数据结构与算法实验报告 2——最近点对

软件 52 陈书新 2015013229

csx15@mails.tsinghua.edu.cn

一、实验简介

实现求平面上最近点对的复杂度为$\Theta(n\log n)$的算法。

首先在$x,y\in[-10^9,10^9]$的二维平面上随机生成若干个整数点，再求出最近的点对以及它们之间的距离。

二、实验环境

操作系统：macOS Sierra 10.12.3

处理器：2.5 GHz Intel Core i7

内存：16GB 1600MHz DDR3

编辑器：Xcode 8.0 8A218a (Release)

编译环境：clang-800.0.38

三、实验过程

随机数的生成直接使用 c++的 random 库自带的库函数，可以随机生成指定范围内的整数，详情见源代码。

图形界面实现方式较为简单，在此不再赘述。

本实验中，我们实现了四种不同的求平面上最近点对的算法，并比较它们的理论时间复杂度和实际运行时间。

第一种算法是纯暴力算法，依次枚举两个点，算出它们的距离并与当前的最小值进行比较。该算法的时间复杂度是$\Theta(n^2)$，且没有什么可以优化的地方。由于时间复杂度较高，在$n\geq 10^6$时已经无法在短时间内运行结束。因此这种算法不参与最后的比较。

第二种算法是优化过的暴力算法。首先我们估计一个最近距离的上界，把它设为当前的最小值。然后将所有点对x轴进行排序，再依次枚举两个点，并实时更新最小值。当两个点的x轴距离大于最小值时，第二个点再往后枚举就是无意义的了，此时跳出循环。我在想出该算法后，再网上并没有查到相关的内容，所以我将该算法以某大神的名字来命名，叫 Yufeng's Algorithm。可以证明，该算法的时间复杂度为$O(n\sqrt{n})$。见补充材料[1]。

第三种算法基于分治法。首先对x轴进行排序。随后递归的计算最近点对。将当前的点分成左右两半，分别计算最近点对后再合并计算最近点对。在合并的过程中，我们需要对这

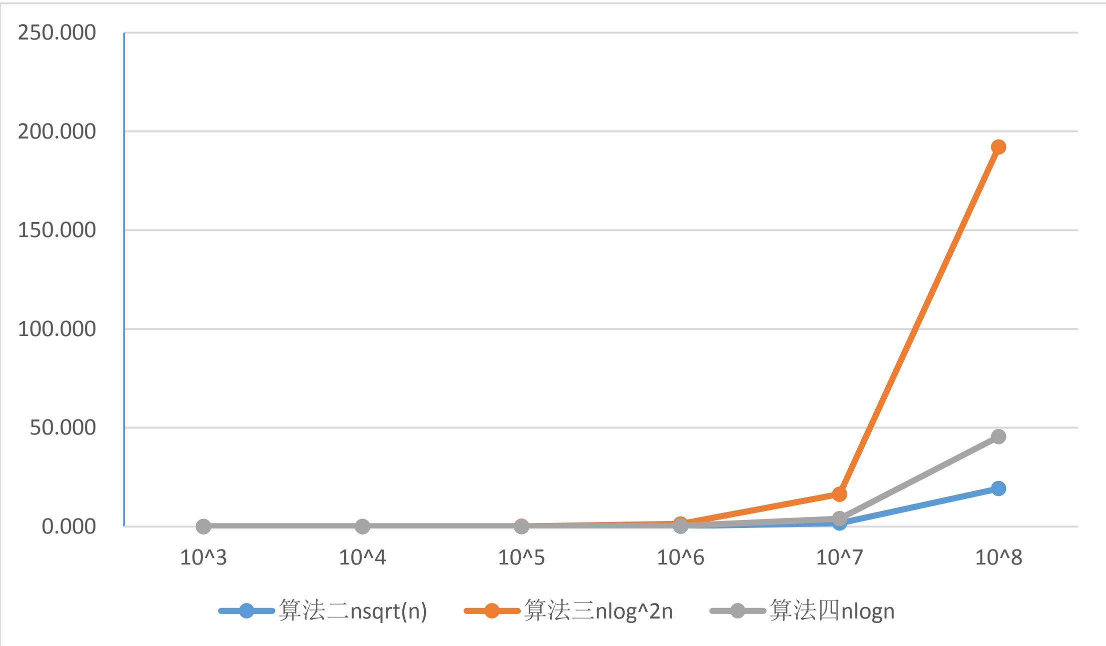
些点的y轴进行排序，然后对于每个点，找到排序后的后面 6 个点，计算距离并与最小值进行比较。这里的y轴排序我们可以使用普通的排序算法，单次时间复杂度为$\Theta(n\log n)$。总的时间复杂度由递推式$T(n)=2T(n/2)+\Theta(n\log n)$决定，这个递推式并不能用主定理解决，只能递归展开证明，该算法的时间复杂度为$\Theta(n\log^2 n)$。见补充材料[2]。

第四种算法是对第三种算法的优化。我们可以使用归并排序代替普通的排序算法，这样可以完美的使用递归的性质，在递归的时候同时进行排序和计算。显然，该算法的时间复杂度为$\Theta(n\log n)$。

得到了理论复杂度之后，我们开始计算实际运行时间。算法二、三、四的实际运行时间见下表。

算法|n	10^3	10^4	10^5	10^6	10^7	10^8
算法二	0.000	0.001	0.013	0.145	1.703	19.227
算法三	0.000	0.007	0.095	1.286	16.387	192.041
算法四	0.000	0.002	0.028	0.338	3.949	45.485

根据上表我们得到折线图。



从上表中我们可以发现，算法二和算法四的实际运行时间符合预期，但是算法三快得离谱，完全优于算法四。我们需要深究算法三的时间复杂度上界。

观察算法三和算法四的折线，我们发现算法四的实际运行时间一直在算法三的两倍左右，我们可以大概估计出算法三的时间复杂度上界应当为$\Theta(n\log n)$。证明需要用到概率论的一些知识，在请教了某叉院大神之后，他告诉我随机生成点时，优化过的暴力是期望线性的。

在$n = 10^8$时，我的证明中给出的最近距离上界约为$3 * 10^5$，但在实际运行中，平均最近距离只有约10^2。因此$O(n\sqrt{n})$的上界比较宽松，可以降低到$O(n)$。加上排序的时间复杂度，算法三的时间复杂度应为$\Theta(n \log n)$。因为算法三的常数小于算法四（不需要辅助空间进行归并排序），因此算法三的速度最快。

四、补充材料

[1] Yufeng's 最近点对算法

[2] 非完美最近点对算法时间复杂度的证明