Performance Measurement (MSS)

Author: 颜晗

Complete date:2021.10.05

Performance Measurement (MSS)

Chapter 1: Introduction

本次项目要求解决最大子矩阵和问题，参考教材中求最大子序列和问题的算法1、2实现两个类似的算法，并且时间复杂度分别为$O(N^6)$和 $O(N^4)$。另外要求自己设计并实现一个新的更好的算法。最后分析比较三种算法的时间与空间复杂度，并且比较样例测试的表现。

最大子矩阵问题：对于给出的N阶方阵，要找出它的一个子矩阵，满足该子矩阵中所有元素和在所有子矩阵中最大，程序要求输出该最大和。

算法简介：通过教材的阅读，可以看出算法1、2通过在遍历数组中的所有元素时再次遍历后面所有可以组合的的元素依次组合，并计算各个组合所有元素的和，依次比较择出最大项。算法2比之算法1节约了遍历第二个基准元素时重复计算前面元素和的时间。而对算法进行二维化的过程中，发现$O(N^4)$的算法2实际上也存在重复计算的问题，优化后可以实现时间复杂度为$O(N^3)$的算法。

Chapter 2: Algorithm Specification

算法一：该算法主要通过循环遍历矩阵，定位两个元素作为对角线形成一个子矩阵，再对该子矩阵求和并记录，如此查遍各子矩阵确定最大值，但是该过程有大量重复计算，运行速度较慢。伪代码如下：

```
1. 最大和 <- 0
2. for i <- 0  to  N
3. for j <- 0 to N  /*定位第一个基准数字*/
```

```
4. {
5.      for m <- i to N
6.      for n <- j to N /*定位第二个基准数字*/
7.      {
8.          暂时和 <- 0
9.          for p <- i to m
10.         for q <- j to n /*以两个基准数为对角线的矩形内部元素和
    计算*/
11.         {
12.             暂时和 <- 暂时和+矩阵[p][q]
13.         }
14.         if 暂时和 > 最大和
15.                 then 最大和 <- 暂时和
16.     }
17.}
```

算法二：同最大子序列问题一样，算法二相比算法一精简了重复遍历相同元素进行计算的步骤，在二号基准数遍历时即计算对应和，但是与一维的子序列问题不同，二维的矩阵更复杂一点，不可简单将遍历的 for 循环删除了事。

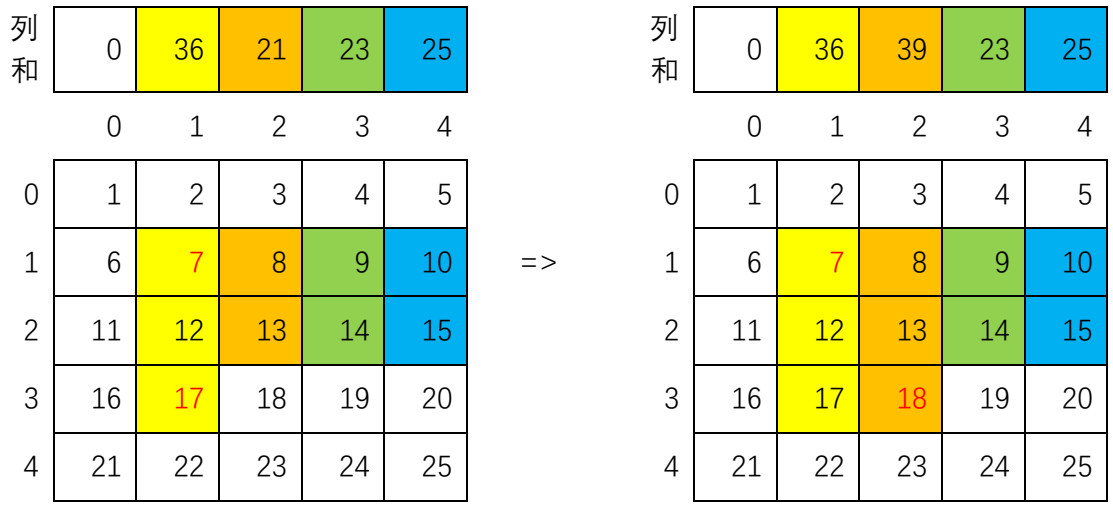
	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

