

VII Selected Topics

✓ VII Selected Topics


 27 Multithreaded Algorithms


 28 Matrix Operations

 29 Linear Programming


 30 Polynomials and the FFT

 31 Number-Theoretic Algorithms

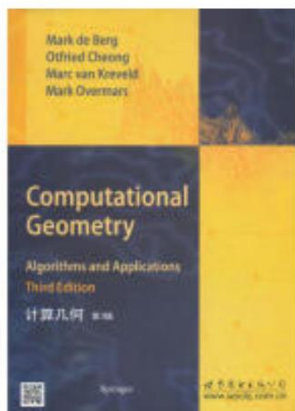
 32 String Matching

 33 Computational Geometry

 34 NP-Completeness

 35 Approximation Algorithms

33 计算几何学



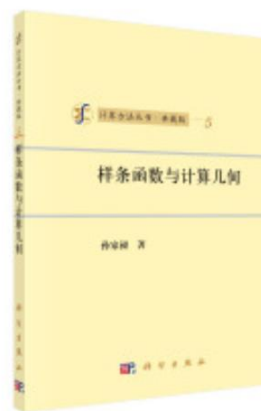
¥49.80

计算几何 第3版



¥168.00

计算几何：C语言描述英文版 2版——经典



¥61.10

样条函数与计算几何

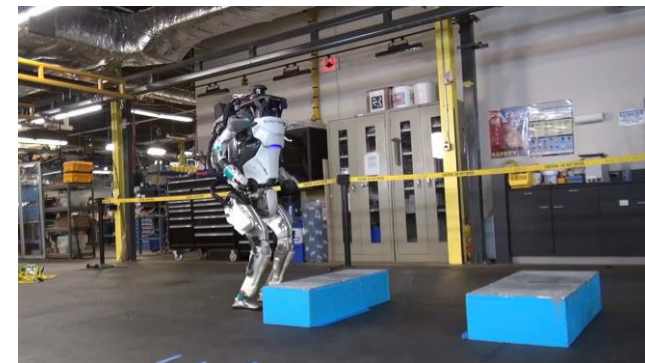
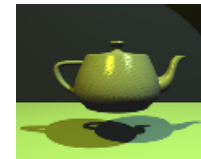


¥18.40

计算几何若干方法及其在空间数据挖掘中

33 计算几何学

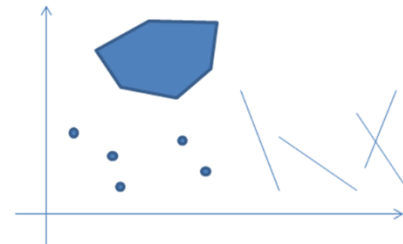
- 计算科学的一个分支，研究求解几何问题的算法
- 应用
 - 电脑绘图
 - 机器人学
 - VLSI(超大规模集成电路) 设计
 - 计算机辅助设计 (CAD)
 - ...



33 计算几何学

- 输入:对一组几何对象的描述

- 一组点
- 一组线段
- 按逆时针顺序排列的多边形顶点
- ...



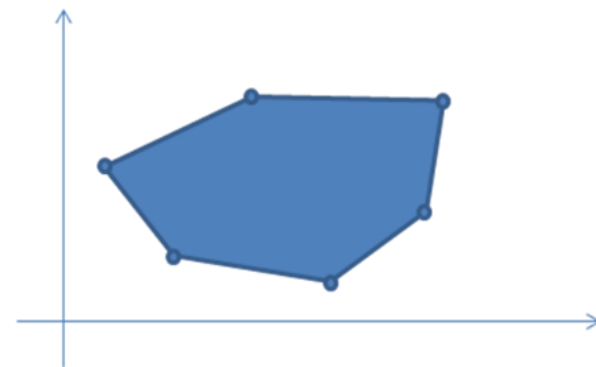
- 输出:通常是对对象查询的响应

- 是否有直线相交
- 一些新的几何对象, 如凸包(最小封闭凸多边形)
- ...



33 计算几何学

- 在这一章中，我们研究一些二维的计算几何算法，也就是在平面上。
- 每个输入对象都表示为一组点 $\{p_1, p_2, p_3, \dots\}$ ，每个点 $p_i = (x_i, y_i)$ ， $x_i, y_i \in \mathbb{R}$. 例如，一个 n - 顶点多边形 P 由一个顶点序列 $p_0, p_1, p_2, p_3, \dots, p_{n-1}$ 表示，序列按顶点在 P 的边界上的出现顺序排列。

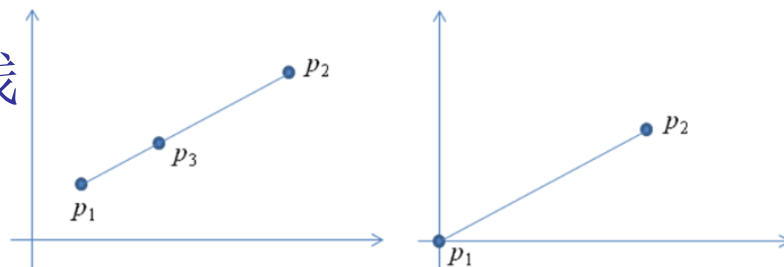


- 计算几何也可以在三维空间中执行，甚至在高维空间(数据库应用)中执行，但是这样的问题及其解决方案可能很难可视化。
- 然而，即使在二维空间中，我们也可以看到计算几何技术的一个很好的例子。

33.1 线段的属性

- 两个不同点 $p_1 = (x_1, y_1)$ 和 $p_2 = (x_2, y_2)$ 的凸组合是任何点 $p_3 = (x_3, y_3)$ 使 $0 \leq \alpha \leq 1$ 范围内的某个 α , 满足 $x_3 = \alpha x_1 + (1 - \alpha) x_2$ 及 $y_3 = \alpha y_1 + (1 - \alpha) y_2$.
我们也把这写为 $p_3 = \alpha p_1 + (1 - \alpha) p_2$.

- 直观上来说, p_3 是经过 p_1 和 p_2 的直线上的任意一点, 且在 p_1 和 p_2 之间。

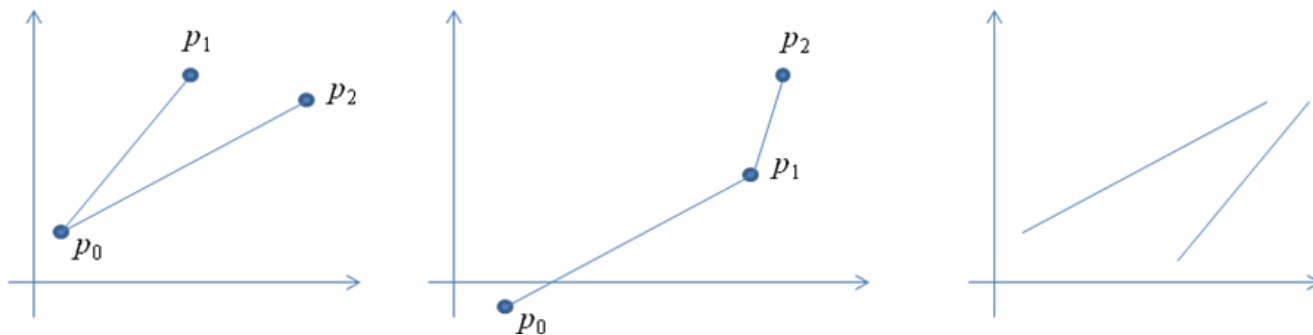


- 给定两个不同的点 p_1 和 p_2 , 线段就是 p_1 和 p_2 的凸组合集。我们称 p_1 和 p_2 为线段的端点。
- 有时 p_1 和 p_2 的顺序很重要, 我们称之为有向段。如果 p_1 是原点 $(0,0)$ 那么我们可以把有向段看作向量 p_2 。

33.1 线段的属性

Questions:

1. 给定两个有向段 $\overrightarrow{p_0p_1}$ 和 $\overrightarrow{p_0p_2}$ ，以公共端点 p_0 为圆心， $\overrightarrow{p_0p_1}$ 是在 $\overrightarrow{p_0p_2}$ 的顺时针方向吗？
2. 给定两个线段 $\overrightarrow{p_0p_1}$ 和 $\overrightarrow{p_1p_2}$ ，如果我们穿过 $\overrightarrow{p_0p_1}$ 然后是 $\overrightarrow{p_1p_2}$ ，我们是否是在点 p_1 左转？
3. 线段 $\overrightarrow{p_1p_2}$ 和 $\overrightarrow{p_3p_4}$ 是否相交？



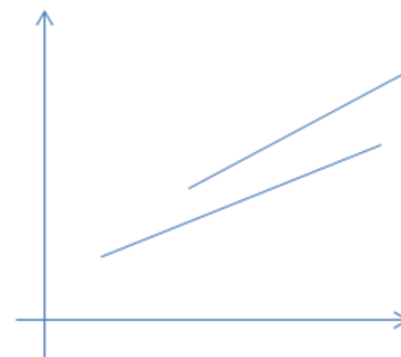
我们可以在 $O(1)$ 时间内回答每个问题。

此外，我们的方法只使用加法、减法、乘法和比较。我们既不需要除法也不需要三角函数，这两种方法的计算成本都很高，而且容易产生舍入误差的问题。

例如,.....

