2018 数据结构基础秋冬学期 Project1 实验报告

指导教师:冯雁

一、 实验目的

对算法时间、空间复杂度进行简单的分析，使用测量手段测量程序的时间、内存开销。

二、 题目描述

给定一个 n*m 的矩阵 A，A 的元素为整数，求 A 的一个子矩阵 B，使得 B 中元素之和最大。

三、 算法描述

算法一：

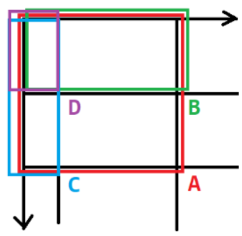
暴力枚举所有子矩阵。枚举子矩阵的左上角、右下角，对于每个子矩阵进行求和。

算法二：

设 S[i][j]表示以元素 a[i][j]为右下角，以元素 a[1][1]为左上角的矩阵的和。则 S 可由递推式：

$$S[i][j]=S[i-1][j]+S[i][j-1]-S[i-1][j-1]+a[i][j];$$

求出。



在求答案时，先按上述方法预处理出 S 数组，并按算法一的方法枚举所有子矩阵，但在求一个左上角为(xi,xj)，右上角为(yi,yj)子矩阵时只需要计算

$$Sum=S[y_i][y_j]-S[x_i-1][y_j]-S[y_i][x_j-1]+S[x_i-1][x_j-1];$$

即在图中 Sum=A-B-C+D，则对于任意一个子矩阵的和可以在常数时间内求出。

算法三:

使用动态规划法，**首先考虑一个一维数组的最大区间和**，例如序列：

i	1	2	3	4	5	6	7	8	9
a[i]	5	7	-4	-9	8	-2	5	2	-7

设：**F[i]为以 a[i]为最后一个元素的区间的最大和。**

则上述序列的 f 数组如下：

i	1	2	3	4	5	6	7	8	9
F[i]	5	12	8	-1	8	6	11	13	6
最大值区间	[1,1]	[1,2]	[1,3]	[1,4]	[5,5]	[5,6]	[5,7]	[5,8]	[5,9]

观察上述表格，可以得到 f 的状态转移方程：

$$F[i] = \begin{cases} 0 & i == 0 \\ F[i-1] + a[i] & F[i-1] > 0 \\ a[i] & F[i-1] <= 0 \end{cases}$$

原理如下：对于 a[i-1]结尾的区间，当在考虑 a[i-1]后面的 a[i]时，F[i]的值可以是 F[i-1]+a[i]或者 a[i]（也就是说 F[i]的值可以是：在以 a[i-1]结尾的区间最大值后面追加一个 a[i]形成一个长度+1 的区间；或者让 a[i]单独成为一个区间。显然当 F[i-1]>0 时，F[i-1]+a[i]>a[i]，否则当 F[i-1]<=0 时，F[i-1]+a[i]<=a[i]）。

由此可以在线性时间内求得 F 数组。最终答案的最大值也就是 F 数组中的最大值。

下面将问题扩展至二维矩阵。

当问题变为二维矩阵时，我们可以对行枚举，对列进行动态规划。例如，用 x,y 表示计算第一行为 x，最后一行为 y 的子矩阵。设 S[i][j]表示第 j 列，从第 1 行到第 i 行的元素和（即第 j 列的前缀和）。则对于每次枚举的 x,y，将原方程中的 a[i]替换为 S[y][i]-S[x-1][i]即可。

$$F[i] = \begin{cases} 0 & i == 0 \\ F[i-1] + S[y][i] - S[x-1][i] & F[i-1] > 0 \\ S[y][i] - S[x-1][i] & F[i-1] <= 0 \end{cases}$$

总结：

上述算法一至算法三，复杂度递减。从算法一到算法二，优化掉了逐一计算每个子矩阵的值得巨大开销。从算法二到算法三，将四维状态枚举优化成二维枚举+一维递推，省去了一层枚举的循环。

现在，我们来考虑为什么算法三是正确的。

一维数组而言：

首先，我们让 F[i]表示的是以 a[i]结尾的区间的最大和，这意味着，我们每求出一个 F[i]都是当前的最优答案，也就是局部最优解。所有局部最优解中最优的解，是全局最优解。

