信息学竞赛中的几何问题

WCK

XC

2024

计算几何问题在信息学竞赛中经常遇到。
虽然胡老师在2022年给集训3讲过计算几何，但是有人那时候还没来到集训队，所以我打算再讲一遍计算几何。
~~其实还有一个原因，就是《雨中的相遇》只有4个人通过。~~

平移旋转

平移旋转是计算几何中最基本、最重要、最常用的方法。
有些几何问题直接求解非常复杂，但是通过平移旋转可以将其转化为特殊情况，更简单地求解。

旋转公式

既然叫计算几何，肯定要计算嘛。怎么计算一个点平移、旋转之后的坐标?
平移很好说，直接将横纵坐标分别加上一个数就行了。

旋转公式

既然叫计算几何，肯定要计算嘛。怎么计算一个点平移、旋转之后的坐标?
平移很好说，直接将横纵坐标分别加上一个数就行了。
而旋转则需要使用公式。

旋转公式

既然叫计算几何，肯定要计算嘛。怎么计算一个点平移、旋转之后的坐标?
平移很好说，直接将横纵坐标分别加上一个数就行了。
而旋转则需要使用公式。
设点 P 的坐标为 (x_0, y_0) ，如果 P 绕着原点**逆时针**旋转 θ后得到点 $P'(x_1, y_1)$ ，则有以下公式:

$$x_1 = x_0 \cdot cos(\theta) - y_0 * sin(\theta)$$

$$y_1 = y_0 \cdot cos(\theta) + x_0 * sin(\theta)$$

这个公式非常重要，一定要背会。（当然也可以自己推）

常用C++数学函数

由于旋转需要使用三角函数，因此很有必要了解一下C++中与计算几何有关的数学函数。

常用C++数学函数

由于旋转需要使用三角函数，因此很有必要了解一下C++中与计算几何有关的数学函数。
首先是sin,cos,tan三个函数，它们都接受一个参数θ，分别返回θ的sin, cos, tan 值。需要注意，这里θ的单位是**弧度制**。

常用C++数学函数

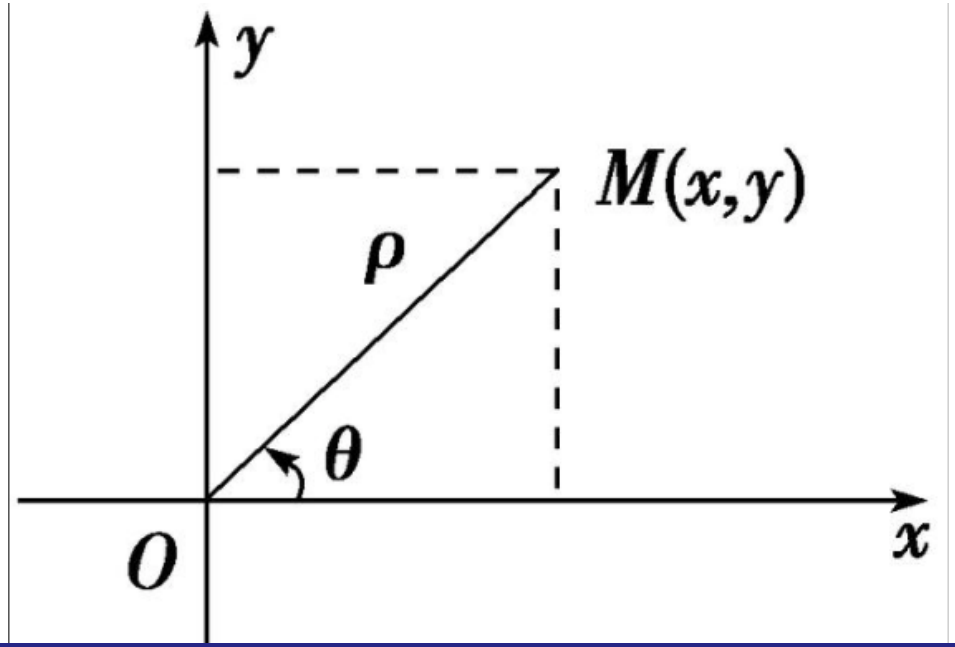
由于旋转需要使用三角函数，因此很有必要了解一下C++中与计算几何有关的数学函数。

首先是sin,cos,tan三个函数，它们都接受一个参数θ，分别返回θ的sin, cos, tan 值。需要注意，这里θ的单位是**弧度制**。

其次是一个非常重要的函数：$atan2$。$atan2(y, x)$ 将返回点 (x, y) 的**极角**。一个点的**极角**表示这个点与原点的连线与x轴的夹角，单位同样是**弧度制**，范围在$(-\pi, \pi]$ 之间。若点 (x, y) 在x轴上方，则这个点的极角为正数；若点 (x, y) 在x轴下方，则这个点的极角为负数。

常用C++数学函数

如下图所示，θ 即为点 M 的极角，因此 $atan2(y,x)=\theta$ 。
一定注意，$atan2$ 的**第一个参数是纵坐标(y)，第二个参数是横坐标(x)！！！** 千万不要搞反了。
例如，$atan2(1,1)=\frac{\pi}{4}$; $atan2(-1,1)=-\frac{\pi}{4}$。



例题

接下来以一道例题，来讲述平移旋转的应用。
给定三个点 $P(x,y), Q_1(x_1,y_1), Q_2(x_2,y_2)$ ，求线段 Q_1Q_2 上离点 P 最近的点的坐标。

解析

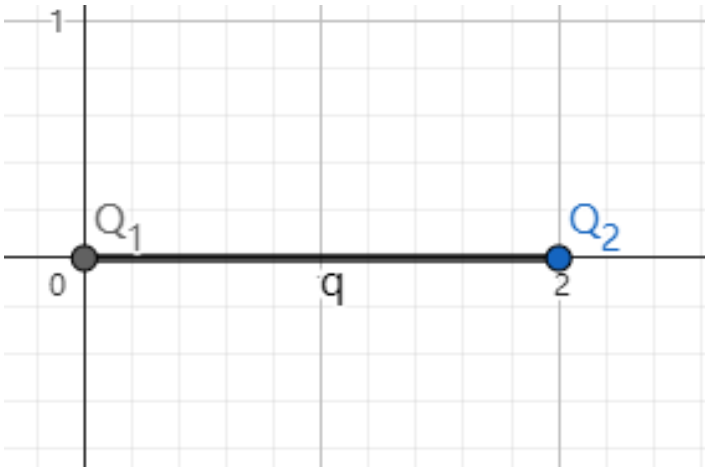
这道题显然可以列点垂式，但是过于麻烦（要推柿子），有没有简单的方法?

~~当然，如果你觉得点垂式更简单，请忽略这句话。~~

解析

这道题显然可以列点垂式，但是过于麻烦（要推柿子），有没有简单的方法?

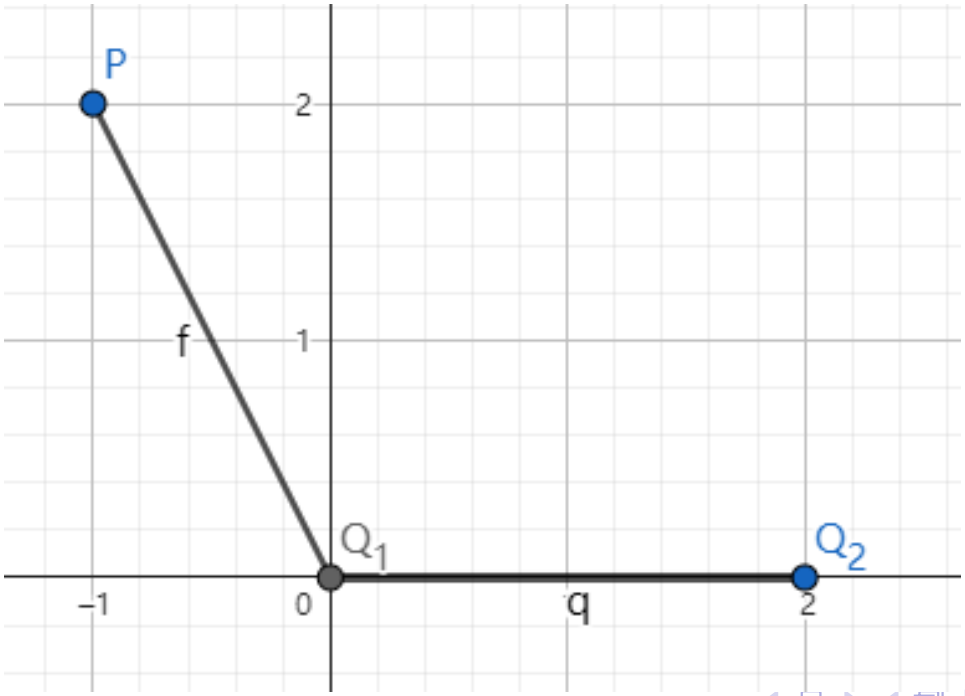
首先我们考虑一种特殊情况：点 Q_1 为原点，且点 Q_2 位于x轴正方向上（如下图所示）。



特殊情况

然后分情况讨论：

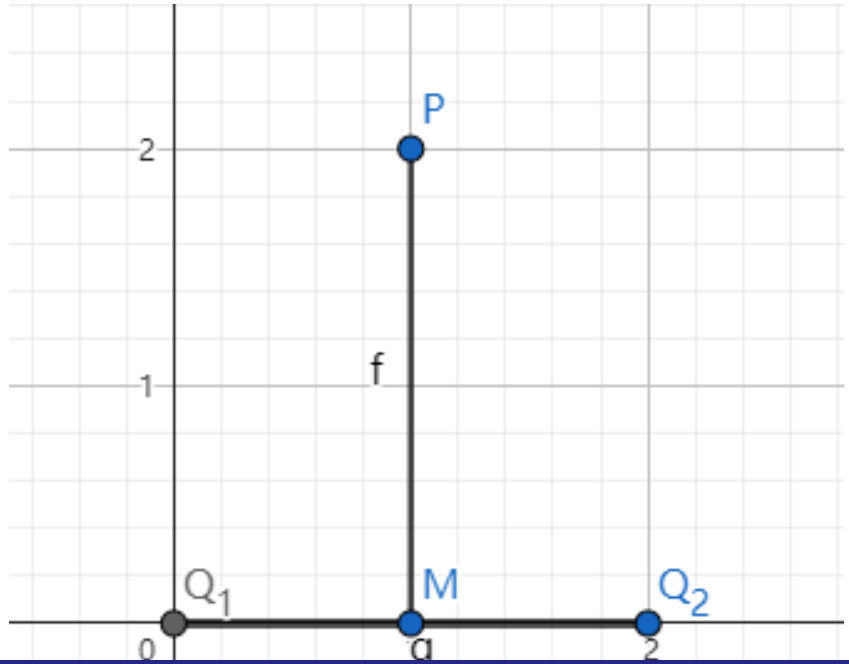
1. 点 P 的x坐标小于0（如下图所示），此时点 Q_1 就是线段 Q_1Q_2 上到点 P 最近的点。



特殊情况

然后分情况讨论：

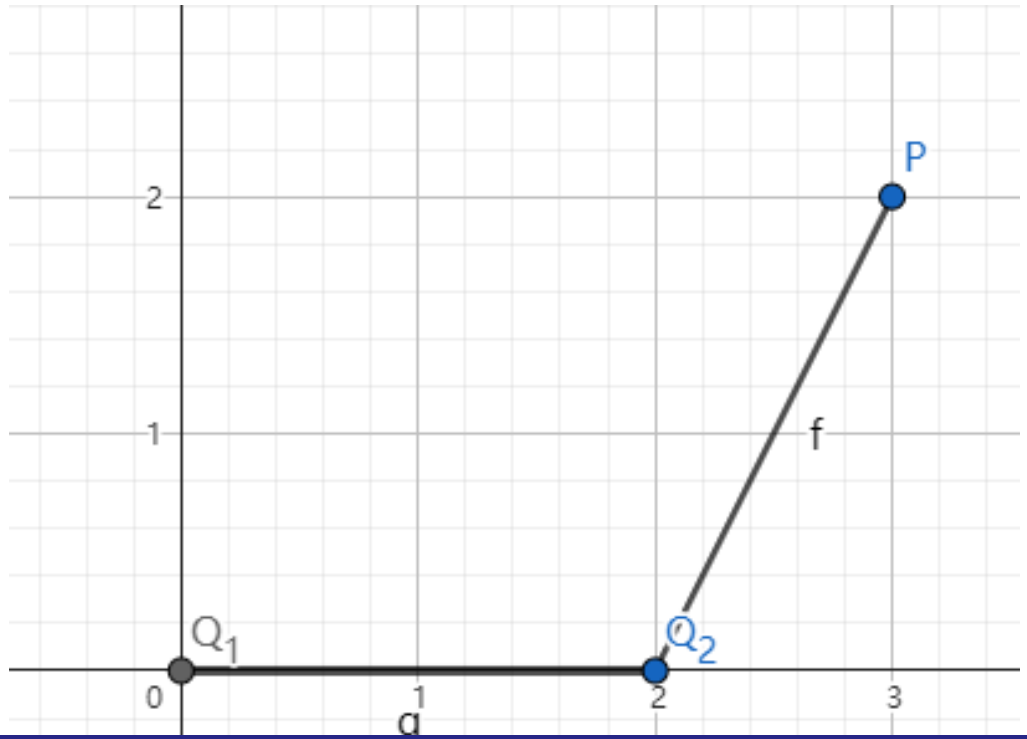
2. 点 P 的x坐标在 0 和点 Q_2 的x坐标之间。过点 P 作 $PM \perp Q_1Q_2$ 于点 M ，显然点 M 的x坐标与点 P 相同，y坐标为 0 。易证该点到点 P 的距离最小。



特殊情况

然后分情况讨论：

3. 点 P 的 x 坐标大于点 Q_2 的 y 坐标，此时点 Q_2 到点 P 的距离最小。



特殊情况

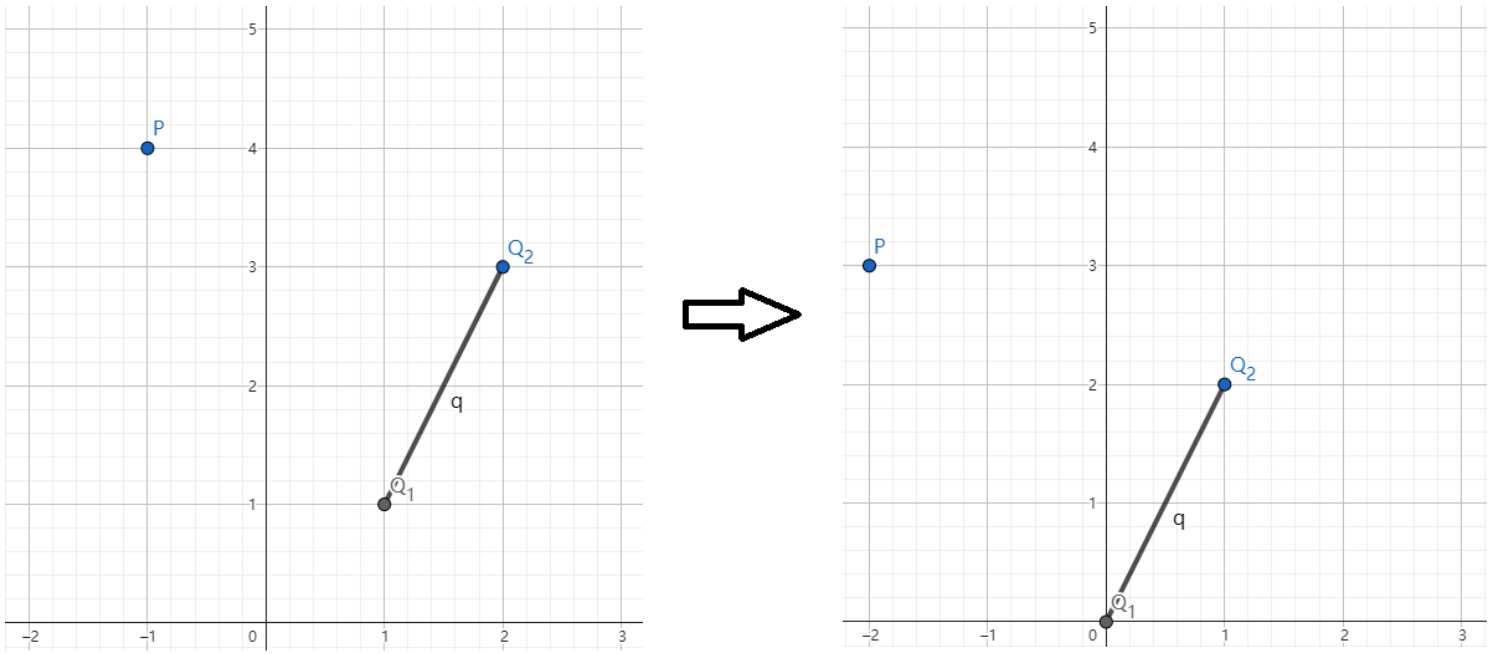
我们似乎做完了这道题。
但是别忘了，我们一开始假设 Q_1 为原点，Q_2 在 x 轴正方向上，对于一般的情况，怎么处理呢?

一般情况

我们似乎做完了这道题。
但是别忘了，我们一开始假设 Q_1 为原点，Q_2 在 x 轴正方向上，对于一般的情况，怎么处理呢?
这时候**平移旋转**就派上了用场。

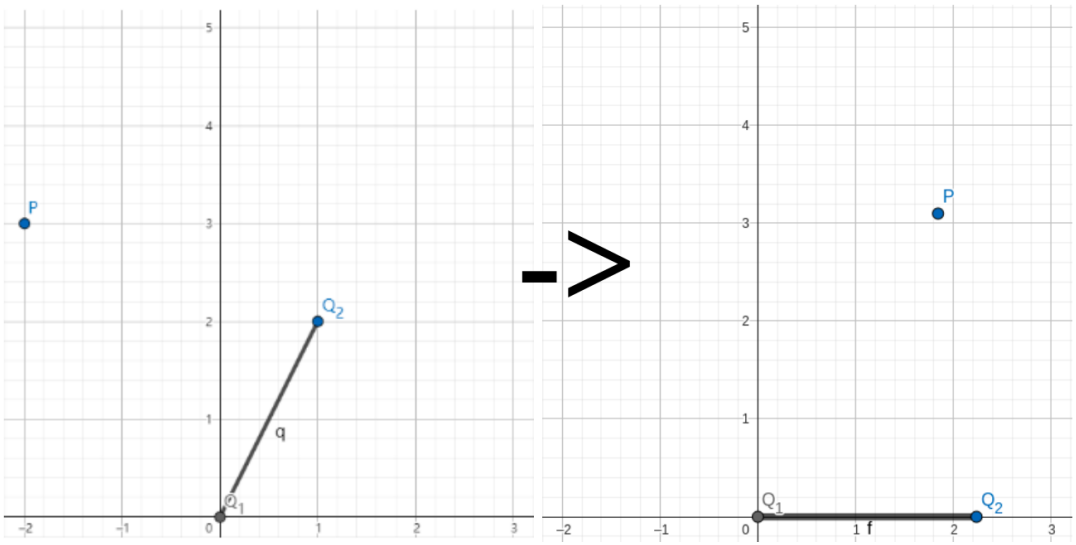
一般情况

首先第一步，我们将点 Q_1 平移到原点上。注意，为了保证三个点之间的相对位置关系不变，点 Q_2 和 P 也要跟着平移相同的方向和长度，如下图所示。



一般情况

然后，我们将点 Q_2 绕点 Q_1（现在 Q_1 位于原点！）旋转至 x 轴正方向。注意，为了保证三个点之间的相对位置关系不变，点 P 也要旋转相同的角度，如下图所示。（容易发现，将点 P 和 Q_2 绕原点顺时针旋转 **点 Q_2 的极角**即可达到目标。）



一般情况

这样，一般情况就被转化成了特殊情况，调用特殊情况下的求解方法，即可得到答案。
注意，在特殊情况下，求得的答案是**平移旋转后**的答案，因此需要将求得的点**平移旋转回去**，才是原问题的答案。
时间复杂度 $O(1)$ 。

代码实现

```cpp
// 将点 p 绕原点逆时针旋转 theta，利用旋转公式
inline Point rot(Point p,double theta){
    Point q;
    q.x = p.x * cos(theta) - p.y * sin(theta);
    q.y = p.y * cos(theta) + p.x * sin(theta);
    return q;
}

int main(){
    Point p,q1,q2;
    scanf("%lf%lf%lf%lf%lf%lf",&p.x,&p.y,&q1.x,&q1.y,&q2.x,&q2.y);
    // 将点 q1 平移到原点，同时p,q2也要跟着平移。
    // 这里不要将q1的横纵坐标都设成0，因为待会儿平移回来要用，你只需要想象q1在原点就行了
    p.x -= q1.x, p.y -= q1.y;
    q2.x -= q1.x, q2.y -= q1.y;
    double theta = atan2(q2.y,q2.x); // 点 q2 的极角
    q2 = rot(q2,-theta), p = rot(p,-theta); // 将点q2旋转到x轴正方向，点p也跟着旋转
```

代码实现

```
28      // 处理特殊情况
29      Point ret;
30      if (p.x < 0)ret = Point(0,0);
31      else if (p.x <= q2.x)ret = Point(p.x,0);
32      else ret = q2;
33      // 旋转回去
34      ret = rot(ret,theta);
35      // 平移回去
36      ret.x += q1.x;
37      ret.y += q1.y;
38      printf("%.3lf %.3lf",ret.x,ret.y);
39      return 0;
40  }
```

