杂度：$O(1)$。

4.1.2 $O(N^4)$算法

- 时间复杂度：$O(N^4)$。使用了四层的嵌套循环。
- 空间复杂度：$O(N^2)$。需要一个$O(N^2)$大小的二维数组来记录已经计算过的结果。

4.2 优化意见

$O(N^3)$算法

对于高维结构的处理，一种常见的方式为对高维结构进行压缩，从而提高算法的性能。使用前面说明的$O(N^3)$算法，将寻找最大子矩阵和问题转化为寻找最大子序列和问题。从而使算法的时间复杂度降低到$O(N^3)$。该算法的空间复杂度为$O(N^2)$。

- 时间复杂度分析：最外层有一层循环，内部有两个两层的嵌套循环。
- 空间复杂度分析：需要一个$O(N^2)$的二维数组来记录矩阵压缩结果。

附录：源代码（C语言实现）

- solvers.h:

```c
#ifndef SOLVERS
#define SOLVERS
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

    #ifndef SUBMATRIX
    #define SUBMATRIX
    /*determine a submatrix exactly*/
    struct subMatrix{
        /*the coordinate of the top-left corner of the submatrix*/
        int x;
        int y;
        /*the length of the submatrix, which is its x-axis projection*/
        int l;
        /*the height of the submatrix, which is its y-axis rojecting*/
        int h;
        /*the maximum sum*/
        int maxsum;
    };
    /*pointer to the submatrix*/
    typedef struct subMatrix *Ptr2SubMatrix;
    #endif

    #ifndef MATRIX
    #define MATRIX
    /*represent the original matrix*/
    struct Matrix{
        /*the size of the original matrix, for instance, a n*n matrix is of size n*/
        int size;
        /*store the elements which is in the matrix*/
        int *Array;
    };
    /*pointer to the original matrix*/
    typedef struct Matrix *Ptr2Matrix;
    #endif

    #ifndef SOLVER
    #define SOLVER
    /*pointer to the solver functions*/
    /*return the maximum sum*/
    typedef void (*solver)(Ptr2Matrix matrix, Ptr2SubMatrix solution);
    #endif

 /*********************************************************************
    *   algorithm0: implement the algorithm run in O(N^6). Traserval *
    *               all possible submatrix and compare their sum with *
    *               the temporary maximum sum. *
 *********************************************************************/
    void algorithm0(Ptr2Matrix matrix, Ptr2SubMatrix solution);

 /*********************************************************************
    *   algorithm1: implement the algorithm run in O(N^4). Traserval *
    *               all possible submatrix and compare their sum with *
    *               the temporary maximum sum. But different with the *
    *               algorithm1, algorithm2 avoid to repeat some compute *
    *               to speed up. *
 *********************************************************************/
```

```
    void algorithm1(Ptr2Matrix matrix, Ptr2SubMatrix solution);

 /*********************************************************************
    *   algorithm2: implement the algorithm run in O(N^3). Divide all    *
    *               possible submatrix into four classes, with length of*
    *               1, 2, 3, 4, respectively. Each classes can be        *
    *               treated as an linear array.                          *
 *********************************************************************/
    void algorithm2(Ptr2Matrix matrix, Ptr2SubMatrix solution);
#endif
```

