Performance Measurement

Liu Licheng &Wei Zhangjian&Shi Neng

Date: 2015-10-01

Chapter 1: Introduction

MAXIMUM SUBMATRIX SUM PROBLEM:

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

Example: For matrix $\begin{bmatrix} 0 & -2 & -7 & 0 \\ 9 & 2 & -6 & 2 \\ -4 & 1 & -4 & 1 \\ -1 & 8 & 0 & -2 \end{bmatrix}$, ***the maximum submatrix is*** $\begin{bmatrix} 9 & 2 \\ -4 & 1 \\ -1 & 8 \end{bmatrix}$

and has the sum of 15.

To measure the performance of a function, we may use C's standard library time.h as the following:

```
#include <time.h>
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 */
        start = clock();    /* records the ticks at the beginning of the function call */
        function();     /* run your function here */
        stop = clock(); /* records the ticks at the end of the function call */
        duration = ((double)(stop - start))/CLK_TCK;
         /* CLK_TCK is a built-in constant = ticks per second */
        … …
        1;
}
```

Chapter 2: Algorithm Specification

O(N^6):

We regard a point(x1,y1) as a start point and another point (x2,y2) as a end point. Then *calculate* $\sum_{k=x1}^{x2}\sum_{l=y1}^{y2}a_{i*j}$*, find the maximum value of* $\sum_{k=x1}^{x2}\sum_{l=y1}^{y2}a_{i*j}$ *while the end point is changing. And then change start point, find the maximum value of* $\sum_{k=x1}^{x2}\sum_{l=y1}^{y2}a_{i*j}$ *eventually.*

O(N^4):

Calculate the sum of the kth column from the ith row to the jth row. Then we can get N numbers. Just like the O(n^2) algorithm of one-dimensional problem. We search all maximum subsequence sum among those N numbers.

O(N^3):

In this program, we use the variable i,j to represent the starting row and the end row, and traverse all the possible situation. I use sum[k] to represent the sum of the kth column from the ith row to the jth row. Then we can solve the problem by the algorithm of one-dimensional problem. As we can see in the program, the complexity is O(n^3)

Chapter 3: Testing Results

In my test, for each combination of N and the type of algorithm, I created a input file(*.in). And every input file is consist of 5 groups of random matrices with certain size(N*N). After running the programs, we can get several output files(*.out), which contain the ticks and total time.

But each pair of ticks and total time means the ticks and total time of the period when the computer calculate a given matrix for K times. Thus, if we want to get the ticks and total time filled in the table following, we should sum all ticks/total time in a same output file up and divide it by 5(the number of groups). Afterwards, divided the total time and you can get the duration, which means the time spent on articulating a single matrix with certain size,using a certain algorithm.

Then let's talk about how to select the proper size of K. At first, I noticed that if the program adopting O(N^4) Version run for 10 times, the TotalTime would be approximately 1 sec(0.842), which was really suitable for test. According to the time complexities of the algorithm, the formula K=(int)1600000/(N/5)^4 was created by me. Later, I replace the exponent 4 with 3 and 6. As for O(N^6) Version, I made some other changes to ensure that all Ks are no less than 1, and that the time during tests is not too long to wait.

Formula:

