中国计算机学会
China Computer Federation

计算几何初步

长沙市雅礼中学 屈运华

前置技能

- 几何基础
- 平面直角坐标系
- 向量（包括向量积）
- 极坐标与极坐标系

图形的记录

- 点

  记录其横纵坐标值即可。用 pair 或开结构体记录均可。

- 向量

  由于向量的坐标表示与点相同，所以只需要像点一样存向量即可（当然点不是向量）。

- 线

  ## 直线与射线

  直线上一点和直线的方向向量。

  ## 线段

  只需要记录左右端点即可。

- 多边形

  开数组按一定顺序记录多边形的每个顶点即可。

- 曲线

  一些特殊曲线，如函数图像等一般记录其解析式。对于圆，直接记录其圆心和半径即可。

基本公式

- 正弦定理

在三角形 $\triangle ABC$ 中，若角 A, B, C 所对边分别为 a, b, c，则有：

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R$$

其中，R 为 $\triangle ABC$ 的外接圆半径。

- 余弦定理

在三角形 $\triangle ABC$ 中，若角 A, B, C 所对边分别为 a, b, c，则有：

$$a^2 = b^2 + c^2 - 2bc\cos A$$
$$b^2 = a^2 + c^2 - 2ac\cos B$$
$$c^2 = a^2 + b^2 - 2ab\cos C$$

目录

向量

向量：既有大小又有方向的量称为向量。数学上研究的向量为 **自由向量**，即只要不改变它的大小和方向，起点和终点可以任意平行移动的向量。记作 $\vec{a}$ 或 $\boldsymbol{a}$。

有向线段：带有方向的线段称为有向线段。有向线段有三要素：**起点，方向，长度**，知道了三要素，终点就唯一确定。我们用有向线段表示向量。

向量的模：有向线段 $\overrightarrow{AB}$ 的长度称为向量的模，即为这个向量的大小。记为：$|\overrightarrow{AB}|$ 或 $|\boldsymbol{a}|$。

零向量：模为 0 的向量。零向量的方向任意。记为：$\vec{0}$ 或 $\mathbf{0}$。

单位向量：模为 1 的向量称为该方向上的单位向量。

平行向量：方向相同或相反的两个 **非零** 向量。记作：$\boldsymbol{a} \parallel \boldsymbol{b}$。对于多个互相平行的向量，可以任作一条直线与这些向量平行，那么任一组平行向量都可以平移到同一直线上，所以平行向量又叫 **共线向量**。

相等向量：模相等且方向相同的向量。

相反向量：模相等且方向相反的向量。

向量的夹角：已知两个非零向量 $\boldsymbol{a}, \boldsymbol{b}$，作 $\overrightarrow{OA} = \boldsymbol{a}, \overrightarrow{OB} = \boldsymbol{b}$，那么 $\theta = \angle AOB$ 就是向量 $\boldsymbol{a}$ 与向量 $\boldsymbol{b}$ 的夹角。记作：$\langle \boldsymbol{a}, \boldsymbol{b} \rangle$。显然当 $\theta = 0$ 时两向量同向，$\theta = \pi$ 时两向量反向，$\theta = \frac{\pi}{2}$ 时我们说两向量垂直，记作 $\boldsymbol{a} \perp \boldsymbol{b}$。并且，我们规定 $\theta \in [0, \pi]$。

注意到平面向量具有方向性，我们并不能比较两个向量的大小（但可以比较两向量的模长）。但是两个向量可以相等。

向量

- **向量加减法：**
  - 设 二维向量P = (x1, y1)， Q = (x2 , y2)。
    - 向量加法定义为： P + Q = (x1 + x2 , y1 + y2)；
    - 向量减法定义为： P - Q = (x1 - x2 , y1 - y2)。
    - 显然有性质 P + Q = Q + P， P - Q = - (Q - P)。

向量

我们定义了一种量，就希望让它具有运算。向量的运算可以类比数的运算，但是我们从物理学的角度出发研究向量的运算。

类比物理学中的位移概念，假如一个人从 A 经 B 走到 C，我们说他经过的位移为 $\overrightarrow{AB}+\overrightarrow{BC}$，这其实等价于这个人直接从 A 走到 C，即 $\overrightarrow{AB}+\overrightarrow{BC}=\overrightarrow{AC}$。

注意到力的合成法则——平行四边形法则，同样也可以看做一些向量相加。

所以我们整理一下向量的加法法则：

1. **向量加法的三角形法则**：若要求和的向量首尾顺次相连，那么这些向量的和为第一个向量的起点指向最后一个向量的终点；
2. **向量加法的平行四边形法则**：若要求和的两个向量 **共起点**，那么它们的和向量为以这两个向量为邻边的平行四边形的对角线，起点为两个向量共有的起点，方向沿平行四边形对角线方向。

这样，向量的加法就具有了几何意义。并且可以验证，向量的加法满足 **交换律与结合律**。

因为实数的减法可以写成加上相反数的形式，我们考虑在向量做减法时也这么写。即：$\boldsymbol{a}-\boldsymbol{b}=\boldsymbol{a}+(-\boldsymbol{b})$。

这样，我们考虑共起点的向量，按照平行四边形法则做出它们的差，经过平移后可以发现 **「共起点向量的差向量」是由「减向量」指向「被减向量」的有向线段。**

这也是向量减法的几何意义。

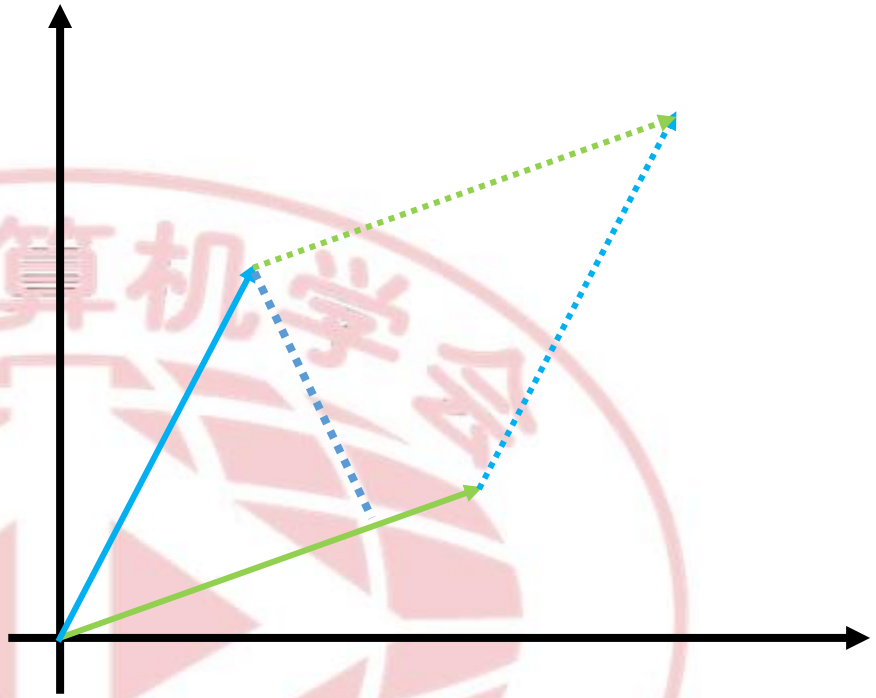
我们有时候有两点 A,B，想知道 $\overrightarrow{AB}$，可以利用减法运算 $\overrightarrow{AB}=\overrightarrow{OB}-\overrightarrow{OA}$ 获得。

向量和点

向量点积

$$\overrightarrow{AB} \times \overrightarrow{CD} = x_1 x_2 + y_1 y_2$$

$$\overrightarrow{AB} \times \overrightarrow{CD} = |AB||CD| \cos\theta$$

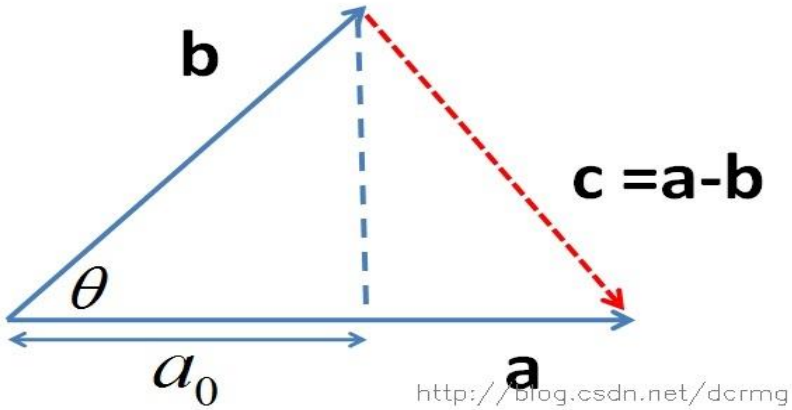


向量叉积

$$\overrightarrow{AB} \times \overrightarrow{CD} = x_1 y_2 - x_2 y_1$$

$$\overrightarrow{AB} \times \overrightarrow{CD} = |AB||CD| \sin\theta$$

点积



$$c = a - b$$

$$c^2 = a^2 + b^2 - 2|a||b|\cos\theta$$

$$(a-b)\bullet(a-b) = a^2 + b^2 - 2a\bullet b = a^2 + b^2 - 2|a||b|\cos\theta$$

$$a\bullet b = |a||b|\cos\theta$$

$$\theta = acc\cos(\frac{a\bullet b}{|a||b|})$$

a·b>0　方向基本相同，夹角在0°到90°之间

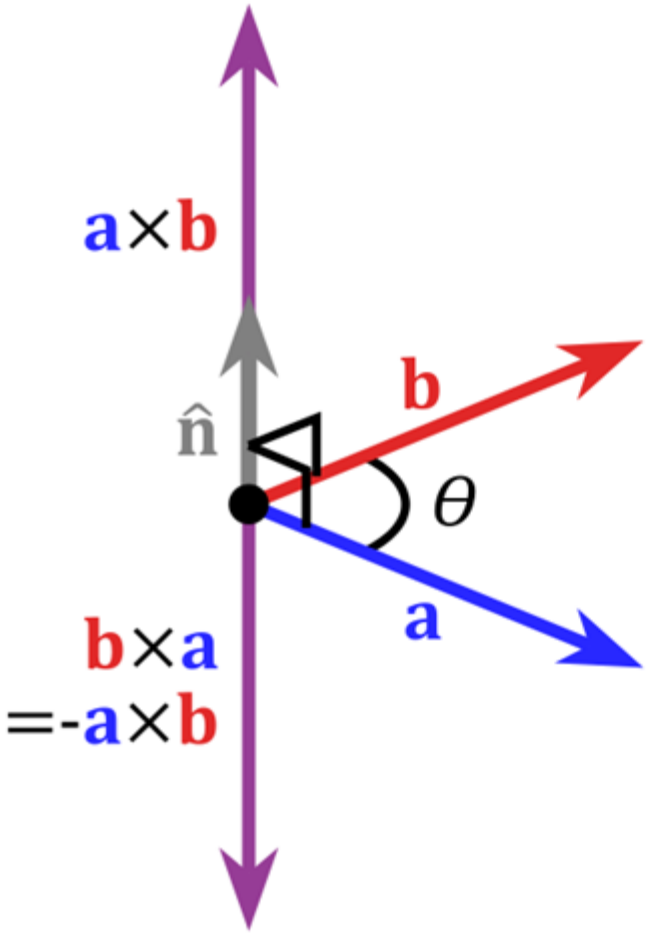
a·b=0　正交，相互垂直

a·b<0　方向基本相反，夹角在90°到180°之间

叉积

定义：向量a与b的外积a×b是一个向量，其长度等于|a×b| = |a||b|sin∠(a,b)，其方向正交于a与b。并且，(a,b,a×b)构成右手系。
特别地，0×a = a×0 = 0.此外，对任意向量a，a×a=0。

