MAXIMUM SUBMATRIX SUM PROBLEM

A student

2022-10-07

1. Introduction

In mathematics, matrices are an extremely important structure, and in many problems we have to make use of various properties of matrices. One of the important problems is also how to find the submatrix with the maximum sum.

Now, given a $n * n$ matrix, our goal is to design an algorithm that finds the submatrix with the maximum sum.

2. Algorithm Specification

I designed three algorithms to find the submatrix with the maximum sum, and their time complexity are $O(n^6)$ $O(n^4)$ $O(n^3)$.

Algorithm1: The $O(n^6)$ Algorithm

Input: $n * n$ matrix

Output: the submatrix with the maximum sum

Main Idea: Iterate through each submatrix, summing and comparing their sum to each other to find the submatrix with the maximum sum.

Pseudo Code:

```
int MaxMatrixSum(int size){

    for(int i1=0;i1<size;i1++){
        for(int i2=0;i2<size;i2++){

            for(int j1=i1;j1<size;j1++){
                for(int j2=i2;j2<size;j2++){

                    Summation of the current submatrix;
                    compare with the previous maximum sum;

                    if(ThisSum>MaxSum){
```

```
                    MaxSum=ThisSum;
                }
            }
        }
    }
}
```

Algorithm2: The $O(n^4)$ Algorithm

Input: $n * n$ matrix

Output: the submatrix with the maximum sum

Main Idea: Merge the submatrix into a one-dimensional array by rows.And then perform the operation of finding the subsequence with maximum sum for a one-dimensional array.

Pseudo Code:

```
int MaxMatrixSum(int size){

    for(int i1=0;i1<size;i1++){
        for(int i2=0;i2<size;i2++){

            for(int j1=i1;j1<size;j1++){
                for(int j2=i2;j2<size;j2++){

                    Merge the current elements into a one-dimensional array;
                    Get the sum of the subsequence of the current one-dimensional array;
                    compare with the previous maximum sum;

                    if(ThisSum>MaxSum){
                      MaxSum=ThisSum;
                    }
                }
            }
        }
    }
```

Algorithm3: The $O(n^3)$ Algorithm

Input: $n * n$ matrix

Output: the submatrix with the maximum sum

Main Idea: Merge the matrix in the smallest unit of row,And then perform the operation of finding the subsequence with maximum sum for a one-dimensional array.

Pseudo Code:

```
int MaxMatrixSum3(int size){

    for(int i=0;i<size;i++){

        for(int j=i;j<size;j++){
            Merge the elements from the i row to the j row;
            Perform the O(n) operation of finding the maximum sum subsequence;
       }
    }
```

3. Testing Results

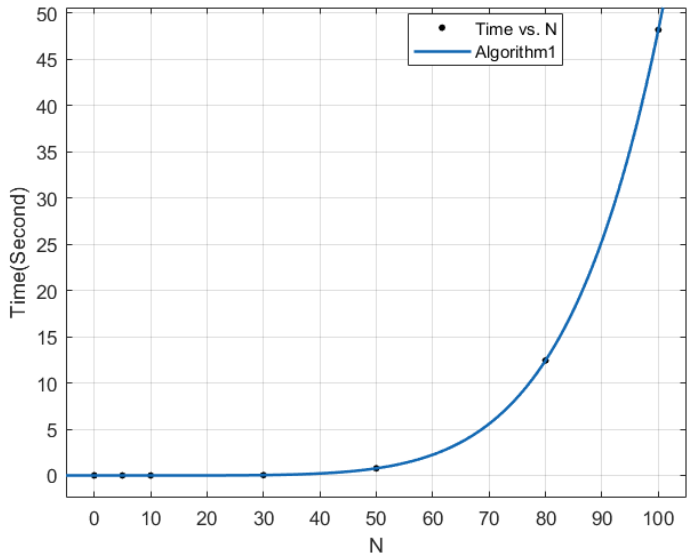
Table 1 shows some typical test cases for verifying the *Maximum Sum Submatrix* implementation and capturing potential bugs.

All the Test Cases are included in the Appendix B.

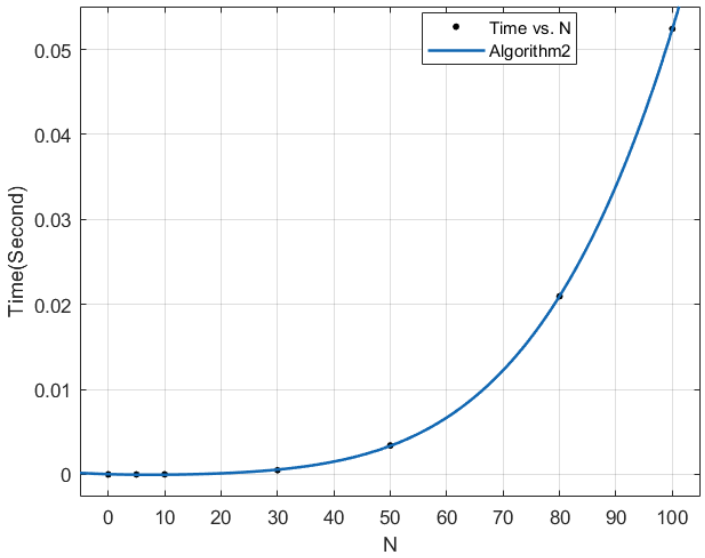
Table 1: Test cases for the implementation.

