计算几何 3：半平面交、随机增量法

Ebola

Institute of Mathematics,
Zhejiang University.

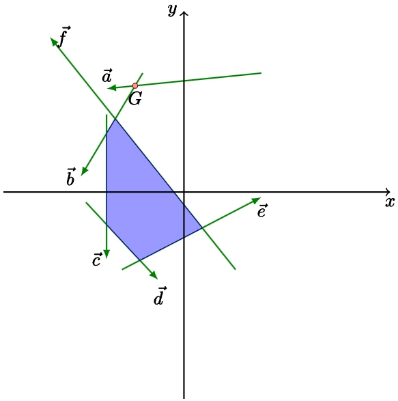
Jan, 2024

1. 半平面交
2. 随机增量法

半平面交

一条直线可以将平面分成两个部分，每个部分都是一个“半平面”。

“半平面交”，顾名思义就是很多个半平面相交的部分。这个区域有可能是无界的，也有可能是有界的。



[CQOI2006] 凸多边形

给定若干个凸多边形（顶点按逆时针顺序给出），求所有凸多边形相交区域的面积。

半平面交

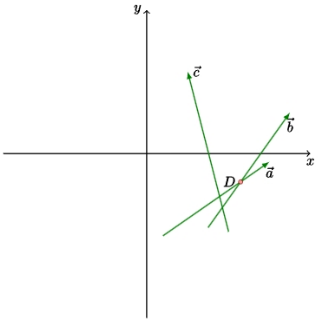
我们用“有源向量”的形式来储存直线，即保存直线上某一个点（源点）坐标、直线的方向向量。

为了方便，我们站在源点，面朝有源向量所指的方向，将左手侧的区域称为这个有源向量的“左半平面”。凸多边形的交其实就是很多个有源向量的左半平面之交。

首先按极角从小到大将所有的有源向量排序，极角可以通过 `atan2(y,x)` 计算，该函数返回一个 $\theta \in (-\pi, \pi]$，$\theta = \arctan \frac{y}{x}$.
如果有极角相同的两个有源向量，我们只保留靠左侧的一个。

半平面交

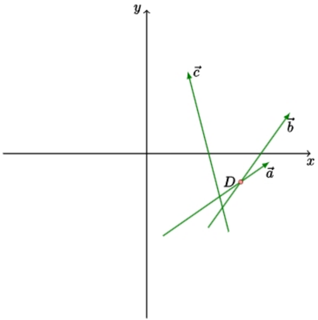
半平面交是一个凸多边形，所以要维护一个凸壳。并保存凸壳上的所有顶点。新加入一个有源向量 $\vec{c}$ 时，可能会出现下图所示的情况。



此时为了维护凸壳，要把 $\vec{b}$ 以及交点 D 弹出。需要一直弹出，直到最后两条直线的交点在 $\vec{c}$ 左侧为止。

半平面交

半平面交是一个凸多边形，所以要维护一个凸壳。并保存凸壳上的所有顶点。新加入一个有源向量 $\vec{c}$ 时，可能会出现下图所示的情况。

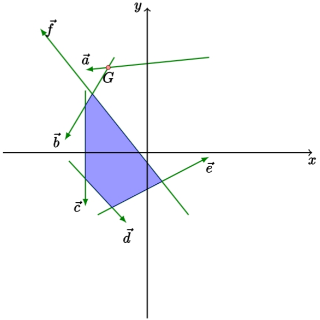


此时为了维护凸壳，要把 $\vec{b}$ 以及交点 D 弹出。需要一直弹出，直到最后两条直线的交点在 $\vec{c}$ 左侧为止。

```
while(!vertex_que.empty() && !l[i].onleft(vertex_que.back()))
    vertex_que.pop_back(), line_que.pop_back();
```

半平面交

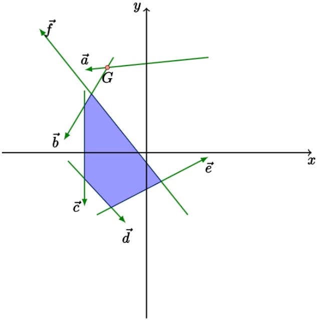
现在继续加入有源向量 $\overrightarrow{f}$，还有可能会出现下图所示的情况：