- solvers.c

```
#include "solvers.h"

void algorithm0(Ptr2Matrix matrix, Ptr2SubMatrix solution){
    int thisSum = 0;
    int MaxSum = 0;
    int n = matrix->size;
    int *a = matrix->Array;
    for(int x=0; x<n; x++){/*x represents the abscissa of the top-left corner of submatrix*/
        for(int y=0; y<n; y++){/*y represents the ordinate of the top-left corner of submatrix*/
            for(int l=1; l<=n-x; l++){/*l represents the length of the submatrix, which is its x-axis projection*/
                for(int h=1; h<=n-y; h++){/*h represents the height of the submatrix, which is its y-axis rojecting*/
                    /*get the sum of the submatrix determined by (x,y,l,h)*/
                    thisSum = 0;
                    for(int i=x; i<x+l; i++){/*represent the abscissa of an element*/
                        for(int j=y; j<y+h; j++){/*represent the ordinate of an element*/
                            thisSum += a[j*n+i];
                        }
                    }
                    if(thisSum>MaxSum){/*record the temporary submatrix with maximum sum*/
                        MaxSum = thisSum;
                        solution->h = h;
                        solution->l = l;
                        solution->x = x;
                        solution->y = y;
                    }
```

```c
                }
            }
        }
    }
    solution->maxsum = MaxSum;
}

void algorithm1(Ptr2Matrix matrix, Ptr2SubMatrix solution){
    int thisSum = 0;
    int MaxSum = 0;
    int n = matrix->size;
    int *a = matrix->Array;
    int sum[n+1][n+1];/*used to record the sums which has been calculated*/

    for(int x=0; x<n; x++){/*x represents the abscissa of the top-left corner of submatrix*/
        for(int y=0; y<n; y++){/*y represents the ordinate of the top-left corner of submatrix*/
            /*initialize the sum array*/
            for(int i=0; i<n+1; i++){/*represent the abscissa of an element*/
                for(int j=0; j<n+1; j++){/*represent the ordinate of an element*/
                    sum[i][j] = 0;
                }
            }
            for(int i=x+1; i<n+1; i++){/*represent the abscissa of an element*/
                for(int j=y+1; j<n+1; j++){/*represent the ordinate of an element*/
                    sum[j][i] = a[(j-1)*n+i-1];
                }
            }
            for(int l=1; l<=n-x; l++){/*l represents the length of the submatrix, which is its x-axis projection*/
                for(int h=1; h<=n-y; h++){/*h represents the height of the submatrix, which is its y-axis rojecting*/
                    thisSum = sum[y+h][x+l]+sum[y+h-1][x+l]+sum[y+h][x+l-1]-sum[y+h-1][x+l-1];
                    sum[y+h][x+l] = thisSum;
                    if(thisSum>MaxSum){/*record the temporary submatrix with maximum sum*/
                        MaxSum = thisSum;
                        solution->h = h;
                        solution->l = l;
                        solution->x = x;
                        solution->y = y;
                    }
                }
            }
        }
    }
    solution->maxsum = MaxSum;
}

void algorithm2(Ptr2Matrix matrix, Ptr2SubMatrix solution){
    int thisSum = 0;
    int MaxSum = 0;
    int n = matrix->size;
    int *a = matrix->Array;
    int y = 0;
    int sum[n][n];/*record the sum of subsequence of a row*/
    for(int x=0; x<n; x++){
        for(int i=0; i<n; i++){/*initialize the sum array*/
            for(int j=0; j<n; j++){
                sum[i][j]=a[i*n+j];
            }
        }
        for(int j=0; j<n; j++){
            for(int i=x+1; i<n; i++){
                sum[j][i] += sum[j][i-1];
            }
        }
        for(int i=0; i<n; i++){
            thisSum = 0;
            y = 0;
            for(int j=0; j<n; j++){
                thisSum += sum[j][i];
                if(thisSum<0){
                    thisSum = 0;
                    y = j+1;
                }
                if(thisSum>MaxSum){
                    MaxSum = thisSum;
                    solution->x = x;
                    solution->y = y;
                    solution->h = j-y+1;
                    solution->l = i-x+1;
                }
            }
        }
    }
    solution->maxsum = MaxSum;
}
```