Test Cases	Design Purpose	result	status
Case 1	Minimum maxtrix with a single element	Result 1	*pass*
Case 2	maxtrix with all zero elements	Result 2	*pass*
Case 3	the result is the original matrix	Result 3	*pass*
Case 4	maxtrix in random elements	Result 4	*pass*
Case 5	matrix with repeated values	Result 5	*pass*
...	...	...	...

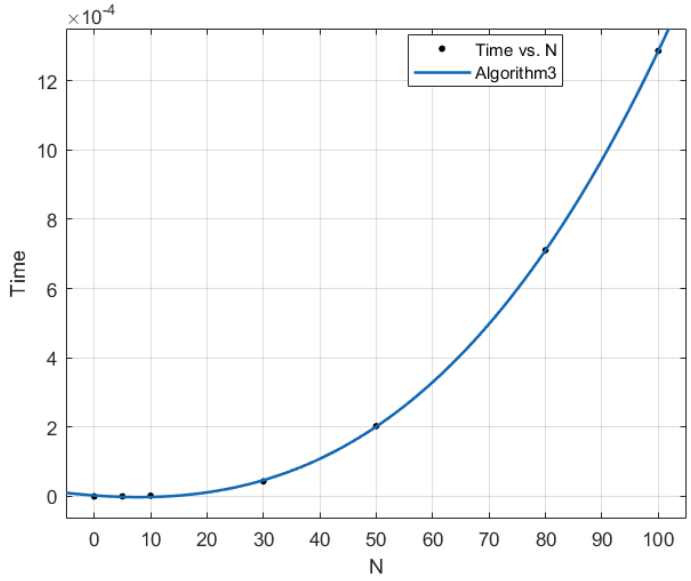
Figures 1 shows the running time of the *Algorithm1* implementation. We observe a quadratic-like curve from the figure, which implies an $O(n^6)$ algorithm (see analysis in the next section).



Figures 2 shows the running time of the *Algorithm2* implementation. We observe a quadratic-like curve from the figure, which implies an $O(n^4)$ algorithm (see analysis in the next section).



Figures 3 shows the running time of the *Algorithm3* implementation. We observe a quadratic-like curve from the figure, which implies an $O(n^3)$ algorithm (see analysis in the next section).



4. Analysis and Comments

For the runtime of *Algorithm1*, there are two subtasks inside the *for*-loop. Specifically,

- finding the sum of the current submatrix takes $O(n^2)$ time units;
- comparing ThisSum and MaxSum takes a constant time c.

This is independent of any particular input. Because the ergodic takes $O(n^4)$ time units.Therefore, both the average and worst time complexity can be computed as:

$$T(n) = O(n^6).$$

There is still considerable room for improvement. So, we design the algorithm2.

For the runtime of *Algorithm2*, there are two subtasks inside the *for*-loop. Specifically,

- merging the matrix into array and add it into the ThisSum takes a constant time c;
- comparing ThisSum and MaxSum takes a constant time c.

This is independent of any particular input. Because the ergodic takes $O(n^4)$ time units.Therefore, both the average and worst time complexity can be computed as:

$$T(n) = O(n^4).$$

There is still considerable room for improvement. So, we design the algorithm3.

For the runtime of *Algorithm3*, there are two subtasks inside the *for*-loop. Specifically,

- Performing the operation of finding the subsequence with maximum sum for a one-dimensional array takes $O(n)$ time units;
- comparing ThisSum and MaxSum takes a constant time c.

This is independent of any particular input. Because the ergodic takes $O(n^2)$ time units.Therefore, both the average and worst time complexity can be computed as:

$$T(n) = O(n^3).$$

That is the best algorithm that I can design so far.

For the space requirement, since we merely need an array to store the $n * n$ integers, the space complexity is $\Theta(n^2)$ and should be optimal.

