从零开始的计算几何

Namomo Winter Camp 2024

杨宗翰 (fstqwq)

逆命 - 上海交通大学
Nemesis - Shanghai Jiao Tong University

January 24, 2024

你是？

我是？
擅长在正式赛乱炸，并将队伍罚时带上新的高度。

- EC Final 2019 H +7 03:06
- Shenyang 2020 A +8 04:39
- EC Final 2021 D +16 04:50
- World Finals Dhaka B +6 04:37
- EC Final 2023 D +13 04:35 [1]

所以今天就来讲一些炸了有意义的题。

[1] wls:「现在有很多队在没有意义的炸 D 题，他们以为改改他们那个区间的大小就能过了，那不可能的，他们炸到结束都不可能让他们过，过了算我输。」

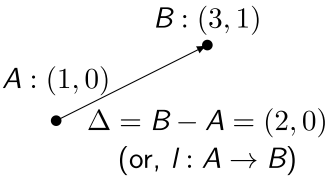
Overview

1. **计算几何**
   - 基础
   - 凸包
   - 凸包上二分
   - 闵可夫斯基和
   - 半平面交
   - 简单多边形
   - 三分
2. **比赛策略**

今天的目标是:

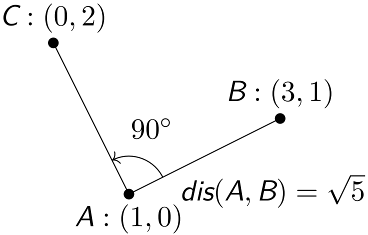
1. 大家都不掉线!
2. 知道什么是叉积!
3. 会做济南几何题 Almost Convex!

点与线的表示



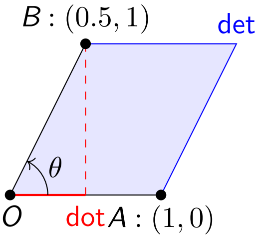

```cpp
typedef double LD;
struct point {
    LD x, y;
    point operator + (const point &a) const {
        return {x + a.x, y + a.y};
    } // 减法类似
    point operator * (const LD a) const {
        return {x * a, y * a};
    } // 除法类似
};
struct line {
    point s, t;
};
point A(0, 0), B(2, 1), Delta = B - A;
line l(A, B);
```

距离与旋转




```cpp
LD sqr (LD x) { return x * x; }
LD dis (const point &a, const point &b) {
    return sqrt(sqr(a.x-b.x) + sqr(a.y-b.y));
}
struct point {
    point rotate (LD t) const {
        return {x * cos(t) - y * sin(t),
                x * sin(t) - y * cos(t)};
    }
    point rot90() const { return {-y, x}; }
    point unit() const {
        return *this / sqrt(sqr(x) + sqr(y));
    }
};
point C = A + (B - A).rot90();
```

点积与叉积



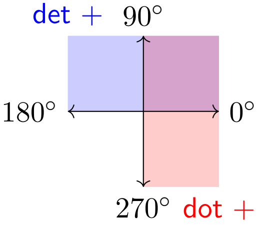
$\text{dot} = |A||B|\cos\theta$ **投影**

$\text{det} = |A||B|\sin\theta$ **面积**

```
LD dot (const point &a, const point &b) {
    return a.x * b.x + a.y * b.y;
}
LD det (const point &a, const point &b) {
    return a.x * b.y - b.x * a.y;
}
```

点积与叉积：方向判断

使用点积与叉积作为工具，我们可以在不计算角度的情况下判断夹角的象限。



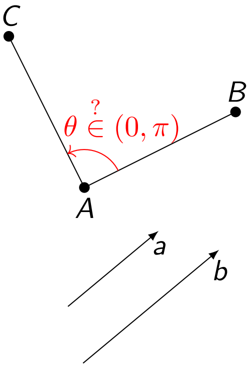
$$\text{dot} = |A||B|\cos\theta \text{ 投影}$$

$$\text{det} = |A||B|\sin\theta \text{ 面积}$$

```
LD dot (const point &a, const point &b) {
      return a.x * b.x + a.y * b.y;
}
LD det (const point &a, const point &b) {
      return a.x * b.y - b.x * a.y;
}
```

点积与叉积：方向判断

使用点积与叉积作为工具，我们可以在不计算角度的情况下判断夹角的象限。



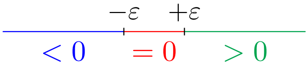

```
bool turn_left (const point &a,
                const point &b,
                const point &c) {
    return sgn (det(b - a, c - a)) > 0;
}
bool same_dir (const line &a, const line &b) {
    return sgn (det(b.t - b.s, a.t - a.s)) == 0
    &&     sgn (dot(b.t - b.s, a.t - a.s)) > 0;
}
```

$$\mathit{normalized}(\vec{a}) \stackrel{?}{=} \mathit{normalized}(\vec{b})$$

浮点数等于零

在使用浮点数表示坐标时，需要使用 epsilon 来实现相等判断。

- 确定 epsilon 取值需要分析题目限制。




```cpp
const LD eps = 1e-8;
int sgn (LD x) {
    return x > eps ? 1 : (x < -eps ? -1 : 0);
}
```

Quiz: 点判等

以下实现点的等于号有潜在问题。问题是什么，如何解决？

- 限制：保证所有不同点之间距离至少为 10^{-5}。

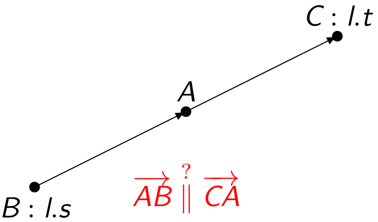
```
const LD eps = 1e-9;
int sgn (LD x) {
    return x > eps ? 1 : (x < 0 ? -1 : 0);
}
LD dot (const point &a, const point &b) {
    return a.x * b.x + a.y * b.y;
}
friend bool operator == (const point &a, const point &b) {
    return sgn (dot (a - b, a - b)) == 0;
}
```

点在线段上

点在线段上有两个要求：

- 点在直线上：叉积判断共线。
- 点在两端点之间：点积判断方向。

注意可能需要处理线段退化的情况：线段退化时直接调用会返回 `true`。



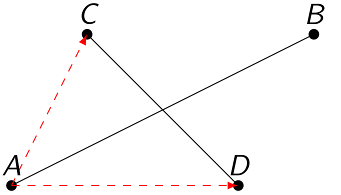

```cpp
bool point_on_segment(const point &a,
                      const line  &l){
    return sgn (det (l.s - a, a - l.t)) == 0
        && sgn (dot (l.s - a, a - l.t)) <= 0;
}
```

线段判交

线段有交分为两种情况：

- 一条线段的端点在另一条线段上。
- 互相严格跨立。

第一种使用 `point_on_segment` 处理，第二种使用叉积判定角度异号。



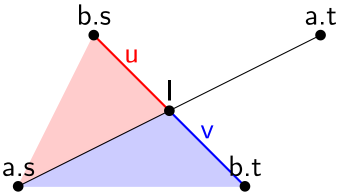
$$\mathrm{sgn}(BAC) \cdot \mathrm{sgn}(BAD) \stackrel{?}{=} -1$$

```
bool two_side(const point &a, const point &b, const line &c) {
    return sgn (det(a - c.s, c.t - c.s))
    * sgn (det(b - c.s, c.t - c.s)) < 0;
}
bool inter_judge(const line &a, const line &b) {
    if (point_on_segment (b.s, a)
     || point_on_segment (b.t, a)
     || point_on_segment (a.s, b)
     || point_on_segment (a.t, b))
        return true;
    return two_side (a.s, a.t, b)
        && two_side (b.s, b.t, a);
}
```

直线求交

使用面积计算等高三角形底边的比例。注意，这里我们使用了除法，会导致精度严重下降。

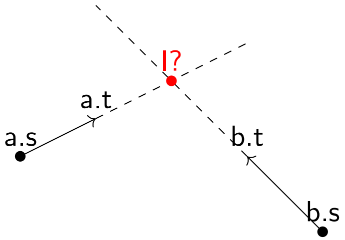
- 通常 epsilon 就是为了克服除法引入的误差——否则大多数计算可用（或等效于）整数。




```cpp
point line_intersect(const line &a,
                     const line &b) {
    LD u = det(a.t - a.s, b.s - a.s);
    LD v = -det(a.t - a.s, b.t - a.s);
    return (b.s * v + b.t * u) / (v + u);
}
```

Quiz: 射线判交

判断两条射线是否有交，射线由 `line` 给出。

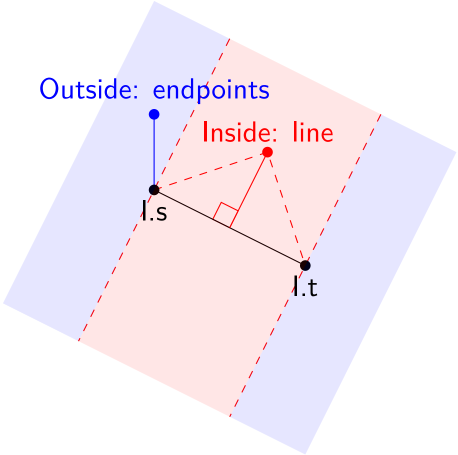



```cpp
struct line {
    point s, t;
};
bool ray_inter_judge(const line &a,
                     const line &b) {
    // TODO: finish ray_inter_judge
}
```

- 把之前的内容视作模板，如何实现最快？
- 如何实现全整数版本？

Quiz: 点到线段距离

以下实现点到线段距离有潜在问题。问题是什么，如何解决？

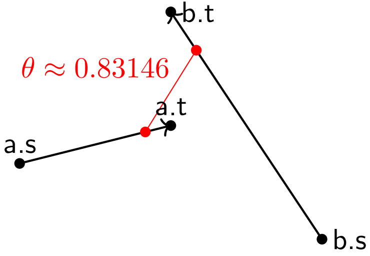



```
LD point_to_line (const point &p,
                  const line  &l) {
    return abs (det (l.t - l.s, p - l.s))
               / dis(l.s, l.t);
}
LD point_to_segment (const point &p,
                     const line  &l) {
 if (sgn (dot (l.s - p, l.t - l.s))
   * sgn (dot (l.t - p, l.t - l.s)) <= 0)
       return point_to_line(p, l);
 return min (dis (p, l.s), dis (p, l.t));
}
```

Quiz: 匀速直线移动的点最近距离

给定两个点，分别从各自的起点出发进行匀速直线移动，并在最后到达各自的终点。求在这个过程中，两个点的最近距离。

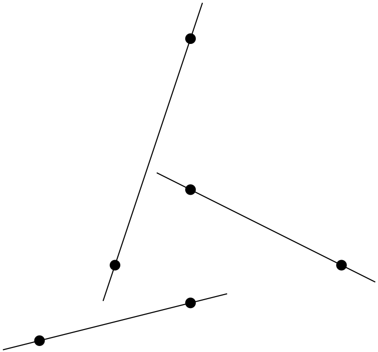
- 需要讨论所有特殊情况。
- 给出一个尽可能简短的解决方案。



Balloon Darts[2]

二维平面上有 n 个可以看做点的气球。
你有 3 个飞镖，飞镖的轨迹可以看做二维平面上的一条直线，经过的气球会被扎中。
问你是否能扎完所有气球。

- $n \le 10^4, |x|, |y| \le 10^9$



[2]GCPC (ICPC German) 2023 B, https://qoj.ac/contest/1402/problem/7653

Balloon Darts

直觉上，我们需要找很多点共线的直线。有没有快速（低于 n^2）判断存在三点共线的方法？

Balloon Darts

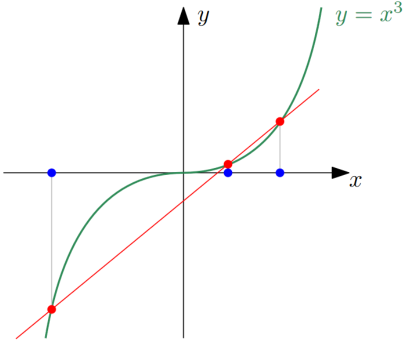
直觉上，我们需要找很多点共线的直线。有没有快速（低于 n^2）判断存在三点共线的方法？

- 3SUM-hard： 如果能在 $O(n^{2-\varepsilon})$ 解决该问题，就能 $O(n^{2-\varepsilon})$ 解决 3SUM 问题。[3]
- 3SUM conjecture: 3SUM 问题目前仍然没有低于平方的做法。

Degeneracy Testing Lemma

对于任意两两不同的实数 a, b, c,

$$(a, a^3), (b, b^3), (c, c^3)\text{共线} \Leftrightarrow a + b + c = 0.$$



[3] https://faculty.unist.ac.kr/algo/wp-content/uploads/sites/362/2020/06/cse520lec20.pdf

