MAXIMUM SUBMATRIX SUM PROBLEM

Author: Zarin Zuo

Date: 2021-09-23

1.Introduction

In the class we have learned several algorithms about the Maximum Subsequence Sum(MSS) Problem. Similarly, we have the Maximum Submatrix Problem if we expand it into 2-dimension.

Given a $N * N$ integer matrix $(a_{ij})_{N*N}$, find the maximum value of $\sum_{k=i}^{m} \sum_{l=j}^{n} a_{kl}$ for all $1 \leqslant i \leqslant m \leqslant N$ and $1 \leqslant j \leqslant n \leqslant N$. Additionally, if all elements are negative, we define the Maximum Submatrix Sum as 0.

On this problem, we are going to compare 2 algorithms and do analysis on their time and space complexities. Besides, we will try to find a better algorithm and prove its efficiency.

2.Algorithm Specification

a.an $O(N^6)$ Algorithm for Maximum Submatrix Sum

Input: $N * N$ integer matrix a_{ij}

Output: 1 integer indicating the Maximum Subsequence Sum

Main Idea:

1. Enumerate i, j, m, n, then adding all the elements together to get the answer

Pseudo Code:

```
ans:=0
for every valid (i,j,m,n)
 sum:=0
 for each a from i to m
  for each b from j to n
   sum:=sum+a_ab
```

ans:=max(ans,sum)

return ans

b.$O(N^4)$ Algorithm for Maximum Submatrix Sum

Input: $N * N$ integer matrix a_{ij}

Output: 1 integer indicating the Maximum Subsequence Sum

Main Idea:

1. First, save $b_{ij} = \sum_{k=1}^{i} \sum_{l=1}^{j} a_{kl}$, then enumerate i, j, m, n, calculate the answer in $O(1)$ time using matrix $(b_{ij})_{N*N}$.

2. Use **Inclusion-Exclusion Principle**.

Pseudo Code:

ans:=0

for every valid (i,j)

b_{ij}:=$\sum_{k=1}^{i} \sum_{l=1}^{j} a_{kl}$

for every valid (i,j,m,n)

sum:=$b_{ij} - b_{i-1,j} - b_{i,j-1} + b_{i-1,j-1}$

ans:=max(ans,sum)

return ans

c.A better $O(N^3)$ Algorithm for Maximum Submatrix Sum

Input: $N * N$ integer matrix a_{ij}

Output: 1 integer indicating the Maximum Subsequence Sum

Main Idea:

1. save $b_{ij} = \sum_{k=1}^{i} \sum_{l=1}^{j} a_{kl}$, then enumerate i, j, calculate the answer in $O(N)$ time using matrix $(b_{ij})_{N*N}$, similar to $O(N)$ method of MSS.

2. Use **Inclusion-Exclusion Principle**.

Pseudo Code:

ans:=0

for every valid (i,j)

b_{ij}:=$\sum_{k=1}^{i} \sum_{l=1}^{j} a_{kl}$

for each i from 1 to N

for each j from 1 to N

sum:=0

for each k from 1 to N

sum:=sum+$b_{jk} - b_{i-1,k} - b_{j,k-1} + b_{i-1,k-1}$

sum:=max(sum,0)

ans:=max(ans,sum)

return ans

3.Testing Results

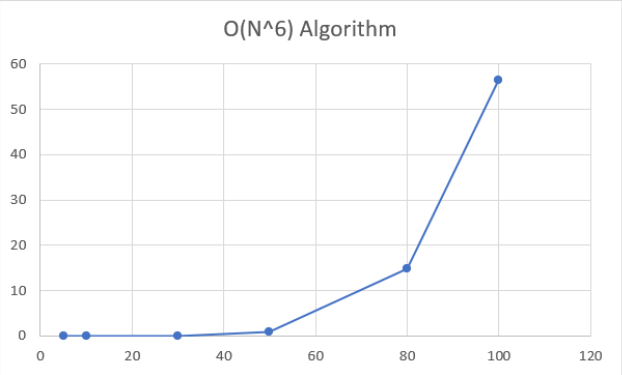
Test cases are omitted. Just use generator.c to generate some test cases.

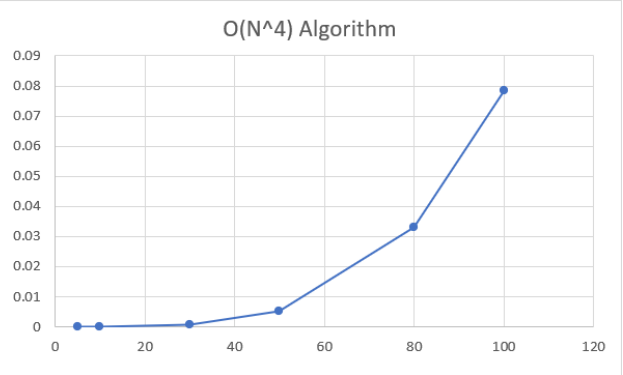
We can easily prove all three algorithms are correct and produce the same output if not overflowed.