33.1 线段的属性

只使用加法，减法，乘法和比较。
既不使用除法也不使用三角函数。



For example,

- 确定两段是否相交的“直接”方法
 - 计算每一段 $y = mx + b$ 的直线方程(m 为斜率, b 为 y 轴截距),
 - 求直线交点(除法求交点),
 - 检查这个点是否在两个线段上。
- 当线段接近平行时, 在计算机上进行该方法对除法运算的精度非常敏感

本节中的方法避免了除法, 因此更加准确。

$$\begin{cases} y = m_1x + b_1 \\ y = m_2x + b_2 \end{cases} \Rightarrow x = \frac{b_2 - b_1}{m_1 - m_2}, y = \frac{m_1b_2 - m_2b_1}{m_1 - m_2}$$

33.1 线段的属性- 叉积

- 计算叉乘是线段法的核心。
- 考虑向量 p_1 和 p_2 ，定义 $p_1 \times p_2$ 的叉积为矩阵的行列式

$$\begin{aligned} p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1 y_2 - x_2 y_1 \\ &= -p_2 \times p_1 \end{aligned}$$

- Cross products(交叉乘积，叉积)：计算几何的核心运算
- 就像图算法中的几个核心运算：
边的松弛，BFS，（优先）队列，DFS，递归

33.1 线段的属性- 叉积

- 计算叉乘是线段法的核心。
- 考虑向量 p_1 和 p_2 ，定义 $p_1 \times p_2$ 的叉积为矩阵的行列式

$$\begin{aligned} p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1 y_2 - x_2 y_1 \\ &= -p_2 \times p_1 \end{aligned}$$

- 等价地， $p_1 \times p_2$ 可以解释为由 $(0,0)$ 、 p_1 、 p_2 和 $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$ 构成的平行四边形的有符号面积吗？见图33.1 (a)。练习:试着证明它。提示: 见图33.1.A)

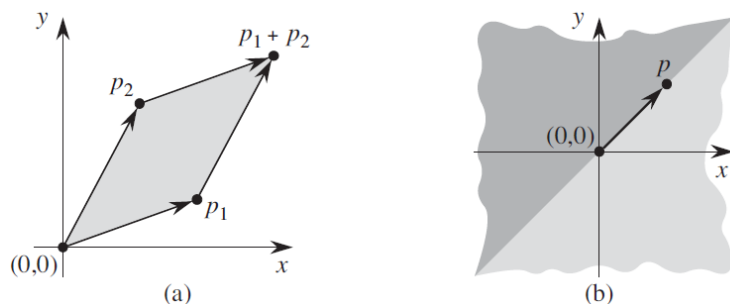
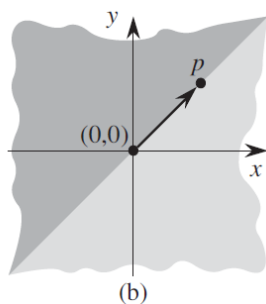
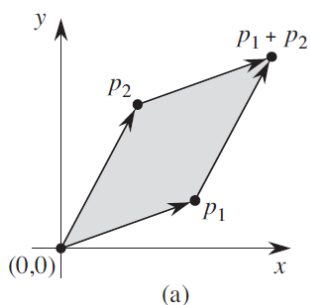


图33.1: (a) $p_1 \times p_2$ 是平行四边形的有符号面积。

(b) 浅阴影区域包含从 p 开始的顺时针方向的向量，深阴影区域包含从 p 开始的逆时针方向的向量。

33.1 线段的属性- 叉积

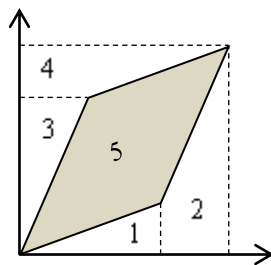
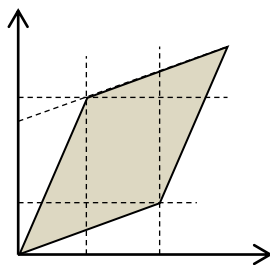
- $p_1 \times p_2$ 可以解释为由 $(0,0)$ 、 p_1 、 p_2 和 $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$ 构成的平行四边形的有符号面积。



$$\begin{aligned} p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1 y_2 - x_2 y_1 \\ &= -p_2 \times p_1 \end{aligned}$$

图33.1: (a) $p_1 \times p_2$ 是平行四边形的有符号面积。

(b) 浅阴影区域包含从 p 开始的顺时针方向的向量，深阴影区域包含从 p 开始的逆时针方向的向量。



$$5 = (1+2+3+4+5) - (1+2+3+4)$$

$$\begin{aligned} &"1" + "2" + "3" + "4" + "5" \\ &= (x_1+x_2)*(y_1+y_2) \\ &"1" = x_1*y_1/2 \\ &"3" = x_2*y_2/2 \\ &"2" = "3" + x_2*y_1 \\ &"4" = "1" + x_2*y_1 \end{aligned}$$

图33.1.A:叉乘证明提示

33.1 线段的属性- 叉积

叉积的一些特征

- 若 $p_1 \times p_2$ 是正的, 那么以原点 $(0, 0)$ 为圆心, p_1 在 p_2 的顺时针方向
- 若是负的, 那么 p_1 在 p_2 的逆时针方向。
- 图33.1(b)显示了相对于向量 p 的顺时针和逆时针区域。
- 如果 $p_1 \times p_2$ 为 0, 则出现一个边界条件, 在这种情况下, 向量共线, 指向相同或相反的方向。

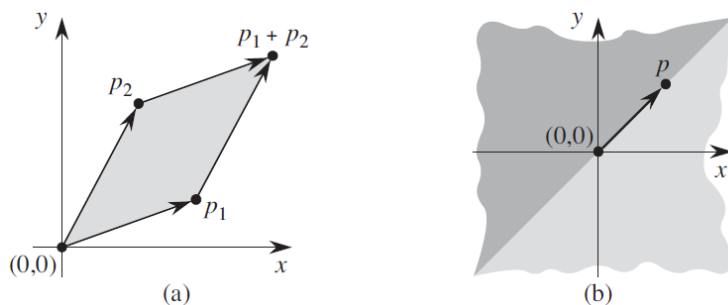


图33.1: (a) $p_1 \times p_2$ 是平行四边形的有符号面积。

(b) 浅阴影区域包含从 p 开始的顺时针方向的向量, 深阴影区域包含从 p 开始的逆时针方向的向量。

33.1 线段的属性- 叉积

1. 判断 $\overrightarrow{p_0p_1}$ 是否在 $\overrightarrow{p_0p_2}$ 的顺时针方向

- 简单地转换为用 p_0 作为原点，计算叉积

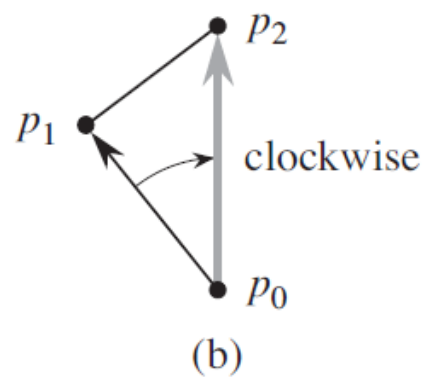
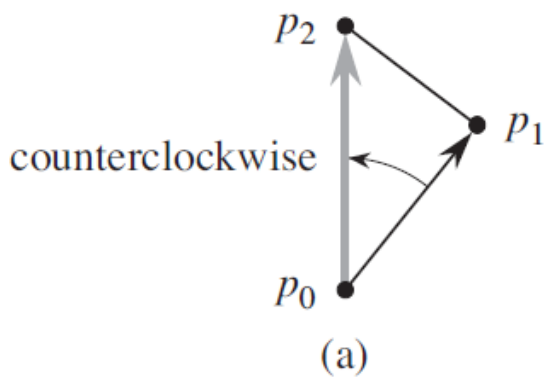
$$(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$$

- 如果是正的, 那就是顺时针方向; 否则, 就是逆时针方向。

33.1 线段的属性- 叉积

2. 确定连续的线段向左或向右转

- 两条连续线段 $\overrightarrow{p_0p_1}$ 和 $\overrightarrow{p_1p_2}$, 在点 p_1 向左转或向右转。
- 同样地, 确定给定角度 $\angle p_0p_1p_2$ 转向的方向。
- 叉乘让我们不用计算角度就能回答这个问题。我们只需检查 $\overrightarrow{p_0p_2}$ 相对于 $\overrightarrow{p_0p_1}$ 是顺时针还是逆时针(参见图33.2)。



33.1 线段的属性- 叉积

2. 确定连续的线段向左或向右转

- 计算叉积 $(p_2 - p_0) \times (p_1 - p_0)$
 - 若是负的, $\overrightarrow{p_0p_2}$ 在 $\overrightarrow{p_0p_1}$ 逆时针方向, 因此我们在 p_1 左转。
 - 正叉积表示顺时针方向和右转。
 - 0 表示点 p_0, p_1, p_2 共线。