即，一开始加入的两条直线的交点 G 落在了 $\overrightarrow{f}$ 右侧。此时要从头开始，弹出 $\overrightarrow{a}$ 以及交点 G，一直重复，直到最前面两条直线的交点在 $\overrightarrow{f}$ 左侧为止。

半平面交

现在继续加入有源向量 $\vec{f}$，还有可能会出现下图所示的情况：

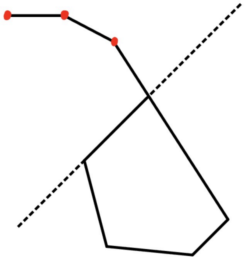


即，一开始加入的两条直线的交点 G 落在了 $\vec{f}$ 右侧。此时要从头开始，弹出 $\vec{a}$ 以及交点 G，一直重复，直到最前面两条直线的交点在 $\vec{f}$ 左侧为止。

```
while(!vertex_que.empty() && !l[i].onleft(vertex_que.front()))
    vertex_que.pop_front(), line_que.pop_front();
```

半平面交

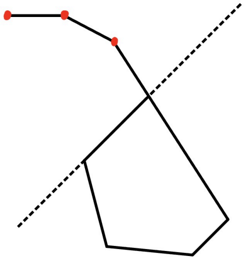
现在所有的有源向量都已经加入完毕了，但是可能会出现下图所示的情况：



显然，虚线以上的部分是需要弹出的，我们可以借助第一个有源向量来判断，即：若末端的点不在第一个有源向量左侧，则弹出，直到末端的点在其左侧为止。

半平面交

现在所有的有源向量都已经加入完毕了，但是可能会出现下图所示的情况：



显然，虚线以上的部分是需要弹出的，我们可以借助第一个有源向量来判断，即：若末端的点不在第一个有源向量左侧，则弹出，直到末端的点在其左侧为止。

```
while(!vertex_que.empty() && !line_que.front().onleft(vertex_que.back()))
    vertex_que.pop_back(), line_que.pop_back();
vertex_que.push_back(line_que.back().crosspoint(line_que.front()));
```

半平面交

完整代码（不含排序和去重）:

```
deque<Line> line_que;
deque<Point> vertex_que;

line_que.push_back(l[0]);
line_que.push_back(l[1]);
vertex_que.push_back(l[0].crosspoint(l[1]));

for(int i = 2; i < tot; i++){
    while(!vertex_que.empty() && !l[i].onleft(vertex_que.back()))
        vertex_que.pop_back(), line_que.pop_back();
    while(!vertex_que.empty() && !l[i].onleft(vertex_que.front()))
        vertex_que.pop_front(), line_que.pop_front();
    vertex_que.push_back(l[i].crosspoint(line_que.back()));
    line_que.push_back(l[i]);
}
while(!vertex_que.empty() && !line_que.front().onleft(vertex_que.back()))
    vertex_que.pop_back(), line_que.pop_back();
vertex_que.push_back(line_que.back().crosspoint(line_que.front()));
```

最小圆覆盖 (P1743)

给定平面上若干个点，求能覆盖所有点的最小圆。

最小圆覆盖

定理 1：最小圆覆盖存在且唯一。

最小圆覆盖

定理 1：最小圆覆盖存在且唯一。

定理 2：最小圆要么是以某两个点为直径的圆，要么是某三个点的外接圆。

最小圆覆盖

定理 1：最小圆覆盖存在且唯一。

定理 2：最小圆要么是以某两个点为直径的圆，要么是某三个点的外接圆。

定义：我们记 $\mathrm{MC}(P_1, ..., P_k; Q_1, ..., Q_n)$ 表示 $P_1, ..., P_k, Q_1, ..., Q_n$ 的最小圆覆盖，其中 $P_1, ..., P_k$ 在圆周上。

最小圆覆盖

定理 1：最小圆覆盖存在且唯一。

定理 2：最小圆要么是以某两个点为直径的圆，要么是某三个点的外接圆。

定义：我们记 $\mathsf{MC}(P_1, \ldots, P_k; Q_1, \ldots, Q_n)$ 表示 $P_1, \ldots, P_k, Q_1, \ldots, Q_n$ 的最小圆覆盖，其中 $P_1, \ldots, P_k$ 在圆周上。

