Fundamentals of Data Structures Laboratory Project 1

MAXIMUM SUBMATRIX SUM PROBLEM

COMPLETE DATE: 2018/10/13

CONTENTS

Chapter 1

Introduction

1.1 Background

In class, we have studied several algorithms for finding the largest sum of subsequences of a one-dimensional sequence, and found that the time complexity of different algorithms is very different. Here, the problem extends to finding the largest sum of subsequences of a two-dimensional matrix. After a certain period of time thinking and summarizing, we design the algorithm. The three algorithms have time complexity of O (N6), O (N4) and O (N3).

(1) O (N6) tries all the possibilities and then compares them in turn to find the largest sum of submatrices. This is an easy way to think of, but the time complexity is the highest of all algorithms.

(2) O (N4) ingeniously borrows the on-line monitoring method in one-dimensional sequence, but in this way, only the on-line measurement method is borrowed in line loop, while the traditional method is still used in column loop.

(3) O (N3) is the bonus of our group. This algorithm borrows the idea of "on-line monitoring" from one-dimensional sequence for columns in two-dimensional sequence, and adds the rows item by item, and reduces the complexity of the algorithm to a minimum.

1.2 Algorithm requirements

In the relevant context, we know that our program needs to traverse the sum of all the sub-matrices under an n*m matrix, then compare the size of each sum, and select the largest sum. So our project has the following main objectives:

(1) Write a program with O (N6), O (N4) algorithm complexity, and can achieve the goal of finding the largest submatrix.

(2) Analyze the algorithm complexity of the two algorithms, and use the data to test the two algorithms, and then draw a certain conclusion according to the running time

of the two algorithms.

(3) consider whether bonus can further reduce the complexity of programs.

1.3 Achieve results

Our group first wrote two algorithms with time complexity O (N6) and O (N4), and analyzed the implementation process of the two algorithms. Then, after discussion of bonus in our group, we came up with an algorithm with time complexity O (N3) and programmed it.

After all the programs are written, the tester tests the efficiency of the program with a set of typical data, and finds that there is no great difference in the speed of the three algorithms when the number of runs and the size of the matrix is small, but when the amount of matrix data increases to a certain extent, and the number of runs is large. The algorithm runs with different time complexity.

Chapter 2

Algorithm Specification

2.1 Algorithm with time complexity of O (N6)

It is easy to imagine that the algorithm calculates the sum of all the submatrices of a matrix, and then selects the largest matrix. Therefore, the time complexity of O(N4) is obtained by cycling the rows and columns of a matrix once, and then for each sum of the submatrices obtained by cycling, the rows and columns also need to be cycling once, so there is O again. (N2), so the time complexity of the algorithm is O (N2 * N4) = O (N6). Here are the key functions for finding the maximum sum of the submatrices:

```
int MaxSum(int N)
{
    int i,j,row_1,row_2,column_1,column_2;
    int Max,ThisSum;
    Max=ThisSum=0;
    for(column_1=0;column_1<N;column_1++)
    for(row_1=0;row_1<N;row_1++)
    for(row_2=row_1;row_2<N;row_2++)
    for(column_2=column_1;column_2<N;column_2++)
    {
        ThisSum=0;
        for(j=column_1;j<=column_2;j++)
        for(i=row_1;i<=row_2;i++)
        ThisSum+=Array[i][j];
        if(Max<ThisSum)
        Max=ThisSum;
    }
    if(Max<=0)
    return 0;
    return Max;
}
```

2.2 Algorithm with time complexity of O (N4)

This is a clever algorithm, affected by the idea of "on-line monitoring" of one-dimensional sequence. This algorithm is easy to imagine. Our group uses column loops in the same way as before, but for rows we use the "on-line detection" method, that is, given a minimum column and a maximum column, for this. The rows between the two columns loop, first defining two variables ThisSum and Max, and assigning an initial value of 0, then starting from the first row, calculating the sum of each row, if less than 0, directly discarding the row and all previous rows, and returning ThisSum to zero, then adding each row to ThisSum one at a time, if in the calculation process When ThisSum is found to be less than zero, ThisSum is zeroed, and if it is greater than Max, the value of ThisSum is assigned to Max, and so on, at the end of all cycles, the value of Max is the submatrix and the maximum.

