

# 平面最近点对

## 概述

给定  $n$  个二维平面上的点，求一组欧几里得距离最近的点对。

下面我们介绍一种时间复杂度为  $O(n \log n)$  的分治算法来解决这个问题。该算法在 1975 年由 [Franco P. Preparata](https://en.wikipedia.org/wiki/Franco_P._Preparata) [https://en.wikipedia.org/wiki/Franco\_P.\_Preparata] 提出，Preparata 和 [Michael Ian Shamos](https://en.wikipedia.org/wiki/Michael_Ian_Shamos) [https://en.wikipedia.org/wiki/Michael\_Ian\_Shamos] 证明了该算法在决策树模型下是最优的。

## 算法

与常规的分治算法一样，我们将这个有  $n$  个点的集合拆分成两个大小相同的集合  $S_1, S_2$ ，并不断递归下去。但是我们遇到了一个难题：如何合并？即如何求出一个点在  $S_1$  中，另一个点在  $S_2$  中的最近点对？这里我们先假设合并操作的时间复杂度为  $O(n)$ ，可知算法总复杂度为  $T(n) = 2T(\frac{n}{2}) + O(n) = O(n \log n)$ 。

我们先将所有点按照  $x_i$  为第一关键字、 $y_i$  为第二关键字排序，并以点  $p_m (m = \lfloor \frac{n}{2} \rfloor)$  为分界点，拆分点集为  $A_1, A_2$ ：

$$A_1 = \{p_i \mid i = 0 \dots m\}$$
$$A_2 = \{p_i \mid i = m + 1 \dots n - 1\}$$

并递归下去，求出两点集各自内部的最近点对，设距离为  $h_1, h_2$ ，取较小值设为  $h$ 。

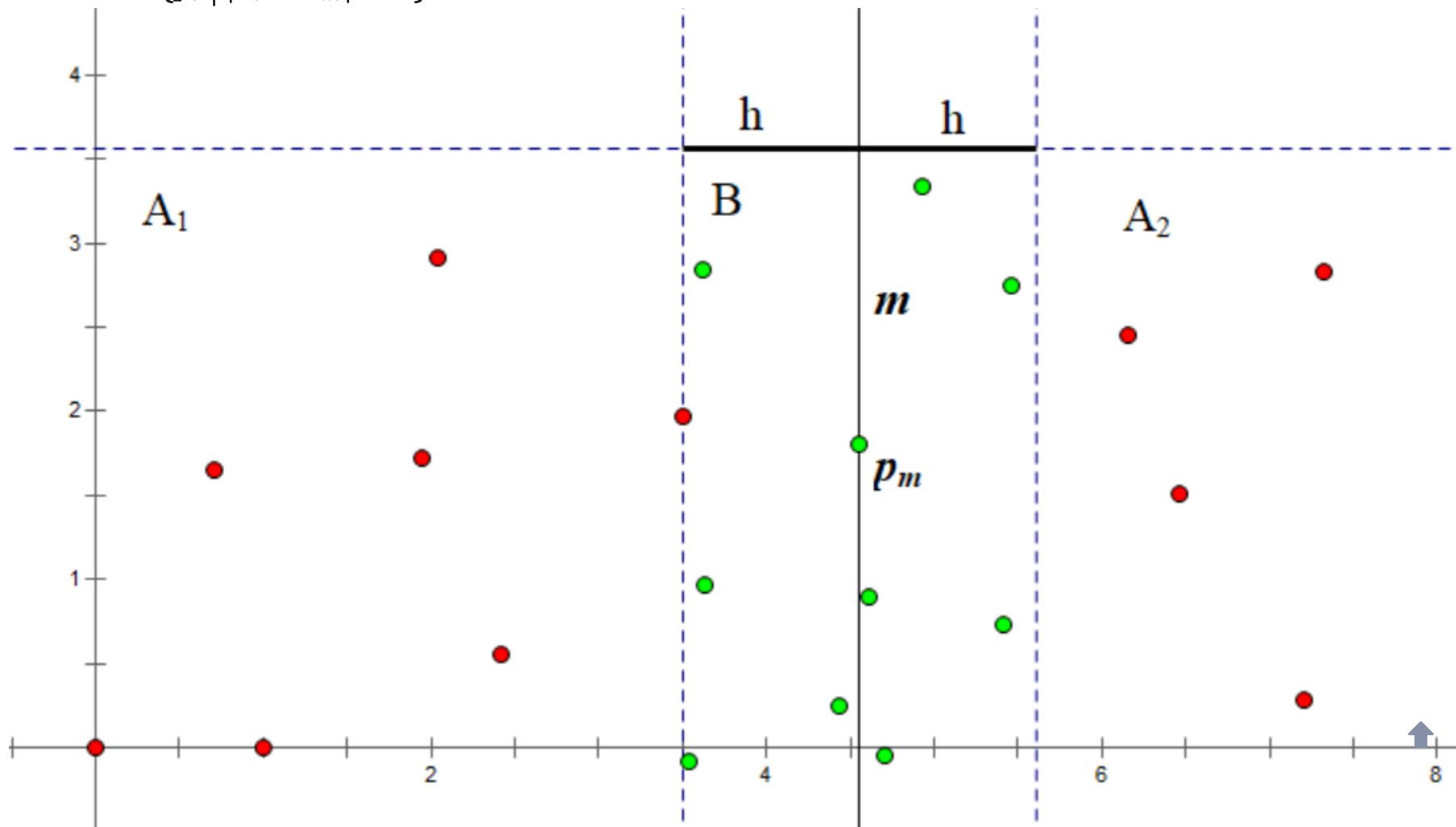
现在该合并了！我们试图找到这样的一组点对，其中一个属于  $A_1$ ，另一个属于  $A_2$ ，且二者距离小于  $h$ 。因此我们将所有横坐标与  $x_m$  的差小于  $h$  的点放入集合  $B$ ：