其次，对于每多考虑一个元素，只需要判断它自身和前面的元素接在一起会不会使答案更优，且对于每一个元素，都不需要考虑其他局部解的具体解法，比如对于某一个 i，F[i-1]+a[i]==a[i]，这时，选择和之前的区间连在一起，或者独立成为一个区间，对之后的决策没有任何影响，最后得出的答案也是一样的。也就是说每一个状态的决策都是无后效性的。

这体现了动态规划解法中重要的三个特点：最优子结构、重叠子问题，决策无后效性。

四、 复杂度分析

以下分析认为 n，m 为同一数量级的变量。

算法一：

枚举左上角和右下角各需要 O(n^2)的时间，对于每一个子矩阵求和需要 O(n^2)的时间。所以总时间复杂度为 O(n^6)。

由于只需要存储原数组，空间复杂度为 O(n^2)。

算法二：

由于省去了算法一中求子矩阵的时间，所以总时间复杂度为 O(n^4)。

空间复杂度为 O(n^2)。

算法三：

对行枚举的时间复杂度为 O(n^2)，对列递推的时间复杂度为 O(n)，总时间复杂度为 O(n^3)。

空间复杂度仍为 O(n^2)。

五、 正确性测试

测试方法：

使用随机数据生成器（详见../code/gen.cpp），对三个算法代码进行对拍（对拍使用 bash 脚本，详见../code/checker.sh）。

数据生成器接受三个参数 n,m,lim，表示题目中 n,m 和矩阵元素绝对值的最大值，生成的数据在区间(-lim,lim)内均匀分布。

测试脚本每次生成一组数据，将三个程序的结果保存至 out1,out2,out3 三个文件，然后使用 diff -w 命令比较三个结果文件。如果发现三个文件结果不同则立即停止测试，否则无限重复测试过程。

测试结果：

三种算法在如下测试情形下**对拍十分钟，未发现结果差异**。

n	5	6	15	12	13	30	51	80	101	120	121
m	5	6	15	13	10	30	51	80	101	113	154

上述数据对于 n,m 的大小关系、奇偶性予以考虑，以保证测试的可靠性。

结论：

三种算法对于相同的数据输出结果相同，算法正确性基本可靠。

六、 性能测试

1. 为保证测试的准确性，测试程序编译时不使用优化器，并取 10~100 次结果平均值记录。
2. 为方便测试，运行时间为 linux 系统下 time 命令输出。
3. 迭代器总次数由函数 Inc(int&x)调用总数决定，使用 gprof 统计。
4. 由于编译时加入了-pg 参数，所以运行时间与正常运行时间相比较慢。
5. 部分数据可见../code/tempdata/Mesure1 等文件。

(迭代器次数为 for 循环中的计数器自增运算的次数，能够更直观反应运算次数)

Testing machine info:

CPU: *Intel® Core™ i7-7700HQ CPU @2.80GHz 2.80GHz*

RAM: 16GB

OS: *Ubuntukylin18.04 LTS 64bits*

Compiler: *GNU G++ v7.3.0*

Compiling Command:

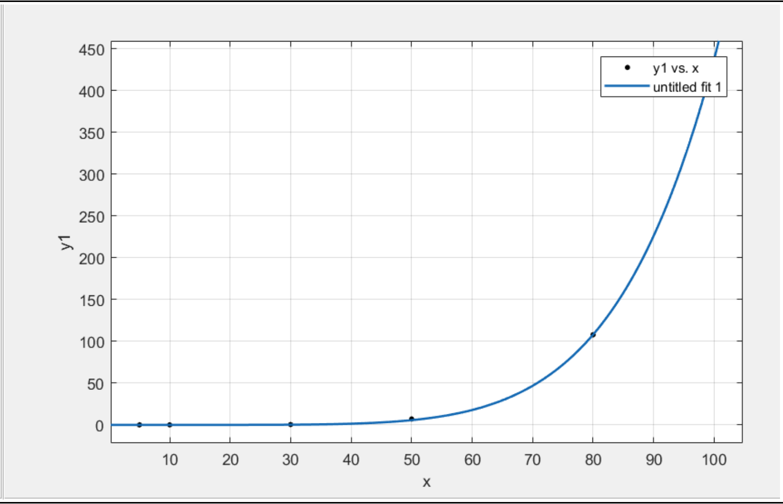
```
g++ ./gen.cpp -o ./gen -O2 -std=c++11
g++ ./Solution1.cpp -o ./Solution1 -g -pg
g++ ./Solution2.cpp -o ./Solution2 -g -pg
g++ ./Solution3.cpp -o ./Solution3 -g -pg
```