Balloon Darts

直觉上，我们需要找很多点共线的直线。有没有快速（低于 n^2）判断存在三点共线的方法？

- 不存在这样的方法。

Balloon Darts

直觉上，我们需要找很多点共线的直线。有没有快速（低于 n^2）判断存在三点共线的方法？

- 不存在这样的方法。
- 但如果能找到一条答案上的直线，$O(n)$ 验证有哪些点在上面是简单的。

如果有答案，我们希望快速找到答案。考虑怎么找直线？

- 有解时，很多点共线。

Balloon Darts

直觉上，我们需要找很多点共线的直线。有没有快速（低于 n^2）判断存在三点共线的方法？

- 不存在这样的方法。
- 但如果能找到一条答案上的直线，$O(n)$ 验证有哪些点在上面是简单的。

如果有答案，我们希望快速找到答案。考虑怎么找直线？

- 有解时，很多点共线。

考虑随机：随机选两个点，构成直线在答案里的概率有多大？

Balloon Darts

直觉上，我们需要找很多点共线的直线。有没有快速（低于 n^2）判断存在三点共线的方法？

- 不存在这样的方法。
- 但如果能找到一条答案上的直线，$O(n)$ 验证有哪些点在上面是简单的。

如果有答案，我们希望快速找到答案。考虑怎么找直线？

- 有解时，很多点共线。

考虑随机：随机选两个点，构成直线在答案里的概率有多大？

- 假设还需要放 k 条直线，至少存在一条穿过不少于 n/k 个点。
- 只看这条直线，随机成功概率一个很松的估计为 $\Omega(\frac{1}{k^2})$。

Balloon Darts

直觉上，我们需要找很多点共线的直线。有没有快速（低于 n^2）判断存在三点共线的方法？

- 不存在这样的方法。
- 但如果能找到一条答案上的直线，$O(n)$ 验证有哪些点在上面是简单的。

如果有答案，我们希望快速找到答案。考虑怎么找直线？

- 有解时，很多点共线。

考虑随机：随机选两个点，构成直线在答案里的概率有多大？

- 假设还需要放 k 条直线，至少存在一条穿过不少于 n/k 个点。
- 只看这条直线，随机成功概率一个很松的估计为 $\Omega(\frac{1}{k^2})$。
- 三阶段同时成功的概率至少为 $\frac{1}{9} \times \frac{1}{4} \times \frac{1}{1} = \frac{1}{36}$。

我不喜欢随机。有没有不用随机的做法？

Balloon Darts

直觉上，我们需要找很多点共线的直线。有没有快速（低于 n^2）判断存在三点共线的方法？

- 不存在这样的方法。
- 但如果能找到一条答案上的直线，$O(n)$ 验证有哪些点在上面是简单的。

如果有答案，我们希望快速找到答案。考虑怎么找直线？

- 有解时，很多点共线。

考虑随机：随机选两个点，构成直线在答案里的概率有多大？

- 假设还需要放 k 条直线，至少存在一条穿过不少于 n/k 个点。
- 只看这条直线，随机成功概率一个很松的估计为 $\Omega(\frac{1}{k^2})$。
- 三阶段同时成功的概率至少为 $\frac{1}{9} \times \frac{1}{4} \times \frac{1}{1} = \frac{1}{36}$。

我不喜欢随机。有没有不用随机的做法？

- 任意 $k+1$ 点中至少有两点在同一条答案直线上。

Balloon Darts

直觉上，我们需要找很多点共线的直线。有没有快速（低于 n^2）判断存在三点共线的方法？

- 不存在这样的方法。
- 但如果能找到一条答案上的直线，$O(n)$ 验证有哪些点在上面是简单的。

如果有答案，我们希望快速找到答案。考虑怎么找直线？

- 有解时，很多点共线。

考虑随机：随机选两个点，构成直线在答案里的概率有多大？

- 假设还需要放 k 条直线，至少存在一条穿过不少于 n/k 个点。
- 只看这条直线，随机成功概率一个很松的估计为 $\Omega(\frac{1}{k^2})$。
- 三阶段同时成功的概率至少为 $\frac{1}{9} \times \frac{1}{4} \times \frac{1}{1} = \frac{1}{36}$。

我不喜欢随机。有没有不用随机的做法？

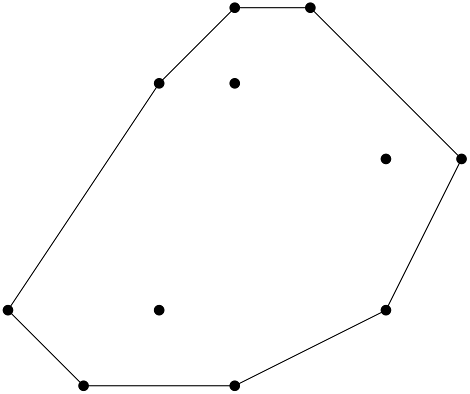
- 任意 $k+1$ 点中至少有两点在同一条答案直线上。
- 因此直接搜索，需要的总搜索次数为 $\binom{4}{2} \times \binom{3}{2} \times \binom{2}{2} = 18$。

凸包

能包含所有给定点的最小凸多边形叫做凸包。

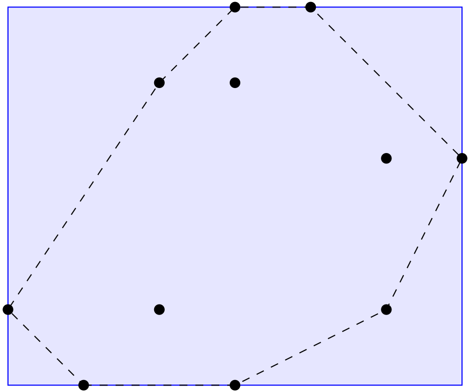
- 凸多边形：每个内角在 $[0,\pi)$ 的简单多边形；如果允许非严格则是 $[0,\pi]$。

或者，点集所有可能的带权平均点集合为凸包。



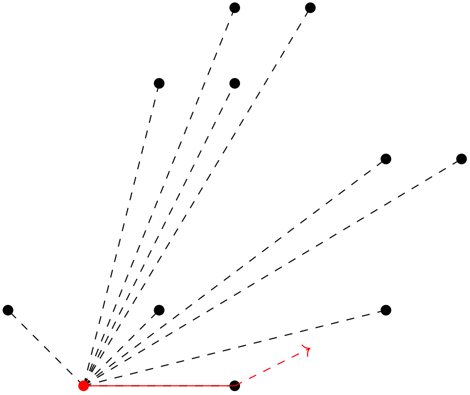
求凸包：性质

任何一个套住所有点的直线“边框”上的点一定可以在凸包上。
因此，我们可以任取 $x = \min x, x = \max x, y = \min y, y = \max y$ 上的点开始构造凸包。



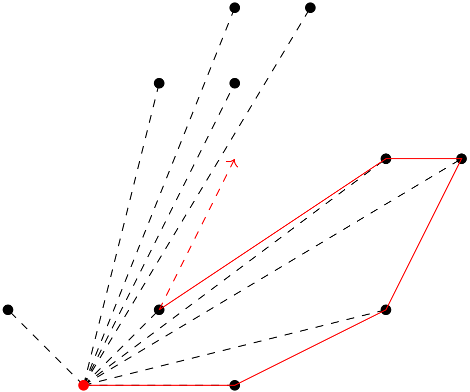
扫描线法：极角序（Graham）

选取左下角的点作为基准，对其余点进行逆时针排序。
用一根细绳尝试绕过按照这个顺序绕过所有点。



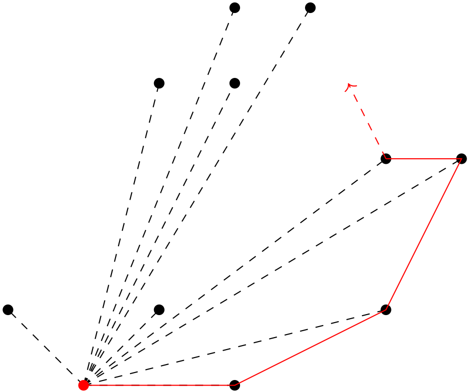
扫描线法：极角序（Graham）

选取左下角的点作为基准，对其余点进行逆时针排序。
用一根细绳尝试绕过按照这个顺序绕过所有点。



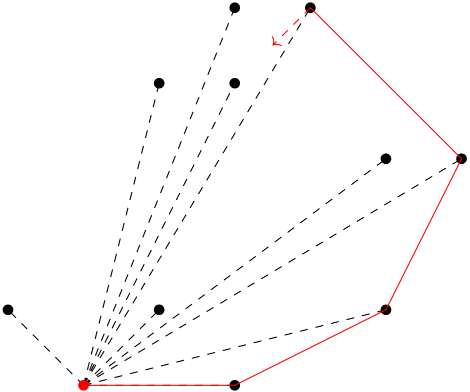
扫描线法：极角序（Graham）

选取左下角的点作为基准，对其余点进行逆时针排序。
用一根细绳尝试绕过按照这个顺序绕过所有点。



扫描线法：极角序（Graham）

选取左下角的点作为基准，对其余点进行逆时针排序。
用一根细绳尝试绕过按照这个顺序绕过所有点。



扫描线法：极角序（Graham）

选取左下角的点作为基准，对其余点进行逆时针排序。

- 由于其他点相对于基准点在 $[0,\pi)$ 的半平面内，因此可以直接使用叉积排序。
- 不要使用 `atan2`：当值域很大时，精度难以区分相近的点。
- 注意处理极角序相同的点：按照到基准点的距离从小到大排序。

用一根细绳尝试绕过按照这个顺序绕过所有点。

- 需要弹出不满足凸性的点。
- 使用单调栈实现。

扫描线法：极角序（Graham）

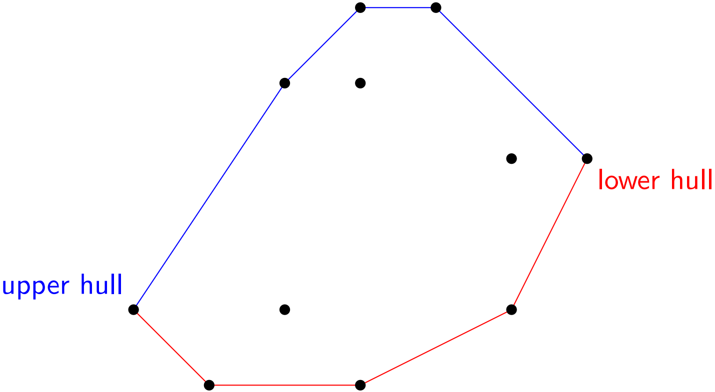
```cpp
vector <point> convex_hull (vector <point> a) {
    if (a.size() <= 2) return a; //或者 return {};
    point base = *min_element(a.begin(), a.end()); //字典序：less <pair>
    sort(a.begin(), a.end(), [&](auto u, auto v) {
        int s = sgn(det(u - base, v - base));
        if (s) return s > 0;
        else return sgn(dis(u, base) - dis(v, base)) < 0;
    });
    vector <point> ret;
    for (auto i : a) {
        while (ret.size() > 1
           && !turn_left(ret[ret.size() - 2], ret[ret.size() - 1], i))
            ret.pop_back();
        ret.push_back(i);
    }
    return ret; //或者在 ret.size() <= 2 时 return {};
}
```

扫描线法：字典序（Andrew）

Graham 不够好：相对较慢，容易写错。

- 相对较慢：排序需要计算 $O(n \log n)$ 次叉积。
- 容易写错：细节比较多，容易出现挂边界的情况。

Andrew 算法扫描两遍计算出上下凸壳，通常在效率和实现上相比 Graham 有优势。



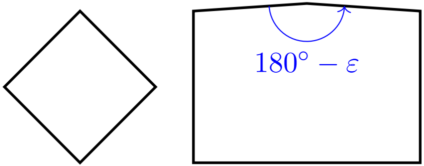
扫描线法：字典序（Andrew）

```cpp
vector <point> convex_hull (vector <point> a) {
    if (a.size() <= 2) return a; //或者 return {};
    sort (a.begin(), a.end()); //字典序: less <pair>
    vector <point> ret;
    for (int i = 0; i < (int) a.size(); i++) {
        while (ret.size() > 1
        && !turn_left(ret[ret.size() - 2], ret[ret.size() - 1], a[i]))
            ret.pop_back();
        ret.push_back a[i]);
    }
    int fixed = (int) ret.size();
    for (int i = (int) a.size() - 2; i >= 0; --i) {
        while (ret.size() > fixed
        && !turn_left(ret[ret.size() - 2], ret[ret.size() - 1], a[i]))
            ret.pop_back();
        ret.push_back(a[i]);
    }
    ret.pop_back (); return ret;  //或者在 ret.size() <= 2 时 return {};
}
```