在二维空间中，外积还有另外一个几何意义就是：|a×b|在数值上等于由向量a和向量b构成的平行四边形的面积。



向量旋转

逆时针旋转

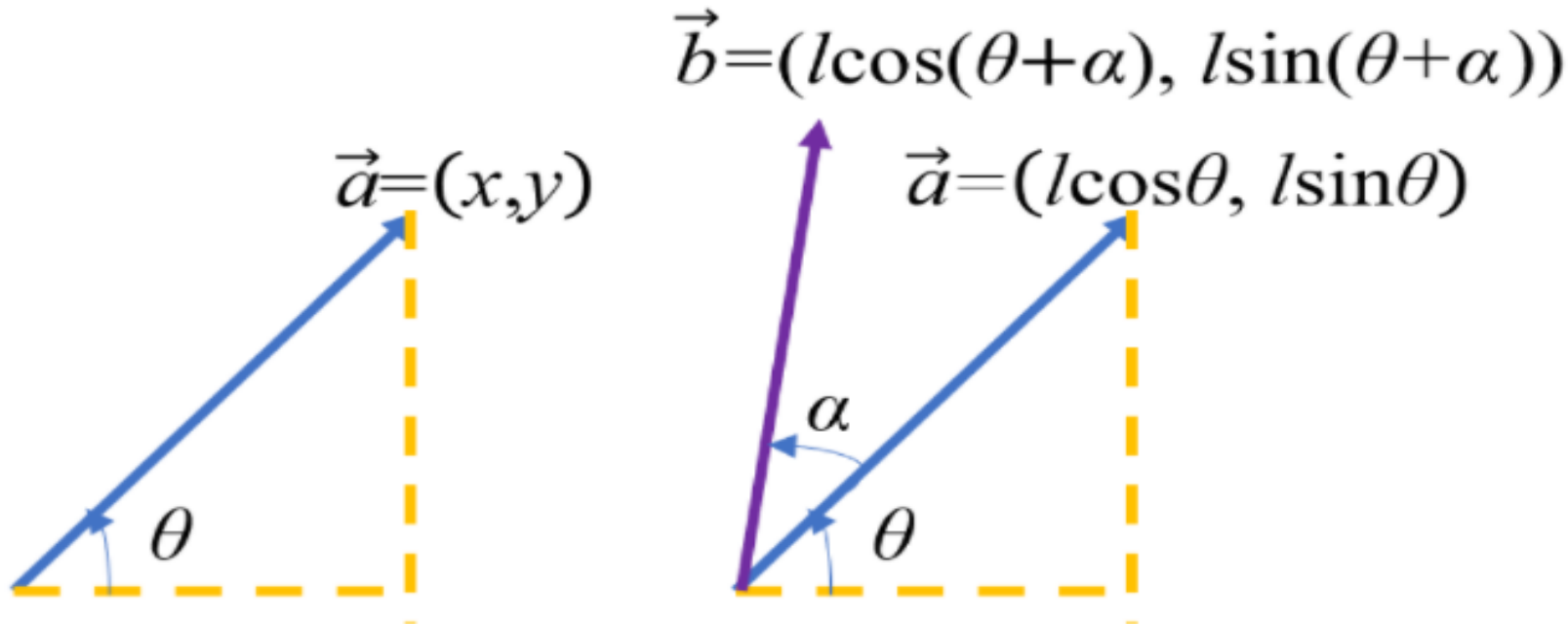
$$(x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)$$

矩阵形式

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

来证明一下?

设 $\boldsymbol{a} = (x, y)$，倾角为 θ，长度为 $l = \sqrt{x^2 + y^2}$。则 $x = l\cos\theta, y = l\sin\theta$。令其逆时针旋转 α 度角，得到向量 $\boldsymbol{b} = (l\cos(\theta + \alpha), l\sin(\theta + \alpha))$。



由三角恒等变换得，

$$\boldsymbol{b} = (l(\cos\theta\cos\alpha - \sin\theta\sin\alpha), l(\sin\theta\cos\alpha + \cos\theta\sin\alpha))$$

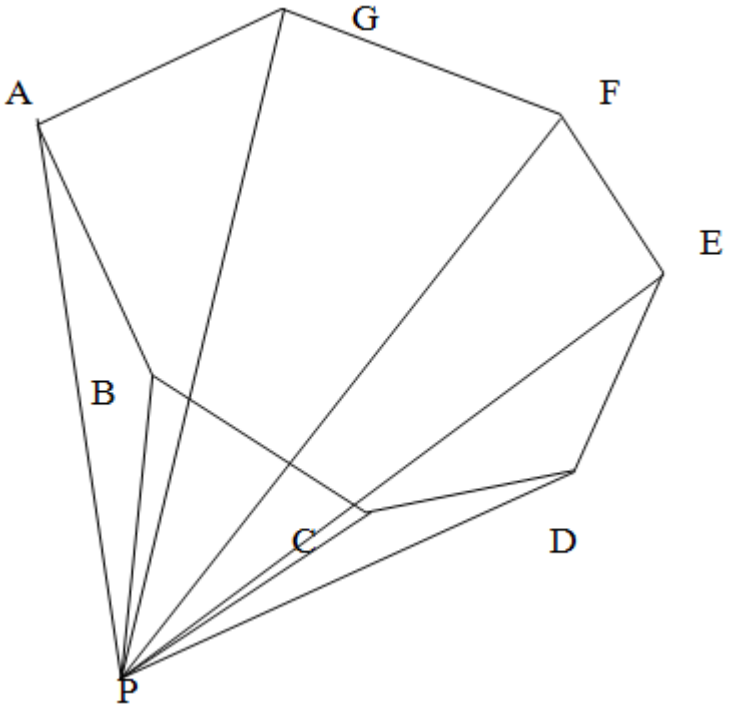
化简，

$$\boldsymbol{b} = (l\cos\theta\cos\alpha - l\sin\theta\sin\alpha, l\sin\theta\cos\alpha + l\cos\theta\sin\alpha)$$

把上面的 x, y 代回来得

$$\boldsymbol{b} = (x\cos\alpha - y\sin\alpha, y\cos\alpha + x\sin\alpha)$$

多边形的面积



$$S = \frac{1}{2}\left|\sum_{i=1}^{n} \mathbf{v}_i \times \mathbf{v}_{i \bmod n+1}\right|$$

考虑向量积的模的几何意义，我们可以利用向量积完成。

将多边形上的点逆时针标记为 $p_1, p_2, \cdots, p_n$，再任选一个辅助点 O，记向量 $\mathbf{v}_i = p_i - O$，那么这个多边形面积 S 可以表示为：

$$S = \frac{1}{2}\left|\sum_{i=1}^{n} \mathbf{v}_i \times \mathbf{v}_{i \bmod n+1}\right|$$

假定多边形顶点坐标顺序为A-G，逆时针为正方向，则有如下结论：

PAB,PBC,PCD均为顺时针，面积为负；

PDE，PEF，PFG，PGA均未逆时针，面积为正；

但无论正负，均可通过P点与顶点连线的矢量叉乘完成，叉乘结果中已包含面积的正负。

极坐标与极坐标系

这样，我们在平面上选一定点 O，称为 **极点**，自极点引出一条射线 Ox，称为 **极轴**，再选择一个单位长度（在数学问题中通常为 1），一个角度单位（通常为弧度）及其正方向（通常为逆时针方向），这样就建立了 **极坐标系**。

在极坐标系下，我们怎么描述位置呢？

设 A 为平面上一点，极点 O 与 A 之间的距离 $|OA|$ 即为 **极径**，记为 ρ；以极轴为始边，OA 为终边的角 $\angle xOA$ 为 **极角**，记为 θ，那么有序数对 (ρ,θ) 即为 A 的 **极坐标**。

由终边相同的角的定义可知，(ρ,θ) 与 $(\rho,\theta+2k\pi)$ $(k\in\mathbb{Z})$ 其实表示的是一样的点，特别地，极点的极坐标为 $(0,\theta)$ $(\theta\in\mathbb{R})$，于是平面内的点的极坐标表示有无数多种。

如果规定 $\rho>0,0\le\theta<2\pi$，那么除极点外，其他平面内的点可以用唯一有序数对 (ρ,θ) 表示，而极坐标 (ρ,θ) 表示的点是唯一确定的。

当然，有时候研究极坐标系下的图形有些不方便，我们想要转到直角坐标系下研究，那么我们有互化公式。

点 $A(\rho,\theta)$ 的直角坐标 (x,y) 可以如下表示：

$$\begin{cases}x=\rho\cos\theta\\y=\rho\sin\theta\end{cases}$$

在编程中，若要求反正切函数，尽量使用 atan2(y, x)，这个函数用途比 atan(x) 广泛。

误差分析

```
6 int dcmp(double x) {
7     static double eps = 1e-8;
8     return (x > eps) - (x < -eps);
9 }
```

欧氏距离

欧氏距离，一般也称作欧几里得距离。在平面直角坐标系中，设点 A, B 的坐标分别为 $A(x_1, y_1), B(x_2, y_2)$，则两点间的欧氏距离为：

$$|AB| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

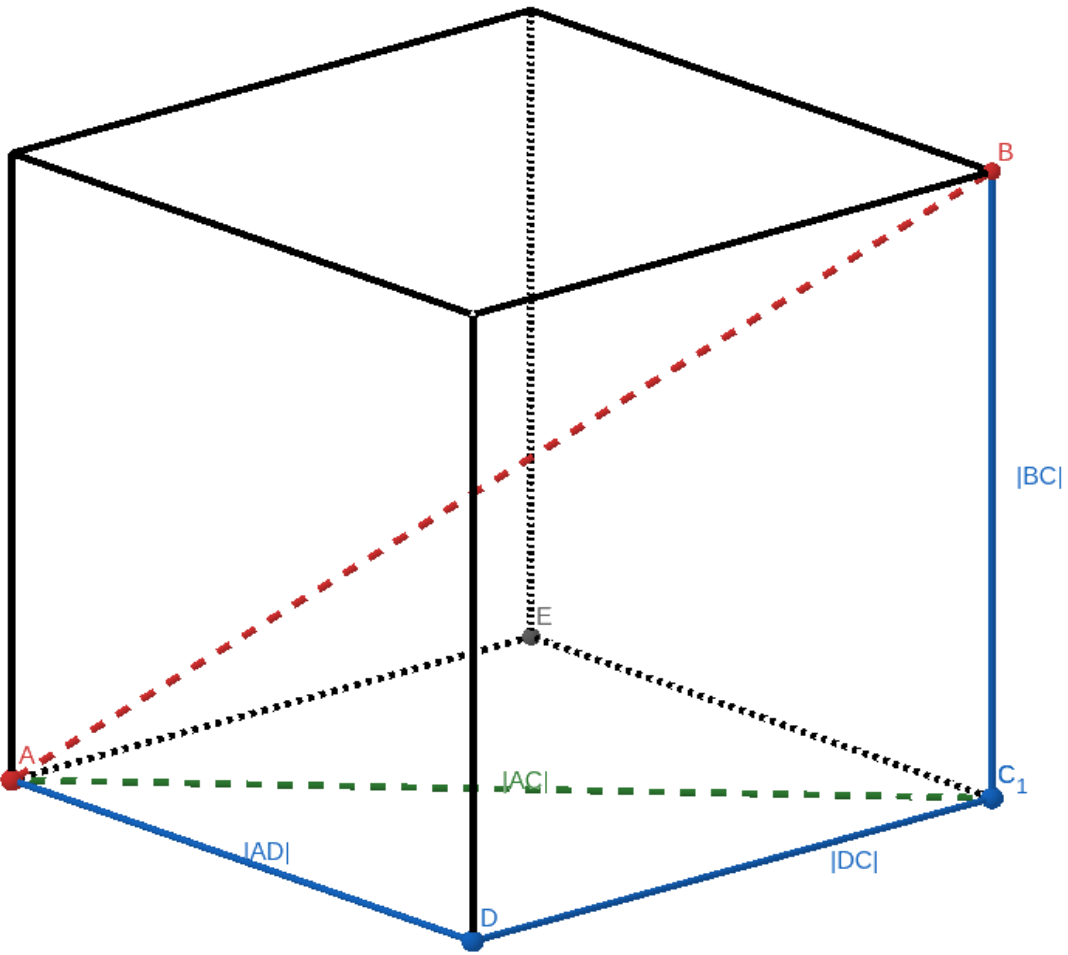
举个例子，若在平面直角坐标系中，有两点 $A(6, 5), B(2, 2)$，通过公式，我们很容易得到 A, B 两点间的欧氏距离：

$$|AB| = \sqrt{(2 - 6)^2 + (2 - 5)^2} = \sqrt{4^2 + 3^2} = 5$$

除此之外，$P(x, y)$ 到原点的欧氏距离可以用公式表示为：

$$|P| = \sqrt{x^2 + y^2}$$

三维空间欧氏距离



我们很容易发现，在 $\triangle ADC$ 中，$\angle ADC = 90°$；在 $\triangle ACB$ 中，$\angle ACB = 90°$。

$$\therefore |AB| = \sqrt{|AC|^2 + |BC|^2}$$
$$= \sqrt{|AD|^2 + |CD|^2 + |BC|^2}$$

由此可得，三维空间中欧氏距离的距离公式为：

$$|AB| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$
$$|P| = \sqrt{x^2 + y^2 + z^2}$$

