Lab:Performance Measurement(MSS)

浙江大学

2023 年 10 月 8 日

1 Question and its introduction

In class, we learn the algorithms of finding the maximum subsequence of a one-dimensional array. Similarly, now we should find algorithms to find the maximum submatrix of a two-dimensional array. So suppose we get a matrix whose size is N*N. We should write two basic functions whose time complexities are $O(N^6)$ and $O(N^4)$ respectively. Both of them can find maximum submatrix's sum. Additionally, if you have a better algorithm to solve the problem, you can write a function to realize it and get bonus.

2 Algorithms and test program

Algorithm 1 N^6 Algorithm

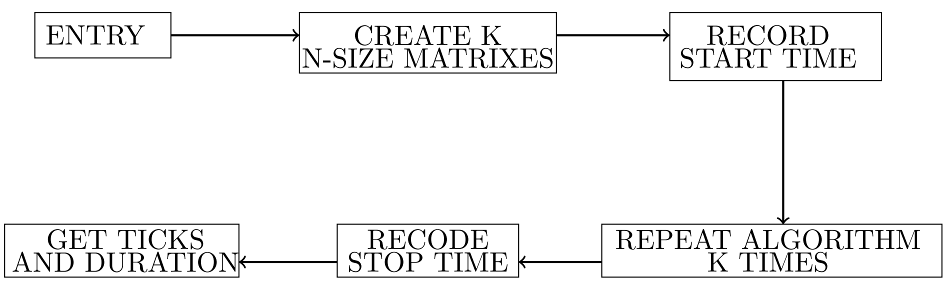
Require: matrix[ ][ ], width, height
Ensure: $MaxSum$ = the sum of maximum submatrix

$MaxSum \leftarrow 0$ (next enumerate all submatrix)
for BeginWidth = 0 to width-1 **do**
  for EndWidth = BeginWidth to width-1 **do**
    for BeginHeight = 0 to height-1 **do**
      for EndHeight = BeginHeight to height-1 **do**
        $ThisSum \leftarrow 0$ (now we can get the range of a submatrix)
        for i = BeginWidth to EndWidth **do**
          for j = BeginHeight to EndHeight **do**
            $ThisSum += matrix[j][i]$ (get the sum of the submatrix)
          end for
        end for
        if $ThisSum > MaxSum$ **then**
          $MaxSum = ThisSum$ (check and replace MaxSum)
        end if
      end for
    end for
  end for
end for
return MaxSum

Algorithm 2 N^4 Algorithm

```
Require: matrix[ ][ ], width, height
Ensure: MaxSum = the sum of maximum submatrix
MaxSum ← 0, Sum[width]
for BeginWidth = 0 to width-1 do
  for BeginHeight = 0 to height-1 do
    Sum[0 to width-1] = 0 (beginning at (BeginWidth, beginHeight))
    for i = BeginHeight to height-1 do
      LineSum = 0
      for j = BeginWidth to width-1 do
        LineSum += matrix[i][j]
        (get the sum of [i][BeginWidth] to [i][j])
        Sum[j] += LineSum (get the sum from [BeginHeight][BeginWidth] to [i][j])
        if Sum[j] > MaxSum then
          MaxSum = Sum[j]
        end if
      end for
    end for
  end for
end for
return MaxSum
```

As for test program, the details of test program(Lab1.c) are shown below.



3 Time and space complexities

3.1 N^6 Algorithm's complexities

Time complexity: Look at the algorithm's pseudo code: Line2 and Line3's time complexity is $O(N + (N-1) + \cdots + 1) = O(N*(N+1)/2)$. Line4 and Line5 are the same. Then notice that Line7 and Line8 can be all submatrix of the main matrix, so the maximum time complexity of Line7 and Line8 are $O(N^2)$, so the total time complexity is $O((\frac{N(N+1)}{2})^2 * N^2) = O(N^6)$

Space complexity: we know that it don't need new array or other data structures. Only a few variables. So the space complexity is $O(1)$