$$B = \{p_i \mid |x_i - x_m| < h\}$$

结合图像，直线  $m$  将点分成了两部分。 $m$  左侧为  $A_1$  点集，右侧为为  $A_2$  点集。



再根据  $B = \{p_i \mid |x_i - x_m| < h\}$  规则，得到绿色点组成的  $B$  点集。

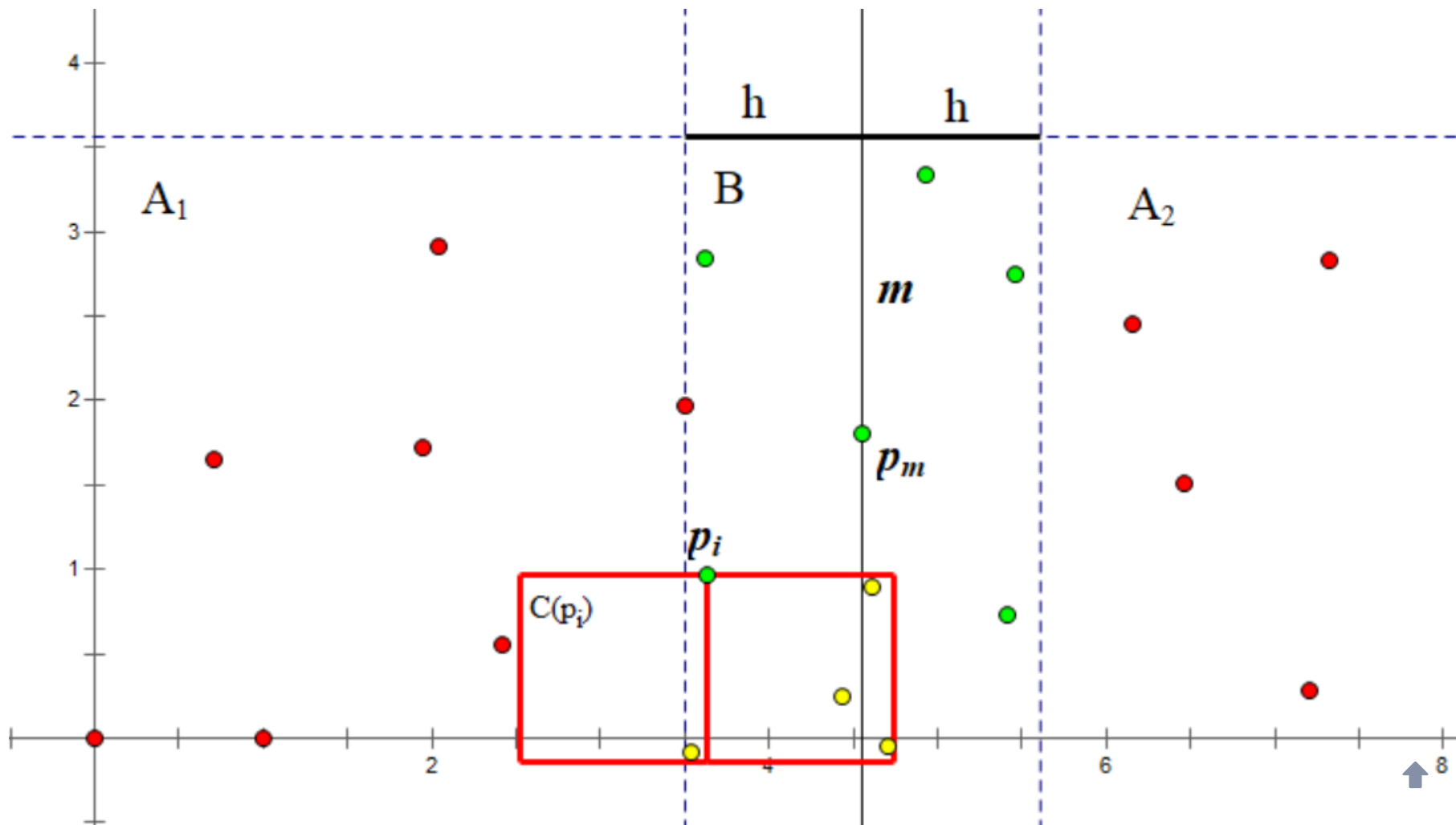


对于  $B$  中的每个点  $p_i$ ，我们当前目标是找到一个同样在  $B$  中、且到其距离小于  $h$  的点。为了避免两个点之间互相考虑，我们只考虑那些纵坐标小于  $y_i$  的点。显然对于一个合法的点  $p_j$ ， $y_i - y_j$  必须小于  $h$ 。于是我们获得了一个集合  $C(p_i)$ ：

$$C(p_i) = \{p_j \mid p_j \in B, y_i - h < y_j \leq y_i\}$$

在点集  $B$  中选一点  $p_i$ ，根据  $C(p_i) = \{p_j \mid p_j \in B, y_i - h < y_j \leq y_i\}$  的规则，得到了由红色方框内的黄色点组成的  $C$  点集。





如果我们将  $B$  中的点按照  $y_i$  排序,  $C(p_i)$  将很容易得到, 即紧邻  $p_i$  的连续几个点。

由此我们得到了合并的步骤:

1. 构建集合  $B$ 。
2. 将  $B$  中的点按照  $y_i$  排序。通常做法是  $O(n \log n)$ ，但是我们可以改变策略优化到  $O(n)$ （下文讲解）。