例题

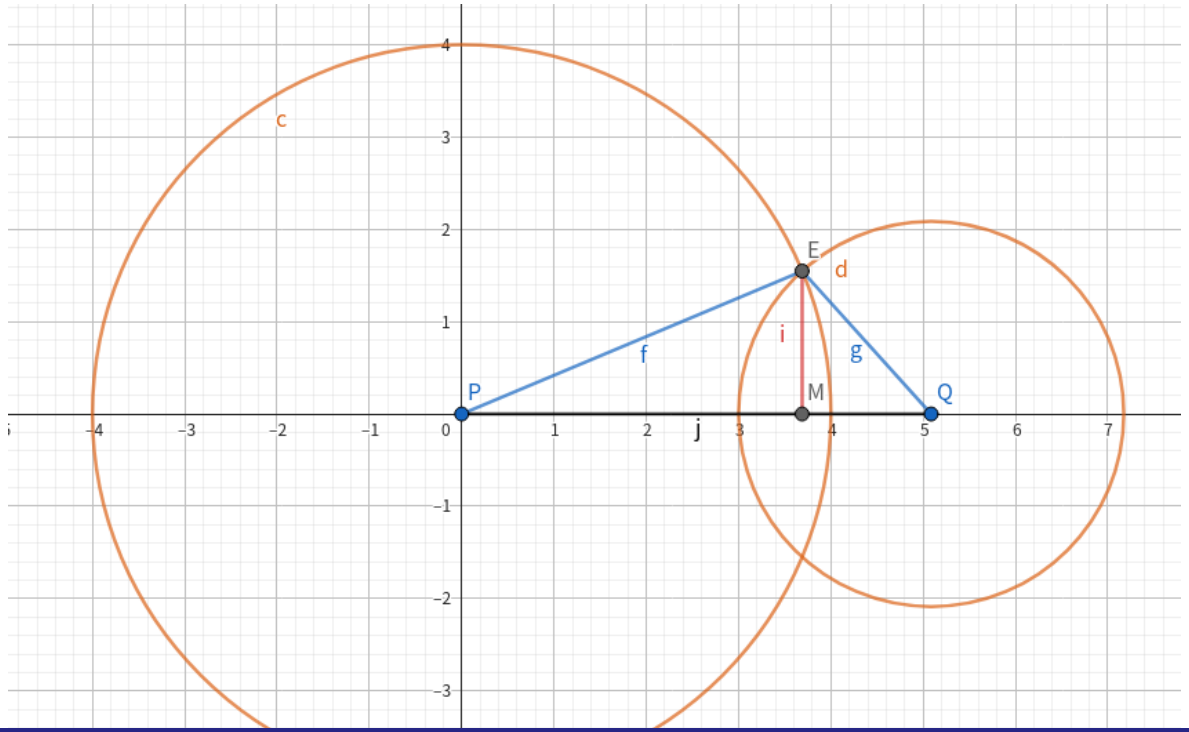
给定两个圆，求两圆交点坐标。

解法

设两个圆的圆心分别为 P ， Q 。
与上一题类似：首先将 P 平移到坐标原点，然后将 Q 旋转到 x 轴正方向。

解法

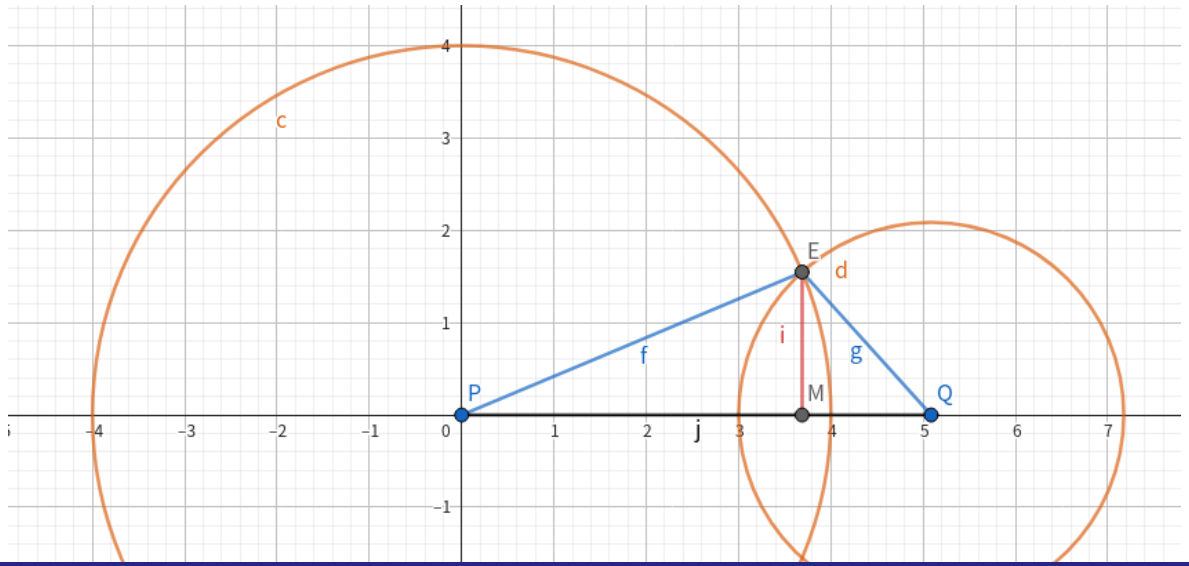
显然两个圆有可能交于两点，并且旋转后，两点一定关于 x 轴对称。我们设 x 轴上方的交点为 E 。
连接 PE , QE , PQ ，作 $EM \perp x$ 轴于点 M ，如下图所示。



解法

容易发现，PQ 是两圆圆心的距离，PE 是圆 P 的半径，QE 是圆 Q 的半径，这三个长度都是已知的。而我们要求的是 EM，也就是 ΔPQE 中 PQ 边上的高。

考虑**等面积法**，我们可以利用《海 伦 公 式》求出 ΔPQE 的面积 S，则 $\frac{2S}{PQ}$ 即为 EM 的值。



解法

求出 EM 的值后，我们就得到了 E 的纵坐标，然后利用勾股定理，我们就可以求出 E 的横坐标了。
另一个交点与 E 关于 x 轴对称，直接求即可。
最后平移旋转回去，即可得到答案。

处理特殊情况

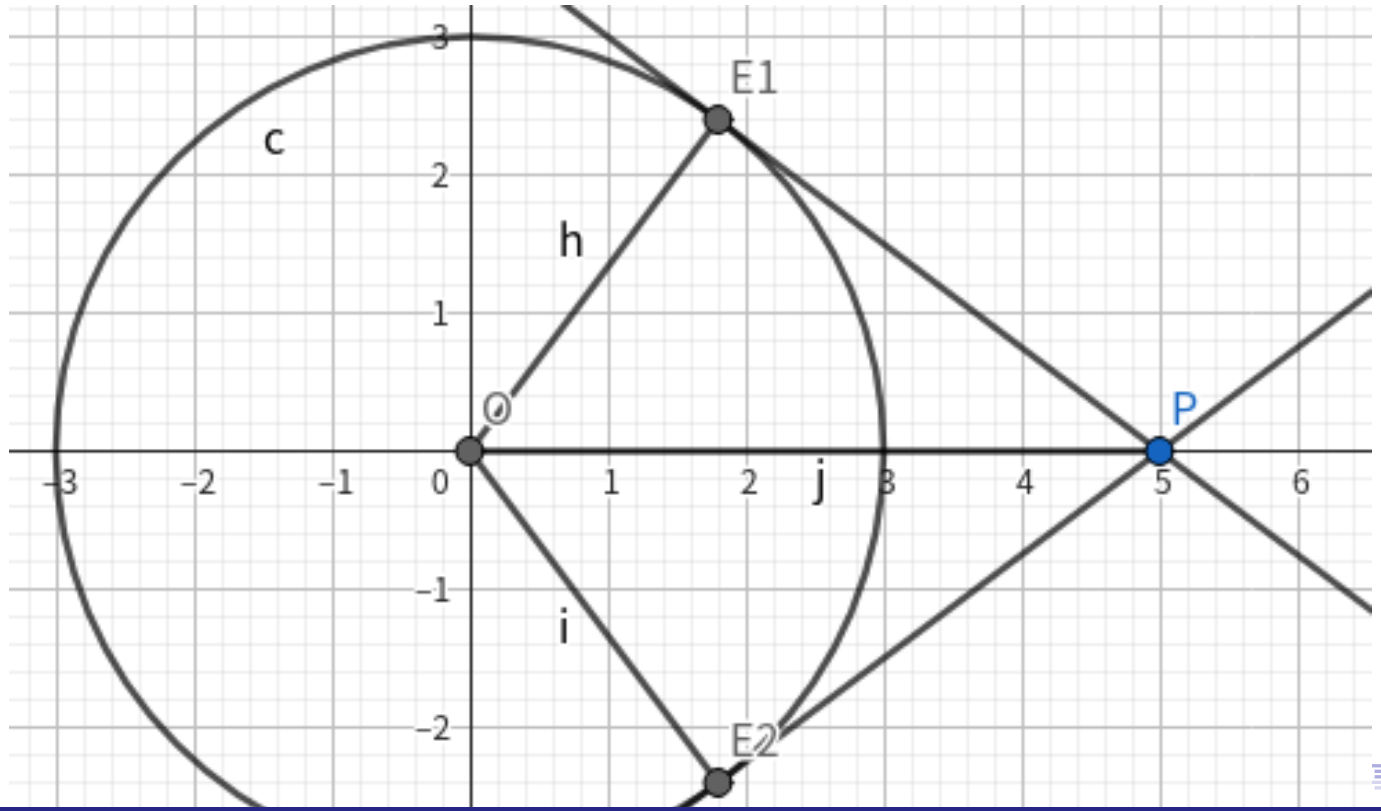
当然，两个圆不一定有两个交点，有可能只有一个交点，或者没有交点。
一个交点的情况比较好判断。
对于没有交点的情况，可以证明该情况下 PE , QE, PQ 三个线段的长度一定无法构成三角形。直接判断即可。
当然本题可以利用余弦定理直接算，这里不再赘述。

例题

给定一个点和一个圆，求点到圆的切点。

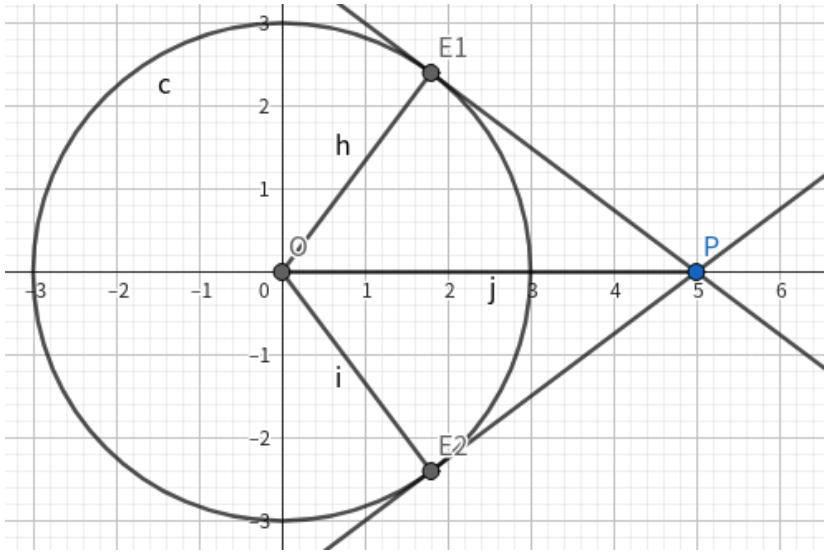
解法

设该点为 P，圆心为 O，两个切点分别为 E_1, E_2。
由于切线一定与半径垂直，所以 $PO \perp OE_1$ 一定成立。



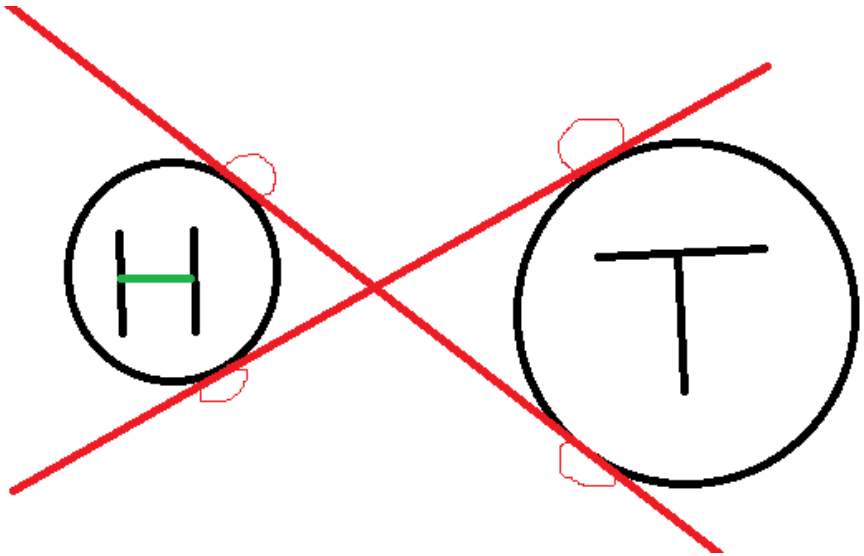
解法

我们已知 OP 和 OE_1 的长度，又因为 $PO \perp OE_1$ ，所以我们可以直接利用勾股定理计算出 PE_1 的长度。
这样，切点问题就被转化成两圆交点问题，直接调用上一题的解法即可。



例题

给定两个圆，求它们的内公切线。
两个圆的内公切线指的是两条直线 l_1, l_2 ，使得 l_1 、l_2 都同时与两个圆相切，且 l_1 , l_2 与两个圆圆心的连线都相交，如下图所示。

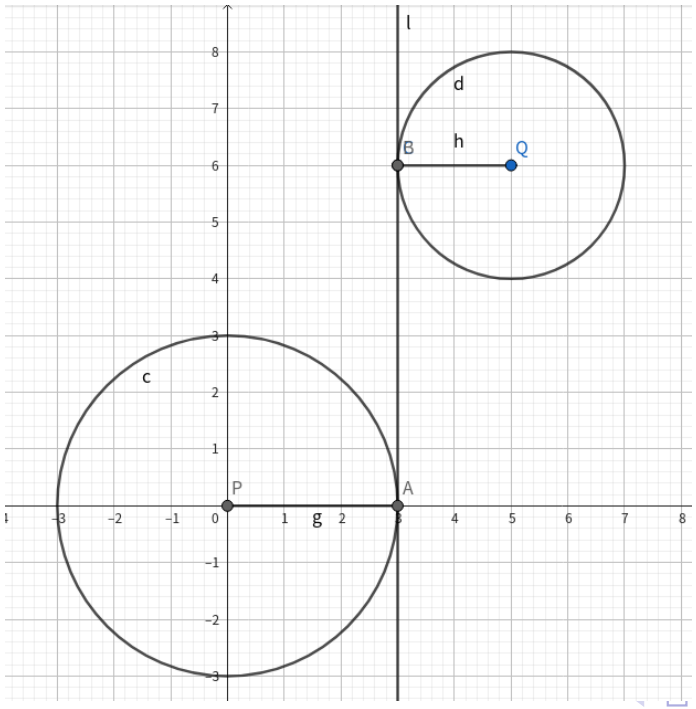


解法

如果像前几题那样，将一个圆的圆心平移到原点，另一个旋转到 x 轴正方向，你会发现答案依然是不好述的。
于是，你需要开动人类智慧：

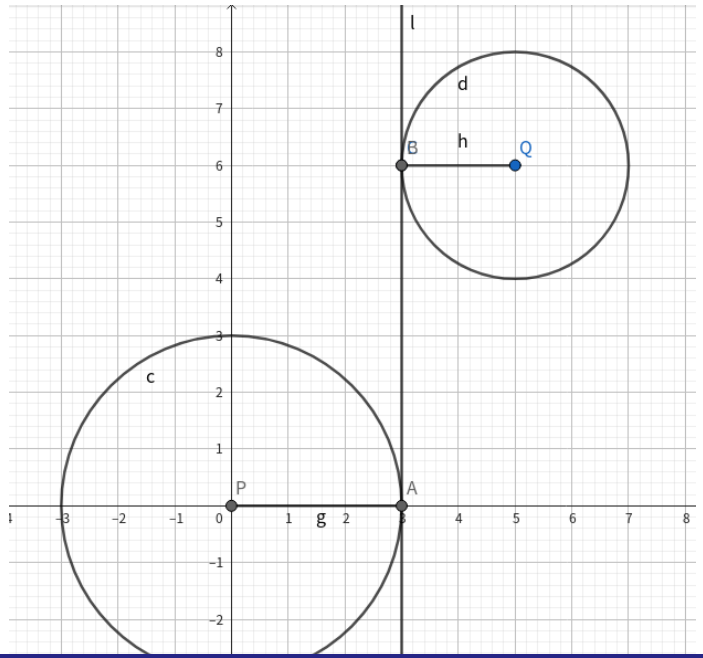
解法

设两个圆的圆心分别为 P, Q，半径分别为 R_p, R_q。我们首先将点 P 平移到原点，然后过点 $A(R_p, 0)$ 作 x 轴的垂线 l，将点 Q 绕原点旋转，使得圆 Q 与直线 l 相切，如下图所示。



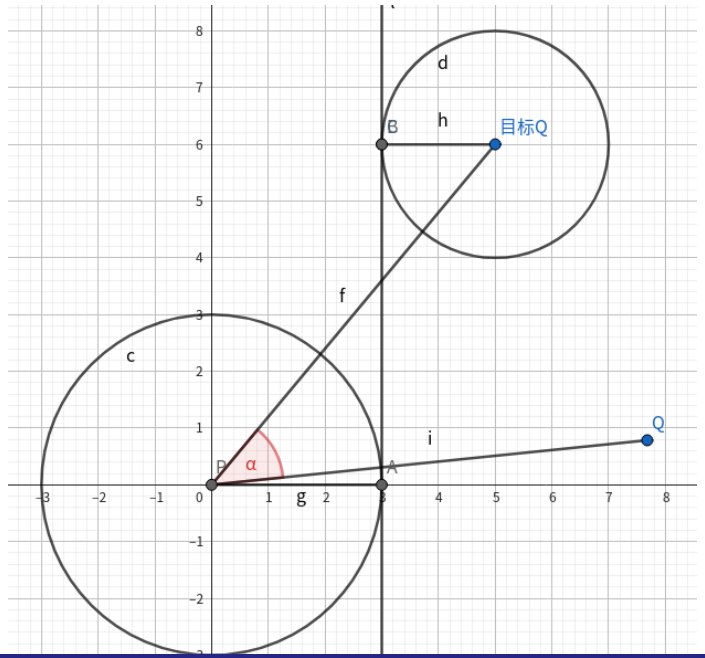
解法

然后你发现，直线 l 恰好是两个圆的内公切线，且此时点 Q 的横坐标为 $R_p + R_q$ 。
两个圆圆心之间的距离是已知的，我们可以利用勾股定理直接求出旋转后点 Q 的纵坐标 y 。



解法

我们求出内公切线后，需要旋转回去，才能得到真正的答案。但是问题来了：如何求出旋转的度数?
容易发现，旋转的度数（下图中的角 α ）恰好是 $(R_p + R_q, y)$ 的极角与（平移之后）点 Q 的极角的差。



例题

由于《雨中的相遇》题解已经发在课程文件里了，这里不再讲述。

简介

向量叉乘是计算几何中一种重要的运算。
注意：此处默认大家都学过向量，如果有人没学过向量，可自学向量后，再次阅读本部分论文。

定义

对于两个向量 $\mathbf{a}, \mathbf{b}$ ，它们的夹角为 θ ，则两个向量**叉乘**后的结果为：

$$\mathbf{a} \times \mathbf{b} = |\mathbf{a}||\mathbf{b}|sin\theta$$

注意，此处θ 是 **有向夹角**：如果向量 $\mathbf{a}$ 在向量 $\mathbf{b}$ 的 **逆时针** 方向，则 θ 的值为 **负数**；如果向量 $\mathbf{a}$ 在向量 $\mathbf{b}$ 的 **顺时针** 方向，则 θ 的值为 **正数** 。

定义

在向量的坐标表示法中：
对于点 $A(x_1, y_1)$ ，$B(x_2, y_2)$ ，则如下等式成立：

$$\overrightarrow{OA} \times \overrightarrow{OB} = x_1 y_2 - y_2 x_1$$

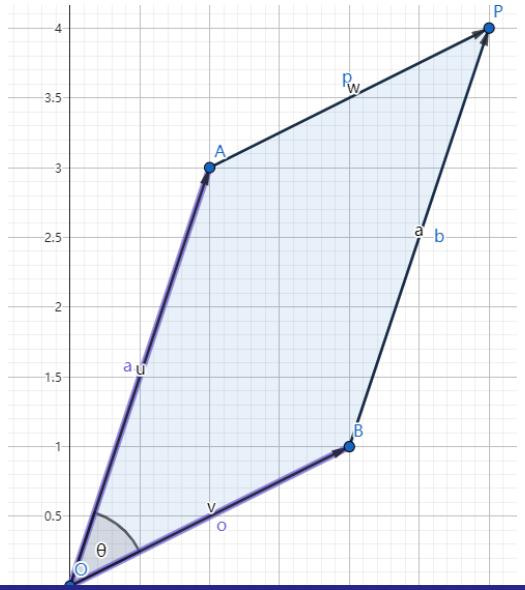
即两个点横纵坐标交叉相乘，再相减。

几何意义

两个向量叉乘的几何意义表示由这**两个向量张成的平行四边形的面积**，如下图所示。

（叉乘后的结果可能是负的，取**绝对值**之后才是真正的面积）。

该面积除以 2，即为两个向量张成的**三角形**的面积。



作用

1. 求面积。
2. 判断一个向量在另一个向量的顺时针方向还是逆时针方向。

例题

给定三个点 A , B , C ， 求 ΔABC 的面积。

解析

由叉乘的定义，显然答案为 $\frac{1}{2}|(\overrightarrow{OB}-\overrightarrow{OA})\times(\overrightarrow{OC}-\overrightarrow{OA})|$ 。
其中 $-$ 表示向量减法。

例题

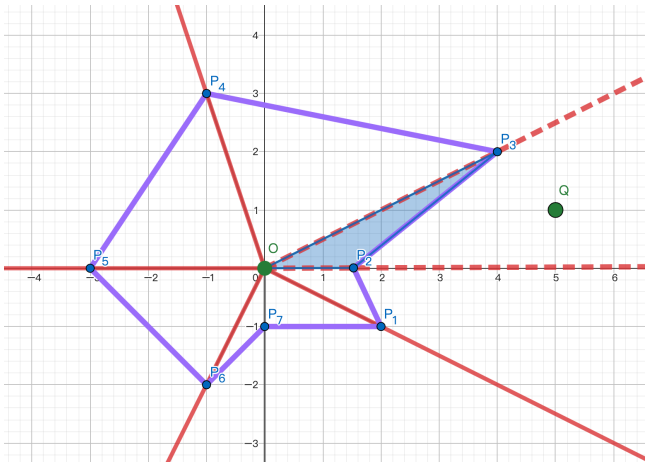
给定一个正 n 边形和 q 次询问，每次询问给定一个点 P ，你需要判断点 P 是否在该多边形内部。**保证多边形的各个顶点按极角从小到大排序**。
$3 \leq n \leq \sqrt{5} \times 10^5, 1 \leq q \leq \sqrt{5} \times 10^5$ 。