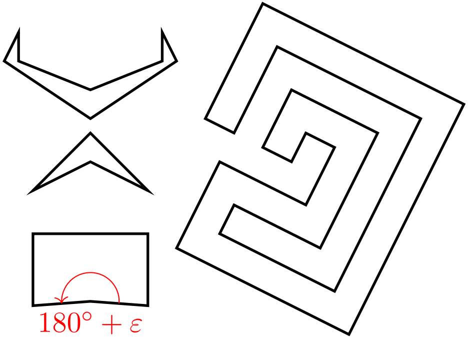
Convex Checker[4]

给定一个一系列点 $\{p\}$，判断 $p_1 - p_2 - p_3 - \cdots - p_n - p_1$ 是否为简单凸多边形。

- $n \leq 2 \times 10^5, |x|, |y| \leq 10^9$
- 此处简单凸多边形定义：无重点、不自交、内角严格小于 π。



是凸包



不是凸包

[4] CCPC Guilin 2023 热身赛 C, https://qoj.ac/contest/1408/problem/7730

Convex Checker

算法 1

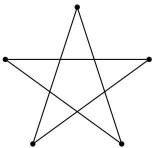
检查所有内角是否为劣角。

Convex Checker

算法 1

检查所有内角是否为劣角。

Wrong Answer: **五角星不是凸包**



$$\angle A = \angle B = \angle C = \angle D = \angle E = 36^\circ < 180^\circ \Rightarrow \text{Convex!}$$

Convex Checker

算法 2

检查所有内角是否为劣角，并且检查内角和是否为 $(n-2)\pi$。

Convex Checker

算法 2

检查所有内角是否为劣角，并且检查内角和是否为 $(n-2)\pi$。

Correct: 内角和一定是 π 的倍数，因此 ε 可以设得很大（如 $\varepsilon = 1$），`atan2` 没有精度问题。

Convex Checker

算法 2

检查所有内角是否为劣角，并且检查内角和是否为 $(n-2)\pi$。

Correct: 内角和一定是 π 的倍数，因此 ε 可以设得很大（如 $\varepsilon = 1$），`atan2` 没有精度问题。

算法 3

运行凸包算法，检查凸包是否一致。

检查一致的一种可行的方式是：比较边的向量集合是否相同。(为什么？)

Convex Checker

算法 2

检查所有内角是否为劣角，并且检查内角和是否为 $(n-2)\pi$。

Correct: 内角和一定是 π 的倍数，因此 ε 可以设得很大（如 $\varepsilon = 1$），`atan2` 没有精度问题。

算法 3

运行凸包算法，检查凸包是否一致。

检查一致的一种可行的方式是：比较边的向量集合是否相同。(为什么 ?)

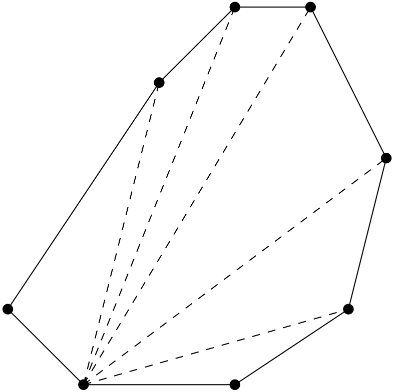
算法 4

魔改 Graham 算法：将左下角的点转到开头，按顺序检查极角序和凸性。

- 以上做法都需要先把凸包转为逆时针凸包。

计算凸包面积

三角形面积是好算的，因此我们可以分而治之。

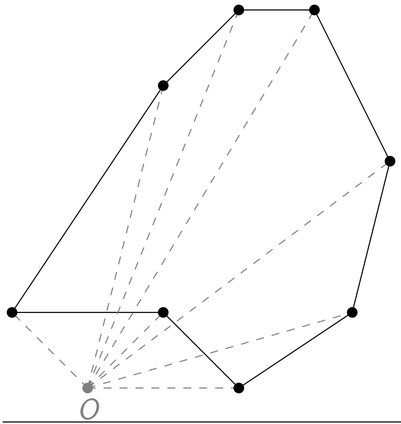


```
LD area(vector <point> &a) {
  LD ret = 0;
  for (int i = 1; i < a.size() - 1; i++)
    ret += det(a[i] - a[0], a[i + 1] - a[0]);
  return ret / 2;
}
```

计算任意多边形面积

可以直接算从原点出发的有向三角形面积，且可以推广到任意多边形。[5]

- 叉积求面积时有符号。确保多边形为逆时针，或者使用 abs。



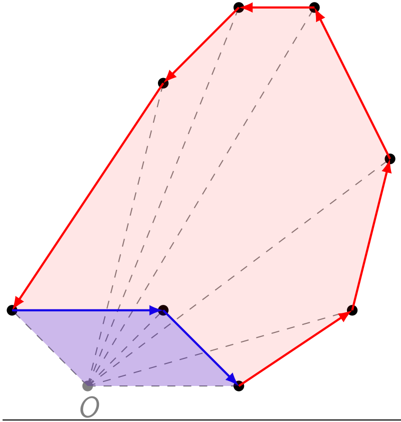

```
LD area(vector <point> &a) {
    LD ret = 0;
    for (int i = 0; i < a.size(); i++) {
        int j = (i + 1) % a.size();
        ret += det(a[i], a[j]);
    }
    return ret / 2;
}
```

[5] 本质上，这是在对区域边界进行线积分，叉积的形式和格林公式得到的 $\oint_{\partial S} xdy - ydx$ 是一致的。

计算任意多边形面积

可以直接算从原点出发的有向三角形面积，且可以推广到任意多边形。[5]

- 叉积求面积时有符号。确保多边形为逆时针，或者使用 abs。



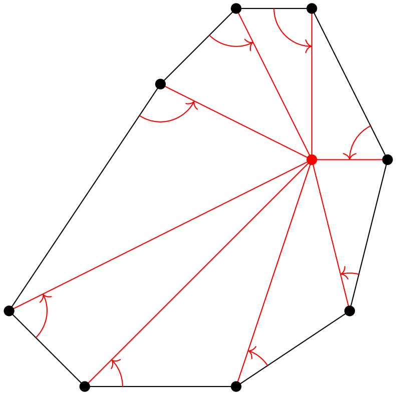

```cpp
LD area(vector <point> &a) {
    LD ret = 0;
    for (int i = 0; i < a.size(); i++) {
        int j = (i + 1) % a.size();
        ret += det(a[i], a[j]);
    }
    return ret / 2;
}
```

[5] 本质上，这是在对区域边界进行线积分，叉积的形式和格林公式得到的 $\oint_{\partial S} x dy - y dx$ 是一致的。

点在凸包内

凸包内的点在所有边界的（非严格）左侧。

- 当非严格时，`turn_left` 应改为 `>=`。



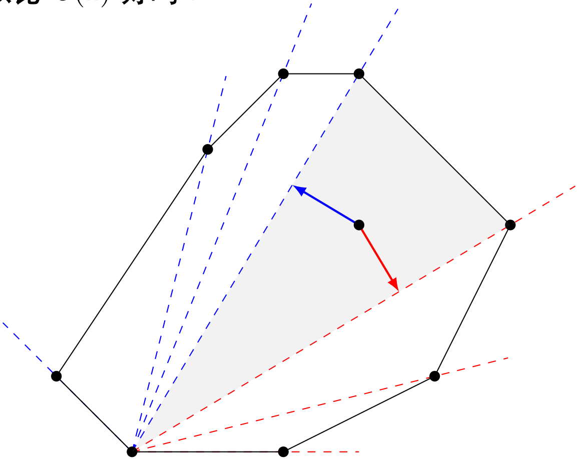
```
bool inside(point p, vector <point> &a) {
    for (int i = 0; i < a.size(); i++) {
        int j = (i + 1) % a.size();
        if (!turn_left(a[i], a[j], p))
            return false;
    }
    return true;
}
```

点在凸包内

可以比 $O(n)$ 好吗？

点在凸包内

可以比 $O(n)$ 好吗？



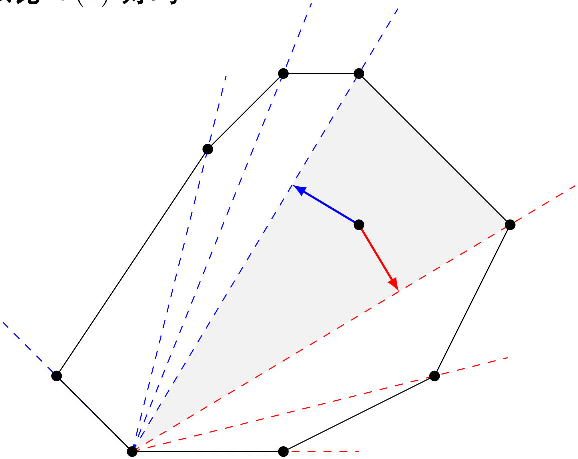
算法 1

尝试在极角序上二分定位所在三角形，然后判断是否在对应三角形内。

- 如何证明正确性？

点在凸包内

可以比 $O(n)$ 好吗？



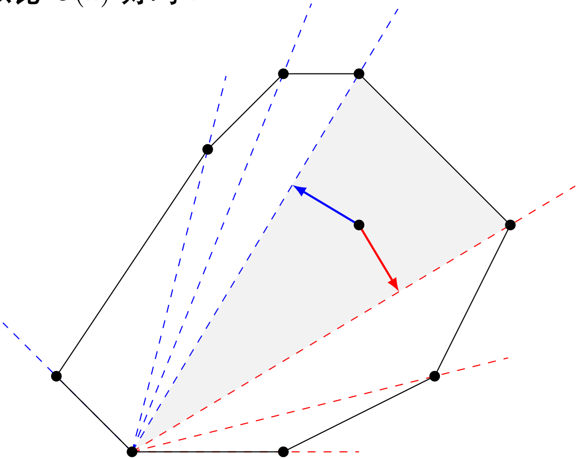
算法 1

尝试在极角序上二分定位所在三角形，然后判断是否在对应三角形内。

- 如何证明正确性？
- No is No：如果不在凸包内，那么一定不在任何一个三角形区域内，一定会返回 `false`。

点在凸包内

可以比 $O(n)$ 好吗？



算法 1

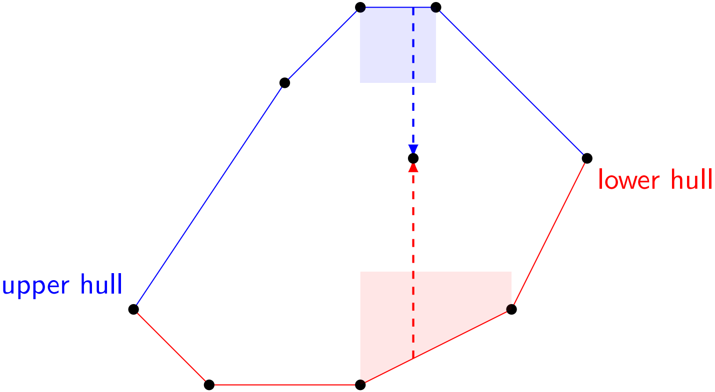
尝试在极角序上二分定位所在三角形，然后判断是否在对应三角形内。

- 如何证明正确性？
- No is No: 如果不在凸包内，那么一定不在任何一个三角形区域内，一定会返回 `false`。
- Yes is Yes: 如果在凸包内，那么一定在相对于基点的一个严格半平面内，极角序能够正确定位所在三角形。

点在凸包内

算法 2

判断在上凸壳下方，且在上凸壳上方。



Quick Review

让我们先快速复习一下之前的内容。

- 在计算几何里，我们更倾向于用向量而非解析式来表达一个对象。
  - 可以利用点积和叉积快速判断向量的方向。
  - 尽可能使用向量运算来规避浮点数运算。
- 凸包是一种~~凸的包~~相对有序的结构。
  - 可以使用排序和扫描线求出凸包。
  - 基于凸包的结构，我们可以进行一系列操作，如 $O(\log n)$ 判断一个点是否在凸包内；相比之下，简单多边形会困难很多。
- 可以使用叉积计算多边形面积。

凸包上二分

凸包提供了一个有序的结构，因此可以进行类似有序表上的二分操作。

- 点在凸包内
- 点到凸包切线
- 凸包外一点到到凸包的距离
- 直线与凸包交点
- ...