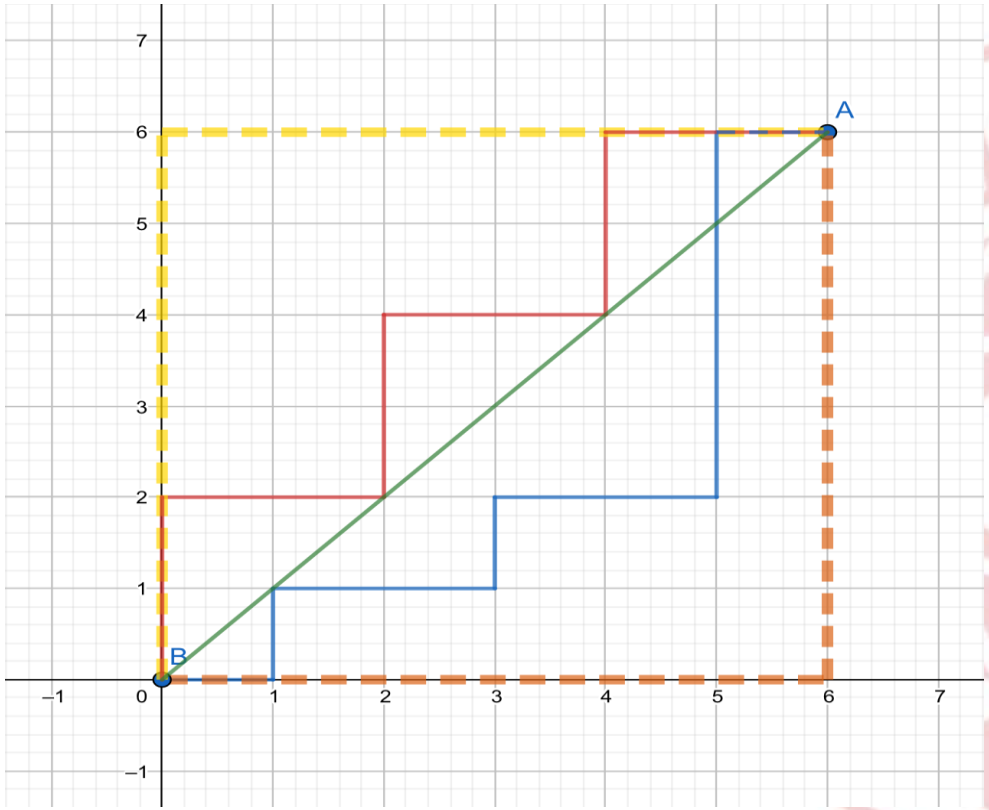
例1 【NOIP2017】奶酪

以此类推，我们就得到了 n 维空间中欧氏距离的距离公式：对于 $\vec{A}(x_{11}, x_{12}, \cdots, x_{1n})$，$\vec{B}(x_{21}, x_{22}, \cdots, x_{2n})$，有

$$\|\overrightarrow{AB}\| = \sqrt{(x_{11}-x_{21})^2 + (x_{12}-x_{22})^2 + \cdots + (x_{1n}-x_{2n})^2}$$
$$= \sqrt{\sum_{i=1}^{n}(x_{1i}-x_{2i})^2}$$

欧氏距离虽然很有用，但也有明显的缺点。两个整点计算其欧氏距离时，往往答案是浮点型，会存在一定误差。

曼哈顿距离



在二维空间内，两个点之间的曼哈顿距离（Manhattan distance）为它们横坐标之差的绝对值与纵坐标之差的绝对值之和。设点 $A(x_1, y_1), B(x_2, y_2)$，则 A, B 之间的曼哈顿距离用公式可以表示为：

$$d(A, B) = |x_1 - x_2| + |y_1 - y_2|$$

在AB间，黄线、橙线都表示曼哈顿距离，而红线、蓝线表示等价的曼哈顿距离，绿线表示欧氏距离。

曼哈顿距离数学性质

- **非负性**

  曼哈顿距离是一个非负数。

  $d(i,j) \geq 0$

- **统一性**

  点到自身的曼哈顿距离为 0。

  $d(i,i) = 0$

- **对称性**

  A 到 B 与 B 到 A 的曼哈顿距离相等，且是对称函数。

  $d(i,j) = d(j,i)$

- **三角不等式**

  从点 i 到 j 的直接距离不会大于途经的任何其它点 k 的距离。

  $d(i,j) \leq d(i,k) + d(k,j)$

例2： [USACO04OPEN]Cave Cows 3

根据题意，对于式子 $|x_1 - x_2| + |y_1 - y_2|$，我们可以假设 $x_1 - x_2 \geq 0$，根据 $y_1 - y_2$ 的符号分成两种情况：

- $(y_1 - y_2 \geq 0) \rightarrow |x_1 - x_2| + |y_1 - y_2| = x_1 + y_1 - (x_2 + y_2)$
- $(y_1 - y_2 < 0) \rightarrow |x_1 - x_2| + |y_1 - y_2| = x_1 - y_1 - (x_2 - y_2)$

只要分别求出 $x + y, x - y$ 的最大值和最小值即能得出答案。

切比雪夫距离

切比雪夫距离（Chebyshev distance）是向量空间中的一种度量，二个点之间的距离定义为其各坐标数值差的最大值。[1]

在二维空间内，两个点之间的切比雪夫距离为它们横坐标之差的绝对值与纵坐标之差的绝对值的最大值。设点 $A(x_1, y_1), B(x_2, y_2)$，则 A, B 之间的切比雪夫距离用公式可以表示为：

$$d(A, B) = \max(|x_1 - x_2|, |y_1 - y_2|)$$

曼哈顿距离与切比雪夫距离相互转化

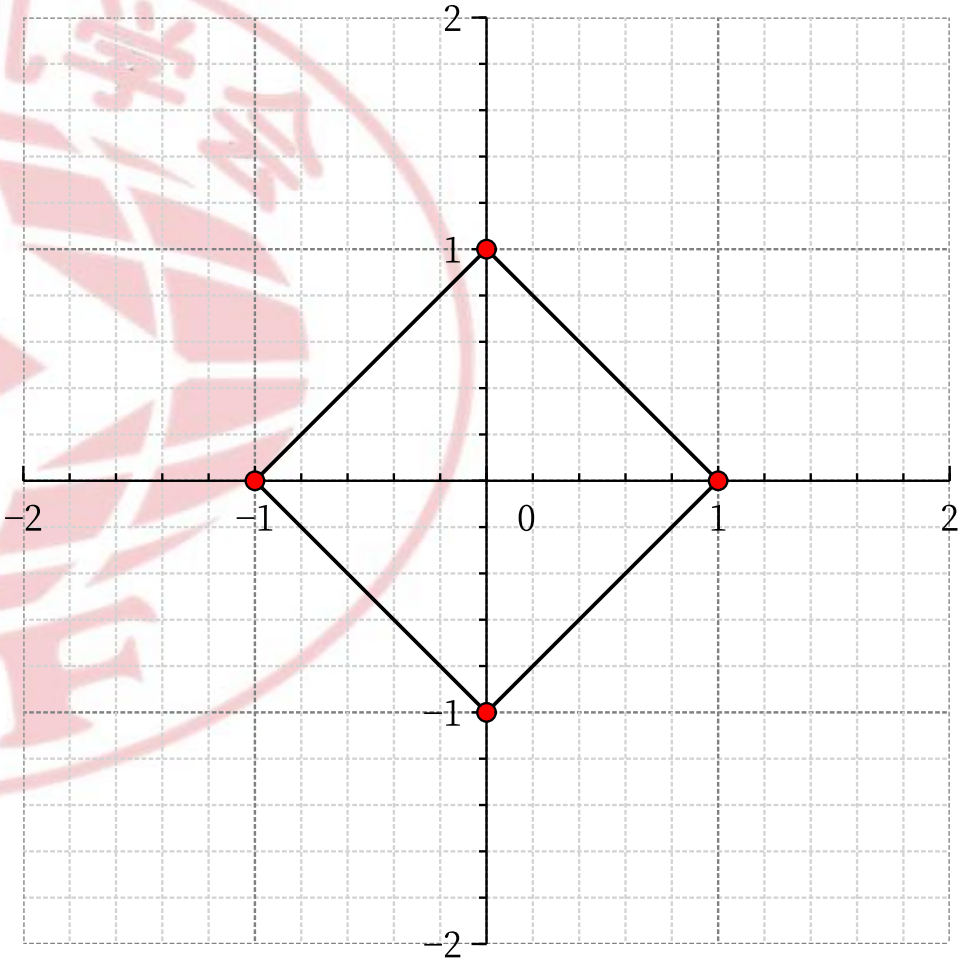
首先，我们考虑画出平面直角坐标系上所有到原点的曼哈顿距离为1的点。

通过公式，我们很容易得到方程 $|x| + |y| = 1$。

将绝对值展开，得到 4 个一次函数，分别是：

$$
\begin{aligned}
y &= -x + 1 \qquad (x \geq 0, y \geq 0) \\
y &= x + 1 \qquad (x \leq 0, y \geq 0) \\
y &= x - 1 \qquad (x \geq 0, y \leq 0) \\
y &= -x - 1 \qquad (x \leq 0, y \leq 0)
\end{aligned}
$$

将这 4 个函数画到平面直角坐标系上，得到一个边长为 $\sqrt{2}$ 的正方形，如 图所示：



曼哈顿距离与切比雪夫距离相互转化

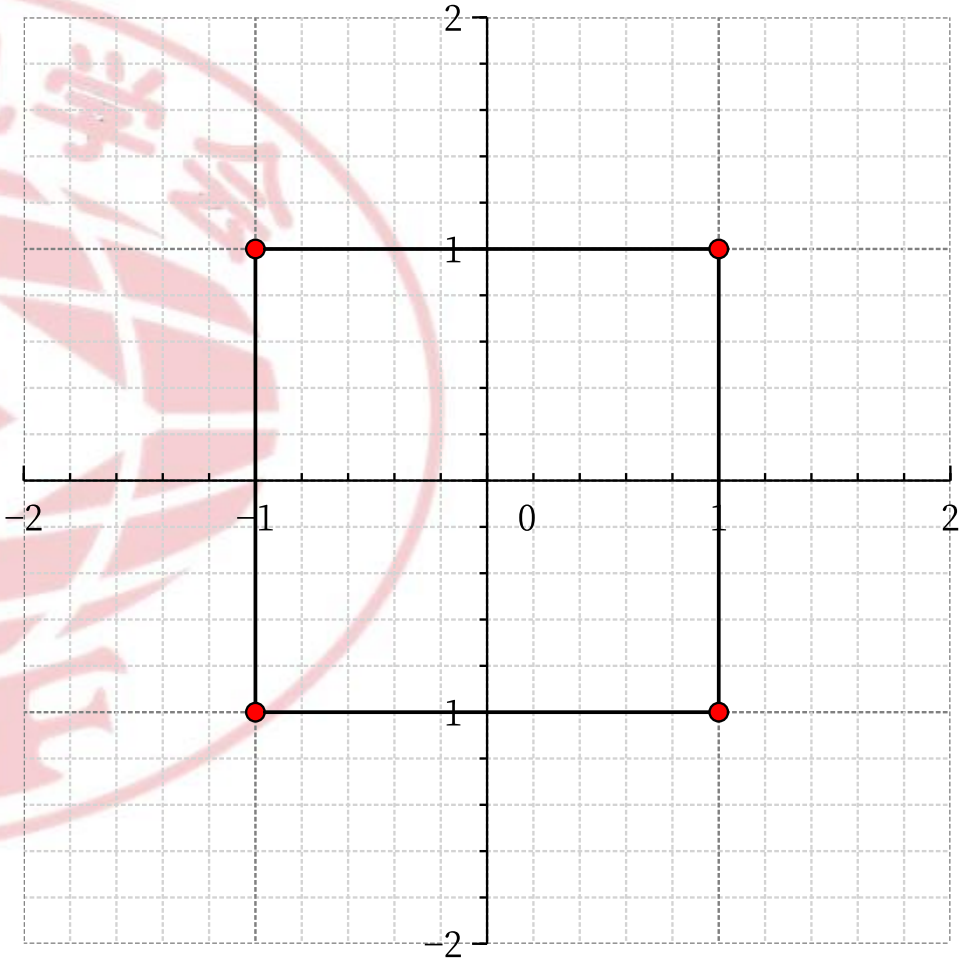
同理，我们再考虑画出平面直角坐标系上所有到原点的 切比雪夫距离为1的点。

通过公式，我们知道 $\max(|x|,|y|)=1$。

我们将式子展开，也同样可以得到可以得到 4 条 线段，分别是：

$$
\begin{aligned}
y &= 1 && (-1 \le x \le 1)\\
y &= -1 && (-1 \le x \le 1)\\
x &= 1, && (-1 \le y \le 1)\\
x &= -1, && (-1 \le y \le 1)
\end{aligned}
$$