3. 对于每个  $p_i \in B$  考虑  $p_j \in C(p_i)$ ，对于每对  $(p_i, p_j)$  计算距离并更新答案（当前所处集合的最近点对）。

注意到我们上文提到了两次排序，因为点坐标全程不变，第一次排序可以只在分治开始前进行一次。我们令每次递归返回当前点集按  $y_i$  排序的结果，对于第二次排序，上层直接使用下层的两个分别排序过的点集归并即可。

似乎这个算法仍然不优， $|C(p_i)|$  将处于  $O(n)$  数量级，导致总复杂度不对。其实不然，其最大大小为 7，我们给出它的证明：

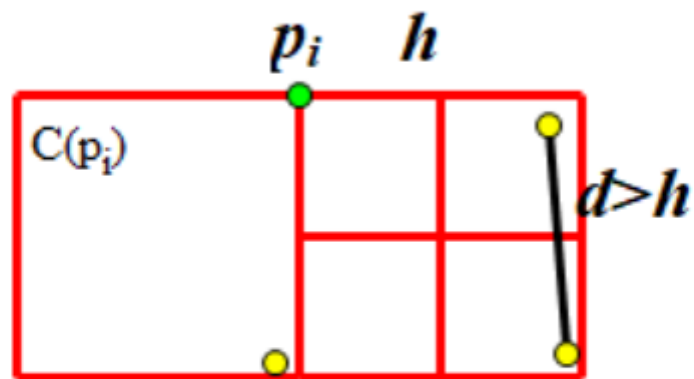
## 复杂度证明

我们已经了解到， $C(p_i)$  中的所有点的纵坐标都在  $(y_i - h, y_i]$  范围内；且  $C(p_i)$  中的所有点，和  $p_i$  本身，横坐标都在  $(x_m - h, x_m + h)$  范围内。这构成了一个  $2h \times h$  的矩形。

我们再将这个矩形拆分为两个  $h \times h$  的正方形，不考虑  $p_i$ ，其中一个正方形中的点为  $C(p_i) \cap A_1$ ，另一个为  $C(p_i) \cap A_2$ ，且两个正方形内的任意两点间距离大于  $h$ 。（因为它们来自同一下层递归）

我们将一个  $h \times h$  的正方形拆分为四个  $\frac{h}{2} \times \frac{h}{2}$  的小正方形。可以发现，每个小正方形中最多有 1 个点：因为该小正方形中任意两点最大距离是对角线的长度，即  $\frac{h}{\sqrt{2}}$ ，该数小于  $h$ 。