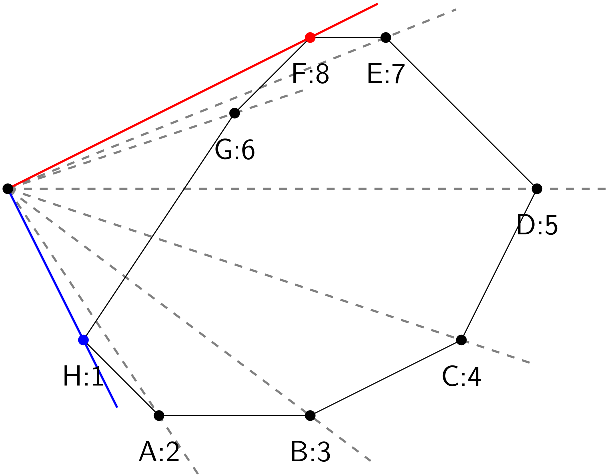
Um_nik: Stop learning useless algorithms, go and solve some problems, learn how to use **binary search**. [6]

[6] https://codeforces.com/blog/entry/92248

点到凸包切线

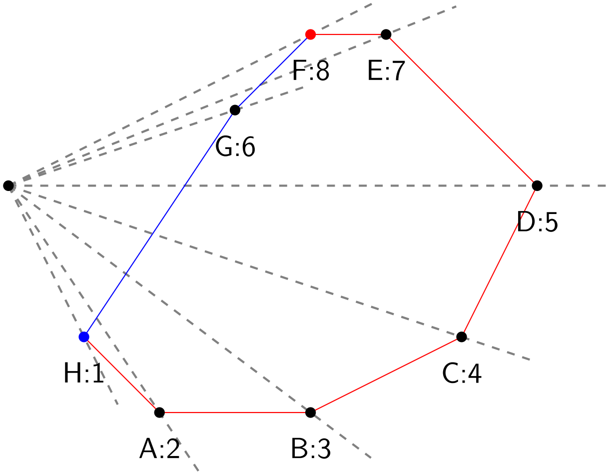
切点即为相对极角序最大和最小的点。

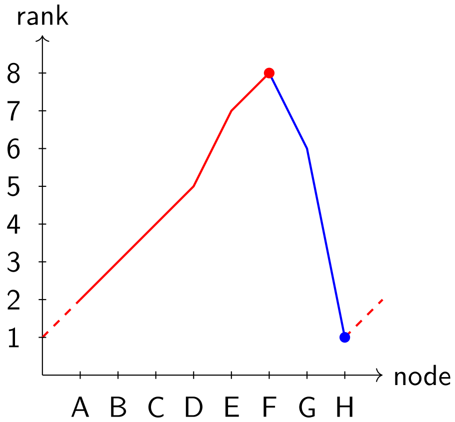
- 如何实现？



点到凸包切线

可以理解为循环单峰函数求极值。





点到凸包切线：二分实现[7]

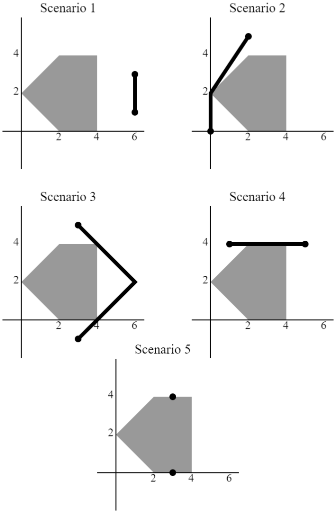
```
int search (auto f) {
    int l = 0, r = (int) a.size() - 1;
    if (f(a[0], a.back())) {
        while (l + 1 < r) {
            int mid = (l + r) / 2;
            if (f(a[mid], a[l]) && f(a[mid], a[mid - 1])) l = mid;
            else r = mid;
        } return l;
    } else {
        while (l + 1 < r) {
            int mid = (l + r) / 2;
            if (f(a[mid], a[r]) && f(a[mid], a[mid + 1])) r = mid;
            else l = mid;
        } return r;
    }
}
vector <point> get_tan(point u) {
    return { a[search ([&](auto x, auto y) {return turn_left(u, y, x);})],
             a[search ([&](auto x, auto y) {return turn_left(u, x, y);})] };
}
```

[7]钱哥 (skip2004) 教我的。需要 C++17。需要严格凸包。要求点在凸包外。

Spaceship Exploration[8]

给定一个凸包形状的障碍物，多次询问凸包（非严格）外两点距离。
其中，距离定义为最短的、至多有一个拐点的折线段长度，且在凸包（非严格）外；无解时定义为 -1。

- $n, q \le 10^5, |x|, |y| \le 10^9$
- 精度要求：相对或绝对 10^{-6}



[8]ICPC Jakarta 2023 D, https://codeforces.com/contest/1906/problem/D

Spaceship Exploration

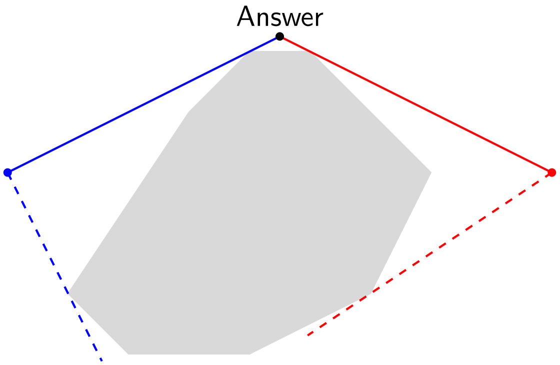
- 考虑无解：在两条平行但不相同的边内，否则一定可以走到无限远的地方有解。

Spaceship Exploration

- 考虑无解：在两条平行但不相同的边内，否则一定可以走到无限远的地方有解。
- 考虑没有拐点：两点之间线段最短。

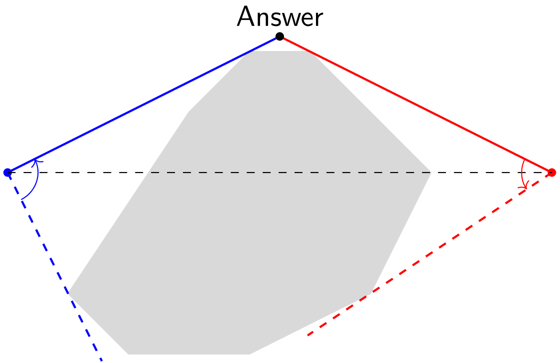
Spaceship Exploration

- 考虑无解：在两条平行但不相同的边内，否则一定可以走到无限远的地方有解。
- 考虑没有拐点：两点之间线段最短。
- 考虑有拐点：一定是为了绕开障碍物，最近的能绕开障碍物的射线是切线。



Spaceship Exploration

- 考虑无解：在两条平行但不相同的边内，否则一定可以走到无限远的地方有解。
- 考虑没有拐点：两点之间线段最短。
- 考虑有拐点：一定是为了绕开障碍物，最近的能绕开障碍物的射线是切线。
- 判定是否有拐点：判定两点连线是否完全处于切线角度范围内即可。



Spaceship Exploration

本题需要正确处理在凸包边界上的切线。一个可能的实现如下。[9]

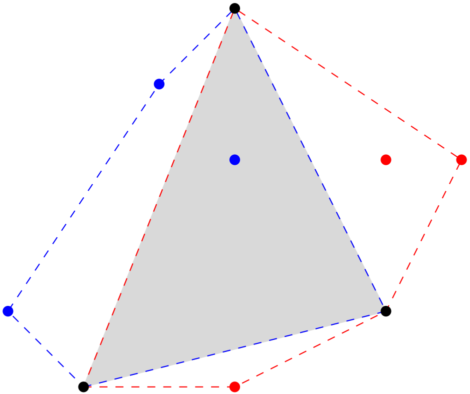
```
map <point, int> id; // point -> index
vector <point> get_tan(point u) {
    if (id.count(u)) return {a[(id[u] + n - 1) % n], a[(id[u] + 1)%n]};
    if (point_on_segment(u, {a[0], a.back()}))
        return {a.back(), a[0]};
    return {a[search([&](auto x, auto y){return turn_left(u, y, x);})],
            a[search([&](auto x, auto y){return turn_left(u, x, y);})]};
}
```

[9] Submission: https://codeforces.com/contest/1906/submission/236298011

Dynamic Convex Hull[10]

给定 n 个点，m 次询问，每次询问给定 k_i 个点，询问这 $n+k_i$ 个点构成的凸包面积。

- $n, m \le 2 \times 10^5, \sum k_i \le 10^6$
- 精度要求：准确答案（乘 2 输出整数）



[10] CCPC Guangzhou 2021 L, https://codeforces.com/gym/103415/problem/L

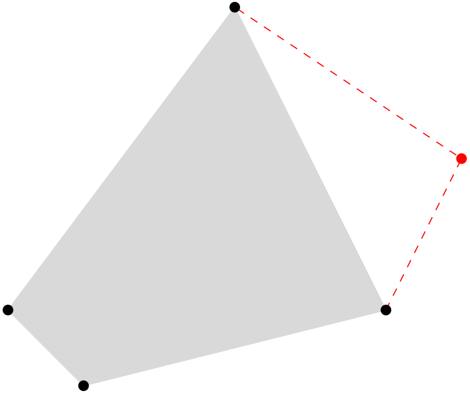
Dynamic Convex Hull

如果每次只加一个凸包外的点呢？

Dynamic Convex Hull

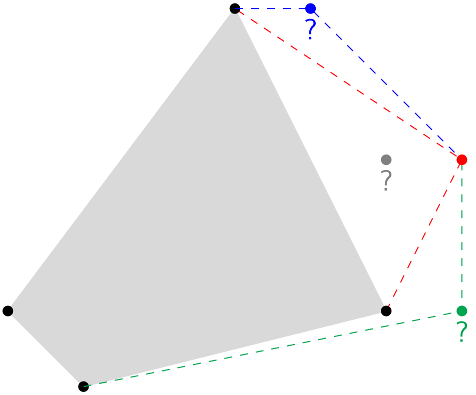
如果每次只加一个凸包外的点呢？

- 额外的边是切线。



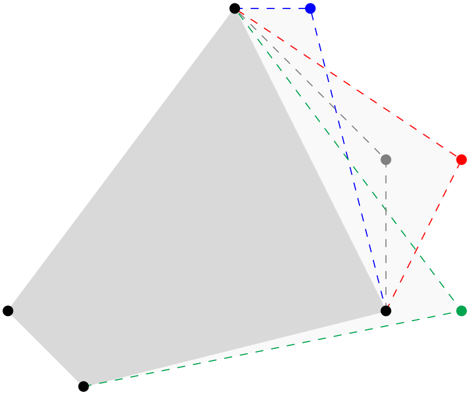
Dynamic Convex Hull

再加一个点时，需要在新凸包上求切点，不依赖高级数据结构是难以做到的。



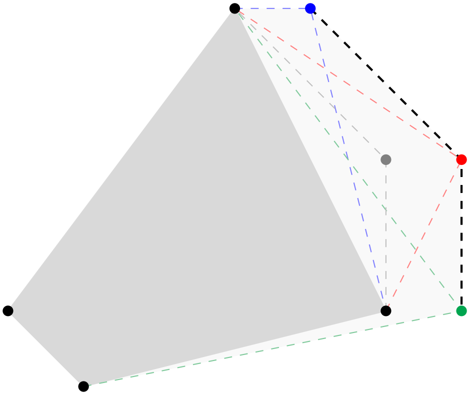
Dynamic Convex Hull

最终形成的形态是对原凸包求出所有切线后，再对不规则区域求凸包，尝试让外部闭合。



Dynamic Convex Hull

最终形成的形态是对原凸包求出所有切线后，再对不规则区域求凸包，尝试让外部闭合。

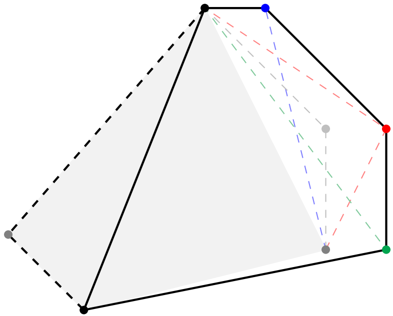


Dynamic Convex Hull

算法

对所有凸包外的点 p_j 求出切点 u_j, v_j，然后对 $\{p_j, u_j, v_j \mid j \in [k_i]\}$ 求新凸包。

- 如果新凸包相邻两个点为原凸包上的点，则计算这段凸包上的边对面积的贡献；
- 否则直接计算这条边的贡献。



Dynamic Convex Hull

算法

对所有凸包外的点 p_{ij} 求出切点 u_{ij}, v_{ij}，然后对 $\{p_{ij}, u_{ij}, v_{ij} \mid j \in [k_i]\}$ 求新凸包。

- 如果新凸包相邻两个点为原凸包上的点，则计算这段凸包上的边对面积的贡献；
- 否则直接计算这条边的贡献。

时间复杂度：$O(\sum k_i(\log n + \log \sum k_i))$。
需要实现：求凸包，点在凸包内，求切点，（求面积时对叉积的）前缀和。
注意特判没有有效新点的情况，以及前缀和实现细节。

```
LL sum[N]; // sum[i] = det(p[0], p[1]) + ... + det(p[i], p[(i + 1) % n])
LL get_sum(int l, int r) {
    if (l < r)
        return sum[r - 1] - (l == 0 ? 0ll : sum[l - 1]);
    else
        return sum[n - 1] - sum[r - 1] + sum[l - 1];
}
```