- test.h

```
#ifndef TEST
#define TEST
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```c

    #ifndef SUBMATRIX
    #define SUBMATRIX
    /*determine a submatrix exactly*/
    struct subMatrix{
        /*the coordinate of the top-left corner of the submatrix*/
        int x;
        int y;
        /*the length of the submatrix, which is its x-axis projection*/
        int l;
        /*the height of the submatrix, which is its y-axis rojecting*/
        int h;
        /*the maximum sum*/
        int maxsum;
    };
    /*pointer to the submatrix*/
    typedef struct subMatrix *Ptr2SubMatrix;
    #endif

    #ifndef MATRIX
    #define MATRIX
    /*represent the original matrix*/
    struct Matrix{
        /*the size of the original matrix, for instance, a n*n matrix is of size n*/
        int size;
        /*store the elements which is in the matrix*/
        int *Array;
    };
    /*pointer to the original matrix*/
    typedef struct Matrix *Ptr2Matrix;
    #endif

    #ifndef SOLVER
    #define SOLVER
    /*pointer to the solver functions*/
    /*return the maximum sum*/
    typedef void (*solver)(Ptr2Matrix matrix, Ptr2SubMatrix solution);
    #endif

    #ifndef RUNTIME
    #define RUNTIME
    /*record the performances of the functions*/
    struct runTime{
        /*the number of execution for the function tested*/
        int iterations;
        /*the number of all ticks in the function run time*/
        clock_t ticks;
        /*the total run time*/
        double totalTime;
        /*the run time for executing the function tested once*/
        double duration;
    };
    /*pointer to the record of the performances of the functions*/
    typedef struct runTime *Ptr2RunTime;
    #endif

/*****************************************************************************
*   tester: check the solution, record the run time, print the result      *
*   parameter:  matrix  ->  the original matrix                              *
*               solution->  record the submatrix with the maximum sum        *
*               solvers ->  an array whose element points to three different  *
*                           functions implementing three different algorithms *
*               index   ->  index of the function to be tested                *
*                           index = 0 means that the function runs in O(N^6)   *
*                           index = 1 means that the function runs in O(N^4)   *
*                           index = 2 means that the function runs in O(N^3)   *
*               iterations  ->  the expected number of execution              *
*                               for the function tested                      *
*****************************************************************************/
void tester(Ptr2Matrix matrix, Ptr2SubMatrix solution, solver *solvers, int index, int iterations);

/*****************************************************************************
*   checker:    The checker checks whether the solution is right or not,      *
*               by comparing it with the solution provided by the standard    *
*               solver. The standard solver is the function runs in O(N^6)      *
*               because it implement the Brute Force algorithm.              *
*   parameter:  solution    ->  the solution to be checked by the checker     *
*               standard    ->  the standard solution                        *
*   returned value: return 0 in the case that the solution is incorrect,      *
*                       return other values while the solution is correct.    *
```

```
*******************************************************************************/
int checker(Ptr2SubMatrix solution, Ptr2SubMatrix standard);

/******************************************************************************
*   printer:    The printer output the test result. The submatrix with maximum*
*               sum and the original matrix will be printed to the file named *
*               "solution.txt" and the performance of the function tested will*
*               be printed the file named "performance.csv"                   *
*   parameter:  matrix  ->  the original matrix                               *
*               solution->  the solution provided by the function tested      *
*               performance ->  the performance of thw function tested         *
*               index   -> index of the function to be tested                 *
*               iterations  ->  the expected number of execution               *
*                               for the function tested                       *
*   returned value: return 0 in the case that printing failed,                 *
*                   return other values while the printer successfully print. *
*******************************************************************************/
int printer(Ptr2Matrix matrix, Ptr2SubMatrix solution, Ptr2RunTime performance, int index);

/******************************************************************************
*   stopwatch:  record the total ticks in the function tested run time.       *
*   parameter:  matrix  ->  the original matrix                               *
*               solution->  A piece of memory used for storing the solution   *
*               solverf ->  the function to be tested                         *
*   returned value: the performence in the function tested run time.           *
*******************************************************************************/
Ptr2RunTime stopwatch(Ptr2Matrix matrix, Ptr2SubMatrix solution, solver solverf, int iterations);
#endif
```

- test.c

```c
#include "test.h"

void tester(Ptr2Matrix matrix, Ptr2SubMatrix solution, solver *solvers, int index, int iterations){
    solver solverf = solvers[index];
    Ptr2RunTime performance = stopwatch(matrix,solution,solverf,iterations);

    Ptr2SubMatrix standard = (Ptr2SubMatrix)malloc(sizeof(struct subMatrix));
    solvers[1](matrix,standard);
    int correctness = checker(solution,standard);
    if(!correctness) {
        printf("The solution is wrong!\n");
        return;
    }

    int print = printer(matrix,solution,performance,index);
    if(!print)  printf("Failed to print the result!\n");
}

Ptr2RunTime stopwatch(Ptr2Matrix matrix, Ptr2SubMatrix solution, solver solverf, int iterations){
    clock_t start,stop;
    int i = 0;
    start = clock();
    while(i++ < iterations)
        solverf(matrix, solution);/*repeat the function calls*/
    stop = clock();
    Ptr2RunTime performance = (Ptr2RunTime)malloc(sizeof(struct runTime));
    performance->iterations = iterations;
    performance->ticks = stop-start;
    performance->totalTime = ((double)(performance->ticks))/CLK_TCK;
    performance->duration = (performance->totalTime)/iterations;
    return performance;
}

int checker(Ptr2SubMatrix solution, Ptr2SubMatrix standard){
    int correction = 1;
    if((solution->h!=standard->h)||(solution->l!=standard->l)||(solution->maxsum!=standard->maxsum)||(solution->x!=standard->x)||(solution->y!=standard->y))
        correction = 0;
    return correction;
}