由此，每个正方形中最多有 **4** 个点，矩形中最多有 **8** 个点，去掉  $p_i$  本身， $\max(C(p_i)) = 7$ 。

## 实现

我们使用一个结构体来存储点，并定义用于排序的函数对象：

 结构体定义

```
1 struct pt {
2     int x, y, id;
3 };
4
5 struct cmp_x {
6     bool operator()(const pt& a, const pt& b) const {
7         return a.x < b.x || (a.x == b.x && a.y < b.y);
8     }
9 };
10
11 struct cmp_y {
12     bool operator()(const pt& a, const pt& b) const { return a.y < b.y; }
13 };
14
15 int n;
16 vector<pt> a;
```

为了方便实现递归，我们引入 `upd_ans()` 辅助函数来计算两点间距离并尝试更新答案：

#### 答案更新函数

```
1 double mindist;
2 int ansa, ansb;
3
4 inline void upd_ans(const pt& a, const pt& b) {
5     double dist =
6         sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y) + .0);
```



```

7   if (dist < mindist) mindist = dist, ansa = a.id, ansb = b.id;
8   }

```

下面是递归本身：假设在调用前  $a[]$  已按  $x_i$  排序。如果  $r - l$  过小，使用暴力算法计算  $h$ ，终止递归。

我们使用 `std::merge()` 来执行归并排序，并创建辅助缓冲区  $t[]$ ， $B$  存储在其中。

#### 主体函数

```

1 void rec(int l, int r) {
2     if (r - l <= 3) {
3         for (int i = l; i <= r; ++i)
4             for (int j = i + 1; j <= r; ++j) upd_ans(a[i], a[j]);
5         sort(a + l, a + r + 1, &cmp_y);
6         return;
7     }
8
9     int m = (l + r) >> 1;
10    int midx = a[m].x;
11    rec(l, m), rec(m + 1, r);
12    static pt t[MAXN];
13    merge(a + l, a + m + 1, a + m + 1, a + r + 1, t, &cmp_y);
14    copy(t, t + r - l + 1, a + l);
15
16    int tsz = 0;
17    for (int i = l; i <= r; ++i)
18        if (abs(a[i].x - midx) < mindist) {
19            for (int j = tsz - 1; j >= 0 && a[i].y - t[j].y < mindist; --j)

```

```

20         upd_ans(a[i], t[j]);
21         t[tsz++] = a[i];
22     }
23 }

```

在主函数中，这样开始递归即可：

#### 调用接口

```

1  sort(a, a + n, &cmp_x);
2  mindist = 1E20;
3  rec(0, n - 1);

```

## 推广：平面最小周长三角形

上述算法有趣地推广到这个问题：在给定的一组点中，选择三个点，使得它们两两的距离之和最小。

算法大体保持不变，每次尝试找到一个比当前答案周长  $d$  更小的三角形，将所有横坐标与  $x_m$  的差小于  $\frac{d}{2}$  的点放入集合  $B$ ，尝试更新答案。（周长为  $d$  的三角形的最长边小于  $\frac{d}{2}$ ）

## 非分治算法

其实，除了上面提到的分治算法，还有另一种时间复杂度同样是  $O(n \log n)$  的非分治算法。

我们可以考虑一种常见的统计序列的思想：对于每一个元素，将它和它的左边所有元素的贡献加入到答案中。平面最近点对问题同样可以使用这种思想。

具体地，我们把所有点按照  $x_i$  为第一关键字、 $y_i$  为第二关键字排序，并建立一个以  $y_i$  为第一关键字、 $x_i$  为第二关键字排序的 multiset。对于每一个位置  $i$ ，我们执行以下操作：

1. 将所有满足  $x_i - x_j \geq d$  的点从集合中删除。它们不会再对答案有贡献。
2. 对于集合内满足  $|y_i - y_j| < d$  的所有点，统计它们和  $p_i$  的距离。
3. 将  $p_i$  插入到集合中。

由于每个点最多会被插入和删除一次，所以插入和删除点的时间复杂度为  $O(n \log n)$ ，而统计答案部分的时间复杂度证明与分治算法的时间复杂度证明方法类似，读者不妨一试。