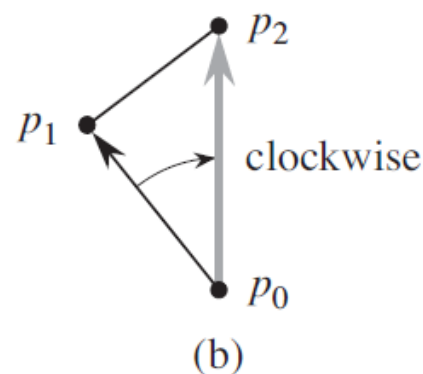
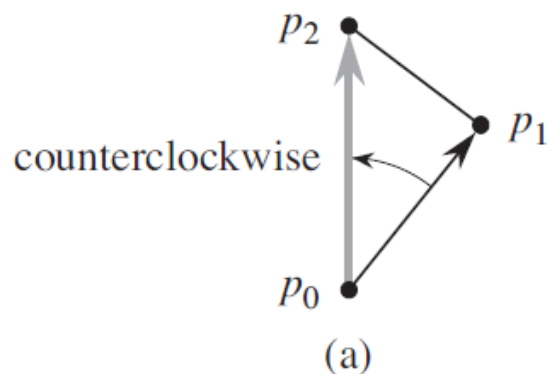


图33.2:用叉乘确定连续线段 $\overrightarrow{p_0p_1}$ 和 $\overrightarrow{p_1p_2}$ 在点 p_1 怎样转弯。我们检查 $\overrightarrow{p_0p_2}$ 相对 $\overrightarrow{p_0p_1}$ 是顺时针还是逆时针。(a)如果逆时针, 该点左转。(b)如果顺时针, 右转。

33.1 线段的属性- 叉积

3. 确定两条线段是否相交

- 通过检查每个线段是否跨越包含另一个线段的线。

相交的情况有5种：相互横跨，or某一个端点 p_i 在另一个线段上（共4个端点，4种情况）

- **横跨**：如果点 p_1 位于线的一侧，点 p_2 位于另一侧，则线段 $\overrightarrow{p_1 p_2}$ 横跨了线。
- 如果 p_1 或 p_2 直接位于直线上，则出现边界情况。

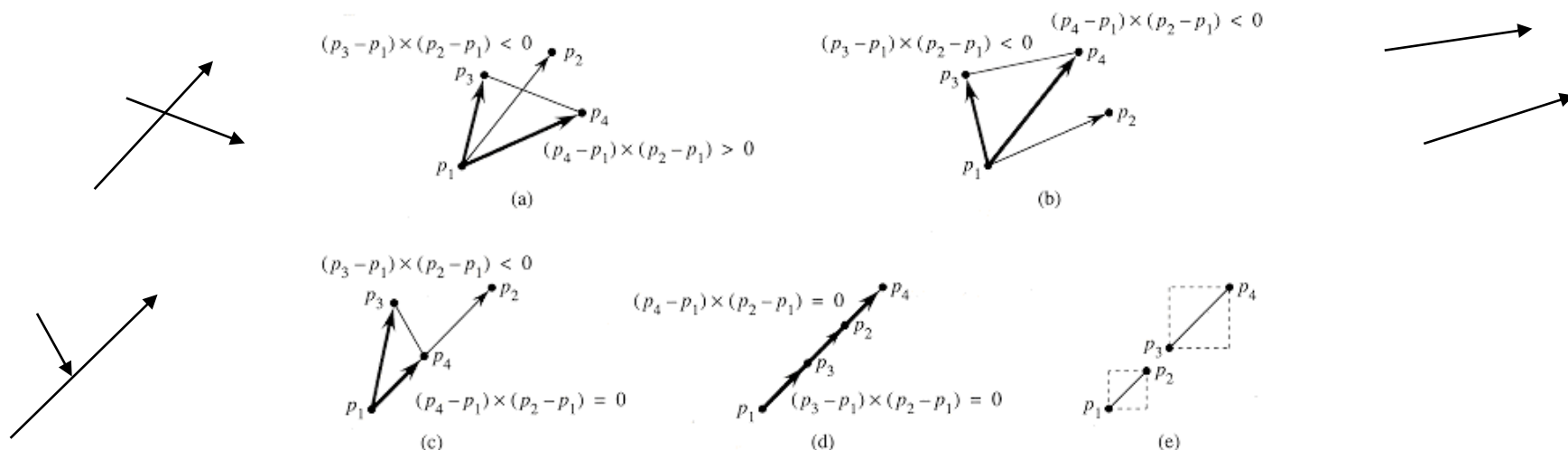


图33.3:程序 SEGMENTS-INTERSECT中的案例

33.1 线段的属性- 叉积

3. 确定两条线段是否相交

- 当且仅当满足下列任一条件时，两条线段相交：
 - ① 每一段都跨越包含另一段的线。
 - ② 一个线段的端点位于另一个线段上。（这个条件来自边界情况。）

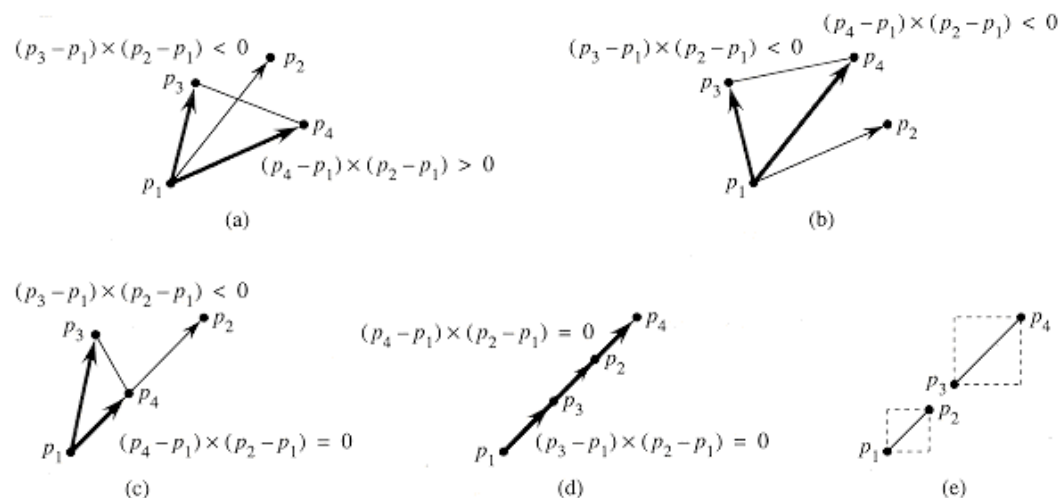


图33.3:程序 SEGMENTS-INTERSECT中的案例

33.1 线段的属性- 叉积

SEGMENTS-INTERSECT returns TRUE if segments $\overrightarrow{p_1p_2}$ and $\overrightarrow{p_3p_4}$ intersect. Return FALSE if not.

- 子程序DIRECTION:使用叉乘计算相对方向。
- ON-SEGMENT:确定一个已知的点是否与线段共线，位于该线段上。

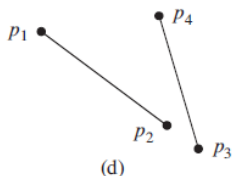
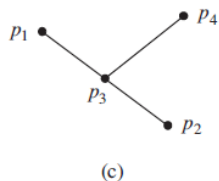
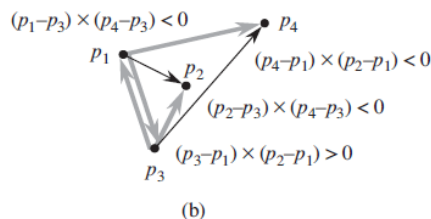
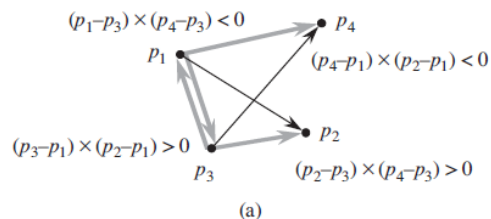