画到平面直角坐标系上，可以得到一个边长为 2 的正方形，如 图所示：



曼哈顿距离与切比雪夫距离相互转化

这2个正方形是相似图形。所以，曼哈顿距离与切比雪夫距离之间会不会有联系呢？

接下来我们简略证明一下：

假设 $A(x_1, y_1), B(x_2, y_2)$，

我们把曼哈顿距离中的绝对值拆开，能够得到四个值，这四个值中的最大值是两个非负数之和，即曼哈顿距离。则 A, B 两点的曼哈顿距离为：

$$\begin{aligned} d(A,B) &= |x_1 - x_2| + |y_1 - y_2| \\ &= \max\{x_1 - x_2 + y_1 - y_2, x_1 - x_2 + y_2 - y_1, x_2 - x_1 + y_1 - y_2, x_2 - x_1 + y_2 - y_1\} \\ &= \max(|(x_1 + y_1) - (x_2 + y_2)|, |(x_1 - y_1) - (x_2 - y_2)|) \end{aligned}$$

我们很容易发现，这就是 $(x_1 + y_1, x_1 - y_1), (x_2 + y_2, x_2 - y_2)$ 两点之间的切比雪夫距离。

所以将每一个点 (x, y) 转化为 $(x + y, x - y)$，新坐标系下的切比雪夫距离即为原坐标系下的曼哈顿距离。

同理，A, B 两点的切比雪夫距离为：

$$\begin{aligned} d(A,B) &= \max\{|x_1 - x_2|, |y_1 - y_2|\} \\ &= \max\left\{\left|\frac{x_1 + y_1}{2} - \frac{x_2 + y_2}{2}\right| + \left|\frac{x_1 - y_1}{2} - \frac{x_2 - y_2}{2}\right|\right\} \end{aligned}$$

而这就是 $(\frac{x_1 + y_1}{2}, \frac{x_1 - y_1}{2}), (\frac{x_2 + y_2}{2}, \frac{x_2 - y_2}{2})$ 两点之间的曼哈顿距离。

所以将每一个点 (x, y) 转化为 $(\frac{x + y}{2}, \frac{x - y}{2})$，新坐标系下的曼哈顿距离即为原坐标系下的切比雪夫距离。

曼哈顿距离与切比雪夫距离相互转化

结论

- 曼哈顿坐标系是通过切比雪夫坐标系旋转 45° 后，再缩小到原来的一半得到的。
- 将一个点 (x, y) 的坐标变为 $(x+y, x-y)$ 后，原坐标系中的曼哈顿距离等于新坐标系中的切比雪夫距离。
- 将一个点 (x, y) 的坐标变为 $(\frac{x+y}{2}, \frac{x-y}{2})$ 后，原坐标系中的切比雪夫距离等于新坐标系中的曼哈顿距离。

碰到求切比雪夫距离或曼哈顿距离的题目时，我们往往可以相互转化来求解。两种距离在不同的题目中有不同的优缺点，应该灵活运用。

例2： [USACO04OPEN]Cave Cows 3

我们考虑将题目所求的曼哈顿距离转化为切比雪夫距离，即把每个点的坐标 (x, y) 变为 $(x + y, x - y)$。

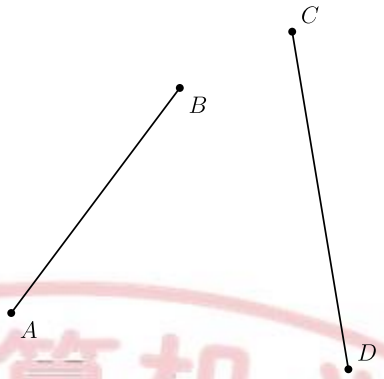
所求的答案就变为 $\max\limits_{i,j \in n} \{ \max \{ |x_i - x_j|, |y_i - y_j| \} \}$。

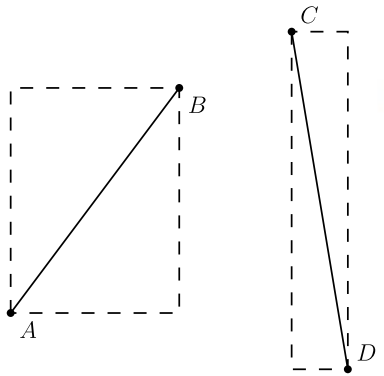
现要使得横坐标之差和纵坐标之差最大，只需要预处理出 x, y 的最大值和最小值即可。

判断

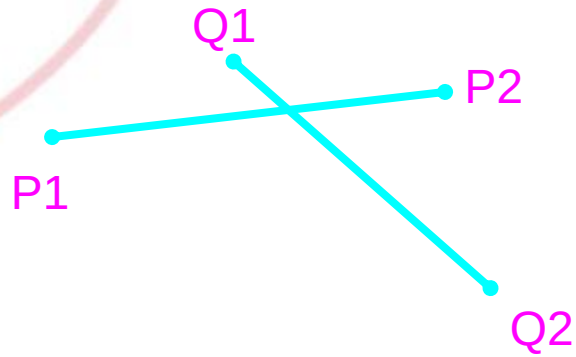
- **判断点是否在线段上**
  - 设点为Q，线段为P1P2 。
  - 判断点Q在该线段上的依据是：
    - (Q - P1) × (P2 - P1) = 0
    - Q 在以 P1，P2为对角顶点的矩形内。
  - 前者保证Q点在直线P1P2上，后者是保证Q点不在线段P1P2的延长线或反向延长线上。

判断



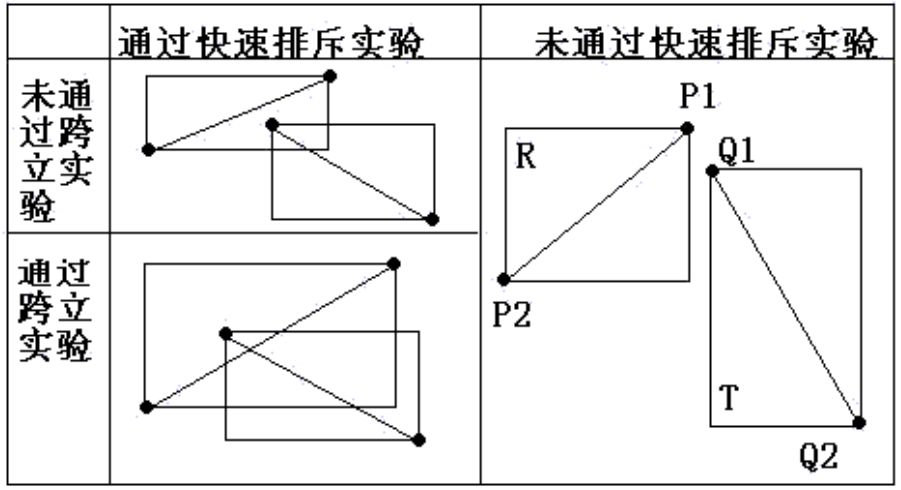


- **判断两线段是否相交**
  - 1.快速排斥试验
    - 设以线段 P1P2 为对角线的矩形为R， 设以线段 Q1Q2 为对角线的矩形为T，如果R和T不相交，显然两线段不会相交。
  - 2.跨立实验
    - 如果两线段相交，则两线段必然相互跨立对方。
    - 若P1P2跨立Q1Q2，则矢量 (P1 - Q1) 和(P2 - Q1)位于矢量(Q2 - Q1) 的两侧.
      - 即(P1 - Q1) × (Q2 - Q1) * (P2 - Q1) × (Q2 - Q1) < 0



判断

- **判断两线段是否相交**
  - 2.跨立实验
    - 上式可改写成(P1 - Q1) × (Q2 - Q1) * (Q2 - Q1) × (P2 - Q1) > 0。
    - 故判断P1P2跨立Q1Q2的依据是：(P1 - Q1) × (Q2 - Q1) * (Q2 - Q1) × (P2 - Q1) >= 0

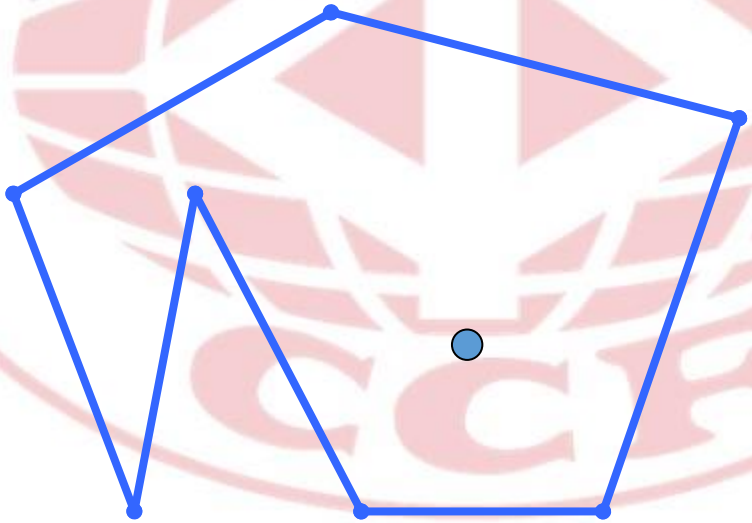


判断

- **判断线段和直线是否相交**
  - 有了上面的基础，这个算法就很容易了。如果线段P1P2和直线Q1Q2相交，则P1P2跨立Q1Q2，即：(P1 - Q1) × (Q2 - Q1) * (Q2 - Q1) × (P2 - Q1) >= 0。

判断

- **判断点是否在多边形中**
  - 判断点P是否在多边形中是计算几何中一个非常基本但是十分重要的算法。

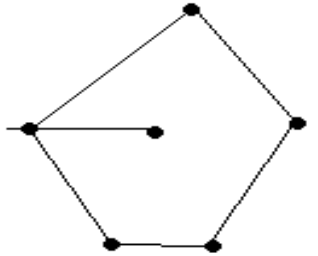


判断

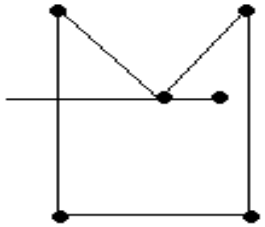
- **判断点是否在多边形中**
  - 以点P为端点，向左方作射线L.
    - 由于多边形是有界的，所以射线L的左端一定在多边形外.
  - 考虑沿着L从无穷远处开始自左向右移动，遇到和多边形的第一个交点的时候，进入到了多边形的内部，遇到第二个交点的时候，离开了多边形，……
  - 所以很容易看出当L和多边形的交点数目C是奇数的时候，P在多边形内，是偶数的话P在多边形外。

判断

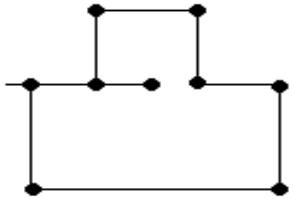
- 但是有些***特殊情况***要加以考虑。如图右图(a)(b)(c)(d)所示。
- 在图(a)中，L和多边形的顶点相交，这时候交点只能计算一个；
- 在图(b)中，L和多边形顶点的交点不应被计算；
- 在图(c)和(d) 中，L和多边形的一条边重合，这条边应该被忽略不计。



图a



图b



图c

图d

判断

- 判断点是否在多边形中
  - 是不是有些*混乱*?
  - 为了统一起见，我们在计算射线L和多边形的交点的时候，
    - 1.对于多边形的水平边不作考虑；
    - 2.对于多边形的顶点和L相交的情况，如果该顶点是其所属的边上纵坐标较大的顶点，则计数，否则忽略；
    - 3.对于P在多边形边上的情形，直接可判断P属于多边形。

