Performance Measurement (MSS)

Date:2022-10-09

Content

Chapter 1: Introduction

Several methods for solving the *Maximum Subsequence Sum* problem are discussed in our text book.This Algorithm extend this problem to 2-dimensional.

The problem is: Given an $N*N$ integer matrix $(aij)_{N*N}$, find the maximum value of $\sum_{m}^{k=i}$ $\sum_{n}^{l=i=j}$ a_{kl} for all 1 ≤ i ≤ m ≤ N and 1 ≤ j ≤ n ≤ N. The maximum submatrix sum is 0 if all the integers are negative.

Task is:

- using O(N^6) and O(N^4) versions of algorithm to finding the maximum submatrix sum.
- Analyze the time and space complexities of the above two versions of algorithms.
- Measure and compare the performances of the above two functions for *N* = 5, 10, 30, 50, 80, 100.
- **Bonus**: Give a better algorithm. Analyze and prove that your algorithm is indeed better than the above two simple algorithms.

Chapter 2: Algorithm Specification

2.1 The main function

The main program records the **running time** of the program and **calls three functions**.

First, determine the size of the array as 5,10,30,50,80100 according to the requirements of the topic, and then determine the number of runs required by the three programs, so as to ensure the **accuracy** and **efficiency** of the program.
I will output the results of the three functions, recording **the array size, ticks, total time spent and average time** spent when they run.
The running times of the first two functions are the same. When running the third function, you need to change the running times to ensure the accuracy of the program. Otherwise, the running time will be too short, leading to inaccurate measurement.

2.2 The data structure

```
int M[105][105],sum[105][105];
```

Set the maximum number of arrays to be no more than 105 * 105, and use sum arrays to record **prefix sum**.

2.3 The O(N^6) version

First, set a time seed, traverse the N * N array, and **randomly** assign values to each element in the range of [- 20,20].
Then enumerate the sub matrix. First enumerate the rows and columns of the first element of the sub matrix, and then enumerate the rows and columns of the last element.
For the obtained submatrix, each element of it is traversed for summation, and the submatrix with the largest sum is recorded.

```
for i=0:N every line
    for j=0:N every column
        for p=i:N every line
            for q=j:N every column
                sum the matrix in [i][j][p][q];//use 2 dimension
            end
        end
    end
end
```

We can see it more clearly through the following diagram.

(i,j)				
				(p,q)

2.4 The O(N^4) version

Compared with the previous algorithm, this algorithm **optimizes the steps of calculating the sum of sub matrices**.

After randomizing each array element, we calculate the sum of all the elements in front of it and store it in a sum array.

```
sum[i][j]=M[i][j]+sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1];
```

			Sum[i-1,j-1]	Sum[i-1,j]
			Sum[i,j-1]	Sum[i,j]

When calculating the sum of sub matrices, we can directly use **the elements of sum array** to calculate the value of the sub matrix without traversing the entire sub matrix.

```
//Summing Subarrays with Prefix Sums
mat=sum[i][j]-sum[p-1][j]-sum[i][q-1]+sum[p-1][q-1];
```

		Sum[i,j]		
				Sum[p,q]

2.5 Bonus——The O(N^3) version

This algorithm does not need to enumerate the head and tail elements of the submatrix.

Let's enumerate the row from **which the submatrix starts**. For the submatrix starting from row i, we enumerate **the number of possible rows** [1, N-i+1]. In this way, we can determine which line the submatrix starts and ends.