图33.3:程序 SEGMENTS-INTERSECT中的案例

```

SEGMENTS-INTERSECT( $p_1, p_2, p_3, p_4$ )
 $d_1 \leftarrow \text{DIRECTION}(p_3, p_4, p_1)$  // 若为0,  $p_1$ 与线段  $p_3p_4$  共线
 $d_2 \leftarrow \text{DIRECTION}(p_3, p_4, p_2)$ 
 $d_3 \leftarrow \text{DIRECTION}(p_1, p_2, p_3)$ 
 $d_4 \leftarrow \text{DIRECTION}(p_1, p_2, p_4)$ 
if (( $d_1 > 0$  and  $d_2 < 0$ ) or ( $d_1 < 0$  and  $d_2 > 0$ )) and
   (( $d_3 > 0$  and  $d_4 < 0$ ) or ( $d_3 < 0$  and  $d_4 > 0$ )) //  $d_1 * d_2 < 0$  and  $d_3 * d_4 < 0$ 
then return TRUE
else if  $d_1 = 0$  and ON-SEGMENT( $p_3, p_4, p_1$ )
then return TRUE
else if  $d_2 = 0$  and ON-SEGMENT( $p_3, p_4, p_2$ )
then return TRUE
else if  $d_3 = 0$  and ON-SEGMENT( $p_1, p_2, p_3$ )
then return TRUE
else if  $d_4 = 0$  and ON-SEGMENT( $p_1, p_2, p_4$ )
then return TRUE
else return FALSE
    
```

```

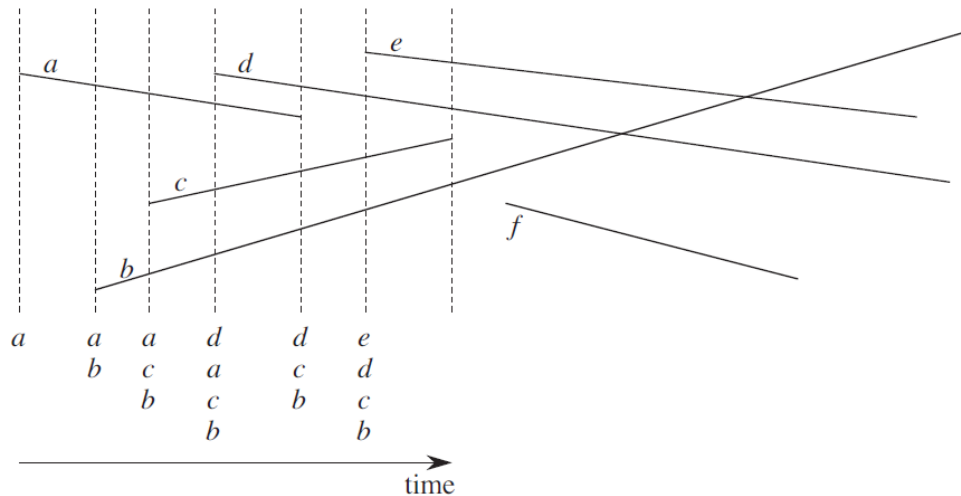
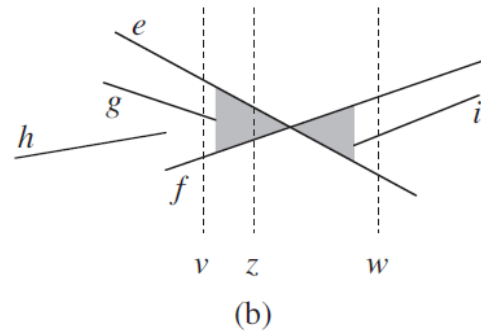
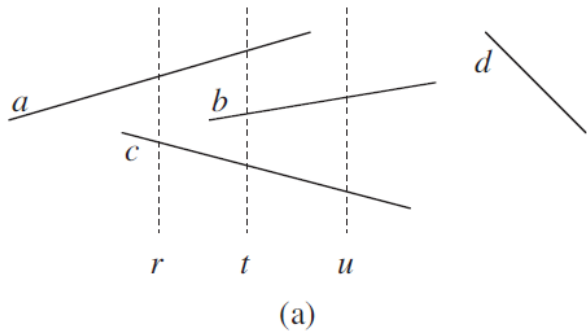
DIRECTION( $p_i, p_j, p_k$ )
return  $(p_k - p_i) \times (p_j - p_i)$ 
    
```

```

ON-SEGMENT( $p_i, p_j, p_k$ )
if  $\min(x_i, x_j) \leq x_k \leq \max(x_i, x_j)$  and
 $\min(y_i, y_j) \leq y_k \leq \max(y_i, y_j)$ 
then return TRUE
else return FALSE
    
```

*33.2 确定任何一对线段是否相交

*该算法使用了一种称为“sweeping”的技术。



- 先对线段的顶点进行排序
- 然后用sweeping技术，依序把线段加入表中或从表中删除

*33.2 确定任何一对线段是否相交

ANY-SEGMENTS-INTERSECT(S)

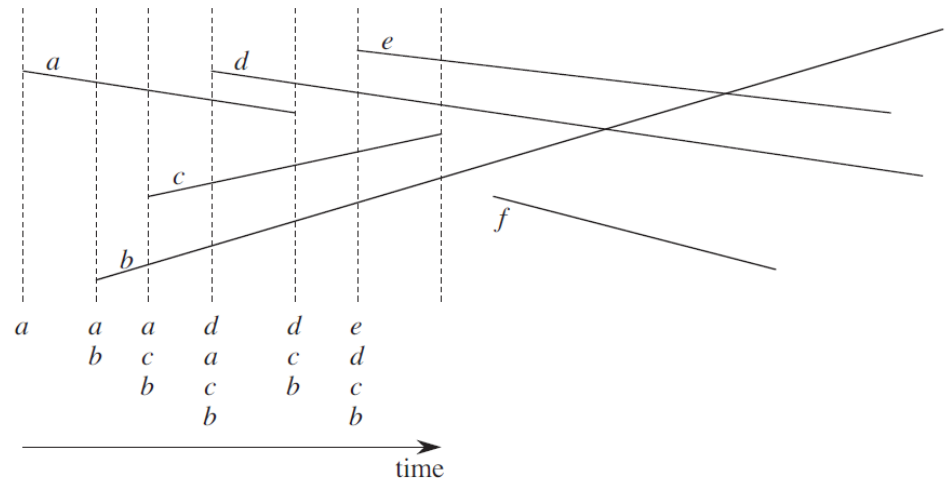
```
1   $T = \emptyset$ 
2  sort the endpoints of the segments in  $S$  from left to right,
    breaking ties by putting left endpoints before right endpoints
    and breaking further ties by putting points with lower
    y-coordinates first
3  for each point  $p$  in the sorted list of endpoints
4      if  $p$  is the left endpoint of a segment  $s$ 
5          INSERT( $T, s$ )
6          if (ABOVE( $T, s$ ) exists and intersects  $s$ )
              or (BELOW( $T, s$ ) exists and intersects  $s$ )
7              return TRUE
8      if  $p$  is the right endpoint of a segment  $s$ 
9          if both ABOVE( $T, s$ ) and BELOW( $T, s$ ) exist
              and ABOVE( $T, s$ ) intersects BELOW( $T, s$ )
10         return TRUE
11     DELETE( $T, s$ )
12 return FALSE
```

- 先对线段（顶点）进行排序, $O(n \lg n)$

- 然后sweeping, $O(n)$

- 书上: 红黑树, $O(\lg n)$

Running Time?



*33.2 确定任何一对线段是否相交

ANY-SEGMENTS-INTERSECT(S)

```
1   $T = \emptyset$ 
2  sort the endpoints of the segments in  $S$  from left to right,
   breaking ties by putting left endpoints before right endpoints
   and breaking further ties by putting points with lower
   y-coordinates first
3  for each point  $p$  in the sorted list of endpoints
4      if  $p$  is the left endpoint of a segment  $s$ 
5          INSERT( $T, s$ )
6          if (ABOVE( $T, s$ ) exists and intersects  $s$ )
              or (BELOW( $T, s$ ) exists and intersects  $s$ )
7              return TRUE
8      if  $p$  is the right endpoint of a segment  $s$ 
9          if both ABOVE( $T, s$ ) and BELOW( $T, s$ ) exist
              and ABOVE( $T, s$ ) intersects BELOW( $T, s$ )
10         return TRUE
11     DELETE( $T, s$ )
12 return FALSE
```

Correctness?