判断

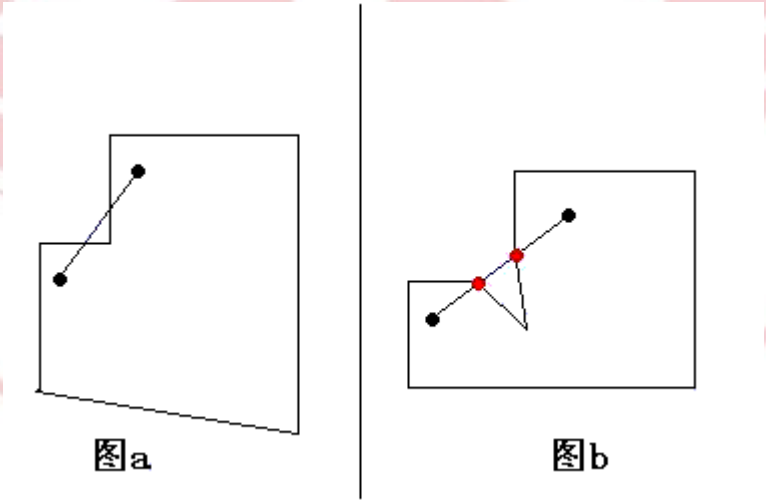
- **判断点是否在多边形中**
  - 做射线L的方法是：设P‘的纵坐标和P相同，横坐标为正无穷大，则P和P’就确定了射线L。
  - 本算法时间复杂度为O(n)

判断

- **判断线段是否在多边形内**
  - 线段在多边形内的第一个必要条件是线段的两个端点都在多边形内 。
  - 多边形可能为凹，所以这不能成为判断的充分条件 。
  - 如果线段和多边形的某条边内交， 则线段一定会有一部分在多边形外(见图a)。
    - *内交：两线段相交且交点不在两线段的端点.*
  - 我们可以得到线段在多边形内的第二个必要条件：线段和多边形的所有边都不内交。

判断

- **判断线段是否在多边形内**
  - 如果多边形的某个顶点和线段相交，还必须判断两相邻交点之间的线段是否包含于多边形内部（反例见图b)。



判断

- **判断线段是否在多边形内**
  - 求出所有和线段相交的多边形的顶点。
  - 按照X为第一关键字，Y为第二关键字将坐标排序，这样相邻的两个点就是在线段上相邻的两交点，如果任意相邻两点的中点也在多边形内，则该线段一定在多边形内。

判断

- 其他诸如判断点是否在圆内，判断折线是否在多边形内，判断多边形是否在多边形内，判断线段、折线、矩形、多边形是否在圆内 等较为简单，在此不再阐述。

计算

- **计算点到线段的最近点**
  - 如果该线段平行于X轴（Y轴）：
    - 过点point作该线段所在直线的垂线，计算出垂足.
    - 如果垂足在线段上则返回垂足，否则返回离垂足近的端点；
  - 如果该线段不平行于X轴也不平行于Y轴:
    - 斜率存在且不为0。
    - 设线段的两端点为pt1和pt2，
    - 斜率为： k = (pt2.y - pt1.y) / (pt2.x - pt1.x);
    - 该直线方程为： y = k* (x - pt1.x) + pt1.y。
    - 其垂线的斜率为 - 1 / k，
    - 垂线方程为： y = (-1 / k) * (x - point.x) + point.y 。

计算

- **计算点到线段的最近点**
  - 如果该线段不平行于X轴也不平行于Y轴：
    - 联立两直线方程解得：
      - x = (k^2 * pt1.x + k * (point.y - pt1.y) + point.x) / (k^2 + 1)
      - y = k * (x - pt1.x) + pt1.y;
    - 再判断垂足是否在线段上，如果在线段上则返回垂足；如果不在则计算两端点到垂足的距离，选择距离垂足较近的端点返回。

计算

- **计算点到折线、矩形、多边形的最近点**
  - 只要分别计算点到每条线段的最近点，记录最近距离，取其中最近距离最小的点即可。

计算

- **计算点到圆的最近距离及交点坐标**
  - 如果该点在圆心，因为圆心到圆周任一点的距离相等，返回UNDEFINED。
  - 连接点P和圆心O
    - 如果PO平行于X轴，则根据P在O的左边还是右边计算出最近点的横坐标
    - 如果PO平行于Y轴，则根据P在O的上边还是下边计算出最近点的纵坐标

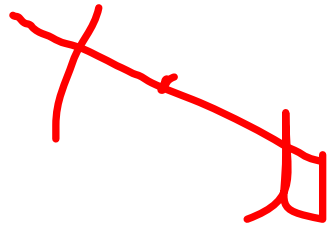
计算

- **计算点到圆的最近距离及交点坐标**
  - 连接点P和圆心O
    - 如果PO不平行于X轴和Y轴
      - 则PO的斜率存在且不为0
      - 直线PO斜率为k = （P.y - O.y）/ (P.x - O.x)
      - 直线PO的方程为：y = k * (x - P.x) + P.y
      - 设圆方程为:(x - O.x) ^2 + (y - O.y) ^2 = r ^2
    - 联立两方程组可以解出直线PO和圆的交点，取其中离P点较近的交点即可。

计算

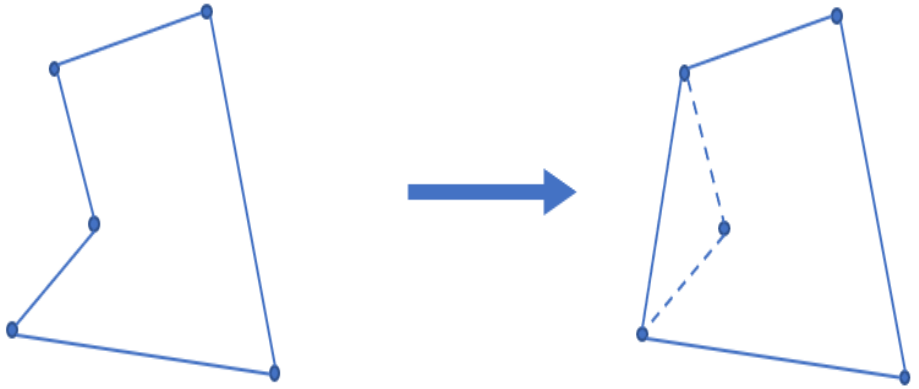
- **求线段或直线与圆的交点**
- 设圆心为O，圆半径为r，直线（或线段）L上的两点为P1,P2。
  - 1. 如果L是线段且P1，P2都包含在圆O内，则没有交点；否则进行下一步。
  - 2. 如果L平行于Y轴，
    - a) 计算圆心到L的距离dis；
    - b) 如果dis > r 则L和圆没有交点；
    - c) 利用勾股定理，可以求出两交点坐标，但要注意考虑L和圆的相切情况。

计算

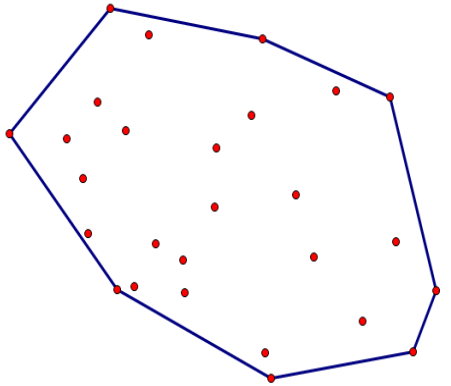


- **求线段或直线与圆的交点**
  - 3. 如果L平行于X轴，做法与L平行于Y轴的情况类似；
  - 4. 如果L既不平行X轴也不平行Y轴，可以求出L的斜率K，然后列出L的点斜式方程，和圆方程联立即可求解出L和圆的两个交点；
  - 5. 如果L是线段，对于2，3，4中求出的交点还要分别判断*是否属于该线段的范围内*。

二维凸包



- 凸多边形是指所有内角大小都在[0,π]范围内的简单多边形。
- 在平面上能包含所有给定点的最小凸多边形叫做凸包。



- 其定义为：对于给定集合X，所有包含S的凸集的交集S被称为X的凸包。
- 实际上可以理解为用一个橡皮筋包含住所有给定点的形态。
- 凸包用最小的周长围住了给定的所有点。如果一个凹多边形围住了所有的点，它的周长一定不是最小，如下图。根据三角不等式，凸多边形在周长上一定是最优的。

凸包求法

- Andrew 算法

该算法的时间复杂度为 $O(n \log n)$，其中 n 为待求凸包点集的大小，同时复杂度的瓶颈也在于对所有点坐标的双关键字排序。

首先把所有点以横坐标为第一关键字，纵坐标为第二关键字排序。

显然排序后最小的元素和最大的元素一定在凸包上。而且因为是凸多边形，我们如果从一个点出发逆时针走，轨迹总是“左拐”的，一旦出现右拐，就说明这一段不在凸包上。因此我们可以用一个单调栈来维护上下凸壳。

因为从左向右看，上下凸壳所旋转的方向不同，为了让单调栈起作用，我们首先 **升序枚举** 求出下凸壳，然后 **降序** 求出上凸壳。

求凸壳时，一旦发现即将进栈的点（P）和栈顶的两个点（S_1, S_2，其中 S_1 为栈顶）行进的方向向右旋转，即叉积小于 0：$\overrightarrow{S_2S_1} \times \overrightarrow{S_1P} < 0$，则弹出栈顶，回到上一步，继续检测，直到 $\overrightarrow{S_2S_1} \times \overrightarrow{S_1P} \geq 0$ 或者栈内仅剩一个元素为止。

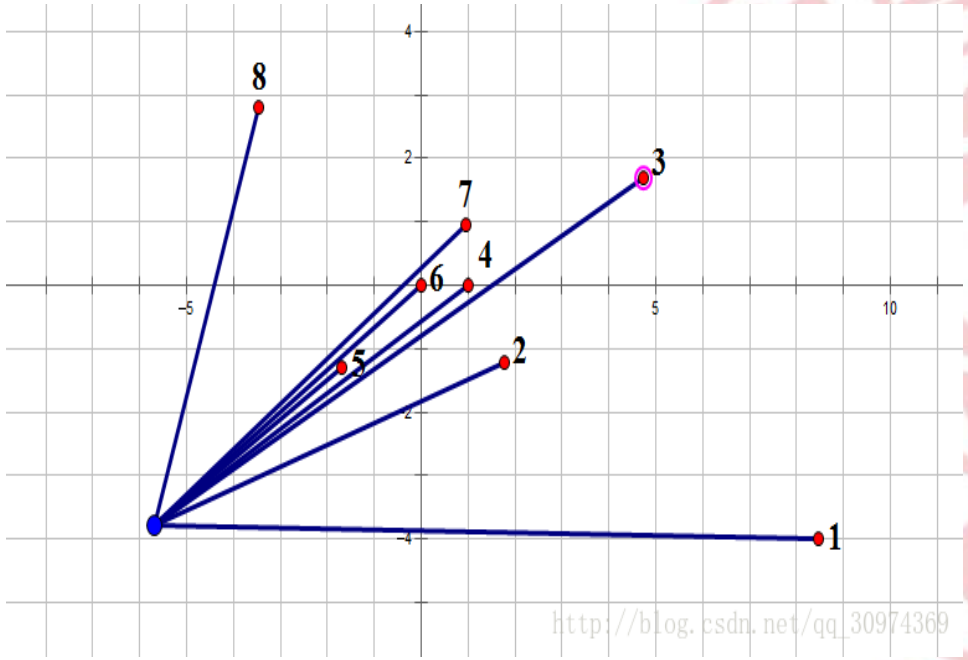
通常情况下不需要保留位于凸包边上的点，因此上面一段中 $\overrightarrow{S_2S_1} \times \overrightarrow{S_1P} < 0$ 这个条件中的“$<$”可以视情况改为 $\leq$，同时后面一个条件应改为 $>$。

Andrew 算法

```cpp
// C++ Version
// stk[] 是整型，存的是下标
// p[] 存储向量或点
tp = 0;                                 // 初始化栈
std::sort(p + 1, p + 1 + n);  // 对点进行排序
stk[++tp] = 1;
//栈内添加第一个元素，且不更新 used，使得 1 在最后封闭凸包时也对单调栈更新
for (int i = 2; i <= n; ++i) {
  while (tp >= 2  // 下一行 * 操作符被重载为叉积
         && (p[stk[tp]] - p[stk[tp - 1]]) * (p[i] - p[stk[tp]]) <= 0)
    used[stk[tp--]] = 0;
  used[i] = 1;  // used 表示在凸壳上
  stk[++tp] = i;
}
int tmp = tp;  // tmp 表示下凸壳大小
for (int i = n - 1; i > 0; --i)
  if (!used[i]) {
    //      ↓求上凸壳时不影响下凸壳
    while (tp > tmp && (p[stk[tp]] - p[stk[tp - 1]]) * (p[i] - p[stk[tp]])
      used[stk[tp--]] = 0;
    used[i] = 1;
    stk[++tp] = i;
  }
for (int i = 1; i <= tp; ++i)  // 复制到新数组中去
  h[i] = p[stk[i]];
int ans = tp - 1;
```