（红数字为基准数，不同色块划定子矩阵范围）

如上图所示，由于在遍历时从左到右，从上到下，如果换行时清零显然会让上面的元素无法计算，但是一路加下去会如上图，本来只需计算 6+10+14 的和，实际上还加上了 7，8，11，12，这就导致了程序出错。

所以我们需要一种机制，既可以算上该计算的，也不能忽视即将

计算的。注意到从一号基准数划定子矩阵时，移动一行或一列时，那一行或列都是整体进入或退出，因此引入“列和”（或“行和”）这一概念。



如上图，列和数组在我们遍历矩阵时，实时将处于同一列的元素相加，这样在二号基准数换行向右遍历时，只需将与一号基准数所处列之间的各列和加起来就可以了，各列互不干扰，清楚划定需要计算的范围。例如，从 15 到 17，只需要将前面暂时记录的子矩阵和清零，更新并加上 1 号列的和即可；从 17 到 18，不需清零，更新并加上 2 号列的和即可，如此依次下去。另外，需要注意的是，当一号基准数更换时，需要计算的列的范围可能有所改变，因此列和数组也要重置清零。伪代码如下：

```
1. 最大和 <- 0
2. for i <- 0  to  N
3. for j <- 0 to N  /*定位第一个基准数字*/
4. {
5.       /*定义一个大小为N的数组，初始化为0*/
6.       /*数组存储两个基准数之间低维下标(列数)相同的元素和*/
7.     for m <- i to N
8.         暂时和 <- 0 /*第二个基准数换行时需重置计算的和*/
9.         for n <- j to N
10.        {
```

```
11.             列和[n] <- 列和[n] + 矩阵[m][n]
12.             暂时和 <- 暂时和 + 列和[n]
13.             if 暂时和 > 最大和
14.                     then 最大和 <- 暂时和
15.         }
16.}
```

额外算法：算法二相比算法一已经简单了两个级数，但是只需仔细分析一个具体样例就可以发现算法二同样存在重复计算的问题。

列和	0	36	39	23	25
	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

=>

列和	0	0	39	23	25
	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

如上图，当一号基准数从 7 更换到 8，二号基准数向右向下遍历时是存在重复计算列和的现象的。也就是说，处于同一行的一号基准数实际上重复计算了一些列和。而对问题进行抽象，求矩阵和就是求矩阵内各列和，而当列和计算出来，两行间的数据缩减成了一行，问题实际上转变成了求最大子序列和。也就是说我们只需要锁定两个基准行，计算行间的列和，再求最大子序列的和即可，而教材中给出了复杂度为 O(N) 的算法。

列和	6	7	8	9	10
	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15

=>

列和	17	19	21	23	25
	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15

3	16	17	18	19	20
4	21	22	23	24	25

3	16	17	18	19	20
4	21	22	23	24	25

（红色标定基准行，可以重叠；黄色为求最大值范围）

如上图，标定两行后，便可从 0 号列开始向后遍历，计算列和同时就将遍历元素的最大值求出；一号基准行相同可以使用同一列和数组。伪代码如下：

```
1. 最大和 <- 0
2. for i <- 0  to  N  /*一号基准列*/
3. {
4.     /*声明列和矩阵*/
5.     /*所有一号基准列数相同的组合共用一个列和矩阵*/
6.     for j <- i to N /*二号基准列*/
7.     {
8.         暂时和 <- 0
9.         for k <- 0 to N
10.        {
11.            列和[k] <- 列和[k] + 矩阵[j][k]
12.            暂时和 <- 暂时和 + 列和[k]
13.
14.            if 暂时和 > 最大和
15.                  then 最大和 <- 暂时和
16.            else if 暂时和 < 0
17.                  then 暂时和 <- 0
18.        }
19.    }
20.}
```