定理：算法正确（找到交点时返回true），当且仅当(if and only if) 有交点

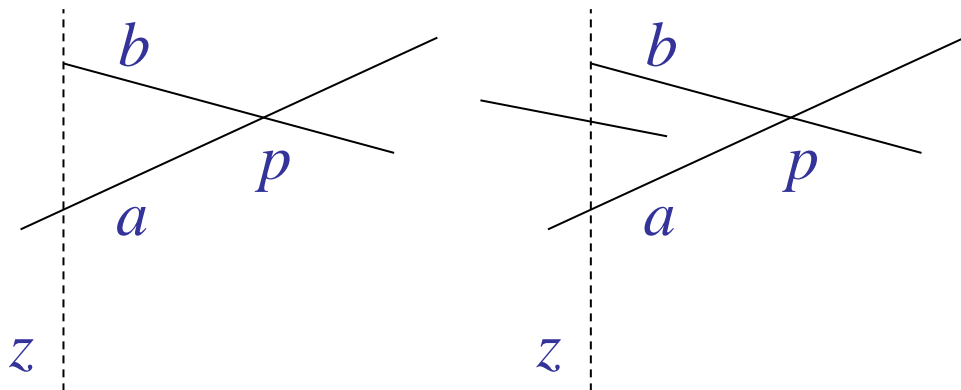
思路：

\Rightarrow ：算法返回true，有交点。

显然。

\Leftarrow ：有交点，一定会被判断出来（会被发现），因此，算法返回true

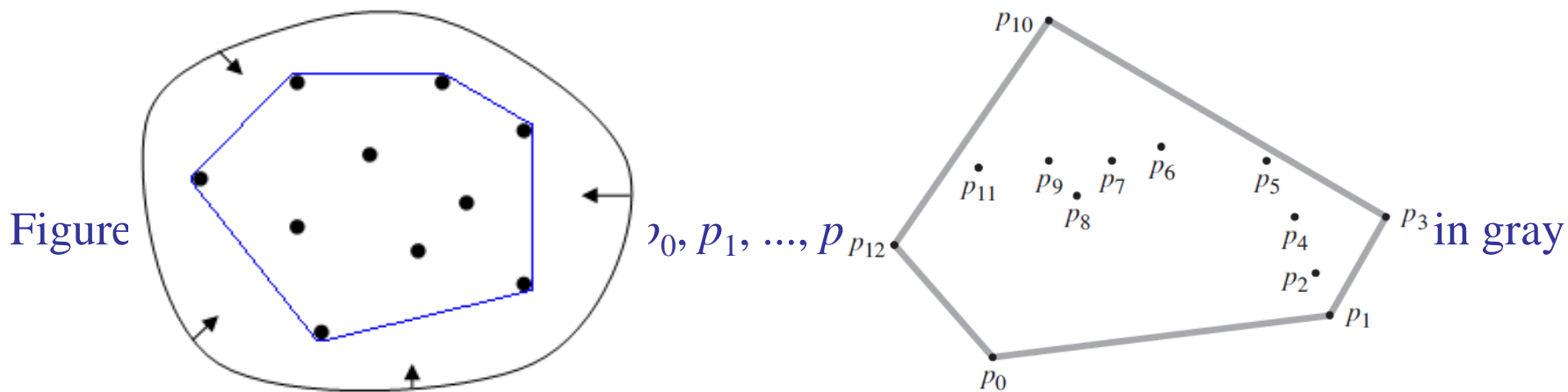
设只有一个交点 p



33.3 求凸包

$\text{CH}(Q)$, 是点集 Q 的凸包, 它是最小的凸多边形 P , 其中 Q 中的每个点要么在 P 的边界上, 要么在 P 的内部。

直观上, 我们可以把 Q 中的每个点想象成从木板上伸出来的钉子。凸包是由一个紧密的橡皮筋包围所有钉子形成的形状(见图33.6)。

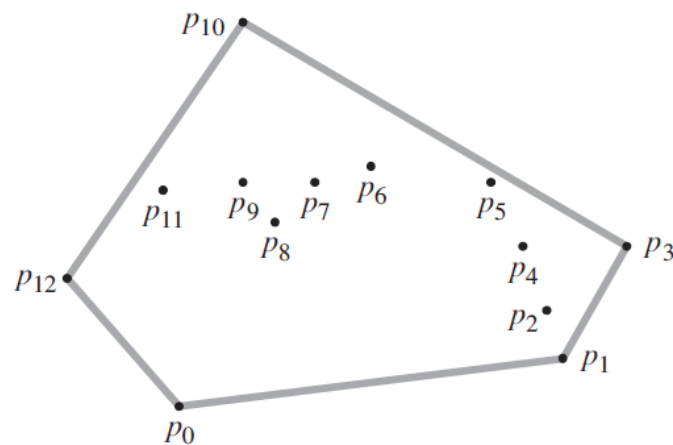


33.3 求凸包

计算一组点的凸包本身就是一个有趣的问题。

此外，其他一些计算几何问题的算法都是从计算凸包开始的。例如，二维的最远对问题(Exer 33.3-3)。

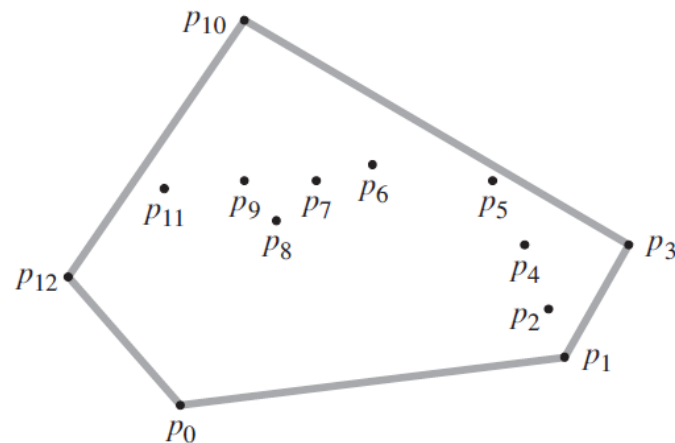
(Exercises: 33.3-3 给定点集合 Q ，证明彼此相距最远的点对一定是 $CH(Q)$ 的顶点)



33.3 求凸包

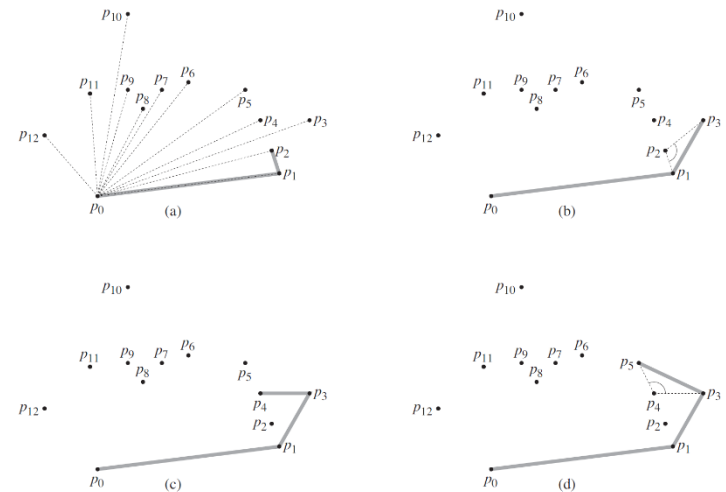
一些计算 n 个点集合的凸包的算法:

- Graham's scan, 时间复杂度 $O(n \lg n)$
- Jarvis's march, 时间复杂度 $O(nh)$, 其中 h 为凸包的顶点数
- 其他几种方法
 - 增量法, $O(n \lg n)$
 - 分治法, $O(n \lg n)$
 - 剪枝搜索, $O(n \lg h)$



33.3求凸包--Graham's scan

Information Processing Letters,
1(4): 132-133, 1972



[引用] An efficient algorithm for determining the convex hull of a finite planar set

RL Graham - Info. Pro. Lett., 1972 - ci.nii.ac.jp

CiNii 論文 - An efficient algorithm for determining the convex hull of a finite planar set CiNii 国立
情報学研究所 学術情報ナビゲータ[サイニィ] 日本の論文をさがす 大学図書館の本をさがす 日本の
博士論文をさがす 新規登録 ログイン English 検索 すべて 本文あり すべて 本文あり 閉じる タイトル
著者名 著者ID 著者所属 刊行物名 ISSN 巻号ページ 出版者 参考文献 出版年 年から 年まで 検索
検索 検索 CiNii窓口業務の再開について An efficient algorithm for determining the convex hull of
a finite planar set GRAHAM RL 被引用文献: 5件 著者 GRAHAM RL 収録刊行物 Info. Pro. Lett ...

☆ 保存 ㊟ 引用 被引用次数: 2330 相关文章 所有 3 个版本 ㊟

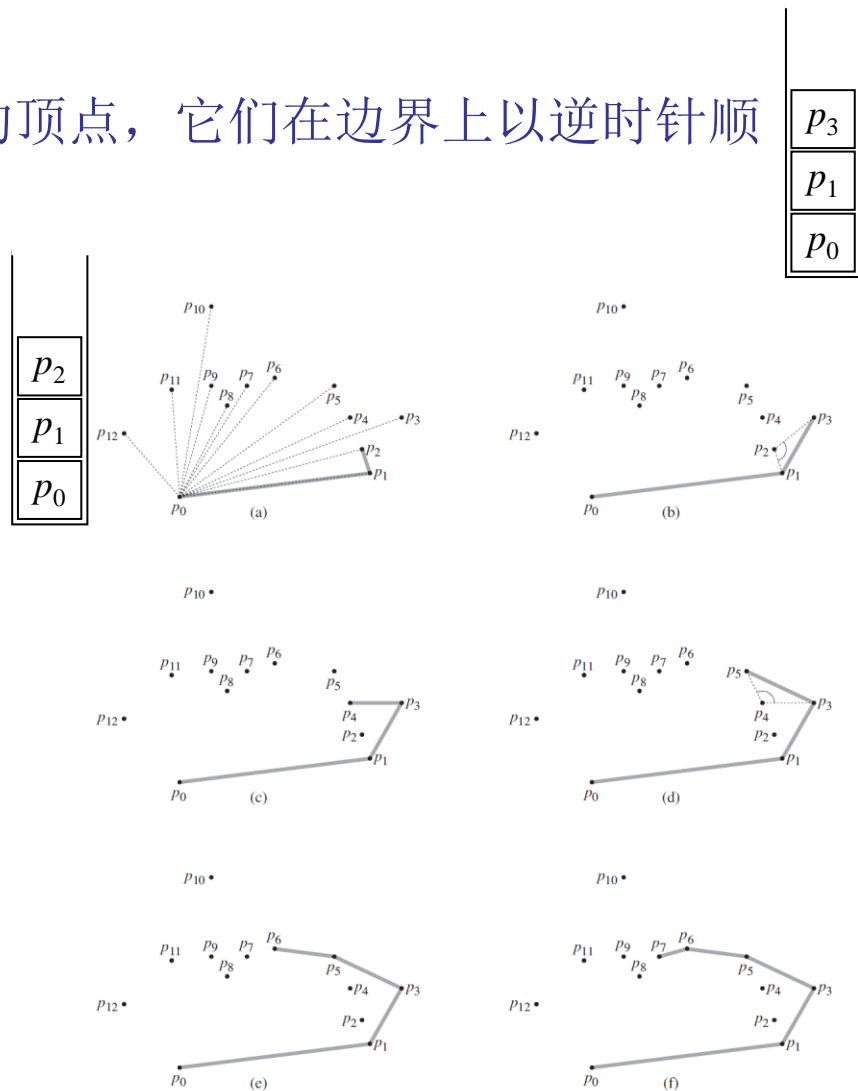
33.3 求凸包--Graham's scan

通过保持候选点的栈S，令连续线段向左或向右转

- 输入集 Q 的每个点都被推入堆栈一次。
- 不是CH(Q)顶点的点最终从堆栈中弹出
- 当算法终止时，堆栈S正好包含CH(Q)的顶点，它们在边界上以逆时针顺序出现。

GRAHAM-SCAN(Q)