解析

设该多边形的各个顶点依次为 $P_1, P_2, ..., P_n$ 。
一般情况下，判断一个点 Q 是否在多边形内部的时间复杂度是 $O(n)$ 的，方法是：首先求出该多边形的面积S（下文会讲多边形的面积公式），然后将点 P 与多边形相邻两个顶点连成三角形，求出这些三角形的面积之和（即 $S_{\Delta QP_1P_2} + S_{\Delta QP_2P_3} + ... + S_{\Delta QP_nP_1}$ ），判断其是否等于 S 即可。时间复杂度 $O(n)$ 。
由于下文要用到这种方法，所以这里称上述方法为**面积法** 。
当然，判断一个点是否在多边形内部的方法有很多（参考 OI Wiki），但是时间复杂度都不低于 $O(n)$ 。那么，本题的总时间复杂度为 $O(nq)$ ，无法通过。

解析

但是别忘了：本题的多边形的顶点是按照极角序排序的！
我们发现，作射线 $OP_1, OP_2.OP_3, ..., OP_n$ 后，整个平面被分成了若干个区域（下图红线）。
我们可以利用二分法快速求出点 Q 在哪两条相邻的射线之间（下图中的两条虚线）。



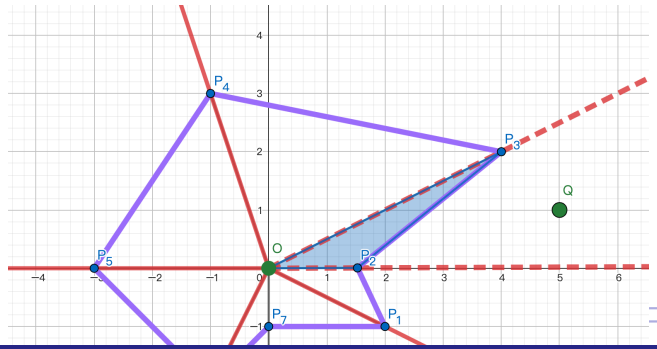
解析

假设点 Q 位于射线 OP_i 和 $OP_{(i \bmod n)+1}$ 之间。容易发现，点 Q 位于该多边形内部，当且仅当点 Q 位于 $\Delta OP_iOP_{(i \bmod n)+1}$ 内部（下图中的蓝色三角形）。

这样，判断一个点是否在多边形内部就被转化成了判断一个点是否在**三角形**内部。使用上文所说的**面积法**，时间复杂度为 $O(3)$，可以接受。

面积法需要求三角形的面积，直接叉乘即可。

单次询问时间复杂度为 $O(logn)$，所以总复杂度为 $O(qlogn)$。



思考

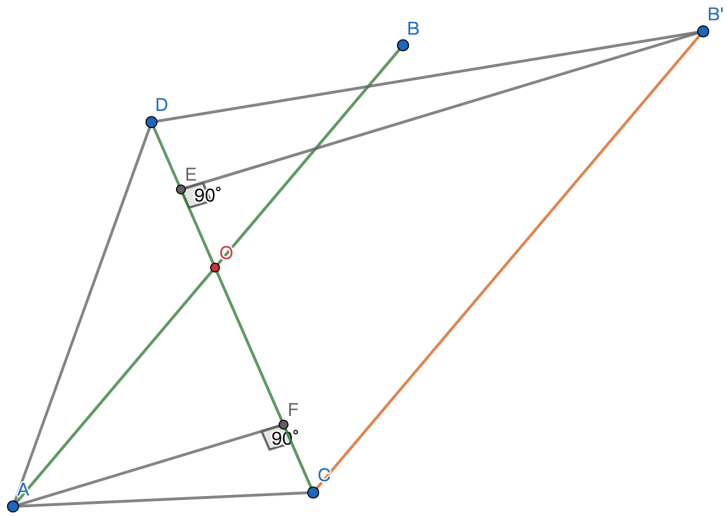
思考：为什么多边形必须按照极角序排序，才能使用这种方法？一般的多边形不行吗？你能举出反例吗？

例题

给定四个点 A, B, C, D ， 求直线 AB 与 CD 的交点。

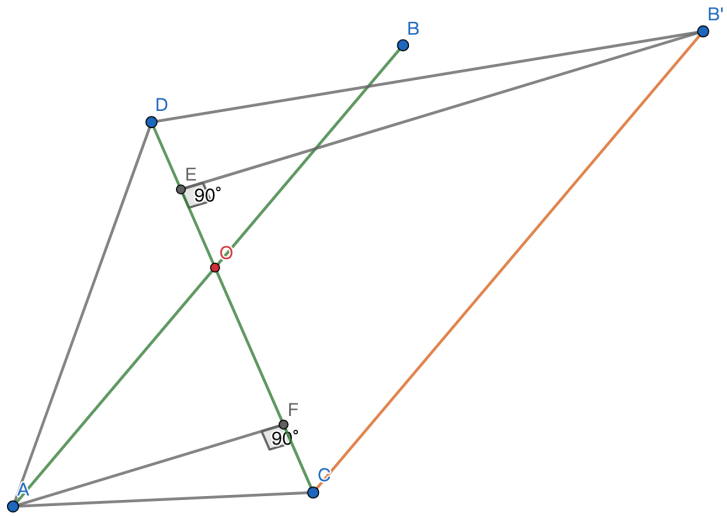
解析

设 AB 与 CD 交于点 O 。我们先考虑一种简单的情况——**线段** AB 与 **线段** CD 有交点。
将线段 AB 平移，使得点 A 与点 C 重合，得到线段 CB' 。连接 $AC, AD, B'D$ 。
作 $B'E \perp CD$ 于点 E ，$AF \perp CD$ 于点 F ，如下图所示。



解析

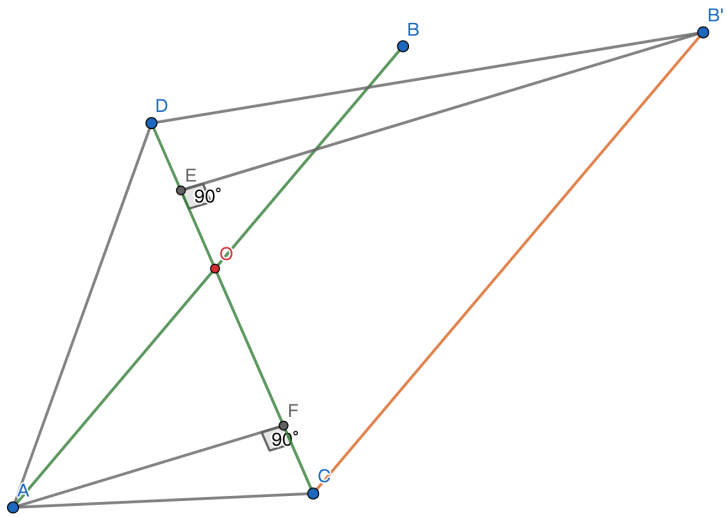
容易发现，ΔAFO $\Delta B'EC$ ，故 $\frac{AO}{B'C} = \frac{AF}{B'E}$ 。
又因为 AF, BE 分别是 $\Delta ACD, \Delta B'CD$ 的高，而这两个三角形是等底的，所以 $\frac{AF}{BE} = \frac{S\Delta ACD}{S\Delta B'CD}$ 。



解析

所以 $\frac{AO}{B'C} = \frac{S\Delta ACD}{S\Delta B'CD} = \frac{\overrightarrow{AC}\times\overrightarrow{AD}}{\overrightarrow{C'B}\times\overrightarrow{CD}}$ 。

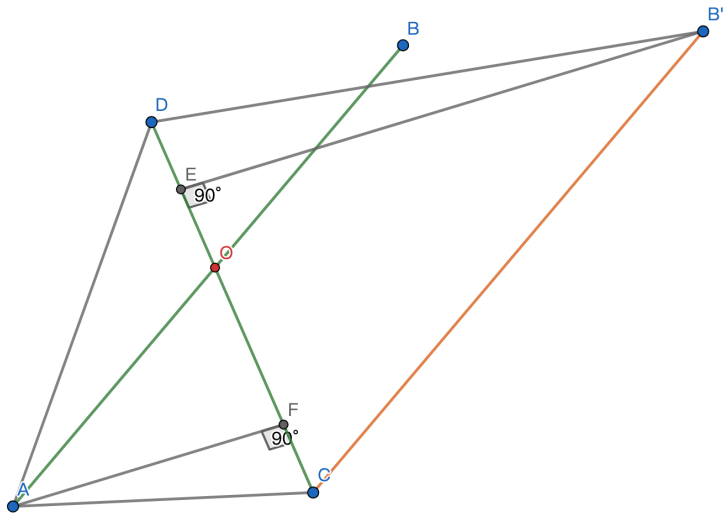
注意，此处不需要取绝对值——可以证明，$\overrightarrow{AC}$ 相对于 $\overrightarrow{AD}$ 的方向（顺时针、逆时针）一定与 $\overrightarrow{C'B}$ 相对于 $\overrightarrow{CD}$ 的方向相同，因此上下符号一致，得到的是正数。



解析

又因为向量是可以任意平移的，所以 $\overrightarrow{CB'} = \overrightarrow{AB}$ 。

所以 $\frac{AO}{AB} = \frac{\overrightarrow{AC} \times \overrightarrow{AD}}{\overrightarrow{AB} \times \overrightarrow{CD}}$ 。



解析

求出 $\frac{AO}{AB}$ 后，显然答案为 $A + \overrightarrow{AB} \cdot \frac{AO}{AB}$ 。
这样，我们就得到了两条直线的交点公式：

$$O = A + \overrightarrow{AB} \cdot \frac{\overrightarrow{AC} \times \overrightarrow{AD}}{\overrightarrow{AB} \times \overrightarrow{CD}}$$

。
实际上，该公式在**线段** AB 与**线段** CD 不相交时依然是成立的，读者自证不难。

解析

特别地，AB 与 CD 可能不存在交点（平行）。此时 $\overrightarrow{AB}$ 与 $\overrightarrow{CD}$ 一定是共线向量，所以它们的夹角 θ 等于 0，所以 $sin\theta$ 也等于 0，所以 $\overrightarrow{AB} \times \overrightarrow{CD} = |AB| \cdot |CD| \cdot sin\theta = 0$。
实际上，判断向量共线的常用方法，就是判断它们的叉积是否等于 0。

多边形面积公式

如果一个多边形的各个顶点分别为 $P_1, P_2, P_3, ..., P_n$ ，则该多边形的面积为：

$$\frac{1}{2}|\overrightarrow{OP_1} \times \overrightarrow{OP_2} + \overrightarrow{OP_2} \times \overrightarrow{OP_3} + ... + \overrightarrow{OP_n} \times \overrightarrow{OP_1}|$$

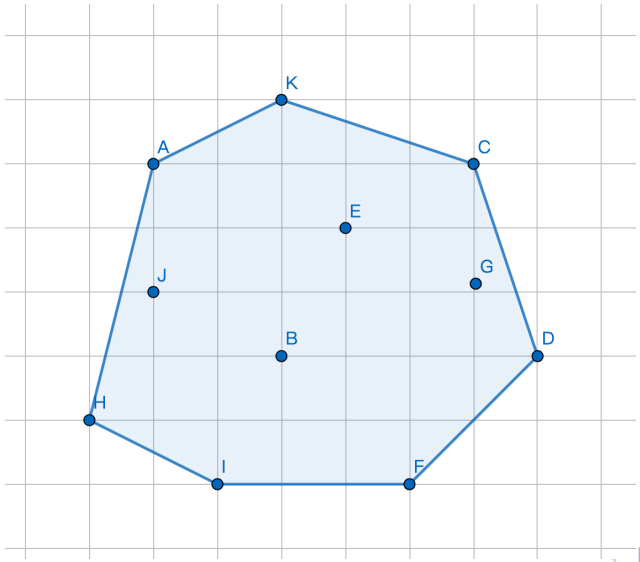
即

$$\frac{1}{2}|\sum_{i=1}^{n} \overrightarrow{OP_i} \times \overrightarrow{OP_{(i \ mod \ n)+1}}|$$

其中 O 为任意一点（一般取坐标原点）。

定义

凸包：包含所给点集中所有点的**最小凸多边形**叫凸包。
形象理解：假如你有一根橡皮筋，你先把它拉到无限大，能套住整个平面。把每个点看作一个钉子，松手后，橡皮筋会套在一些钉子上，橡皮筋形成的**凸多边形**就是凸包。
更形象的理解可参阅下图。



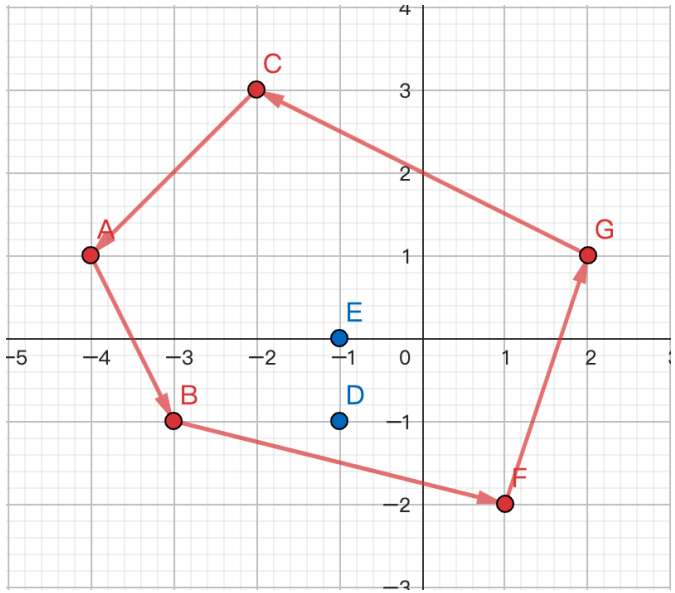
性质

凸包有许多优美的性质，比如平面最远点对一定在凸包上。（下文将会详细介绍这些性质）

凸包上点的性质

在求凸包之前，我们首先研究一下**凸包上的点**有什么性质。

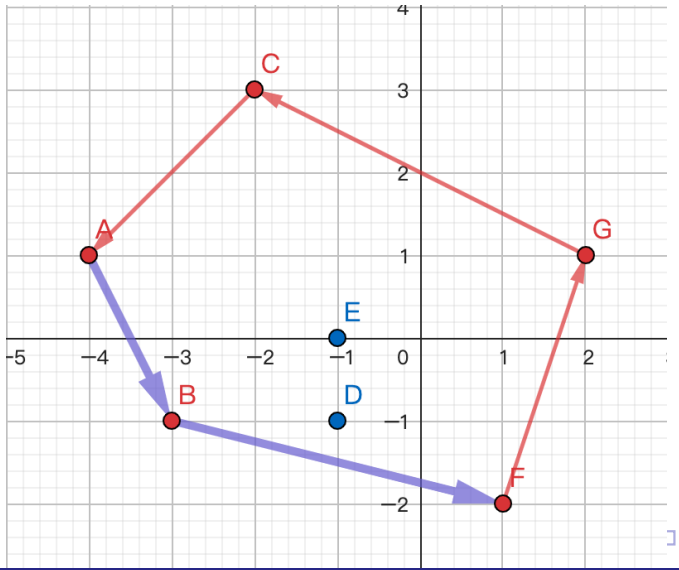
设凸包上的点**按逆时针顺序**分别为 $P_1, P_2, P_3, ..., P_n$ 。依次作向量 $\overrightarrow{P_1P_2}, \overrightarrow{P_2P_3}, \overrightarrow{P_3P_4}, ..., \overrightarrow{P_nP_1}$ ，如下图所示。（凸包上的点已经标注为红色）。



凸包上点的性质

任取两个相邻的向量 $\overrightarrow{PQ}, \overrightarrow{QM}$ ，可以发现一个**重要的性质**：$\overrightarrow{QM}$ 一定在 $\overrightarrow{PQ}$ 的**逆时针方向**。

例如下图中两条相邻的紫色向量（$\overrightarrow{AB}, \overrightarrow{BF}$）。我们发现，$\overrightarrow{BF}$ 在 $\overrightarrow{AB}$ 的**逆时针方向**，也就是说，将 $\overrightarrow{AB}$ 逆时针旋转不超过 $180°$ 后，能够与向量 $\overrightarrow{BF}$ 共线。



凸包上点的性质

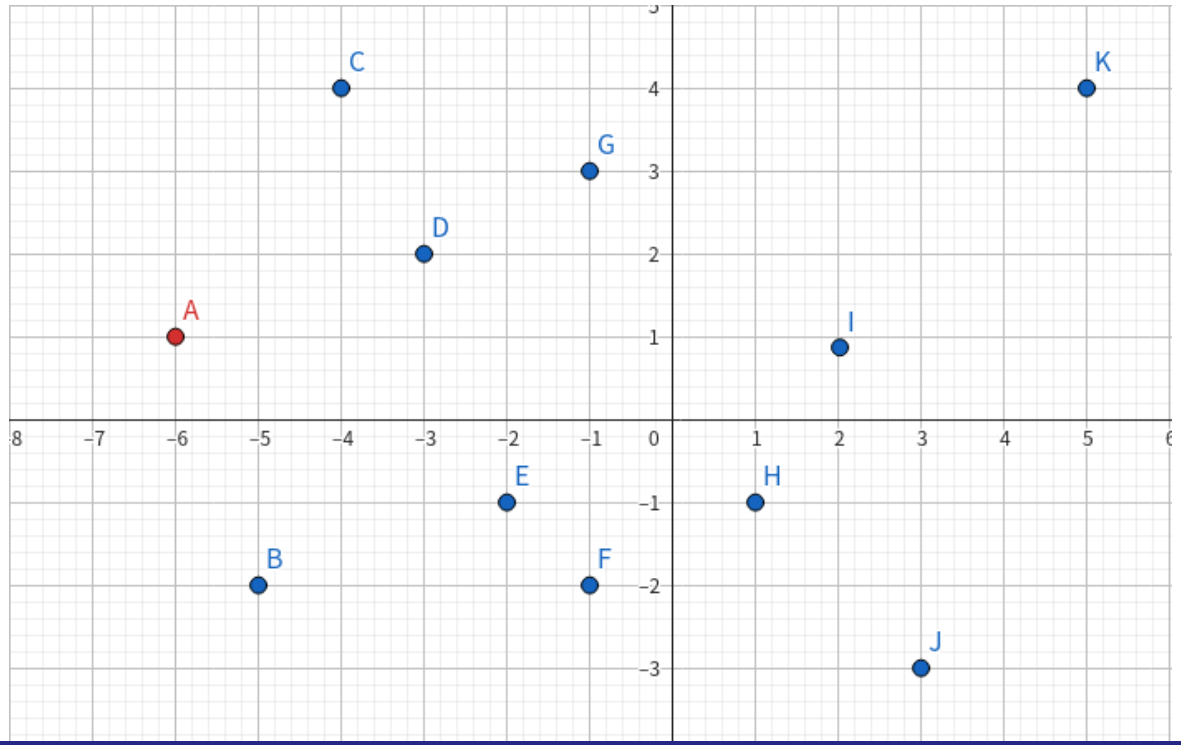
这个性质非常重要，几乎所有求凸包的算法都依赖于这个性质，包括下文所述的 Andrew 算法。
如何判断一个向量在另一个向量的顺时针方向还是逆时针方向?
可以使用**叉乘**，如果 $\mathbf{a} \times \mathbf{b} < 0$ ，则向量 $\mathbf{a}$ 在向量 $\mathbf{b}$ 的**逆时针方向**。

算法

其实求凸包的方法和**DP斜率优化**很相似，如果你理解了斜率优化，理解下面的算法将会十分容易。
首先，将所有点按照横坐标为第一关键字，纵坐标为第二关键字排序。此时你会发现，排在最开头的点（也就是最左侧的点）和排在最后的点（也就是最右侧的点）一定在凸包上。