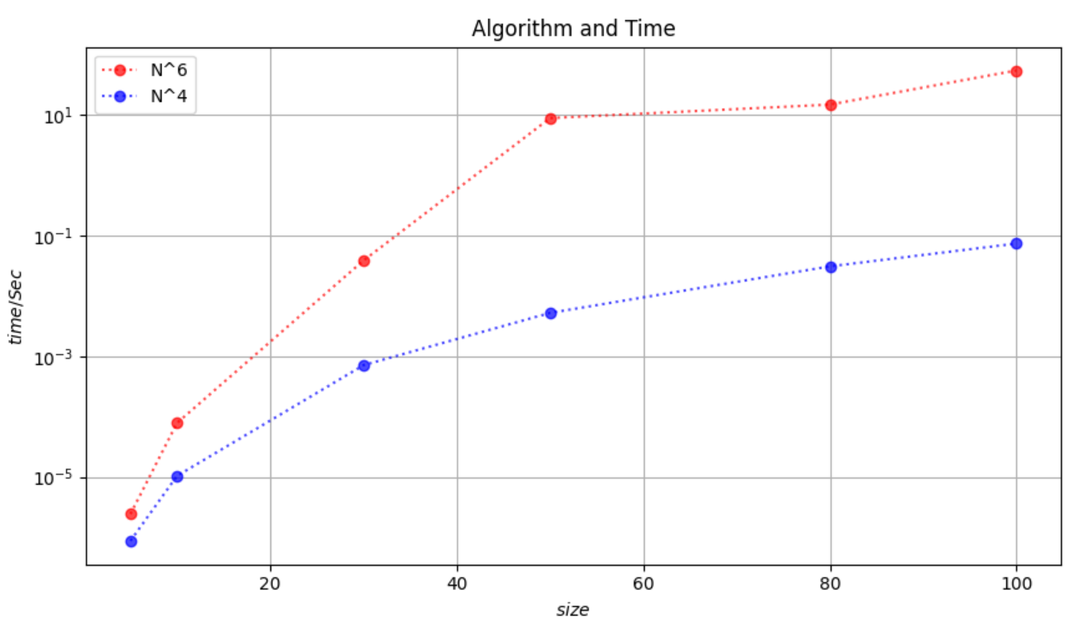
3.2 N^4 Algorithm's complexities

Time complexity: Look at the algorithm's pseudo code: Line2 and Line3 check all coordinates of matrix, so their time complexity is $O(N^2)$. Then Line4 sets Sum[0] to Sum[width-1] as 0, time complexity is $O(N)$. From Line5 to Line14, check all submatrix beginning from $(BeginWidth, BeginHeight)$, so the maximum time from Line5 to Line14 is $O(N^2)$. Then the total time complexity is $O(N^2 * (N + N^2)) = O(N^4)$

Space complexity: The difference between N^4 and N^6 algorithm is that N^4 algorithm use a array to store the results. And the size of the array is N. So the space complexity of N^4 algorithm is $O(N)$

4 Test and result:N=5,10,30,50,80,100

	N	5	10	30	50	80	100
$O(N^6)$ version	Interations(K)	500000	70000	100	10	2	1
	Ticks	1213	5484	3853	8800	29431	54101
	Total Time(sec)	1.213	5.484	3.853	8.800	29.431	54.101
	Duration(sec)	2.426e-6	7.834e-5	3.853e-2	8.800	14.716	54.101
$O(N^4)$ version	Interations(K)	500000	70000	6000	1000	500	100
	Ticks	436	729	4259	5209	15370	7335
	Total Time(sec)	0.4360	0.7290	4.259	5.209	15.370	7.335
	Duration(sec)	8.720e-7	1.041e-5	7.098e-4	5.209e-3	3.074e-2	7.335e-2



Obviously we can find N^6 and N^4 algorithms are different. N^4 Algorithm is faster than N^6. The reason is that , $\frac{N^6}{N^4} = N^2$, if N is very large, N^2 can eliminate the difference of coefficients of order term and become the main factor of run time. And the intrinsical reason is that, N^6 Algorithm checks all submatrix and each check is independent. So finding all submatrix uses $O(N^4)$ time, and getting the sum of submatrix uses $O(N^2)$ time — $O(N^6)$ totally. But $O(N^4)$ Algorithm just checks all points in matrix, which uses $O(N^2)$ time. Then checking all sum of submatrix that starts from the point. Array Sum[] help us remember the sum of submatrix and don't do extra compute. So getting the sum of submatrix uses $O(N^2)$ time. So $O(N^4)$ Algorithm is faster than $O(N^6)$.