Because the method of column loop is invariable, the time complexity of the algorithm is O (N2). For rows, the idea of "on-line monitoring" is adopted, so the complexity is O (N), and then the algorithm complexity is O (N) when all the elements on a row are added each time. In a word, the complexity of the algorithm is O (N * N). *N2) = O (N4).

The following is the key function of this algorithm:

```
int MaxSum(int N)
{
    int column_1,column_2;
    int i,j;
    int Max,ThisSum;
    Max=ThisSum=0;
    for(column_1=0;column_1<N;column_1++)
    for(column_2=column_1;column_2<N;column_2++)
    {
        ThisSum=0;
        for(i=0;i<N;i++)
        for(j=column_1;j<=column_2;j++)
        {
            ThisSum+=Array[i][j];
            if(j==column_2)
            {
                if(ThisSum<0)
                ThisSum=0;
                else if(ThisSum>Max)
                Max=ThisSum;
            }
        }
    }
    if(Max<0)
    return 0;
    return Max;
}
```

2.3 Bonus --- An algorithm with O (N3) time complexity (column "on-line monitoring", row by row addition)

Firstly, a double-layer loop is carried out on the rows. For each loop to get the data between two rows, the data of the first row is traversed and stored with an array, and then the array is processed with the method of "on-line monitoring" to find the largest sequence. The elements of the next row are added and then the array is "monitored online" again, so that after the end of the loop, we can get the sum of the largest submatrix of the entire matrix.

The algorithm still loops the rows first, generating a time complexity of O (N2), followed by an array traversal and a one-dimensional sequence of "on-line monitoring" operations that are actually performed at the same level of the loop, resulting in a time complexity of O (N). Overall, the entire algorithm's time complexity is O (N2*N) = O (N3).

The following is the key function of this algorithm:

```
int MaxSum(int N)
{
    int row_1,row_2,i;
    Max=ThisSum=0;
    for(row_1=0;row_1<N;row_1++)
    {
        for(i=0;i<N;i++)
        Array_2[i]=0;
        for(row_2=row_1;row_2<N;row_2++)
        {
            for(i=0;i<N;i++)
            Array_2[i]+=Array_1[row_2][i];
            ThisSum=0;
            for(i=0;i<N;i++)
            {
                ThisSum+=Array_2[i];
                if(ThisSum<0)
                ThisSum=0;
                else
                if(Max<ThisSum)
                Max=ThisSum;
            }
        }
    }
    if(Max<=0)
    return 0;
    return Max;
}
```

