实验报告

软 31 万璐瑶 2023012127

2025 年 3 月 10 日

1 实验要求

实现求平面上最近点对的复杂度为 Θ(nlgn) 的算法，要求：

- 能够随机生成大量平面点（要求可达到一千万个点）；
- 有方便测试的界面或者命令行，能够读入文本文件（含平面点位置信息）作为输入。
- 分析比较在不同输入规模情况下 Θ(n2) 和 Θ(nlgn) 算法的实际运行时间，提交实验报告。

代码运行方式以及运行参数详见 readme.txt 文件。

2 实验环境

- 操作系统：Windows 11
- 编程语言：C++
- 编译器：g++
- IDE：Visual Studio Code

3 算法分析

在算法实现部分，我们实现了两种算法，分别是 $O(n \lg n)$ 的分治算法和 $O(n^2)$ 的暴力算法。分治算法的基本思想是将平面点集分为两部分，分别求解两部分的最近点对，然后再求解跨越两部分的最近点对，复杂度为 $O(n \lg n)$。

暴力算法的基本思想是枚举遍历所有点对，找出距离最小的点对，复杂度为 $O(n^2)$。

分治算法中先对输入的点对进行归并排序，保证 y 值升序排序，`find_closest_pair2`中将点集分为两部分，分别求解两部分的最近点对，然后求解跨越两部分的最近点对。跨越两部分的最近点对的求解方法是：由于所有点已经按照 y 值升序排序，由课上所讲内容得知，在 δd 范围内，对每个点最多只需计算与其后面的 7 个点的距离，便可这个点与所有点的找出最小的距离，因此合并操作的复杂度为 $O(n)$。

又主定理可知，分治算法的时间复杂度为 $O(n \lg n)$。

Listing 1 找最近点对

```
int find_closest_pair2(vector<pairr> pairs) { // 优化后的算法
    if (pairs.size() == 2) {
        return distance(pairs[0], pairs[1]); // 两个点直接求距离
    } else if (pairs.size() == 1) {
        return INT_MAX; // 一个点无法求最近点对距离，设为最大值
    }
    int mid = pairs.size() / 2;
    vector<pairr> left_pairs, right_pairs;
    for(int i = 0; i < mid; i++) {
        left_pairs.push_back(pairs[i]);
    }
    for(int i = mid; i < pairs.size(); i++) {
        right_pairs.push_back(pairs[i]);
    }
    int d1 = find_closest_pair2(left_pairs);
    int d2 = find_closest_pair2(right_pairs); // 分别递归求左右两边的最近点对
    // 合并两侧求最近点对
    int d = min(d1, d2);
    int minn = d;
    for(int i = 0; i < pairs.size(); i++) {
        for (int j = 1; j < 8; j++) {
            if (i + j < pairs.size()) {
                if ((pairs[i].x - pairs[i + j].x) * (pairs[i].x - pairs[i + j].x) < d) {
                    int dis = distance(pairs[i], pairs[i + j]);
                    if (dis < minn) {
                        minn = dis;
                    }
                }
            }
        }
    }
    return minn;
}
```

4 实验设计

可见分治算法的复杂度为 $O(n \lg n)$，暴力算法的复杂度为 $O(n^2)$，因此我们设计了两种算法，分别求解最近点对，然后比较两种算法的运行时间。再调用 <chrono> 库中的函数，计算两种算法的运行时间，此时 'duration'的单位为纳秒，目的是更好地精确比较在数据规模较小的时候两种算法的时间差。

同时通过读取输入数的方式，设计了两种读取数据的方法，用户可通过文件输入，自带的三个.txt 文件分别为 100 个点，500 个点和 1000 个点，用户也可自行输入文件，文件格式为每行一个点，x 和 y 坐标用空格隔开，且首行表示该文件内的点数。同时，我们设计了一个随机生成平面点的函数`random()`，用户也可以通过输入 2 选择随机生成数据，生成随机数调用的函数是`rand()`, 在 C 语言中，`rand()` 函数生成的随机数范围是从 0 到 `RAND_MAX`。`RAND_MAX` 是一个定义在 `<stdlib.h>` 中的宏，表示 `rand()` 能生成的最大值。通常情况下，`RAND_MAX` 的值为 32767，故生成的点的 x 和 y 坐标均为 0 到 32767 之间的随机数，生成的点的数量可由用户

输入。经测试，改函数生成的点的数量可达到一千万个点。

生成数据后通过命令行读取用户命令调用两种算法，便于测试时间。

5 结果分析

表 1: 时间复杂度分析

n	$O(n\lg n)$ 算法	$O(n^2)$ 算法
100	0	0
500	1556800	2683900
1000	6508600	10882200
5000	12133300	258947300
10000	29822400	1033896700
50000	161094400	22109884700
100000	335465000	/

可见复杂度为 $O(n\lg n)$ 的算法基本随 n 的增长呈线性变化，而复杂度为 $O(n^2)$ 的算法随 n 的增长呈指数级变化。当 $n > 50000$ 时，$O(n^2)$ 算法的运行时间长于一分钟，是低效的算法。

在数据较小时 ($n < 100$)，两种算法的运行时间差异不大，但随着数据量的增大，$O(n\lg n)$ 算法的优势逐渐显现，因此在大数据量下，$O(n\lg n)$ 算法是更优的选择。