Appendix A: Source Code (in C)

The main codes:

```c
#include<stdio.h>

int a[150][150];
//The matrix a
int MaxMatrixSum(int n);

int main(){

    int size;
    //The size of the matrix

    scanf("%d",&size);
    for(int i=0;i<size;i++){
        for(int j=0;j<size;j++){
            scanf("%d",&a[i][j]);
        }
    }
    //Store the matrix in a

    MaxMatrixSum(size);
    //Call the function to find the submatrix

    return 0;
}
```

Algorithm1:

```
int MaxMatrixSum1(int size){

    int ThisSum=0,MaxSum=a[0][0];
    //Used to record the sum of each matrix
    int x=0,y=0,m=0,n=0;
    //Used to record the max matrix

    for(int i1=0;i1<size;i1++){
        for(int i2=0;i2<size;i2++){
        //Determine the position of the first number

            for(int j1=i1;j1<size;j1++){
                for(int j2=i2;j2<size;j2++){
                //Determine the position of the last number

                    ThisSum=0;

                    for(int k1=i1;k1<=j1;k1++){
                        for(int k2=i2;k2<=j2;k2++){
                            ThisSum+=a[k1][k2];
                            //Summation
                        }
                    }

                    if(ThisSum>MaxSum){
                    //Compare sum
                        x=i1;
                        y=i2;
                        m=j1;
                        n=j2;
                        MaxSum=ThisSum;
                        //Determine the position of the maximum sun submatrix

                    }
                }
            }
        }
    }

    for(int i=x;i<=m;i++){
        for(int j=y;j<=n;j++){
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
    printf("%d\n",MaxSum);
```

```
    //Print the result
}
```

Algorithm2:

```
int MaxMatrixSum2(int size){

    int ThisSum=0,MaxSum=a[0][0];
    //Used to record the sum of each matrix

    int x=0,y=0,m=0,n=0;
    //Used to record the max matrix

    for(int i1=0;i1<size;i1++){
        for(int i2=0;i2<size;i2++){
        //Determine the position of the first number
            int temp[150]={0};
            //Design a array to hold the merged matrix

            for(int j1=i1;j1<size;j1++){
                ThisSum=0;
                for(int j2=i2;j2<size;j2++){
                //Determine the position of the last number
                    temp[j2]+=a[j1][j2];
                    //Add the current element into the array
                    ThisSum+=temp[j2];
                    //Compute the current sum
                    if(ThisSum>MaxSum){
                    //Compare sum
                        x=i1;
                        y=i2;
                        m=j1;
                        n=j2;
                        MaxSum=ThisSum;
                        //Determine the position of the maximum sun submatrix
                    }
                }
            }
        }
    }

    for(int i=x;i<=m;i++){
        for(int j=y;j<=n;j++){
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
```

```
    printf("%d\n",MaxSum);
    //Print the result
}
```

Algorithm3:

```
int MaxMatrixSum3(int size){

    int ThisSum=0,MaxSum=a[0][0];
    //Used to record the sum of each matrix

    int x=0,y=0,m=0,n=0,start=0;
    //Used to record the max matrix

    for(int i=0;i<size;i++){
    //Determine the first row

        int temp[150]={0};

        for(int j=i;j<size;j++){
        //Determine the last row

            ThisSum=0;
            start=0;

            for(int k=0;k<size;k++){
            //The Operation to find the maximum sum subsequence
                temp[k]+=a[j][k];
                ThisSum+=temp[k];
                if(ThisSum>MaxSum){
                    MaxSum=ThisSum;
                    x=i;
                    m=j;
                    n=k;
                    y=start;
                    //Determine the position of the maximum sum sub matrix
                }
                else if(ThisSum<0){
                    ThisSum=0;
                    start=k+1;
                }
            }
       }
    }

    for(int i=x;i<=m;i++){
        for(int j=y;j<=n;j++){
```

```
            printf("%d ",a[i][j]);
        }
        printf("\n");
    }
    printf("%d\n",MaxSum);
    //Print the result
}
```

Appendix B: Test Case

Case1
3
Result1
3

Case2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Result2
0

Case3
1 2 1 3 6 6 5 4
8 7 7 5 7 6 9 9
4 5 8 9 5 2 1 2
1 1 1 2 4 4 5 9
1 6 1 8 4 3 4 1
4 9 2 5 9 6 3 4
6 8 8 5 8 6 3 5
8 6 9 9 8 7 8 4

Result3
1 2 1 3 6 6 5 4
8 7 7 5 7 6 9 9
4 5 8 9 5 2 1 2
1 1 1 2 4 4 5 9
1 6 1 8 4 3 4 1
4 9 2 5 9 6 3 4
6 8 8 5 8 6 3 5
8 6 9 9 8 7 8 4

Case4
5 -11 -7 -12 2 -5 4 5
-13 -1 -3 5 -18 0 -10 5
7 3 -9 -1 8 -6 -7 4
5 -19 -4 -1 2 -19 -15 1
2 9 -2 -15 -7 -16 -2 -2
5 7 -14 2 -11 0 -13 -3
-12 -14 2 -17 -3 -13 -17 0
-3 2 -17 -18 3 7 -5 -16

Result4
2 9
5 7

Case5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5

Result5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5

Declaration

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