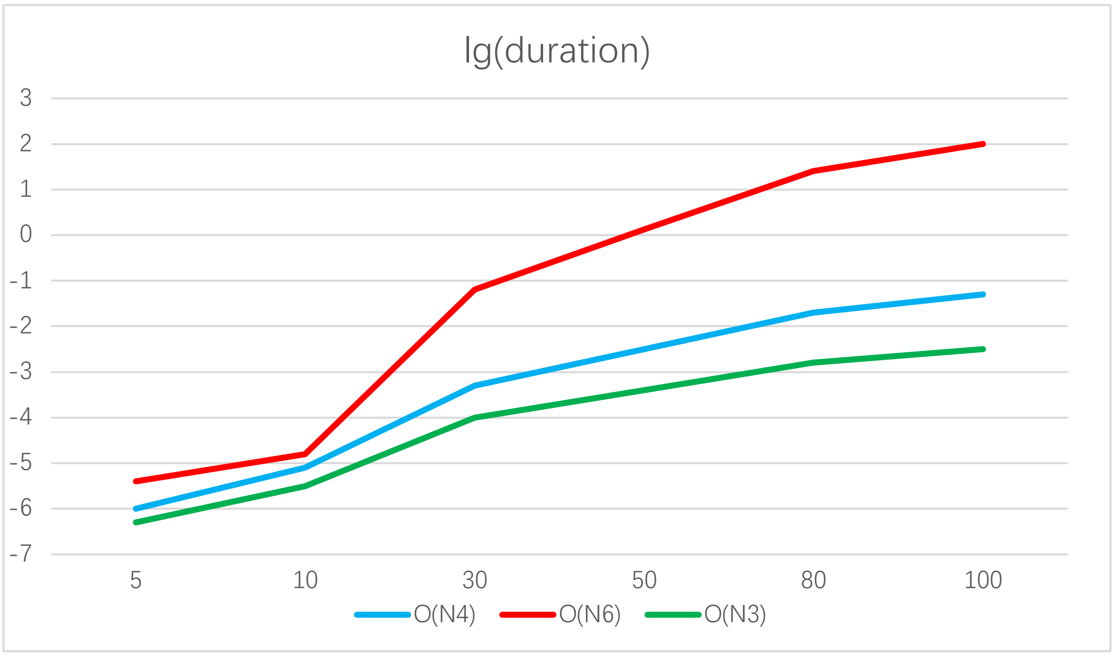
Chapter 3

Testing Results

The three procedure test results are as follows:

	N	5	10	30	50	80	100
O(N^6)	Iteration (K)	50000	10000	100	1	1	1
	Ticks	186578	1256371	6268499	1314076	22550945	87478054
	Total Time (sec)	0.186578	1.256371	6.268499	1.314076	22.550945	87.478054
	Duration (sec)	0.000004	0.000013	0.062685	1.314076	22.550945	87.478054
O(N^4)	Iteration (K)	200000	100000	1000	100	100	100
	Ticks	164152	800701	453289	322799	2036773	4899705
	Total Time (sec)	0.164152	0.800701	0.453289	0.322799	2.036773	4.899705
	Duration (sec)	0.000001	0.000008	0.000453	0.003228	0.020368	0.048997
O(N^3)	Iteration (K)	1000000	100000	10000	1000	1000	1000
	Ticks	450518	310396	850459	393676	1562859	3032053
	Total Time (sec)	0.450518	0.310396	0.850459	0.393676	1.562859	3.032053
	Duration (sec)	0.0000005	0.000003	0.000085	0.000394	0.001562	0.003032

A graph for comparing time complexities:



Chapter 4

Analysis and Comments

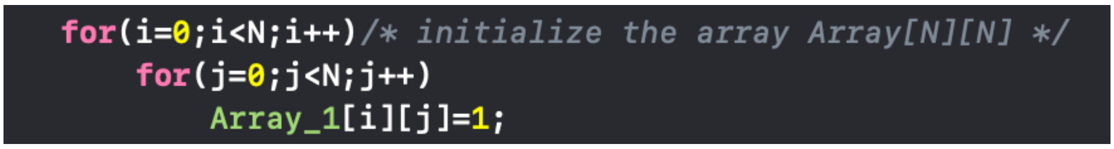
Analysis and evaluation:

The test matrix is all positive, and the time taken by the algorithm is the longest of all possible cases. After testing, it can be found that the time required for N6 and N4 is power function with N, but N6 grows faster than N4, and it takes a long time to run to the later larger matrices. It can be seen that the importance of algorithm optimization is self-evident.

IDE: Xcode v10.0 in macOS Mojave

Test code:

Initialize an array consisting N*N with all one. Unlike using the random matrix, only in this way can we detect the correctness of the test results. For example, the maximum sum of this kind of matrix is N*N.



```
for(i=0;i<N;i++)/* initialize the array Array[N][N] */
    for(j=0;j<N;j++)
        Array_1[i][j]=1;
```

O(N6) part:

The function MaxSum () which calculates the sum of the largest submatrix is N6 in complexity. First inputs the matrix order N and the number of cycles K, then inputs the element values for the array, then executes the MaxSum () function, at the same time calls the clock () function to record the time point start and stop before and after, and uses start-stop to represent the duration, and finally outputs the maximum sum and the program running time.

The time complexity is O(N6), because there are 6 superposition cycles. The given matrix costs O(N2) space complexity. In the function, the extra space complexity is O(1), for every variable varys again and again, in other words, updates when it comes to a new duration. Therefore, the overall space complexity is O(N2).

As for further possible improvements, the time complexity can be smaller. There are too many variables. It is likely that we can make them fewer. So we have thought of the N4 algorithm.