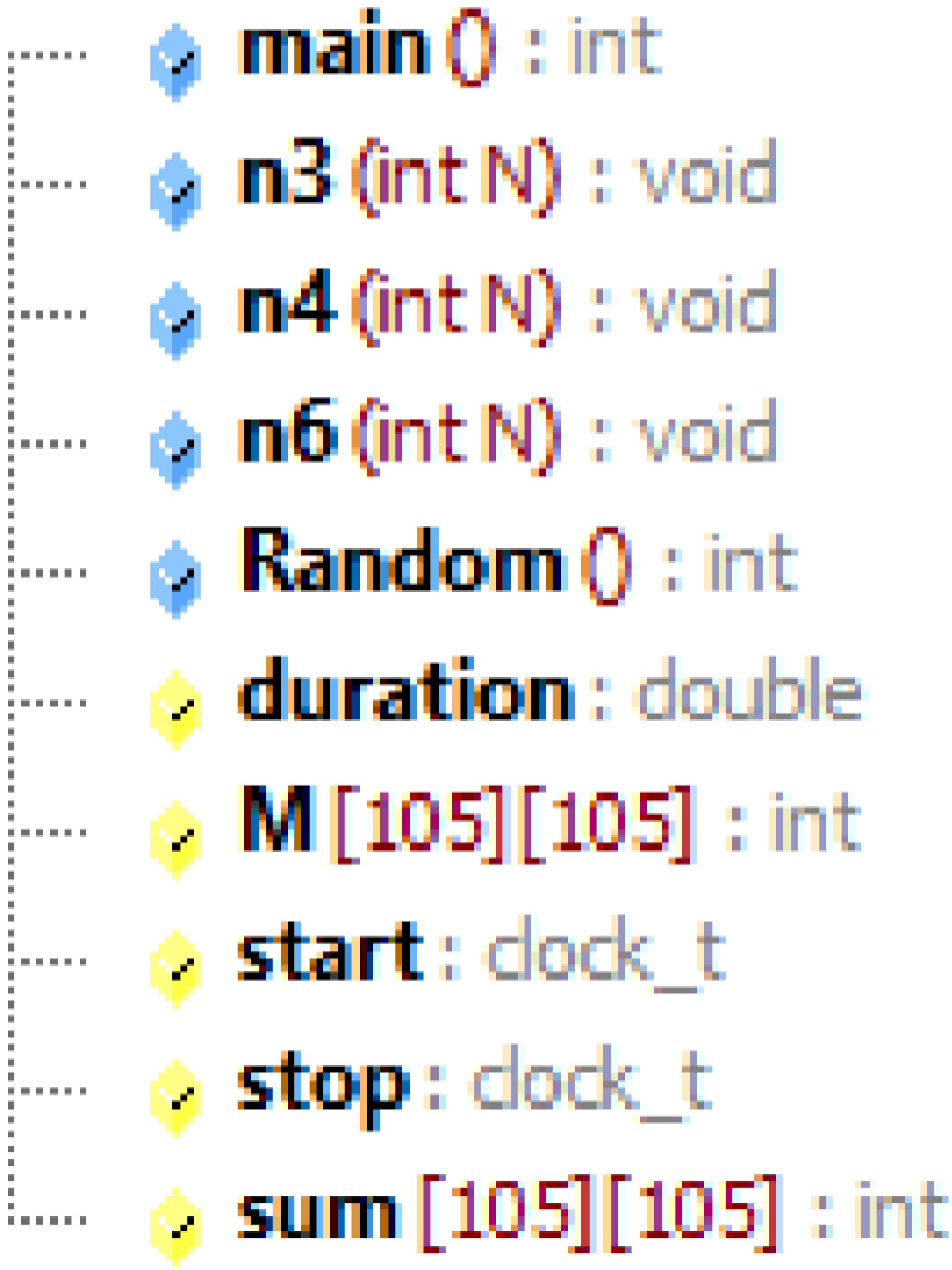
For each case, we start to traverse from the first column. Through the prefix and the sum of any column we know from this column to the previous column, we find **the largest one**, that is, we find a number of columns to minimize the sum of the first column added to this column. This number is the first column of the sub matrix we are looking for.

After finding the submatrix, we have calculated its total by prefix sum, and use a variable max to record this number.