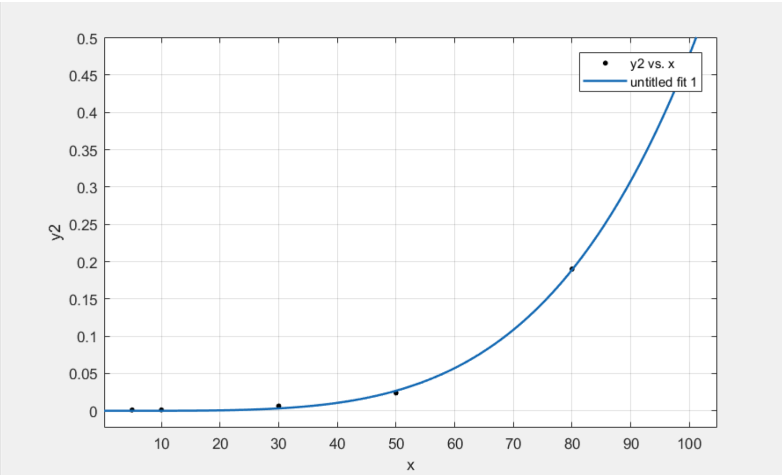
Testing Code: *(See file ../code/test.sh)*

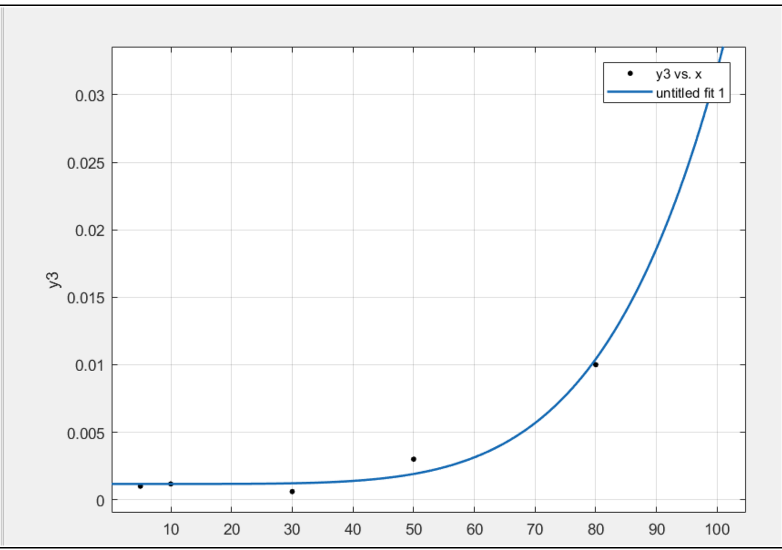
Algorithm1 O(n^6):						
No.	1	2	3	4	5	6
n,m	5	10	30	50	80	100
Test_cases	100	100	100	50	20	10
real_time(s)	0.00211	0.00307	0.39660	7.053	107.402	437.516
sys_time(s)	0.00091	0.00161	0.00480	0.024	0.020	0.024
user_time(s)	0.00107	0.00130	0.38000	7.000	107.773	437.423
Iterator_Counts	2,110	64,295	27,140,035	518,281,975	3,845,610,464	4,604,198,924
Algorithm2 O(n^4):						
No.	1	2	3	4	5	6
Test_cases	100	100	100	50	20	10
n,m	5	10	30	50	80	100
real_time(s)	0.00281	0.00296	0.04730	0.031	0.188	0.545
sys_time(s)	0.00101	0.00098	0.00050	0.008	0.000	0.000
user_time(s)	0.00107	0.00120	0.00640	0.024	0.190	0.477
Iterator_Counts	360	3,795	232,035	1,694,475	10,769,760	26,027,700
Algorithm3 O(n^3):						
No.	1	2	3	4	5	6
Test_cases	100	100	100	50	20	10
n,m	5	10	30	50	80	100
real_time(s)	0.00206	0.00211	0.00250	0.003	0.020	0.040
sys_time(s)	0.00097	0.00087	0.00190	0.000	0.000	0.008
user_time(s)	0.00100	0.00116	0.00060	0.003	0.010	0.032
Iterator_Counts	210	725	15,375	67,625	269,000	520,250