Dynamic Convex Hull

Fun fact：有一个队伍快速通过了这个题却没做出简单题，被教练喷“只会抢一血”。

Place	School	Team	Solved	Penalty	A 1	B 2	C 136	D 1	E 2	F 75	G 1	H 209	I 218	J 2	K 22	L 1
1	1 清华大学	三杯两盏咸酒	7	1027		+ 10/273	+ 1/64	+ 1/105		+ 1/130		+ 1/25	+ 1/32		+ 1/218	
2	2 暨南大学	追光者	7	1203		+ 2/183	+ 1/81			+ 1/168	+ 1/232	+ 4/39	+ 1/44		+ 5/296	
3	3 上海交通大学	洒满黄浦江	6	713			+ 2/43		+ 2/169	+ 2/200	- 5/299	+ 2/18	+ 1/24		- 14/298	+ 2/159
4	4 浙江大学	Phantom Ensemble	5	513			+ 2/64		- 2/299	+ 1/92		+ 2/27	+ 1/33		+ 2/237	
5	5 南京大学	虚幻黄昏	5	519			+ 3/58			+ 2/113		+ 1/41	+ 1/46	- 4/297	+ 2/181	
6	6 杭州电子科技大学	杭电2021-1队	5	576			+ 1/75			+ 2/172		+ 1/53	+ 1/28		+ 2/208	
7	浙江大学	The cell phone battery is dead	5	595			+ 1/93		- 2/291	+ 1/202		+ 1/58	+ 1/34		+ 2/188	

Minimum Euclidean Distance[11]

给定一个凸包。
每次询问给定一个圆，问凸包内选一点，使得到圆内均匀随机一点的距离平方的期望最小，输出这个最小的距离平方的期望。

- $n, q \le 5000, |x|, |y| \le 10^9$
- 精度要求：相对或绝对 10^{-4}
- **Bonus**: $n, q \le 10^5$
- Hint: 点 p 到一个圆心为 C, 半径为 R 的圆内均匀随机一点的期望距离平方是

$$\frac{1}{2}R^2 + dis^2(C, p)$$

（证明留作 微积分 习题）

[11] ICPC EC Online (I) 2023 K, https://qoj.ac/contest/1485/problem/8082; 数据很弱，AC 不代表代码正确。

Minimum Euclidean Distance

根据提示，本题其实是凸包内选一个点到一个目标点（圆心）尽可能近。
因此，我们只需要判定目标点是否在凸包内，以及求出目标点在凸包外时到凸包的距离。

- 前者我们已经掌握了，后者能在 $O(\log n)$ 的时间内求出来吗？

Minimum Euclidean Distance

根据提示，本题其实是凸包内选一个点到一个目标点（圆心）尽可能近。
因此，我们只需要判定目标点是否在凸包内，以及求出目标点在凸包外时到凸包的距离。

- 前者我们已经掌握了，后者能在 $O(\log n)$ 的时间内求出来吗？

算法 1

使用凸包上二分，计算凸包上离目标点最近的顶点，然后这个顶点附近的边查询最近距离。

如何证明正确性？

Minimum Euclidean Distance

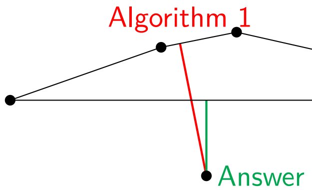
根据提示，本题其实是凸包内选一个点到一个目标点（圆心）尽可能近。
因此，我们只需要判定目标点是否在凸包内，以及求出目标点在凸包外时到凸包的距离。

- 前者我们已经掌握了，后者能在 $O(\log n)$ 的时间内求出来吗？

算法 1

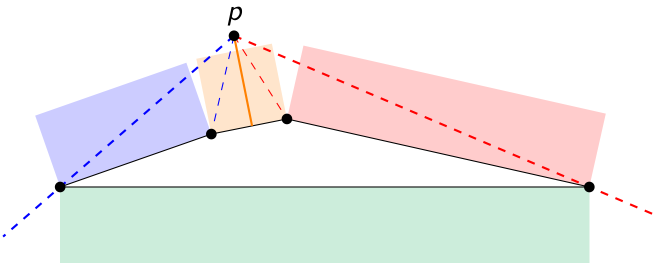
使用凸包上二分，计算凸包上离目标点最近的顶点，然后这个顶点附近的边查询最近距离。

Wrong Answer: 顶点距离不是的单峰函数；顶点距离不意味着线段距离。



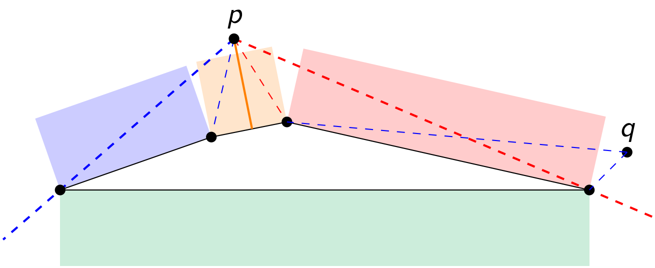
Minimum Euclidean Distance

- 每条凸包上的线段掌管的范围是一系列凸包外不交的区域。
- 每个凸包外的点到凸包上最近点只和能 “直接看到”，也即切线内的边有关。



Minimum Euclidean Distance

- 每条凸包上的线段掌管的范围是一系列凸包外不交的区域。
- 每个凸包外的点到凸包上最近点只和能“直接看到”，也即切线内的边有关。

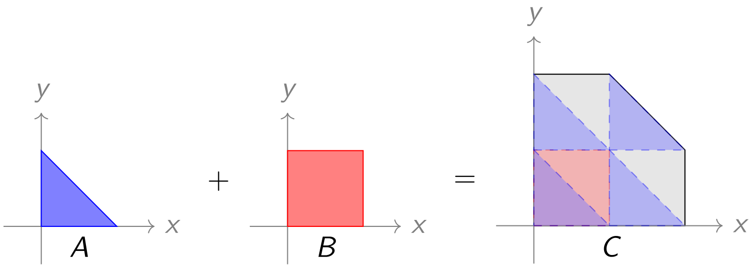


算法 2

先求出切线，然后在近侧二分顶点连线与线段的点积符号，求出的变号点附近即是答案。
注意可能没有变号点：如点 q，此时需要有类似点到线段距离的特判。

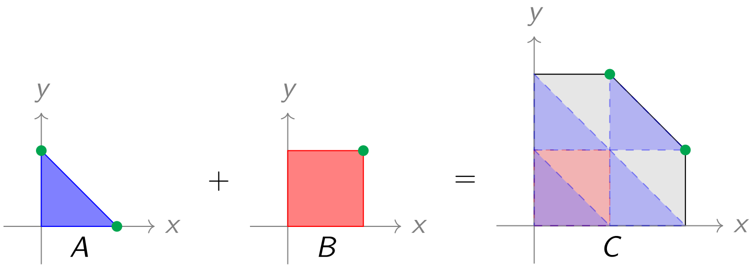
闵可夫斯基和

将凸包视为在内部的点集。给定两个凸包 A, B，求凸包 $C = \{a + b \mid a \in A, b \in B\}$。



闵可夫斯基和

将凸包视为在内部的点集。给定两个凸包 A, B，求凸包 $C = \{a + b \mid a \in A, b \in B\}$。

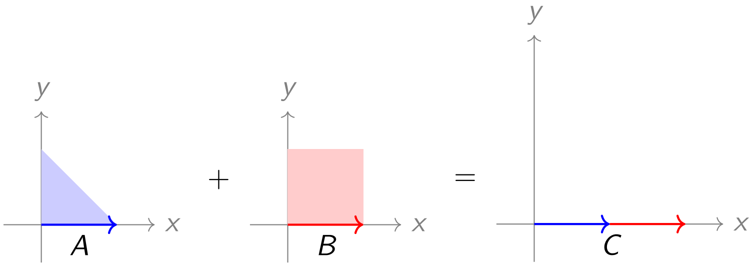


一些观察：

- 在 C 上的顶点一定是原凸包上最靠近这个方向的点拼出来的。
- 凸包的边是按照极角序排好序的。

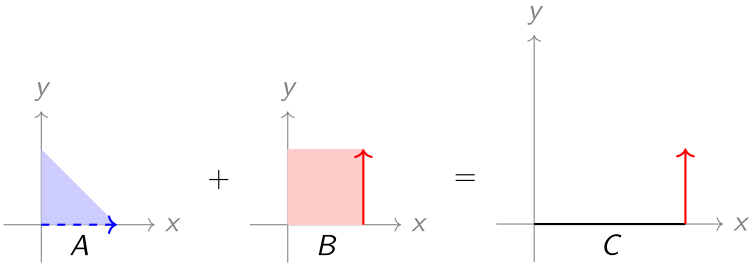
闵可夫斯基和

维护双指针，依次合并两个有序表。



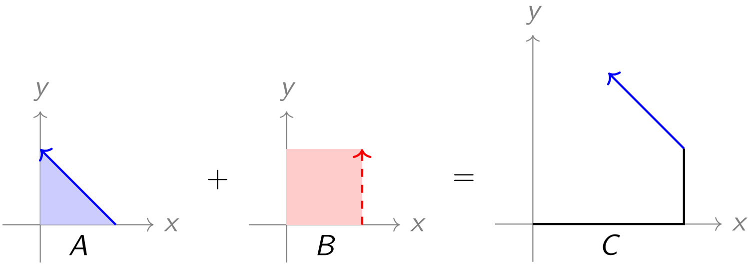
闵可夫斯基和

维护双指针，依次合并两个有序表。



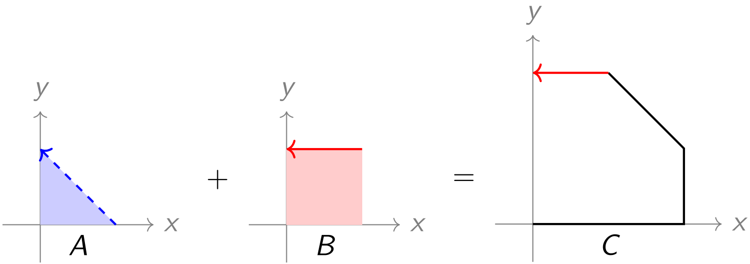
闵可夫斯基和

维护双指针，依次合并两个有序表。



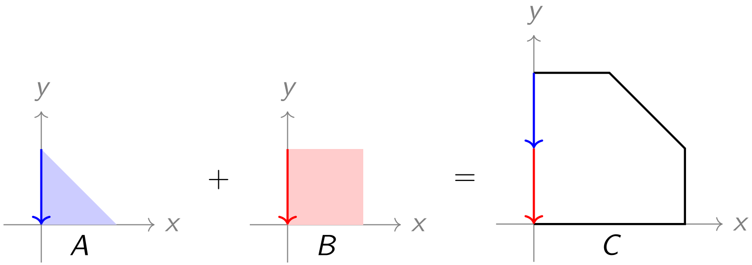
闵可夫斯基和

维护双指针，依次合并两个有序表。



闵可夫斯基和

维护双指针，依次合并两个有序表。



闵可夫斯基和

注意这个代码无法处理凸包退化，或者精度较差的情况：这些情况需要特殊处理。

```
vector <point> minkovski (vector <vector <point>> a) {
    // a[0], a[1]: 起点为左下的逆时针凸包
    for (int i = 0; i < 2; i++) a[i].push_back(a[i].front());
    int i[2] = {0, 0},
      len[2] = {(int) a[0].size() - 1, (int) a[1].size() - 1};
    vector <point> ret;
    ret.push_back(a[0][0] + a[1][0]);
    do { // 输入退化时会死循环，需特判
        int d = sgn(det(a[1][i[1] + 1] - a[1][i[1]],
                        a[0][i[0] + 1] - a[0][i[0]])) >= 0;
        ret.push_back(a[d][i[d] + 1] - a[d][i[d]] + ret.back());
        i[d] = (i[d] + 1) % len[d];
    } while(i[0] || i[1]);
    return ret; // 结果不是严格凸包
}
```

最远点对

给定 n 个点，求其中一对距离最大的点。输出这个距离。

最远点对

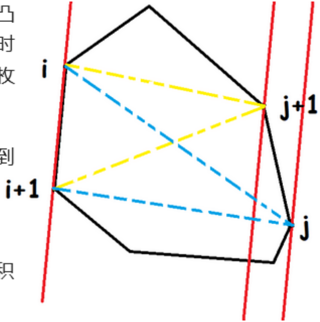
给定 n 个点，求其中一对距离最大的点。输出这个距离。

- 旋转卡壳！

首先使用任何一种凸包算法求出给定所有点的凸包，有着最长距离的点对一定在凸包上。而由于凸包的形状，我们发现，逆时针地遍历凸包上的边，对于每条边都找到离这条边最远的点，那么这时随着边的转动，对应的最远点也在逆时针旋转，不会有反向的情况，这意味着我们可以在逆时针枚举凸包上的边时，记录并维护一个当前最远点，并不断计算、更新答案。

