Chapter 33

Computational Geometry

songyou@buaa.edu.cn

截止目前，本课已经学过的部分算法（设计方法）：

1暴力搜索 → 2分治 → 3递归 → 4随机算法

↓

8动规 ← 7顺序统计 ← 6优先队列 ← 5堆排序

↓

9贪心 → 10图算法（搜索、递归、回溯、边松弛、DP、贪心）

↓

11网络流（图算法应用） → 12计算几何 …

VII Selected Topics

33 Computational Geometry

计算几何本身是一个浩瀚的学科，希望本节课能为你打开对计算几何算法认知的一个小门缝，从此热爱计算几何算法。

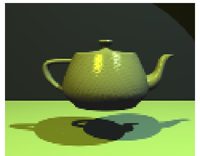
33 Computational Geometry

- A branch of CS (Computing Science) that studies algorithms for solving geometric problems
  计算几何是计算科学领域的重要分支，其专注于研究求解几何问题的算法

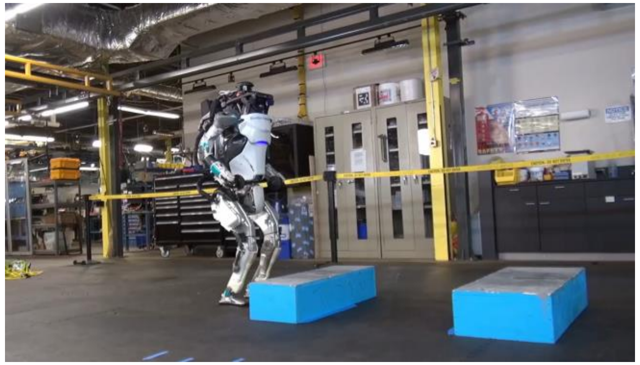
- Applications
  - Computer graphics
  - Robotics
  - VLSI(very large-scale integration) design
  - Computer- aided design

  …



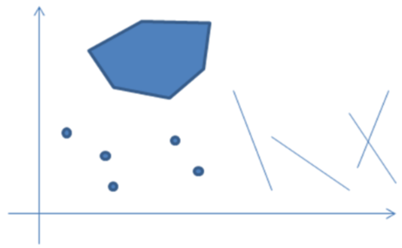


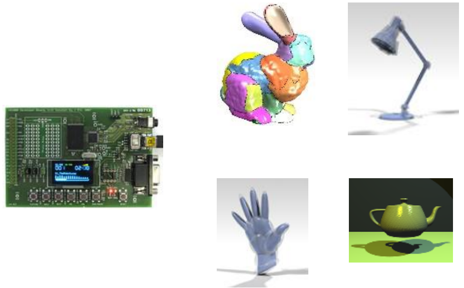


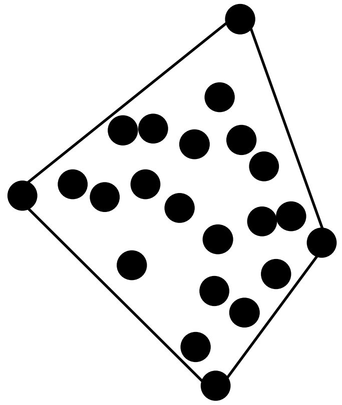
33 Computational Geometry

- Input: a description of a set of geometric objects
  - a set of points（点集）
  - a set of line segments （线段集）
  - the vertices of a polygon in counterclockwise order
    多边形上以逆时针方向的若干个顶点
  - …





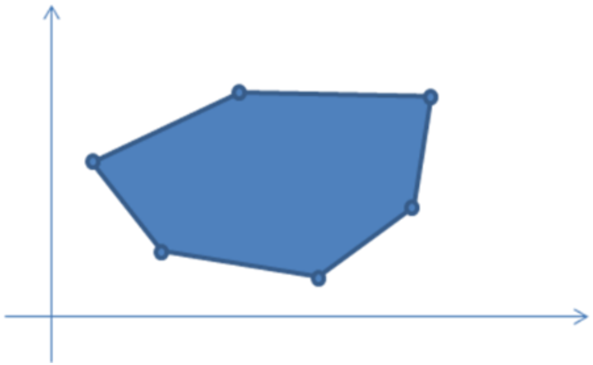
- Output: is often a response to a query about the objects
  - whether any of the lines intersect （线段是否相交）
  - some new geometric object, such as the convex hull (smallest enclosing convex polygon) …
    一些新的几何对象，如凸包等



33 Computational Geometry

- In this chapter, we look at a few computational-geometry algorithms in **two dimensions**, that is, in the plane.

- Each input object is represented as a set of points $\{p_1, p_2, p_3, \ldots\}$, where each $p_i = (x_i, y_i)$ and $x_i, y_i \in$ R. For example, an n-vertex polygon P is represented by a sequence $p_0, p_1, p_2, p_3, \ldots, p_{n-1}$ of its vertices in order of their appearance on the boundary of P.

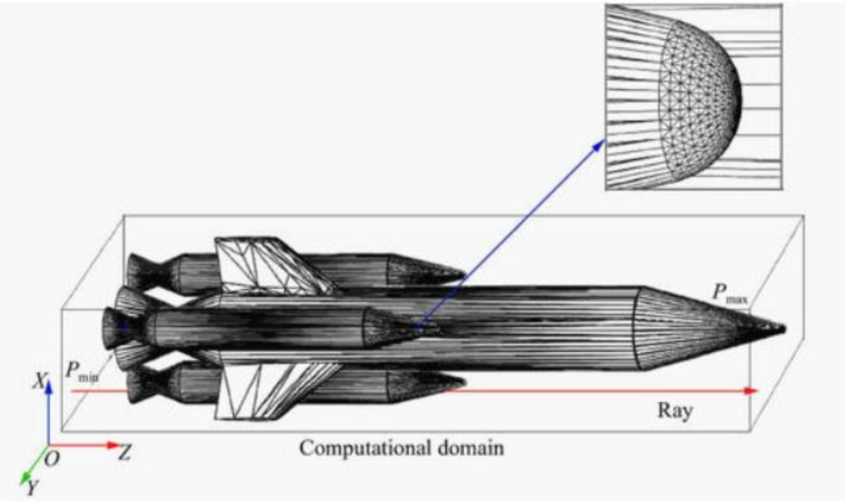


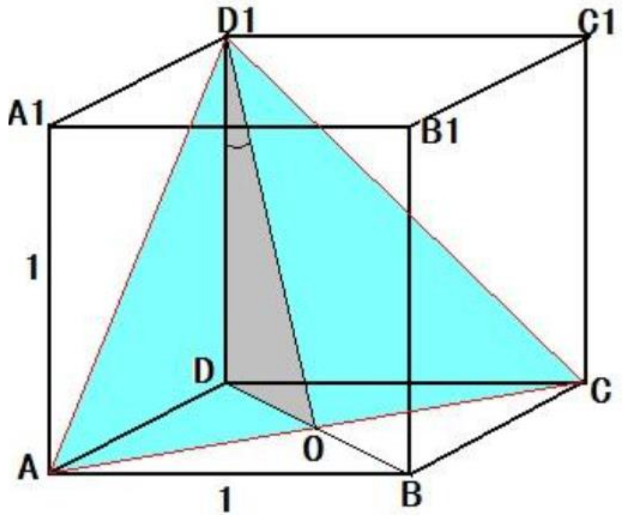
输入为点的集合

33 Computational Geometry

- Computational geometry can also be performed in three dimensions, and even in higher-dimensional spaces (an application to database), but such problems and their solutions can be very difficult to visualize.
- Even in two dimensions, however, we can see a good sample of computational-geometry techniques.

三维或高维情况的几何算法，跟二维情况一样，但三维问题的可视化很困难。



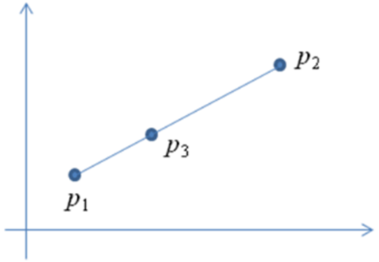


33.1 Line-segment properties

- A ***convex combination***(凸组合) of two distinct points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is any point $p_3 = (x_3, y_3)$ such that for some α in the range $0 \le \alpha \le 1$, we have

  $x_3 = \alpha x_1 + (1 - \alpha) x_2$ and $y_3 = \alpha y_1 + (1 - \alpha) y_2$.

  We also write that $p_3 = \alpha p_1 + (1 - \alpha) p_2$.



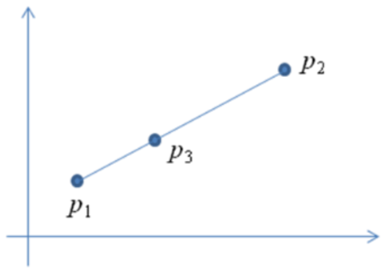
- p_3 is any point that is on the line passing through p_1 and p_2 and is on or between p_1 and p_2 on the line.

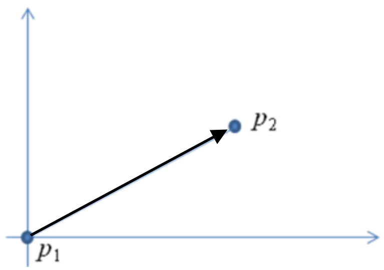
两个点 p_1 和 p_2 的凸组合 p_3 是线段 p_1p_2 上的任意一个点（含端点）。

33.1 Line-segment properties

- A ***convex combination***(凸组合) of $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$:
  $p_3 = \alpha p_1 + (1 - \alpha) p_2$, ($x_3 = \alpha x_1 + (1-\alpha) x_2$ and $y_3 = \alpha y_1 + (1 - \alpha)y_2$), $0 \le \alpha \le 1$.
- Given two distinct points p_1 and p_2, the ***line segment*** is the set of convex combinations of p_1 and p_2 . We call p_1 and p_2 the ***endpoints*** of segment.
- Sometimes the ordering of p_1 and p_2 matters, and we speak of the **directed segment** $\overrightarrow{p_1p_2}$. If p_1 is the origin (0, 0), then we can treat the directed segment as the **vector** p_2 .

已知两点 p_1 和 p_2，其凸组合的集合构成了线段 p_1p_2，点 p_1 和 p_2 称为线段的端点。如果考虑线段方向的含义，称 $\overrightarrow{p_1p_2}$ 为有向线段。如果 p_1 是原点，则该有向线段也称为矢量 p_2 。

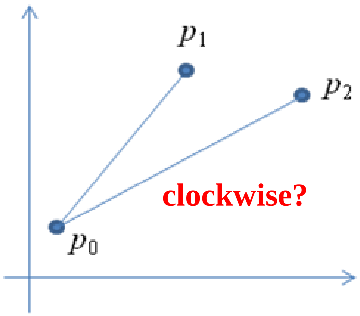




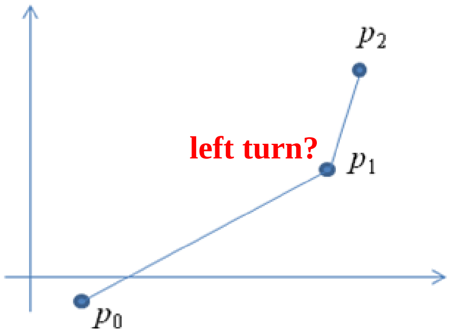
33.1 Line-segment properties

Questions:

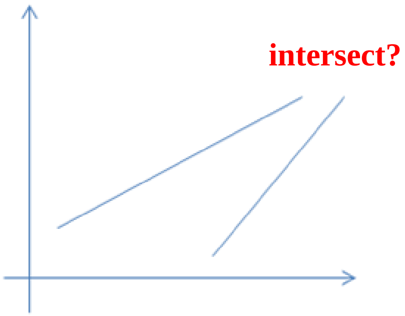
1. Given two directed segments $\overrightarrow{p_0 p_1}$ and $\overrightarrow{p_0 p_2}$, is $\overrightarrow{p_0 p_1}$ **clockwise** from $\overrightarrow{p_0 p_2}$ with respect to their common endpoint p_0 ?
2. Given two line segments $\overrightarrow{p_0 p_1}$ and $\overrightarrow{p_1 p_2}$, if we traverse $\overrightarrow{p_0 p_1}$ and then $\overrightarrow{p_1 p_2}$, do we make a **left turn** at point p_1?
3. Do line segments $\overrightarrow{p_1 p_2}$ and $\overrightarrow{p_3 p_4}$ **intersect**?