引理 1：若 $Q_n \notin \mathsf{MC}(\varnothing; Q_1, \ldots, Q_{n-1})$，那么

$$\mathsf{MC}(\varnothing; Q_1, \ldots, Q_n) = \mathsf{MC}(Q_n; Q_1, \ldots, Q_{n-1}).$$

最小圆覆盖

定理 1：最小圆覆盖存在且唯一。

定理 2：最小圆要么是以某两个点为直径的圆，要么是某三个点的外接圆。

定义：我们记 $\mathsf{MC}(P_1, \ldots, P_k; Q_1, \ldots, Q_n)$ 表示 $P_1, \ldots, P_k, Q_1, \ldots, Q_n$ 的最小圆覆盖，其中 $P_1, \ldots, P_k$ 在圆周上。

引理 1：若 $Q_n \notin \mathsf{MC}(\varnothing; Q_1, \ldots, Q_{n-1})$，那么

$$\mathsf{MC}(\varnothing; Q_1, \ldots, Q_n) = \mathsf{MC}(Q_n; Q_1, \ldots, Q_{n-1}).$$

引理 2：若 $Q_n \notin \mathsf{MC}(P_1; Q_1, \ldots, Q_{n-1})$，那么

$$\mathsf{MC}(P_1; Q_1, \ldots, Q_n) = \mathsf{MC}(P_1, Q_n; Q_1, \ldots, Q_{n-1}).$$

最小圆覆盖

定理 1：最小圆覆盖存在且唯一。

定理 2：最小圆要么是以某两个点为直径的圆，要么是某三个点的外接圆。

定义：我们记 $\mathsf{MC}(P_1, ..., P_k; Q_1, ..., Q_n)$ 表示 $P_1, ..., P_k, Q_1, ..., Q_n$ 的最小圆覆盖，其中 $P_1, ..., P_k$ 在圆周上。

引理 1：若 $Q_n \notin \mathsf{MC}(\varnothing; Q_1, ..., Q_{n-1})$，那么

$$\mathsf{MC}(\varnothing; Q_1, ..., Q_n) = \mathsf{MC}(Q_n; Q_1, ..., Q_{n-1}).$$

引理 2：若 $Q_n \notin \mathsf{MC}(P_1; Q_1, ..., Q_{n-1})$，那么

$$\mathsf{MC}(P_1; Q_1, ..., Q_n) = \mathsf{MC}(P_1, Q_n; Q_1, ..., Q_{n-1}).$$

引理 3：若 $Q_n \notin \mathsf{MC}(P_1, P_2; Q_1, ..., Q_{n-1})$，那么

$$\mathsf{MC}(P_1, P_2; Q_1, ..., Q_n) = \mathsf{MC}(P_1, P_2, Q_n; Q_1, ..., Q_{n-1}).$$

最小圆覆盖

引理 1：若 $Q_n \notin \text{MC}(\varnothing; Q_1, ..., Q_{n-1})$，那么

$$\text{MC}(\varnothing; Q_1, ..., Q_n) = \text{MC}(Q_n; Q_1, ..., Q_{n-1}).$$

证明：若不然，我们令当前圆为 $\text{MC}(\varnothing; Q_1, ..., Q_{n-1})$，让它逐渐变换为 $\text{MC}(\varnothing; Q_1, ..., Q_n)$，过程如下：

1. 逐渐增大半径，直到与 $\text{MC}(\varnothing; Q_1, ..., Q_n)$ 大小相同；
2. 逐渐平移到 $\text{MC}(\varnothing; Q_1, ..., Q_n)$ 的位置。

最小圆覆盖

引理 1：若 $Q_n \notin \text{MC}(\varnothing; Q_1, ..., Q_{n-1})$，那么

$$\text{MC}(\varnothing; Q_1, ..., Q_n) = \text{MC}(Q_n; Q_1, ..., Q_{n-1}).$$

证明：若不然，我们令当前圆为 $\text{MC}(\varnothing; Q_1, ..., Q_{n-1})$，让它逐渐变换为 $\text{MC}(\varnothing; Q_1, ..., Q_n)$，过程如下：

1. 逐渐增大半径，直到与 $\text{MC}(\varnothing; Q_1, ..., Q_n)$ 大小相同；
2. 逐渐平移到 $\text{MC}(\varnothing; Q_1, ..., Q_n)$ 的位置。