凸包求法

- Graham 扫描法



- 找到最靠近左下的一个点
- 其他的点按照极角排序
- 然后把1丢到凸包的栈里面，准备开始扫描
- 检查2号点是否在1的一侧，（检查一下是不是凸多边形）
- 这里检查到2号可行，先加入到栈中
- 检查到3更加靠近外侧（如果加入3号就会形成凹多边形，显然3在凸包中，而2不在）
- 然后把2号点弹出栈，判断1号和3号节点的关系（同判断2号）
- 依次这么判断，最后所有凸包上的点都会在栈中

Graham 扫描法

```
struct Node{
        int x,y;
}p[MAX],S[MAX];//p储存节点的位置，S是凸包的栈
inline bool cmp(Node a,Node b)//比较函数，对点的极角进行排序 {
        double A=atan2((a.y-p[1].y),(a.x-p[1].x));
        double B=atan2((b.y-p[1].y),(b.x-p[1].x));
        if(A!=B)return A<B;
        else    return a.x<b.x; //这里注意一下，如果极角相同，优先放x坐标更小的点
}
long long Cross(Node a,Node b,Node c)//计算叉积 {
        return 1LL*(b.x-a.x)*(c.y-a.y)-1LL*(b.y-a.y)*(c.x-a.x);
}
void Get()//求出凸包 {
        p[0]=(Node){INF,INF};int k;
        for(int i=1;i<=n;++i)//找到最靠近左下的点
                if(p[0].y>p[i].y||(p[0].y==p[i].y&&p[i].x<p[0].x))
                 {p[0]=p[i];k=i;}
        swap(p[k],p[1]);
        sort(&p[2],&p[n+1],cmp);//对于剩余点按照极角进行排序
        S[0]=p[1],S[1]=p[2];top=1;//提前在栈中放入节点
        for(int i=3;i<=n;)//枚举其他节点
        {
                if(top&&Cross(S[top-1],p[i],S[top])>=0)
                        top--;//如果当前栈顶不是凸包上的节点则弹出
                else  S[++top]=p[i++];//加入凸包的栈中
        }
        //底下这个玩意用来输出凸包上点的坐标
        //for(int i=0;i<=top;++i)
        //    printf("(%d,%d)\n",S[i].x,S[i].y);
}
```

例题3:[USACO5.1]圈奶牛Fencing the Cows

- 农夫约翰想要建造一个围栏用来围住他的奶牛，可是他资金匮乏。他建造的围栏必须包括他的奶牛喜欢吃草的所有地点。对于给出的这些地点的坐标，计算最短的能够围住这些点的围栏的长度。

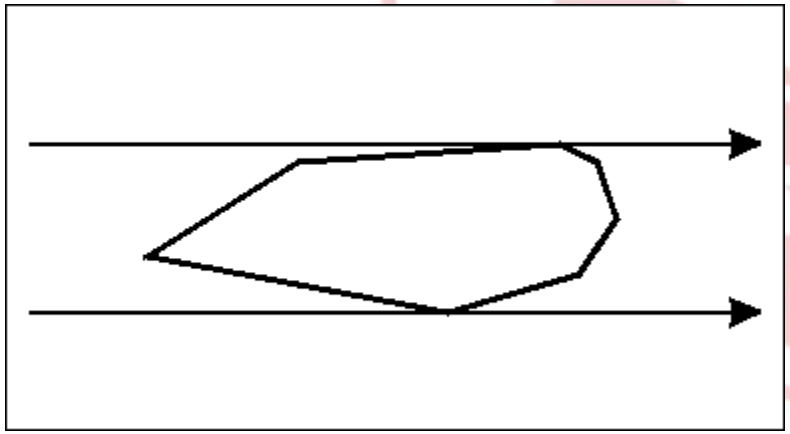
1 按x坐标为第一关键字,y坐标为第二关键字从小到大排序

2 从左到右枚举点，如果当前点相对于前面两个是向外的，就不断让栈顶出去。之后把点加入栈中。从而得到下凸包。

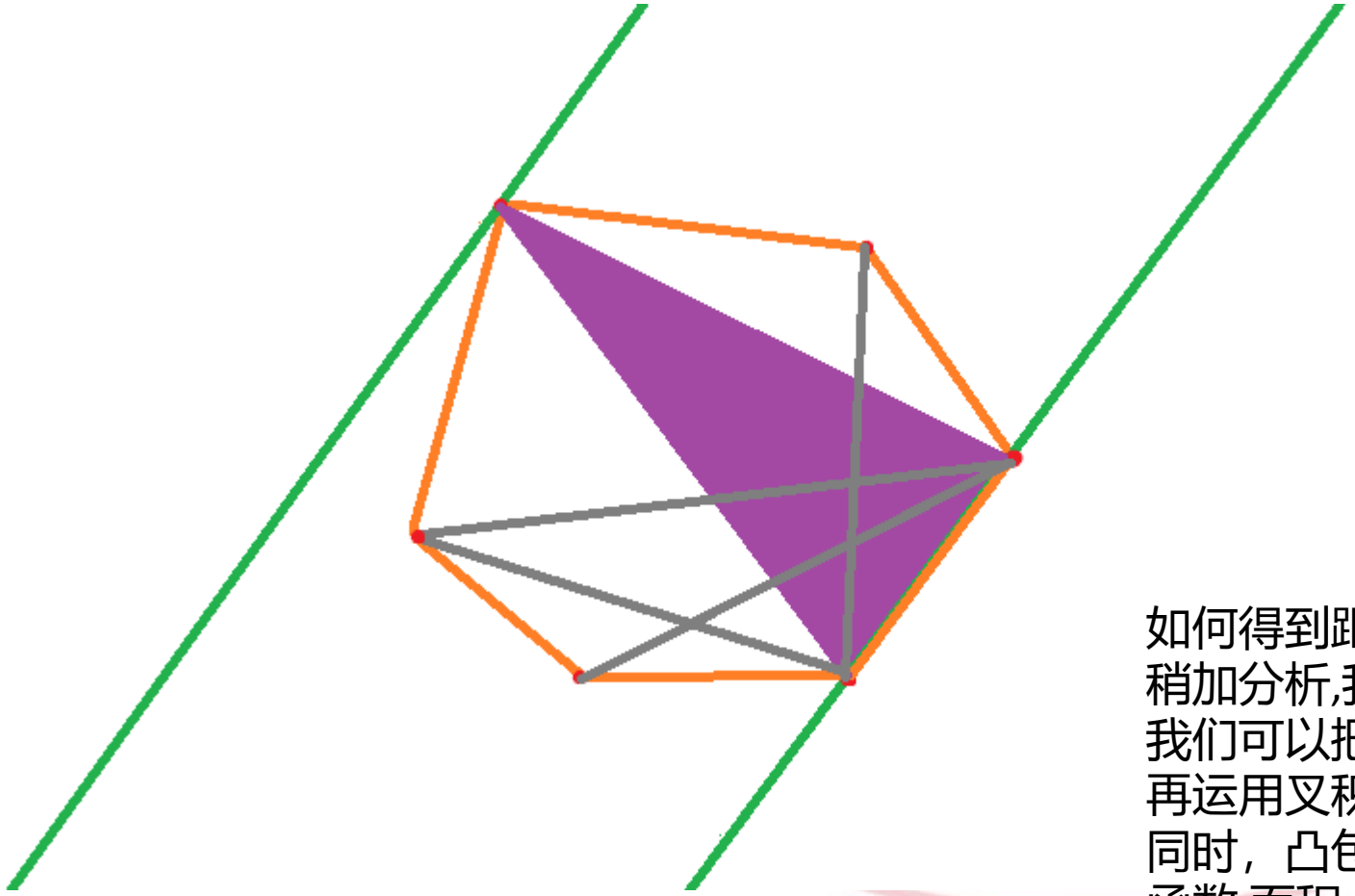
3 从右到左再做一遍。

旋转卡壳

- 旋转卡壳算法在凸包算法的基础上，通过枚举凸包上某一条边的同时维护其他需要的点，
- 能够在线性时间内求解如凸包直径、最小矩形覆盖等和凸包性质相关的问题。
- ”旋转卡壳“是很形象的说法，因为根据我们枚举的边，可以从每个维护的点画出一条或平行或垂直的直线，为了确保对于当前枚举的边的最优性，我们的任务就是使这些直线能将凸包正好卡住。
- 而边通常是按照向某一方向旋转的顺序来枚举，所以整个过程就是在边”旋转“，边”卡壳“。



旋转卡壳



如何得到距离每条对应边的的最远点呢?
稍加分析,我们可以发现,
我们可以把点到直线的距离化解为三角形的面积,
再运用叉积进行比较。
同时，凸包上的点依次与对应边产生的距离成单峰函数,面积上升到最高点后，又会下降。

