OI 计算几何 Part 3 杂项

Computational Geometry in OI

daklqw

Zhenhai High School

August 1, 2024

Outline

对于规则的图形，其面积是容易求得的。但是对于复杂的形状实际上是难以方便计算的。这时候就需要用到一些数学工具。

辛普森积分

Example

给定函数 $f(x)$，求 $f(x)$ 在区间 $[a,b]$ 上的积分。

Proposition (Simpson's rule)

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{6}\left(f(a)+f\left(\frac{a+b}{2}\right)+f(b)\right).$$

其误差为 $-\dfrac{(b-a)^5}{2880}f^{(4)}(\xi)$.

自适应辛普森积分

可以看到仅单次运用不够平常使用。一般的想法是将区间分为多段，并且将分段积分的值累加起来。

```
double Simpson(double a, double b) {
  double f = calc(a, b);
  double m = (a + b) / 2;
  double l = calc(a, m), r = calc(m, r);
  if (std::abs(l + r - f) < eps) return f;
  return Simpson(a, m) + Simpson(m, b);
}
```

自适应辛普森积分

考虑例子 $f(x)=\sin^2 x$，积分区间为 $[0,4\pi]$。

则由于 $f(0)=f(\pi)=f(2\pi)=f(3\pi)=f(4\pi)=0$，因此自适应辛普森算法 `Simpson(0, 4*Pi)` 会得到 0。

这说明，自适应辛普森可能难以面对一些极端情况。需要对图形做很好的分割才能得到较为精确的解。

Example (NOI2005 月下柠檬树)

给一棵树，由若干拼接起来的圆台表示，给定光线的角度（平行光），求其地上的阴影面积，答案保留两位小数。

圆台个数 ≤ 500。

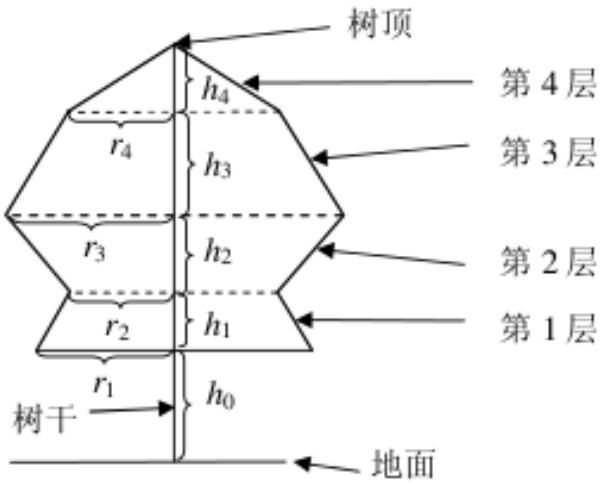


图 1　柠檬树的纵剖面图

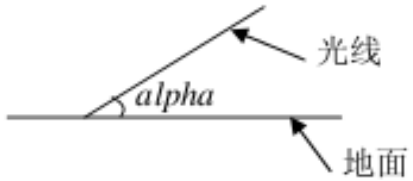


图 2 月光角度示意图

NOI2005 月下柠檬树

Algorithm

圆投影下来仍然是一个圆，因此圆台投影是两个圆及其公切线构成的形状。

其形状是典型的不规则形状。并且对精度的要求不高，所以使用数值积分是很合适的。

以树干投影为 x 轴，令 $f(x_0)$ 为 x_0 处阴影 y 坐标的最大值，则 $2\int_{-\infty}^{\infty} f(x)\mathrm{d}x$ 即是答案。

圆的面积并

Example

给定一堆圆，求他们并集的面积，保证区域连通，数据随机生成。

$n \leq 500$。

Algorithm

这同样是不规则图形，因此也可以用数值积分。

定义 $f(x_0)$ 为直线 $x = x_0$ 与所有圆相交的长度（因此 $f(x)$ 可以 $O(n)$ 计算），对 f 进行积分即可。

格林公式

Theorem (Green's Theorem)

$$\oint_C (L\mathrm{d}x + M\mathrm{d}y) = \iint_D \left(\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}\right) \mathrm{d}x\mathrm{d}y$$

即可以将面积积分和曲线积分互相转化。只需要得到边界形状即可计算面积。

格林公式-计算面积

$$\oint_C (L\mathrm{d}x + M\mathrm{d}y) = \iint_D \left(\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}\right) \mathrm{d}x\mathrm{d}y$$

构造 $\frac{\partial M}{\partial x} = 1, \frac{\partial L}{\partial y} = -1$，即 $M = x, L = -y$。此时等式右侧变成了面积的两倍。

左侧为 $\oint_C (x\mathrm{d}y - y\mathrm{d}x)$。一般来说，曲线边界都是由规则形状拼接而成。因此考虑对直线段以及圆弧分别计算。

简单多边形面积

Example

给定一个简单多边形，求它的面积。要求线性。

简单多边形面积

Algorithm

考虑线段 $x = x_0 + k\Delta_x t, y = y_0 + k\Delta_y t (t \in [0, t_0])$，**则** $\mathrm{d}x = \Delta_x \mathrm{d}t, \mathrm{d}y = \Delta_y \mathrm{d}t$。

$$
\begin{aligned}
&\int_\ell (x\mathrm{d}y - y\mathrm{d}x) \\
=&\int_0^{t_0} ((x_0 + k\Delta_x t)\Delta_y - (y_0 + k\Delta_y t)\Delta_x)\,\mathrm{d}t \\
=&t_0(x_0\Delta_y - y_0\Delta_x)
\end{aligned}
$$

即 $(x_0, y_0) \times (\Delta_x t_0, \Delta_y t_0)$，**即线段与坐标原点围成的三角形面积的一倍。对所有边计算并累加即可。**

简单多边形面积

这样的计算很像一种容斥：逆时针的边带来正贡献，顺时针的边带来负贡献。

同时，计算与原点的选择无关。

圆的面积并

Example

给定一堆圆，求他们并集的面积。要求 $O(n^2 \log n)$。

Simpson 积分的复杂度中有关于精度的因子。同时其对极端情况的适应性不好，所以尝试使用 Green 公式。

圆的面积并

Algorithm

考虑圆弧 $x = x_0 + r\cos\theta, y = y_0 + r\sin\theta(\theta \in [\theta_0, \theta_1])$，**则** $\mathrm{d}x = -r\sin\theta\mathrm{d}\theta, \mathrm{d}y = r\cos\theta\mathrm{d}\theta$。

$$
\begin{aligned}
&\int_\ell (x\mathrm{d}y - y\mathrm{d}x) \\
&= \int_{\theta_0}^{\theta_1} ((x_0 + r\cos\theta)r\cos\theta + (y_0 + r\sin\theta)r\sin\theta)\,\mathrm{d}\theta \\
&= \int_{\theta_0}^{\theta_1} \left(rx_0\cos\theta + ry_0\sin\theta + r^2\right)\mathrm{d}\theta \\
&= (r^2 + rx_0\sin\theta - ry_0\cos\theta)\Big|_{\theta=\theta_0}^{\theta=\theta_1}
\end{aligned}
$$

圆的面积并

Algorithm (Cont.)

此时，只需要求圆和圆的交点，再对每个圆跑线段覆盖以得出形状的边界（即未被覆盖的地方）。然后对所有的边界圆弧段套用公式。

时间复杂度 $O(n^2 \log n)$。

Outline

点在简单多边形内

Example

给定一个简单多边形和一个点 P，问点 P 是否在多边形内部。

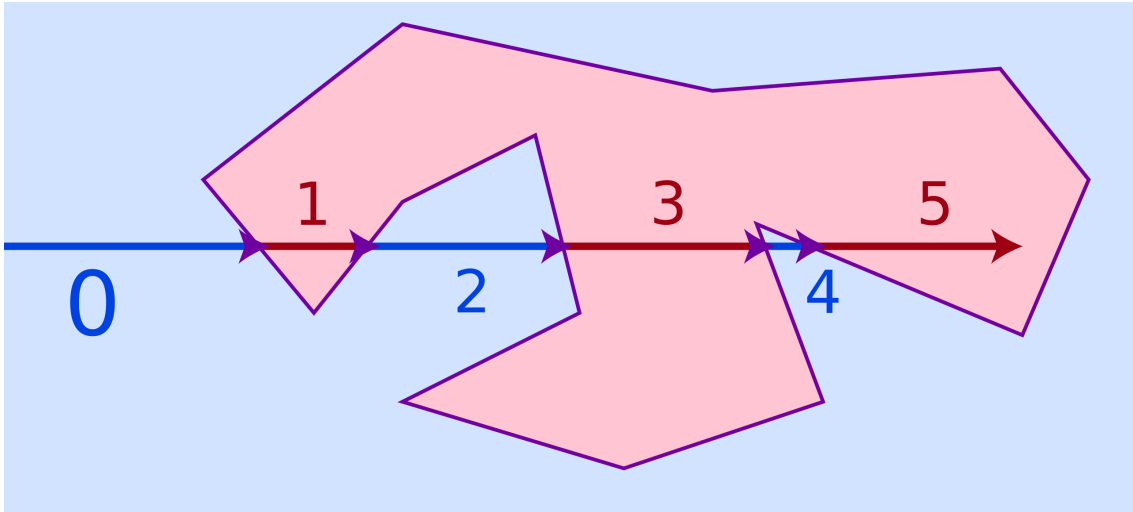
点在简单多边形内

Example

给定一个简单多边形和一个点 P，问点 P 是否在多边形内部。

Algorithm (射线法)