Q1



Q2

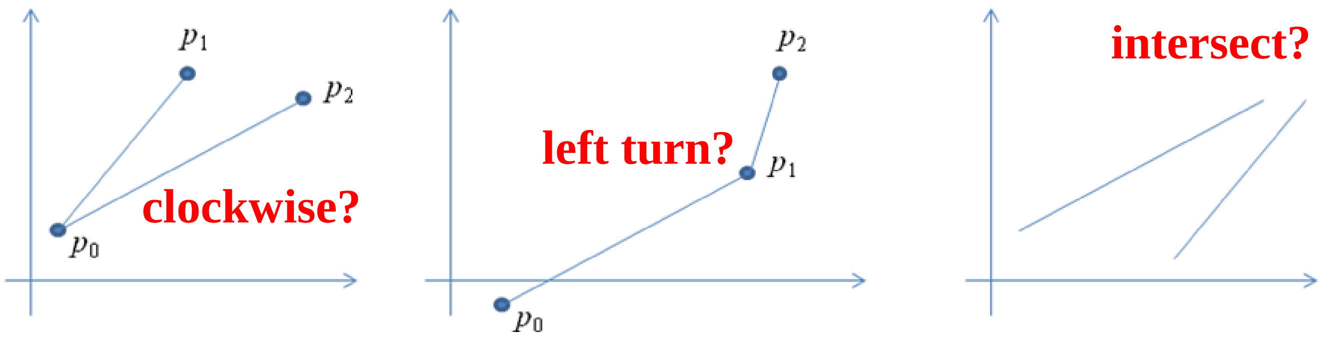


Q3

33.1 Line-segment properties

Questions:

1. Is $\overrightarrow{p_0 p_1}$ clockwise from $\overrightarrow{p_0 p_2}$ with respect to their common endpoint p_0 ?
2. If we traverse $\overrightarrow{p_0 p_1}$ and then $\overrightarrow{p_1 p_2}$, do we make a left turn at point p_1?
3. Do line segments $\overrightarrow{p_1 p_2}$ and $\overrightarrow{p_3 p_4}$ intersect?

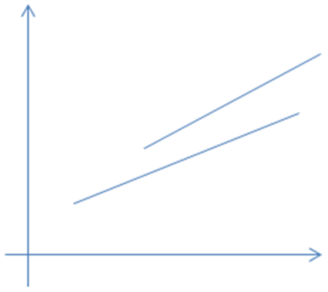


We can answer each question in $O(1)$ time.

Moreover, our methods will use only ***additions* (+)**, ***subtractions* (-)**, ***multiplications* (*)**, and ***comparisons* (==)**. We need **neither division(/) nor trigonometric functions (sin, cos, tan, …)**, both of which can be computationally expensive and prone to problems with round-off error. For example, …

33.1 Line-segment properties

Use only ***additions***, ***subtractions***, ***multiplications***, and ***comparisons***.
Neither division nor trigonometric functions.



For example,

- the "straightforward" method of determining whether two segments **intersect**
  - Compute the line equation of the form $y = mx + b$ for each segment (m is the **slope**(斜率) and b is the y-intercept (y 轴截距)),
  - find the point of intersection of the lines (uses division to find the point of intersection),
  - and check whether this point is on both segments.
- When the segments are nearly parallel, this method is very sensitive to the precision of the division operation on real computers.

$$\begin{aligned} y_1 &= mx_1 + b \\ y_2 &= mx_2 + b \end{aligned} \quad \Rightarrow \quad \begin{aligned} m &= \frac{y_2 - y_1}{x_2 - x_1} \\ b &= y_1 - mx_1 \end{aligned}$$

已知两点的坐标 $(x_1, y_1), (x_2, y_2)$，求过两点的直线方程

$$\begin{aligned} y &= m_1 x + b_1 \\ y &= m_2 x + b_2 \end{aligned} \quad \Rightarrow \quad x = \frac{b_2 - b_1}{m_1 - m_2},\ y = \frac{m_1 b_2 - m_2 b_1}{m_1 - m_2}$$

已知两条直线的方程，求两条直线的交点

The method in this section, which **avoids division**, is much more **accurate**.

33.1 Line-segment properties- **Cross products**

- Computing cross products is at the **heart** of our line-segment methods.
- For vectors p_1 and p_2, define **cross product** $p_1 \times p_2$ as the determinant(行列式) of a matrix

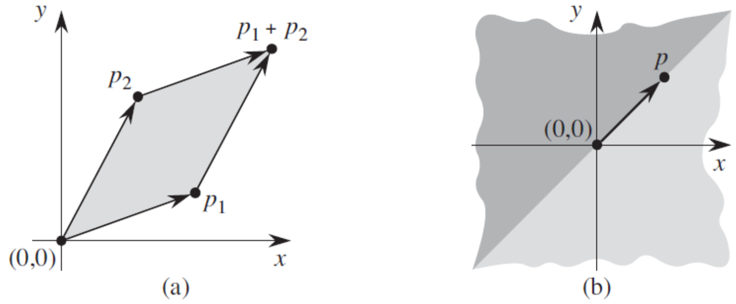
$$\begin{aligned} p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1 y_2 - x_2 y_1 \\ &= -p_2 \times p_1 \end{aligned}$$

- 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：Recursion, BFS, DFS, **Relax edge**, …

33.1 Line-segment properties- **Cross products**

cross product

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



Equivalently, $p_1 \times p_2$ can be interpreted as the signed area of the parallelogram formed by the four points (0, 0), p_1, p_2, and $p_1+p_2 = (x_1+x_2, y_1+y_2)$? See Figure 33.1(a). (Exercise: try to prove it. Hint: see Figure 33.1.a)

Figure 33.1:

(a) $p_1 \times p_2$ is the signed area of the parallelogram.

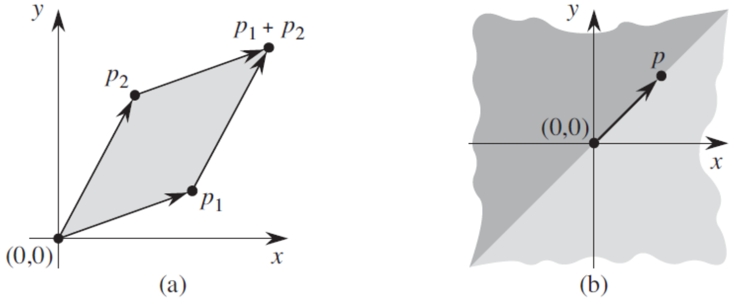
(b) The lightly shaded region contains vectors that are **clockwise** from *p*.

浅阴影区域包含从 *p* 顺时针方向的矢量

(c) The darkly shaded region contains vectors that are **counterclockwise** from *p*.

33.1 Line-segment properties- **Cross products**

$p_1 \times p_2$ can be interpreted as the signed area of the parallelogram formed by the four points (0, 0), p_1, p_2, and $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$.



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

Figure 33.1:
(a) $p_1 \times p_2$ is the signed area of the parallelogram.
(b) The lightly shaded region contains vectors that are **clockwise** from p.
(c) The darkly shaded region contains vectors that are **counterclockwise** from p.

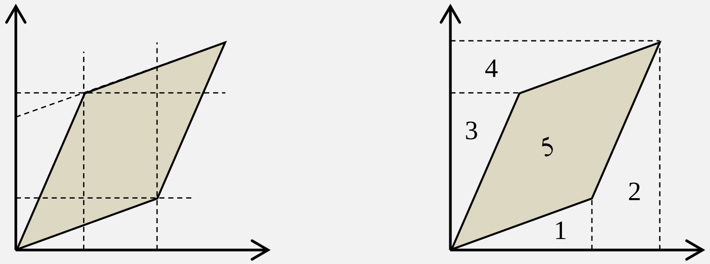


Figure 33.1.a: Hint for proof of cross product

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

"1" + "2" + "3" + "4" + "5" = $(x_1+x_2)*(y_1+y_2)$

"1" = $x_1*y_1/2$

"3" = $x_2*y_2/2$

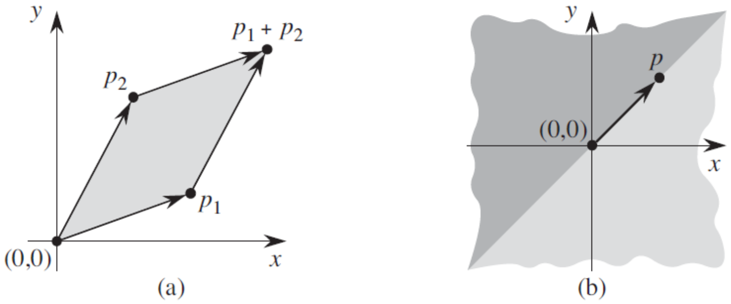
"2" = "3" + $x_2*y_1 = x_2*y_2/2 + x_2*y_1$

"4" = "1" + $x_2*y_1 = x_1*y_1/2 + x_2*y_1$

33.1 Line-segment properties- **Cross products**

Some characteristics of cross products

- if $p_1 \times p_2$ is positive, then p_1 is clockwise from p_2 with respect to the origin (0, 0).
- if negative, then p_1 is counterclockwise from p_2.
- Figure 33.1(b) shows the clockwise and counterclockwise regions relative to a vector p.
- A boundary condition arises if $p_1 \times p_2$ is 0, in this case, the vectors are ***collinear***(共线), pointing in either the same or opposite directions.



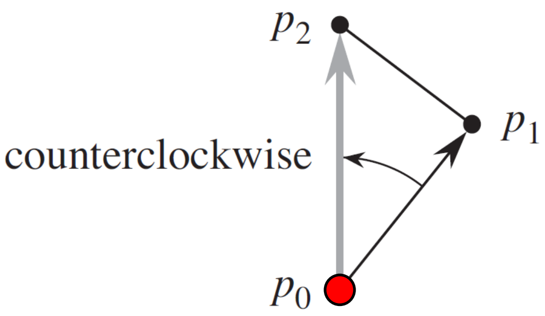
$$
\begin{aligned}
p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\
&= x_1 y_2 - x_2 y_1 \\
&= -p_2 \times p_1
\end{aligned}
$$

Figure 33.1:
(a) $p_1 \times p_2$ is the signed area of the parallelogram.
(b) The lightly shaded region contains vectors that are **clockwise** from p.
(c) The darkly shaded region contains vectors that are **counterclockwise** from p.

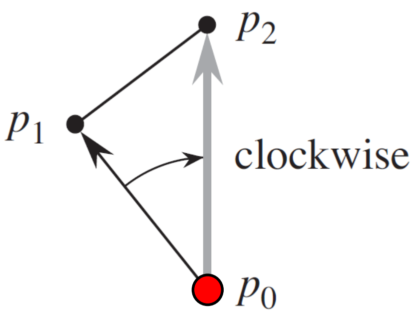
33.1 Line-segment properties- **Cross products**

1. Determine whether $\overrightarrow{p_0 p_1}$ is clockwise from $\overrightarrow{p_0 p_2}$ with respect to their common endpoint p_0

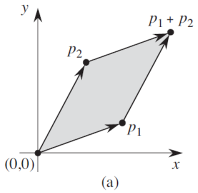
- Simply translate to use p_0 as the origin, compute the cross product $(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0)$
- If positive, clockwise; else, counterclockwise.



(a)



(b)

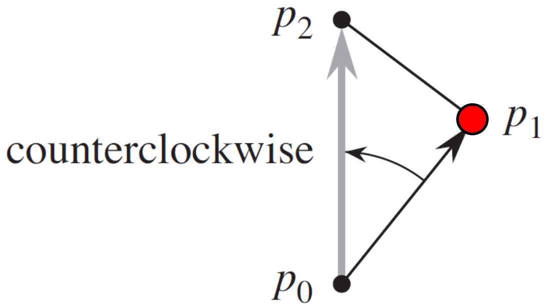


$$
\begin{aligned}
p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\
&= x_1 y_2 - x_2 y_1 \\
&= -p_2 \times p_1
\end{aligned}
$$

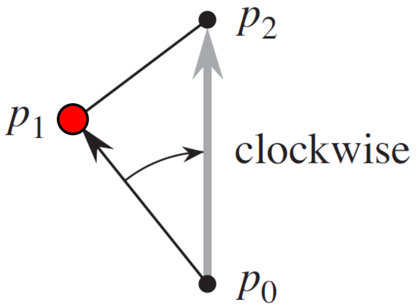
33.1 Line-segment properties- **Cross products**

2. Determining whether consecutive segments turn left or right

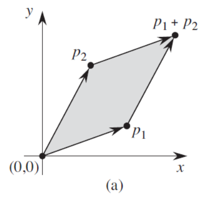
- Whether two consecutive line segments $\overrightarrow{p_0p_1}$ and $\overrightarrow{p_1p_2}$ turn left or right at point p_1.
- Equivalently, determine which way a given angle $\angle p_0p_1p_2$ turns.
- Cross products allow us to answer this question without computing the angle. We simply check whether $\overrightarrow{p_0p_1}$ is clockwise or counterclockwise relative to $\overrightarrow{p_0p_2}$.