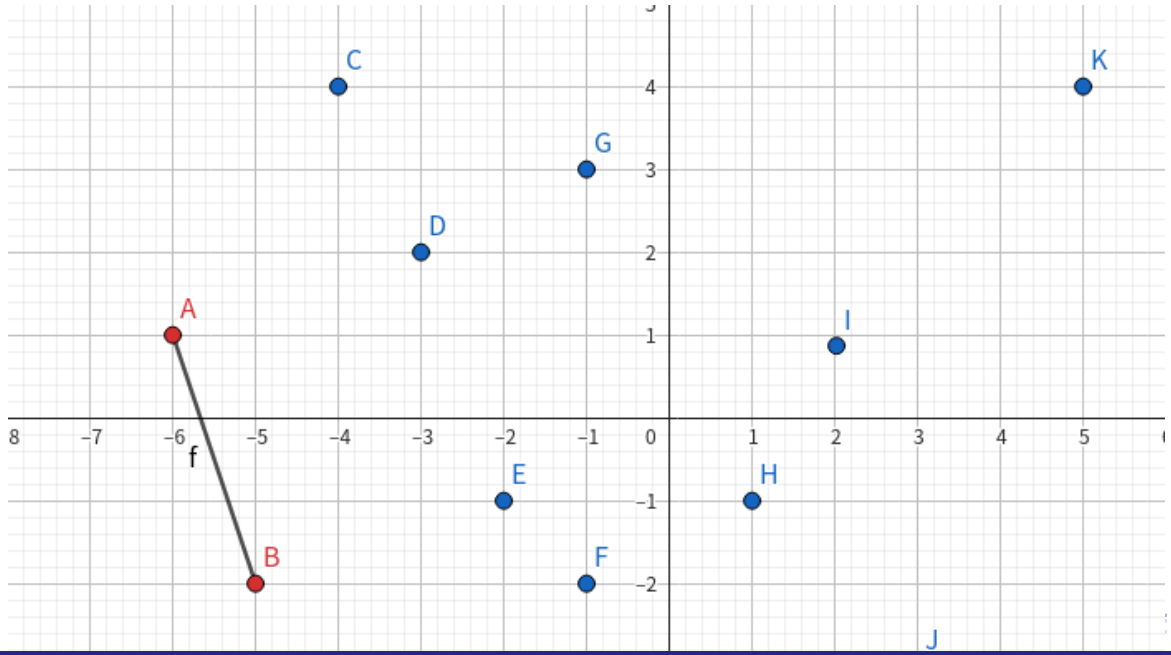
算法

我们维护一个**栈**，来记录凸包上的点。首先将最左侧的点入栈，并标记为凸包上的点（下图中的红色）。此时栈中的点为 $\{A\}$。



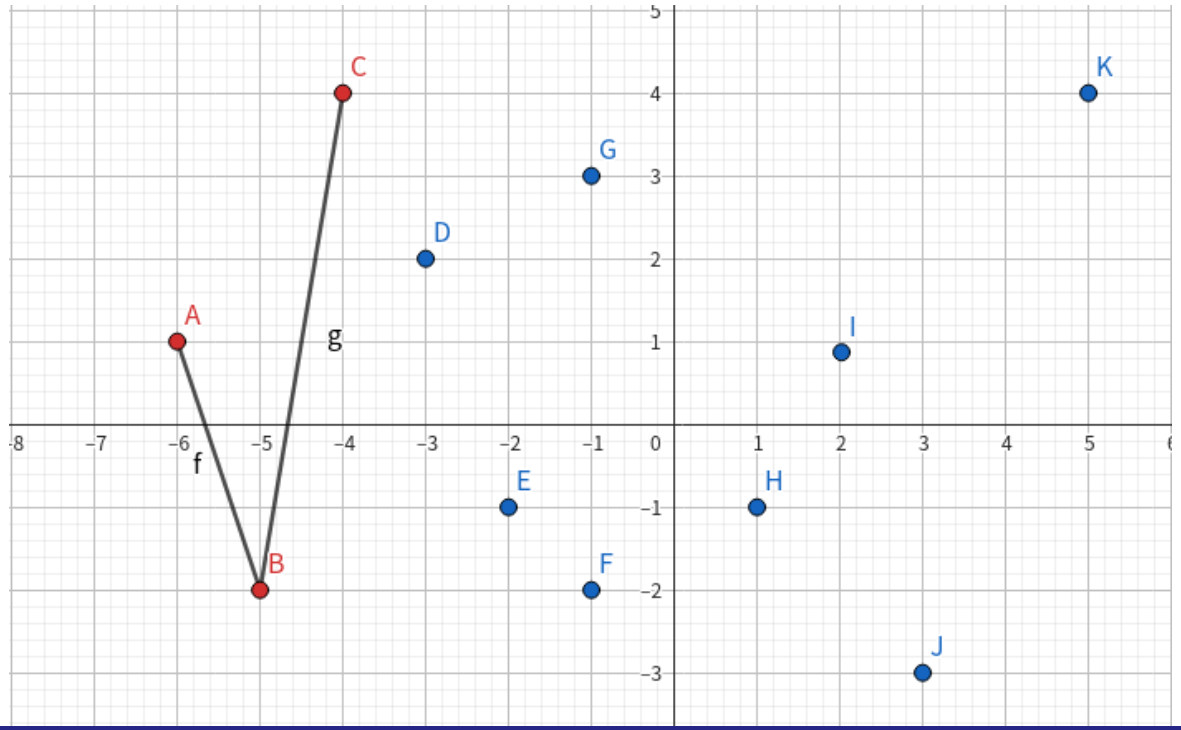
算法

按排序后的顺序依次处理各点。接下来处理点 B 。由于目前栈中只有一个点 A ，只会形成一个向量 $\overrightarrow{AB}$ ，因此无需考虑顺时针还是逆时针方向的问题，直接连接 AB ，将点 B 标记为凸包上的点，并入栈。此时栈中的点为 $\{A, B\}$ 。



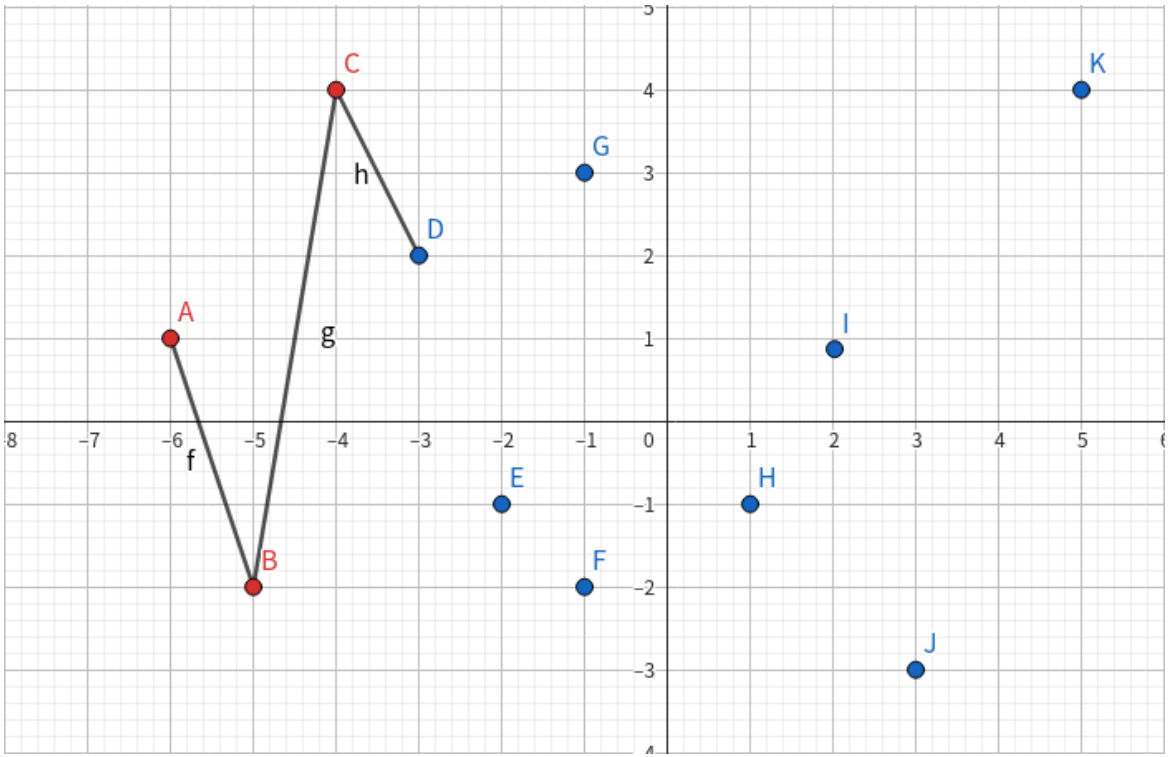
算法

接下来是点 C 。注意到向量 $\overrightarrow{BC}$ 在向量 $\overrightarrow{AB}$ 的**逆时针方向**，因此点 C **可能**成为凸包上的点，因此连接 BC ，并将点 C 入栈。栈中的点为 $\{A, B, C\}$ 。



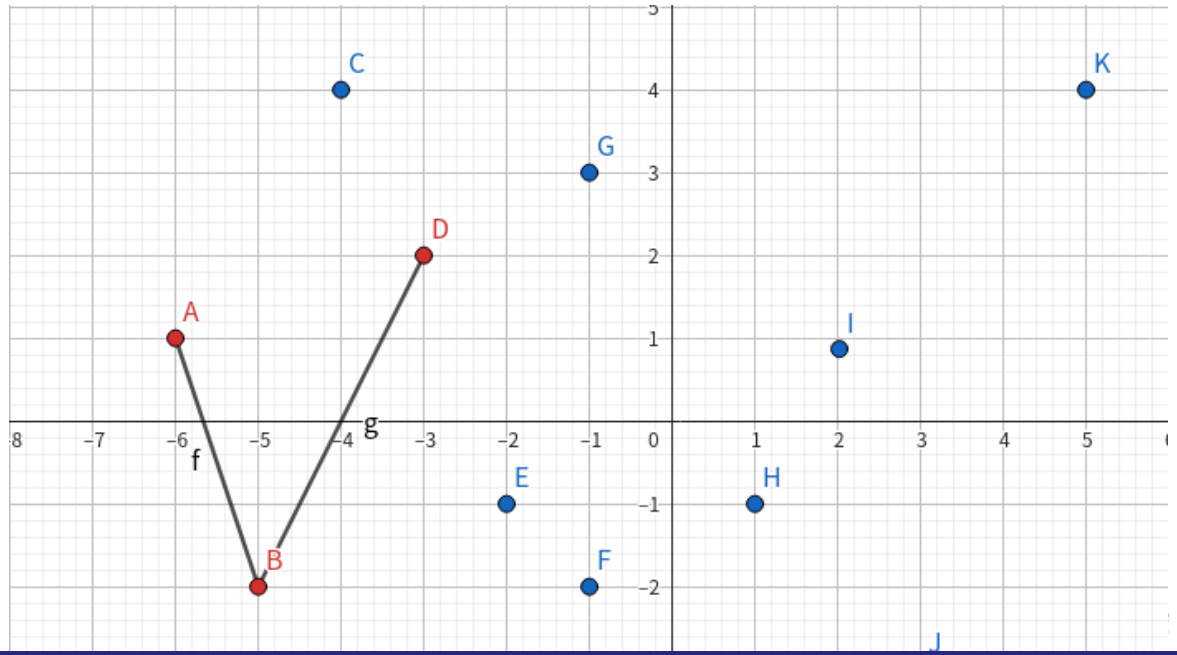
算法

接下来处理点 D，此时你发现：向量 $\overrightarrow{CD}$ 在向量 $\overrightarrow{BC}$ 的**顺时针方向**，不符合凸包的性质！（大悲



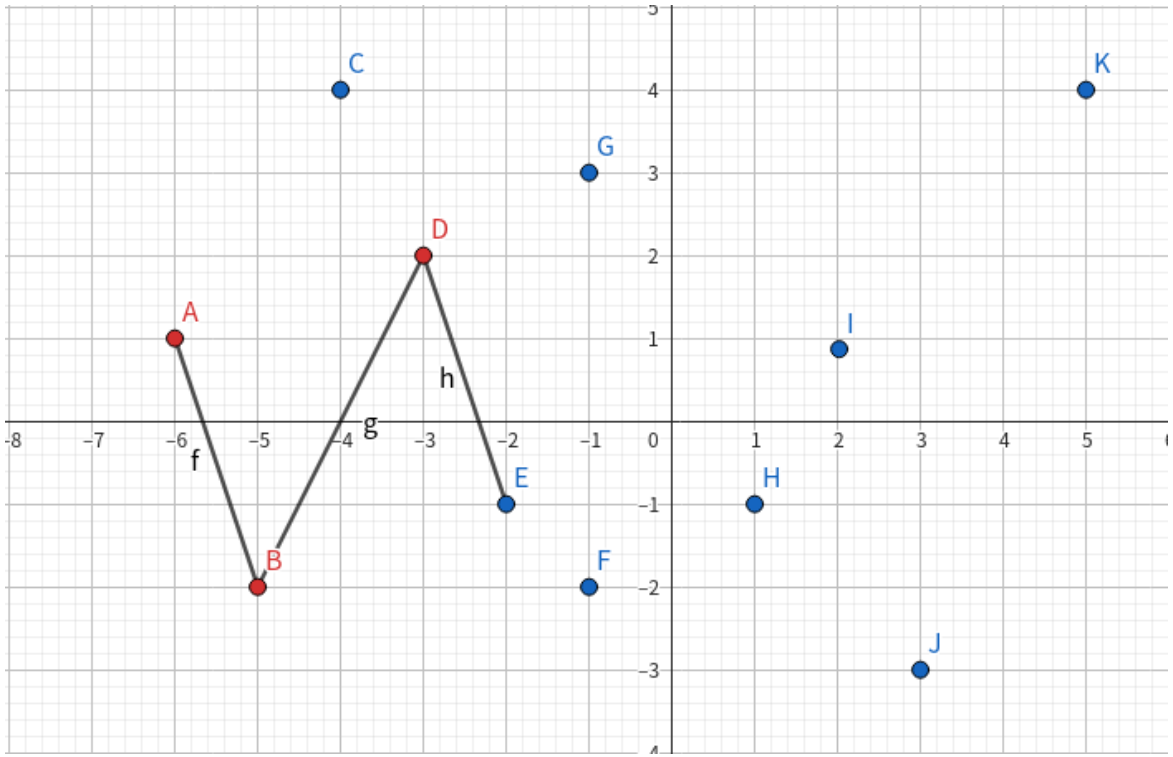
算法

遇到这种情况，我们将点 C 退栈，并标记为**不在凸包上的点**，直接将 D 的上一个点设置为点 B （新的栈顶）。连接 BD ，我们发现 $\overrightarrow{BD}$ 在 $\overrightarrow{AB}$ 的**逆时针方向**，一切恢复正常。将点 D 入栈，此时栈中的点为 $\{A, B, D\}$ 。



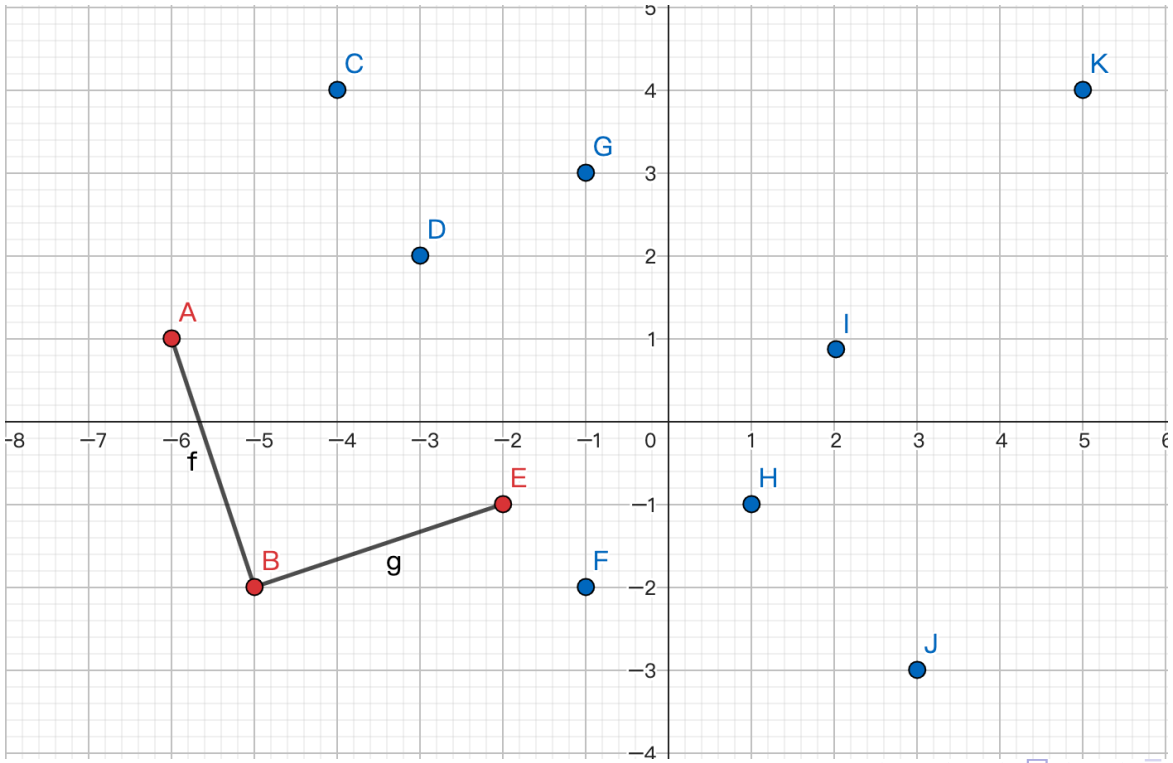
算法

继续处理点 E，连接 DE，发现 $\overrightarrow{DE}$ 又在 $\overrightarrow{BD}$ 的 **顺时针方向**。



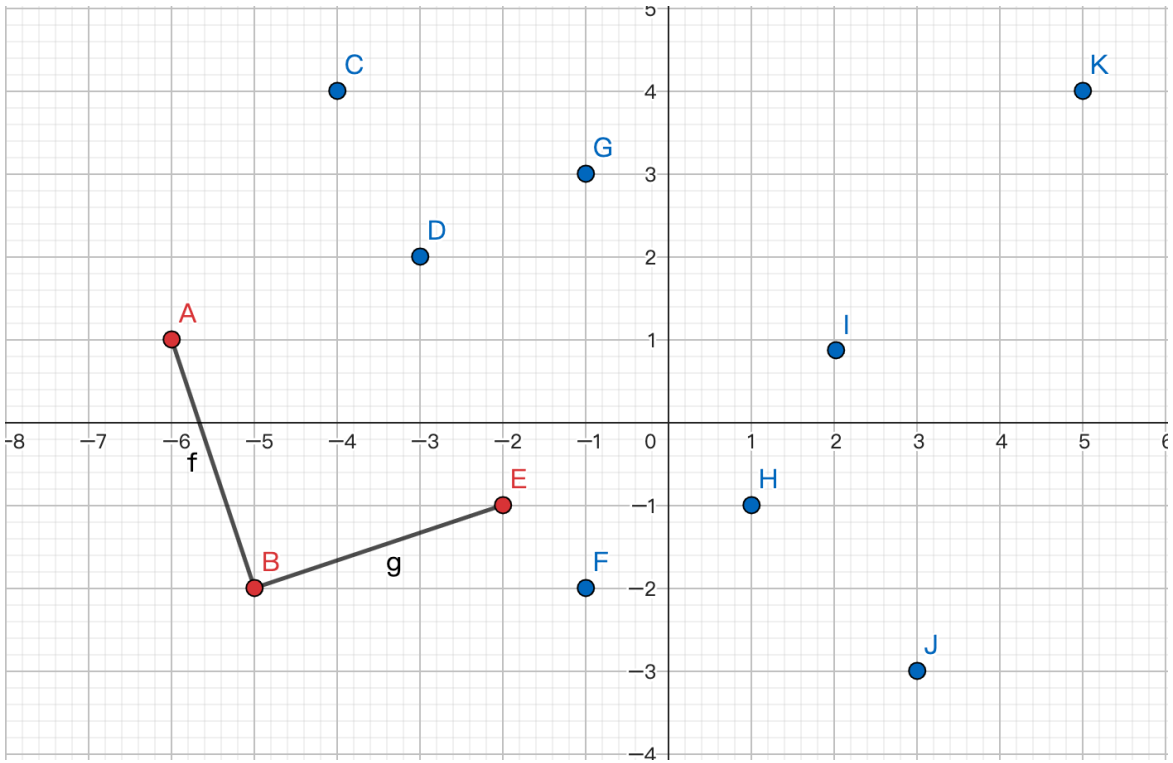
算法

将点 D 退栈，连接 BE ，$\overrightarrow{BE}$ 在 $\overrightarrow{AB}$ 的逆时针方向。将点 E 入栈，连接 BE 。此时栈中的点为 $\{A, B, E\}$ 。