```

```c
int printer(Ptr2Matrix matrix, Ptr2SubMatrix solution, Ptr2RunTime performance, int index){
    FILE *txt = fopen("Test_cases\\solution.txt", "a");/*open the "solution.txt" file*/
    if(!txt)    return 0;

    /*print the original matrix to the solution.txt file*/
    fprintf(txt, "Algorithm%d:\n", index);
    fprintf(txt, "original matrix:\tsize:%d\n",matrix->size);
    for(int j=0; j < matrix->size; j++){/*represent the ordinate of an element*/
        for(int i=0; i < matrix->size; i++){/*represent the abscissa of an element*/
            fprintf(txt, "%10d", matrix->Array[j*(matrix->size)+i]);
        }
        fprintf(txt, "\n");
    }

    /*print the solution submatrix to the solution.txt file*/
    fprintf(txt, "solution:%d\nx:%d\ty:%d\tl:%d\th:%d\n",solution->maxsum,solution->x,solution->y,solution->l,solution->h);
    for(int j= solution->y; j< solution->y+solution->h; j++){
        for(int i= solution->x; i< solution->x+solution->l; i++){
            fprintf(txt, "%10d", matrix->Array[j*(matrix->size)+i]);
        }
        fprintf(txt, "\n");
    }
    fclose(txt);

    /*print the performance of the function tested to the performance.csv file*/
    FILE *csv = fopen("Test_cases\\performance.csv", "a");
    if(!csv)    return 0;
    fprintf(csv, "Algorithm%d,%d,%d,%ld,%f,%f\n", index, matrix->size, performance->iterations, performance->ticks, performance->totalTime, performance->duration);
    fclose(csv);

    return 1;
}
```

- header.c

```c
#include <stdio.h>
/*used to produce the header of the table which record the performance of the functions*/
int main(){
    FILE *fp = fopen("Test_cases\\performance.csv","a");
    fprintf(fp,"Algorithm,Size,Interations(K),Ticks,Total Time(sec),Duration(sec)\n");
    fclose(fp);
}
```

- main.c

```c
#include "test.h"
#include "solvers.h"

int main(int argc,char *argv[]){
    /*print the parametes, which determine the iterations, algorithm, and the size of matrix*/
    printf("iterations:%d\n",atoi(argv[1]));
    printf("algorithm:%d\n",atoi(argv[2]));
    printf("size:%d\n",atoi(argv[3]));

    /*create a matrix with size of argv[3]*argv[3]*/
    Ptr2Matrix matrix = (Ptr2Matrix)malloc(sizeof(struct Matrix));
    matrix->Array = (int *)malloc(sizeof(int)*atoi(argv[3])*atoi(argv[3]));
    matrix->size = atoi(argv[3]);
    srand((unsigned)time(NULL));
    for(int i=0; i<(matrix->size)*(matrix->size); i++){
        matrix->Array[i] = rand()%21-10;/*produce a random number in the interval [-10,10]*/
    }
    /*create a struct to store solution*/
    Ptr2SubMatrix solution = (Ptr2SubMatrix)malloc(sizeof(struct subMatrix));
    /*initialize the solution*/
    solution->h = 0;
    solution->l = 0;
    solution->maxsum = 0;
    solution->x = 0;
    solution->y = 0;

    /*store the pointer which point to the solver*/
    solver solvers[3];
    solvers[0] = algorithm0;
    solvers[1] = algorithm1;
    solvers[2] = algorithm2;

    int index = atoi(argv[2]);
    int iterations = atoi(argv[1]);

    /*start the test*/
    tester(matrix, solution, solvers, index, iterations);

    printf("finished the test\n");
}
```

DECLARATION

I hereby declare that all the work done in this project titled "MAXIMUM SUBMATRIX SUM PROBLEM" is of my independent effort.