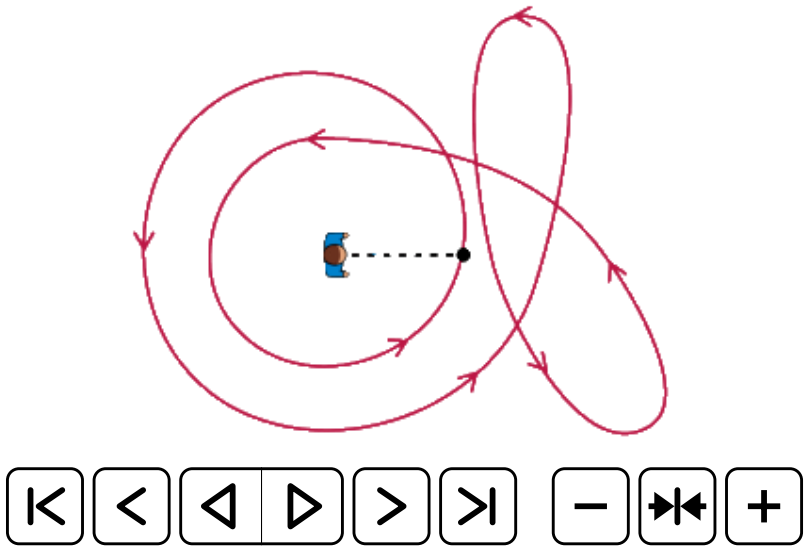
从点 P 发射一条射线，如果射线穿过了多边形的奇数条边，则点在多边形内部。



点在多边形内

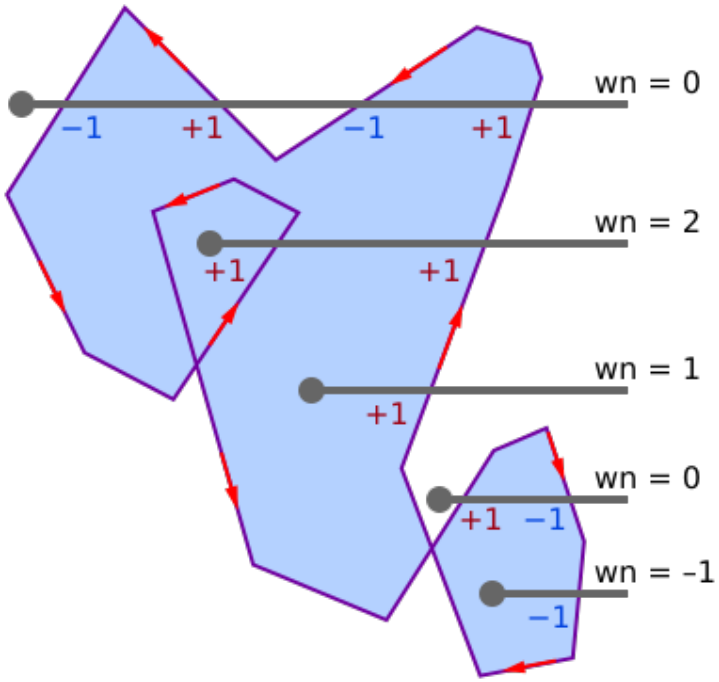
Algorithm (卷绕数算法)

在多边形边上有固定方向移动的一个点 Q，考察 PQ 旋转的度数，如果 Q 遍历完多边形后旋转的圈数（即卷绕数）是 0 则在多边形外，否则在多边形内部。



点在多边形内

另一种计算卷绕数的方法，避免了计算角度。



Outline

平面最近点对

Example

给定平面上 n 个点，找出最近的点对。$n \leq 10^5$。

平面最近点对

Algorithm (分治算法)

分治算法，即将区间分为两侧，分别求解，然后合并。这里约定按 x 轴分割区间，左半边是 x 坐标 $\leq \mathrm{mid}$ 的点，右半边是剩下的点。

假设左区间的最近点对为 d_l，右区间为 d_r，则令 $d = \min(d_l, d_r)$，选取左区间距离右边界 $\leq d$ 的点，和右区间距离左边界 $\leq d$ 的点，得到点集 S。

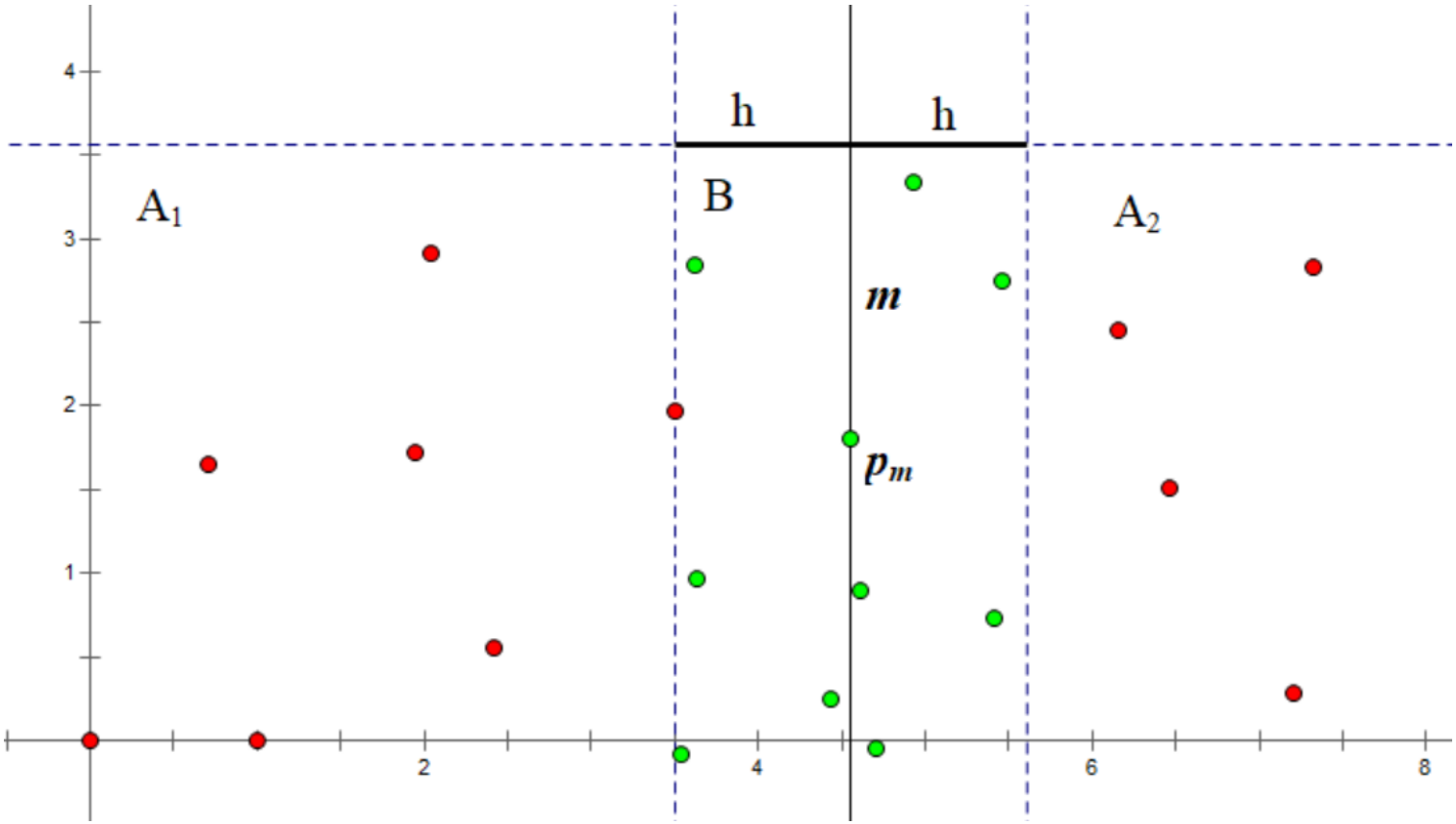
平面最近点对

Algorithm (Cont.)

如果 S 中的点按 y 坐标排序了，则可以对其中每个点 $p \in S$，枚举 $|y_p - y_q| \leq d$ 的 $q \in S$，并更新答案。由于左右区间的最小距离 $\geq d$，因此有效的点个数是 $O(1)$ 的。

为了得到按 y 坐标排序的结果，其实可以在分治的同时做归并排序，即合并的时候同时按 y 坐标归并两组点集。

平面最近点对



平面最近点对

Algorithm (随机算法（常见乱搞，不推荐）)

重复 K 次，每次选择一个随机单位向量 v，让所有点按其在这个向量上投影长度进行排序。然后用每个点与其相邻 $M = O(\log n)$ 个点的距离来更新答案。时间复杂度 $O(Kn \log n)$。