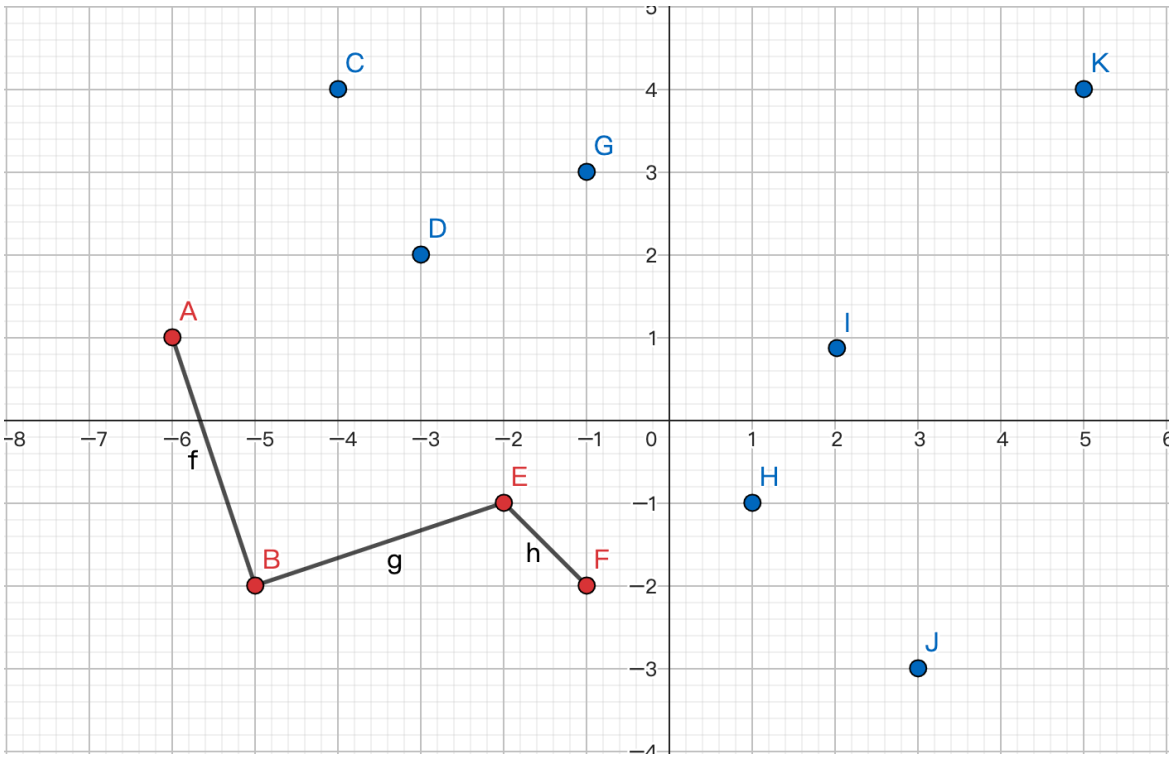


算法

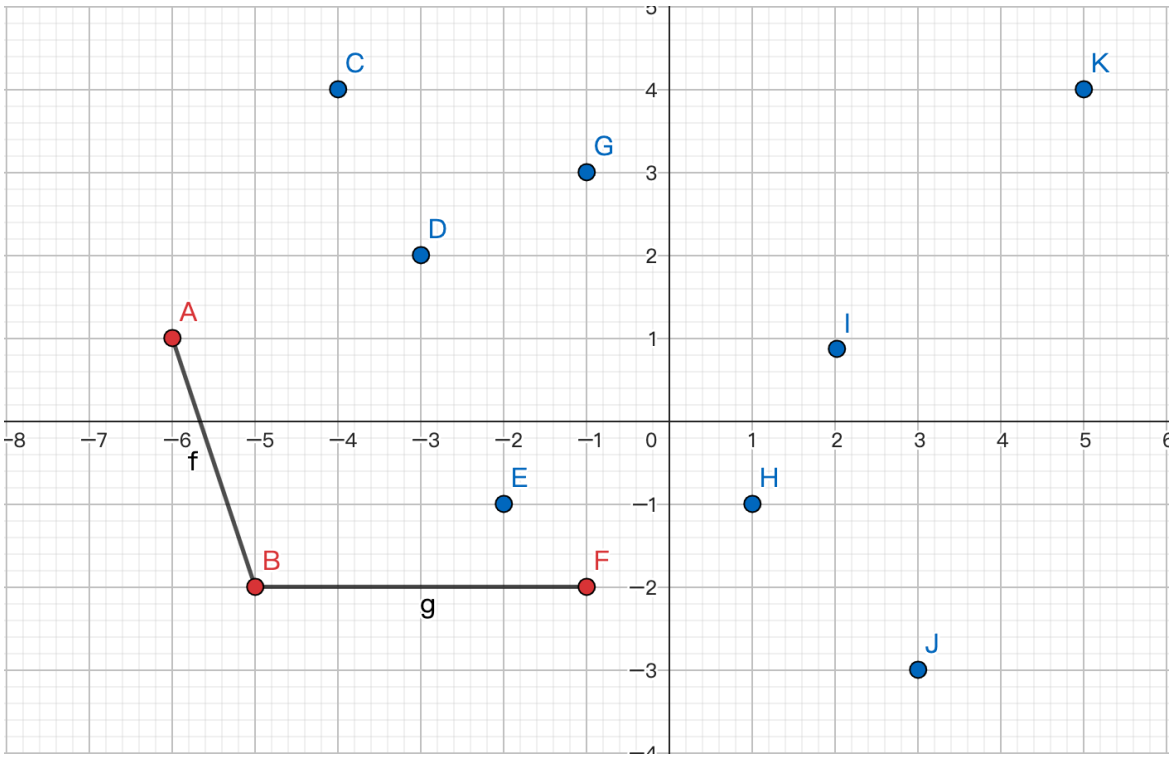
按照此流程继续处理到点 I 。（接下来是动画演示）



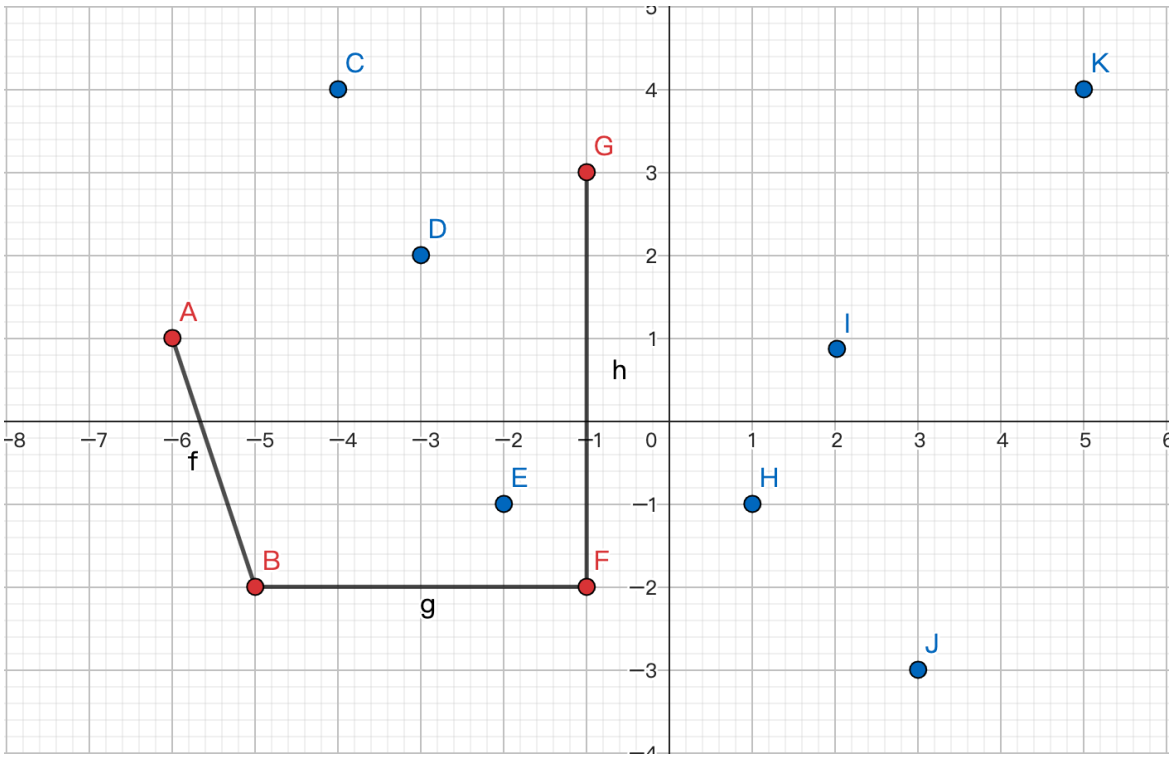
算法



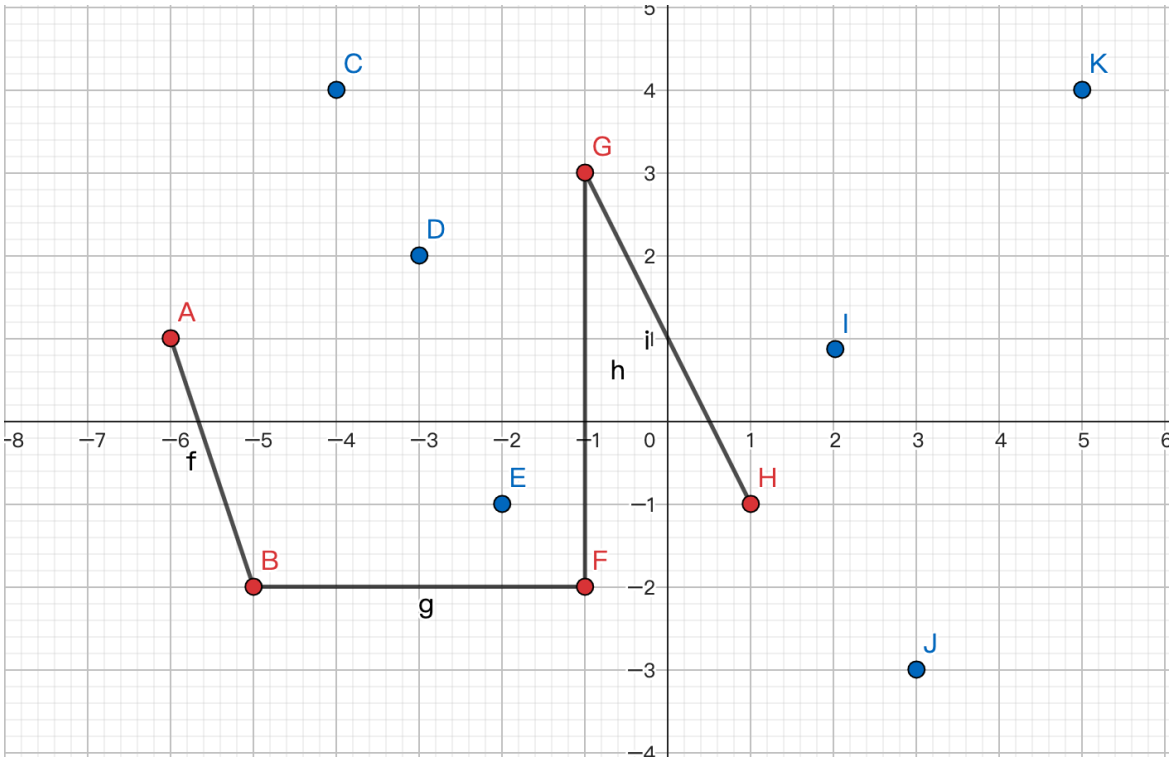
算法



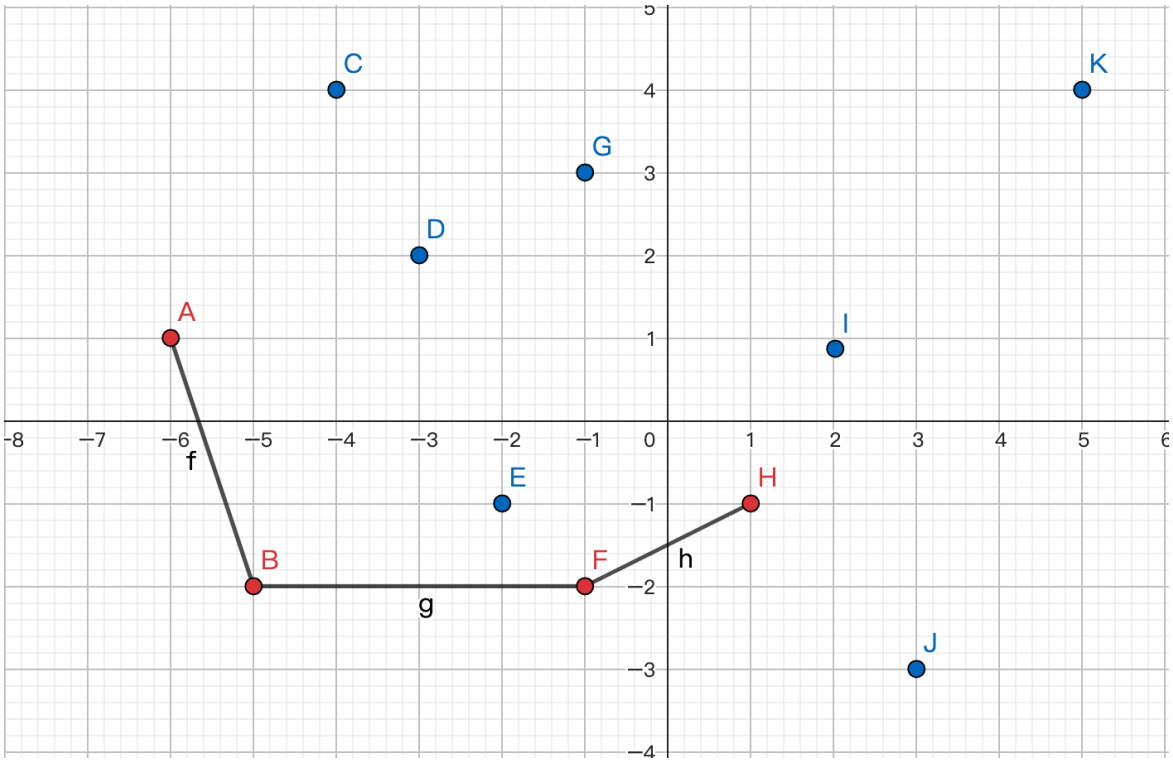
算法



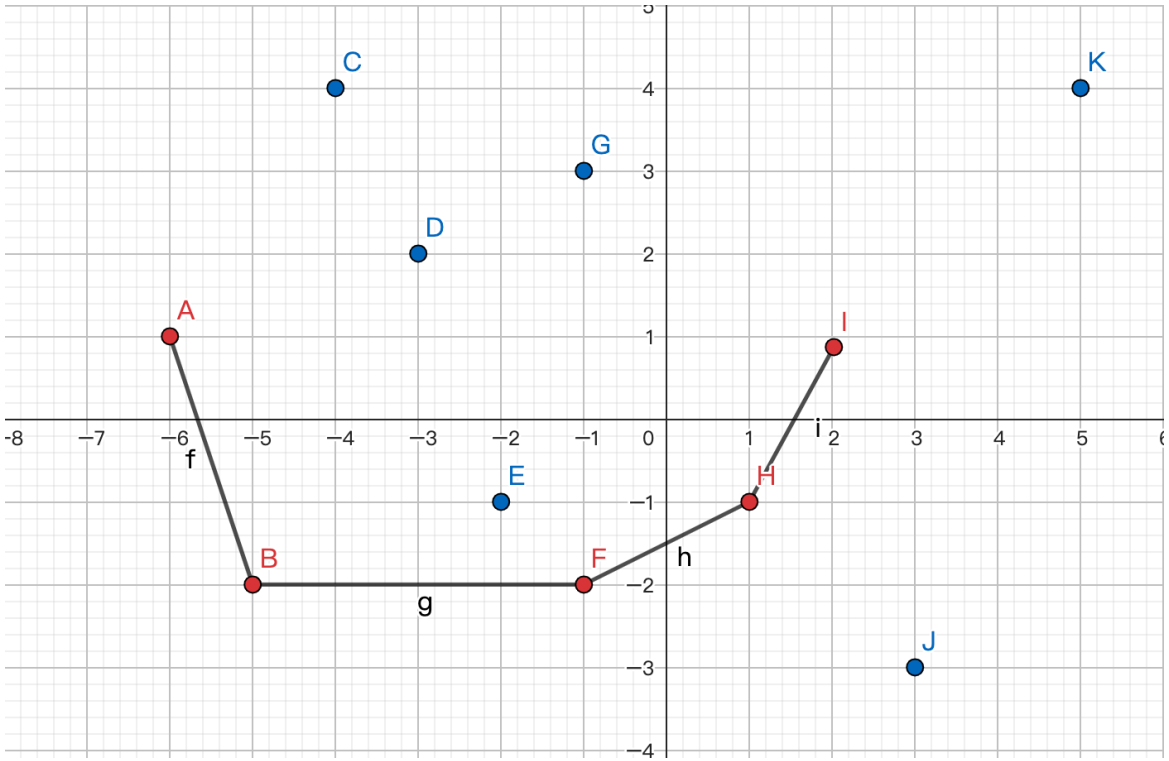
算法



算法



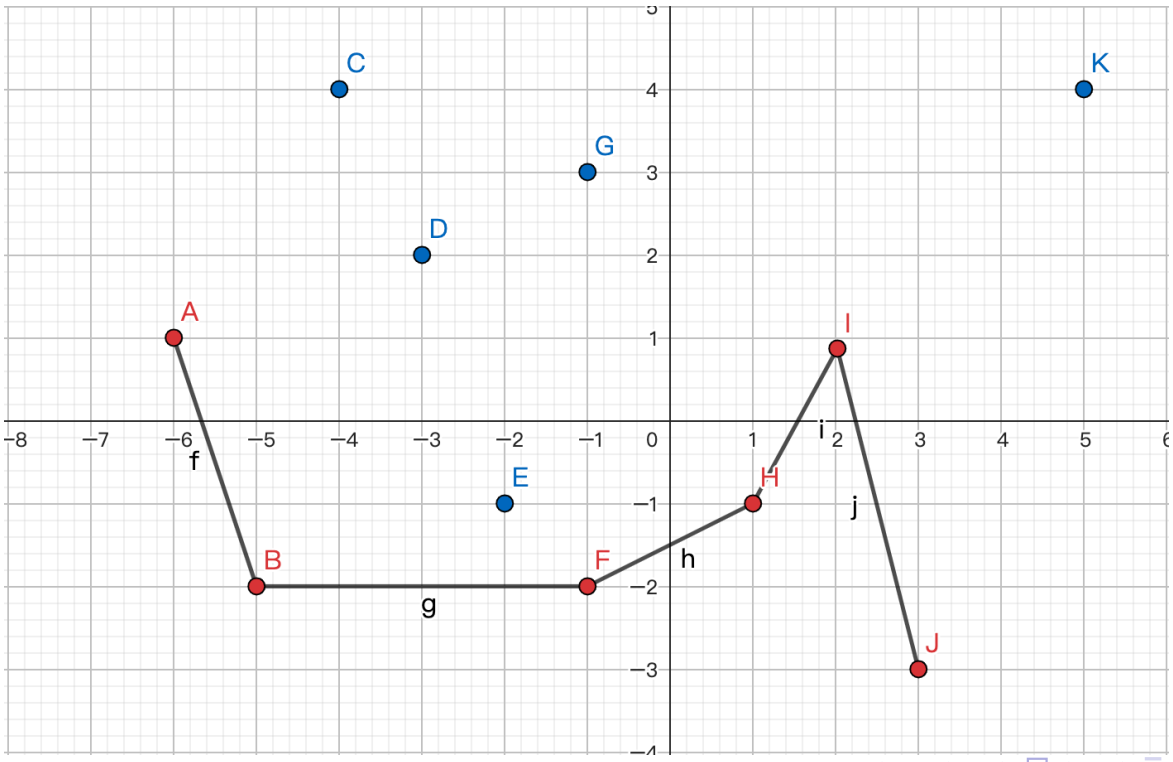
算法



算法

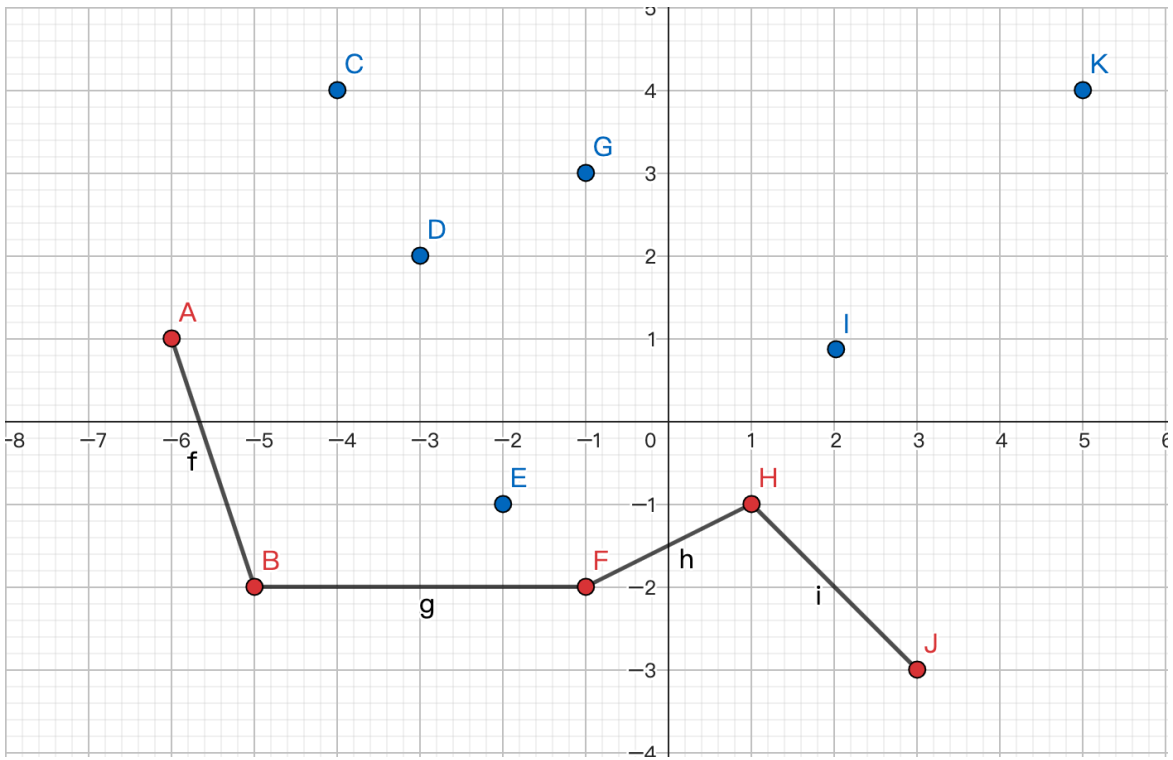
此时栈中的点为 $\{A, B, F, H, I\}$ 。

接下来处理点 J ，连接 IJ ，发现 $\overrightarrow{IJ}$ 在 $\overrightarrow{HI}$ 的顺时针方向。