The core steps are as follows.

```
/*initialize variable, The temp array stores the sum added to each column of the sub array,and thisnum stores the maximum value added from the beginning to the current position*/
let int temp[N],thisnum=0;
//Traverse each column of the subarray
for(every column in the matrix) {
    Calculate the value added to each column into temp[p];
//  Calculate the value added by the previous array up to this point
    thisnum<=temp[p]+thisnum;
//  Update the value of max
    if(thisnum>max)max=>thisnum;
//  Ensure that thisnum is not negative
    if(thisnum<0)thisnum=>0;
}
```

2.6 structure



Chapter 3: Testing Results

3.1 Result

```
n6 condition
N       K       TICKS   TOTAL_TIME      DURATION
5       10000   47      0.047000        0.000005
10      1000    132     0.132000        0.000132
30      50      2773    2.773000        0.055460
50      10      9157    9.157000        0.915700
80      1       12005   12.005000       12.005000
100     1       43798   43.798000       43.798000
n4 condition
N       K       TICKS   TOTAL_TIME      DURATION
5       10000   24      0.024000        0.000002
10      1000    16      0.016000        0.000016
30      50      64      0.064000        0.001280
50      10      80      0.080000        0.008000
80      1       48      0.048000        0.048000
100     1       120     0.120000        0.120000
n3 condition
N       K       TICKS   TOTAL_TIME      DURATION
5       10000   8       0.008000        0.000001
10      1000    8       0.008000        0.000008
30      1000    125     0.125000        0.000125
50      1000    612     0.612000        0.000612
80      500     1140    1.140000        0.002280
100     200     867     0.867000        0.004335

--------------------------------
Process exited after 71.05 seconds with return value 0
请按任意键继续. . .
```

3.2 Chart

3.2.1 The O(N^6) version

N	5	10	30	50	80	100
Iterations(K)	10000	1000	50	10	1	1
Ticks	47	132	2773	9157	12005	43798
Total Time(sec)	0.047	0.132	2.773	9.157	12.005	43.798
Duration(sec)	0.000005	0.000132	0.05546	0.9157	12.005	43.798

3.2.2 The O(N^4) version

N	5	10	30	50	80	100
Iterations(K)	10000	1000	50	10	1	1
Ticks	24	16	64	80	48	120
Total Time(sec)	0.024	0.016	0.064	0.08	0.048	0.12
Duration(sec)	0.000002	0.000016	0.00128	0.008	0.048	0.12

3.2.3 The O(N^3) version

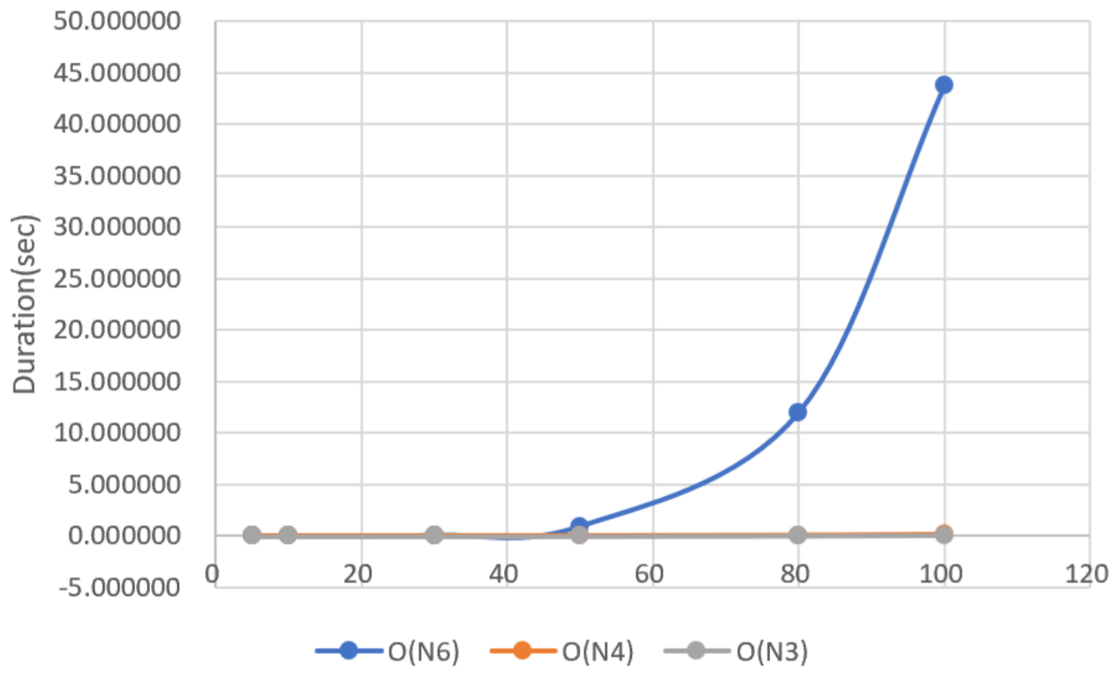
N	5	10	30	50	80	100
Iterations(K)	10000	1000	1000	1000	500	200
Ticks	8	8	125	612	1140	867
Total Time(sec)	0.008	0.008	0.125	0.612	1.14	0.867