O(N4) part:

MaxSum () for calculating the sum of the largest submatrix is N4. Unlike N6, three integer variables ThisSum, column_1, column_2 are defined. There is a quadruple loop inside the function. The method is to find the sum of the largest submatrix between column_1 and column_2, and to iterate through all columns. The best combination is found among them. First inputs the matrix order N and the number of cycles K, then inputs the element values for the array, then executes the MaxSum () function, at the same time calls the clock () function to record the time point start and stop before and after, and uses start-stop to represent the duration, and finally outputs the maximum sum and the program running time.

The time complexity is O(N4), because there are 4 superposition cycles. The given matrix costs O(N2) space complexity. In the function, the extra space complexity is O(1), for every variable varys again and again, in other words, updates whenever it comes to a new duration. Therefore, the overall space complexity is O(N2).

As for further possible improvements, the time complexity can be smaller. And I don’t think the space complexity can be less. So we have thought of the bonus algorithm.

Bonus O(N3):

Under such circumstance, we define a two-dimensional array and a one-dimensional array. The function MaxSum (), which computes the sum of the largest submatrix, has a triple loop, N3 complexity, Max records the maximum sum currently read, ThisSum is responsible for continual summation, and assigns value to Max if it finds a larger sum. Note that every row_1 plus 1, all elements in array Array_2 need to be reset to 0. The main function is the same as above.

The time complexity is O(N3), because there are 3 superposition cycles. The given matrix costs O(N2) space complexity. In the function, the extra space complexity is O(N), for we define a new array, namely Array_2. Therefore, the overall space complexity is O(N2).

As for further possible improvements, the time complexity maybe come to the limit. It is likely that we can shrink the space complexity by not defining another array. So the space complexity could be a breach.

Appendix

Source Code (in C)

O（N^6）:

```
#include<stdio.h>
#include<time.h>
int MaxSum(int N);
int Array[500][500];
clock_t start,stop;
double duration;
int main()
{
    int N;
    int i,j,Max;
    int K,k;
    printf("Input a number N, using the N row N column of the matrix:\n");
    scanf("%d",&N);/* using N rows and N columns of array Array */
    printf("Input a number K, K is the number of cycles of functions:\n");
    scanf("%d",&K);/* number of iterations */
    printf("Matrix initialization:\n");
    for(i=0;i<N;i++)/* initialize the array Array[N][N] */
    for(j=0;j<N;j++)
    scanf("%d",&Array[i][j]);
    start=clock();
    for(k=0;k<K;k++)/* iterative for K */
    Max=MaxSum(N);
    stop=clock();
    printf("the maximum submatrix has the sum of %d\n",Max);
    duration=((double)(stop-start))/CLK_TCK;
    printf("The program runs %d times and consumes %lf
seconds.\n",K,duration);
```

```
    printf("The program runs %d times and consumes %lf
milliseconds.\n",K,stop-start);
    return 0;
}
int MaxSum(int N)
{
    int i,j,row_1,row_2,column_1,column_2;/* i, row_1, row_2 are row
parameters; j, column_1, column_2 are column parameters */
    int Max,ThisSum;
    Max=ThisSum=0;/* initialize the Max and Thissum */
    for(column_1=0;column_1<N;column_1++)/* column_1 starts from the first
column, when row_1>=N, column_1 plus one until colunm_1>=N */
    for(row_1=0;row_1<N;row_1++)/* row_1 starts from the first row, when
row_2>=N, row_1 adds one until row_1>=N */
    for(row_2=row_1;row_2<N;row_2++)/*    row_2 starts from row_1, until
row_2>=N */
    for(column_2=column_1;column_2<N;column_2++)/* column_2 starts from
column_1, until column_2>=N */
    {
        ThisSum=0;/* every time column_2 changes, ThisSum returns to zero */
        for(j=column_1;j<=column_2;j++)/* j starts from column_1, until
j>column_2 */
        for(i=row_1;i<=row_2;i++)/* i starts from row_1, until i>row_2 */
        ThisSum+=Array[i][j];/* sum from Array[row_1][colunm_1] to
Array[row_2][colunm_2] */
        if(Max<ThisSum)
         Max=ThisSum;/* update Max */
    }
    if(Max<=0)/* if Max<=0, return 0, otherwise return Max */
    return 0;
    return Max;
}
```