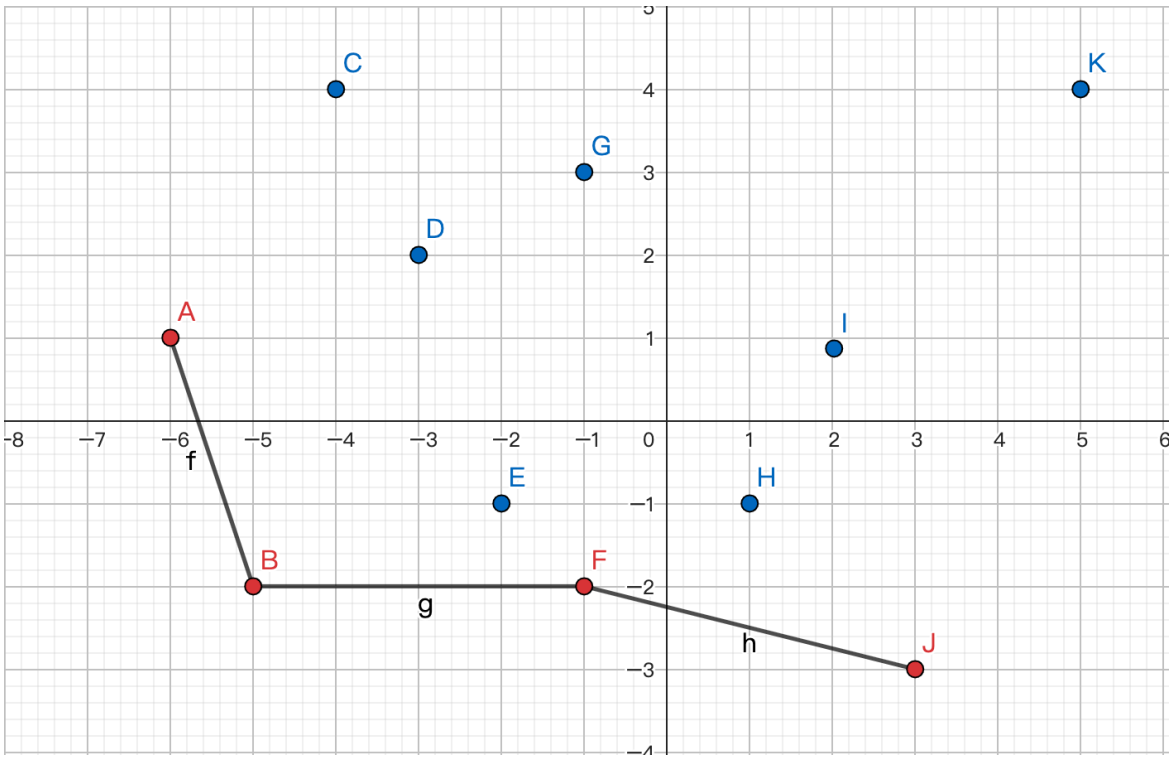
算法

将点 I 退栈，连接 HI ，此时你发现 $\overrightarrow{HJ}$ 依然在 $\overrightarrow{FH}$ 的顺时针方向。



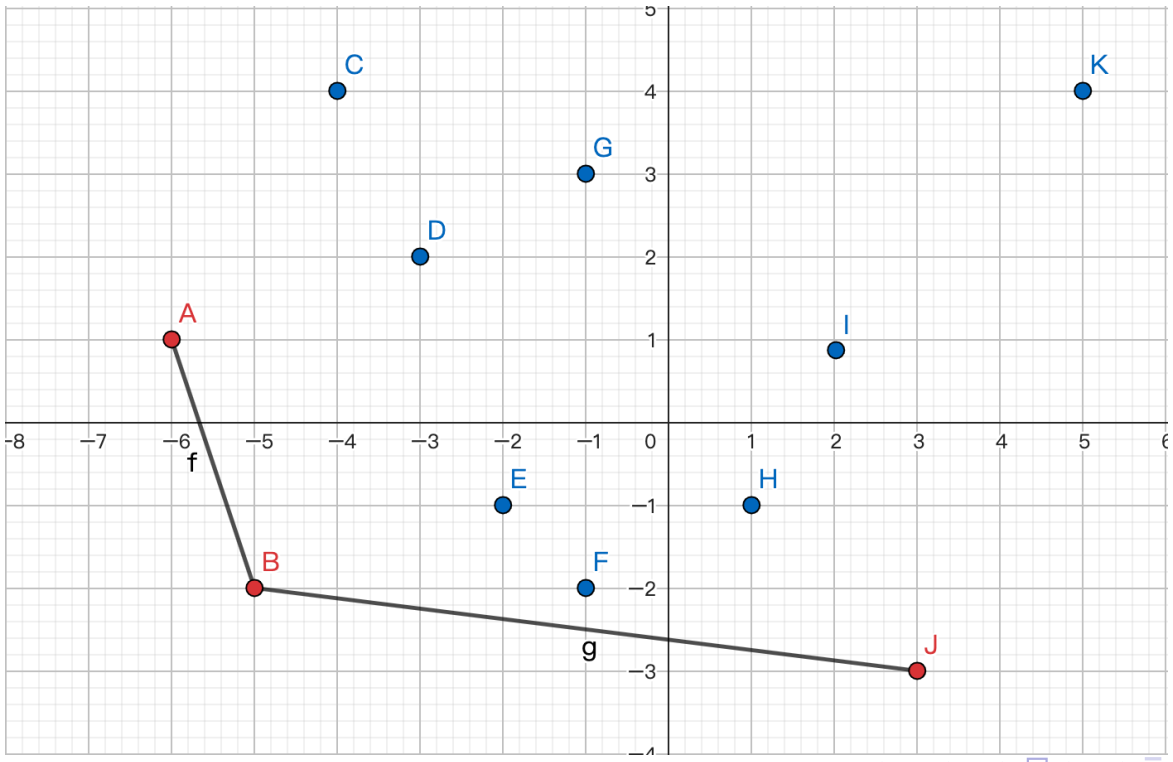
算法

继续将点 H 退栈，连接 FJ ，$\overrightarrow{FJ}$ 依然在 $\overrightarrow{BF}$ 的顺时针方向。



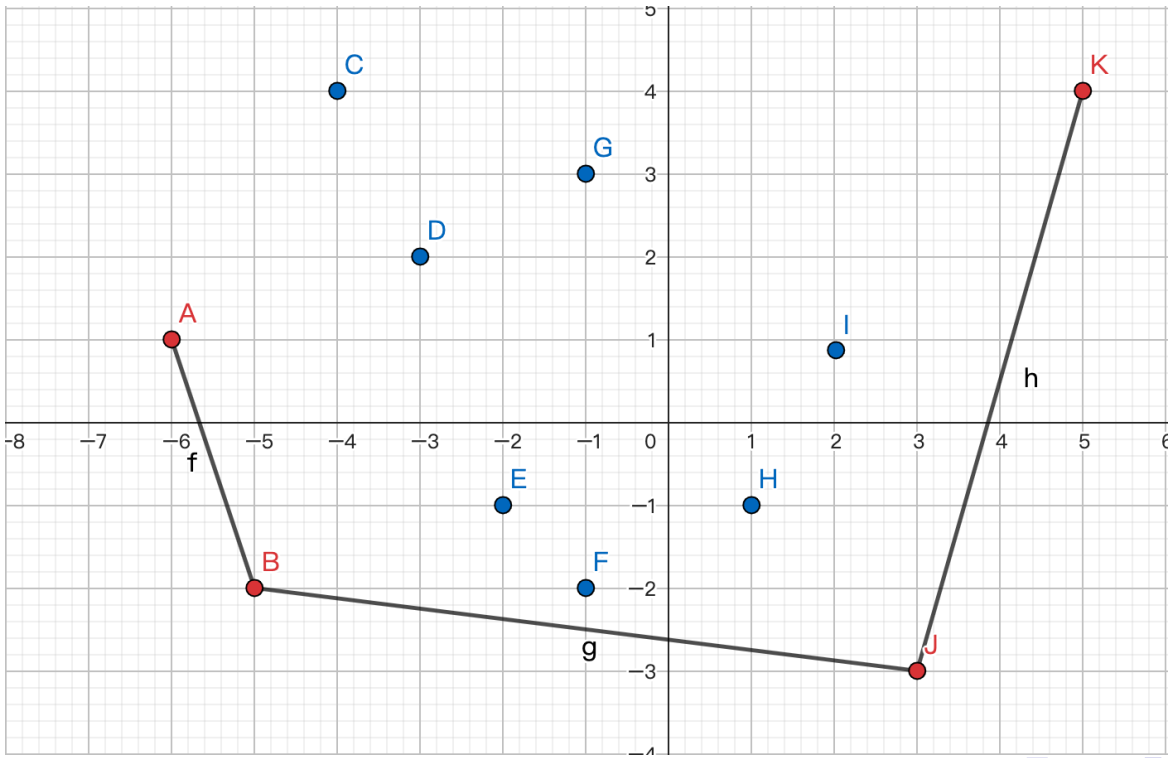
算法

将点 F 退栈，连接 BJ 。此时 $\overrightarrow{BJ}$ 在 $\overrightarrow{AB}$ 的逆时针方向。将点 J 入栈，此时栈中的点为 $\{A, B, J\}$ 。



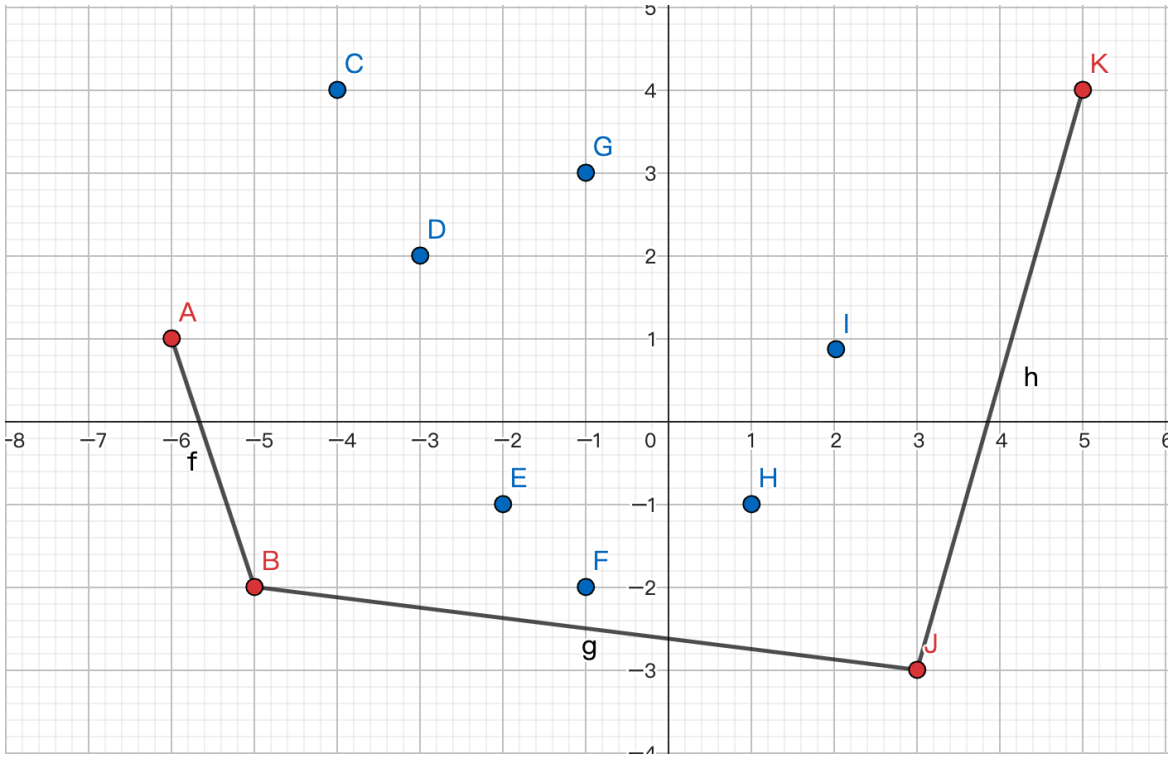
算法

下一个点是 K 。连接 JK ， $\overrightarrow{JK}$ 在 $\overrightarrow{BJ}$ 的逆时针方向，将点 K 入栈。栈中的点为 $\{A, B, J, K\}$ 。