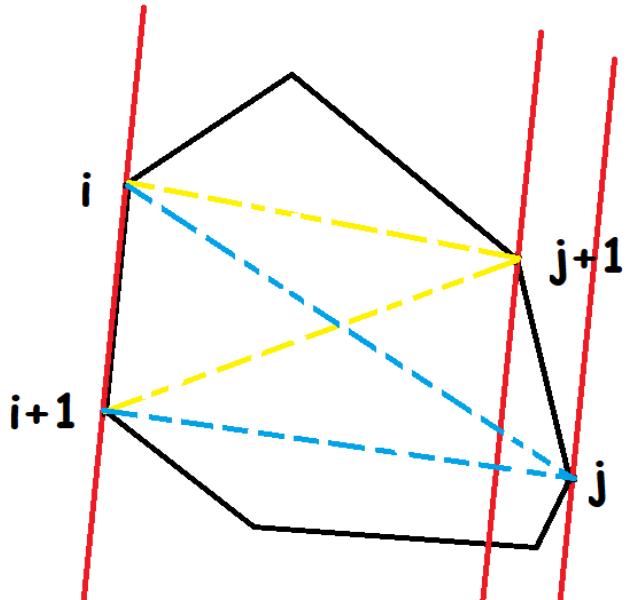
例4：[USACO03FALL]Beauty

- 给定平面上N个点，求所有点对之间的最长距离。

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。

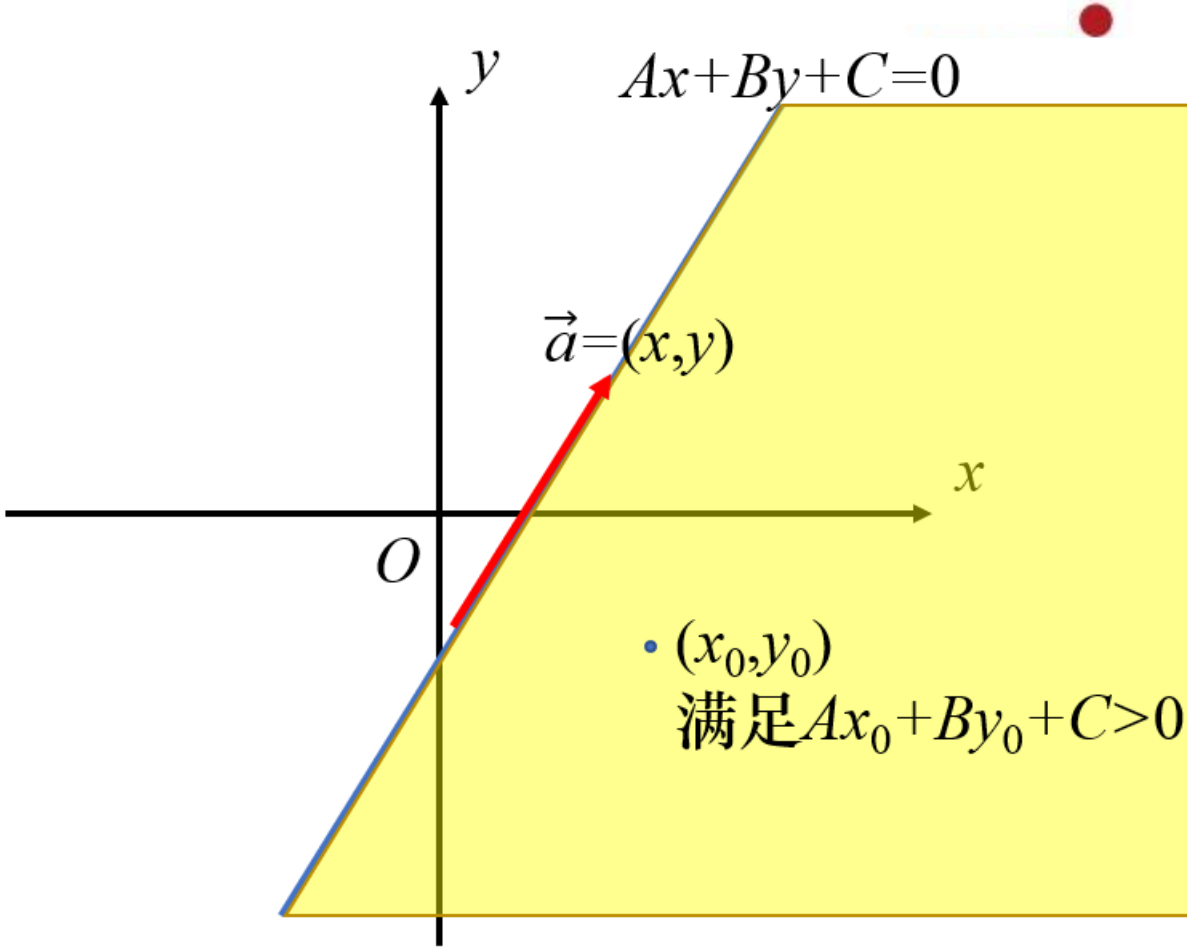


求凸包直径

```cpp
// C++ Version
int sta[N], top;  // 将凸包上的节点编号存在栈里，第一个和最后一个节点编号相同
bool is[N];
ll pf(ll x) { return x * x; }
ll dis(int p, int q) { return pf(a[p].x - a[q].x) + pf(a[p].y - a[q].y); }
ll sqr(int p, int q, int y) { return abs((a[q] - a[p]) * (a[y] - a[q])); }
ll mx;
void get_longest() {  // 求凸包直径
  int j = 3;
  if (top < 4) {
    mx = dis(sta[1], sta[2]);
    return;
  }
  for (int i = 1; i <= top; ++i) {
    while (sqr(sta[i], sta[i + 1], sta[j]) <=
           sqr(sta[i], sta[i + 1], sta[j % top + 1]))
      j = j % top + 1;
    mx = max(mx, max(dis(sta[i + 1], sta[j]), dis(sta[i], sta[j])));
  }
}
```

半平面

- 一条直线和直线的一侧。半平面是一个点集，因此是一条直线和直线的一侧构成的点集。当包含直线时，称为闭半平面；当不包含直线时，称为开半平面。
- 解析式一般为 AX+BY+C>=0。
- 在计算几何中用向量表示，整个题统一以向量的左侧或右侧为半平面。



半平面交

- 半平面交是指多个半平面的交集。因为半平面是点集，所以点集的交集仍然是点集。在平面直角坐标系围成一个区域。
- 这就很像普通的线性规划问题了，得到的半平面交就是线性规划中的可行域。一般情况下半平面交是有限的，经常考察面积等问题的解决。
- 它可以理解为向量集中每一个向量的右侧的交，或者是下面方程组的解。

$$\begin{cases} A_1x + B_1y + C \geq 0 \\ A_2x + B_2y + C \geq 0 \\ \cdots \end{cases}$$

半平面交解法 - S&I 算法

- 极角排序
  - 直接以向量为自变量，调用atan2(double y,double x)这个函数，以返回值为关键字排序，得到新的边（向量）集。
  - 排序时，如果遇到共线向量（且方向相同），则取靠近可行域的一个。比如两个向量的极角相同，而我们要的是向量的左侧半平面，那么我们只需要保留左侧的向量。判断方法是取其中一个向量的起点或终点与另一个比较，检查是在左边还是在右边。
- 维护单调队列

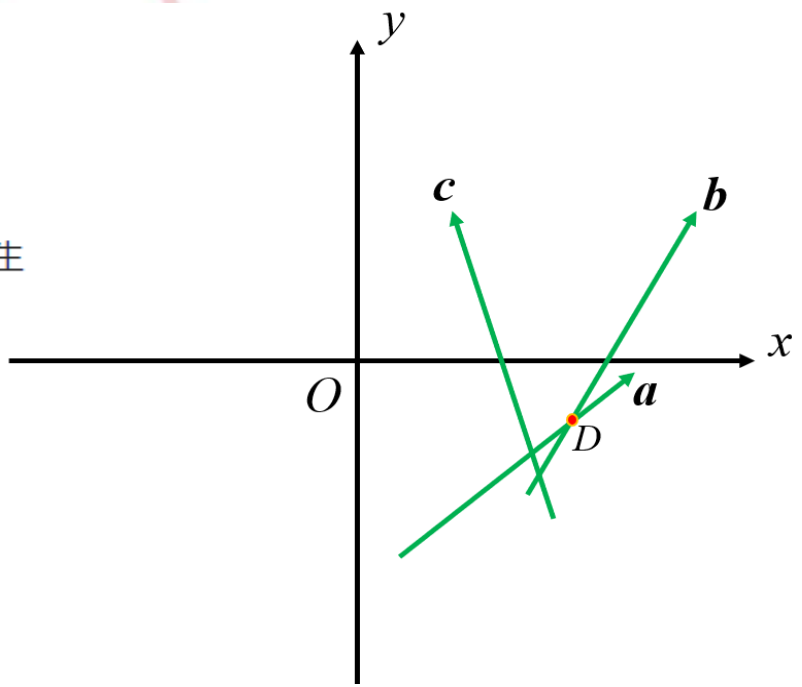
因为半平面交是一个凸多边形，所以需要维护一个凸壳。因为后来加入的只可能会影响最开始加入的或最后加入的边（此时凸壳连通），只需要删除队首和队尾的元素，所以需要用单调队列。

我们遍历排好序了的向量，并维护另一个交点数组。当单队中元素超过 2 个时，他们之间就会产生交点。

对于当前向量，如果上一个交点在这条向量表示的半平面交的异侧，那么上一条边就没有意义了。

如 图，假设取向量左侧半平面。极角排序后，遍历顺序应该是 $\vec{a} \to \vec{b} \to \vec{c}$。当 $\vec{a}$ 和 $\vec{b}$ 入队时，在交点数组里会产生一个点 D（交点数组保存队列中相同下标的向量与前一向量的交点）。

接下来枚举到 $\vec{c}$ 时，发现 D 在 $\vec{c}$ 的右侧。而因为 **产生 D 的向量的极角一定比 $\vec{c}$ 要小**，所以产生 D 的向量（指 $\vec{b}$）就对半平面交没有影响了。

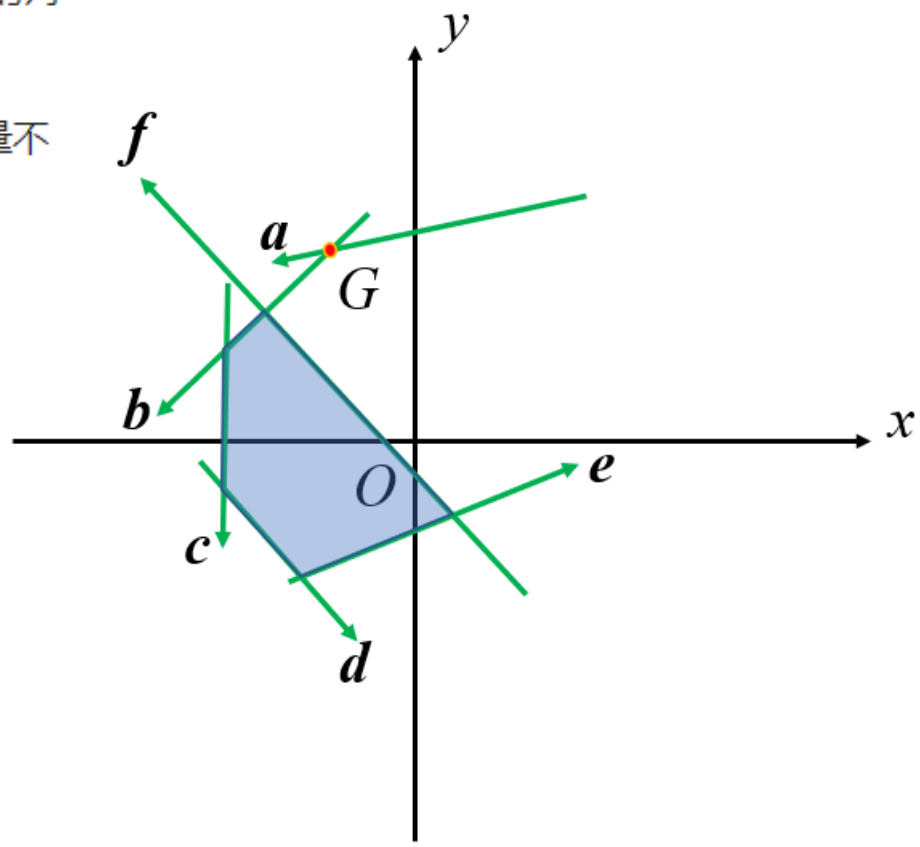


半平面交解法 - S&I 算法

还有一种可能的情况是快结束的时候，新加入的向量会从队首开始造成影响。

仍然假设取向量左侧半平面。加入向量 $\vec{f}$ 之后，第一个交点 G 就在 $\vec{f}$ 的右侧，我们把上面的判断标准逆过来看，就知道此时应该删除向量 $\vec{a}$，也即 **队首** 的向量。

最后用队首的向量排除一下队尾多余的向量。因为队首的向量会被后面的约束，而队尾的向量不会。此时它们围成了一个环，因此队首的向量就可以约束队尾的向量。



得到半平面交

如果半平面交是一个凸n边形，最后在交点数组里会得到n个点。我们再把它们首尾相连，就是一个统一方向（顺或逆时针）的n多边形。

此时就可以用三角剖分求面积了。（求面积是最基础的考法）

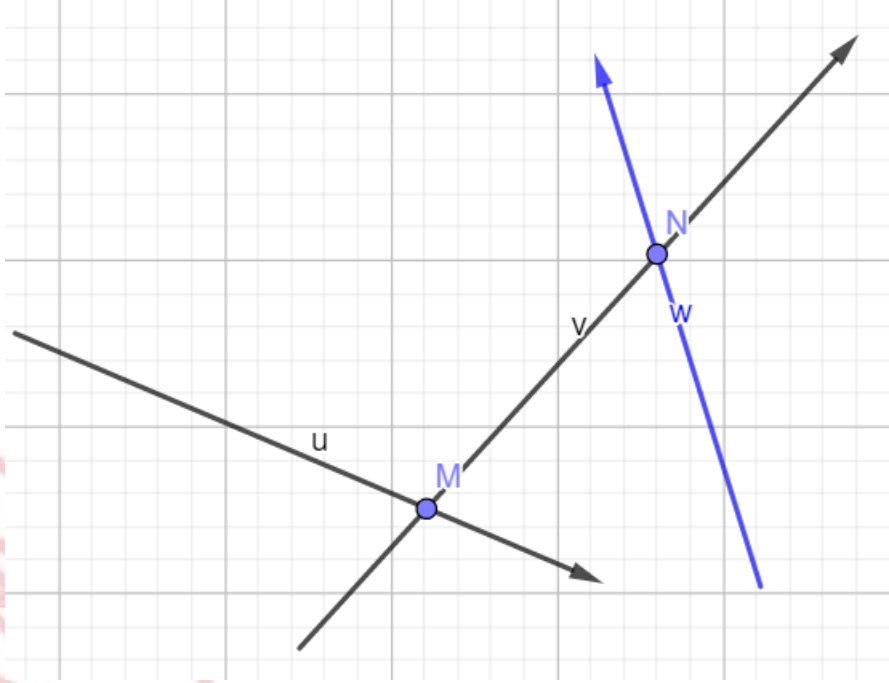
偶尔会出现半平面交不存在或面积为 0 的情况，注意考虑边界。

半平面交解法 - S&I 算法

注意事项

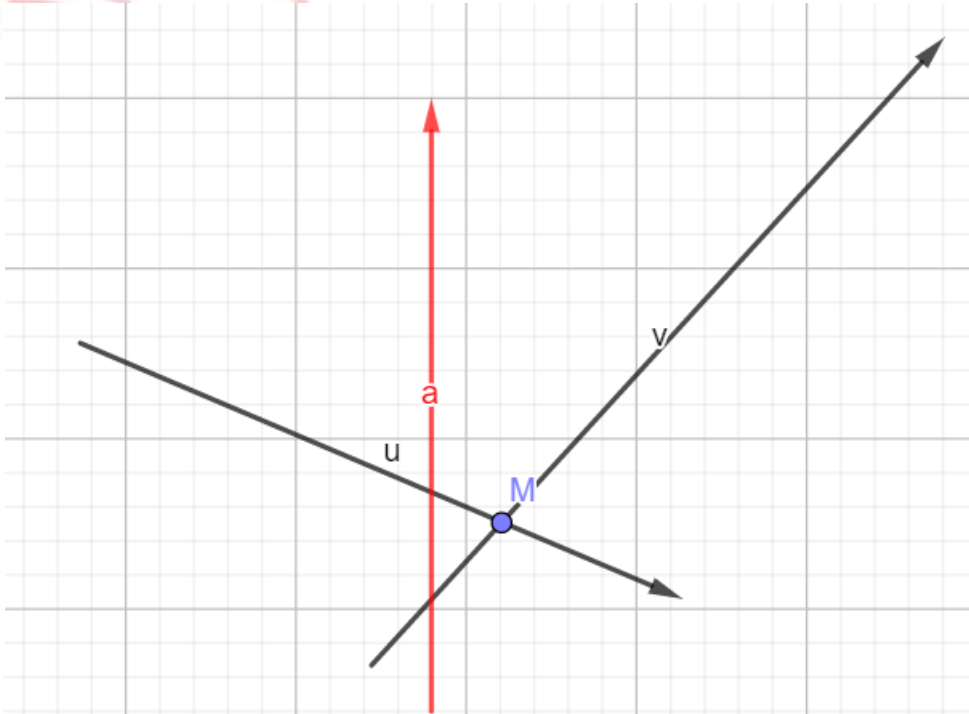
当出现一个可以把队列里的点全部弹出去的向量（即所有队列里的点都在该向量的右侧），则我们必须先处理队尾，再处理队首。
因此在循环中，我们先枚举 --r 的部分，再枚举 ++l 的部分，才不会错。
原因如下。