O(N^4) :

```
#include<stdio.h>
#include<time.h>
int MaxSum(int N);
int Array[500][500];
clock_t start,stop;
double duration;
int main()
{
    int N;
    int i,j,Max;
    int K,k;
    printf("Input a number N, using the N row N column of the matrix:\n");
    scanf("%d",&N);/* using N rows and N columns of array Array */
    printf("Input a number K, K is the number of cycles of functions:\n");
    scanf("%d",&K);/* number of iterations */
    printf("Matrix initialization:\n");
    for(i=0;i<N;i++)/* initialize the array Array[N][N] */
    for(j=0;j<N;j++)
    scanf("%d",&Array[i][j]);
    start=clock();
    for(k=0;k<K;k++)/* iterative for K */
    Max=MaxSum(N);
    stop=clock();
    printf("the maximum submatrix has the sum of %d\n",Max);
    duration=((double)(stop-start))/CLK_TCK;
    printf("The program runs %d times and consumes %lf
seconds.\n",K,duration);
    printf("The program runs %d times and consumes %lf
milliseconds.\n",K,stop-start);
    return 0;
}
```

```
int MaxSum(int N)
{
    int column_1,column_2;
    int i,j;/* i is row parameter; j, column_1, column_2 are column parameters
*/
    int Max,ThisSum;
    Max=ThisSum=0;/* initialize the Max and Thissum */
    for(column_1=0;column_1<N;column_1++)/* column_1 starts from the first
column, when column_2>=N, column_1 plus one until colunm_1>=N */
    for(column_2=column_1;column_2<N;column_2++)
    {
        ThisSum=0;/* every time column_2 changes, ThisSum returns to zero */
        for(i=0;i<N;i++)/* find the largest submatrix between column_1 and
column_2 */
        for(j=column_1;j<=column_2;j++)
        {
            ThisSum+=Array[i][j];
            if(j==column_2)
            {
                if(ThisSum<0)
                ThisSum=0;
                else if(ThisSum>Max)
                Max=ThisSum;
            }
        }
    }
    if(Max<0)/* if Max<=0, return 0, otherwise return Max */
    return 0;
    return Max;
}
```

Bonus———O(N^3) :

```
#include<stdio.h>
int MaxSum(int N);
int Array_1[1000][1000];
int Array_2[1000];
int main()
{
    int i,j,N;
    int Max;
    scanf("%d",&N);/* using N rows and N columns of array Array_1, using N
columns of array Array_2 */
    for(i=0;i<N;i++)
    for(j=0;j<N;j++)
    scanf("%d",&Array_1[i][j]);/* initialize the array Array_1[N][N] */
    Max=MaxSum(N);
    printf("the maximum submatrix has the sum of %d\n",Max);
}
int MaxSum(int N)
{
    int row_1,row_2,i;/* row_1, row_2 are row parameters of array Array_1, i is
column parameter of array Array_2 */
    int Max,ThisSum;
    Max=ThisSum=0;/* initialize the Max and Thissum */
    for(row_1=0;row_1<N;row_1++)
    {
        for(i=0;i<N;i++)/* every time row_1 plus one, array Array_2 returns to
zero */
        Array_2[i]=0;
        for(row_2=row_1;row_2<N;row_2++)
        {
            for(i=0;i<N;i++)/* enumerate all subsequences of each column */
            Array_2[i]+=Array_1[row_2][i];
            ThisSum=0;
            for(i=0;i<N;i++)/* find the largest subsequence of array Array_2 */
            {
                ThisSum+=Array_2[i];
                if(ThisSum<0)
                ThisSum=0;
                else
                if(Max<ThisSum)
                Max=ThisSum;
            }
        }
    }
    if(Max<=0)/* if Max<=0, return 0, otherwise return Max */
    return 0;
    return Max;
}
```

Declaration

We hereby declare that all the work done in this project titled " MAXIMUM SUBMATRIX SUM PROBLEM " is of our independent effort as a group.

Duty Assignments:

Programmer: Guo Han
Tester: Wang Ruopeng
Report Writer: Mu Qingfeng