# 平面最近点对

王曦 2021192010

数学与统计学院

算法设计与分析

2023 年 03 月 20 日

# 1. 实验内容

# 实验内容

1. 对平面上给定的 $n$ 个点的坐标, 求距离最短的两点.
2. 随机生成平面上 $n$ 个点的坐标, 用蛮力法求距离最短的两点.
3. 随机生成平面上 $n$ 个点的坐标, 用分治法求距离最短的两点.
4. 分别对 $N = 1\text{e}5 \sim 1\text{e}6$ 统计算法运行时间, 比较理论效率实测效率的差异, 分析和比较蛮力法和分治法的算法效率.
5. 将算法执行的过程用图形界面输出.
6. 补充介绍和分析求解平面最近点对的非分治法、随机化法和期望线性法.

# 2. 实验环境与约定

# 实验环境与约定

- 实验环境: GNU C++17 (O2).
- 约定所有实际运行时间都不包含数据生成和输入的时间.

# 3. 数据生成器

# 数据生成器

- gen.cpp 中用 C++ 的 STL 中的 mt19937 作为随机数生成器, 分别生成平面点的 x 坐标和 y 坐标. 因 mt19937 生成的随机数值域较大, 而两点间的距离是浮点数, 为保证较好的精度, 可将坐标的绝对值限制在 $MAXA = 1e5$ 范围内. 注意要先将生成的随机数存放到变量中后再对 $MAXA$ 取模, 直接写作 $rnd()\%MAXA$ 会出现生成的坐标都为非负数的现象.
- 上述过程有极小的概率出现两个重合的点, 可忽略这种数据, 因为这可作为对算法边界情况的测试.
- 生成所有测试数据后保存在文件中, 以保证各算法使用的数据相同, 避免因数据不同产生较大的误差.

# 4. 蛮力法求平面最近点对

# 蛮力法求平面最近点对

- 时间复杂度: $O(n^2)$.
- **演示**: 在 $n = 10$ 且 $|x|, |y| \le 10$ 的小数据上验证正确性.

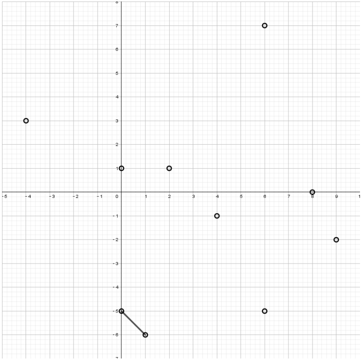

Figure 1: 小数据示意图

# 实际运行效率与理论运行效率

| 数据规模 $n$ | 100000 | 300000 | 500000 | 700000 | 1000000 |
|---|---|---|---|---|---|
| 实际运行时间 (ms) | 42047 | 392638 | 1069330 | 2072129 | 4257908 |
| 理论运行时间 (ms) | 42773.2 | 384958.8 | 1069330 | 2095886.8 | 4277320.0 |
| 误差 | $-1.73\%$ | $1.96\%$ | $0.0\%$ | $-1.15\%$ | $-0.45\%$ |

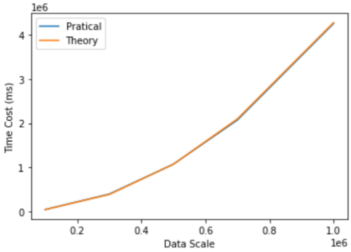


Figure 2: 蛮力法的实际运行时间与理论运行时间的关系

# 解释

- 由图象知: 蛮力法的时间复杂度为 $O(n^2)$.
- 实际运行中, 即使是 $n = 1\text{e}5$ 的最小数据, 开启 O2 优化, 也需要相当的时间才能计算出结果. 这表明: 在平面最近点对问题中, 蛮力法直观, 但效率不优.

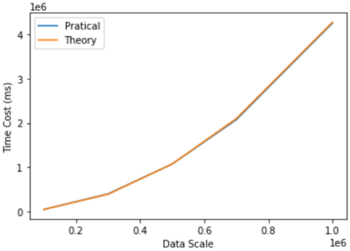


Figure 3: 蛮力法的实际运行时间与理论运行时间的关系

# 5. 分治法求平面最近点对

# 分治法求平面最近点对

- 时间复杂度: $O(n \log n)$.
- **演示**: 在 $n = 10$ 且 $|x|, |y| \leq 10$ 的小数据上验证正确性.
- 对比 $O(n \log n)$ 和 $O(n \log^2 n)$ 的两种写法.