Chapter 3: Testing Results

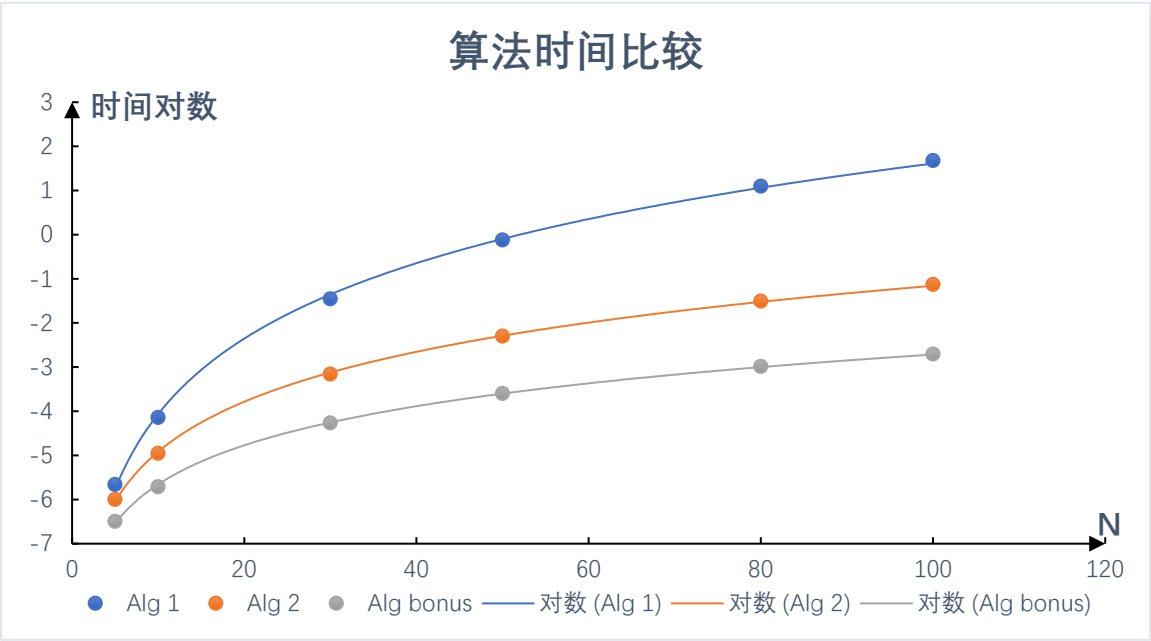
由于数据庞大，不便粘贴，且对于该问题基本需遍历所有子矩阵和，不存在较特殊的数据，下面只列表将数据特征与算法表现列出，正确性由各算法互相验证。

	$O(N^6)$ version	$O(N^4)$ version	$O(N^3)$ version
完全随机矩阵	pass	pass	pass
全正矩阵	pass	pass	pass
全负矩阵	pass	pass	pass

而对于不同的 N，各个算法的表现如下表。

	N	5	10	30	50	80	100
$O(N^6)$ version	**Iterations(K)**	100000	10000	10	1	1	1
	Ticks	217	717	349	757	12526	47580
	Total Time(sec)	0.217	0.717	0.349	0.757	12.526	47.58
	Duration(sec)	0.00000217	0.0000717	0.0349	0.757	12.526	47.58
$O(N^4)$ version	**Iterations(K)**	100000	10000	1000	10	10	10
	Ticks	100	111	689	50	310	741
	Total Time(sec)	0.1	0.111	0.689	0.05	0.31	0.741
	Duration(sec)	0.000001	0.0000111	0.000689	0.005	0.031	0.0741
$O(N^3)$ version	**Iterations(K)**	100000	100000	5000	1000	100	100
	Ticks	32	193	270	252	103	196
	Total Time(sec)	0.032	0.193	0.27	0.252	0.103	0.196
	Duration(sec)	0.00000032	0.00000193	0.000054	0.000252	0.00103	0.00196