(a)



(b)



$$\begin{aligned} p_1 \times p_2 &= \det \begin{pmatrix} x_1 & x_2 \\ y_1 & y_2 \end{pmatrix} \\ &= x_1y_2 - x_2y_1 \\ &= -p_2 \times p_1 \end{aligned}$$

33.1 Line-segment properties- **Cross products**

2. Determining whether consecutive segments turn left or right

To do this, we compute the cross product $(p_2 - p_0) \times (p_1 - p_0)$

- If negative, $\overrightarrow{p_0p_2}$ is counterclockwise from $\overrightarrow{p_0p_1}$, and thus we make a left turn at p_1.
- A positive cross product indicates a clockwise orientation and a right turn.
- A cross product of 0 means that points p_0 , p_1 , and p_2 are collinear.

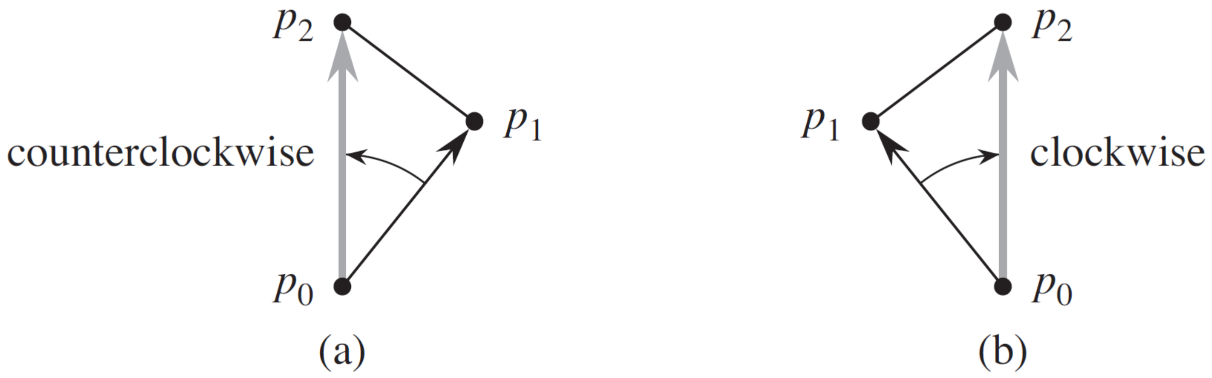


Fig 33.2: Using the cross product to determine how consecutive line segments $\overrightarrow{p_0p_1}$ and $\overrightarrow{p_1p_2}$ turn at point p_1. We check whether $\overrightarrow{p_0p_2}$ is clockwise or counterclockwise relative to $\overrightarrow{p_0p_1}$. (a) If counterclockwise, the point makes a left turn. (b) If clockwise, a right turn.

33.1 Line-segment properties- **Cross products**

3. Determining whether two line segments intersect

By checking whether each segment **straddles**(横跨) the line containing the other.

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，$i \in [1,2,3,4]$）

- **Straddles:** A segment $\overrightarrow{p_1 p_2}$ straddles a line if point p_1 lies on one side of the line and point p_2 lies on the other side.
  横跨直线：线段的两个端点在直线的两边
- *A boundary case* arises if p_1 or p_2 lies directly on the line.

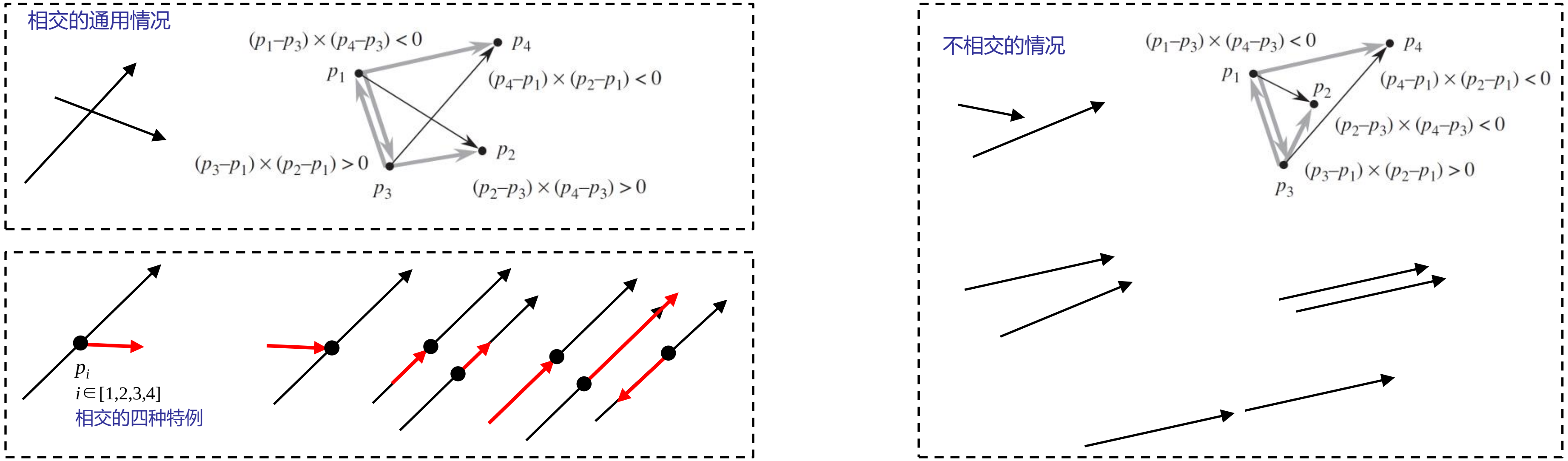


Figure 33.3: Cases in the procedure SEGMENTS-INTERSECT

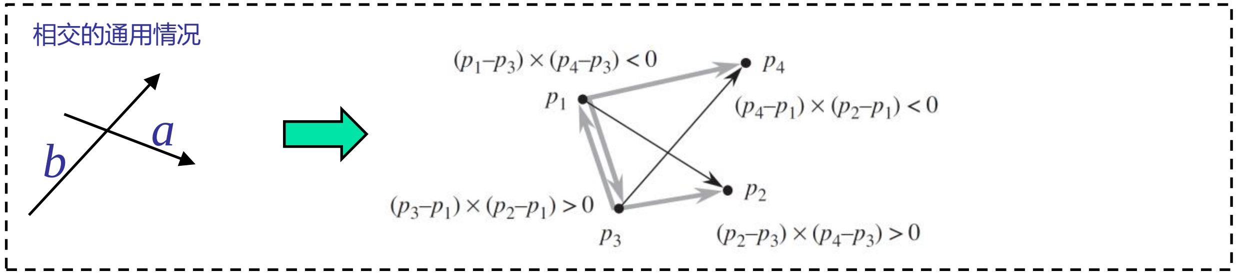
33.1 Line-segment properties- **Cross products**

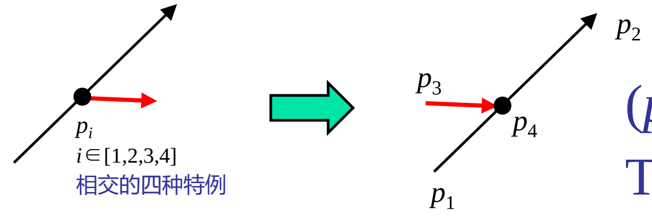
3. Determining whether two line segments intersect

Two line segments intersect if and only if either of the following conditions holds:

① Each segment straddles the line containing the other. 线段 a 横跨包括线段 b 的直线

② An endpoint of one segment lies on the other segment. 线段 a 的一个端点在线段 b 上
(This condition comes from the boundary case.)





$(p_4 - p_1) \times (p_2 - p_1) == 0$

Then, if p_4 lies on $\overrightarrow{p_1p_2}$?

Figure 33.3: Cases in the procedure SEGMENTS-INTERSECT

33.1 Line-segment properties- **Cross products**

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

- Subroutine DIRECTION: computes relative orientations using the cross product.
- ON-SEGMENT: determines whether a point known to be collinear with a segment lies on that segment.

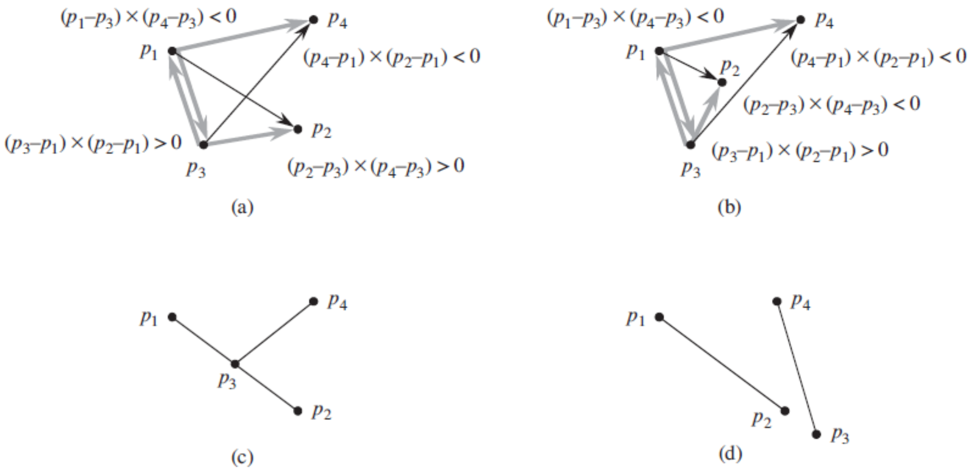


Figure 33.3: Cases in the procedure SEGMENTS-INTERSECT

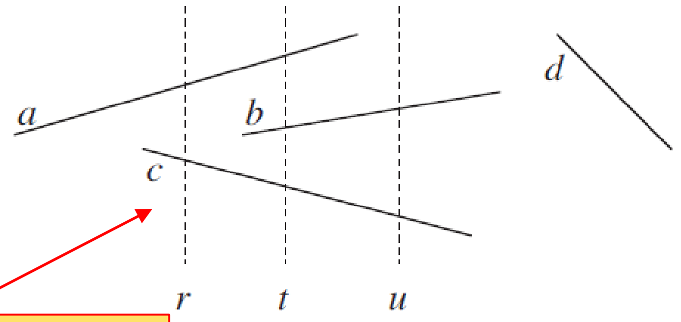
SEGMENTS-INTERSECT(p_1, p_2, p_3, p_4)
$d_1 \leftarrow$ DIRECTION(p_3, p_4, p_1) // 若为0，p_1与线段 p_3p_4 共线
$d_2 \leftarrow$ DIRECTION(p_3, p_4, p_2)
$d_3 \leftarrow$ DIRECTION(p_1, p_2, p_3)
$d_4 \leftarrow$ DIRECTION(p_1, p_2, p_4)
if (($d_1 > 0$ and $d_2 < 0$) or ($d_1 < 0$ and $d_2 > 0$)) and
(($d_3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

DIRECTION(p_i, p_j, p_k) // 求叉积
return $(p_k - p_i) \times (p_j - p_i)$

ON-SEGMENT(p_i, p_j, p_k) // 是否凸组合
if $\min(x_i, x_j) \le x_k \le \max(x_i, x_j)$ and
$\min(y_i, y_j) \le y_k \le \max(y_i, y_j)$
    return TRUE
else
    return FALSE

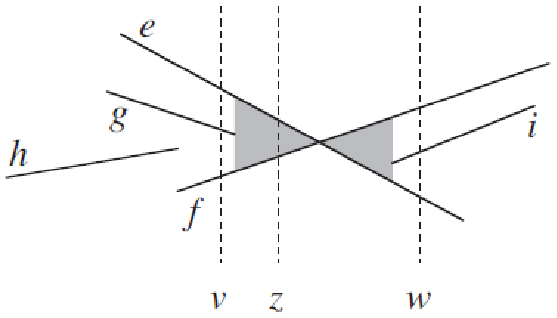
牢记定义，用好定义

*33.2 Determining whether any pair of segments intersects 是否存在线段相交



扫描线处的前序关系：
$a >_r c$

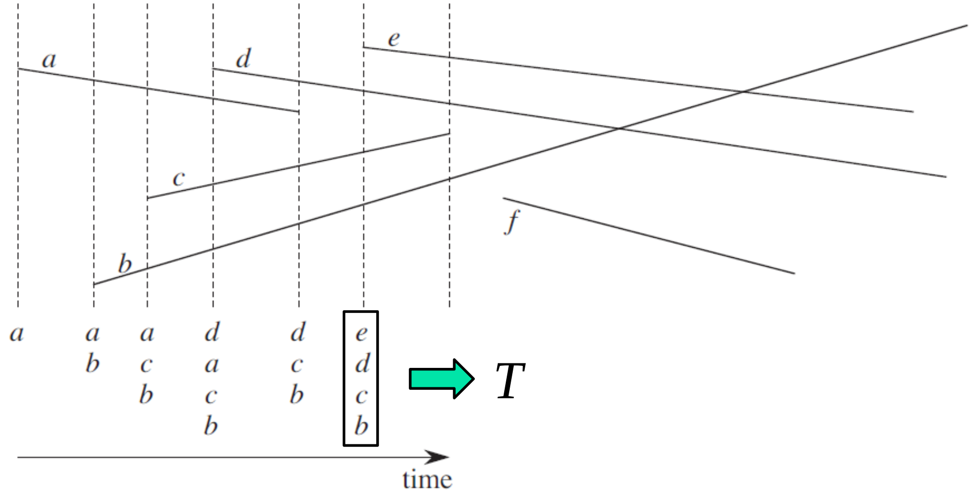
(a)