N	5	10	30	50	80	100
Duration(sec)	0.000001	0.000008	0.000125	0.000612	0.00228	0.004335

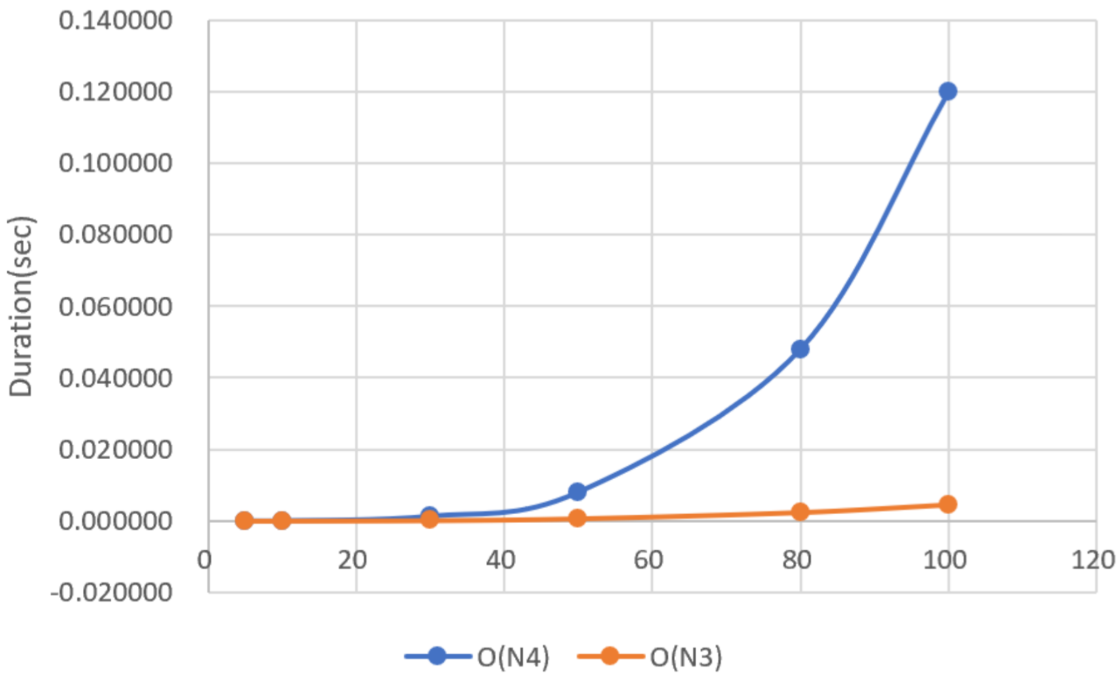
Chapter 4: Analysis and Comments

4.1 Chart Analysis

4.1.1 The $O(N^6)$ $O(N^4)$ $O(N^3)$ version Comparison Chart



4.1.2 The $O(N^4)$ $O(N^3)$ version Comparison Chart



4.2 Analysis of complexity

4.2.1 The O(N^6) version

We have traversed six times, the first two times to determine the number of rows and columns of **the head element,** the third and fourth times to determine the number of rows and columns of **the tail element**, and the last two times to traverse each row and column of the sub matrix to **sum**.