求出凸包后的数组自然地是按照逆时针旋转的顺序排列，不过要记得提前将最左下角的 1 节点补到数组最后，这样在挨个枚举边 $(i, i+1)$ 时，才能把所有边都枚举到。

枚举过程中，对于每条边，都检查 $j+1$ 和边 $(i, i+1)$ 的距离是不是比 j 更大，如果是就将 j 加一，否则说明 j 是此边的最优点。判断点到边的距离大小时可以用叉积分别算出两个三角形的面积（如图，黄、蓝两个同底三角形的面积）并直接比较。



OI-Wiki 上对于旋转卡壳的介绍。(https://oi-wiki.org/geometry/rotating-calipers/)

最远点对：闵可夫斯基和

给定 n 个点，求其中一对距离最大的点。输出这个距离。

要求

使用闵可夫斯基和解决这个问题。

最远点对：闵可夫斯基和

给定 n 个点，求其中一对距离最大的点。输出这个距离。

要求

使用闵可夫斯基和解决这个问题。

- 令 A 为点集构成的凸包。我们要求的其实是

$$\max |a| : a \in \{x - y \mid x, y \in A\}.$$

最远点对：闵可夫斯基和

给定 n 个点，求其中一对距离最大的点。输出这个距离。

要求

使用闵可夫斯基和解决这个问题。

- 令 A 为点集构成的凸包。我们要求的其实是

$$\max |a| : a \in \{x - y \mid x, y \in A\}.$$

算法

求出 A 和 $-A$ 的闵可夫斯基和，输出顶点中到零点最远的。

- 观察算法运行流程，可以发现这个做法和旋转卡壳的双指针本质相同。
- 实现的时候一定要注意：取负后需要转换为左下角在第一个的逆时针凸包。

凸包判交

给定两个凸包，判断他们是否存在公共点。

凸包判交

给定两个凸包，判断他们是否存在公共点。

算法 1

对于两个凸包，分别判断每个顶点是否在另一个凸包内。

如何证明正确性？

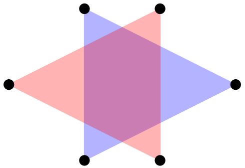
凸包判交

给定两个凸包，判断他们是否存在公共点。

算法 1

对于两个凸包，分别判断每个顶点是否在另一个凸包内。

Wrong Answer: 存在顶点在另一个凸包内不是必要条件。



凸包判交

算法 2

半平面交。

好长，不会，还没学。

凸包判交

算法 2

半平面交。

好长，不会，还没学。

算法 3

闵可夫斯基和。

怎么使用闵可夫斯基和判定凸包是否有交？

凸包判交

算法 2

半平面交。

好长，不会，还没学。

算法 3

闵可夫斯基和。

怎么使用闵可夫斯基和判定凸包是否有交？

- 求出 A 和 $-B$ 的闵可夫斯基和，判定零点是否在里面。

凸包判交: Revisited

算法 2

半平面交。

什么是半平面交？为什么可以使用半平面交？

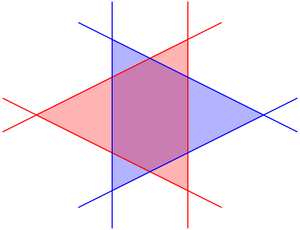
凸包判交: Revisited

算法 2

半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。



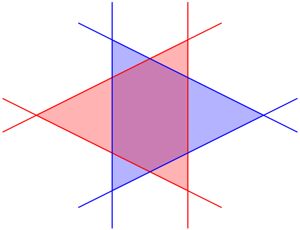
凸包判交: Revisited

算法 2

半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。



凸包判交: Revisited

算法 2

半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。

怎么求半平面交？

凸包判交: Revisited

算法 2

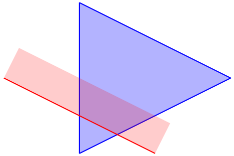
半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。

怎么求半平面交？

- $O(n^2)$ 算法相对简单：每次用一条直线去切割当前的凸包。



凸包判交: Revisited

算法 2

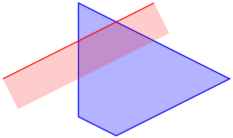
半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。

怎么求半平面交？

- $O(n^2)$ 算法相对简单：每次用一条直线去切割当前的凸包。



凸包判交: Revisited

算法 2

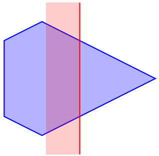
半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。

怎么求半平面交？

- $O(n^2)$ 算法相对简单：每次用一条直线去切割当前的凸包。



凸包判交: Revisited

算法 2

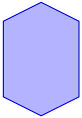
半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。

怎么求半平面交？

- $O(n^2)$ 算法相对简单：每次用一条直线去切割当前的凸包。



凸包判交：Revisited

算法 2

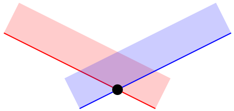
半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。

怎么求半平面交？

- $O(n^2)$ 算法相对简单：每次用一条直线去切割当前的凸包。
- $O(n \log n)$ 算法直觉上来说和求凸包类似。对所有直线按照极角排序，然后一个一个放进栈里，遇到不满足单调性的直线就弹掉。



凸包判交: Revisited

算法 2

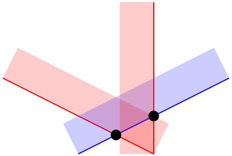
半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。

怎么求半平面交？

- $O(n^2)$ 算法相对简单：每次用一条直线去切割当前的凸包。
- $O(n\log n)$ 算法直觉上来说和求凸包类似。对所有直线按照极角排序，然后一个一个放进栈里，遇到不满足单调性的直线就弹掉。



凸包判交: Revisited

算法 2

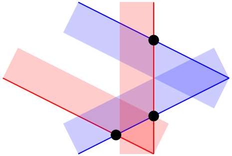
半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。

怎么求半平面交？

- $O(n^2)$ 算法相对简单：每次用一条直线去切割当前的凸包。
- $O(n \log n)$ 算法直觉上来说和求凸包类似。对所有直线按照极角排序，然后一个一个放进栈里，遇到不满足单调性的直线就弹掉。



凸包判交: Revisited

算法 2

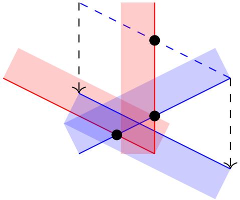
半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。

怎么求半平面交？

- $O(n^2)$ 算法相对简单：每次用一条直线去切割当前的凸包。
- $O(n \log n)$ 算法直觉上来说和求凸包类似。对所有直线按照极角排序，然后一个一个放进栈里，遇到不满足单调性的直线就弹掉。



凸包判交: Revisited

算法 2

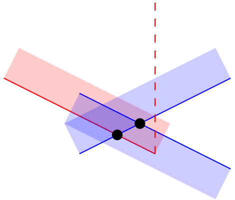
半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。

怎么求半平面交？

- $O(n^2)$ 算法相对简单：每次用一条直线去切割当前的凸包。
- $O(n \log n)$ 算法直觉上来说和求凸包类似。对所有直线按照极角排序，然后一个一个放进栈里，遇到不满足单调性的直线就弹掉。



凸包判交：Revisited

算法 2

半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。

怎么求半平面交？

- $O(n^2)$ 算法相对简单：每次用一条直线去切割当前的凸包。
- $O(n\log n)$ 算法直觉上来说和求凸包类似。对所有直线按照极角排序，然后一个一个放进栈里，遇到不满足单调性的直线就弹掉。

如何实现？

凸包判交: Revisited

算法 2

半平面交。

什么是半平面交？为什么可以使用半平面交？

- 凸包可以看做由每条边左侧的半平面的交集。
- 如果若干个半平面交集非空且有界，那么会形成一个凸包。

怎么求半平面交？

- $O(n^2)$ 算法相对简单：每次用一条直线去切割当前的凸包。
- $O(n \log n)$ 算法直觉上来说和求凸包类似。对所有直线按照极角排序，然后一个一个放进栈里，遇到不满足单调性的直线就弹掉。

如何实现？

- 半平面交很容易写错（或者不优）。
- Fun fact: 半个月前我还修了一个我自己板子里的错误。这个板子已经在长达三年的若干次修订前通过了无数道题了。

半平面交: $O(n^2)$

```cpp
bool two_side(point a, point b, line c) {
    return sgn (det (a - c.s, c.t - c.s))
        * sgn (det (b - c.s, c.t - c.s)) < 0; }
vector <point> cut (const vector<point> &c, line l) {
    vector <point> ret;
    for (int i = 0; i < (int) c.size(); ++i) {
        int j = (i + 1) % (int) c.size();
        if (turn_left (l.s, l.t, c[i])) // turn left 必须 <=
            ret.push_back(c[i]);
        if (two_side (c[i], c[j], l))
            ret.push_back(line_intersect (l, {c[i], c[j]}));
    }
    return ret;
}
```

UBC 在 ICPC WF Moscow 上使用类似的代码在 0:12 拿到了 C 题[12]一血。

[12] https://qoj.ac/contest/782/problem/2162

半平面交: $O(n\log n)$

```
int half(point a){ return a.y > 0||(a.y == 0 && a.x > 0)?1:0; }
bool turn_left(line a, line b, line c) {
  return turn_left(a.s, a.t, line_inter(b, c)); }
bool is_para(line a, line b){return!sgn(det(a.t-a.s,b.t-b.s));}
bool cmp(line a, line b) {
  int sign = half(a.t - a.s) - half(b.t - b.s);
  int dir = sgn(det(a.t - a.s, b.t - b.s));
  if (!dir && !sign) return sgn(det(a.t-a.s, b.t-a.s)) < 0;
  else return sign ? sign > 0 : dir > 0; }
vector <point> hpi(vector <line> h) {
  sort(h.begin(), h.end(), cmp);
  vector <line> q(h.size()); int l = 0, r = -1;
  for(auto &i : h) {
   while (l < r && !turn_left(i, q[r - 1], q[r])) --r;
   while (l < r && !turn_left(i, q[l], q[l + 1])) ++l;
   if (l <= r && is_para(i, q[r])) continue;
   q[++r] = i; }
  while (r - l > 1 && !turn_left(q[l], q[r - 1], q[r])) --r;
  while (r - l > 1 && !turn_left(q[r], q[l], q[l + 1])) ++l;
  if(r - l < 2) return {};
  vector <point> ret(r - l + 1);
  for(int i = l; i <= r; i++)
    ret[i - l] = line_inter(q[i], q[i == r ? l : i + 1]);
  return ret;
}
```

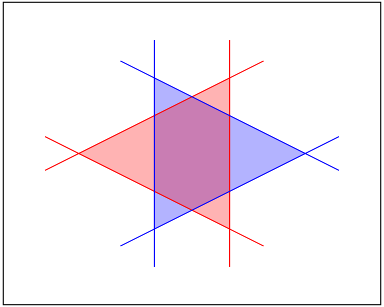
（抱歉压行，因为不压就放不下了。）

- cmp 表示直线的比较函数。由于范围是 360°，应当划分为两个半平面，然后再用叉积排序。对于平行的限制，需要让最紧的限制在第一个。
- is_para 比较两条直线是否平行，用于跳过同方向更松的限制。
- turn_left(a, b, c) 表示 a 和 b 的交点是否在 c 左侧。

半平面交：实现细节

一定要加框！

算法无法区分开集和空集。除非保证能形成闭合区域，一定要在充分大的位置加上边框。



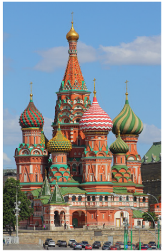
- 此时，判断是否开集 $\Leftrightarrow$ 是否交出来有边框上的部分。

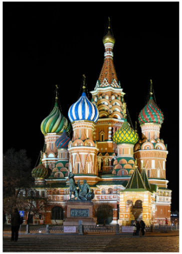
Domes[13]

你在红场上对着 n 个圆顶拍照片。从广场的不同角度拍照，圆顶的顺序会发生变化。

给定一个排列，请计算红场上能得到圆顶按照这个顺序从左到右的拍照点构成的面积。
红场视为一个二维平面上的 $(0,0)-(dx,dy)$ 的方形区域，圆顶视为红场内的一个点，拍照视为过拍照点的一个半平面投影。

- $n \le 100, 0 \le x, y, dx, dy \le 10^5$
- 精度要求：相对或绝对 10^{-3}





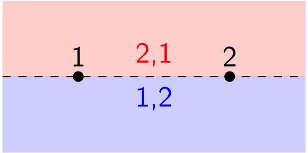
[13] ICPC WF Moscow 2020 C, https://qoj.ac/contest/782/problem/2162