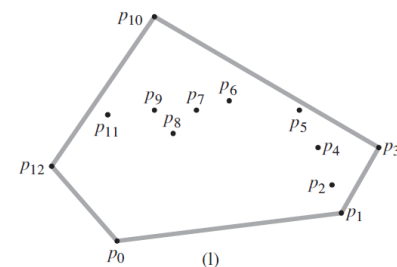
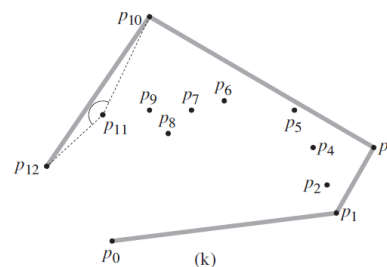
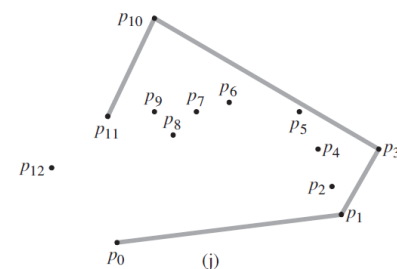
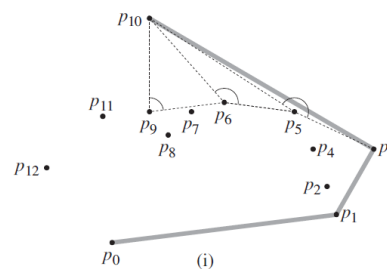
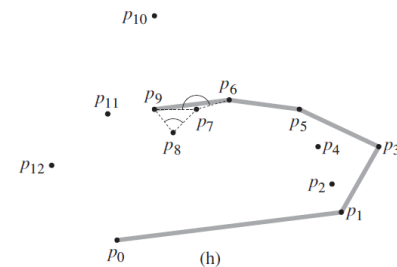
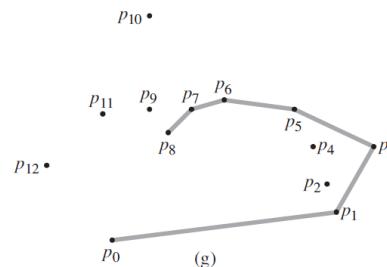
- 1 设 p_0 是 Q 中y坐标最小的点，或者是最左边的点
- 2 设 $\langle p_1, p_2, \dots, p_m \rangle$ 是 Q 中剩下的点，围绕 p_0 按极角逆时针顺序排序(如果有多个点具有相同的角度，留下离 p_0 最远的那个点)
- 3 PUSH(p_0, S)
- 4 PUSH(p_1, S)
- 5 PUSH(p_2, S)
- 6 for $i \leftarrow 3$ to m
- 7 while (由NEXT-TO-TOP(S)、TOP(S)和 p_i 组成的连续线段不向左转弯) // 直线，栈顶点在凸包边上；右转，点在凸包内
- 8 POP(S)
- 9 PUSH(p_i, S) // p_i 可能为凸包顶点，入栈
- 10 return S



33.3 求凸包--Graham's scan

GRAHAM-SCAN(Q)

- 1 设 p_0 是 Q 中 y 坐标最小的点, 或者是最左边的点
- 2 设 $\langle p_1, p_2, \dots, p_m \rangle$ 是 Q 中剩下的点, 围绕 p_0 按极角逆时针顺序排序(如果有多个点具有相同的角度, 留下离 p_0 最远的那个点)
- 3 PUSH(p_0, S)
- 4 PUSH(p_1, S)
- 5 PUSH(p_2, S)
- 6 **for** $i \leftarrow 3$ **to** m
- 7 **while** (由NEXT-TO-TOP(S)、TOP(S)和 p_i 组成的连续线段不向左转弯) // 直线, 栈顶点在凸包边上; 右转, 点在凸包内
- 8 POP(S)
- 9 PUSH(p_i, S) // p_i 可能为凸包顶点, 入栈
- 10 **return** S



33.3 求凸包--Graham's scan

GRAHAM-SCAN(Q)

- 1 设 p_0 是 Q 中 y 坐标最小的点, 或者是最左边的点
- 2 设 $\langle p_1, p_2, \dots, p_m \rangle$ 是 Q 中剩下的点, 围绕 p_0 按极角逆时针顺序排序(如果有多个点具有相同的角度, 留下离 p_0 最远的那个点)
- 3 PUSH(p_0, S)
- 4 PUSH(p_1, S)
- 5 PUSH(p_2, S)
- 6 for $i \leftarrow 3$ to m
- 7 while (由NEXT-TO-TOP(S)、TOP(S)和 p_i 组成的连续线段不向左转弯) // 直线, 栈顶点在凸包边上; 右转, 点在凸包内
- 8 POP(S)
- 9 PUSH(p_i, S) // p_i 可能为凸包顶点, 入栈
- 10 return S

$T(n) = ?$

$\Theta(n)$

$O(n \lg n)$, 用归并排序和叉积法比较角度。

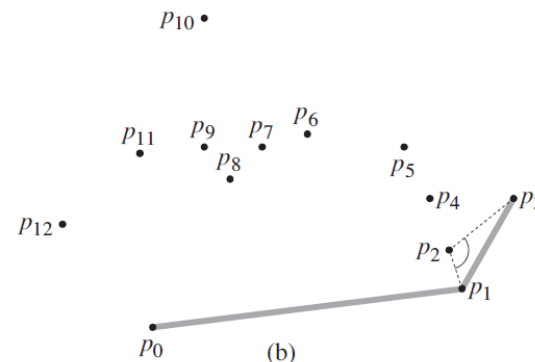
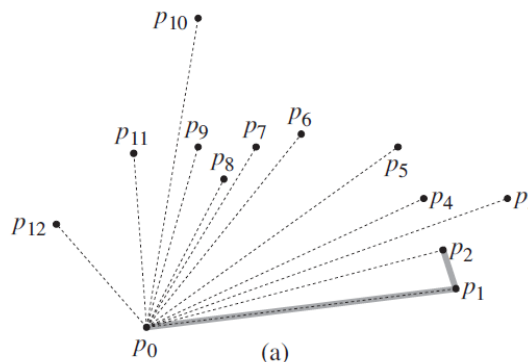
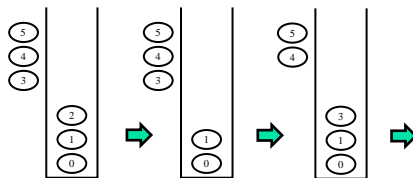
$O(1)$

$O(1)$

$O(1)$

$O(n-3)$

Aggregate analysis: while循环总共花费 $O(n)$ 时间。对于 $i = 0, 1, \dots, m$, 每个点 p_i 只被推入栈 S 一次, 每个PUSH操作最多有一个POP操作。至少有三个点 p_0, p_1 和 p_m 从未从堆栈中弹出, 因此实际上总共执行最多 $(m - 2)$ 个POP操作。

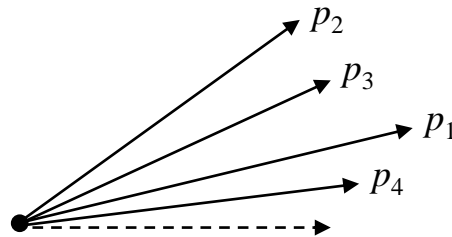


33.3 Finding the convex hull--Graham's scan

GRAHAM-SCAN(Q)

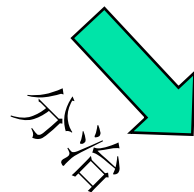
- 1 let p_0 be the point in Q with the minimum y -coordinate, or the leftmost such point in case of a tie
- 2 let $\langle p_1, p_2, \dots, p_m \rangle$ be the remaining points in Q ,
sorted by polar angle in counterclockwise order around p_0 (if more than one point has the same angle, remove all but the one that is farthest from p_0)

$O(n \lg n)$, using merge sort and the cross-product method to compare angles ?

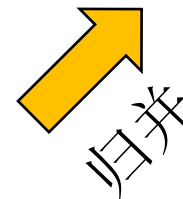


Input: p_1, p_2, p_3, p_4

按极角序
Output: $p_4 < p_1 < p_3 < p_2$



$p_1, p_2 \Rightarrow p_1 < p_2$
 $p_3, p_4 \Rightarrow p_4 < p_3$



33.3 Finding the convex hull--Jarvis's march

Information Processing Letters,
2(1): 18-21, 1973



[引用] On the identification of the convex hull of a finite set of points in the plane

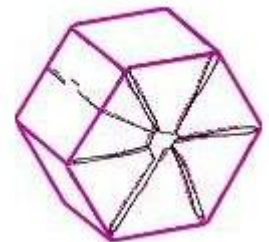
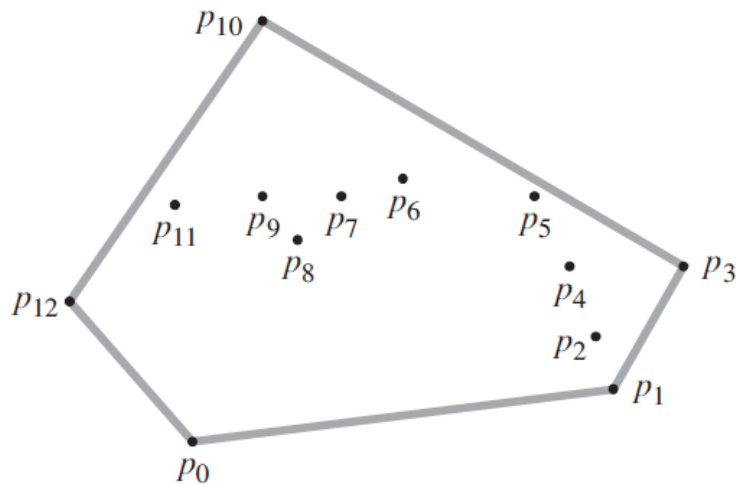
RA Jarvis - Information processing letters, 1973 - Elsevier

In most cases far less than $n(m+1)$ operations are necessary because of a powerful point deletion mechanism that can easily be included. The operations are themselves trivial (computationally inexpensive) and consist of angle comparisons only. Even these angle comparisons need not be actually carried out if an improvement suggested in a later section is implemented. Although Graham's algorithm [1] requires $O(n \log n)$ operations, the operations are themselves more complicated than those of the ...