Because two-dimensional arrays are used, the space complexity is O(N^2).

4.2.2 The O(N^4) version

We have traversed for four times. The first two times determine the number of rows and columns of **the head element**, and the third and fourth times determine the number of rows and columns of **the tail element**. The sum of the submatrix is **directly obtained** through the two-dimensional prefix sum.

Because two-dimensional arrays are used, the space complexity is O(N^2).

4.2.3 The O(N^3) version

We have traversed the matrix for three times, the first time to determine **which line the submatrix starts from**, the second time to determine **the number of rows of the submatrix**, the third time to traverse each column of the matrix to **find the largest submatrix**, and the sum of the largest submatrix is obtained through the two-dimensional prefix sum.

Because two-dimensional arrays are used, the space complexity is O(N^2).

Appendix: Source Code

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

//Set the array size in advance to prevent out of bounds
int M[105][105],sum[105][105];

//Functions with operation complexity of O (n6)
void n6(int N);
//Functions with operation complexity of O (n4)
void n4(int N);
//Functions with operation complexity of O (n3)
void n3(int N);

//Randomly give an integer from - 20 to 20
int Random() {
    int a=rand()%41-20;
//   printf("%d\n",a);
    return a;
}


```

```
clock_t start, stop; /* clock_t is a built-in type for processor time (ticks)*/
double duration;/*records the run time (seconds) of a function*/
int main () {
    /*clock() returns the amount of processor time (ticks) that has elapsed
    since the program began running */

//  Set the size of N and the number of times the function runs
    int a[]= {5,10,30,50,80,100},i;
    int k[]= {10000,1000,50,10,1,1};

//  output
    printf("n6 condition\n");
    printf("N\tK\tTICKS\tTOTAL_TIME\tDURATION\n");
//  Run six different array sizes
    for(i=0; i<6; i++) {
        start = clock(); /* records the ticks at the becinnina of the function call*/
//      Run the function for many times and take the average value
        int j;
        for(j=0; j<k[i]; j++)
            n6(a[i]);/*run your function here */
        stop = clock();/*records the ticks at the end of the function call*/
//      take the average value
        duration = ((double)(stop - start))/CLK_TCK;
        double single_duration=duration/k[i];
//      output the value
        printf("%-6d\t%-d\t%-6d\t%6lf\t%6lf\n",a[i],k[i],(stop - start),duration,single_duration);
    }

    printf("n4 condition\n");
    printf("N\tK\tTICKS\tTOTAL_TIME\tDURATION\n");
//  Run six different array sizes
    for(i=0; i<6; i++) {
        start = clock(); /* records the ticks at the becinnina of the function call*/
        int j;
        for(j=0; j<k[i]; j++)
            n4(a[i]);/*run your function here */
        stop = clock();/*records the ticks at the end of the function call*/
//      take the average value
        duration = ((double)(stop - start))/CLK_TCK;
        double single_duration=duration/k[i];
//      output the value
        printf("%-6d\t%-d\t%-6d\t%6lf\t%6lf\n",a[i],k[i],(stop - start),duration,single_duration);
    }