(b)

The algorithm uses a technique known as "sweeping"

- 先对线段的顶点进行排序
- 维护一个表 T，然后用sweeping技术，依序把"线段（顶点）"加入表 T 中或从表 T 中删除，扫到左端点时加线段，扫到右端点时删线段
- 没有竖线（垂直于横坐标的线段），不关心有多少个交点（更难），也不用求出每一个交点坐标。



*33.2 Determining whether any pair of segments intersects 是否存在线段相交

ANY-SEGMENTS-INTERSECT(S)

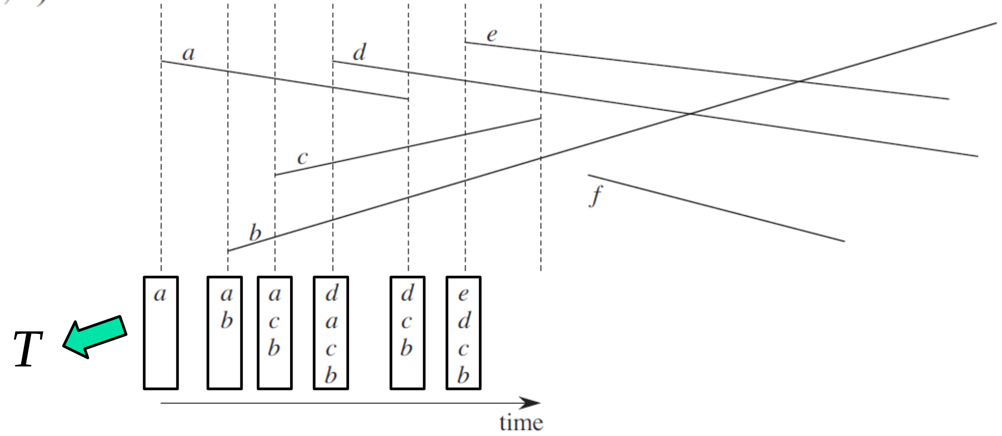
```
T = Ø   // 维护一个有顺序关系的线段集合 T
sort the endpoints of the segments in S from left to right,
        breaking ties by putting left endpoints before right endpoints
        and breaking further ties by putting points with lower
        y-coordinates first
for each point p in the sorted list of endpoints
    if p is the left endpoint of a segment s
        INSERT(T, s)
        if (ABOVE(T, s) exists and intersects s)
            or (BELOW(T, s) exists and intersects s)
            return TRUE
    if p is the right endpoint of a segment s
        if both ABOVE(T, s) and BELOW(T, s) exist
            and ABOVE(T, s) intersects BELOW(T, s)
            return TRUE
        DELETE(T, s)
return FALSE
```

- 先对线段（顶点）进行排序，$O(n\lg n)$
- 然后sweeping，$O(n)$

Insert(T, s), Delete(T, s),
维护 T 里线段的顺序关系：
【chap13, 红黑树，$O(\lg n)$】*

$O(n\lg n)$

Running Time?



*33.2 Determining whether any pair of segments intersects 是否存在线段相交

```
ANY-SEGMENTS-INTERSECT(S)
T = ∅   // 维护一个有顺序关系的线段集合 T
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
for each point p in the sorted list of endpoints
        if p is the left endpoint of a segment s
                INSERT(T, s)
                if (ABOVE(T, s) exists and intersects s)
                        or (BELOW(T, s) exists and intersects s)
                        return TRUE
        if p is the right endpoint of a segment s
                if both ABOVE(T, s) and BELOW(T, s) exist
                        and ABOVE(T, s) intersects BELOW(T, s)
                        return TRUE
                DELETE(T, s)
return FALSE
```

Correctness?

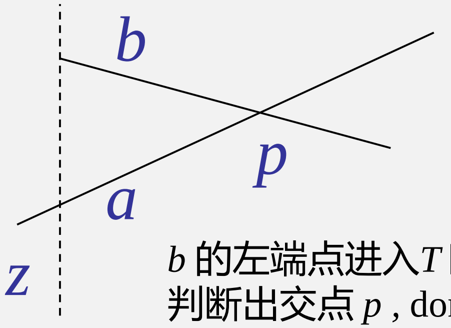
定理：算法正确（找到交点时返回ture），当且仅当(if and only if) 有交点。

思路：

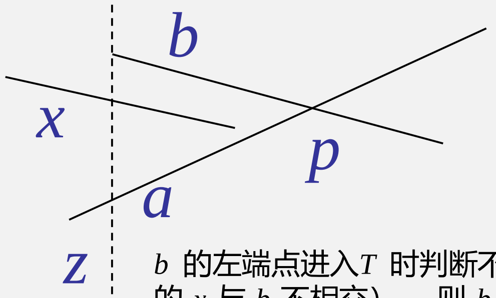
⇨：算法返回true，有交点。显然。

⇦：**有交点，一定会被判断出来（会被发现）**，因此，算法返回ture。

设只有一个交点 p



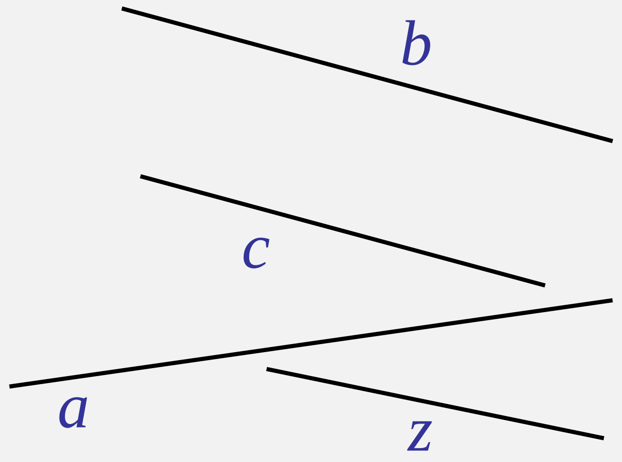
b 的左端点进入 T 时判断出交点 p，done



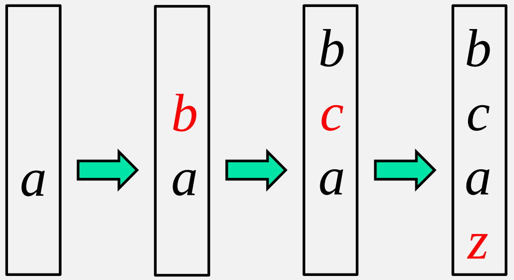
b 的左端点进入 T 时判断不出交点 p（b 下方的 x 与 b 不相交），则 b 和 a 之间一定有线段 x，且 x 与 a 也不相交（因为假设只有一个交点，或 b加入前交点就被判断出来了），x 的右端点检测到时，能判断出交点 p

*33.2 Determining whether any pair of segments intersects 是否存在线段相交

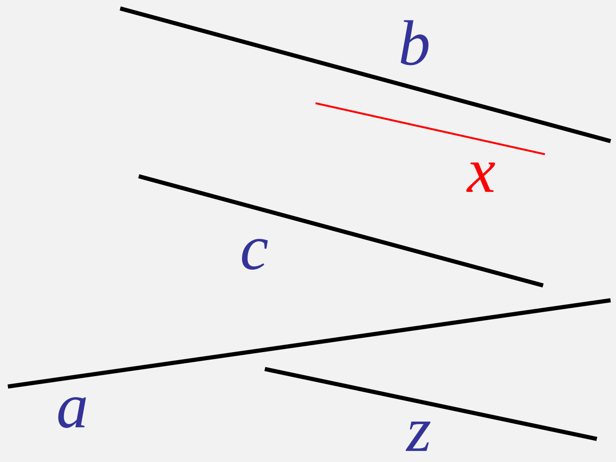
思考：维护 T 里线段的顺序关系：如何判断一条新加入线段 x 在 T 中的哪个位置（x 加入前其他线段的顺序已知【在 x 的左端点处的顺序，x 加入前，T 中也没有线段移除】）？即，x 的左端点 x_L 在哪条线段上方，在哪条线段下方（x 加入前，没有线段相交）？



T 中：已知 $z < a < c < b$

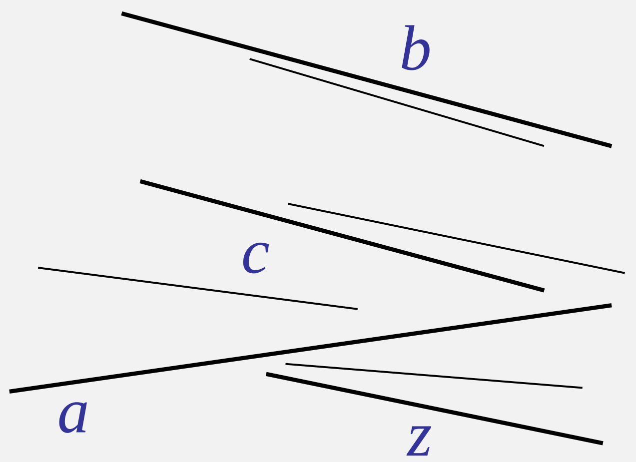


依序加入每条线段时的情况

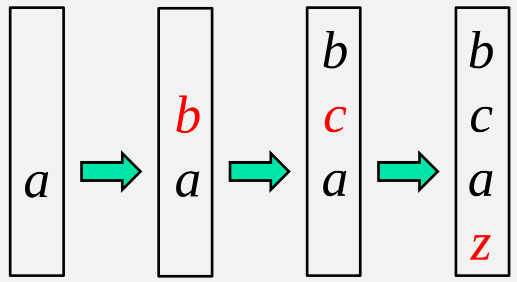


*33.2 Determining whether any pair of segments intersects 是否存在线段相交

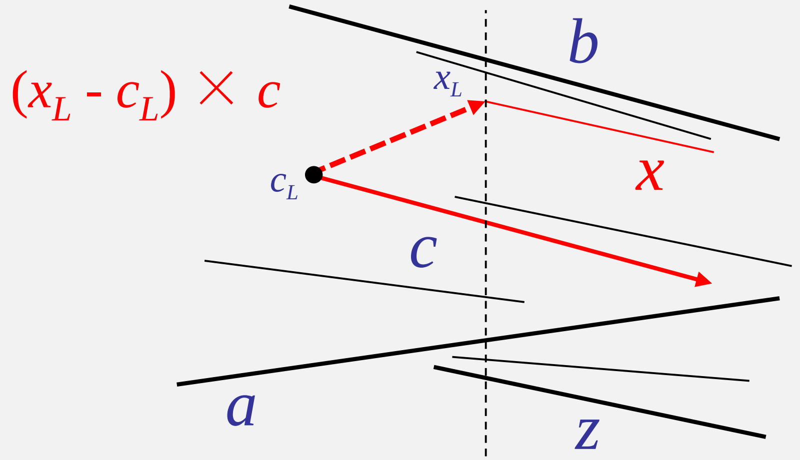
思考：维护 T 里线段的顺序关系：如何判断一条新加入线段 x 在 T 中的哪个位置（x 加入前其他线段的顺序已知【在 x 的左端点处的顺序，x 加入前，T 中也没有线段移除】）？即，x 的左端点 x_L 在哪条线段上方，在哪条线段下方（x 加入前，没有线段相交）？



T 中：已知 $z < a < c < b$



依序加入每条线段时的情况



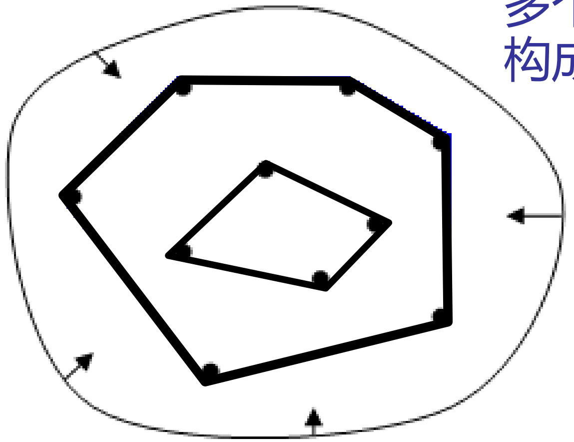
加入 x 时，用二分法，取 T 中的中间线段 c (T 中的线段的顺序在构造时 T 时已知)，以 c 的左端 c_L 为共同端点，求线段 c_Lx_L 与 c 的叉积，可得 x 在 x_L 处是在 c 的上面或下面；若 x 在 c 的上面，则在 T 的上半段线段集合中递归判断；...

33.3 Finding the convex hull

CH(Q), the ***convex hull*** of a set Q of points, is the smallest convex polygon P for which each point in Q is either on the boundary of P or in its interior.