O(N^6) Version: K=(int)160000/(N/5)^6 , N<=30

K=1 , N>30

O(N^4) Version: K=(int)1600000/(N/5)^4

O(N^3) Version: K=(int)1600000/(N/5)^3

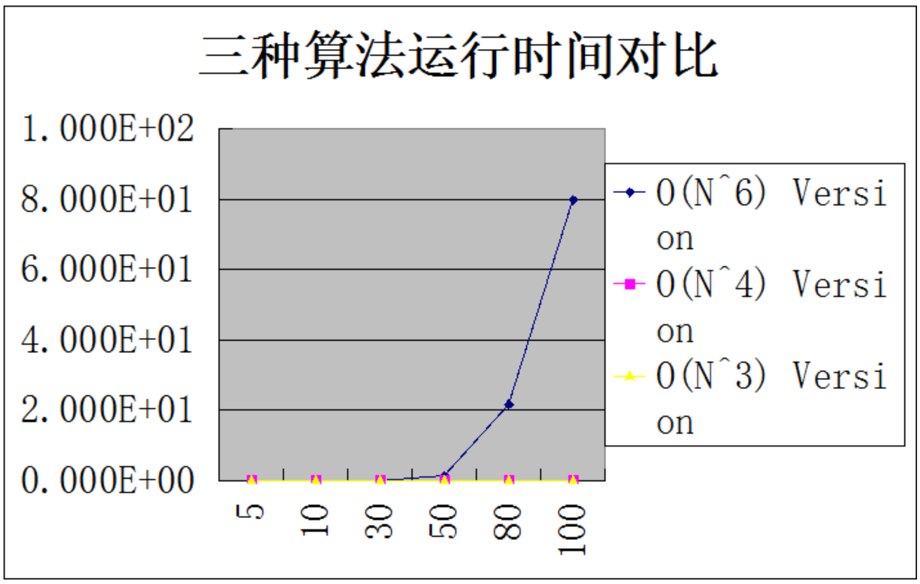
Ticks=(Ticks1+Ticks2+...+Ticks5)/5;

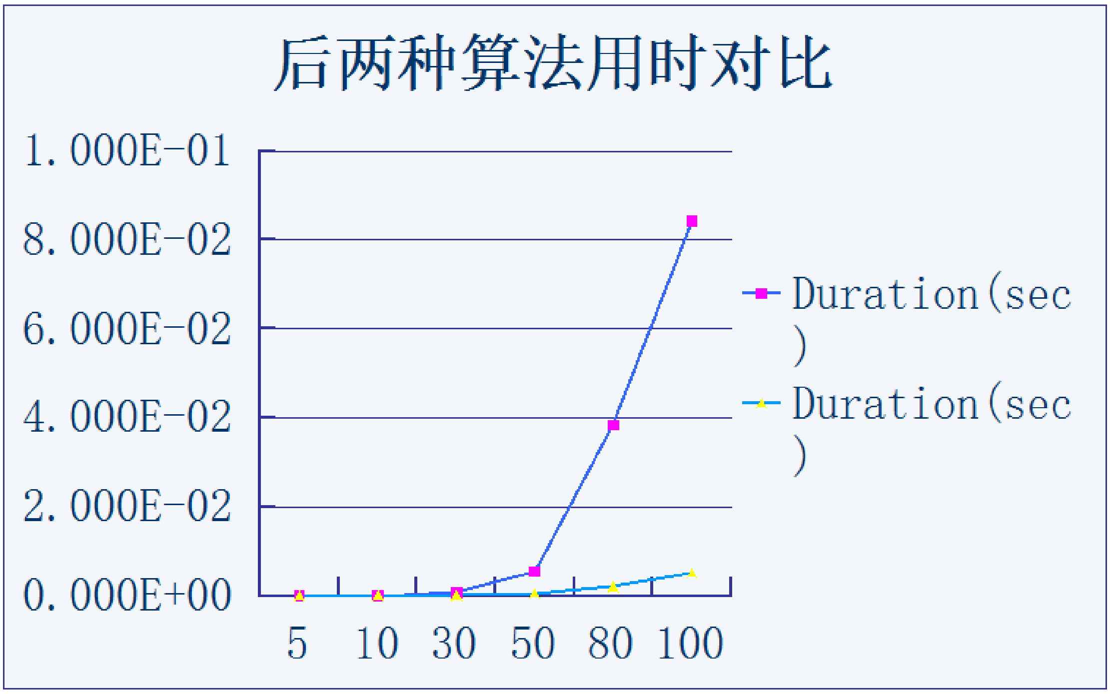
TotalTime=(TotalTime1+TotalTime2+...+TotalTime5)/5;

Duration=TotalTime/K;

Current Status:_pass_______

	N	5	10	30	50	80	100
	Iterations(K)	160000	2500	3	1	1	1
O(N^6)	Ticks	657	344	185	1309	21728	79736
Version	Total Time(sec)	0.657	0.344	0.185	1.309	21.728	79.736
	Duration(sec)	4.106E-06	1.376E-04	6.167E-02	1.309E+00	2.173E+01	7.974E+01
	Iterations(K)	1600000	100000	1234	160	24	10
O(N^4)	Ticks	1666	1033	890	870	921	842
Version	Total Time(sec)	1.666	1.033	0.89	0.87	0.921	0.842
	Duration(sec)	1.041E-06	1.033E-05	7.212E-04	5.438E-03	3.838E-02	8.420E-02
	Iterations(K)	1600000	200000	7407	1600	390	200
O(N^3)	Ticks	891	702	1012	925	829	1038
Version	Total Time(sec)	0.891	0.702	1.012	0.925	0.829	1.038
	Duration(sec)	5.569E-07	3.510E-06	1.366E-04	5.781E-04	2.126E-03	5.190E-03