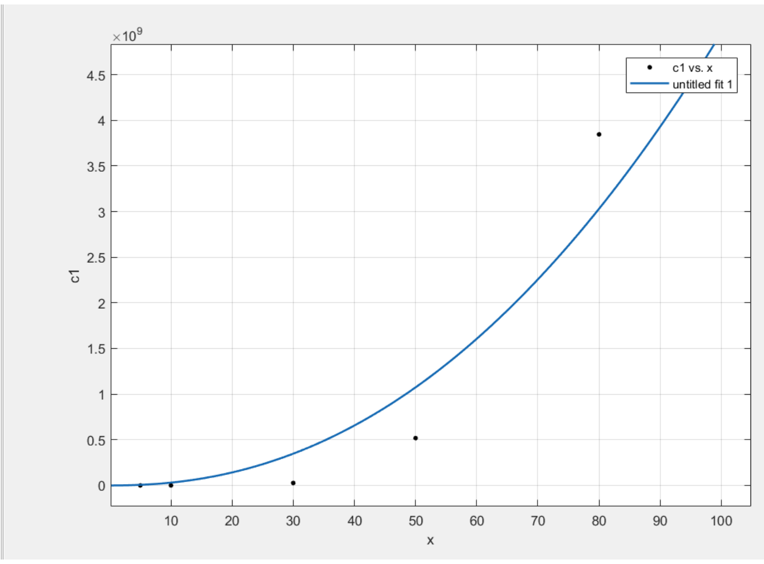
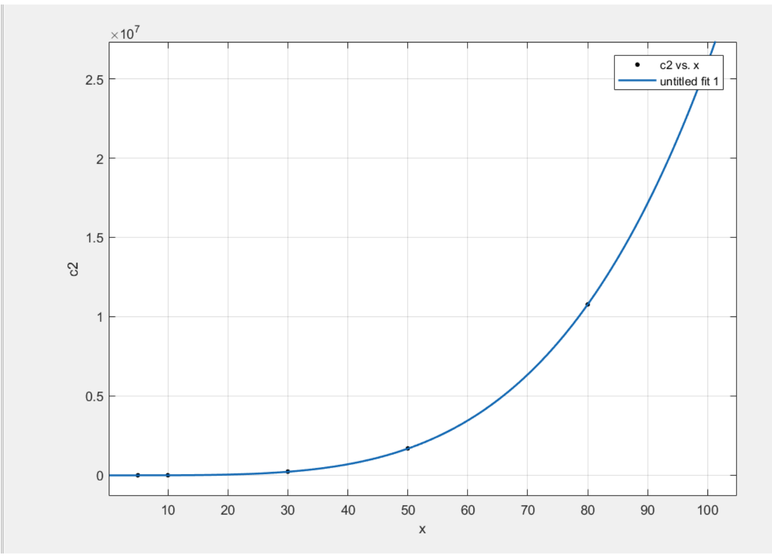
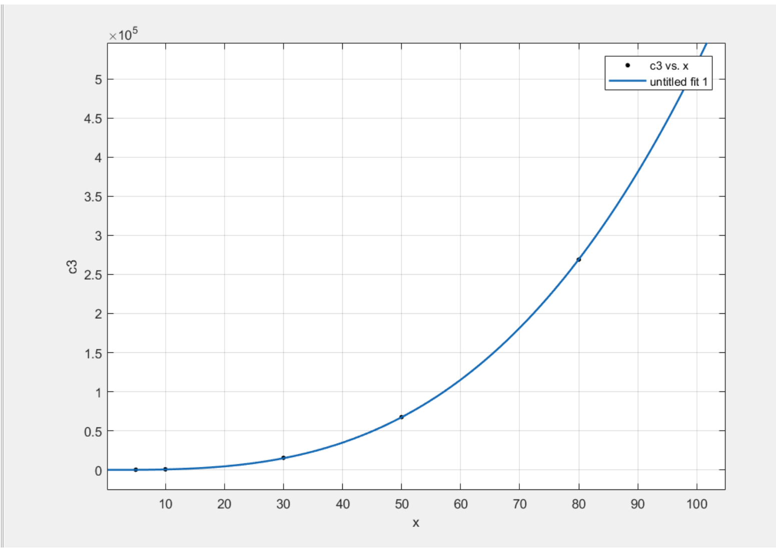
以下图表数据由 **Matlab** 生成。由于取样点较少，所以拟合的曲线可能存在偏差。