点集 Q 的凸包：是一个凸多边形 P，Q 中的每一个点在 P 的内部，或在 P 的边上（含顶点）。

Intuitively, we can think of each point in Q as being a nail sticking out from a board. The convex hull is then the shape formed by a tight rubber band that surrounds all the nails (See Figure 33.6).



多个凸包
构成凸层

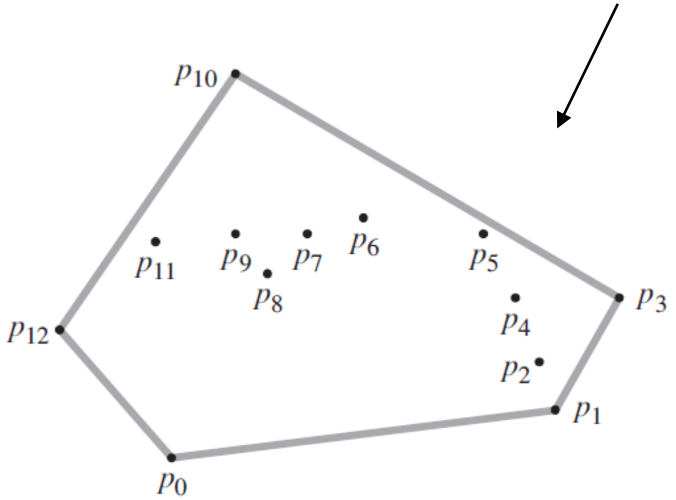
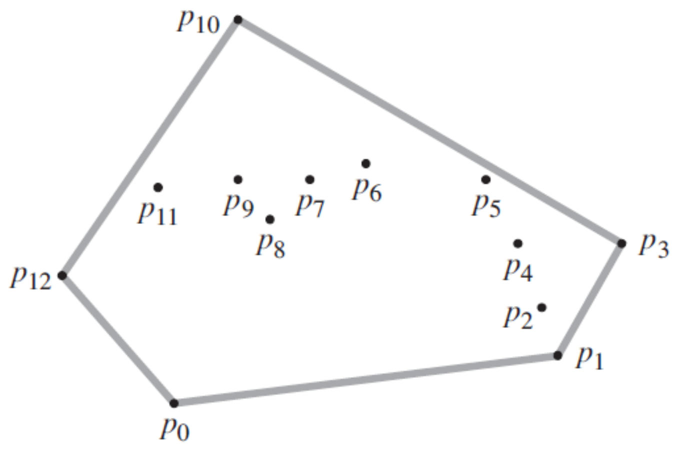


Figure 33.6: A set of points $Q = \{p_0, p_1, ..., p_{12}\}$ with its convex hull CH(Q) in gray

33.3 Finding the convex hull

- Computing the convex hull of a set of points is an interesting problem in its own right.



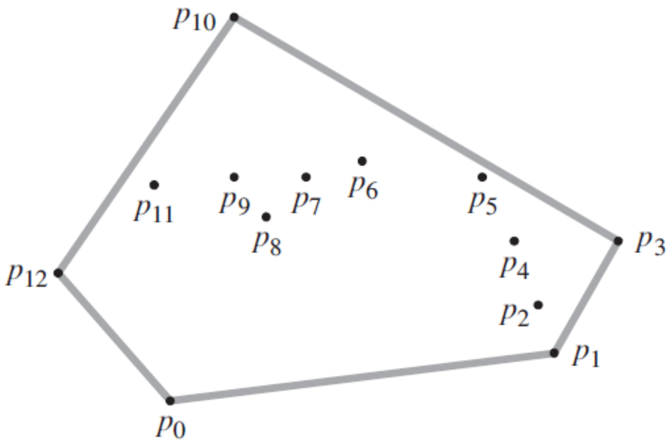
- Moreover, algorithms for some other computational-geometry problems start by computing a convex hull. Consider, for example, the two-dimensional ***farthest-pair problem*** (Exer 33.3-3). (Exercises: 33.3-3 Given a set of points Q, prove that the pair of points farthest from each other must be vertices of CH(Q))

计算凸包本身是一个有趣的问题。求凸包有许多应用，比如求最远点对。

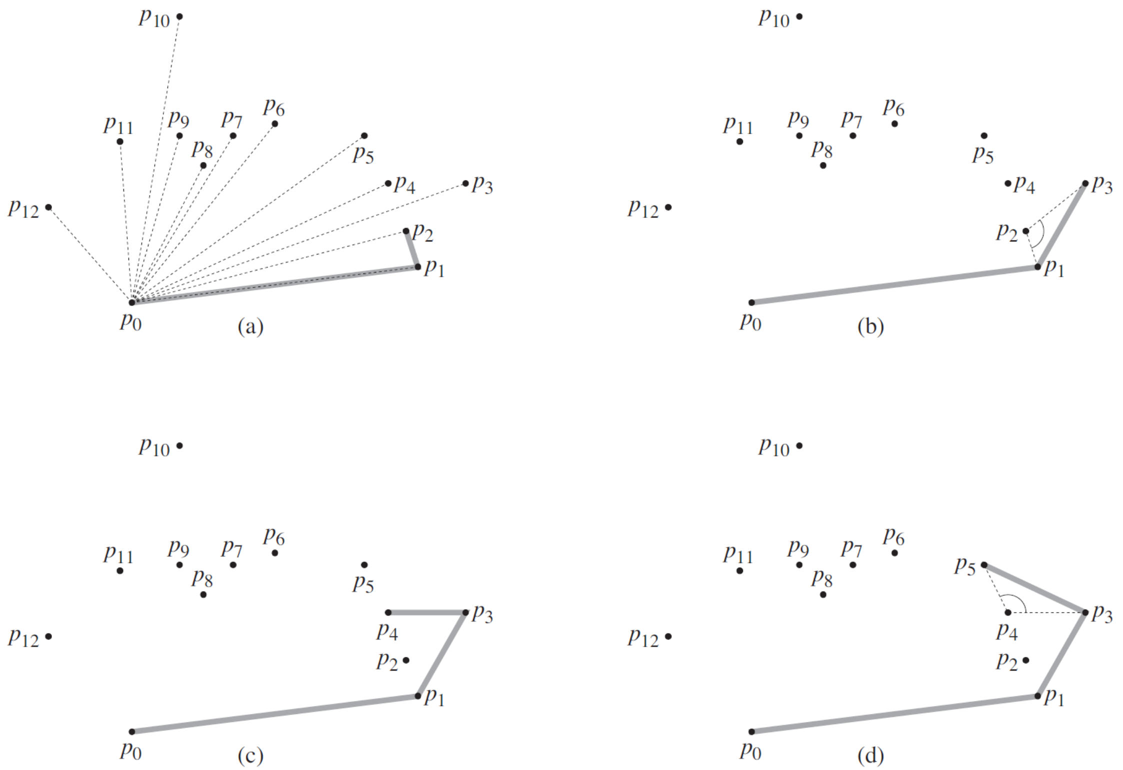
33.3 Finding the convex hull

Some algorithms that compute the convex hull of a set of *n* points:

- Graham's scan (格雷厄姆), runs in $O(n\lg n)$ time
- Jarvis's march (贾维斯), runs in $O(nh)$ time, where *h* is the number of vertices of the convex hull.
- Additional several methods
  - *incremental method*, $O(n\lg n)$
    增量法（每次增加一点，更新当前凸包）
  - *divide-and-conquer method*, $O(n\lg n)$
  - *prune-and-search method*, $O(n\lg h)$
    剪枝-搜索法：先找凸包上部分，然后找下部分



33.3 Finding the convex hull--**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. Proc. Lett., 1972 - cir.nii.ac.jp

An efficient algorithm for determining the convex hull of a finite planar set | CiNii Research …

An efficient algorithm for determining the convex hull of a finite planar set …

☆ 保存 99 引用 被引用次数：2700 相关文章 所有 3 个版本 »

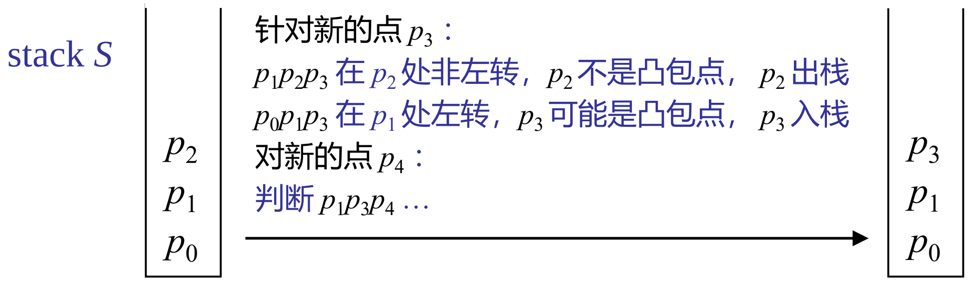
33.3 Finding the convex hull--**Graham's scan**

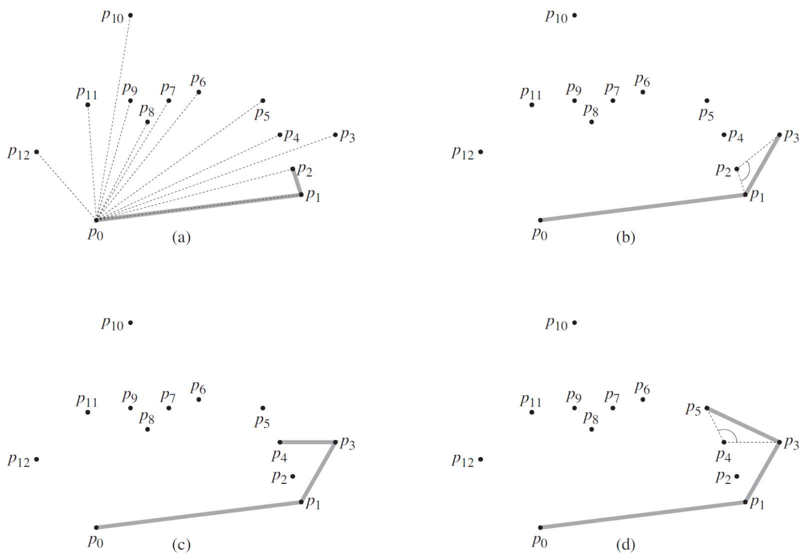
By maintaining a **stack *S*** of candidate points, **consecutive segments turn left or right**

- Each point of the input set Q is pushed once onto the stack. 每个点入栈一次
- The points that are not vertices of CH(Q) are eventually popped from the stack. 如果不是凸包的顶点，则顶点出栈
- When the algorithm terminates, stack S contains exactly the vertices of CH(Q), in counterclockwise order of their appearance on the boundary.

以栈 S 的顶点 top 为中间点（连接点），判断两条连续线段在点 top 处的“左转”或“非左转”，来决定“新点”入栈或 top 出栈操作。

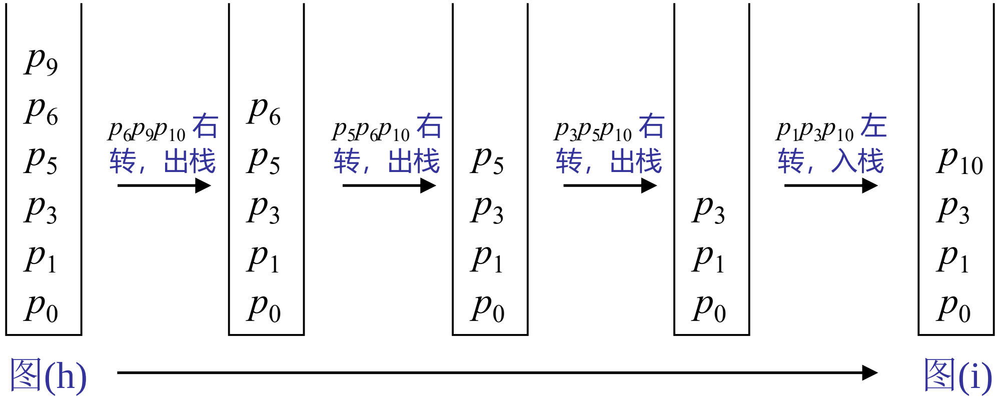
GRAHAM-SCAN(Q)
1 let p_0 be the point in Q with the minimum y-coordinate,
   or the leftmost such point in case of a tie
2 let $<p_1, p_2, ..., p_m>$ be the remaining points in Q,
   sorted by polar angle in counterclockwise order around p_0 (if more than one point has the same angle, remove all but the one that is farthest from p_0)
3 PUSH(p_0, S), PUSH(p_1, S), PUSH(p_2, S) // 初始的3个可能凸包点
4 **for** $i \leftarrow 3$ **to** m
5   **while (** the consecutive segments formed by points
      NEXT-TO-TOP(S), TOP(S), and p_i make a nonleft turn**)**
      // 当非左转，即，直线（栈顶点在凸包边上）或右转（栈顶点在凸包内）
6   POP(S)
7   PUSH(p_i, S) // p_i 可能为凸包顶点，入栈
8 **return** S