这玩意正确性不好保证（我证不来），比较依赖参数的选择。并且这个算法也没复杂度优势，所以分治算法还是得学会的。

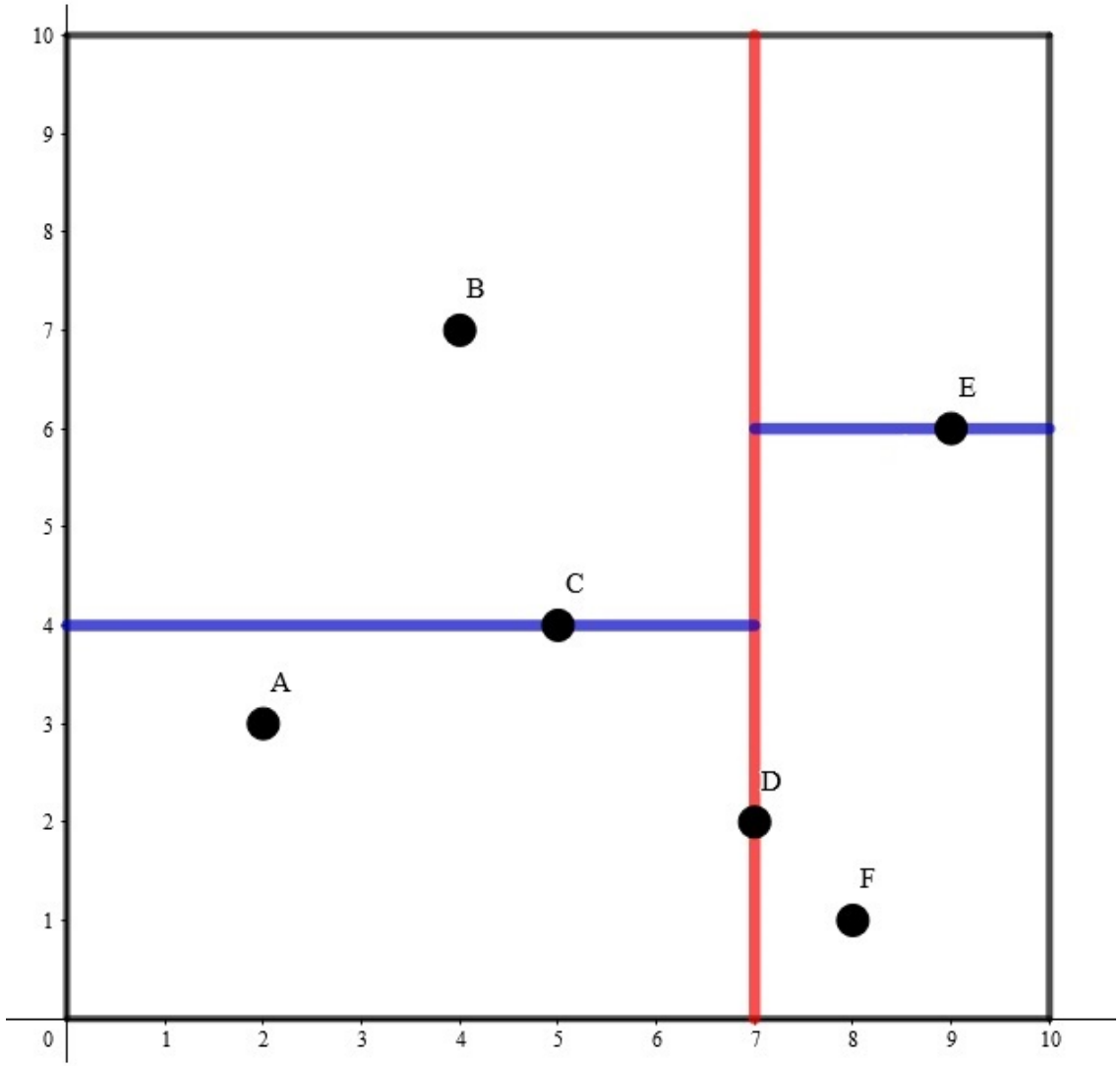
K-D Tree

Algorithm (建树 $O(n \log n)$)

K-D 树将点集按各维分割，以建起一棵树。类似于平衡树，每个节点作为该层用来分割的中位，存储其子树的信息（例如 *Bounding Box*）。每层用来分割的维度是不同的。

```
1 Node* build(vec* l, vec* r, int d) {
2   vec* x = mediam(l, r, d % DIM);
3   Node* lp = build(l, x, d + 1);
4   Node* rp = build(x + 1, r, d + 1);
5   return new Node(*x, lp, rp);
6 }
```

K-D Tree



K-D Tree

Algorithm (动态加点/删点)

加点的操作，就像普通的二叉树一样，每次和当前节点的关键字比较，选择左右子树直到一个可以放置的位置。而删点的话就打个删除标记。但是这就像普通的二叉树一样，会不平衡。因此需要加重构：

类似于替罪羊树，固定一个平衡常数 $\frac{1}{2} < \alpha < 1$，如果一个节点的某个子树大于 α 倍的其本身子树大小，则重构（用 *K-D* 树的 *build* 函数）。

K-D Tree

Algorithm (动态加点/删点)

此时，树的深度不会超过 $O(\log n)$。同时，由于 α 的存在，对于 $O(k)$ 大小的子树只有插入/删除 $O((\alpha - 1/2)k)$ 次后才会重构。

重构的均摊复杂度是 $O(\log^2 n)$。

K-D Tree

Algorithm (范围查询)

每个点维护一个 *Bounding Box*，即维护其子树所有点的每维坐标最小最大值。

此时范围查询即可看 *Bounding Box* 与查询区间：

- 是否相交，不相交则放弃这个子树。
- 是否包含，如果包含则直接使用区间信息。
- 相交却不完全包含，则递归计算。

时间复杂度 $O(kn^{1-\frac{1}{k}})$。二维下是 $O(\sqrt{n})$。

K-D Tree

Algorithm (最近点查询)

同样利用 *Bounding Box*。维护一个最佳答案 d，对于查询点 P，看 *Bounding Box* 是否包含 Q 使得 $|Q-P|<d$。如果是则递归。

对于随机的点分布，这么做是期望 $O(\log n)$ 的。但是最坏情况仍是 $O(n)$。

Outline

矩形面积并

Example

给定 n 个与坐标轴平行的矩形，求它们面积的并。要求 $O(n \log n)$。

由于矩形并的边界可能有 $O(n^2)$ 条边，所以直接使用先前的算法是行不通的。

矩形面积并

Algorithm

从 x 轴的角度看，图形只有 $O(n)$ 个不同的状态。因此可以采用扫描线。需要一个区间数据结构，支持：

- 区间加 1。
- 区间减 1。
- 询问非零位置个数。

注意所有值相同的极长区间，区间操作只会增加或减少 $O(1)$ 个这样的区间。平衡树可以直接实现这样的操作。

矩形面积并

Algorithm (Cont.)

但是在线段树上有更简单的实现方法。

每个点维护一个不被 pushdown 的 lazytag，表示这个区间是否加了数。

此时只需要修改线段树的 update：

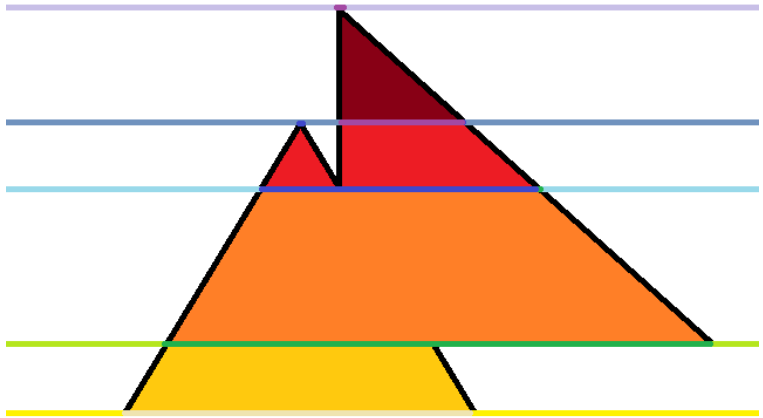
```
1 void update(int u) {
2   if (tag[u] > 0) sum[u] = len[u];
3   else sum[u] = sum[ls[u]] + sum[rs[u]];
4 }
```

三角形面积并

Example

给定 n 个三角形，求它们面积的并。$n \leq 100$。

同样套用扫描线，所有的线段交点处是阶段的改变点。



三角形面积并

Algorithm

求出所有交点，那么每个阶段里应被计算的面积是一堆梯形的组合。

因此在一个阶段中 $x = x_0$ 切到图形长度的函数 $f(x_0)$ 实际上是线性的。

此时区间面积的积分，可以用 *Simpson* 公式完全精确地计算。实际上由于这是线性的，直接取中点就能计算出整段面积。

$f(x_0)$ 是一个线段并问题，因此单次计算 $f(x)$ 要 $O(n \log n)$ 时间。一共有 $O(n^2)$ 个交点，因此时间复杂度 $O(n^3 \log n)$。

线段交点

Example

给定 n 条线段，一共有 m 个交点。要求在 $O((n+m)\log n)$ 时间内输出所有交点。

和先前的扫描线相比，这是个动态计算断点的扫描线。

线段交点

Algorithm

注意到每个阶段每条线段的高度关系是固定的，只有在产生交点后线段的高度关系才会发生变化（交换相邻点）。

同时，交点只会发生在高度相邻的线段之间。因此只需要在插入线段的时候和相邻的线段计算交点。

扫描线转换阶段的时间点除了线段的两个端点，还有线段之间的交点。

通过额外维护一个交点横坐标的优先队列可以知道下个断点在哪。

数点问题

Example (k 维数点)

平面上一共 n 个点 $\in \mathbb{R}^k$，有 Q 个询问：查询有多少点 $(x_1, x_2, \ldots, x_k) \in \mathbb{R}^k$ 满足 $\forall i \in [k], x_i \leq a_i$。

大量区间问题可以转化为数点问题，并且信息只需结合律即可维护。

k 维数点问题可以由多种方法解决。其中大多数是使用能够降维的方法进行堆叠。一般来说，使用这些方法得到的 k 维数点算法的时间复杂度是 $O(n \log^{k-1} n)$。

数点问题

Example (单点修改，区间查询)

维护一个序列，支持单点加上一个数；支持查询序列一个区间数的和。

这实际上是一种二维数点。其中一维是数组的下标，另一维是操作的时间。

实际上区间加，区间查询和也是一种二维数点，因为区间加区间查可以拆分成区间加等差数列单点查询以及单点加区间查询的两个问题。

数点问题

Algorithm (降维方法)