各算法平均用时与 N 的关系如下图。



需要格外指出的是，对于额外算法来说，全负和全正矩阵的计算要比完全随机矩阵的计算快出较为稳定的一段时间,例如,对于N=100,迭代次数为 100 的条件，全负矩阵通常比完全随机矩阵快 0.04s 左右，暂时未发现原因。

Chapter 4: Analysis and Comments

算法一：经历了六重循环，注意遍历，可列出数学表达式

$$\sum_{i=0}^{N}\sum_{j=0}^{N}\sum_{m=i}^{N}\sum_{n=j}^{N}(m-i)(n-j)$$

由表达式得到结果，约去较小项，可得时间复杂度为O(N^6)。

对于算法一，只需要空间存储最大值以及各子矩阵的和，空间复杂度为O(1)。

算法二：同算法一，得出数学表达式

$$\sum_{i=0}^{N}\sum_{j=0}^{N}(N-i)(N-j)$$

约去较小项，时间复杂度为O(N^4)。

除去最大值与子矩阵和外，还需要一个大小为N的数组来储存列和。空间复杂度为O(N)。

算法三：数学表达式为

$$\sum_{I=0}^{N}N(N-i)$$

约去较小项，时间复杂度为O(N^3)。

同算法二，所用空间相同，空间复杂度为O(N)。

二维矩阵情况多样，但是基本无及特殊的情况，所有算法都需要对矩阵进行大规模遍历，唯一可想的一处再化简就是，对于额外算法，定位一号基准行时，如果计算列和时发现该行全负，可以直接跳过，将下一行确定为下一个一号基准行，可再跳过一些计算。