在第一过程中，原来在圆内的点一直还在圆内；在第二过程中，不可能有原来的点跑出去，否则它就不会再进来了，从而与“覆盖”性质矛盾。

最小圆覆盖

引理 1：若 $Q_n \notin \mathrm{MC}(\varnothing; Q_1, \dots, Q_{n-1})$，那么

$$\mathrm{MC}(\varnothing; Q_1, \dots, Q_n) = \mathrm{MC}(Q_n; Q_1, \dots, Q_{n-1}).$$

证明：若不然，我们令当前圆为 $\mathrm{MC}(\varnothing; Q_1, \dots, Q_{n-1})$，让它逐渐变换为 $\mathrm{MC}(\varnothing; Q_1, \dots, Q_n)$，过程如下：

1. 逐渐增大半径，直到与 $\mathrm{MC}(\varnothing; Q_1, \dots, Q_n)$ 大小相同；
2. 逐渐平移到 $\mathrm{MC}(\varnothing; Q_1, \dots, Q_n)$ 的位置。

在第一过程中，原来在圆内的点一直还在圆内；在第二过程中，不可能有原来的点跑出去，否则它就不会再进来了，从而与"覆盖"性质矛盾。
但是在第一或第二过程中，一定会有某个时刻，圆周刚好碰到 Q_n；这时候当前圆是一个半径不超过 $\mathrm{MC}(\varnothing; Q_1, \dots, Q_n)$ 的圆覆盖，且不同于 $\mathrm{MC}(\varnothing; Q_1, \dots, Q_n)$；这与最小圆覆盖的唯一性矛盾！

引理 2、引理 3 的证明类似（课上有时间可以证一下）。

随机增量法

现在我们有了理论基础，接下来用随机增量法来求解最小圆覆盖。

随机增量法

现在我们有了理论基础，接下来用随机增量法来求解最小圆覆盖。

“增量法”的意思是将一个问题化为规模刚好小一层的子问题。解决子问题后加入当前的对象。

随机增量法

现在我们有了理论基础，接下来用随机增量法来求解最小圆覆盖。

“增量法”的意思是将一个问题化为规模刚好小一层的子问题。解决子问题后加入当前的对象。

具体来说，在最小圆覆盖问题中，就是先解决前 i 个点的最小圆覆盖，然后加入第 $i+1$ 个点，并修正答案；一直到加完所有点为止。

随机增量法

现在我们有了理论基础，接下来用随机增量法来求解最小圆覆盖。

“增量法”的意思是将一个问题化为规模刚好小一层的子问题。解决子问题后加入当前的对象。

具体来说，在最小圆覆盖问题中，就是先解决前 i 个点的最小圆覆盖，然后加入第 $i+1$ 个点，并修正答案；一直到加完所有点为止。

“随机增量法”就是先把所有点的顺序打乱，然后做“增量法”。

随机增量法

我们用增量法来求 MC($\varnothing$; $Q_1, \dots, Q_n$)，即从 MC($\varnothing$; Q_1) 开始，逐渐增加 $Q_2, \dots, Q_{n-1}$。

随机增量法

我们用增量法来求 $\mathrm{MC}(\varnothing; Q_1, \ldots, Q_n)$，即从 $\mathrm{MC}(\varnothing; Q_1)$ 开始，逐渐增加 $Q_2, \ldots, Q_{n-1}$。

假设现在已经有了 $\mathrm{MC}(\varnothing; Q_1, \ldots, Q_{i-1})$，如果 Q_i 也被它覆盖，那么直接考虑下一个点；否则，根据引理 1，我们知道：

$$\mathrm{MC}(\varnothing; Q_1, \ldots, Q_i) = \mathrm{MC}(Q_i; Q_1, \ldots, Q_{i-1}).$$

随机增量法

我们用增量法来求 $\mathrm{MC}(\varnothing; Q_1, \ldots, Q_n)$，即从 $\mathrm{MC}(\varnothing; Q_1)$ 开始，逐渐增加 $Q_2, \ldots, Q_{n-1}$。