常用的降维方法有：

- 排序（即扫描线）：对应时间这维
- 分治（可堆叠）：通过计算贡献消去一维
- 区间数据结构（一般可堆叠，即树套树）：
  - 树状数组（难堆叠）：常数最小
  - 线段树：常数较好，动态开点空间偏大
  - 平衡树：常数较差，省空间

数点问题

Algorithm (排序)

对询问和点集按某一维坐标排序，并且做扫描线。由于操作顺序解除了这一维的依赖关系，相当于成功降维。

例如二维数点中，询问变成了区间 $y \leq y_0$ 查询，点变成了单点加操作。

数点问题

Algorithm (分治)

分治的关键点在于计算左半区间对右半的贡献。因为分治的这一维依赖关系已经消失，相当于成功降维。

如果剩下的维数大于一维，可以调用低一维的数点算法。

如果剩下的维数只有一维，可以通过类似归并排序的做法，来对左右区间剩下的那维进行排序查询。

Outline

平面图

Definition (Planar Graph)

平面图是能够嵌入到平面的图，即存在一种嵌入方式，使得没有任何边相交。

Theorem

一张图是平面图当且仅当它不含有 K_5 或 $K_{3,3}$ 作为细分的子图。

欧拉平面图定理

Theorem

V 是顶点数量，E 是边的数量，F 是面的数量，对于一个有限、连通的平面图，有：

$$V - E + F = 2.$$

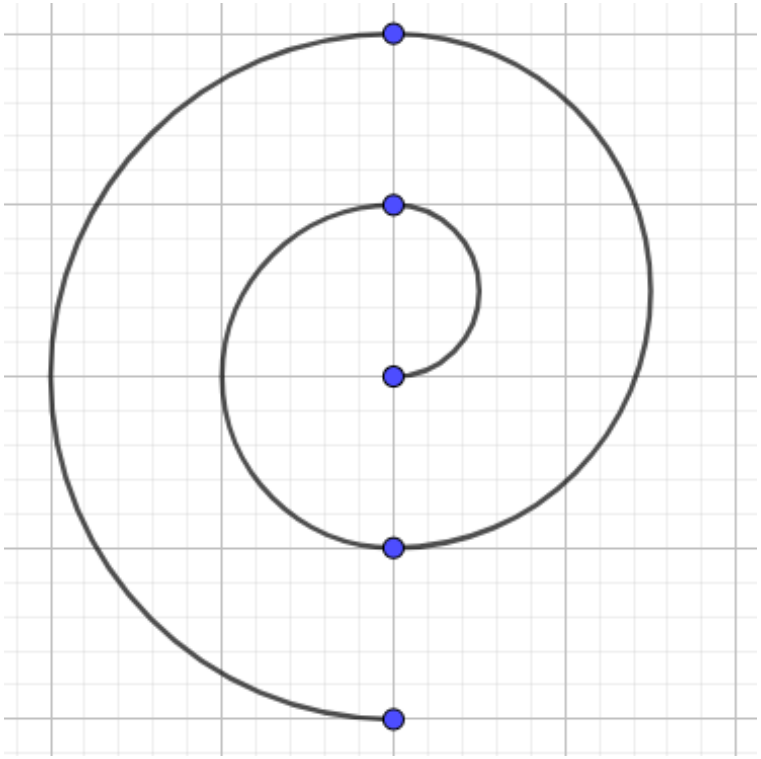
当有 C 个连通块时候，公式变为：

$$V - E + F = C + 1.$$

Example (SOJ 685. 压缩 ▸ SOJ 685)

平面上给定若干不相交的“得便”结构，即若干半圆首尾相接的结构；再给定若干与坐标轴平行的直线，问整个平面被分成多少区域。

半圆个数 ≤ 3000，直线条数 ≤ 500。



SOJ 685. 压缩

Algorithm

这题是基础的直线与圆求交问题。可以发现，欧拉平面图定理中的 V, E 是容易计算的，从这再得出 F。

由于这题的简化，所有的交点都会在直线上。因此只需要在直线上统计点的个数即可。

对偶图

Definition

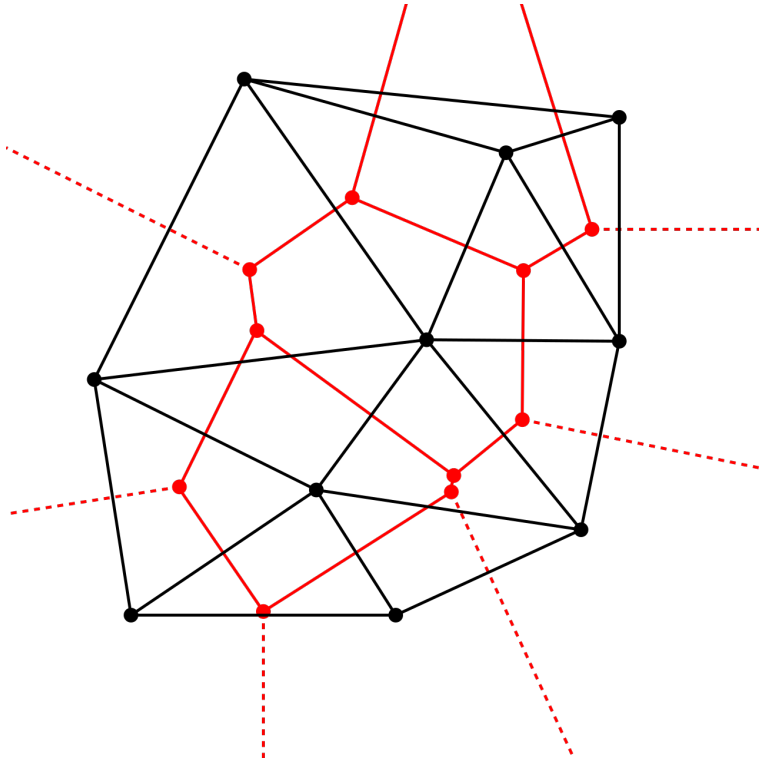
平面图的对偶是

- 面 $\to$ 点：每个面变成一个点。
- 边 $\to$ 边：每条边在对偶图中连接着其两侧的面。
- 点 $\to$ 面：每个点在对偶图中是其原平面图周围的面对应的点围成的面。

三角剖分

Definition (Triangulation)

一个点集的三角剖分是通过给点集加尽量多的边，使其变为一张每个面都是三角形的平面图。



三角剖分-性质

Proposition

按上文定义的三角剖分一定包含点集的凸包。

Proposition

假设有 n 个点，凸包上有 k 个点，则三角剖分一定会得到 $(2n-k-2)$ 个三角形。

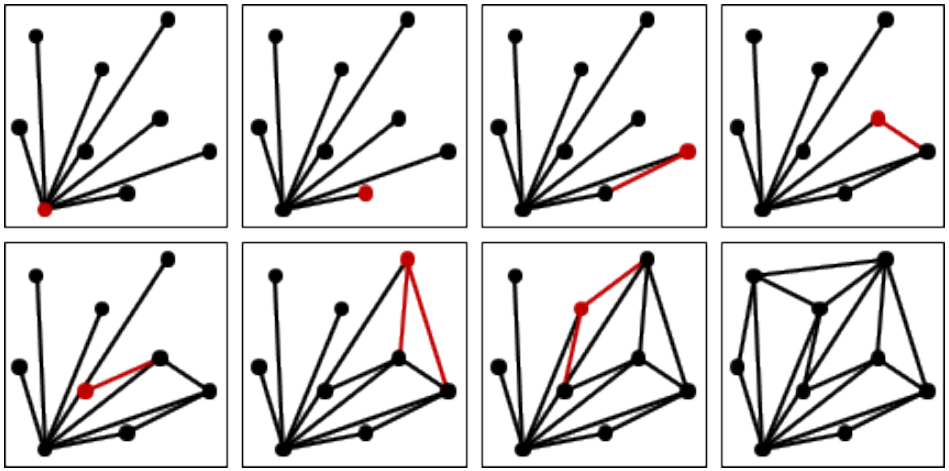
由欧拉平面图定理可以计算出一共有 $(3n-k-3)$ 条边。

构建三角剖分

Algorithm (Graham 扫描法)

实际上 *Graham* 扫描法建立凸包的时候就产生了一组三角剖分。

首先加入基准点到其他所有点的边，共 $n-1$ 条边。在每个点入栈时加边，出栈时加边，此时会有共 $(n-1)+(k-2)+2(n-k)=3n-k-3$ 条边。

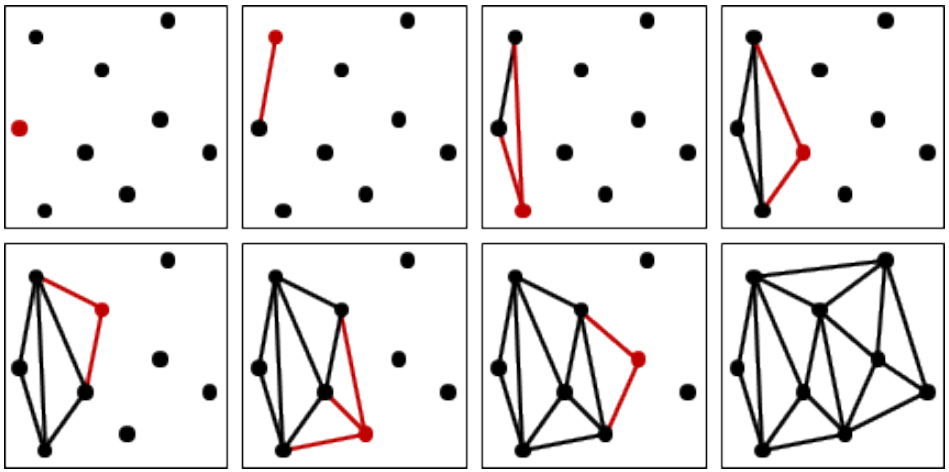


构建三角剖分

Algorithm (增量法)