# 实际运行效率与理论运行效率

| 数据规模 $n$ | 100000 | 300000 | 500000 | 700000 | 1000000 |
|---|---|---|---|---|---|
| 实际运行时间 (ms) | 54 | 171 | 329 | 433 | 601 |
| 理论运行时间 (ms) | 57.7 | 189.7 | 329.0 | 472.4 | 692.8 |
| 误差 | $-6.85\%$ | $-10.94\%$ | $0.0\%$ | $-9.10\%$ | $-15.27\%$ |

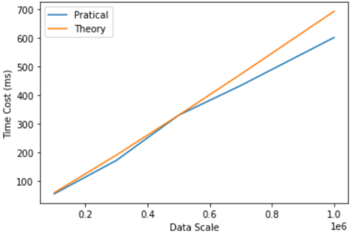


Figure 4: 分治法的实际运行时间与理论运行时间的关系

# 解释

- $n$ 较小时, 实际运行效率与理论时间复杂度 $O(n\log n)$ 接近.
- n 较大时, 实际运行时间低于理论运行时间, 这是因为点数 n 增大但值域未增大时, 一些区域的点会变得更加密集, 则合并时在区域 $[points[mid].x - dis, points[mid].x + dis]$ 中的点较少, 进而合并时需检查的点的对数较少, 故实际运行效率更高.

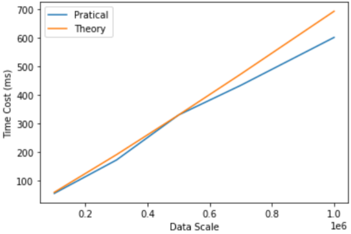


Figure 5: 分治法的实际运行时间与理论运行时间的关系

# 6. 非分治法求平面最近点对

# 非分治法求平面最近点对

- 思路: 类似于统计序列的思想, 对每个点, 将它和它左边的所有元素的贡献加入答案中. 具体地, 将所有点以 $x$ 坐标为第一关键字、以 $y$ 坐标为第二关键字非降序排列后, 将点逐个加入集合中. 具体地, 维护一个以 $y$ 坐标为第一关键字、以 $x$ 坐标为第二关键字的 multiset 和当前的最优解 $ans$. 对每个点 i, 做如下操作:

(1) 因集合以 $y$ 为第一关键字, 则集合中满足 $x_i - x_j \geq dis$ 的点 $j$ 显然不是最优解, 删除即可.

(2) 对集合中满足 $|x_i - x_j| < dis$ 的点 $j$, 暴力更新答案.

(3) 将点 $i$ 加入集合中.

# 非分治法求平面最近点对

- 用洛谷的题目”P1429 平面最近点对（加强版）”(https://www.luogu.com.cn/problem/P1429) 验证正确性.


Figure 6: 用洛谷验证非分治法的正确性

# 实际运行效率与理论运行效率

| 数据规模 $n$ | 100000 | 300000 | 500000 | 700000 | 1000000 |
|---|---|---|---|---|---|
| 实际运行时间 (ms) | 12 | 35 | 57 | 81 | 117 |
| 理论运行时间 (ms) | 10.0 | 32.9 | 57.0 | 81.8 | 120.0 |
| 误差 | 16.7% | 6.00% | 0.00% | −0.99% | −2.56% |

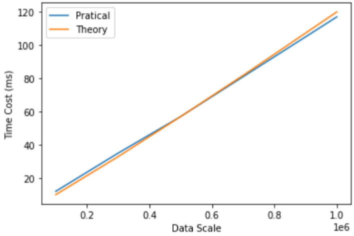


Figure 7: Multiset 法的实际运行时间与理论运行时间的关系

# 解释

- 由图象知: Multiset 法的时间复杂度为 $O(n\log n)$.
- 与分治法不同, Multiset 法在运行过程中会逐渐删除集合中的点, 即使初始点数 $n$ 较大, 经过几轮迭代后也会匀速减小至与 $n$ 较小时相当的规模, 故 $n$ 较小和较大时, 实际运行时间都与理论运行时间接近.
- $n$ 较小时误差较大可能是 multiset 本身的常数导致的, $n$ 较大时, multiset 本身的常数对实际运行时间的影响被冲淡.

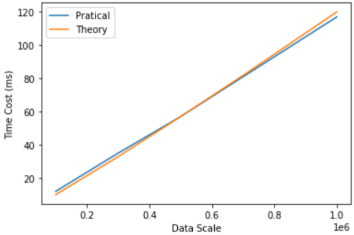


Figure 8: Multiset 法的实际运行时间与理论运行时间的关系

# 7. 随机化法求平面最近点对

# 随机化法求平面最近点对

- 思路: 将所有点绕原点随机旋转同一角度后按 $x \cdot y$ 非降序排列. 可以证明: 随机旋转后, 最优解的两点在数组中相距不远. 具体地, 只需取每个点之前的 $k = 50$ 个点更新答案.
- 用洛谷的题目"P7883 平面最近点对（加强加强版）" (https://www.luogu.com.cn/problem/P7883) 验证正确性.