☆ 保存 引用 被引用次数: 890 相关文章 所有 4 个版本

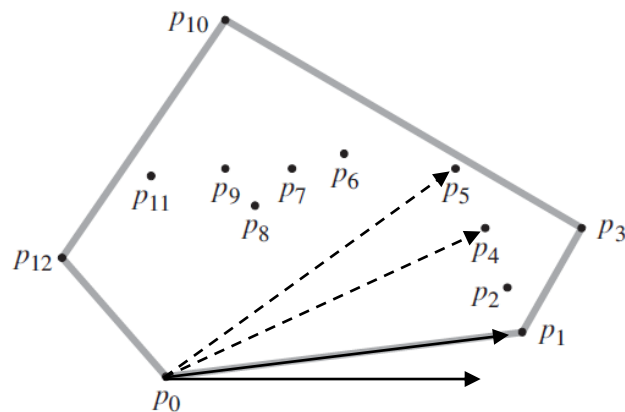
33.3 求凸包--Jarvis's march

- *Jarvis's march* 通过一种被称为包裹包装(或礼品包装)的技术计算了一组点Q的凸包。
- 算法运行时间为 $O(nh)$ ，其中 h 为 $CH(Q)$ 的顶点数。
- 是否Jarvis's march 渐进地快于运行时间为 $O(n \lg n)$ 的Graham's scan?



33.3 求凸包--Jarvis's march

- 直观来说, Jarvis's march 就像在集合Q周围裹上一张绷紧的纸
 - 我们先把纸的末端粘到集合的最低点, 也就是Graham's scan开始时的 p_0 , 这个点是凸包的顶点。
 - 我们把纸向右拉, 使它绷紧, 然后我们把它拉得更高, 直到它碰到一个点。这个点也必定是凸包的顶点。
 - 保持纸张绷紧, 我们继续这样绕着顶点集, 直到我们回到原点 p_0 。
- Jarvis's march 运行时间为 $O(nh)$?



33.3 求凸包--Jarvis's march

- 直观来说, Jarvis's march 就像在集合Q周围裹上一张绷紧的纸
- Jarvis's march 运行时间为 $O(nh)$?

p_0 为端点: $p_0p'_1 p_0p'_2 \dots p_0p'_n$, 找出极坐标角最小的点 p_1 (CH point) $O(n)$

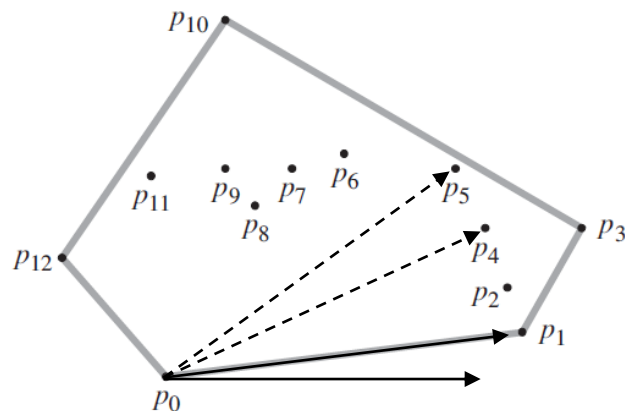
p_1 为端点: $p_1p'_1 p_1p'_2 \dots p_1p'_n$, 找出极坐标角最小的点 p_i (CH point) $O(n)$

...

共 h 个CH point

如何求极坐标角最小?

叉积



33.3 Finding the convex hull--Jarvis's march

- Intuitively, Jarvis's march simulates wrapping a taut piece of paper around the set Q .
- Jarvis's march has a running time of $O(nh)$?

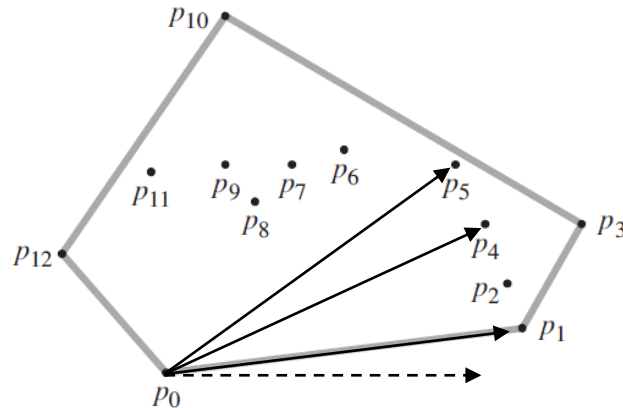
p_0 为端点: $p_0p'_1$ $p_0p'_2$... $p_0p'_n$, 找出极坐标角最小的点 p_1 (CH point) $O(n)$
 p_1 为端点: $p_1p'_1$ $p_1p'_2$... $p_1p'_n$, 找出极坐标角最小的点 p_i (CH point) $O(n)$

...

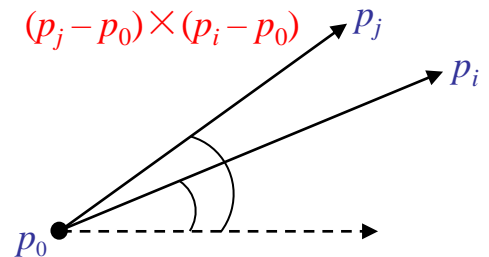
共 h 个CH point

如何求极坐标角最小?

叉积

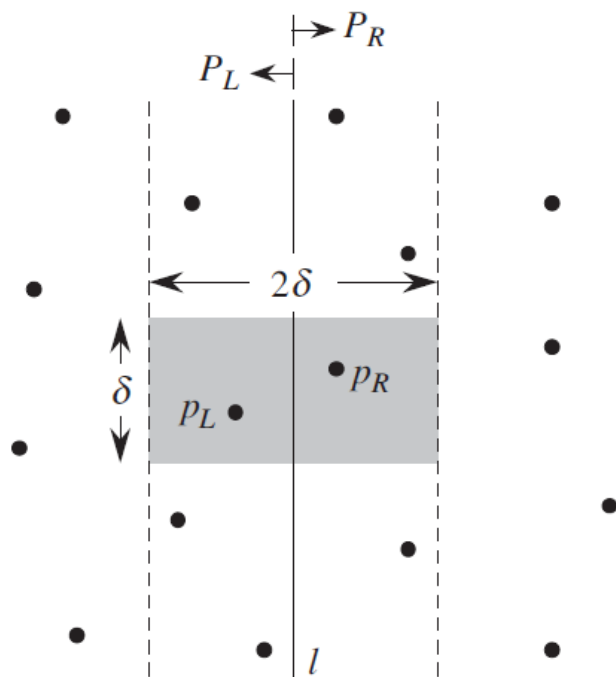


p_0p_j 和 p_0p_i , 哪个向量的极坐标角小 (p_0 为原点)?

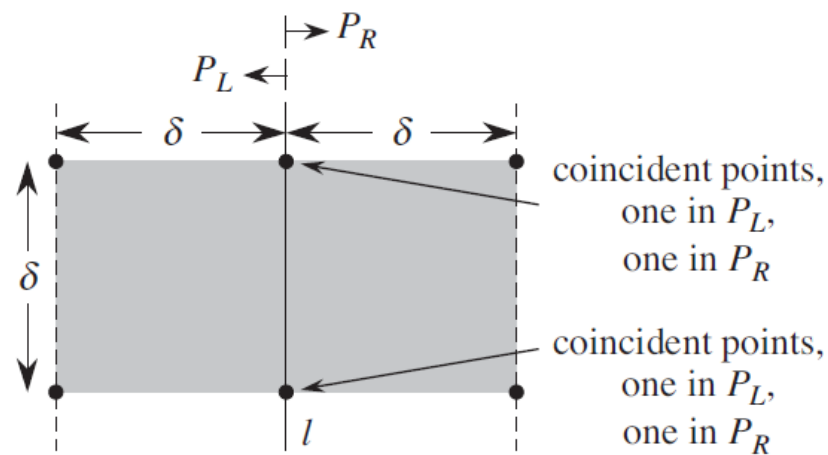


*33.4 找到最近点对

分治



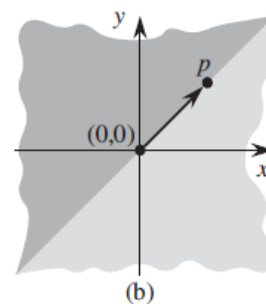
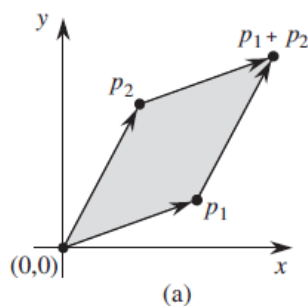
(a)