一般情况下，我们在队列（队列顺序为 $\{\vec{u},\vec{v}\}$）后面加一条边（向量 $\vec{w}$），会产生一个交点 N，缩小 $\vec{v}$ 后面的范围。



但是毕竟每次操作都是一般的，因此可能会有把M点“挤出去”的情况。

如果此时出现了向量 $\vec{a}$，使得 M 在 $\vec{a}$ 的右侧，那么 M 就要出队了。此时如果从队首枚举 `++l`，显然是扩大了范围。实际上 M 点是由 $\vec{u}$ 和 $\vec{v}$ 共同构成的，因此需要考虑影响到现有进程的是 $\vec{u}$ 还是 $\vec{v}$。而因为我们在极角排序后，向量是逆时针顺序，所以 $\vec{v}$ 的影响要更大一些。



就如上图，如果 M 确认在 $\vec{a}$ 的右侧，那么此时 $\vec{v}$ 的影响一定不会对半平面交的答案作出任何贡献。

而我们排除队首的原因是 **当前向量的限制比队首向量要大**，这个条件的前提是队列里有不止两个线段（向量），不然就会出现上面的情况。

所以一定要先排除队尾再排除队首。

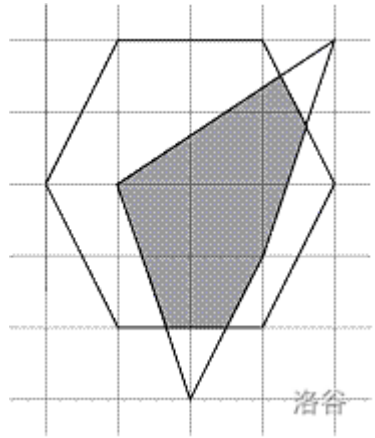
半平面交解法 - S&I 算法

```
friend bool operator<(seg x, seg y) {
  db t1 = atan2((x.b - x.a).y, (x.b - x.a).x);
  db t2 = atan2((y.b - y.a).y, (y.b - y.a).x);  // 求极角
  if (fabs(t1 - t2) > eps)                      // 如果极角不等
    return t1 < t2;
  return (y.a - x.a) * (y.b - x.a) >
         eps;  // 判断向量x在y的哪边，令最靠左的排在最左边
}
```

```
// pnt its(seg a,seg b)表示求线段a,b的交点
// s[]是极角排序后的向量
// q[]是向量队列
// t[i]是s[i-1]与s[i]的交点
//【码风】队列的范围是(l,r]
//求的是向量左侧的半平面
int l = 0, r = 0;
for (int i = 1; i <= n; ++i)
  if (s[i] != s[i - 1]) {
    // 注意要先检查队尾
    while (r - l > 1 && (s[i].b - t[r]) * (s[i].a - t[r]) >
                                eps)  // 如果上一个交点在向量右侧则弹出队尾
      --r;
    while (r - l > 1 && (s[i].b - t[l + 2]) * (s[i].a - t[l + 2]) >
                                eps)  // 如果第一个交点在向量右侧则弹出队首
      ++l;
    q[++r] = s[i];
    if (r - l > 1) t[r] = its(q[r], q[r - 1]);  // 求新交点
  }
while (r - l > 1 &&
       (q[l + 1].b - t[r]) * (q[l + 1].a - t[r]) > eps)  // 注意删除多余元素
  --r;
t[r + 1] = its(q[l + 1], q[r]);  // 再求出新的交点
++r;
//这里不能在t里面++r需要注意一下……
```

例4:[CQOI2006]凸多边形

- 逆时针给出n个凸多边形的顶点坐标，求它们交的面积。
- 例如n=2时，两个凸多边形如下图：
- 则相交部分的面积为5.233。



显然凸多边形的交就是所有按逆时针方向的多边形的边的左半平面的交集

大致的思路是，先将所有边按极角排序，维护一个双端队列，每次当队首或者队尾的边与半平面交已经无交点时，就将其踢出，保证剩下的都是与半平面交有关的边，并且对于多个平行的向量，可以将更靠左的向量保留，其他的删除，然后求出其交点后计算面积即可

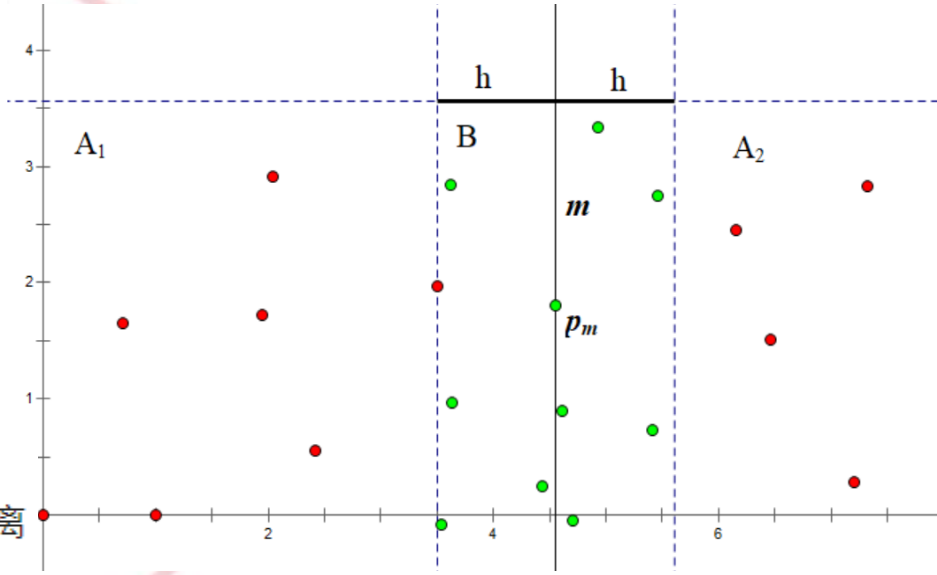
平面最近点对

- 给定个二维平面上的点，求一组欧几里得距离最近的点对。

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

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

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



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

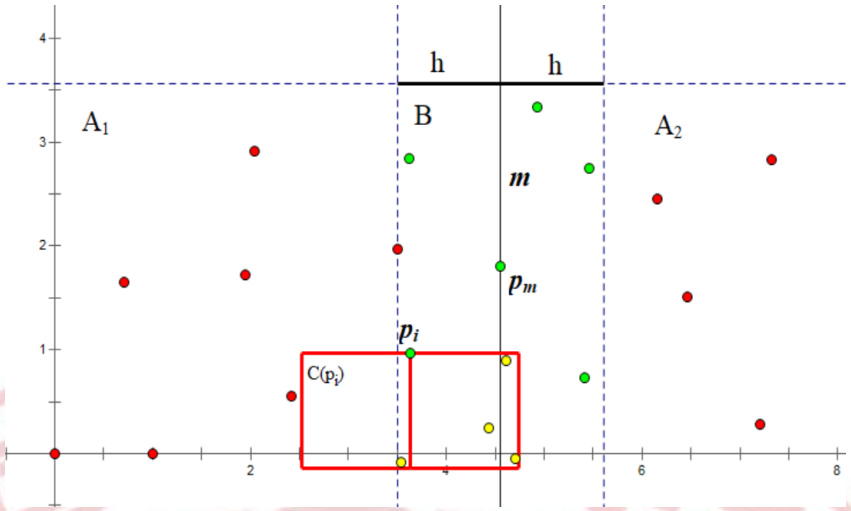
现在该合并了！我们试图找到这样的一组点对，其中一个属于 A_1，另一个属于 A_2，且二者距离小于 h。因此我们将所有横坐标与 x_m 的差小于 h 的点放入集合 B：

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

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

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

平面最近点对

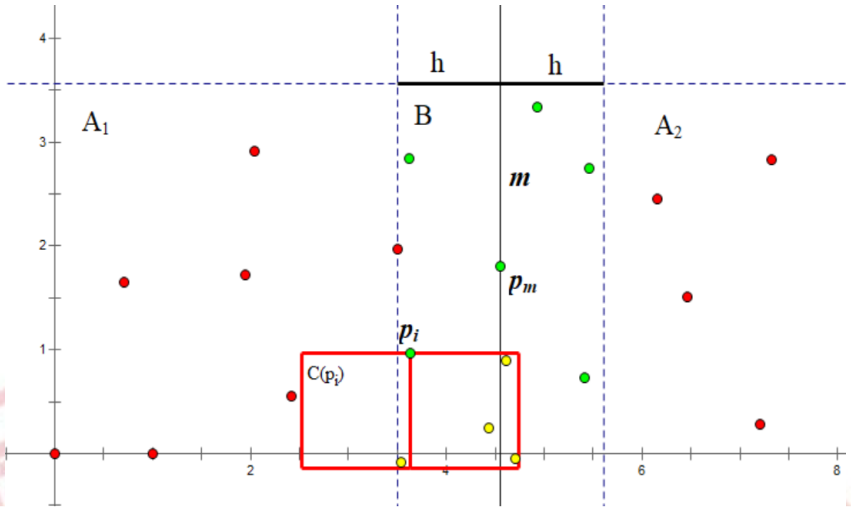


对于 B 中的每个点 p_i，我们当前目标是找到一个同样在 B 中、且到其距离小于 h 的点。为了避免两个点之间互相考虑，我们只考虑那些纵坐标小于 y_i 的点。显然对于一个合法的点 p_j，$y_i - y_j$ 必须小于 h。于是我们获得了一个集合 $C(p_i)$：

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

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

平面最近点对



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

由此我们得到了合并的步骤：

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

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

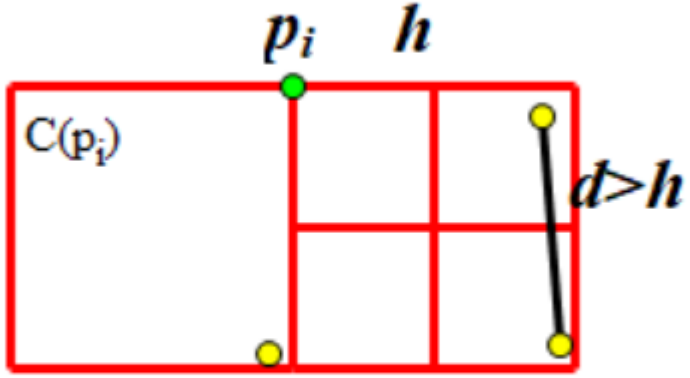
似乎这个算法仍然不优，$|C(p_i)|$ 将处于 $O(n)$ 数量级，导致总复杂度不对。其实不然，其最大大小为 7，我们给出它的证明：

复杂度证明

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

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

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



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

回顾总结

- 1. 向量和点
  - 平移，旋转，叉积，点积
  - 模长，单位向量，法向量
- 2.直线，线段
  - 点与直线的关系
  - 判断
  - 计算
- 3.距离
  - 欧式距离
  - 曼哈顿距离
  - 切比雪夫距离
- 4.多边形
  - 凸包
  - 旋转卡壳
  - 半平面交
  - 平面最近点对

谢谢各位的聆听！