//  resize the run times
    k[2]=1000;
    k[3]=1000;
    k[4]=500;
```

```c
    k[5]=200;
    printf("n3 condition\n");
    printf("N\tK\tTICKS\tTOTAL_TIME\tDURATION\n");
//  Run six different array sizes
    for(i=0; i<6; i++) {
        start = clock(); /* records the ticks at the beginning of the function call*/
//      take the average value
        int j;
        for(j=0; j<k[i]; j++)
            n3(a[i]);/*run your function here */
        stop = clock();/*records the ticks at the end of the function call*/
        duration = ((double)(stop - start))/CLK_TCK;
        double single_duration=duration/k[i];
//      output the value
        printf("%-6d\t%-d\t%-6d\t%6lf\t%6lf\n",a[i],k[i],(stop - start),duration,single_duration);
    }
    /* CLK_TCK is a built-in constant = ticks per second*/
    return 0;
}

void n6(int N) {
//  initialize variable
    int i,j,p,q,m,n,max=0;

    int s;
//  Sowing time Seed
    srand((unsigned int)time(0));

//  Random out an array
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            M[i][j]=Random();

//  Traverse each line to set the end point
    for(i=0; i<N; i++) {
//      Traverse each column to set the end point
        for(j=0; j<N; j++) {
//          Traverse each line to set the start point
            for(p=0; p<=i; p++) {
//              Traverse each column to set the start point
                for(q=0; q<=j; q++) {
//                  s represents the sum of the array
                    s=0;
//                  Calculate the sum of submatrixes
                    for(m=p; m<=i; m++) {
                        for(n=q; n<=j; n++) {
                            s+=M[m][n];
                        }
                    }
//                  Update the value of max
                    if(s>max)max=s;
                }
```

```
            }
        }
    }
//  printf("%d",max);
}

void n4(int N) {
//  initialize variable
    int i,j,p,q,m,n,max=0,mat;
//  Sowing time Seed
    srand((unsigned int)time(0));

//  Randomize an array and find its prefix sum
    for(i=1; i<=N; i++) {
        for(j=1; j<=N; j++) {
            M[i][j]=Random();
//          printf("%d,",M[i][j]);
//          Prefix Sum
            sum[i][j]=M[i][j]+sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1];
        }
    }

//  Traverse each line to set the end point
    for(i=1; i<=N; i++) {
//      Traverse each column to set the end point
        for(j=1; j<=N; j++) {
//          Traverse each line to set the start point
            for(p=1; p<=i; p++) {
//              Traverse each column to set the start point
                for(q=1; q<=j; q++) {
//                  Summing Subarrays with Prefix Sums
                    mat=sum[i][j]-sum[p-1][j]-sum[i][q-1]+sum[p-1][q-1];
//                  Update the value of max
                    if(mat>max)max=mat;
                }
            }
        }
    }
//  printf("%d",max);
}


void n3(int N) {
//  initialize variable
    int i,j,p,q,m,n,max=0;
//  Sowing time Seed
    srand((unsigned int)time(0));

//  Randomize an array and find its prefix sum
    for(i=1; i<=N; i++) {
        for(j=1; j<=N; j++) {
            M[i][j]=Random();
//          Prefix Sum
            sum[i][j]=M[i][j]+sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1];
        }
```

```
    }

//  Traverse the number of array lines to determine which line to start from
    for(i=1; i<=N; i++) {
//      Iterate through the array, starting from line i,
//      and determine the number of remaining lines of the sub array to be calculated
        for(j=1; j<N-i+1; j++) {
//          initialize variable
//          The temp array stores the sum added to each column of the sub array,
//          and thisnum stores the maximum value added from the beginning to the current position
            int temp[N],thisnum=0;
//          Traverse each column of the subarray
            for(p=1; p<=N; p++) {
//              Calculate the value added to each column
                temp[p]=sum[i+j-1][p]-sum[i][p]+M[i][p]-sum[i+j-1][p-1]+sum[i][p-1];
//              Calculate the value added by the previous array up to this point
                thisnum+=temp[p];
//              Update the value of max
                if(thisnum>max)max=thisnum;
//              Ensure that thisnum is not negative
                if(thisnum>0)continue;
                else thisnum=0;
            }
        }
    }
//  printf("%d",max);
}
```

Declaration

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