Figure 9: 用洛谷验证随机化法的正确性

# 实际运行效率与理论运行效率

| 数据规模 $n$ | 100000 | 300000 | 500000 | 700000 | 1000000 |
|---|---|---|---|---|---|
| 实际运行时间 (ms) | 132 | 394 | 663 | 921 | 1334 |
| 理论运行时间 (ms) | 128.1 | 393.5 | 663.0 | 934.7 | 1345.2 |
| 误差 | 2.95% | 0.13% | 0.00% | $-1.49\%$ | $-0.84\%$ |

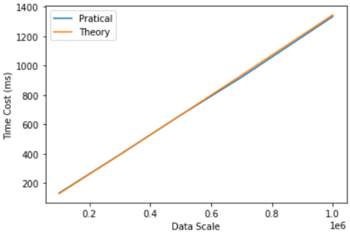


Figure 10: 随机化法的实际运行时间与理论运行时间的关系

# 解释

- 由图像知: 随机化法的时间复杂度为 $O(n\log n + k\cdot n)$.
- 随机化算法的时间复杂度是稳定的 $O(n\log n + k\cdot n)$, 故 $n$ 较小或较大时实际运行时间都与理论运行时间接近.

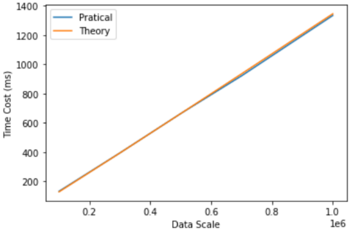


Figure 11: 随机化法的实际运行时间与理论运行时间的关系

# 8. 期望线性法求平面最近点对

# 期望线性法求平面最近点对

- 思路:

(1) 循环下面的过程直至删完所有点:

(i) 随机选一个点, 求它到其他所有点的最短距离 $d$.

(ii) 将所有点划分到 $l = \left\lfloor \frac{d}{3} \right\rfloor$ 的网格中, 如 $\left(\left\lfloor \frac{x}{l} \right\rfloor, \left\lfloor \frac{y}{l} \right\rfloor\right)$.

(iii) 删除九宫格内的孤立点, 则这些点的最近点对距离 $\geq \frac{2\sqrt{2}}{3}d$, 其中 $\frac{2\sqrt{2}}{3} < 1$.

(2) 取最后一个 $d$, 将所有点划分到 $\left(\left\lfloor \frac{x}{d} \right\rfloor, \left\lfloor \frac{y}{d} \right\rfloor\right)$ 的网格中, 暴力求九宫格内的答案.

# 期望线性法求平面最近点对

- 用洛谷的题目"P1429 平面最近点对（加强版）"(https://www.luogu.com.cn/problem/P1429) 验证正确性.


Figure 12: 用洛谷验证期望线性法的正确性

# 实际运行效率与理论运行效率

| 数据规模 $n$ | 100000 | 300000 | 500000 | 700000 | 1000000 |
|---|---|---|---|---|---|
| 实际运行时间 (ms) | 368 | 421 | 872 | 1480 | 3028 |
| 理论运行时间 (ms) | 48.1 | 347.7 | 872.0 | 1597.9 | 3036.5 |
| 误差 | 86.9% | 17.4% | 0.0% | −8.0% | −0.3% |

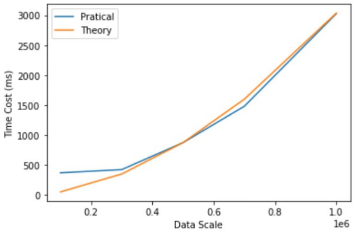


Figure 13: 期望线性法的实际运行时间与理论运行时间的关系

# 解释

- 由图象知: $n$ 较大时, 实际运行效率近似于 $O(n)$; $n$ 较小时, 实际运行效率低于理论时间复杂度.
- $n$ 较小时, 实际运行时间高于理论运行时间, 这是因为 $n$ 较小时, 各点间相对稀疏. 随机选一个点, 它到其他所有点的最短距离 $d$ 较大, 此时所有点划分到的网格 $\left(\left\lfloor\frac{x}{l}\right\rfloor, \left\lfloor\frac{y}{l}\right\rfloor\right)$ 较大, 需检查的点对较多; $n$ 较大时, 各点间相对密集, 此时所有点划分到的网格 $\left(\left\lfloor\frac{x}{l}\right\rfloor, \left\lfloor\frac{y}{l}\right\rfloor\right)$ 较小, 需检查的点对较少.