假设现在已经有了 $\mathrm{MC}(\varnothing; Q_1, \ldots, Q_{i-1})$，如果 Q_i 也被它覆盖，那么直接考虑下一个点；否则，根据引理 1，我们知道：

$$\mathrm{MC}(\varnothing; Q_1, \ldots, Q_i) = \mathrm{MC}(Q_i; Q_1, \ldots, Q_{i-1}).$$

```
// 增量法计算 MC(空; Q[1],...,Q[n])
void getMC_fix0(Point &ans, double &r){
    ans = Q[1];
    r = 0;
    for(int i = 2; i <= n; i++){
        if(in_circle(Q[i], ans, r)) continue;
        // 计算 MC(Q[i]; Q[1],...,Q[i-1])
        getMC_fix1(i, ans, r);
    }
}
```

随机增量法

我们用增量法来求 $\mathrm{MC}(Q_i; Q_1, \ldots, Q_{i-1})$，即从 $\mathrm{MC}(Q_i; Q_1)$ 开始，逐渐增加 $Q_2, \ldots, Q_{i-1}$。

随机增量法

我们用增量法来求 $\mathrm{MC}(Q_i; Q_1, ..., Q_{i-1})$，即从 $\mathrm{MC}(Q_i; Q_1)$ 开始，逐渐增加 $Q_2, ..., Q_{i-1}$。

当加入一个点 Q_j 时，如果它被 $\mathrm{MC}(Q_i; Q_1, ..., Q_{j-1})$ 覆盖，直接考虑下一个点；否则，根据引理 2，我们知道：

$$\mathrm{MC}(Q_i; Q_1, ..., Q_j) = \mathrm{MC}(Q_i, Q_j; Q_1, ..., Q_{j-1}).$$

随机增量法

我们用增量法来求 $\mathrm{MC}(Q_i; Q_1, \ldots, Q_{i-1})$，即从 $\mathrm{MC}(Q_i; Q_1)$ 开始，逐渐增加 $Q_2, \ldots, Q_{i-1}$。

当加入一个点 Q_j 时，如果它被 $\mathrm{MC}(Q_i; Q_1, \ldots, Q_{j-1})$ 覆盖，直接考虑下一个点；否则，根据引理 2，我们知道：

$$\mathrm{MC}(Q_i; Q_1, \ldots, Q_j) = \mathrm{MC}(Q_i, Q_j; Q_1, \ldots, Q_{j-1}).$$

```
// 增量法计算 MC(Q[i]; Q[1],...,Q[i-1])
void getMC_fix1(int i, Point &ans, double &r){
    ans.x = 0.5 * (Q[i].x + Q[1].x);
    ans.y = 0.5 * (Q[i].y + Q[1].y);
    r = Length(Q[i] - Q[1]) / 2;
    for(int j = 2; j < i; j++){
        if(in_circle(Q[j], ans, r)) continue;
        // 计算 MC(Q[i],Q[j]; Q[1],...,Q[j-1])
        getMC_fix2(i, j, ans, r);
    }
}
```

随机增量法

我们用增量法来求 $\mathrm{MC}(Q_i, Q_j; Q_1, \ldots, Q_{j-1})$，即从 $\mathrm{MC}(Q_i, Q_j; \varnothing)$ 开始，逐渐增加 $Q_1, \ldots, Q_{j-1}$。

随机增量法

我们用增量法来求 $\mathrm{MC}(Q_i, Q_j; Q_1, \ldots, Q_{j-1})$，即从 $\mathrm{MC}(Q_i, Q_j; \varnothing)$ 开始，逐渐增加 $Q_1, \ldots, Q_{j-1}$。

当加入一个点 Q_k 时，如果它被 $\mathrm{MC}(Q_i, Q_j; Q_1, \ldots, Q_{k-1})$ 覆盖，直接考虑下一个点；否则，根据引理 3，我们知道：

$$\mathrm{MC}(Q_i, Q_j; Q_1, \ldots, Q_k) = \mathrm{MC}(Q_i, Q_j, Q_k; Q_1, \ldots, Q_{k-1}) = \Delta Q_i Q_j Q_k \text{的外接圆}.$$

随机增量法

我们用增量法来求 $\mathrm{MC}(Q_i, Q_j; Q_1, \ldots, Q_{j-1})$，即从 $\mathrm{MC}(Q_i, Q_j; \varnothing)$ 开始，逐渐增加 $Q_1, \ldots, Q_{j-1}$。