Appendix: Source Code (in C)

```c
/*这个程序用于测试本次项目所写的三种算法
 *请依照提示输入，尽可能避免非法输入
 */
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

void init(int N,int A[]);//Initialize an array with size of N
int maxsubmatricsum_1(int N,int A[N][N]);//the function to be test ( O(N^6) version )
int maxsubmatricsum_2(int N,int A[N][N]);//the function to be test ( O(N^4) version )
int maxsubmatricsum_3(int N,int A[N][N]);//the function to be test ( O(N^3) version )

clock_t start,stop;
double duration;

//first version,this version is for testing or some special example you want to input yourself
int main()
{
 int N;
 int i,j,k;
 int (*maxsubmatricsum) (int N,int A[N][N]);
 while(1){
  printf("请输入矩阵大小N,如果想退出程序，请输入任一负数: ");
  scanf("%d",&N); //input the size of the matric
  if(N<0)return 0;//if N=-1,terminate the program
  int matric[N][N];
  printf("是否想自己输入数据(1-手动输入;0-自动生成): ");
  scanf("%d",&i);
  if(i==1){
```

```
30.    printf("请输入矩阵数据：\n");
31.    for (i=0;i<N;i++){
32.     for(j=0;j<N;j++){
33.      scanf("%d",&matric[i][j]); //you could input the data
   you want
34.     }
35.    }
36.   }
37.   else if(i==0){
38.    srand(clock());     //Reset the random number seed
39.    for(i=0;i<N;i++){
40.     for(j=0;j<N;j++){
41.      matric[i][j]=rand()%100;  //generate the random number
42.      if(rand()%2)matric[i][j]=-matric[i][j];
43.      //the random number can only be positive,
44.      //the symbol need also be random.
45.     }
46.    }
47.    for(i=0;i<N;i++){
48.     for(j=0;j<N;j++){
49.      printf("%3d ",matric[i][j]);
50.     }
51.     printf("\n");
52.    }
53.   }
54.   else {
55.    printf("错误输入，请从头开始\n");
56.    continue;
57.   }
58.   int ver;
59.   printf("输入想测试的算法版本，1-O(N^6),2-O(N^4),3-
   O(N^3)：  ");
60.   scanf("%d",&ver);
61.
62.   switch(ver)//determine the function
63.   {
64.    case 1:maxsubmatricsum=maxsubmatricsum_1;
65.     break;
66.    case 2:maxsubmatricsum=maxsubmatricsum_2;
67.     break;
68.    case 3:maxsubmatricsum=maxsubmatricsum_3;
69.     break;
70.    default:printf("错误输入，请从头开始\n");
71.     continue;
```

```
72.     break;
73.   }
74.   int K;
75.   printf("请输入迭代次数:  ");
76.   scanf("%d",&K);
77.
78.   int result;
79.   start=clock(); //the starting time
80.   for(k=0;k<K;k++){ //Increase program running time,you can
    change the number of iterations
81.    result=maxsubmatricsum(N,matric);
82.    }
83.   stop=clock(); //the ending time
84.   duration=((double)(stop-
   start))/CLK_TCK; //the time we have used
85.   printf("the max is %d.\nthe time of it is %lf s.\n",resul
   t,duration); //print the result
86. }
87. return 0;
88.}
89.int maxsubmatricsum_1(int N,int A[N][N])
90.{
91. int i,j,m,n,p,q;
92. int Max=0;  //the maximum is 0 at first
93. int temp;
94. for(i=0;i<N;i++){
95.   for(j=0;j<N;j++){ //the i,j is used to determine the firs
   t number
96.
97.    for(m=i;m<N;m++){
98.     for(n=j;n<N;n++){ //the m,n is used to determine the se
   cond number
99.
100.       temp=0; //the sum of one submatric,at first ,it is 0
   .
101.       for(p=i;p<=m;p++){
102.        for(q=j;q<=n;q++){ //the p,q is used to determine t
   he number to be add to the sum
103.         temp+=A[p][q];
104.        }
105.       }
106.       if(temp>Max)Max=temp; // update the maximum
107.
108.     }
```

```
109.     }
110.
111.    }
112.   }
113.
114.   return Max;
115.  }
116.  int maxsubmatricsum_2(int N,int A[N][N])
117.  {
118.   int i,j,m,n;
119.   int Max=0; //the maximum is 0 at first
120.   int temp;
121.   for(i=0;i<N;i++){
122.    for(j=0;j<N;j++){ //the i,j is used to determine the fi
    rst number
123.     int ttemp[N]; //store the sum of elements which are in
     the same column
124.     init(N,ttemp);
125.     for(m=i;m<N;m++){
126.      temp=0;  //the sum of one submatric,when the second n
    umber is at a new row,it is 0.
127.      for(n=j;n<N;n++){ //the m,n is used to determine the
    second number
128.       ttemp[n]+=A[m][n];
129.       temp+=ttemp[n];
130.
131.       if(temp>Max)Max=temp; // update the maximum
132.
133.      }
134.     }
135.
136.    }
137.   }
138.
139.
140.   return Max;
141.  }
142.  int maxsubmatricsum_3(int N,int A[N][N])//the function to
     be test ( O(N^3) version )
143.  {
144.   int i,j,k;
145.   int Max=0; //the maximum is 0 at first
146.   int temp;
```

```
147.   for(i=0;i<N;i++){ //the i is used to determine the first
    raw
148.    int ttemp[N];
149.    init(N,ttemp);
150.
151.    for(j=i;j<N;j++){ //the j is used to determine the seco
   nd raw
152.     temp=0;
153.     for(k=0;k<N;k++){ //the k is used to determine the col
   umn to be calculate
154.      ttemp[k]+=A[j][k];
155.      temp+=ttemp[k];
156.      if(temp>Max)Max=temp; // update the maximum
157.      else if(temp<0)temp=0; //if the sum of the front subm
   atric is negative,reset the sum
158.     }
159.
160.    }
161.
162.   }
163.
164.   return Max;
165.
166.  }
167.  void init(int N,int A[])
168.  {
169.   int i;
170.   for(i=0;i<N;i++){
171.    A[i]=0;
172.   }
173.  }
```

Declaration

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

我在此声明，在这个名为“Performance Measurement (MSS)”的项目中所做的所有工作都是我独立完成的。