(b)

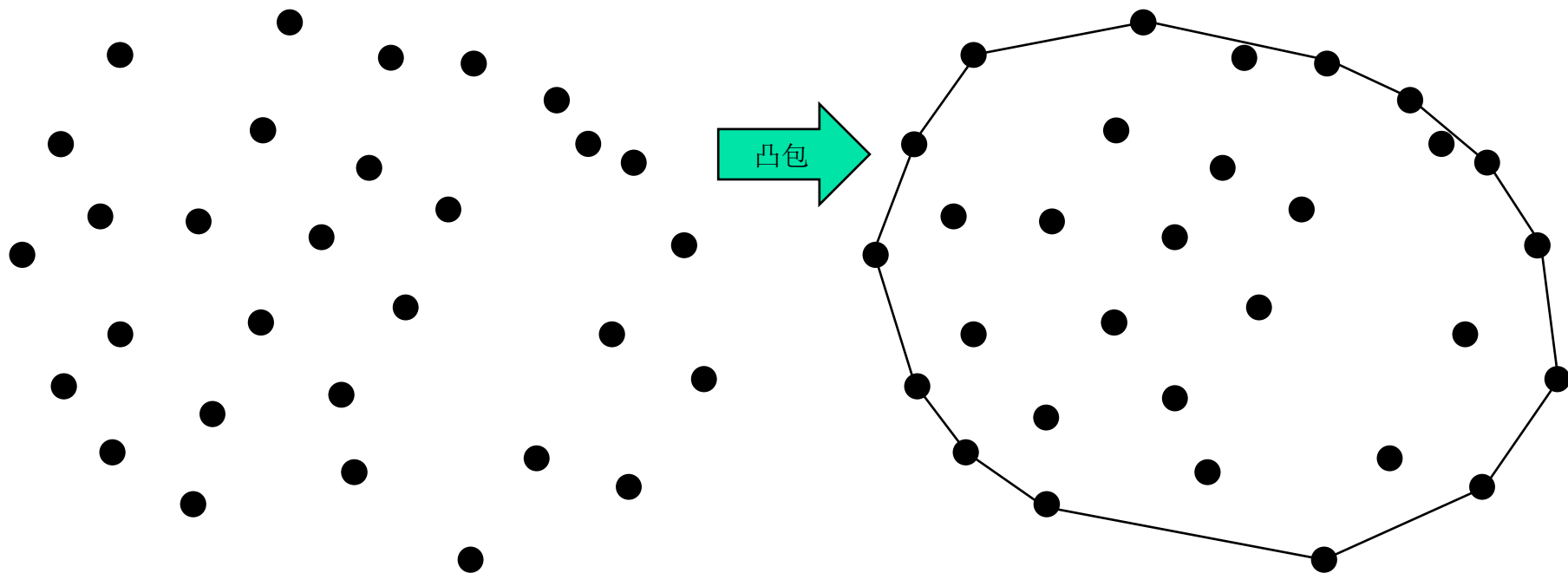
Exercises

$p_1 \times p_2$ 可以解释为由 $(0, 0)$, p_1 , p_2 , 和 $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$ 四个点组成的平行四边形的有符号面积吗?



Exercise 2

对类似下面这种特征的点集（左图）求凸包，从理论上讲，Jarvis's march 和 Graham's scan 哪个更快？

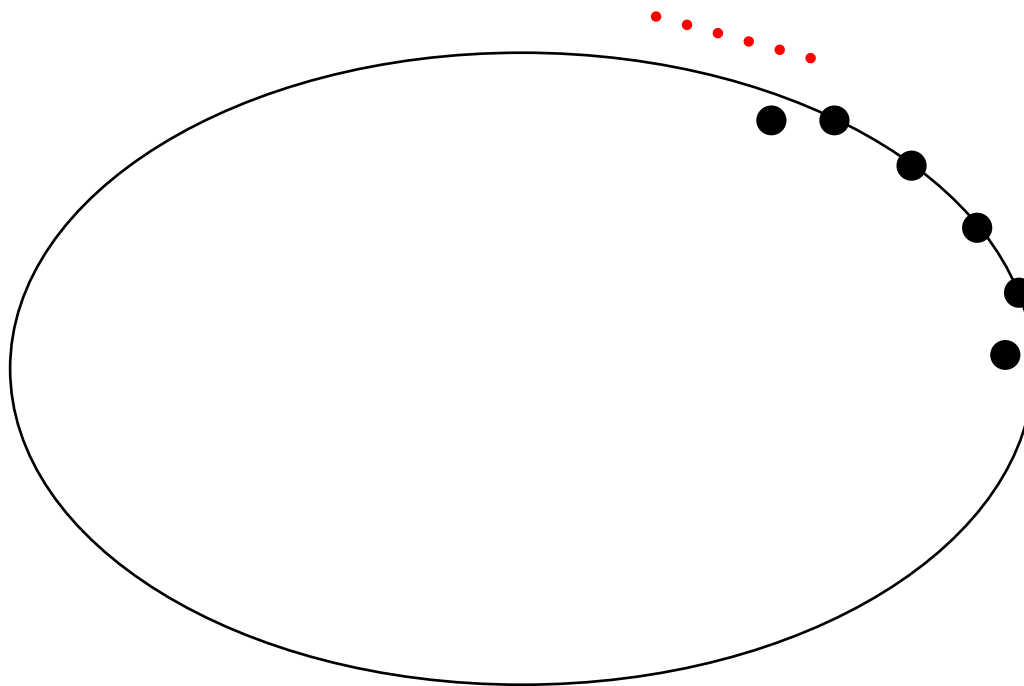


Q: 30个点

CQ: $h = 14$ 个顶点

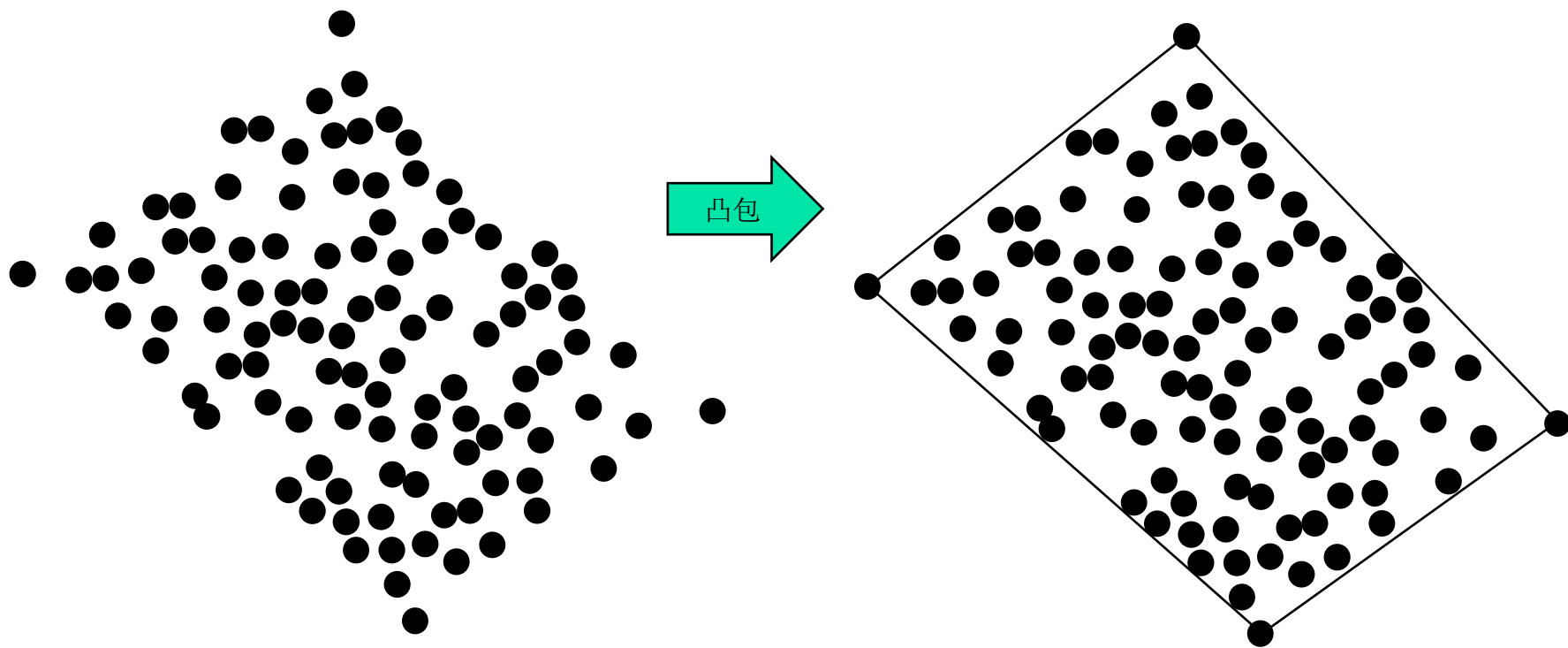
Exercise 3

这个呢？（沿着椭圆周长上随机产生一系列的点，内部有极少量零星的点）



Exercise 4

对类似下面这种特征的点集（左图）求凸包，从理论上讲，Jarvis's march 和 Graham's scan 哪个更快？



Q: #个点？

CQ: $h = 4$ 个顶点