<table>
<tr><td></td><td colspan="2">UserTime</td></tr>
<tr><td>Algorithm1</td><td>Results
General model Power1:
f(x) = a*x^b
Coefficients (with 95% confidence bounds):
a = 1.27e-10 (8.083e-11, 1.732e-10)
b = 6.268 (6.189, 6.348)

Goodness of fit:
SSE: 1.83
R-square: 1
Adjusted R-square: 1
RMSE: 0.6764</td><td>
</td></tr>
<tr><td>Algorithm2</td><td>Results
General model Power1:
f(x) = a*x^b
Coefficients (with 95% confidence bounds):
a = 2.386e-09 (7.477e-10, 4.024e-09)
b = 4.151 (4, 4.301)

Goodness of fit:
SSE: 2.185e-05
R-square: 0.9999
Adjusted R-square: 0.9999
RMSE: 0.002337</td><td>
</td></tr>
<tr><td>Algorithm3</td><td>Results
General model Power2:
f(x) = a*x^b+c
Coefficients (with 95% confidence bounds):
a = 5.353e-13 (-2.765e-12, 3.836e-12)
b = 5.38 (4.04, 6.719)
c = 0.001164 (-0.00014, 0.002468)

Goodness of fit:
SSE: 1.79e-06
R-square: 0.9976
Adjusted R-square: 0.9961
RMSE: 0.0007724</td><td>
</td></tr>
<tr><td></td><td colspan="2">Iterator_Counts</td></tr>
</table>

Algorithm1	Results General model Power1: f(x) = a*x^b Coefficients (with 95% confidence bounds): a = 1.915e+05 (-1.171e+06, 1.554e+06) b = 2.207 (0.6332, 3.78) Goodness of fit: SSE: 1.203e+18 R-square: 0.9472 Adjusted R-square: 0.934 RMSE: 5.484e+08	
Algorithm2	Results General model Power1: f(x) = a*x^b Coefficients (with 95% confidence bounds): a = 0.3257 (0.3121, 0.3392) b = 3.951 (3.942, 3.96) Goodness of fit: SSE: 2.684e+08 R-square: 1 Adjusted R-square: 1 RMSE: 8191	
Algorithm3	Results General model Power1: f(x) = a*x^b Coefficients (with 95% confidence bounds): a = 0.6577 (0.6186, 0.6969) b = 2.949 (2.936, 2.962) Goodness of fit: SSE: 4.455e+05 R-square: 1 Adjusted R-square: 1 RMSE: 333.7	

对于运行时间，用函数 f(x) = a*x^b 拟合数据，算法一的 b=6.286，算法 2 的 b=4.151，而算法三的 b=5.38，但 R-square 值较小，数据拟合度较差，可能由于运行时间过短，易受其他因素影响；对于迭代器次数，因算法一的常数较小，所以拟合度较低，但算法二、三拟合度几乎为 100%。

结论：实际测量的程序运行时间增长与理论预期基本一致。

七、 总结

此次实验中，通过逐步的算法优化和反复调试，以及之后的一系列测试，我组成员充分的了解了算法优越性的重要性和算法分析的必要性，从算法一简单的暴力枚举到算法三中的动态规划，整体算法复杂度得到了充分的降低。

通过计算、测量程序的时间复杂度，并优化算法，使我组成员充分了解了算法分析与算法优化的重要性，并从中掌握了代码分析、测试与优化的基本方法，在课程学习中有重要意义。