从左到右加入点，维护右半边的凸壳，每当加入一个新点，尝试尽可能连接凸壳上的点，使得连接出的边和凸壳没有相交。

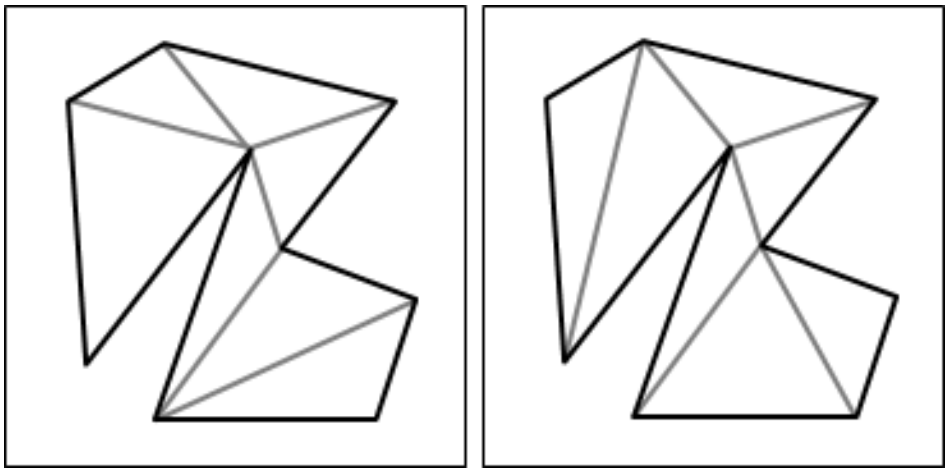
(类似凸包，分离的两组三角剖分可以合并，只用合并公切线以内的部分即可)



简单多边形三角剖分

Definition (Polygon Triangulation)

给定一个简单多边形，要求添加尽量多的边，使得其内部的面全部为三角形。



Delaunay 三角剖分

Definition (Delaunay Triangulation)

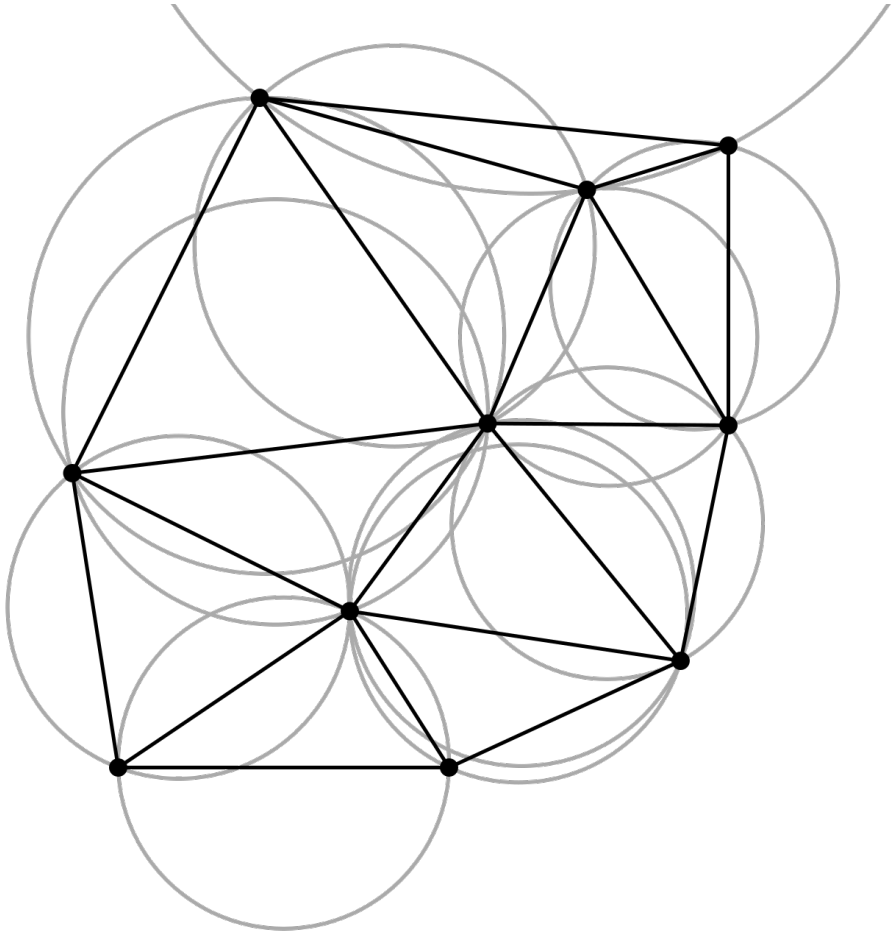
Delaunay 三角剖分是点集的一组三角剖分，满足：

- 任意三角形的外接圆不包含其他点。
- 最大化最小角（所有三角形中的最小角）。

Proposition

如果不存在四点共圆，那么 *Delaunay* 三角剖分是唯一的。

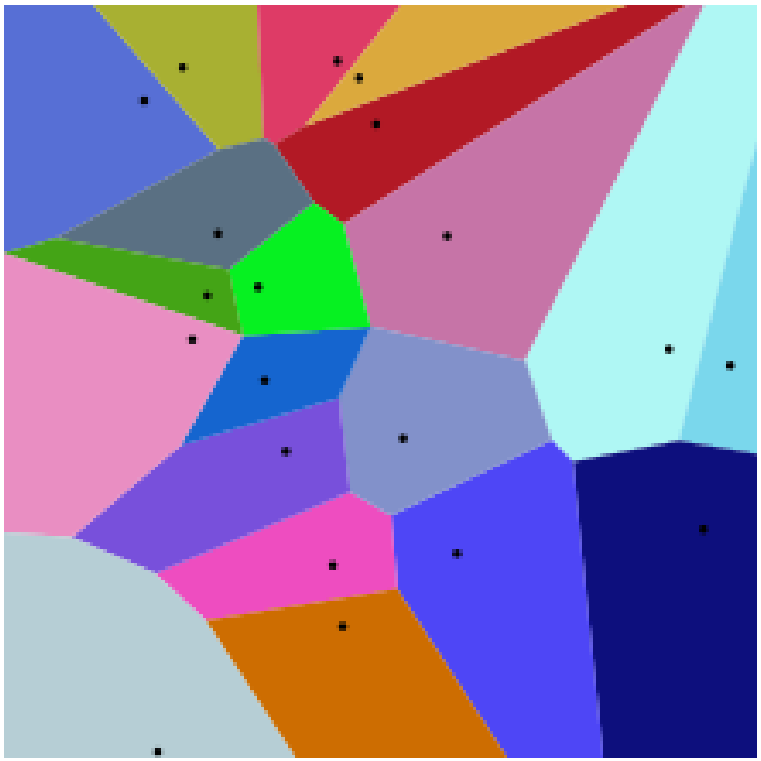
Delaunay 三角剖分



Voronoi 图

Definition (Voronoi Diagram)

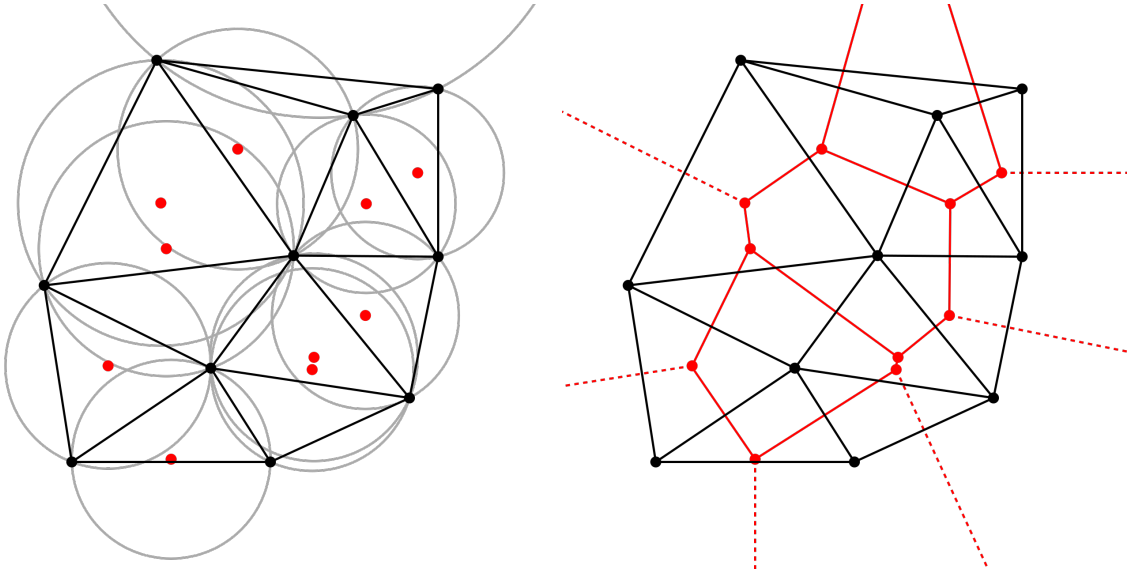
给定一个点集，将平面划分为若干区域，使得平面上的每个点属于其离得最近的点集中的点所在的区域。



Delaunay 三角剖分

Proposition

给定平面上点集，其 *Voronoi* 图是 *Delaunay* 三角剖分的对偶图。其中 *Delaunay* 三角剖分的外接圆心对应 *Voronoi* 图的顶点。