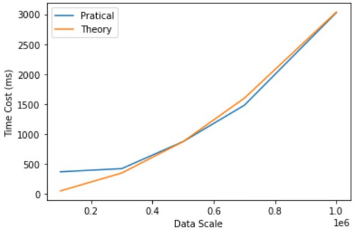


Figure 14: 期望线性法的实际运行时间与理论运行时间的关系

# 解释

- 该算法本身是期望线性的，但实现中采用了 hash_table 和多重循环，带来较大的常数，导致实际运行时间的增长类似于抛物线.
- 下面对实际运行时间取 log，图象如下图所示:

| 数据规模 $n$ | 100000 | 300000 | 500000 | 700000 | 1000000 |
|---|---|---|---|---|---|
| log 实际运行时间 (log ms) | 8.52 | 8.71 | 9.76 | 10.53 | 11.56 |

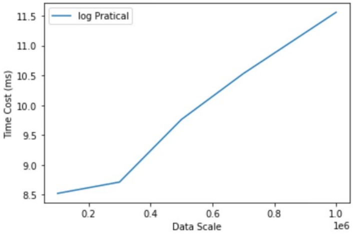


Figure 15: 期望线性法的 log 实际运行时间与理论运行时间的关系

# 解释

- 由图象知: 随数据规模增大, 折线的斜率先陡增, 再稍降低后趋于稳定.
- 随着输入点数 $n$ 增大, 哈希表中的元素增多. 因坐标范围不增, 则可能出现重合的点, 增大哈希冲突的概率, 使得哈希表单次查询的时间复杂度可能退化为 $O(n)$. 但数据规模继续增大时, 单词查询的时间复杂度趋于稳定, 此时哈希表的均摊时间复杂度占主导, 故折线斜率稍降低后趋于稳定.

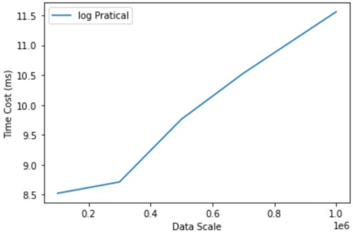


Figure 16: 期望线性法的 log 实际运行时间与理论运行时间的关系

# 9. 不同算法的对比

# 蛮力法

- 性能上, 蛮力法 $O(n^2)$ 的时间复杂度远高于其他四个算法, 实际运行时间也远高于其他四个算法的实际运行时间.
- 蛮力法是涉及到的 5 个算法中最为简单形象的算法.

# 非蛮力法的对比

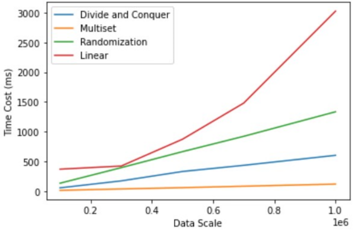


Figure 17: 非蛮力法的实际运行时间的对比

- 分治法、非分治法 (Multiset 法)、随机化法、期望线性法的实际运行时间的对比如上图所示.
- 整体上, Multiset 法的性能最优, 期望线性法的性能最差 (实现原因).

# 非蛮力法的对比

- 分治法因采用递归实现, 故运行效率低于 Multiset 法.
- 随机化法的时间复杂度为 $O(n \log n + k \cdot n)$, 而分治法的时间复杂度为 $O(n \log n)$, 由图象知: 随机化法的实际运行时间稍高于分治法, 这与时间复杂度是对应的. 具体地, 虽然随机化法的时间复杂度 $O(n \log n + k \cdot n)$ 中 $O(k \cdot n)$ 不是主项, 但实现时取 $k = 50$, 在本次实验的数据范围上, 有 $k > \log n$. 这表明: 虽然时间复杂度 $O(n \log n + k \cdot n)$ 对固定的 $k$ 可视为 $O(n \log n)$, 但当非主项的规模与主项相当, 甚至有可能超过主项时, 它对实际运行效率的影响不可忽视.

# 非蛮力法的对比

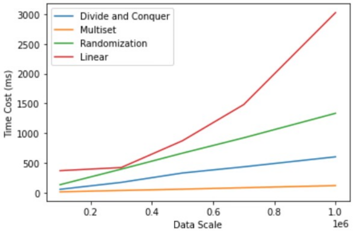


Figure 18: 非蛮力法的实际运行时间的对比

- 期望线性法的算法本身是期望线性的, 但因实现较为繁琐, 且实现用到了 hash_table 和多重循环等, 导致常数较大. 这表明: 当时间复杂度的常数较大时, 它对实际运行效率的影响不要忽视.
- 直观程度上, 非蛮力法都不如蛮力法直观. 在非蛮力法中, Multiset 法最直观且效率最高, 随机化法依赖于数学直觉, 而分治法和期望线性法因存在划分网格和递归的过程, 最不直观.

谢　谢!

Thank you!