Here below are some results for comparation on their running time on my personal computer.

	N	5	10	30	50	80	100
$O(N^6)$ Version	Iterations(K)	1E5	1E4	10	1	1	1
	Ticks	238	829	448	930	14774	56315
	Total Time(sec)	0.238	0.829	0.448	0.93	14.774	56.315
	Duration(sec)	2.38E-6	8.29E-5	0.0448	0.93	14.774	56.315
$O(N^4)$ Version	Iterations(K)	1E6	1E4	1E3	100	10	10
	Ticks	849	101	695	509	332	784
	Total Time(sec)	0.849	0.101	0.695	0.509	0.332	0.784
	Duration(sec)	8.49E-7	1.01E-5	6.95E-4	5.09E-3	0.0332	0.0784

The results is directly shown in the graph below; Please notice the different time scale.





4.Analysis and Comments

a.$O(N^6)$ Algorithm

Time Complexity Analysis: $O(N^6)$. There are 6 for-loop nested together, and each loop will run at most $N+1$ times.

Space Complexity Analysis: $O(1)$. Apparently.

b.$O(N^4)$ Algorithm

Time Complexity Analysis: $O(N^4)$. Calculate $(b_{ij})_{N*N}$ can be done in $O(N^2)$. To enumerate i, j, m, n, we simply need 4 for-loop nested together, and each loop will run at most $N+1$ times.

Space Complexity Analysis: $O(N^2)$. That's for saving $(b_{ij})_{N*N}$.

c.$O(N^3)$ Algorithm

Time Complexity Analysis: $O(N^3)$. Calculate $(b_{ij})_{N*N}$ can be done in $O(N^2)$. We need to enumerate i, j which will cost $O(N^2)$, and need another N-times for-loop in the enumration to get the maximum sum, so it's $O(N^3)$.

Space Complexity Analysis: $O(N^2)$. That's for saving $(b_{ij})_{N*N}$.

Appendix

The realization of the 3 algorithms in C can be found in solution.c, which seperates the $(b_{ij})_{N*N}$ calculation and algorithm proceeding and gives a simple structure to get the running times. A generator to generate available input can be found in generator.c. Note: Both generator and solution use stdin and stdout.

These codes are for comparation with the pseudo codes. You may better read the original code (to get some comments).

How to use the generator: Input the size N you want, and it will give out a $N*N$ matrix whose elements are in the range $[-100, 100]$.

How to use the solution: Input the size N and the $N*N$ matrix, and it will give out the answer and if you want, ticks and time used. You should change the function Algo_Nx() in main() to switch algorithms. Following are main codes of the algorithms. Matrix a_{ij} and b_{ij} are indicated by 2-dimensional array $area[][]$ and $sum[][]$.

The $O(N^6)$ Algorithm

```
int Algo_N6(){
    int ret=0;
    for(int i=1;i<=n;++i) for(int j=1;j<=n;++j){
        for(int k=i;k<=n;++k) for(int l=j;l<=n;++l){
            int sum=0;
            for(int m=i;m<=k;++m) for(int n=j;n<=l;++n){
                sum+=area[m][n];
            }
            if(sum>ret) ret=sum;
        }
    }
    return ret;
}
```

The $O(N^4)$ Algorithm

```
int Algo_N4(){
    DealSum();
    int ret=0;
    for(int i=1;i<=n;++i) for(int j=1;j<=n;++j){
        for(int k=i;k<=n;++k) for(int l=j;l<=n;++l){
            int nows=sum[k][l]-sum[i-1][l]-sum[k][j-1]+sum[i-1][j-1];
            if(nows>ret) ret=nows;
        }
    }
    return ret;
}
```

The $O(N^3)$ Algorithm

```
int Algo_N3(){
    DealSum();
    int ret=0;
```

```
    for(int i=1;i<=n;++i) for(int j=i;j<=n;++j){
        int nows=0;
        for(int k=1;k<=n;++k){
            nows+=sum[j][k]-sum[i-1][k]-sum[j][k-1]+sum[i-1][k-1];
            if(nows<0) nows=0;
            if(nows>ret) ret=nows;
        }
    }
    return ret;
}
```

The Pre-deal of b_{ij}

```
void DealSum(){
    for(int i=1;i<=n;++i) for(int j=i;j<=n;++j){
        sum[i][j]=area[i][j]+sum[i-1][j]+sum[i][j-1]-sum[i-1][j-1];
    }
}
```

Declaration

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