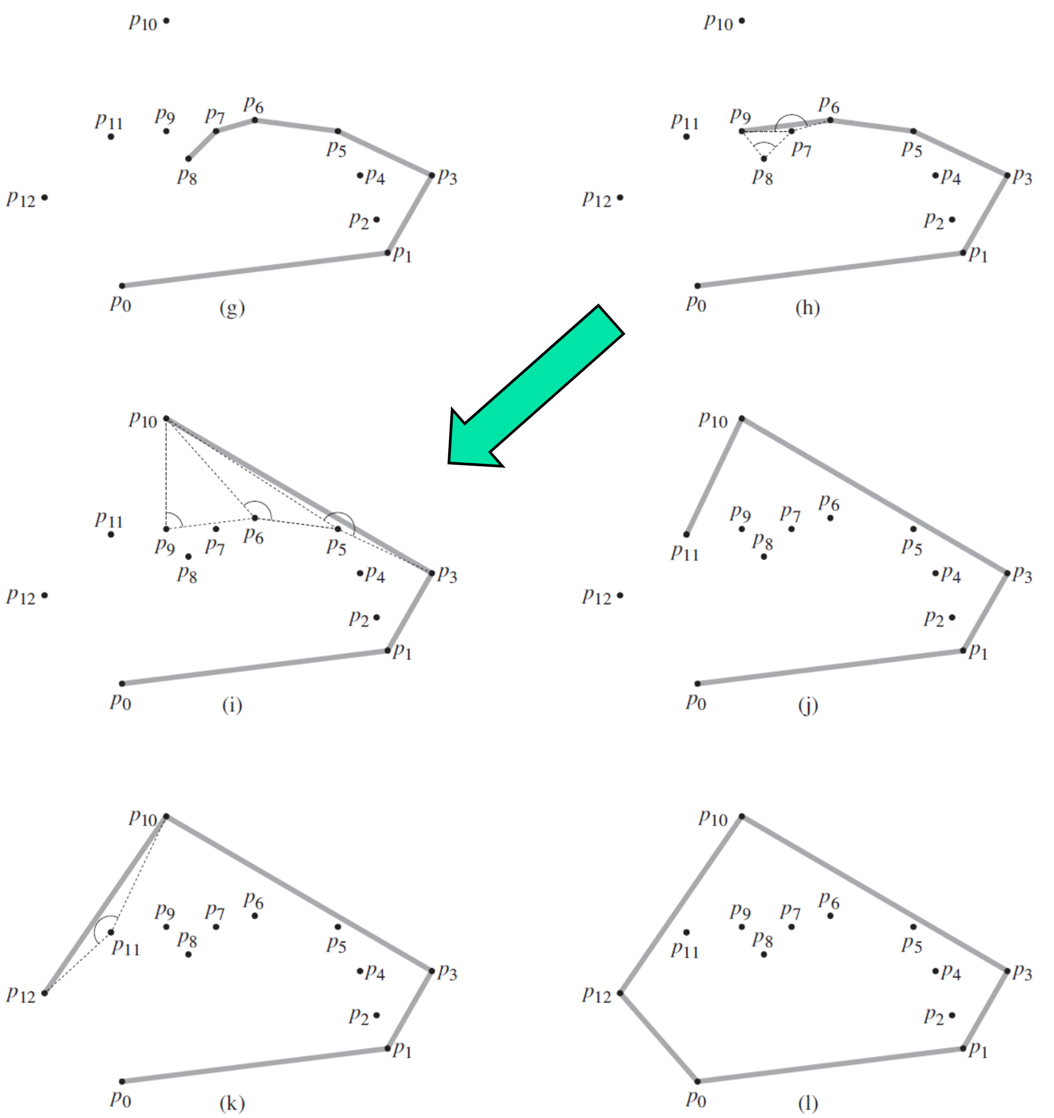




33.3 Finding the convex hull--**Graham's scan**

```
GRAHAM-SCAN(Q)
let p0 be the point in Q with the minimum y-coordinate,
      or the leftmost such point in case of a tie
let <p1, p2, ..., pm> be the remaining points in Q,
      sorted by polar angle in counterclockwise order around p0 (if
      more than one point has the same angle, remove all but the one
      that is farthest from p0 )
PUSH(p0, S),  PUSH(p1, S),  PUSH(p2, S) // 初始的3个可能凸包点
for i ← 3 to m
    while ( the consecutive segments formed by points
            NEXT-TO-TOP(S), TOP(S), and pi make a nonleft turn)
          // 当非左转，即，直线（栈顶点在凸包边上）或右转（栈顶点在凸包内）
        POP(S)
    PUSH(pi, S) // pi 可能为凸包顶点，入栈
return S
```





33.3 Finding the convex hull--**Graham's scan**

GRAHAM-SCAN(Q)

let p_0 be the point in Q with the minimum y-coordinate, or the leftmost such point in case of a tie

let $<p_1, p_2, ..., p_m>$ 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)

PUSH(p_0, S), PUSH(p_1, S), PUSH(p_2, S) // 初始的3个可能凸包点

for $i \leftarrow 3$ **to** m

while (the consecutive segments formed by points NEXT-TO-TOP(S), TOP(S), and p_i make a nonleft turn**)**

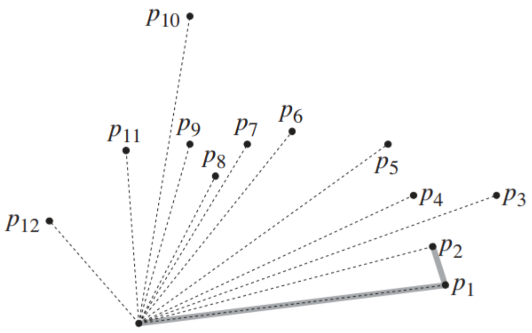
// 当非左转，即，直线（栈顶点在凸包边上）或右转（栈顶点在凸包内）

POP(S)

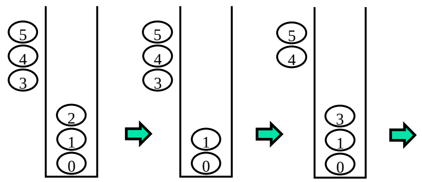
PUSH(p_i, S) // p_i 可能为凸包顶点，入栈

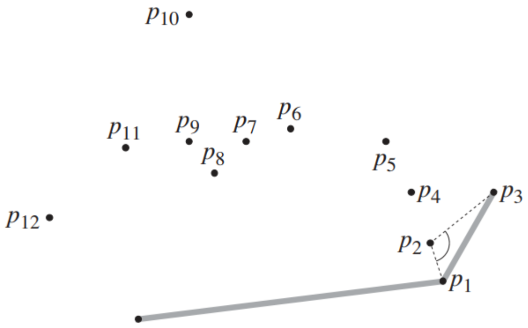
return S

$T(n) = ?$



(a)





(b)

33.3 Finding the convex hull--**Graham's scan**

GRAHAM-SCAN(Q)

let p_0 be the point in Q with the minimum y-coordinate, or the leftmost such point in case of a tie

let $\langle p_1, p_2, \ldots, p_m \rangle$ 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)

PUSH(p_0, S), PUSH(p_1, S), PUSH(p_2, S) // 初始的3个可能凸包点

for $i \leftarrow 3$ **to** m

    while (the consecutive segments formed by points NEXT-TO-TOP(S), TOP(S), and p_i make a nonleft turn**)**

        // 当非左转，即，直线（栈顶点在凸包边上）或右转（栈顶点在凸包内）

        POP(S)

    PUSH(p_i, S) // p_i 可能为凸包顶点，入栈

return S

$T(n)$:

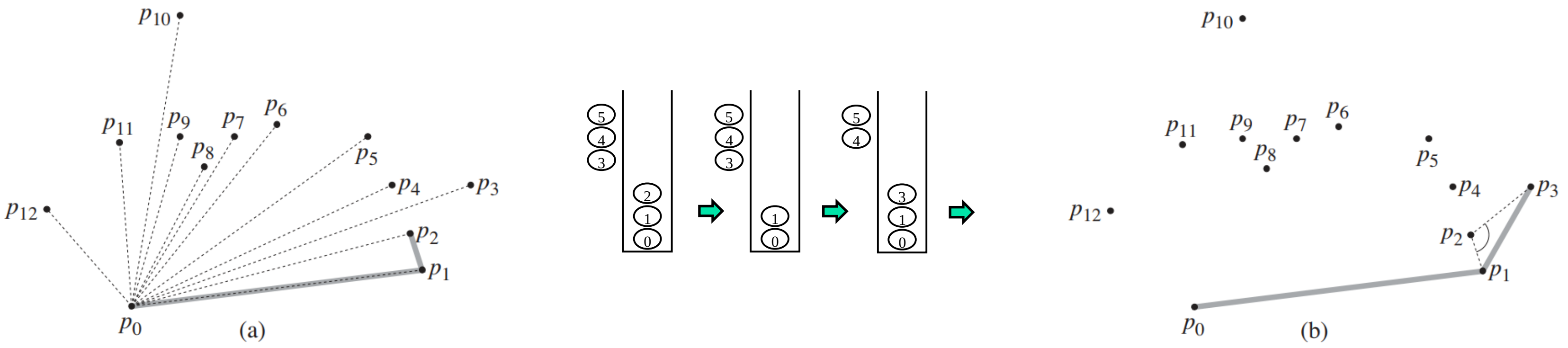
$\Theta(n)$

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

$O(1)$

$O(n-3)$

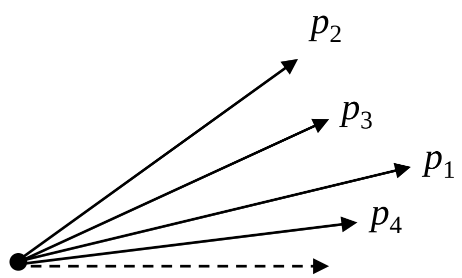
Aggregate analysis: **while loop** takes $O(n)$ time overall. For i = 0, 1, ..., m, each point p_i is pushed onto stack S exactly once, there is at most one POP operation for each PUSH operation. At least three points p_0, p_1, and p_m are never popped from the stack, so that in fact at most (m - 2) POP operations are performed in total?



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 $\langle p_1, p_2, \ldots, p_m \rangle$ 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

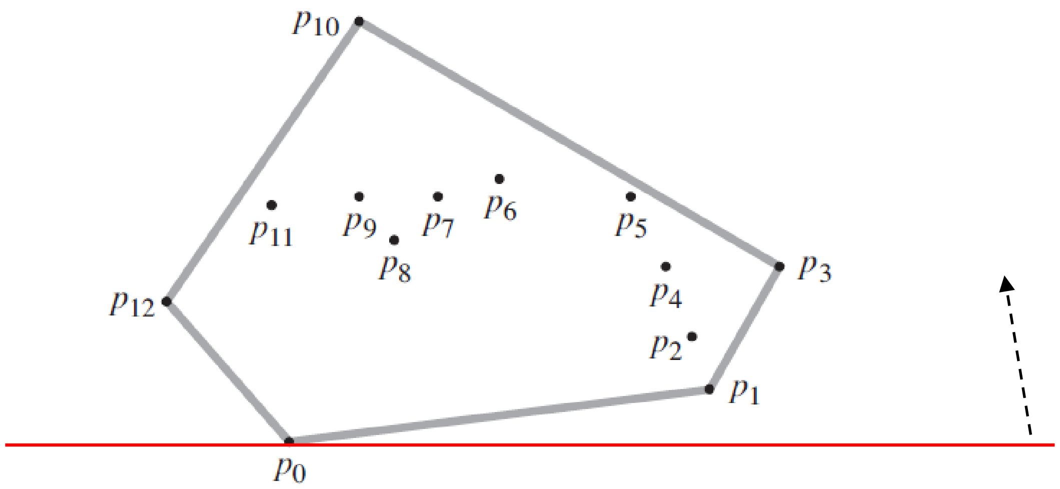
按极角序

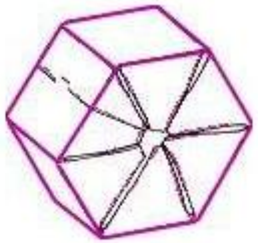
Output: $p_4 < p_1 < p_3 < p_2$

如何做?