Delaunay 三角剖分

Algorithm (判断点在三角形外接圆内)

对于逆时针排列的三点 A, B, C，对于点 D，如果行列式

$$\det \begin{pmatrix} A_x & A_y & A_x^2 + A_y^2 & 1 \\ B_x & B_y & B_x^2 + B_y^2 & 1 \\ C_x & C_y & C_x^2 + C_y^2 & 1 \\ D_x & D_y & D_x^2 + D_y^2 & 1 \end{pmatrix} > 0$$

则 D 在三角形 ABC 的外接圆内。

$$\det \begin{pmatrix} A_x & A_y & A_x^2+A_y^2 & 1 \\ B_x & B_y & B_x^2+B_y^2 & 1 \\ C_x & C_y & C_x^2+C_y^2 & 1 \\ D_x & D_y & D_x^2+D_y^2 & 1 \end{pmatrix}$$

$$= \det \begin{pmatrix} A_x-D_x & A_y-D_y & (A_x-D_x)^2+(A_y-D_y)^2 \\ B_x-D_x & B_y-D_y & (B_x-D_x)^2+(B_y-D_y)^2 \\ C_x-D_x & C_y-D_y & (C_x-D_x)^2+(C_y-D_y)^2 \end{pmatrix}$$

Delaunay 三角剖分

Definition

全局 Delaunay 条件：任意三角形的外接圆不包含其他圆的点。

局部 Delaunay 条件：对于边 bc，相邻三角形 Δabc 和 Δbcd 满足：Δabc 的外接圆不包含 d，Δbc 的外接圆不包含 a。

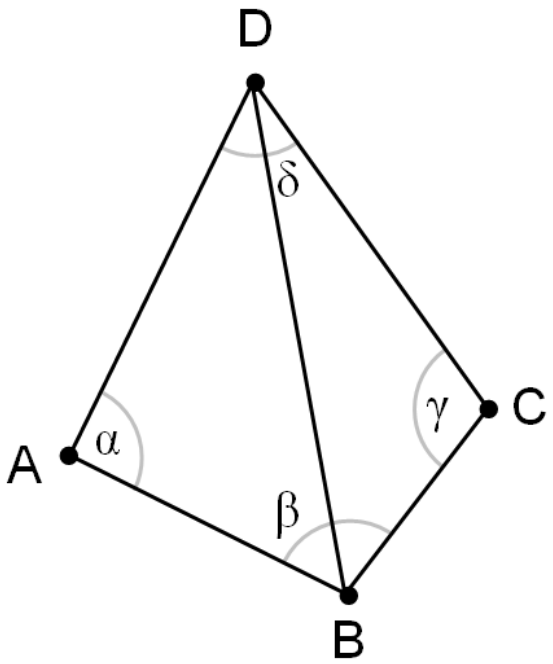
Proposition

三角剖分满足全局 *Delaunay* 条件当且仅当其满足局部 *Delaunay* 条件。

Delaunay 三角剖分

Proposition

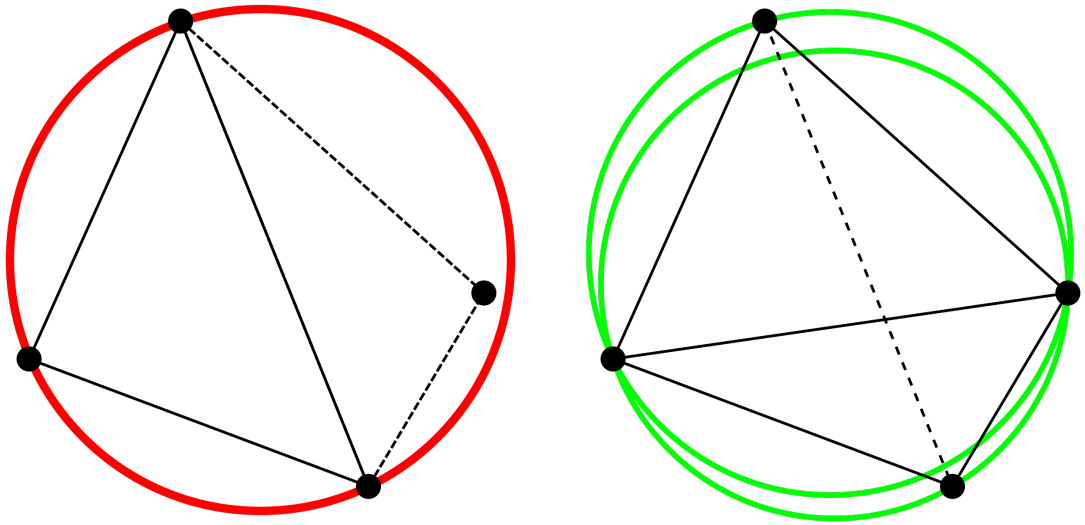
如果 $\alpha + \gamma \leq \pi$ 则满足 *Delaunay* 条件。



Delaunay 三角剖分

Algorithm (翻转操作)

四边形中有两条对角线，总有一条满足 *Delaunay* 条件。因此可以通过“翻转”一条边来满足条件。



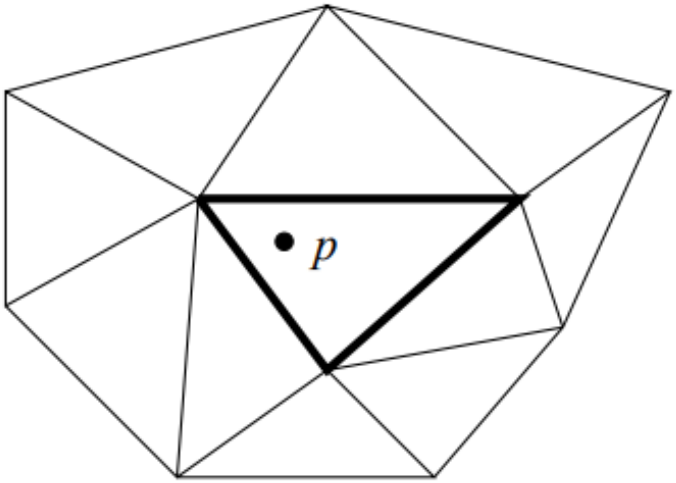
Delaunay 三角剖分

Algorithm (增量算法)

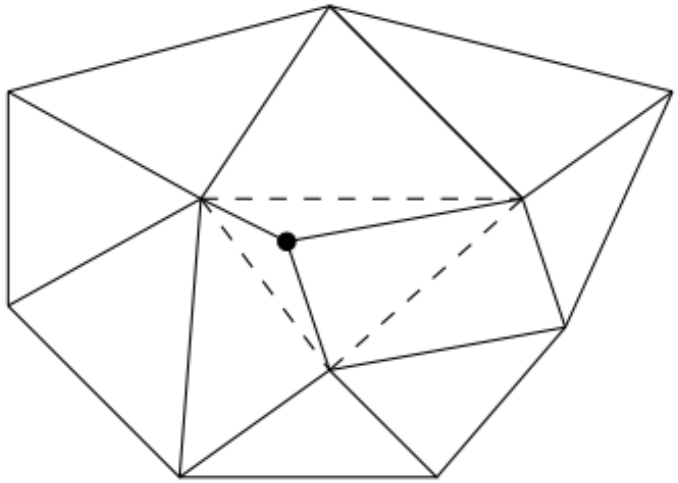
增加点时，找到其所可能影响的边（因为这些边的另一侧一定是一个三角形），递归使用翻转操作使得满足 *Delaunay* 条件。单次插入时间复杂度 $O(n)$。

随机插入点期望只会翻转 $O(1)$ 条边，但是定位点所在的三角形需要 $O(\log n)$。

Delaunay 三角剖分

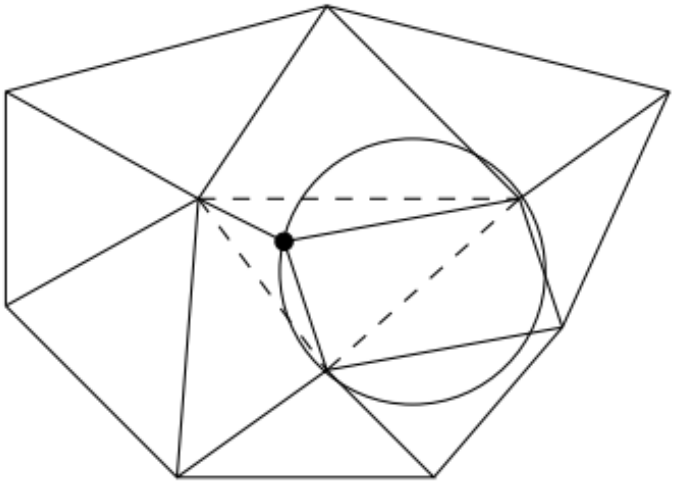


(a)

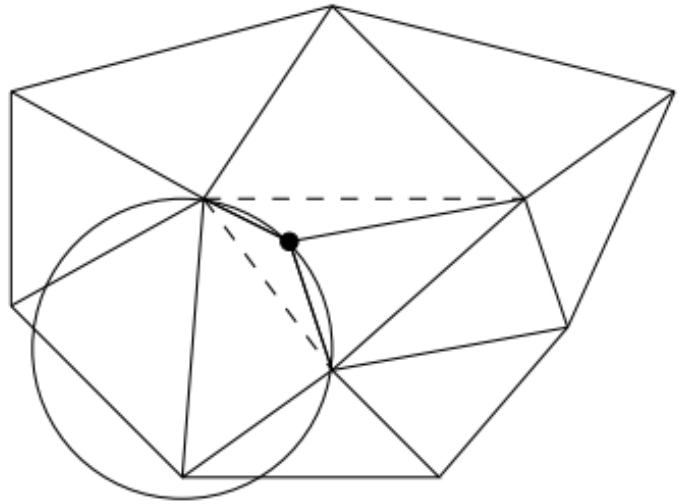


(b)

