Performance Measurement （M S S)

Date： 2021-10-2

1. Introduction

We should use the methods that we had used in *Maximum Subsequence Sum* problem to solve this question in two dimensional . That is ,given an N*N integer matrix and we will find the maximum sub-matrix in this matrix. And we should use two Algorithm , one runs in O(N^6) , while one runs in O(n^4).

2. Algorithm Specification

- O(n^6):

  In this algorithm, we use the loop traversal algorithm. That means I will calculate all the sub-matrix and get the maximum. The data structure I used is an one dimensional matrix , compared with two-dimensional matrix , one dimensional matrix is easy to understand and calculate.

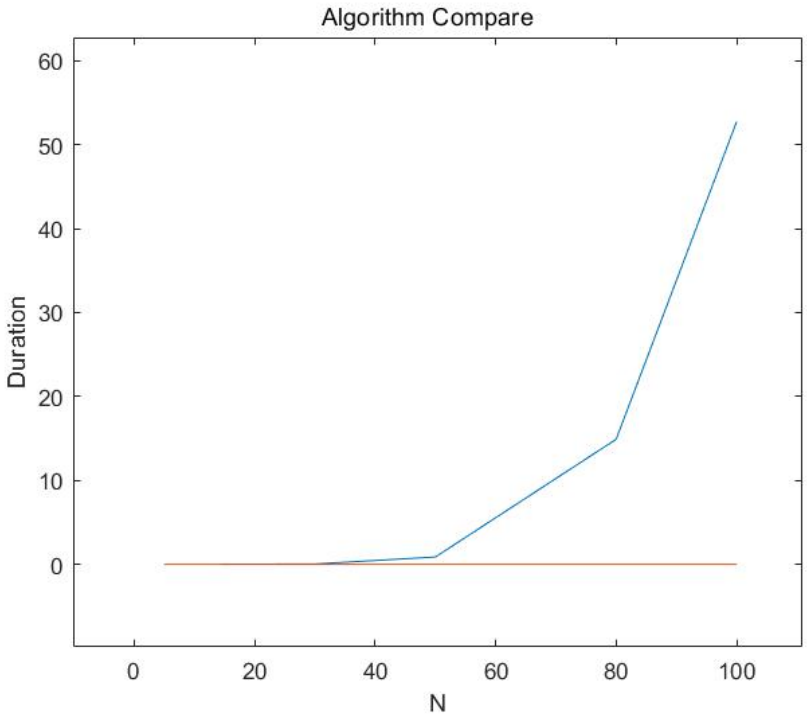
  I use 6 "for" to traverse all the sub-matrix and get the maximum.

- O(n^4):

  In this algorithm, we use the dynamic programming algorithm ， that is we need not to calculate all the sub-matrix. I traverse the i,j and calculate the a[k,i] to a[k,j] as sum b[k] , then the problem is in the textbook : the calculate the maximum of an array b[k]. And the biggest maximum of all i,j is the Maximum of the problem. The data structure I used is an one dimensional matrix as a[i,j] and an array as b[k].

3. Test results

\	N	5	10	30	50	80	100
O(n^6) Version	Iterations(K)	10^6	10^5	100	1	1	1
	Ticks	2992	1049	4522	877	14903	52748
	Total Time(sec)	2.992	1.049	4.522	0.877	14.903	52.748
	Duration(sec)	2.992*10^(-6)	1.049*10^(-5)	0.04522	0.877	14.903	52.748
O(n^4) Version	Iterations(K)	10^7	10^6	1000	1000	100	100
	Ticks	2762	192	58	275	111	192
	Total Time(sec)	2.762	0.192	0.058	0.275	0.111	0.192
	Duration(sec)	2.762*10^(-7)	1.92*10^(-7)	5.8*10^(-5)	2.75*10^(-4)	1.11*10(-3)	1.92*10^(-3)



I use 6 samples to test these two programs. the sample I used is in file "Sample" . In each test , I will set the Iterations(K) and test it by different input N , than it will output the answer 、 Ticks、 Total time and Duration, in every test ,I will make sure the ticks is bigger than 10. In my test , when N get bigger , the program may shut down , that because we input too many (when N is 100 , we should input 10000!) numbers. The question can be solved by entering the number in several times , for example , when N is 100, we input 1000 number at a time and input 10 times, in this way we input totally 10000 numbers.

4. Analysis and Comments

[Analysis]

In first program , we use 6 "for". Traverse i,j we use O((n(n+1))/2) that is the same as O(n^2) and the I traverse all the (p,q) and (m,n) in matrix that is the same as O(n^4). so the time complexities is O(n^6).