Domes

考虑两个点的时候怎么做。

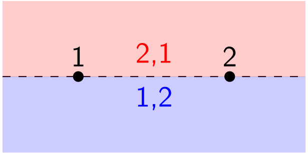
Domes

考虑两个点的时候怎么做。



Domes

考虑两个点的时候怎么做。



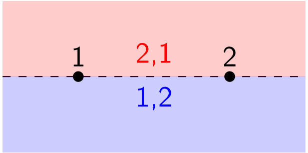
算法

找到代表每对点之间限制的半平面，随后对这些半平面以及代表广场限制的半平面求交。

如何证明正确性？

Domes

考虑两个点的时候怎么做。



算法

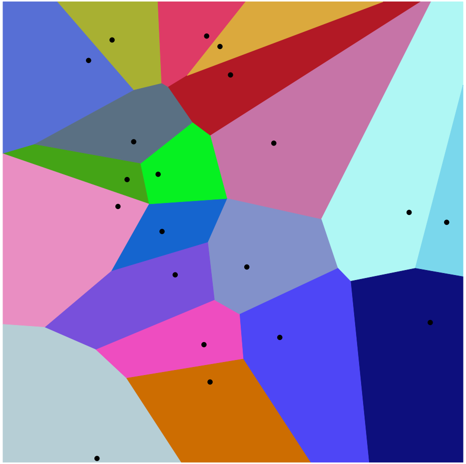
找到代表每对点之间限制的半平面，随后对这些半平面以及代表广场限制的半平面求交。

如何证明正确性？~~（送一下就过了）留作思考题~~

- 这个题是半平面交不需要加框的例子——原因是已经加好了。

Voronoi 图

给定 n 个点，求出对于每个点，以他作为最近点的区域。



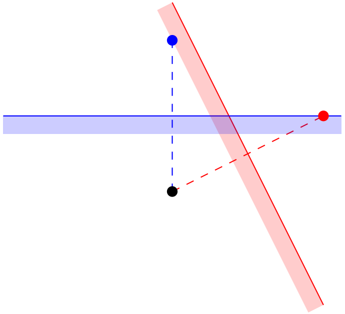
https://en.wikipedia.org/wiki/Voronoi_diagram#/media/File:Euclidean_Voronoi_diagram.svg

Voronoi 图

给定 n 个点，求出对于每个点，以他作为最近点的区域。

$O(n^2 \log n)$ 算法

对于每个点，求出其他点到自己的连线的中垂线，将所有中垂线靠自己一侧求交。



- 这个做法对于最近点和最远点是类似的，因此我们可以类似定义最远点 Voronoi 图。

Voronoi 图

给定 n 个点，求出对于每个点，以他作为最近点的区域。

$O(n^2 \log n)$ 算法

对于每个点，求出其他点到自己的连线的中垂线，将所有中垂线靠自己一侧求交。

$O(n^2)$ 算法

使用切凸包的 $O(n^2)$ 半平面交算法，对于每个点以随机的顺序去切凸包。
这么做，期望复杂度会是（令人惊讶的）$O(n^2)$ 的。[a]

[a] 还是钱哥教我的，证明请见钱哥博客：https://www.cnblogs.com/skip2004/p/17831417.html

- 为什么能够做到 $\sum_{i=1}^{n} O(n^2) = O(n^2)$？
- 一个必要但不充分的证据是平面图的欧拉公式：$V - E + F = 2$。

平面图欧拉公式：$V - E + F = 2$

$$\text{顶点数} - \text{边数} + \text{面数} = 2.$$

有几个比较重要的推论：

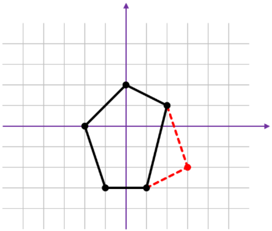
- 平面图上存在一个顶点度数不超过 5；
- n 个点的平面图上最多有 $3n - 6$ 条边；
  - 类似地，可以推出点、线、面三者两两存在线性限制。
- 皮克定理：对于格点简单多边形，若其内部有 i 个格点，边界上有 b 个格点，那么面积满足

$$S = i + \frac{b}{2} - 1.$$

Convex Hull Extension[14]

给定一个整数顶点的严格凸包，问有多少个整点满足：

- 严格在凸包外；
- 能将原本有 n 个顶点的凸包扩展成 $n+1$ 个点的严格凸包。



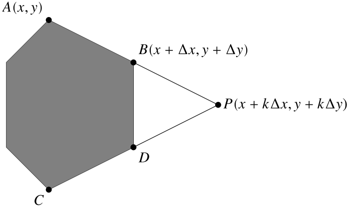
如果有无数解，输出 `infinitely many`。

- $n \le 50, |x|, |y| \le 1000$

[14] ICPC ECNA 2023 C, https://qoj.ac/contest/1406/problem/7693

Convex Hull Extension

满足条件的点在相邻的边夹住的区域。

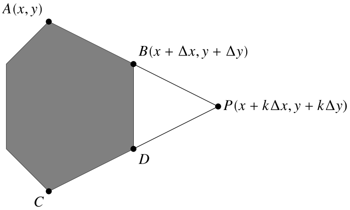


标算

暴力数三角形内点数。整点直线斜率至少为 $O(1/|x|)$，因此总复杂度大约是 $O(n|x|^2)$。由于 $n \leq 50, |x|, |y| \leq 1000$，可以通过。

Convex Hull Extension

满足条件的点在相邻的边夹住的区域。



标算

暴力数三角形内点数。整点直线斜率至少为 $O(1/|x|)$，因此总复杂度大约是 $O(n|x|^2)$。由于 $n \le 50, |x|, |y| \le 1000$，可以通过。

Hacked! 可以构造出斜率为 $1/1000 - 1/1001 \approx 10^{-6}$ 的斜率差，从而使得值域达到 10^9。[15]

[15] https://zhtluo.com/cp/rant-on-incorrect-ecna-2023-c-computational-geometry.html

Convex Hull Extension

类欧几里得算法

类欧可以 $O(\log|x|)$ 计算直线下整点，因此可以计算出三角形面积内整点数量。

我写了一半放弃了。会爆 `int128`。难写程度感觉达到了 dls 的大作[16]的 $1/4$。

[16] ICPC Jinan F 2022, https://qoj.ac/contest/1053/problem/5142

Convex Hull Extension

类欧几里得算法

类欧可以 $O(\log |x|)$ 计算直线下整点，因此可以计算出三角形面积内整点数量。

我写了一半放弃了。会爆 `int128`。难写程度感觉达到了 dls 的大作[16]的 $1/4$。

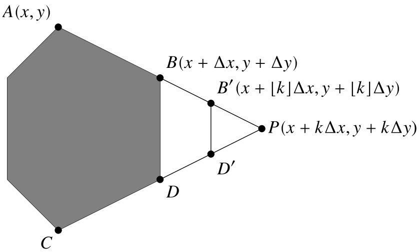
皮克定理

使用皮克定理优化暴力。

将边沿射线方向延长尽可能多整数倍。使用皮克定理计算 $BDD'B'$ 中的格点数。此时，

$$|B'P| < |AB|, |D'P| < |CD|, |B'D'| \leq |BD|,$$

因此可以在原值域大小的复杂度内枚举 x 或者 y 轴计算 $B'D'P$ 点数。复杂度 $O(n|x|)$。

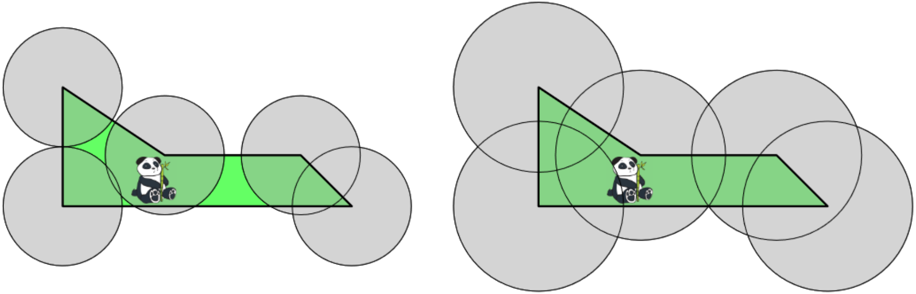


[16] ICPC Jinan F 2022, https://qoj.ac/contest/1053/problem/5142

Panda Preserve[17]

熊猫保护区的边界是一个简单多边形。在每个顶点有一个信号发射器，能够覆盖一个半径为 r 圆形的范围。现在让你找到一个最小的 r 使得所有圆的并能将整个多边形区域覆盖。

- $n \le 2000, |x|, |y| \le 10^4$
- 精度要求：相对或绝对 10^{-6}



(a) An insufficient range for covering the park. (b) The minimal range for covering the park.

[17]ICPC WF 2018 Beijing G, https://qoj.ac/contest/425/problem/2433

Panda Preserve

当 r 越来越大时，多边形会被渐渐覆盖。

Panda Preserve

当 r 越来越大时，多边形会被渐渐覆盖。考察最后一个被覆盖的多边形内的点。

- 最后一个被覆盖的多边形内的点到最近顶点的距离恰好是 r。

Panda Preserve

当 r 越来越大时，多边形会被渐渐覆盖。考察最后一个被覆盖的多边形内的点。

- 最后一个被覆盖的多边形内的点到最近顶点的距离恰好是 r。
- 且其他所有多边形内的点到最近顶点距离不超过 r。

Panda Preserve

当 r 越来越大时，多边形会被渐渐覆盖。考察最后一个被覆盖的多边形内的点。

- 最后一个被覆盖的多边形内的点到最近顶点的距离恰好是 r。
- 且其他所有多边形内的点到最近顶点距离不超过 r。

算法

求出顶点的 Voronoi 图。考察所有在多边形内的 Voronoi 图面的顶点，以及多边形和 Voronoi 图面的交点，在这些点中求出离最近点最远的距离。

- 交点是好求的：拿多边形的边去切 Voronoi 图的面（凸包）就行了。
- 如何判断一个点是否在简单多边形内？

点在多边形内

与点在凸包内类似，我们有从极角和水平坐标系两种处理这个问题的方法。

卷绕数法 (Winding Number)

从点往多边形上拉一条细绳，然后绳头绕多边形走一圈。
如果起点在这一圈自转了一周，那么就在多边形里。

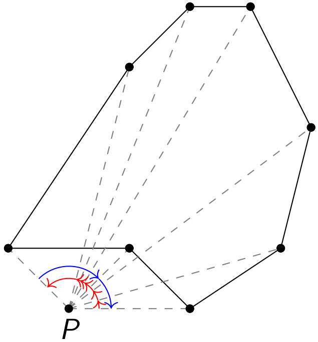
射线法

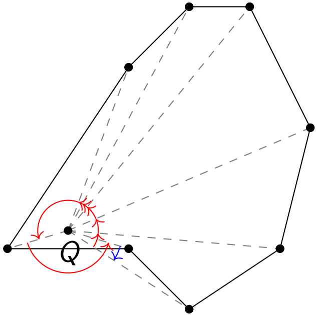
从这个点向无限远处射一条（具有穿透力的）激光。
如果穿越边界次数为奇数次，那么就在多边形里。

点在多边形内

卷绕数法 (Winding Number)

从点往多边形上拉一条细绳，然后绳头绕多边形走一圈。
如果起点在这一圈自转了一周，那么就在多边形里。





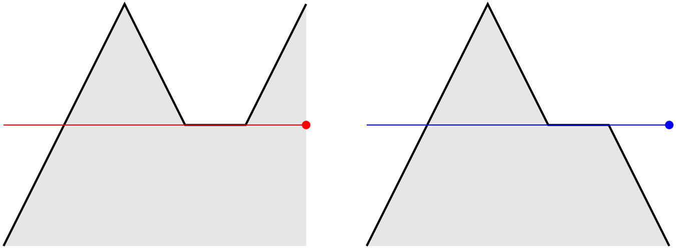
- 特殊处理边界上的点，防止 `NaN`。
- 存在整数实现，但是通常没必要。

点在多边形内

射线法

从这个点向无限远处射一条（具有穿透力的）激光。
如果穿越边界次数为奇数次，那么就在多边形里。

需要处理好从边界上穿出去的情况。



- 一种解决办法：将平行的直线看做倾角 ε 的来处理。

Panda Preserve: 算法细节

熊猫保护区的边界是一个简单多边形。在每个顶点有一个信号发射器，能够覆盖一个半径为 r 圆形的范围。现在让你找到一个最小的 r 使得所有圆的并能将整个多边形区域覆盖。

算法

求出顶点的 Voronoi 图。考察所有在多边形内的 Voronoi 图面的顶点，以及多边形和 Voronoi 图面的交点，在这些点中求出离最近点最远的距离。

- 求 V 图：使用半平面交 $O(n^2 \log n)$ 或者 $O(n^2)$。
- 交点：使用多边形的边去切 Voronoi 图的面。
- 顶点：使用点在多边形内算法进行判定，对于在多边形内的更新答案。

总复杂度：V 图 $+O(n^2)$。

Panda Preserve: Revisited

熊猫保护区的边界是一个简单多边形。你可以在一个任意的位置设置一个信号发射器，能够覆盖一个半径为 r 圆形的范围。现在让你找到一个最小的 r 使得所有圆的并能将整个多边形区域覆盖。