Delaunay 三角剖分

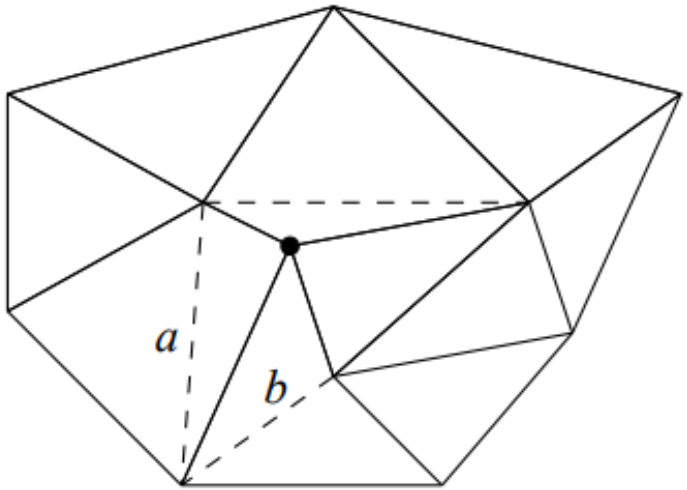


(c)

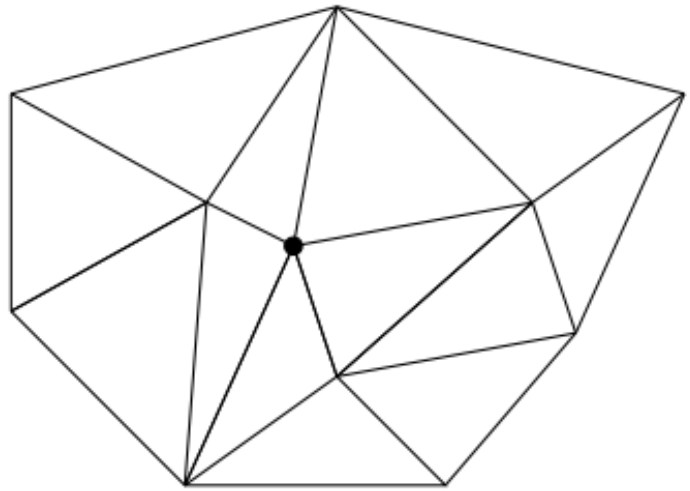


(d)

Delaunay 三角剖分



(e)



(f)

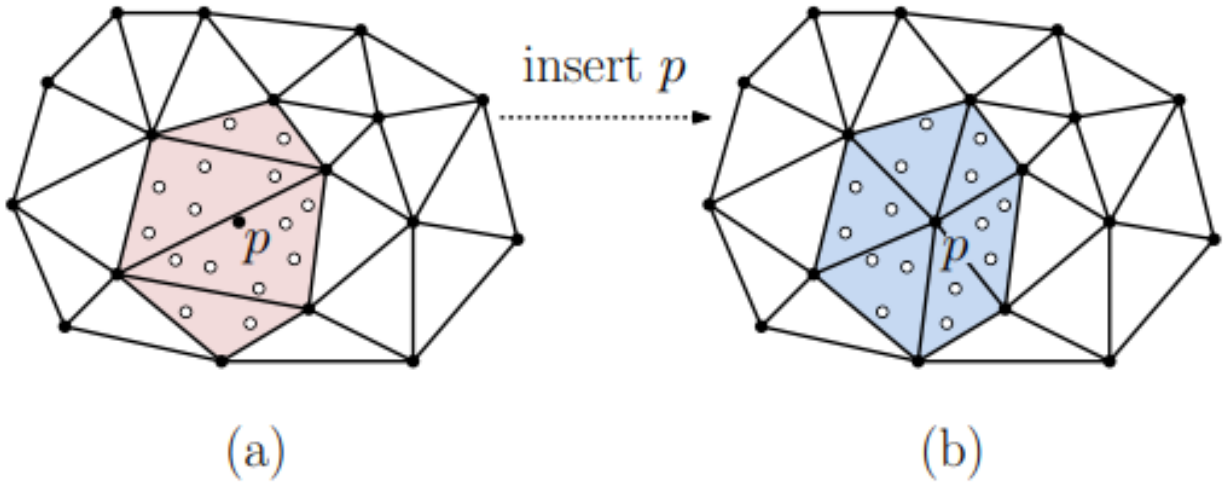
Delaunay 三角剖分

Algorithm ($O(\log n)$ 查找所在三角形)

在插入点的过程中，维护每个三角形内未插入的点（也就是插入时可以直接索引定位到）。

但是插入新的点会引入边，导致需要将三角形内未插入的点重新分配。

在随机数据下期望每个点会被重分配 $O(\log n)$ 次。



Delaunay 三角剖分

Algorithm (分治算法)

存在 $O(n \log n)$ 的分治算法，跑得非常快但是细节较繁琐。并且 *OI* 中考过的 *Delaunay* 三角剖分相关题目也基本上 $O(n^2)$ 起步。

如果有兴趣可以学 ▸ *OI-wiki:* 三角剖分，以及 ▸ *Samuel Peterson* 的文章。

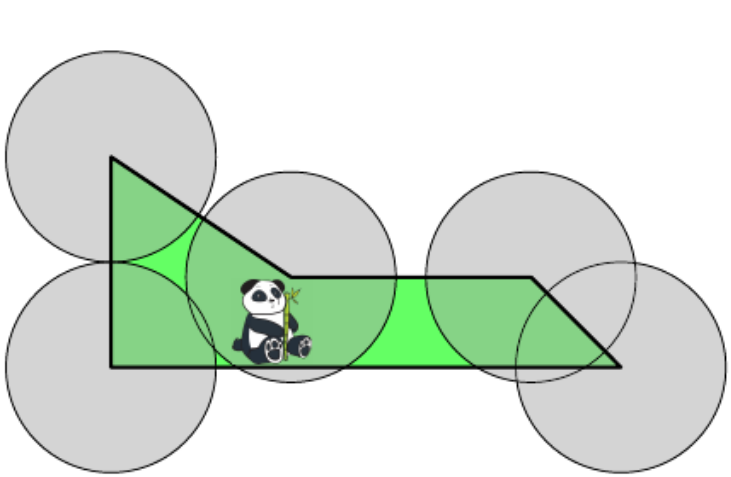
Panda Preserve

Example (World Final 2018 Panda Preserve)

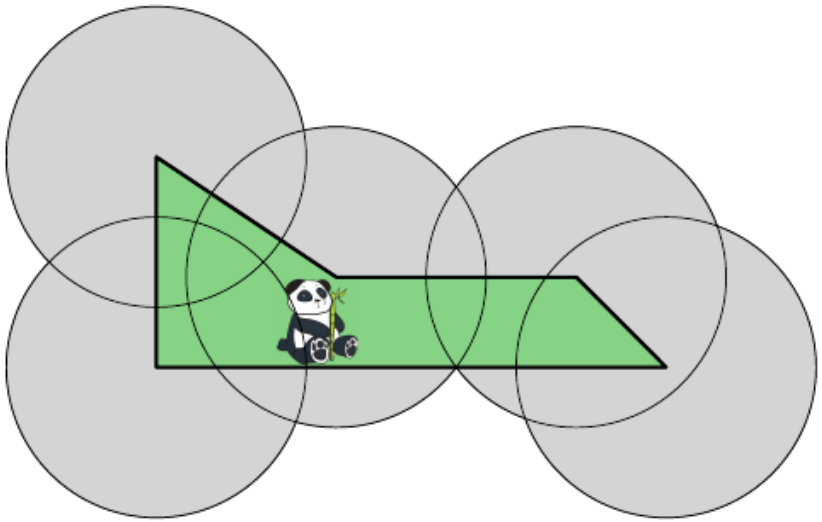
给定 n 个顶点的简单多边形，多边形的每个顶点处会放置一个半径为 r 的圆。

你要确定一个最小的 r，使得所有圆的并包含这个简单多边形。

$n \leq 200$。



(a) An insufficient range for covering the park.



(b) The minimal range for covering the park.

Algorithm

问题等价于，为多边形内每个点寻找其距离最近的顶点。

如果能建立顶点的 *Voronoi* 图，那么就可以通过多边形和每个顶点在 *Voronoi* 图中的区域求交来求得，每个顶点所关联的最近点的集合。

答案就是从顶点到集合最远点距离的最大值。由于 $n \leq 200$，甚至直接写半平面交求 *Voronoi* 图都能过。

Outline

Pick 定理

Theorem (Pick's Theorem)

格点中的简单多边形，其面积 S 和内部格点数 I，边上格点数 B 之间的关系为：$S = I + B/2 - 1$。

当需要计算多边形内部格点数，边上格点数等问题时，有可能可以用到。

最小圆覆盖

Example

给定平面上的一个点集，求一个直径最小的圆，使得其能覆盖所有点。

Algorithm (随机增量法)

这个最小的圆要么由点集中两个点构成一条直径，要么由点集中三个点唯一确定。因此关键是寻找边界上的点。

考虑增量法，以随机顺序加入点。

最小圆覆盖

Proposition

如果第 i 次加入的点不在前 $i-1$ 个点的最小覆盖圆内，则这个点一定在前 i 个点最小覆盖圆的边界上。

Algorithm (随机增量法 Cont.)