5 A better Algorithm and test

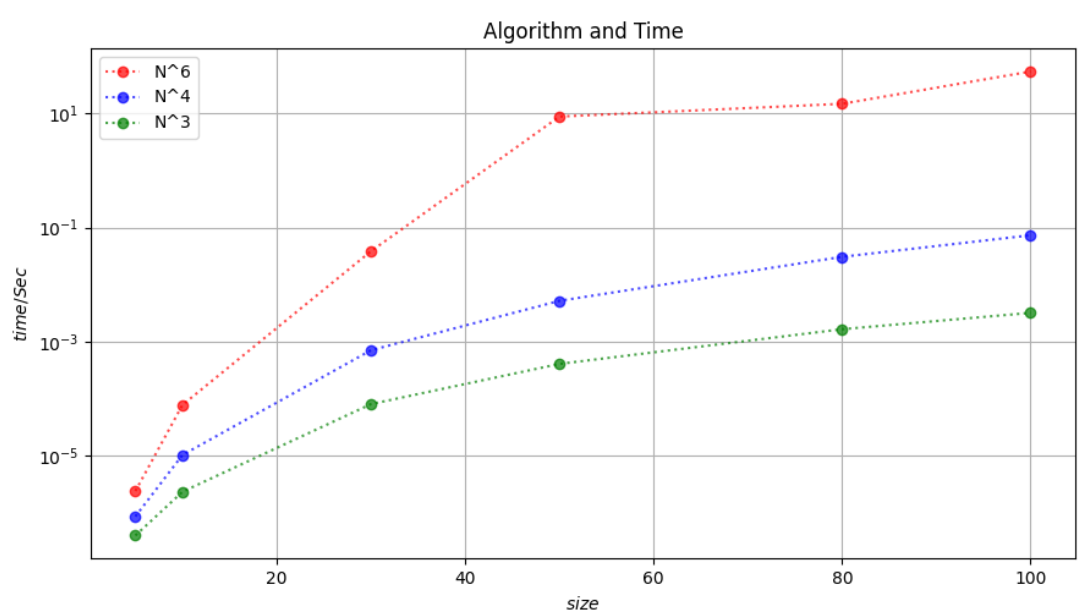
Algorithm 3 N^3 Algorithm

Require: matrix[ ][ ], width, height

Ensure: $MaxSum$ = the sum of maximum submatrix

```
MaxSum ← 0, Sum[width]
for BeginHeight = 0 to height-1 do
  Sum[0 to width-1] = 0
  for EndHeight = BeginHeight to height-1 do
    for k = 0 to width-1 do
      Sum[k] += matrix[EndHeight][k]
      ThisSum = MaxSubsequenceSum(Sum, width)
      if ThisSum > MaxSum then
        MaxSum = ThisSum
      end if
    end for
  end for
end for
return MaxSum
```

	N	5	10	30	50	80	100
$O(N^3)$ version	Interations(K)	500000	70000	10000	5000	1500	700
	Ticks	202	165	808	2046	2488	2244
	Total Time(sec)	0.202	0.165	0.808	2.046	2.488	2.244
	Duration(sec)	4.040e-7	2.357e-6	8.080e-5	4.092e-4	1.659e-3	3.206e-3



The N^3 Algorithm's time complexity is $O(N^3)$ and space complexity is $O(N)$. It uses a function called MaxSubsequenceSum: it's the same as teacher's ppt and it can find the max subarray of a one-dimensional array in $O(N)$ time. So we just need to check height from 0 to height-1: its time complexity is $O(N^2)$. So the total time complexity is $O(N^3)$. In test we can see that the $O(N^3)$ Algorithm is faster than $O(N^4)$ and $O(N^6)$, it's better algorithm.