In second program , we traverse i and in every traverse , we will set b[k] to 0 , and then we traverse j and k. That means we our time complexities is O((n^2)*(n(n+1)/2)) that is same as O(n^4).

[Comments]

We can see that the O(n^4) program is much fast than O(n^6) , especially when the data is vary large. Even though the number of cycles is 10 times more than the first algorithm, the running time is faster. And the duration is much smaller than the first one. As the N increasing , the duration of the first function is rapidly increasing, but the duration of second function is really slowly increasing (We can't even tell!) . This shows that the optimization of algorithm time complexity has a great impact on the running time of the program.

*Appendix: Source Code

1. O(n^6):

```
1 ///=======================///
```

```c
/// this is O(n^6) version///
///======================///

#include <stdio.h>
#include <math.h>
#include <time.h>
#include <string.h>
clock_t start,stop;
double duration;

int MaxSum = 0;
int NowSum = 0;
int N;
int CalMat(int a[],int i,int j,int m,int n);//get the sum of (i,j) to (m,n) in a ,i<m,j<n
void Get_Max(int a[]);//get the MaxSum
int main(){
    scanf("%d",&N);
    int a[N*N];
    memset(a,0,sizeof(int)*N*N);
    for(int k=0;k<N*N;k++){
        scanf("%d",&a[k]);
    }
    ////At this point, N*N matrix input is complete, (i,j) corresponds to a[(j-1)*N+i-1]

    start = clock();
    for (int k=0;k<10000;k++){  /// Repeat Call the Func by k times
        MaxSum = 0;//Reset the MaxSum
        Get_Max(a);
    }
    printf("The Max is %d\n",MaxSum);

    stop = clock();
    duration = ((double)(stop-start))/CLK_TCK;
    printf("The ticks is %d\n",stop-start);
    printf("The total time is %f\n",duration);
    printf("The duration time is %f",duration/10000);
    return 0;
}
int CalMat(int a[],int i,int j,int m,int n){
    int sum = 0;
    for (int y=j;y<=n;y++){
        for (int x=i;x<=m;x++){
            sum = sum+a[(y-1)*N+x-1];
        }
    }
    return sum;
}
void Get_Max(int a[]){
    for (int i=1;i<=N;i++){
        for (int j=1;j<=N;j++){
            for (int m=i;m<=N;m++){
                for(int n=i;n<=N;n++){
                    NowSum = CalMat(a,i,j,m,n);
                    if (NowSum>=MaxSum){MaxSum=NowSum;NowSum=0;}
                } //We use 6 for!
            }
```

```
        }
    }
}
```

2. O(n^4):

```
///=======================///
/// this is O(n^4) version///
///=======================///

#include <stdio.h>
#include <math.h>
#include <time.h>
#include <string.h>
clock_t start,stop;
double duration;

int MaxSum = 0;
int NowSum = 0;
int N;
int CalMat(int a[],int i,int j,int m,int n);//get the sum of (i,j) to (m,n) in a,i<m,j<n
void Get_Max(int a[]);//get the MaxSum
int main(){
    scanf("%d",&N);
    int a[N*N];
    memset(a,0,sizeof(int)*N*N);
    for(int k=0;k<N*N;k++){
        scanf("%d",&a[k]);
    }
   ////At this point, N*N matrix input is complete, (i,j) corresponds to a[(j-1)*N+i-1]
    start = clock();
    for (int k=0;k<10000;k++){  /// Repeat Call the Func by k times
        MaxSum = 0;//Reset
        Get_Max(a);
    }
    printf("The Max is %d\n",MaxSum);

    stop = clock();
    duration = ((double)(stop-start))/CLK_TCK;
    printf("The ticks is %d\n",stop-start);
    printf("The total time is %f\n",duration);
    printf("The duration time is %f",duration/10000);
}

void Get_Max(int a[]){
    int b[N];
    int Max = 0;
    for(int i=1;i<=N;i++){
        memset(b,0,sizeof(int)*N);
        for (int j=i;j<=N;j++){
            int sum = 0;//for every i,j ,we calculate the sum of a[i,k] to a[j,k] and store it in b[k]
            for  (int k=1;k<=N;k++){
```

```
                b[k-1]=b[k-1]+a[(k-1)*N+j-1];
                sum = sum + b[k-1];
                if (sum<0){sum = 0;}//if the sum<0,that means the 0
is bigger
                if (sum>Max){Max=sum;}//else the Max is sum
            }
        }
    }
    if(Max>MaxSum){MaxSum = Max;}
}
```

Declaration:

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