- $n \le 2000, |x|, |y| \le 10^4$
- 精度要求：相对或绝对 10^{-6}

Panda Preserve: Revisited

熊猫保护区的边界是一个简单多边形。你可以在一个任意的位置设置一个信号发射器，能够覆盖一个半径为 r 圆形的范围。现在让你找到一个最小的 r 使得所有圆的并能将整个多边形区域覆盖。

- $n \le 2000, |x|, |y| \le 10^4$
- 精度要求：相对或绝对 10^{-6}

算法：Voronoi 图

求出最远点 Voronoi 图，然后求每个面到对应最远点的最小距离。
时间复杂度：$O(n^2)$。

当然能做，但是好像是不是有点麻烦了？

Panda Preserve: Revisited

熊猫保护区的边界是一个简单多边形。你可以在一个任意的位置设置一个信号发射器，能够覆盖一个半径为 r 圆形的范围。现在让你找到一个最小的 r 使得所有圆的并能将整个多边形区域覆盖。

- $n \leq 2000, |x|, |y| \leq 10^4$
- 精度要求：相对或绝对 10^{-6}

算法：Voronoi 图

求出最远点 Voronoi 图，然后求每个面到对应最远点的最小距离。
时间复杂度：$O(n^2)$。

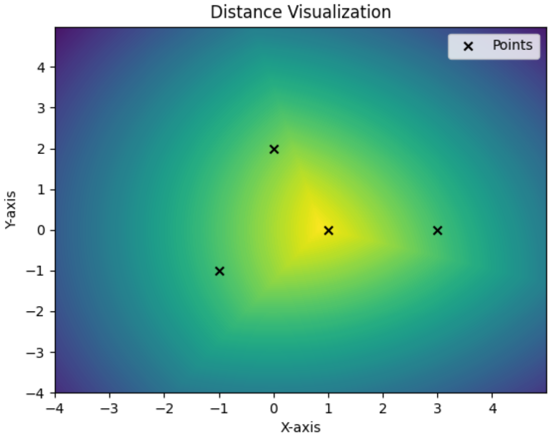
当然能做，但是好像是不是有点麻烦了？

题目翻译：最小覆盖圆

充分性：如果所有顶点被覆盖了，那么他们组成的凸包也一定被覆盖了。
必要性：存在一个顶点没有被覆盖，自然没有覆盖完整个简单多边形。

最小覆盖圆

我在第一次做这个题的时候，写了一个退火过了，觉得自己很牛。
事实上不是我牛，而是爬山也能过。



最小覆盖圆

对于凸函数，爬山不失为一种策略，但是三分通常更好用。

- 一个道理是，爬山需要调整参数等等，写完可能 WA/TLE；
- 而对于三分，我们能够分析他的可靠性和复杂度。

对于二维的凸函数，如何进行三分呢？

最小覆盖圆

对于凸函数，爬山不失为一种策略，但是三分通常更好用。

- 一个道理是，爬山需要调整参数等等，写完可能 WA/TLE；
- 而对于三分，我们能够分析他的可靠性和复杂度。

对于二维的凸函数，如何进行三分呢？

三分套三分

对 x 进行三分，此时需要知道 $g(x^*) = \min_y f(x^*, y)$ 的值，这个值对 y 进行三分得到。

如何实现呢？

最小覆盖圆：三分实现

请指出以下这份实现可以改进的地方。

```
const LD eps = 1e-6;
LD solve_y(LD x) {
    LD l = -1e4, r = 1e4;
    while (r - l > eps) {
        LD lmid = (l * 2 + r) / 3;
        LD rmid = (l + r * 2) / 3;
        if (eval(x, lmid) < eval(x, rmid)) // eval 为 O(n) 求最远距离
            r = mid;
        else
            l = mid;
    }
    return eval(x, (l + r) / 2);
}
LD solve_x() {
    // 类似 solve_y，调用 solve_y 进行三分
    ...
```

最小覆盖圆：三分实现

`while (r - l > eps)` 可能导致死循环。

- 在 l, r 绝对值较大时，浮点数的分度值可能小于 `eps`，导致循环无法得到新值。
- 一种修改是改为固定次数：`for (int T = 50; T; T--)` 来控制相对误差为 $\left(\frac{2}{3}\right)^{50}$。

最小覆盖圆：三分实现

`while (r - l > eps)` 可能导致死循环。

- 在 l, r 绝对值较大时，浮点数的分度值可能小于 `eps`，导致循环无法得到新值。
- 一种修改是改为固定次数：`for (int T = 50; T; T--)` 来控制相对误差为 $\left(\frac{2}{3}\right)^{50}$。

三等分点效率上没有优势。

- 理想情况下，三分越接近二分越好；
- 但是过于靠近 $1/2 \pm \varepsilon$ 可能导致这个“求导”操作精度太差；
- 除以 3 本身就是一个掉精度的操作，但这个操作并没有换来任何优势；
- 一种可能的修改方式是修改为类似 $\frac{7}{16}$ 和 $\frac{9}{16}$ 分位点。

最小覆盖圆：黄金三分

为了达到 10^{-10} 的相对精度，我们需要对每一位进行至少 $\log_2 10^{10} \approx 34$ 次三分操作，每次操作需要求两个点值。
在三分套三分时，我们所需的次数要乘起来，达到 $34 \times 2 \times 34 \times 2$ 次，注意到每次求两个点值连乘起来让效率变差了 4 倍。可以规避掉吗？

最小覆盖圆：黄金三分

为了达到 10^{-10} 的相对精度，我们需要对每一位进行至少 $\log_2 10^{10} \approx 34$ 次三分操作，每次操作需要求两个点值。
在三分套三分时，我们所需的次数要乘起来，达到 $34 \times 2 \times 34 \times 2$ 次，注意到每次求两个点值连乘起来让效率变差了 4 倍。可以规避掉吗？

黄金三分

令 $\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887\ldots$。每次取 $1 - \frac{1}{\varphi}, \frac{1}{\varphi}$ 两个点进行三分。

好处是？

最小覆盖圆：黄金三分

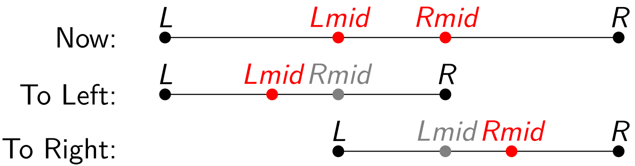
为了达到 10^{-10} 的相对精度，我们需要对每一位进行至少 $\log_2 10^{10} \approx 34$ 次三分操作，每次操作需要求两个点值。
在三分套三分时，我们所需的次数要乘起来，达到 $34 \times 2 \times 34 \times 2$ 次，注意到每次求两个点值连乘起来让效率变差了 4 倍。可以规避掉吗？

黄金三分

令 $\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887\ldots$。每次取 $1 - \frac{1}{\varphi}, \frac{1}{\varphi}$ 两个点进行三分。

好处是？

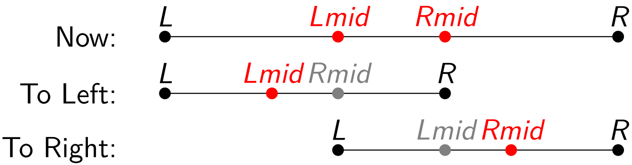
- 总能省下一次求值。



最小覆盖圆：黄金三分

黄金三分

令 $\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887\ldots$。每次取 $1 - \frac{1}{\varphi}, \frac{1}{\varphi}$ 两个点进行三分。



对比 $\frac{1}{2} \pm \varepsilon$ 三分与黄金三分，我们发现黄金三分的效率更高：

$$2\log_2 X = \log_{\sqrt{2}} X > \log_{\varphi} X$$

代入 $X = 10^{10}$: $\quad 68 > 48$

最小覆盖圆：随机增量法

- 数据范围: $n \le 10^5$

最小覆盖圆：随机增量法

- **数据范围**: $n \leq 10^5$

```
circle min_circle (vector <point> p) {
    circle ret({0, 0}, 0);
    shuffle (p.begin(), p.end(), rng); // 随机排列，重要
    for (int i = 0; i < p.size(); ++i) if (!in_circle(p[i], ret)) {
        ret = circle (p[i], 0);
        for (int j = 0; j < i; ++j) if (!in_circle(p[j], ret)) {
            ret = make_circle (p[j], p[i]);
            for (int k = 0; k < j; ++k) if (!in_circle(p[k], ret))
                ret = make_circle (p[i], p[j], p[k]);
        }
    } return ret; }
```

尽管是三重循环，算法期望复杂度是 $O(n)$ 的。

最小覆盖圆：随机增量法

- 数据范围：$n \le 10^5$

```cpp
circle min_circle (vector <point> p) {
    circle ret({0, 0}, 0);
    shuffle (p.begin(), p.end(), rng); // 随机排列，重要
    for (int i = 0; i < p.size(); ++i) if (!in_circle(p[i], ret)) {
        ret = circle (p[i], 0);
        for (int j = 0; j < i; ++j) if (!in_circle(p[j], ret)) {
            ret = make_circle (p[j], p[i]);
            for (int k = 0; k < j; ++k) if (!in_circle(p[k], ret))
                ret = make_circle (p[i], p[j], p[k]);
        }
    } return ret; }
```

尽管是三重循环，算法期望复杂度是 $O(n)$ 的。

- 随机排列后，第 x 不在前 $x-1$ 个点的最小覆盖圆里的概率是 $O(1/x)$ 的。

最小覆盖圆：随机增量法

- 数据范围：$n \leq 10^5$

```cpp
circle min_circle (vector <point> p) {
    circle ret({0, 0}, 0);
    shuffle (p.begin(), p.end(), rng); // 随机排列，重要
    for (int i = 0; i < p.size(); ++i) if (!in_circle(p[i], ret)) {
        ret = circle (p[i], 0);
        for (int j = 0; j < i; ++j) if (!in_circle(p[j], ret)) {
            ret = make_circle (p[j], p[i]);
            for (int k = 0; k < j; ++k) if (!in_circle(p[k], ret))
                ret = make_circle (p[i], p[j], p[k]);
        }
    } return ret; }
```

尽管是三重循环，算法期望复杂度是 $O(n)$ 的。

- 随机排列后，第 x 不在前 $x-1$ 个点的最小覆盖圆里的概率是 $O(1/x)$ 的。
- 每个 i 有 $O(1/i)$ 的概率枚举 i 个 j，每个 j 有 $O(1/j)$ 的概率枚举 j 个 k。

总结

- 我们介绍了一些几何工具，以及对应实现。
- 一个问题可能有多种解决方法。
- 选择正确的工具解决问题比会做题本身更重要。
- 判断题目难度比会做题本身更重要。

比赛建议：一些烂梗

- 「菜就多练」
  - 如果有志于在比赛里解决几何题，多训练有几何的场。
  - 如果只补过题而没有在训练里通过几何题，比赛现编是不现实的。
- 「输不起就别玩」
  - 开几何通常是比赛的胜负手。
  - 这个决策必须有着绝对信任和责任。
  - 不开几何比乱开几何好。
    - ~~「那道几何毁了我的 XCPC 梦」~~

比赛建议：一些胡言乱语

- 实现难度、精度、运行效率、泛用性是无法兼得的。
  - 最短的、最快的代码不一定是最好的代码。
  - 自己熟悉，并且能做出有效修改的才是最好的代码。
- 计算几何题通常也会有大量边界情况与分类讨论。
  - 因此，一个通常正确的策略也许是推迟实现几何题。
  - 反其道而行之：几何题容易抢一血气球？
- 在超越自己掌握的模板库时，一个通常正确的决策是放弃几何题。
  - 在水平 $< L$ 或者 $> R$ 的时候，一道题目不会成为队伍的负担。
  - 在水平 $< R$ 但以为自己 $> R$ 时，容易引发事故。
  - 即使掌握，也要当做带有有随机性、有精度问题、有分类讨论的大模拟题来谨慎处理。
    - 即使没用到随机算法，随机性可能来自：数据太弱、太强、有错、评测机环境……
  - 准确感受自己的水平！

Thank you!

Epilogue: Almost Convex[18]

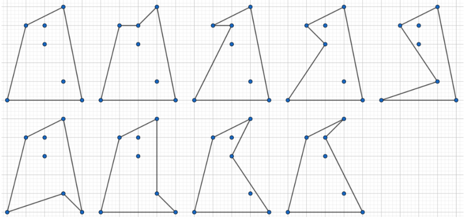
还记得今天的目标 3 吗？

对于 n 个不存在三点共线的点集，令凸包上的顶点数为 $|R|$。求：

- 顶点数 $\leq |R| + 1$，
- 所有 n 个点都在内部或者边界

的简单多边形数量。

- $n \leq 2000, |x|, |y| \leq 10^6$



[18]ICPC Jinan 2023 M, https://contest.ucup.ac/contest/1472/problem/7906?v=1