算法

此时，最后一个点 K 已经处理完了，但是你发现，凸包似乎只有下半边，上半边还没有求出来。



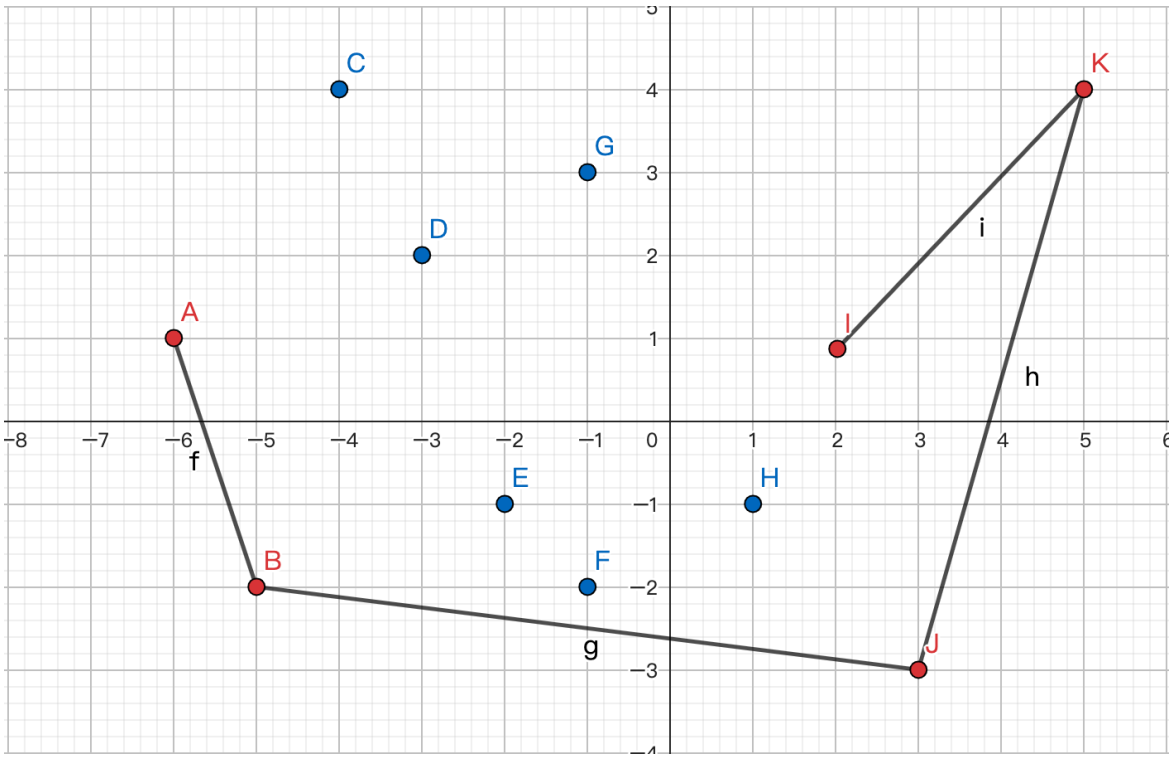
算法

我们将栈弹空，然后将最右侧的点（K）入栈。再**倒序扫描一遍这些点**，按照刚才的流程再做一遍，即可求出凸包的上半部分。

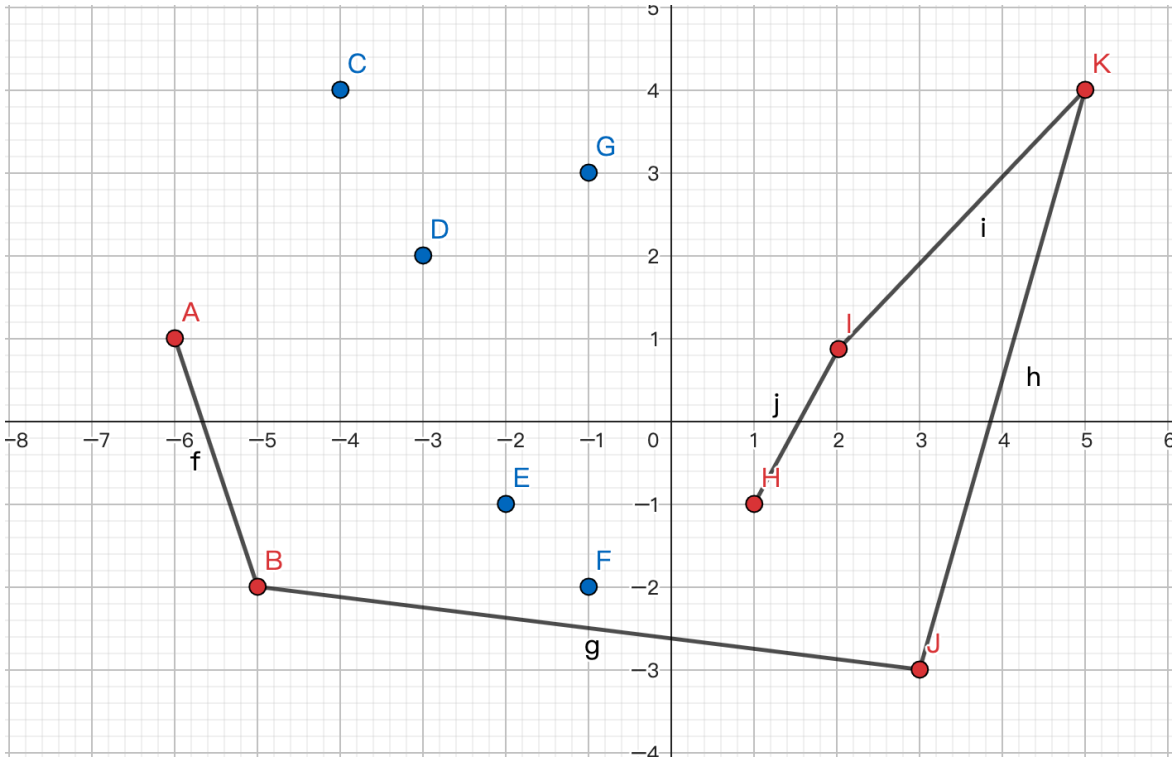
注意，在处理上半部分的过程中，要忽略掉已经在凸包上的点。

（以下为动图演示）

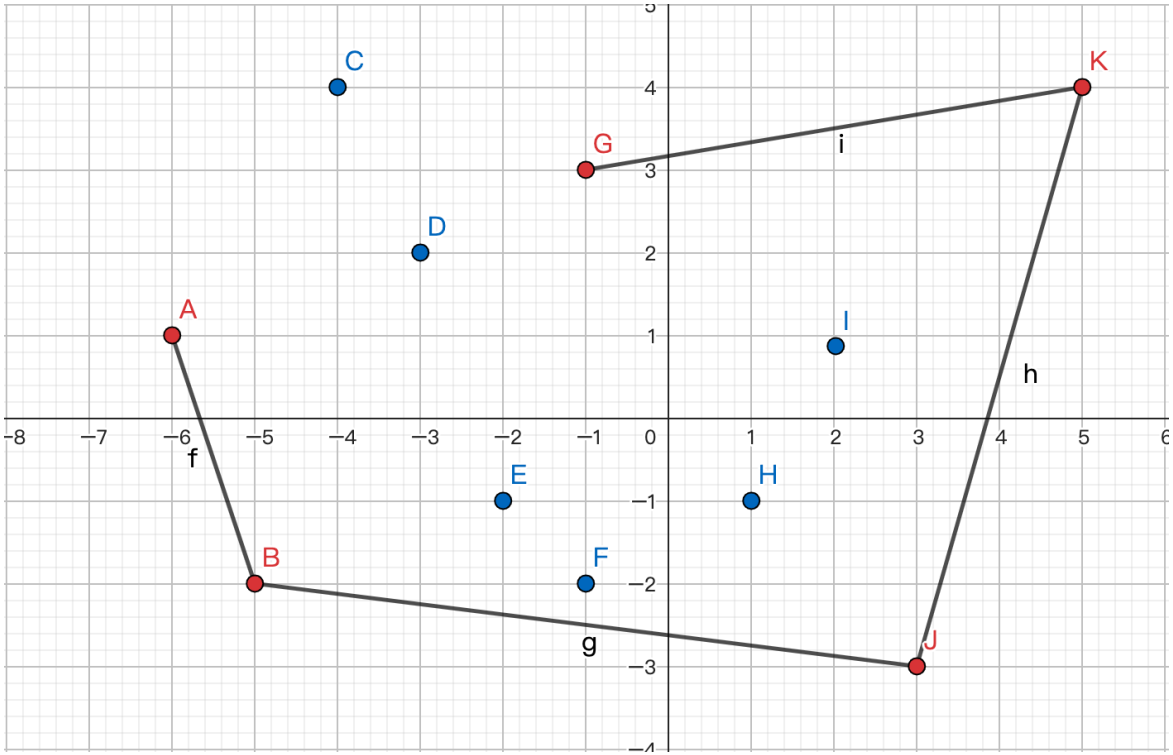
算法



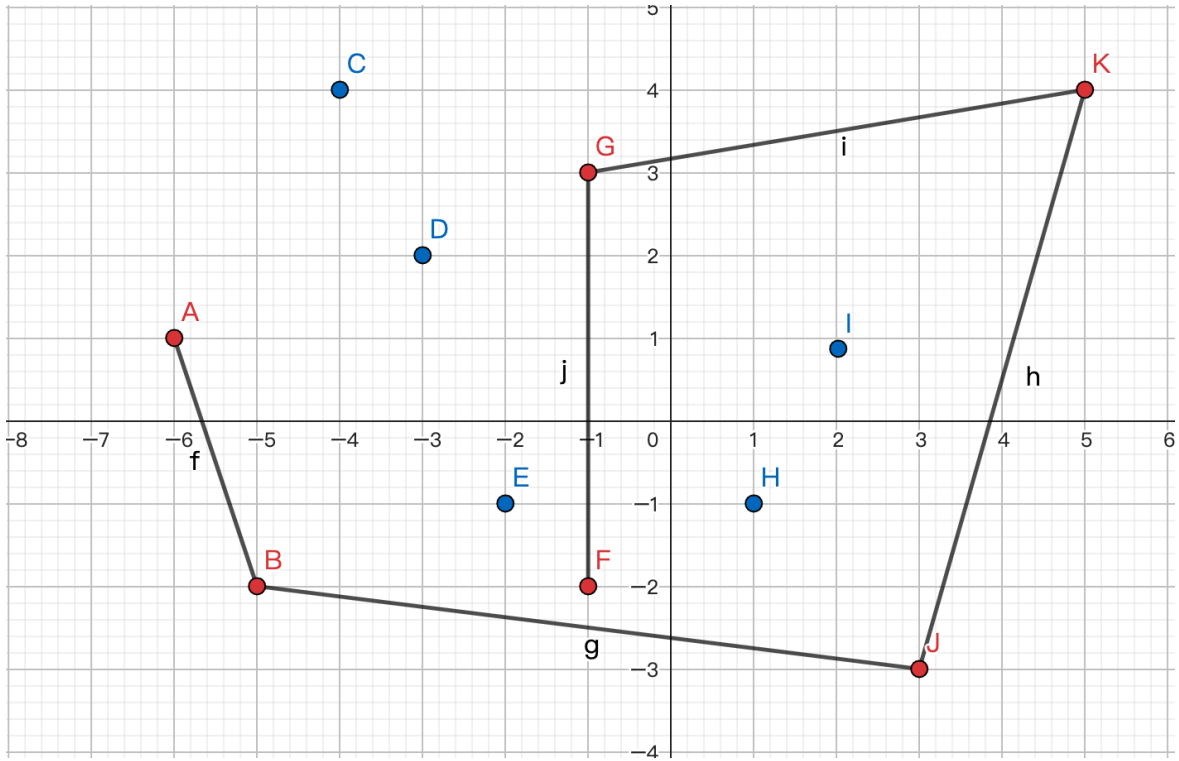
算法



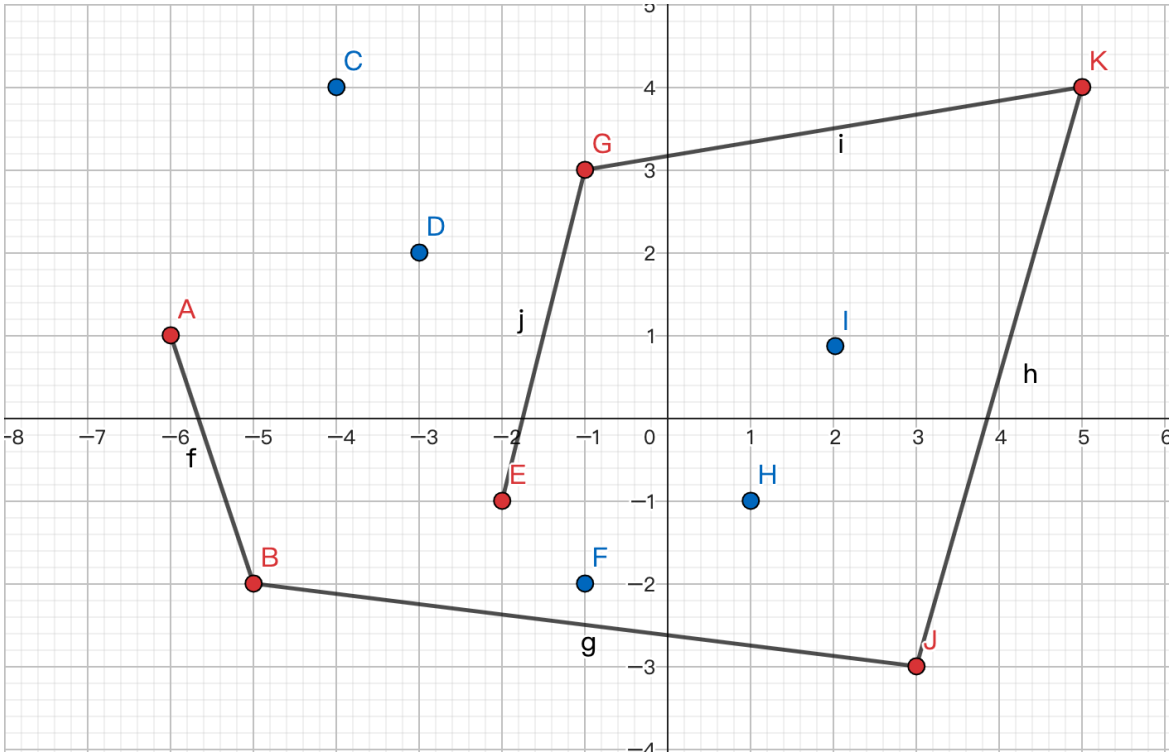
算法



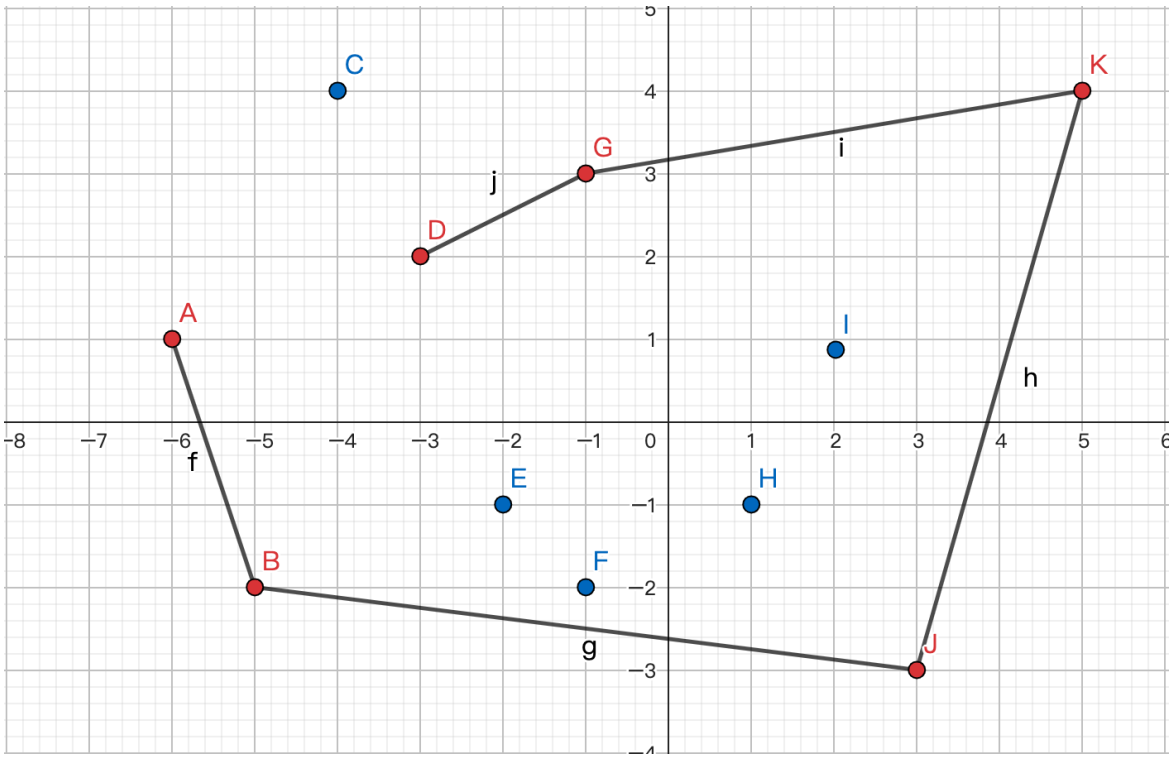
算法



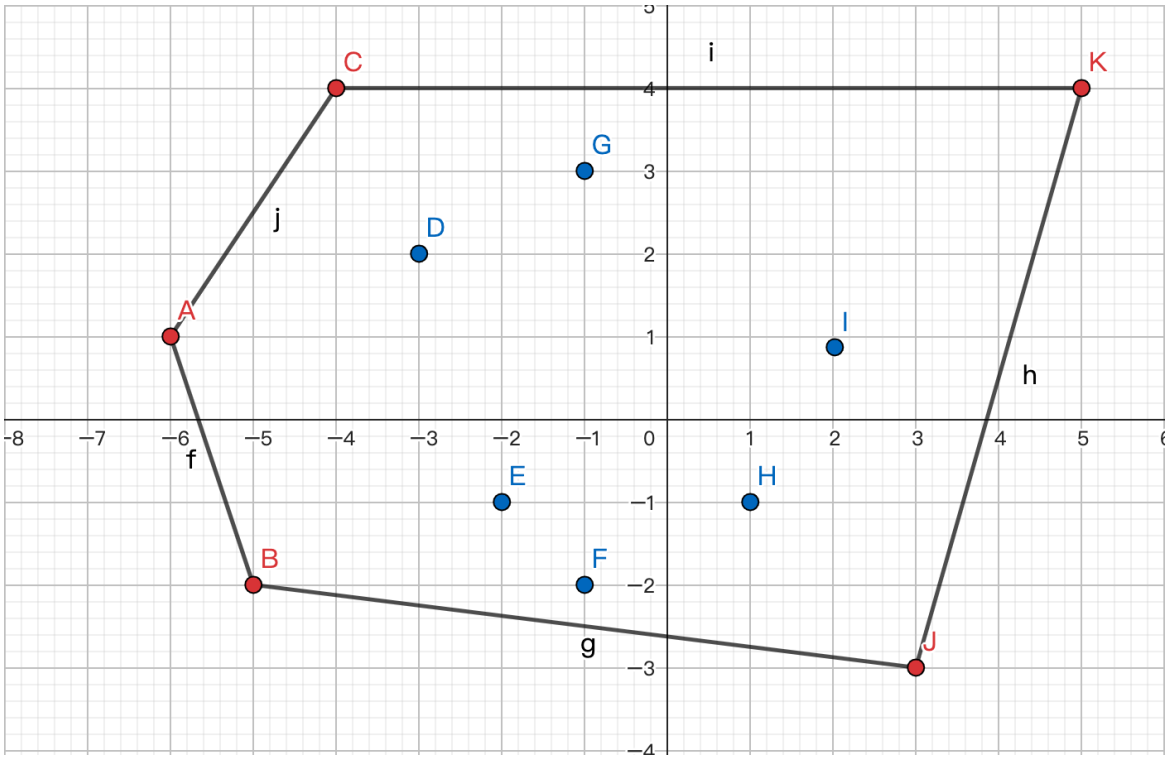
算法



算法



算法



算法的总体流程

下面，回顾一下算法的总体流程。

1. 将所有点按照横坐标为第一关键字，纵坐标为第二关键字排序。
2. 维护一个栈，来记录凸包中的点。首先将最左侧的点压入栈中。
3. 按从左往右的顺序依次扫描各点。设当前处理的点为 M ，栈顶的点为 P 。

    (1) 如果栈中只有一个元素，则将点 M 压入栈中，并标记为凸包上的点。

算法的总体流程

(2) 如果栈中的元素个数大于等于 2，设当前栈中第二个点（即栈顶出栈后新的栈顶）为 Q。

i. $\overrightarrow{PM}$ 在 $\overrightarrow{QM}$ 的**逆时针方向**，直接将点 M 入栈，并标记为凸包上的点。

ii. $\overrightarrow{PM}$ 在 $\overrightarrow{QM}$ 的**顺时针方向**，则将点 P 退栈，并标记为**不在凸包上的点**。重复执行该步骤，直到满足条件 (i) 或栈中元素只剩一个为止。然后，将点 M 入栈，并标记为凸包上的点。

4. 扫描到最后一个点后，再倒序扫描一个点，注意**忽略掉已经在凸包上的点**，执行相同的流程即可。

时间复杂度

对所有点排序的时间复杂度为 $O(n\ log\ n)$，用栈扫描的时间复杂度为 $O(n)$，所以总时间复杂度为 $O(n\ log\ n)$。

例题

给定 n 个点，求出这 n 个点两两距离的最大值。

例题

首先，我们要利用凸包的性质：**平面最远点对一定在凸包上**。
我们首先求这 n 个点的凸包。
那如何求出凸包上距离最远的点对呢？这时需要使用**旋转卡壳**算法。

插一句题外话

如果你仔细观察“旋转卡壳”这四个字，你发现每个字都是多音字！！！

- 旋（xuán/xuàn）
- 转（zhuǎn/zhuàn）
- 卡（kǎ/qiǎ）
- 壳（ké/qiào）

所以，这个算法一共有 16 种读法。（bushi

（以上内容引用自胡老师原话）

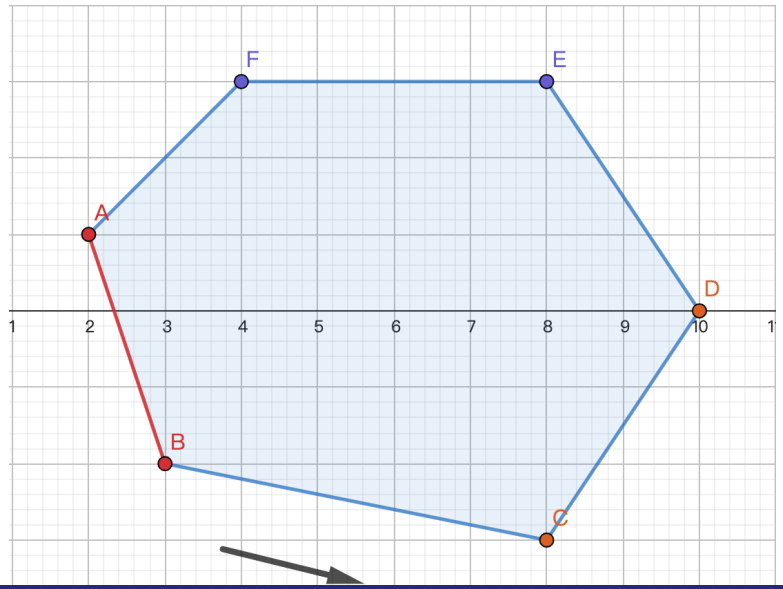
算法流程

求凸包上的最远点对，一个很简单的方法就是首先暴力枚举凸包上的一条边 PQ，然后用 $O(n)$ 的时间找到距离这条边最远的点 M 。这么做的时间复杂度为 $O(n^2)$ ，无法接受。
有没有什么更快的方法，可以求出距离凸包一条边最远的点?

算法流程

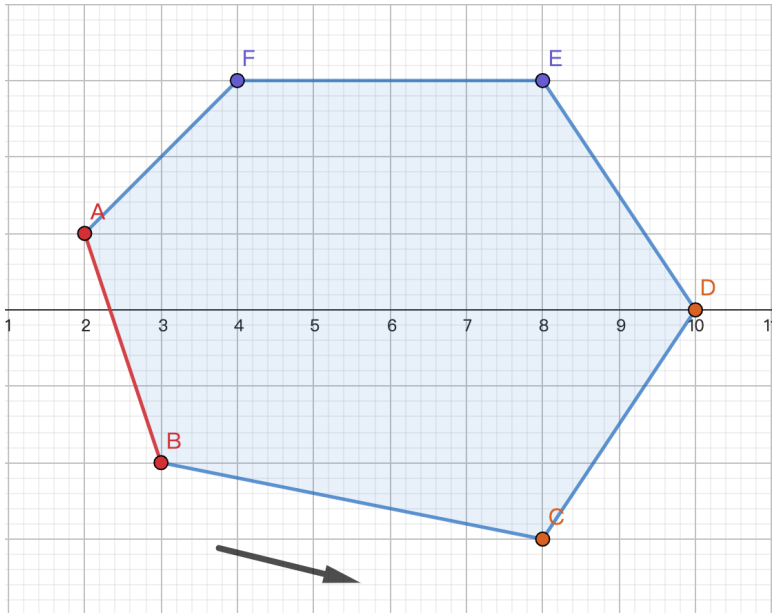
我们研究一下凸多边形上的所有点到其中一条边的距离的变化规律。

如下图所示，我们选定凸包上的一条边 AB ，如果我们从点 B 出发，绕着凸包逆时针走一圈，我们发现到 AB 的距离会先变大，再变小。



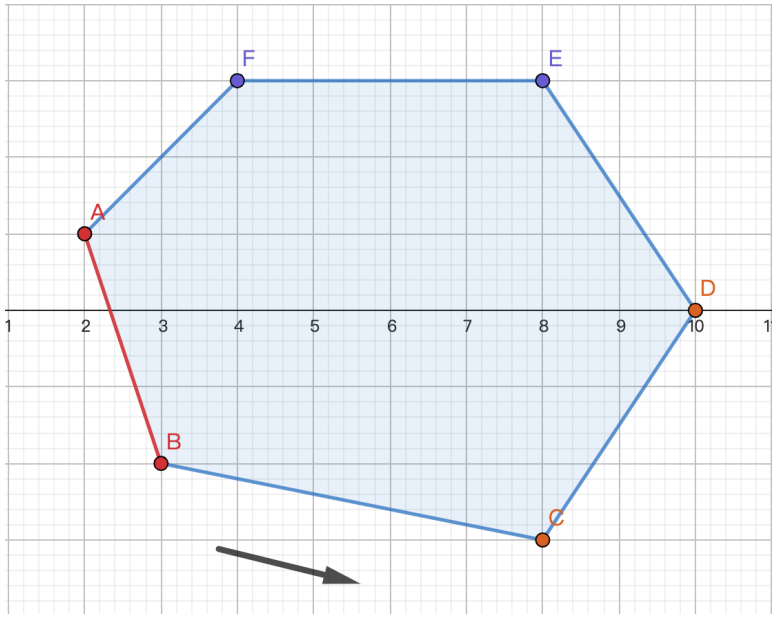
算法流程

具体而言，对于点 C 和点 D，它们到线段 AB 的距离都小于它们在逆时针方向上下一个点到 AB 的距离（距离变大）；而对于点 E 和点 F，它们到线段 AB 的距离都大于它们在逆时针方向上的下一个点到 AB 的距离（距离变小）。



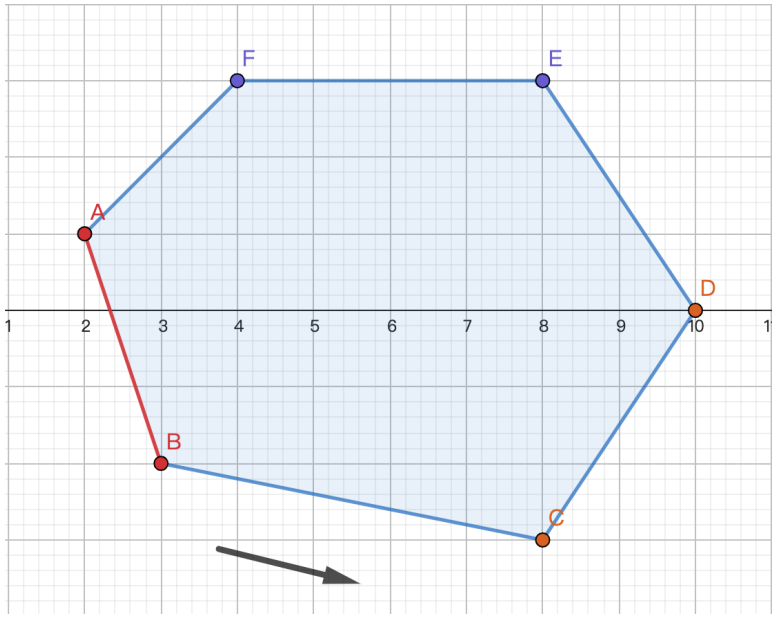
算法流程

不难发现，到线段 AB 距离最远的点是点 E ，而点 E 正是**第一个距离变小的点**。



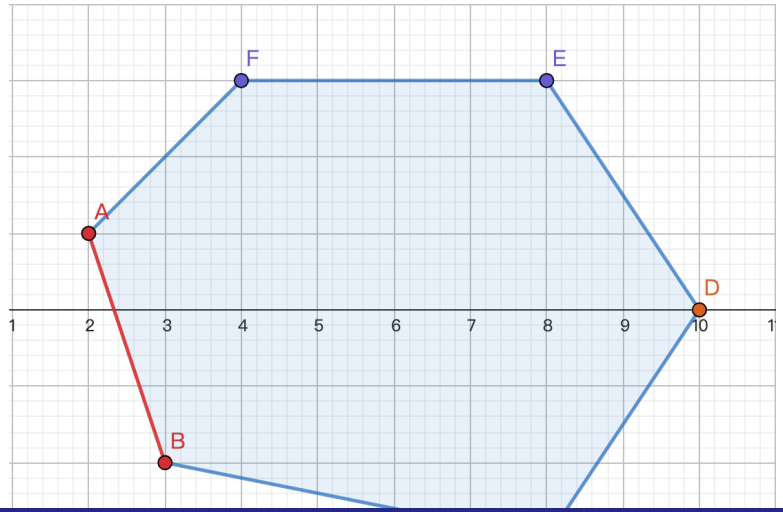
算法流程

不难发现，到线段 AB 距离最远的点是点 E ，而点 E 正是**第一个距离变小的点**。



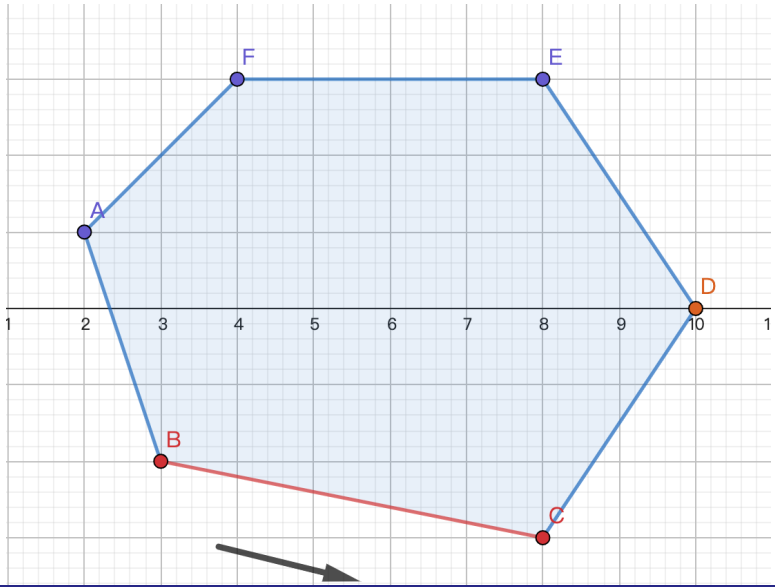
算法流程

我们可以记录一个**指针** P，表示距离当前线段最远的点。一开始将P 指向点 B 。（思考：为什么不能指向其他点？）
然后，我们比较 P 和 P 在逆时针方向上的下一个点 Q 到 AB 的距离。如果点 P 到 AB 的距离大于点 Q 到 AB 的距离，则点 P 就是距离 AB 最远的点，否则，不断令 $P = Q$ （往后找），直到符合上述条件。



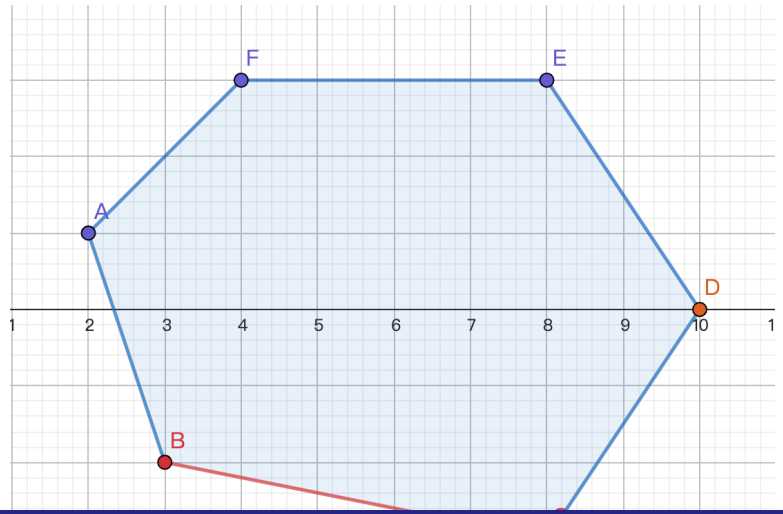
算法流程

就这样，我们找到了距离 AB 最远的点——E 。
有同学可能会问：这样一个一个往后找，复杂度不还是 $O(n)$ 的吗?
但是，这仅仅是单次的时间复杂度。我们接下来要寻找距离 BC 最远的点。