当加入一个点 Q_k 时，如果它被 $\mathrm{MC}(Q_i, Q_j; Q_1, \ldots, Q_{k-1})$ 覆盖，直接考虑下一个点；否则，根据引理 3，我们知道：

$$\mathrm{MC}(Q_i, Q_j; Q_1, \ldots, Q_k) = \mathrm{MC}(Q_i, Q_j, Q_k; Q_1, \ldots, Q_{k-1}) = \triangle Q_iQ_jQ_k\text{的外接圆}.$$

```
// 增量法计算 MC(Q[i],Q[j]; Q[1],...,Q[j-1])
void getMC_fix2(int i, int j, Point &ans, double &r){
    ans.x = 0.5 * (Q[i].x + Q[j].x);
    ans.y = 0.5 * (Q[i].y + Q[j].y);
    r = Length(Q[i] - Q[j]) / 2;

    for(int k = 1; k < j; k++){
        if(in_circle(Q[k], ans, r)) continue;
        // 计算 MC(Q[i],Q[j],Q[k]; Q[1],...,Q[k-1])
        // 由 Q[i], Q[j], Q[k] 的外接圆唯一确定
        geto(Q[i], Q[j], Q[k], ans, r);
    }
}
```

随机增量法

一开始必须把所有点的打乱，否则会被善意的出题人卡到 $O(n^3)$。

只要进行了打乱，复杂度是期望 $O(n)$ 的，19 世纪就已经有数学家证明了这一点，我们不要求掌握。

[CF442E] Gena and Second Distance

在 $W \times H$ 的矩形内有 n 个特殊点。矩形内任意一点的价值是和它**第二近**的特殊点的距离，求矩形内价值最大的点的价值。

$W, H \leq 10^6$， $n \leq 1000$

[CF442E] Gena and Second Distance

如果价值最大的点是 P，最大价值是 r，这就意味着以 P 为圆心，半径为 r 的圆（记为 $C(P,r)$）内部**恰好**有一个特殊点 S，边界上 **至少**有一个特殊点（记其中一个为 Q）。

[CF442E] Gena and Second Distance

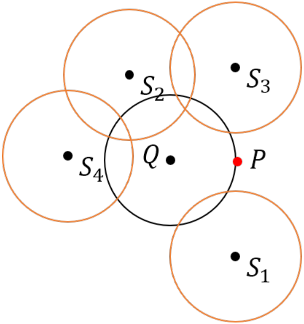
如果价值最大的点是 P，最大价值是 r，这就意味着以 P 为圆心，半径为 r 的圆（记为 $C(P,r)$）内部**恰好**有一个特殊点 S，边界上 **至少**有一个特殊点（记其中一个为 Q）。

注意到：

- S 在 $C(P,r)$ 内部 **等价于** P 在 $C(S,r)$ 内部；
- Q 在 $C(P,r)$ 边界上 **等价于** P 在 $C(Q,r)$ 边界上。

[CF442E] Gena and Second Distance

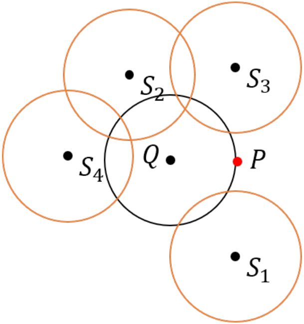
我们首先二分 r，接下来考虑如何判断答案是否大于等于 r。我们不妨来枚举**边界上的**特殊点 Q。设其它的特殊点为 $S_1, \ldots, S_m$，我们希望 $C(Q, r)$ 的边界上存在一个点 P，使它至多只在一个 $C(S_i, r)$ 内部。（当然，如果点 P 不在任何一个 $C(S_i, r)$ 内部，说明答案可以大于 r）



P点的价值大于r

[CF442E] Gena and Second Distance

我们首先二分 r，接下来考虑如何判断答案是否大于等于 r。我们不妨来枚举**边界上的**特殊点 Q。设其它的特殊点为 $S_1, \dots, S_m$，我们希望 $C(Q,r)$ 的边界上存在一个点 P，使它至多只在一个 $C(S_i,r)$ 内部。(当然，如果点 P 不在任何一个 $C(S_i,r)$ 内部，说明答案可以大于 r)