33.3 Finding the convex hull--**Jarvis's march**

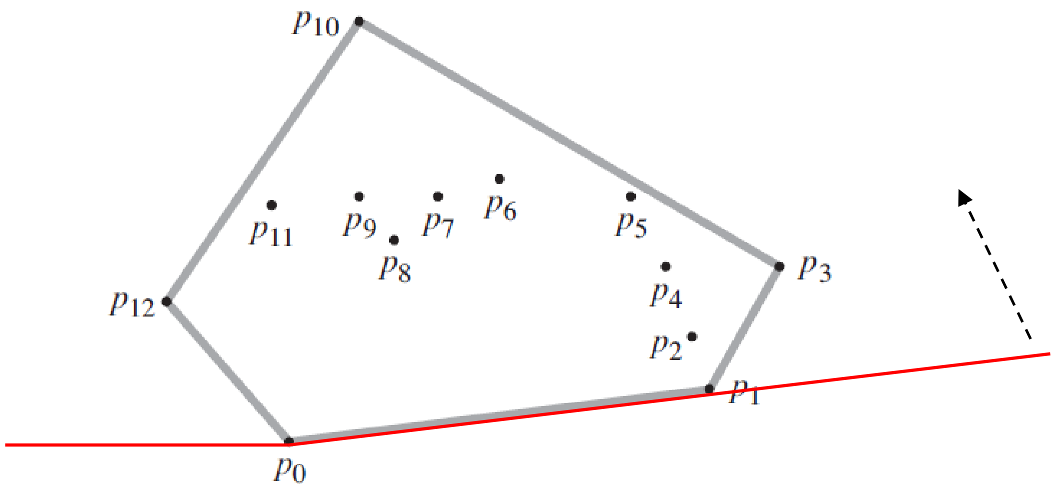
- ***Jarvis's march*** computes the convex hull of a set Q of points by a technique known as ***package wrapping*** (or ***gift wrapping***).
- The algorithm runs in time $O(nh)$, where h is the number of vertices of CH(Q).
- Is Jarvis's march asymptotically faster than Graham's scan, whose running time is $O(n\lg n)$?

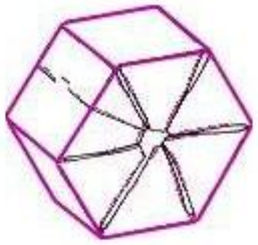




33.3 Finding the convex hull--**Jarvis's march**

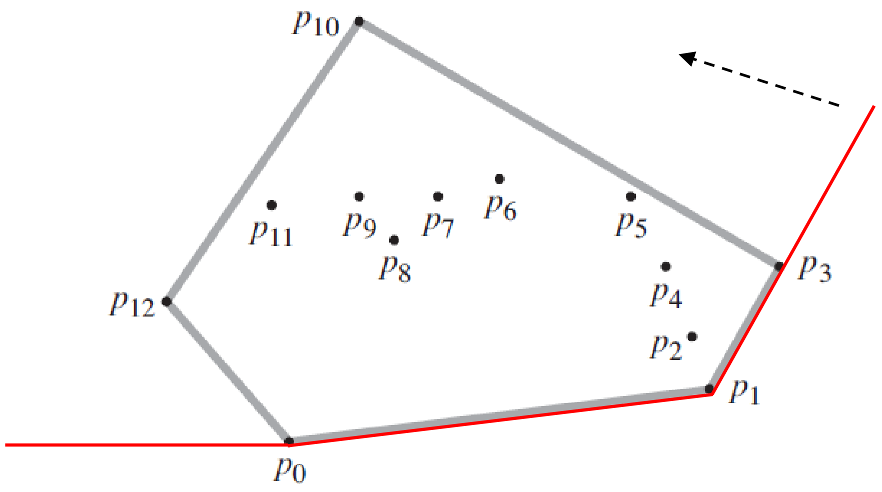
- ***Jarvis's march*** computes the convex hull of a set Q of points by a technique known as ***package wrapping*** (or ***gift wrapping***).
- The algorithm runs in time $O(nh)$, where h is the number of vertices of CH(Q).
- Is Jarvis's march asymptotically faster than Graham's scan, whose running time is $O(n\lg n)$?





33.3 Finding the convex hull--**Jarvis's march**

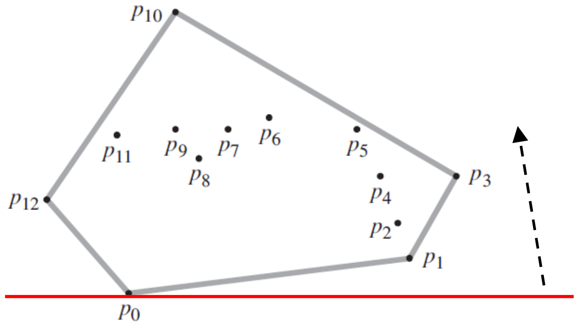
- ***Jarvis's march*** computes the convex hull of a set Q of points by a technique known as ***package wrapping*** (or ***gift wrapping***).
- The algorithm runs in time $O(nh)$, where h is the number of vertices of CH(Q).
- Is Jarvis's march asymptotically faster than Graham's scan, whose running time is $O(n\lg n)$?

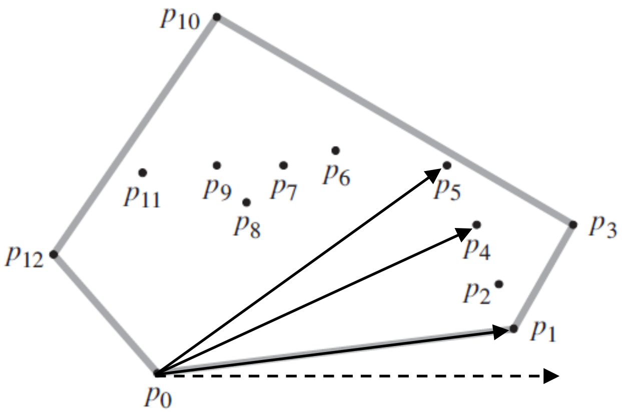


33.3 Finding the convex hull--**Jarvis's march**

- Intuitively, Jarvis's march simulates wrapping a taut piece of paper around the set Q.
  - We start by taping the end of the paper to the lowest point in the set.
  - We pull the paper to the right to make it taut, and then we pull it higher until it touches a point. This point must also be a vertex of the convex hull.
  - Keeping the paper taut, we continue in this way around the set of vertices until we come back to our original point p_0.
- Jarvis's march has a running time of $O(nh)$?

Jarvis 方法模拟用一张绷直的纸进行包装过程。





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 $O(nh)$?

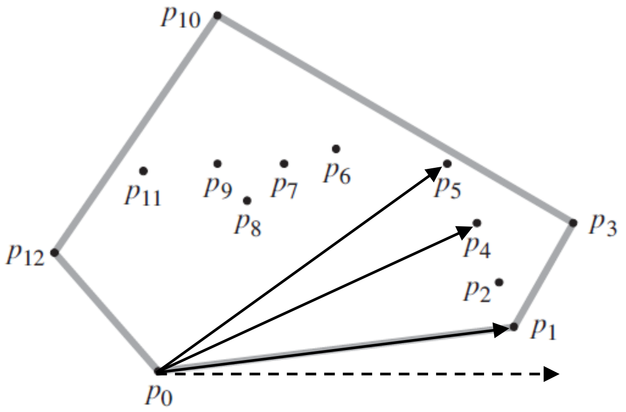
p_0为端点： $p_0p'_1$ $p_0p'_2$ … $p_0p'_n$ ，找出极坐标角最小的点p_1 (CH point) $O(n)$

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) $O(n)$

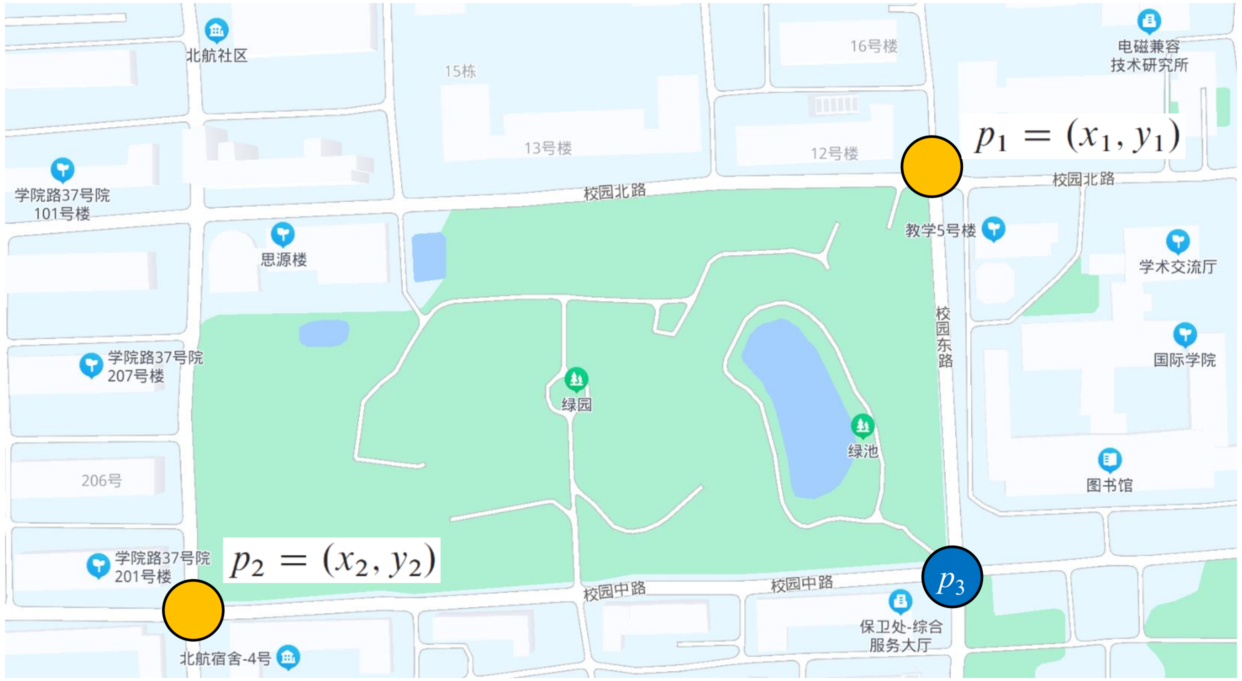
…

如何求极坐标角最小?

共 h 个CH point



*33.4 Finding the closest pair of points



Euclidean distance

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

***L_m*-distance** (Minkowski distance)

$(|x_1-x_2|^m + |y_1-y_2|^m)^{1/m}$

L_1 : Manhattan distance

$|x_1-x_2| + |y_1-y_2|$

L_2 : Euclidean distance

L_∞ : Visual distance

$\max(|x_1-x_2| , |y_1-y_2|)$，晚点见到你距离

（p_2 走到 p_3 花的时间，此时能看到 p_1）

$\min(|x_1-x_2| , |y_1-y_2|)$，早点见到你距离

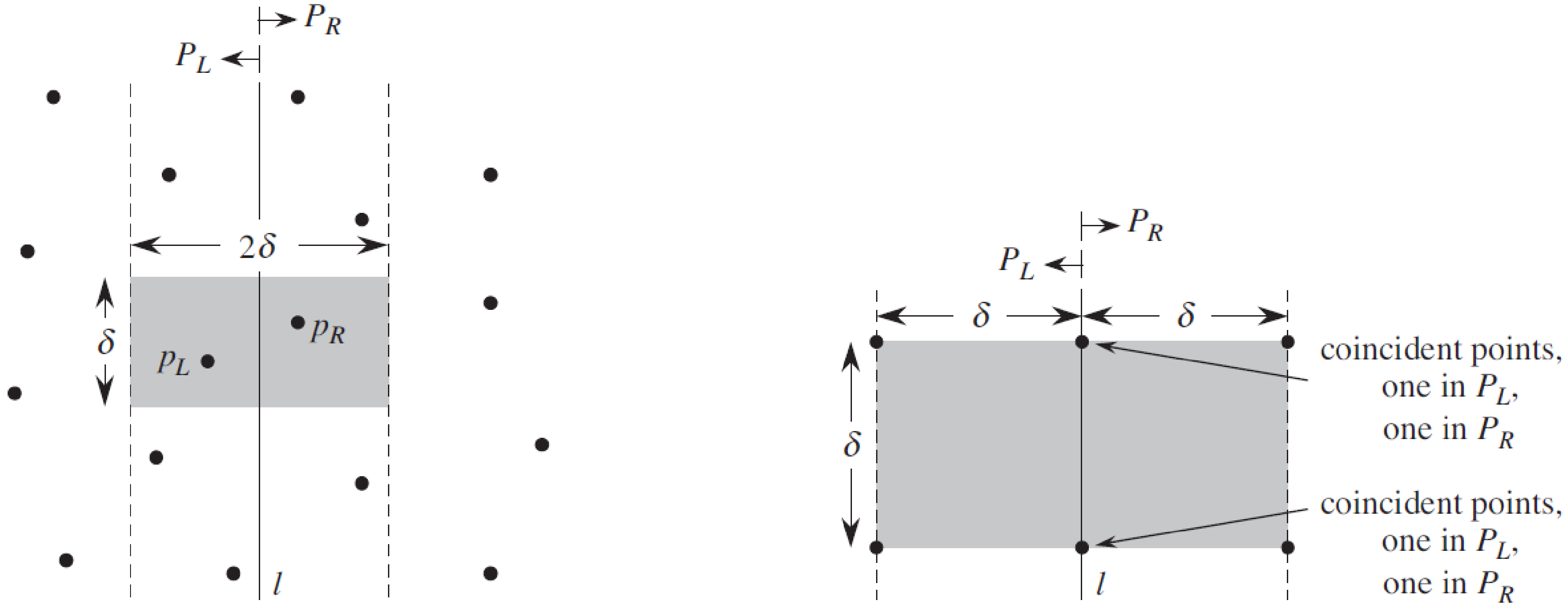
（p_1 走到 p_3 花的时间，此时能看到 p_2）

*33.4 Finding the closest pair of points

1. Brute method: $C(n, 2)$, $O(n^2)$
2. divide-and-conquer: $O(n \lg n)$

Euclidean distance

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



(a) (b)

Summary

- Cross products

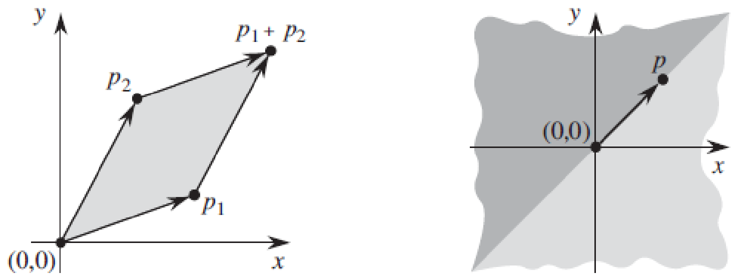
- Divide and conquer
- Merge sort
- Stack
- Binary search
- Red-black tree*

- Aggregate analysis

Exercise 1

$p_1 \times p_2$ can be interpreted as the signed area of the parallelogram formed by the four points $(0, 0)$, p_1, p_2, and $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$?