#### 参考代码

```
1  #include <algorithm>
2  #include <cmath>
3  #include <cstdio>
4  #include <set>
5  const int N = 200005;
6  int n;
7  double ans = 1e20;
8  struct point {
9      double x, y;
10     point(double x = 0, double y = 0) : x(x), y(y) {}
```

```
11 };
12
13 struct cmp_x {
14     bool operator()(const point &a, const point &b) const {
15         return a.x < b.x || (a.x == b.x && a.y < b.y);
16     }
17 };
18
19 struct cmp_y {
20     bool operator()(const point &a, const point &b) const { return a.y < b.y; }
21 };
22
23 inline void upd_ans(const point &a, const point &b) {
24     double dist = sqrt(pow((a.x - b.x), 2) + pow((a.y - b.y), 2));
25     if (ans > dist) ans = dist;
26 }
27
28 point a[N];
29 std::multiset<point, cmp_y> s;
30
31 int main() {
32     scanf("%d", &n);
33     for (int i = 0; i < n; i++) scanf("%lf%lf", &a[i].x, &a[i].y);
34     std::sort(a, a + n, cmp_x());
35     for (int i = 0, l = 0; i < n; i++) {
36         while (l < i && a[i].x - a[l].x >= ans) s.erase(s.find(a[l++]));
37         for (auto it = s.lower_bound(point(a[i].x, a[i].y - ans));
38              it != s.end() && it->y - a[i].y < ans; it++)
39             upd_ans(*it, a[i]);
40         s.insert(a[i]);
41     }
```



```
42     printf("%.4lf", ans);
43     return 0;
44 }
```

## 习题

- UVA 10245 "The Closest Pair Problem"[难度：低] [[https://uva.onlinejudge.org/index.php?option=onlinejudge&page=show\\_problem&problem=1186](https://uva.onlinejudge.org/index.php?option=onlinejudge&page=show_problem&problem=1186)]
- SPOJ #8725 CLOPPAIR "Closest Point Pair"[难度：低] [<https://www.spoj.com/problems/CLOPPAIR/>]
- CODEFORCES Team Olympiad Saratov - 2011 "Minimum amount"[难度：中] [<http://codeforces.com/contest/120/problem/J>]
- SPOJ #7029 CLOSEST "Closest Triple"[难度：中] [<https://www.spoj.com/problems/CLOSEST/>]
- Google Code Jam 2009 Final "Min Perimeter"[难度：中] [<https://codingcompetitions.withgoogle.com/codejam/round/0000000000432ad5/0000000000433195>]

---

## 参考资料与拓展阅读



本页面中的分治算法部分主要译自博文 [Нахождение пары ближайших точек](http://e-maxx.ru/algorithm/nearest_points) [[http://e-maxx.ru/algorithm/nearest\\_points](http://e-maxx.ru/algorithm/nearest_points)] 与其英文翻译版 [Finding the nearest pair of points](https://github.com/e-maxx-eng/e-maxx-) [<https://github.com/e-maxx-eng/e-maxx->]

`eng/blob/master/src/geometry/nearest_points.md`。其中俄文版版权协议为 **Public Domain + Leave a Link**；英文版版权协议为 **CC-BY-SA 4.0**。

知乎专栏：计算几何 - 最近点对问题 [https://zhuanlan.zhihu.com/p/74905629]

🔗 本页面最近更新：2021/9/9 22:43:56，[更新历史](https://github.com/OI-wiki/OI-wiki/commits/master/docs/geometry/nearest-points.md) [https://github.com/OI-wiki/OI-wiki/commits/master/docs/geometry/nearest-points.md]

✎ 发现错误？想一起完善？在 [GitHub](https://oi-wiki.org/edit-landing/?ref=/geometry/nearest-points.md) 上编辑此页！ [https://oi-wiki.org/edit-landing/?ref=/geometry/nearest-points.md]

👤 本页面贡献者：[Xeonacid](https://github.com/Xeonacid) [https://github.com/Xeonacid]，[StudyingFather](https://github.com/StudyingFather) [https://github.com/StudyingFather]，[Enter-tainer](https://github.com/Enter-tainer) [https://github.com/Enter-tainer]，[ShaoChenHeng](https://github.com/ShaoChenHeng) [https://github.com/ShaoChenHeng]，[sswhy](https://github.com/sswhy) [https://github.com/sswhy]，[countercurrent-time](https://github.com/countercurrent-time) [https://github.com/countercurrent-time]，[lr1d](https://github.com/lr1d) [https://github.com/lr1d]

© 本页面的全部内容在 [CC BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/deed.zh) [https://creativecommons.org/licenses/by-sa/4.0/deed.zh] 和 [SATA](https://github.com/zTrix/sata-license) [https://github.com/zTrix/sata-license] 协议之条款下提供，附加条款亦可能应用

## 评论



[5](https://github.com/OI-wiki/gitment/issues/283) [https://github.com/OI-wiki/gitment/issues/283] comments

Anonymous ▾