Chapter 4: Analysis and Comments

```
//search all the matrix
        //(stx,sty) is the starting point
        for(testi=1;testi<=testtimes;testi++){
            max=0;
            for (stx=1;stx<=n;stx++)                    //n times
                for (sty=1;sty<=n;sty++)                //n times
                    //(enx,eny) is the end point
                    for (enx=stx;enx<=n;enx++)          //n,n-1,n-2...1 times
                        for (eny=sty;eny<=n;eny++)      //n,n-1,n-2...1 times
                {
                            //sum represents the sum of the current matrix
                            sum=0;
                            for (i=stx;i<=enx;i++)      //<=n times
                                for (j=sty;j<=eny;j++)  ///<=n times
                                {
                                    sum+=a[i][j];

                                }
                            if (sum>max) max=sum;
                        }


        }
```

Time Complexity of algorithm 1: O(n^6)

Space Complexity of algorithm 1: O(n^2) (The array a)

```
for(testi=1;testi<=testtimes;testi++){
            max=0;
            for (i=1;i<=n;i++)//n times
            {
                memset(pre,0,sizeof(pre));
                for (j=i;j<=n;j++)//n times
                {
                    //calculate the sum of the kth column   from the ith row to the jth row
                    for (k=1;k<=n;k++)//n times
                        pre[k]+=a[j][k];
                    //the following is just like the O(n^2) algorithm of one-dimensional
//problem
                    for (sty=1;sty<=n;sty++)//n times
                    {
                        sum=0;
                        for (y=sty;y<=n;y++)//<=//n times
                        {
                            sum+=pre[y];
                            if (sum>max) max=sum;
                        }
                    }
                }
            }
        }
```

Time Complexity of algorithm 2:   n*n*(n+n)*n --> O(n^4)

Space Complexity of algorithm 2: O(n^2+n)=O(n^2)   (The array a and pre)

```
for (i=1;i<=n;i++)//n times
        {
            memset(sum,0,sizeof(sum));
            for (j=i;j<=n;j++)     //n times
            {
                sum1=0;
                max=0;
                //calculate the sum of the kth column   from the ith row to the jth row
                for (k=1;k<=n;k++)//n times
                    sum[k]+=a[j][k];

                //then we can solve the problem by the algorithm of one-dimensional
problem
                for (k=1;k<=n;k++)//n times
                {
```

```
                    if (sum1>0) sum1+=sum[k];//if the previous is positive then it may be in
//the answer
                    else sum1=sum[k];//else we just drop it and restart from sum[k]
                    if (sum1>max) max=sum1;
                }
                if (max>ans)ans=max;
            }
        }

        }
```

Time Complexity of algorithm 3: n*n*(n+n) --> O(n^3)

Space Complexity of algorithm 3: O(n^2+n)=O(n^2) (The array a and sum)

Firstly, obviously, the space complexity of these 3 algorithms are really close to each other, and we can declare that memory is not the major thing we should consider of this problem.

Comparing the time complexity of algorithm with the diagrams above, I find out that as the size of N grows the time of algorithm1 rises much faster than other 2 algorithms. Even the curves of algorithm2 and algorithm3 are nearly coincident if we put 3 algorithm into the same graph.

When I separate the curve of algorithm from the graph, there's no doubt that the growth rate of algorithm2 is much more higher than algorithm3.

All in all, if the size of N is large enough, the trends of these 3 algorithms all correspond to their time complexity well. The algorithm3 performs so well that I can hardly think of possible improvements. If any, the capabilities of the array a and sum are decided numbers, I think they could be changed according to the size of n(after reading the variable n, use malloc() to distribute each array some just right memory space).

Declaration :

We hereby declare that all the work done in this project titled " Performance Measurement" is of our independent effort as a group.

Duty Assignments:

Programmer: ***Shi Neng***

Tester: ***Wei Zhangjian***

Report Writer: ***Liu Licheng***