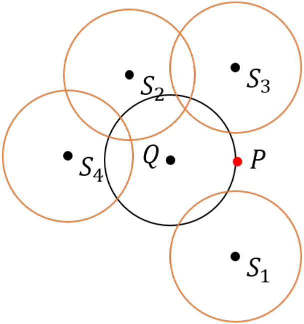


*P*点的价值大于*r*

把 $C(S_i,r)$ 与 $C(Q,r)$ 的交点全部求出来，按逆时针排序，做线段覆盖（差分法），检查是否有至多被一条线段覆盖的位置即可。

[CF442E] Gena and Second Distance

我们首先二分 r，接下来考虑如何判断答案是否大于等于 r。我们不妨来枚举**边界上的**特殊点 Q。设其它的特殊点为 $S_1, \dots, S_m$，我们希望 $C(Q, r)$ 的边界上存在一个点 P，使它至多只在一个 $C(S_i, r)$ 内部。（当然，如果点 P 不在任何一个 $C(S_i, r)$ 内部，说明答案可以大于 r）



P点的价值大于r

把 $C(S_i, r)$ 与 $C(Q, r)$ 的交点全部求出来，按逆时针排序，做线段覆盖（差分法），检查是否有至多被一条线段覆盖的位置即可。
复杂度：二分答案、枚举 Q、枚举 S_i 求所有交点并排序，$O(\log r \cdot n^2 \log n)$

[CF442E] Gena and Second Distance

这个复杂度过不去，考虑优化。

[CF442E] Gena and Second Distance

这个复杂度过不去，考虑优化。

先枚举，确定某个点为 Q，然后二分答案 r，接下来一样的求交、排序、判定。对于每一个 Q，复杂度是 $O(\log r \cdot n \log n)$。

[CF442E] Gena and Second Distance

这个复杂度过不去，考虑优化。

先枚举，确定某个点为 Q，然后二分答案 r，接下来一样的求交、排序、判定。对于每一个 Q，复杂度是 $O(\log r \cdot n \log n)$。

现在，看起来外层循环枚举 Q 还有一个 $O(n)$，其实不然。当枚举下一个 Q 时，**先判断**它的答案能否大于之前的答案 r，如果不能，则没必要在这个 Q 里二分答案。

[CF442E] Gena and Second Distance

这个复杂度过不去，考虑优化。

先枚举，确定某个点为 Q，然后二分答案 r，接下来一样的求交、排序、判定。对于每一个 Q，复杂度是 $O(\log r \cdot n \log n)$。

现在，看起来外层循环枚举 Q 还有一个 $O(n)$，其实不然。当枚举下一个 Q 时，**先判断**它的答案能否大于之前的答案 r，如果不能，则没必要在这个 Q 里二分答案。

现在，假设以每个特殊点为 Q，求得的答案分别为 $r_1, ..., r_n$，如果 $r_2, ..., r_i$ 均不超过 r_1，我们是不会以 $2, ..., i$ 为 Q 点进行二分的。换句话说，如果我们以 $j_1, j_2, ..., j_k$ 这些点为 Q 进行了二分，那么 $r_{j_1}, ..., r_{j_k}$ 一定是一个严格上升子序列。

[CF442E] Gena and Second Distance

这个复杂度过不去，考虑优化。

先枚举，确定某个点为 Q，然后二分答案 r，接下来一样的求交、排序、判定。对于每一个 Q，复杂度是 $O(\log r \cdot n \log n)$。

现在，看起来外层循环枚举 Q 还有一个 $O(n)$，其实不然。当枚举下一个 Q 时，**先判断**它的答案能否大于之前的答案 r，如果不能，则没必要在这个 Q 里二分答案。

现在，假设以每个特殊点为 Q，求得的答案分别为 $r_1, \ldots, r_n$，如果 $r_2, \ldots, r_i$ 均不超过 r_1，我们是不会以 $2, \ldots, i$ 为 Q 点进行二分的。换句话说，如果我们以 $j_1, j_2, \ldots, j_k$ 这些点为 Q 进行了二分，那么 $r_{j_1}, \ldots, r_{j_k}$ 一定是一个严格上升子序列。

将一个任意序列随机打乱，其最长严格上升子序列的长度期望为 $O(\log n)$。因此，我们的复杂度降低到了 $O(\log r \cdot n \log^2 n + n^2 \log n)$，后面加的那个 $n^2 \log n$ 是因为上面提到的“先判断⋯⋯”。

Thank You