算法流程

我们发现，随着当前边从 AB 逆时针旋转到 BC ，距离该边最远的点也会逆时针旋转，不会出现反向的情况。
也就是说，距离 BC 最远的点一定在距离 AB 最远的点的 **逆时针方向**。
我们直接从 E 开始，仿照刚才的流程，不断跳到逆时针方向的下一个点，直到距离变小了。



总结

算法流程总结:
按逆时针顺序依次扫描每条边。维护一个指针 P ，表示距离当前边最远的点。初始情况下，令 P 指向第一条边的某个端点。然后，不断令点 P 指向逆时针方向的下一个点，直到距离变小了。此时点 P 即为距离当前边最远的点。

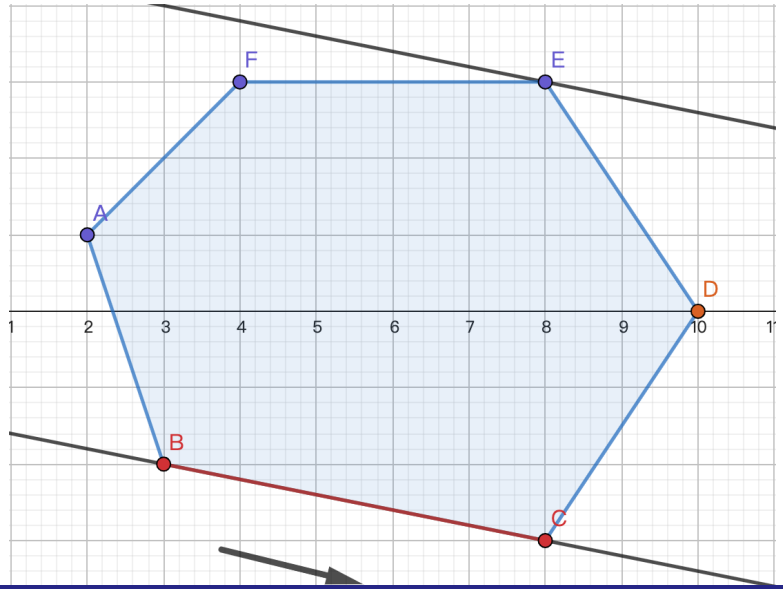
时间复杂度

容易发现，点 P 最多绕着凸多边形转两圈，所以指针 P 的移动是 $O(n)$ 的。故总时间复杂度为 $O(n)$ 。
（如果算上求凸包的复杂度，本题时间复杂度应为 $O(n\ log\ n)$

算法名称由来

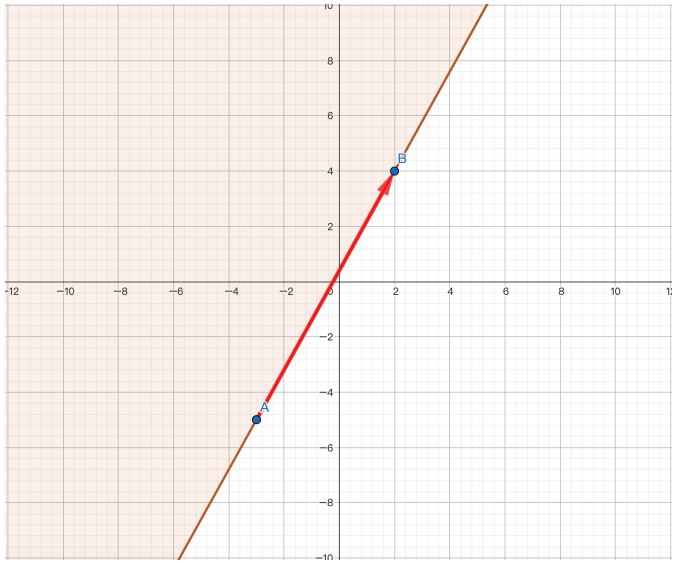
在上述过程中，如果我们作直线 BC ，并过点 E （距离 BC 最远的点）作 BC 的平行线，这个凸包似乎就被这两条线给“卡”住了。随着一条线逆时针旋转，另一条线也会逆时针旋转。

这就是“旋转卡壳”一词的由来。



定义

半平面，故名思义，就是「平面的一半」。
具体而言，一条直线的左侧（右侧）就是一个半平面。
一般情况下，我们使用**有向线段**表示半平面，在题目中统一规定有向线段的左侧为半平面，如下图所示。



定义

注意，**有向线段**不等于**向量**——向量是没有起点的，可以任意平移，而有向线段是有起点的。
可以这样理解：有向线段=起点+向量。

定义

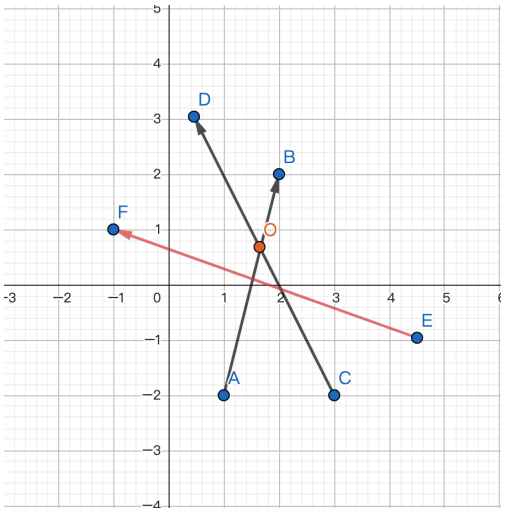
求若干个半平面的交集，就是「半平面交」。

算法

首先，我们将所有有向线段看作向量，按照极角序排序。（即，对于有向线段 AB ，我们按 $\overrightarrow{AB}$ 的极角排序）。特别地，对于平行的线段，我们保留更靠左侧的那一个。（可以使用叉乘判断是否共线，以及一点在有向线段的左侧/右侧）

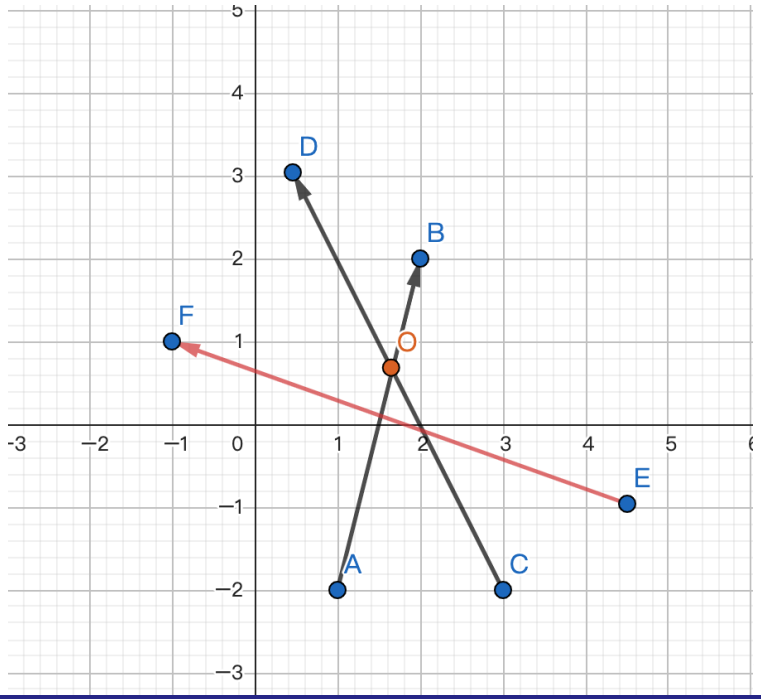
算法

然后，我们维护一个**双端队列**，表示组成半平面交的所有有向线段。按照极角序依次处理每个有向线段。每插入一个有向线段时，我们检查**队尾的两条有向线段的交点是否位于新的有向线段左侧**。如果不满足该条件，则不断删除队尾的有向线段，直至满足上述条件或队列中有向线段个数小于 2 。最后，将新的有向线段插入队尾。



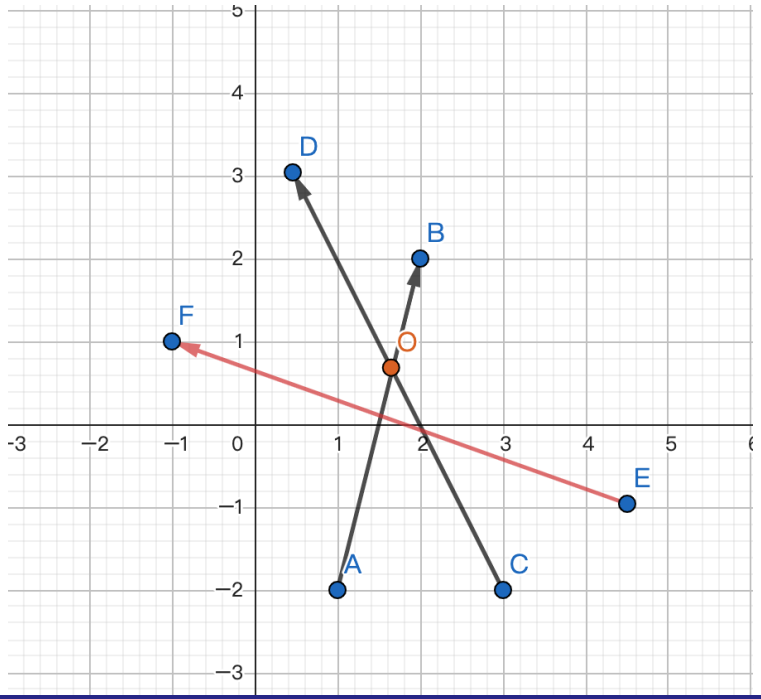
算法

例如下图中，我们先处理有向线段 AB 。由于此时队列中没有别的向量与之形成交点，所以直接将该有向线段插入队尾。此时队列中的有向线段为 $\left\{\overrightarrow{AB}\right\}$ 。



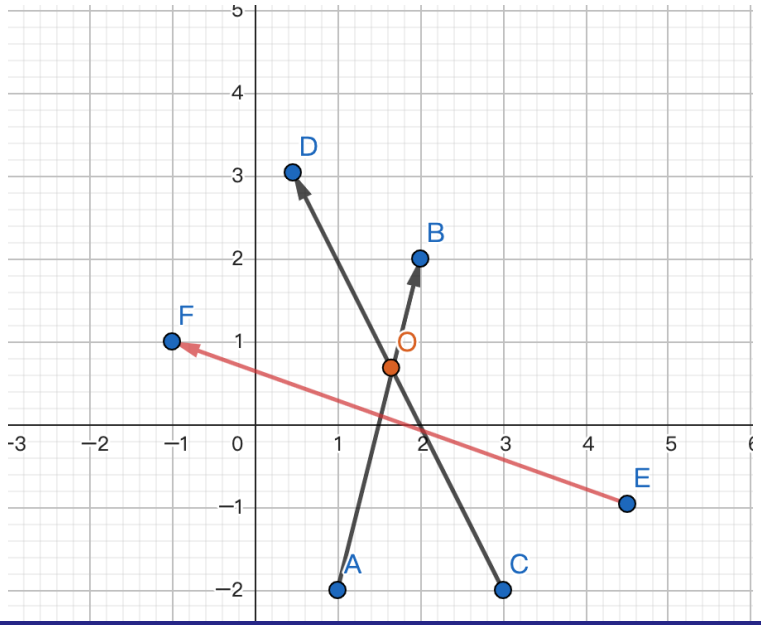
算法

然后处理有向线段 $\overrightarrow{CD}$ 。此时有向线段 $\overrightarrow{CD}$ 会与有向线段 AB 形成一个交点 O 。将 $\overrightarrow{CD}$ 插入队尾，此时队列中的有向线段为 $\left\{\overrightarrow{AB}, \overrightarrow{CD}\right\}$ 。



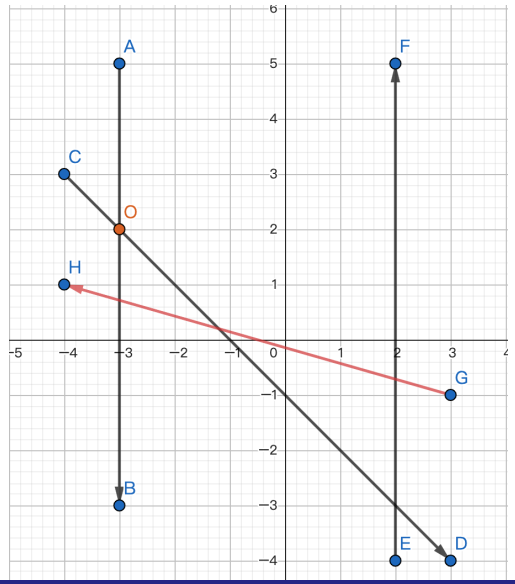
算法

然后处理有向线段 $\overrightarrow{EF}$ 。你会发现，$\overrightarrow{AB}$ 与 $\overrightarrow{CD}$ 的交点 O在 $\overrightarrow{EF}$ 的 **右侧**，也就是说 O 不在 $\overrightarrow{EF}$ 所表示的半平面内。此时，我们直接删除队尾有向线段 $\overrightarrow{CD}$ 。然后将 $\overrightarrow{EF}$ 入队。队列中的有向线段为 $\left\{\overrightarrow{AB}, \overrightarrow{EF}\right\}$ 。



算法

还有一种情况，就是处理到最后的时候，新加入的有向线段可能会影响队首的有向线段。如下图所示，依次插入有向线段 $\overrightarrow{AB}, \overrightarrow{CD}, \overrightarrow{EF}$ 后，有向线段 $\overrightarrow{AB}$ 与 $\overrightarrow{CD}$ 产生了一个交点 O 。再插入有向线段 $\overrightarrow{GH}$ ，由于 O 在 $\overrightarrow{GH}$ 右侧，所以此时需要**删除队首的有向线段** $\overrightarrow{AB}$ 。

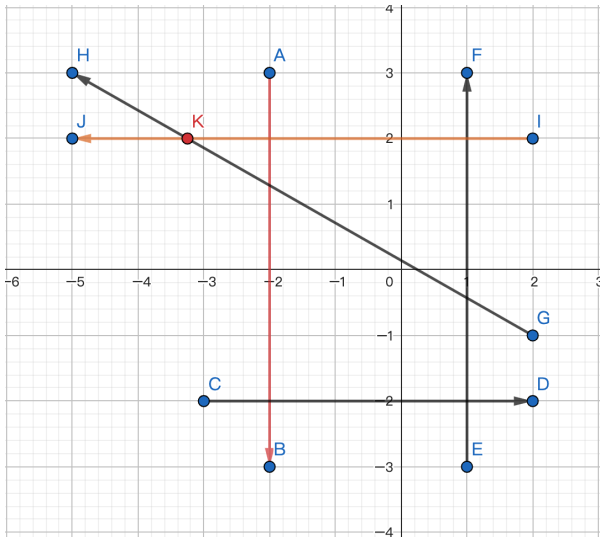


算法

就这样，我们依次处理每个有向线段，然后按照上述方法，不停地从队列中删除多余的有向线段，就能求得半平面交……吗?

算法

其实，我们忽略了一种非常重要的情况：整个算法结束后，**队首的有向线段**可能会**排除队尾的有向线段**。
如下图所示，依次插入有向线段 $\overrightarrow{AB}, \overrightarrow{CD}, \overrightarrow{EF}, \overrightarrow{GH}, \overrightarrow{IJ}$ 后，有向线段 $\overrightarrow{GH}$ 与有向线段 $\overrightarrow{IJ}$ 的交点 K 在队首有向线段 $\overrightarrow{AB}$ 的右侧，因此需要删除有向线段 $\overrightarrow{IJ}$。



算法

下面，我们来回顾一下算法的整体流程。

1. 将所有有向线段按照极角序排序。对于极角相等的有向线段（平行），保留左侧的那一个。
2. 按极角序从小到大依次扫描各有向线段。维护一个双端队列，存储有向线段。
3. 不断删除队尾的有向线段，直到队列的大小小于 2，或队尾的两条有向线段的交点在当前有向线段的左侧。
4. 不断删除队首的有向线段，直到队列的大小小于 2，或队首的两条有向线段的交点在当前有向线段的左侧。

（**注意，操作3和操作4的顺序不能调换，具体原因可参考 OI Wiki。**）

算法

5. 将当前有向线段插入队尾。
6. 扫描完所有有向线段后，不断弹出队尾的有向线段，直到队列的大小小于 2 ，或队尾的两条有向线段的交点在队首的有向线段的左侧。

友情提示：求线段交点的方法，上文已经讲述过，这里不再赘述。

最后一步

执行完上述算法后，双端队列里会留下一堆有向线段。把队列中相邻的两条有向线段（含队尾和队首）的交点分别求出来，依次连接，即可得到半平面交的结果——一个多边形。
可以证明，半平面交的结果只可能是：

1. 无交集。
2. 一条直线。
3. 凸多边形。
4. 一个无限大的平面。

加边框

有些题目为了避免半平面交的结果是一个无限大的平面，会用一个矩形将所有有向线段“框”住。这时，只需要将矩形的边框也当作半平面（有向线段），正常求解即可。

特殊情况

有时半平面交是不存在的。这种情况下，算法结束后，双端队列里只会剩下一个有向线段。直接判断就可以了。

例题

给定一个 n 边形，求一点集 S ，使得 S 中任意一点 P 都满足如下条件：
对于多边形上任意一点 Q ，线段 PQ 与多边形不会有其他交点（除点 Q 外）。
为了简化问题，你只需要输出点 S 内的所有点组成的图形的面积。

例题

这个问题通常被称为**多边形的核**。
把多边形的每条边看作有向线段，求半平面交，即可得到答案。
正确性显然。

例题

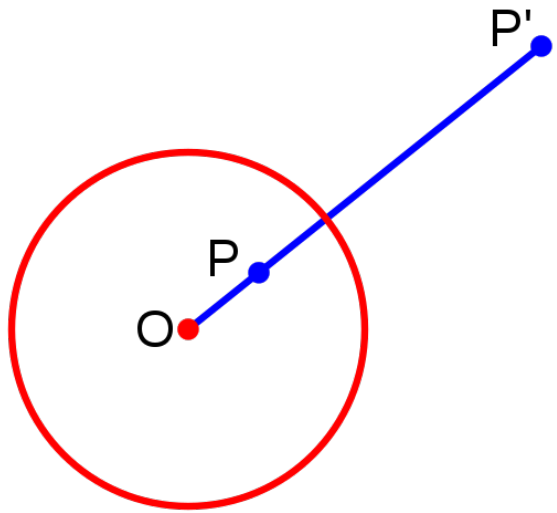
给定 n 个凸多边形，求它们的面积并。
$1 \leq n \leq 15, 3 \leq$ 每个多边形的边数 ≤ 11 。

例题

首先考虑扫描线，发现用扫描线是不好述的。
发现 n 比较小，考虑容斥原理，将面积相加，再减去面积交。
接下来的问题就是如何求凸多边形的面积交。只需要把所有凸多边形的边看作有向线段，求半平面交，得到的就是这些凸多边形的交集。

定义

反演变换与平移、旋转、轴对称类似，都是图形的变换。
给定反演中心 O 和反演半径 R，对于一点 P，其反演点 P' 位于射线 OP 上，且 $OP \cdot OP' = R$。
注意，此处 $OP \cdot OP'$ 指的是线段 OP 与线段 OP' 的长度之积。



定义

一个图形的**反演图形**为该图形上所有点反演后组成的图形。

性质

反演变换的性质：
1. 对于两个图形（圆/直线），如果它们反演之前相交，则它们反演之后也相交；相切、相离同理。
2. 对于一个**不经过反演中心**的圆，其反演后依然是一个圆。
3. （**很重要**）对于一个**经过反演中心**的圆，其反演后是一条不经过反演中心的直线。
4. 如果图形 A 的反演图形是图形 B，则图形 B 的反演图形是图形 A。

例题

给定两个圆，求**经过原点**，且与两圆相切的所有圆。
注意是**经过原点**，不是以原点为圆心。

解法

考虑以原点为反演中心，任意长度为反演半径，进行反演变换。
显然，反演后，所求的圆（过原点的圆）会被反演成一条直线。
因为所求的圆要与两圆相切，所以反演后得到的直线也应同时与两个圆相切。
此时，问题就被转化成了：给定两个圆，求它们的所有公切线。
上文讲述过内公切线的求法（平移旋转），外公切线的求法是类似的，这里不再赘述。
求出两圆的公切线后，反演回去就可以了。

解法

如何求一个图形反演后的图形?
这一步应该是有公式的，~~但我似乎没搜到，原论文也没有讲~~，于是我想到了一种非常笨的方法:
根据“三点确定一个圆”的原理，我们只需要在原来的图形（圆/直线）上找到三个点，求出它们的反演点，然后求这三个点的外接圆，即可得到该图形（圆/直线）的反演图形。
求外接圆可以直接解方程。

例题

给定 n 个圆，求一经过原点的圆，使其与最多的圆相交。

解法

还是按照刚才的方法，以原点为反演中心进行反演，问题转化为：求作一直线，使得该直线与最多的圆相交。
容易发现，最优情况下，所求的直线一定会与某一个圆相切，否则可以通过平移的方式，让它与一个圆相切，此时答案不会变劣。
我们可以枚举相切的那个圆。容易发现，对于其它的圆，该直线的角度在一定范围内时，才会与这个圆产生交点。
我们可以把每个圆产生贡献的角度区间求出来，问题被转化为：给定一堆区间，求哪个点被最多的区间覆盖。
直接使用扫描线等方法即可。

总结

本文介绍了计算几何题的许多常用算法。实际上，这些算法已经足以解决大部分的计算几何问题。然而，由于时间原因，还有一些计算几何算法本文没有提及，如三角剖分、平面网格化、随机增量法等。

此外，本文介绍的计算几何算法仅限于二维计算几何，其实还有一些三维计算几何的算法，本文没有介绍，如三维凸包等。

有兴趣的同学们可以自行访问 OI Wiki 等网站学习。

谢谢大家

谢谢大家！

XC万岁！！！

伟大的XC精神万岁！！！