如果第 i 个点在目前的最小覆盖圆外，则以这个点为边界上的点，从半径 0 开始重建最小覆盖圆。即重新从 1 枚举到 $i-1$，寻找边界上的点。再次利用这个性质，如果枚举到的点不在最小覆盖圆里，则一定在边界上。

最小圆覆盖

Algorithm (随机增量法 Cont.)

此时已经确定了两个在边界上的点 $j < i$。为了让 j 的枚举满足随机增量算法，需要枚举 $k < j$ 作为第三个点来构建出 $\{1, 2, \ldots, j, i\}$ 的最小覆盖圆。

最小圆覆盖

```
for (int i = 1; i <= n; ++i)
  if (distance(P[i], O) > R) {
    O = P[i], R = 0;
    for (int j = 1; j < i; ++j)
      if (distance(P[j], O) > R) {
        O = (P[i] + P[j]) / 2; R = distance(P[i], O);
        for (int k = 1; k < j; ++k)
          if (distance(P[k], O) > R) {
            O = getCenter(P[i], P[j], P[k]);
            R = distance(P[i], O);
          }
      }
  }
```

最小圆覆盖

Proposition

在随机点顺序的情况下，该算法期望运行时间是 $O(n)$。

最终期望复杂度为，即每层循环只有至多 $\frac{3}{i}$ 的概率进入下一层。（详细证明见原论文 ▸ Smallest enclosing disks）

$$\sum_{i=1}^{n}\left(1+\frac{1}{i}\sum_{j=1}^{i}\left(1+\frac{1}{j}\sum_{k=1}^{j}1\right)\right)=O(n).$$

反演

Definition (Inversion)

在欧几里得空间中，给定反演中心 O，则其他所有点 P 在反演变换中对应着由以下条件唯一确定的点 P'：

- OP 和 OP' 同向。
- $|OP| \cdot |OP'| = 1$。

反演变换是一个双射。

反演

Geogebra 中尝试

Proposition (二维反演变换的性质)

- 过 O 的直线 $\Leftrightarrow$ 过 O 的直线。
- 不过 O 的直线 $\Leftrightarrow$ 过 O 的圆。
- 不过 O 的圆 $\Leftrightarrow$ 不过 O 的圆。
- 保角。

Outline

向量

以下均与二维类似：

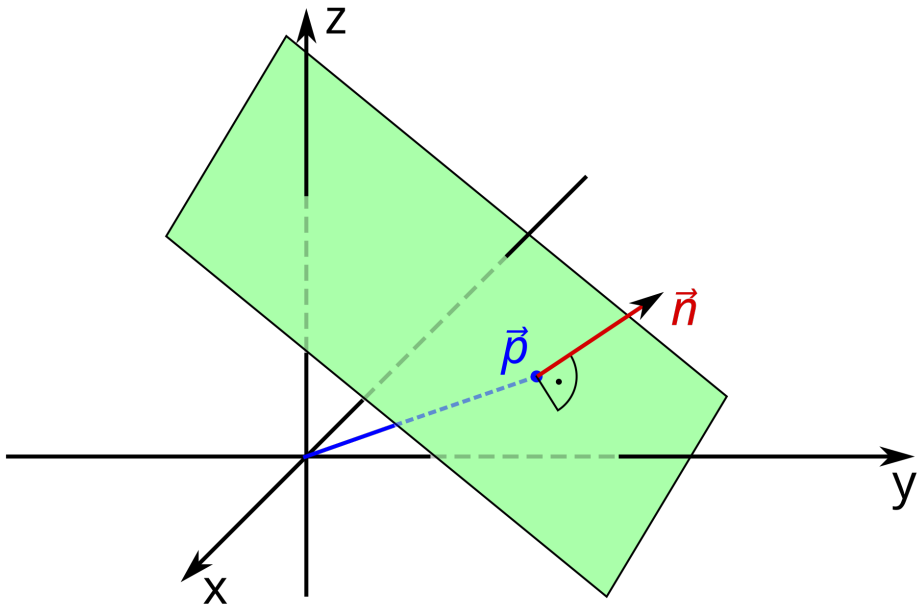
- 向量基础操作；向量内积
- 向量叉积（之前已定义）

平面表示法

Definition (平面)

三维空间中的平面与二维空间中的直线相似，都是“与法向量 (A, B, C) 内积为固定值的点集”：

$$Ax + By + Cz + d = 0$$



二面角

Algorithm

平面的二面角等于其法向量的夹角。因此可以使用内积和叉积来获得角度：

$$\cos\alpha = \frac{\langle a, b\rangle}{|a| \cdot |b|}, \sin\alpha = \frac{|a \times b|}{|a| \cdot |b|}.$$

直线

三维空间中的直线并不方便使用方程来表示。

一般使用参数化的形式，即 $p = p_0 + kv$.

三维凸包

Definition (Convex Hull (3D))

回忆凸包的定义，三维凸包也是点构成的凸集。三维凸包有类似平面图的性质，因为去掉其一个面后留下来的就是平面图。
一般每个面都用逆时针存下三个三角形。

Algorithm (Gift-Wrapping)

卷包裹算法利用了这条性质：对于任意一条在凸包边界上的边，选取任意凸包内的点组成三角形，那么其在凸包面上关联的三角形一定是斜率最大的。

三维凸包

Algorithm (Cont.)

所以，卷包裹算法从最外侧的一条边开始，不断寻找得到斜率最大的点，得到凸包面上的三角形。这个三角形有可能会得到新的边，因此递归计算。

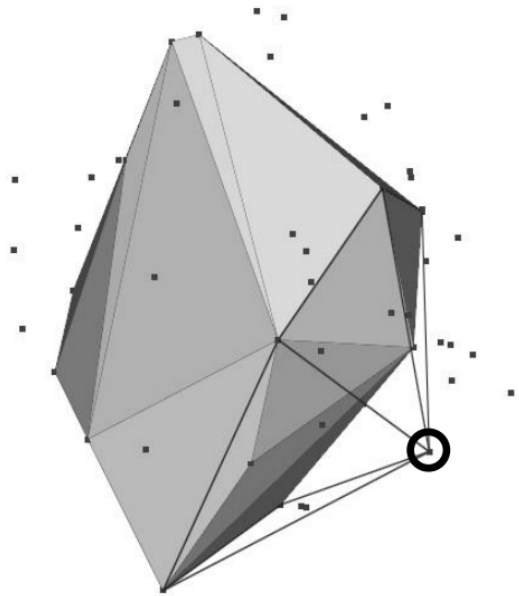
斜率的比较类似二维凸包，只需要计算四个点构成的单纯形体积正负。

这个算法给出了一个 $O(n^2)$ 的三维凸包实现。

三维凸包

Algorithm (增量法)

一个个将点加进凸包。在一个点加入的时候，这个点能看到的所有面都应该被剔除（类比二维）。判断是否能看见只需要判断面与点构成单纯形的体积正负。



三维凸包

Algorithm (分治法)

分治法就和分治求凸包一样，使用相离凸包的合并算法。可以做到 $O(n \log n)$ 时间复杂度。感兴趣可以自学。

ZJOI2018 保镖

Example

给定一个点集，求反演中心在一个矩形区域内均匀随机选取时，凸包顶点数量的期望。

点集大小 $n \leq 2000$。

其实这题可以做到 $O(n \log n)$ 但是这种情况下考场难以写出所以当年出题人抠掉了两个 0。

ZJOI2018 保镖

Algorithm

在反演中，圆是特殊的那个。为了找到特殊的圆，可以考虑 *Delaunay* 三角剖分。

Delaunay 三角的外接圆保证没有其他点在这个外接圆内部。那么：

- 如果反演中心在这个圆外，那么反演之后还是没有点在这个圆内部。也就是还是 *Delaunay* 圆。
- 如果反演中心在圆内，那么反演之后所有点都在这个圆内。这样的圆称为支配圆。

ZJOI2018 保镖

Algorithm (Cont.)

类似的，对于一个三角形的外接圆如果是支配圆（包含所有点的圆），那么

- 反演中心在圆外，则仍然是支配圆。
- 在圆内，则变为 Delaunay 圆。

因为不存在四点共圆的时候 Delaunay 三角剖分唯一，所以 Delaunay 三角形个数也就是 Delaunay 圆的个数和凸包点数相关。

ZJOI2018 保镖

Algorithm (Cont.)

回忆三角剖分的内容，关系为 圆 + 凸包点数 $= 2n - 2$。
因此只需要求出所有的 Delaunay 圆和支配圆，求点在 Delaunay 圆外和点在支配圆内的概率即可。这部分是矩形和圆求交。
剩下的问题是求出所有 Delaunay 圆和支配圆。

ZJOI2018 保镖

Algorithm (Cont.)

回忆 Delaunay 三角剖分判断点在外接圆内时的判别式：

$$\det \begin{pmatrix} A_x & A_y & A_x^2+A_y^2 & 1 \\ B_x & B_y & B_x^2+B_y^2 & 1 \\ C_x & C_y & C_x^2+C_y^2 & 1 \\ D_x & D_y & D_x^2+D_y^2 & 1 \end{pmatrix} < 0$$

本质等价于，把所有点 (x, y) 投影到抛物面 (x, y, x^2+y^2) 后计算单纯性方向的判别式。

其中 A, B, C 在平面上逆时针排列，因此到了抛物面上拥有了明确的方向性。

ZJOI2018 保镖

Algorithm (Cont.)

这对应着三维凸包上的面，因为 Delaunay 圆/支配圆要求判别式的符号一直 < 0 / > 0。

这对应三维凸包的下表面和上表面。并且抛物面 $z = x^2 + y^2$ 也保证了只要 A, B, C 不三点共线，其三角形是有上下之分的。

因此写三维凸包即可，时间复杂度 $O(n^2)$。