证明：叉积是由这 4 个点构成的平行四边形的面积。

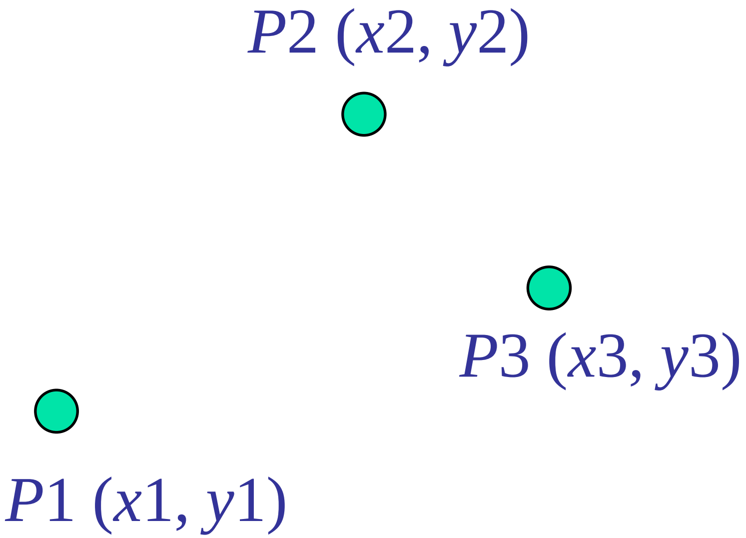


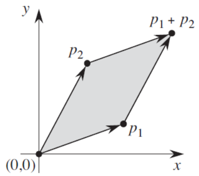
(a) (b)

Exercise 2

求线段 $P1P2$ 和 $P1P3$ 的叉积

(表达式 $(xi - xj)(yi - yj)$ 不需要再化解)



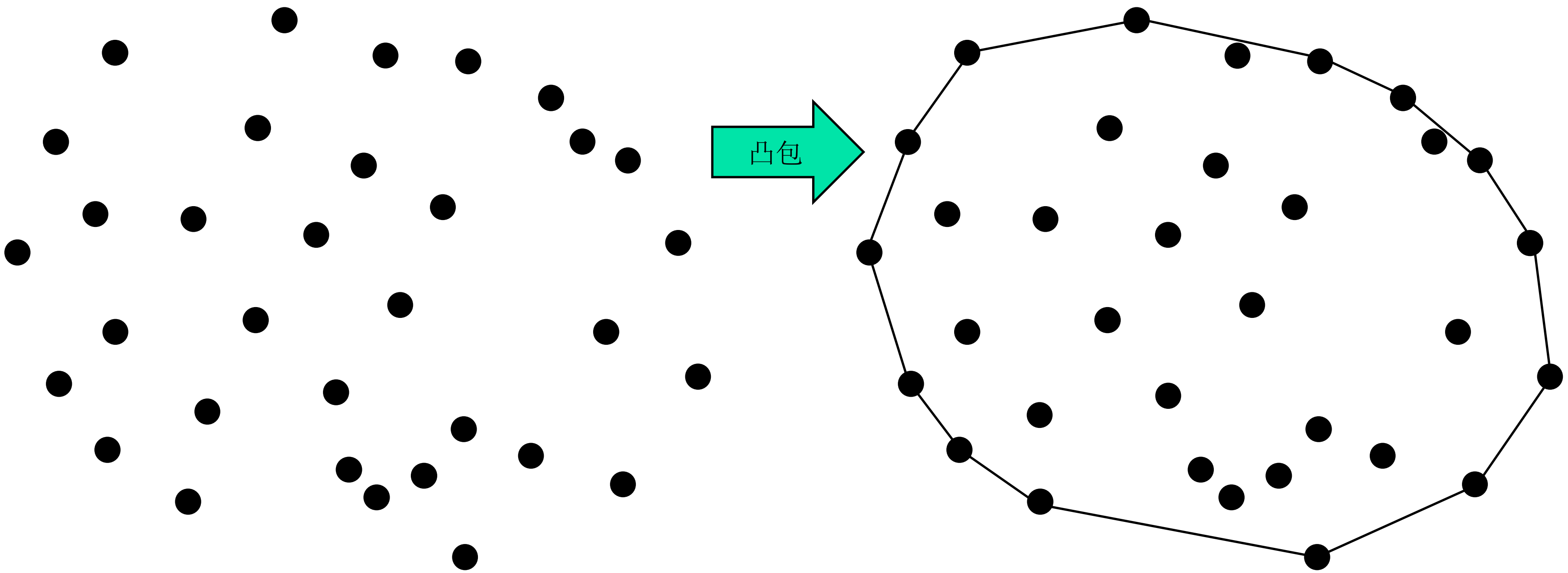


(a)

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

Exercise 3

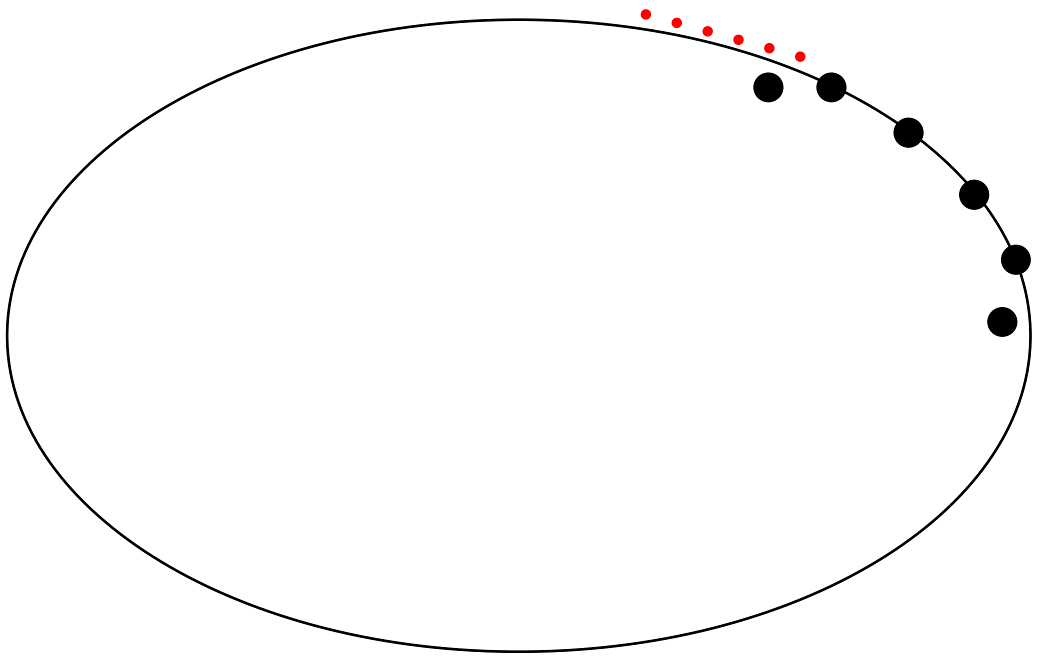
对类似下面这种特征的点集（左图）求凸包，从理论上讲，Jarvis's march 和 Graham's scan 哪个更快？ （填 Jarvis 或 Graham）



#个点？

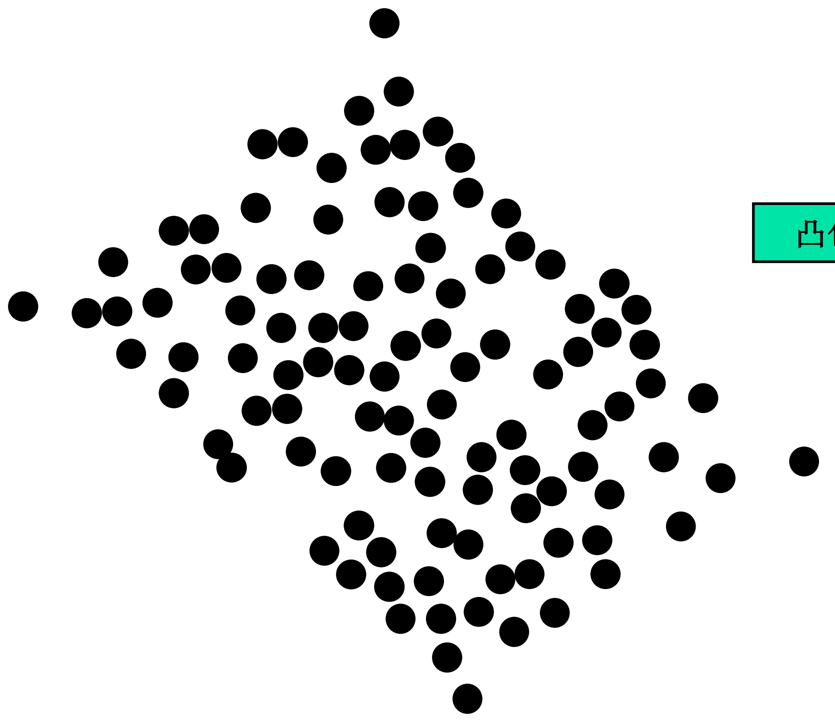
Exercise 4

这个呢？ （沿着椭圆周长上随机产生一列的点，内部有极少量零星的点） （填 Jarvis 或 Graham）

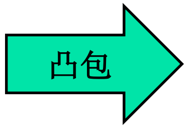


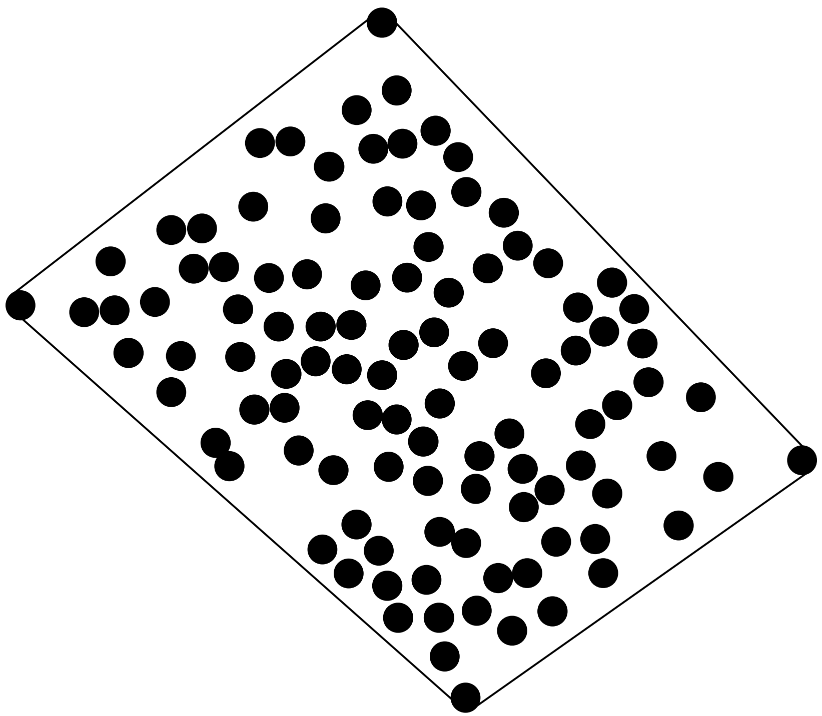
Exercise 5

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？ （填 Jarvis 或 Graham）



Q: #个点？





Ch(Q): h = 4个顶点