3.6 Finding the convex hull

问题定义

输入：平面上的n个点的集合Q

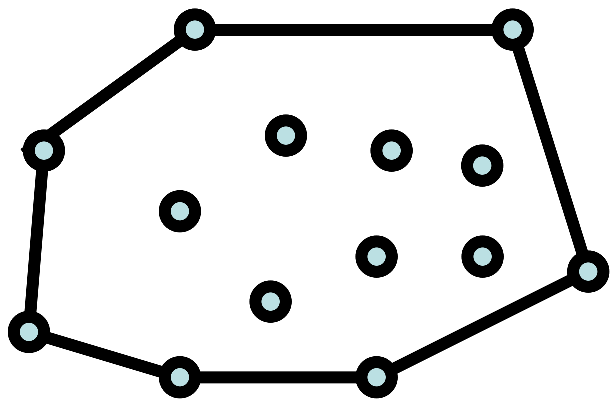
输出：CH(Q): Q的convex hull

Q的convex hull是一个最小凸多边形P，Q的点或者在P上或者在P内

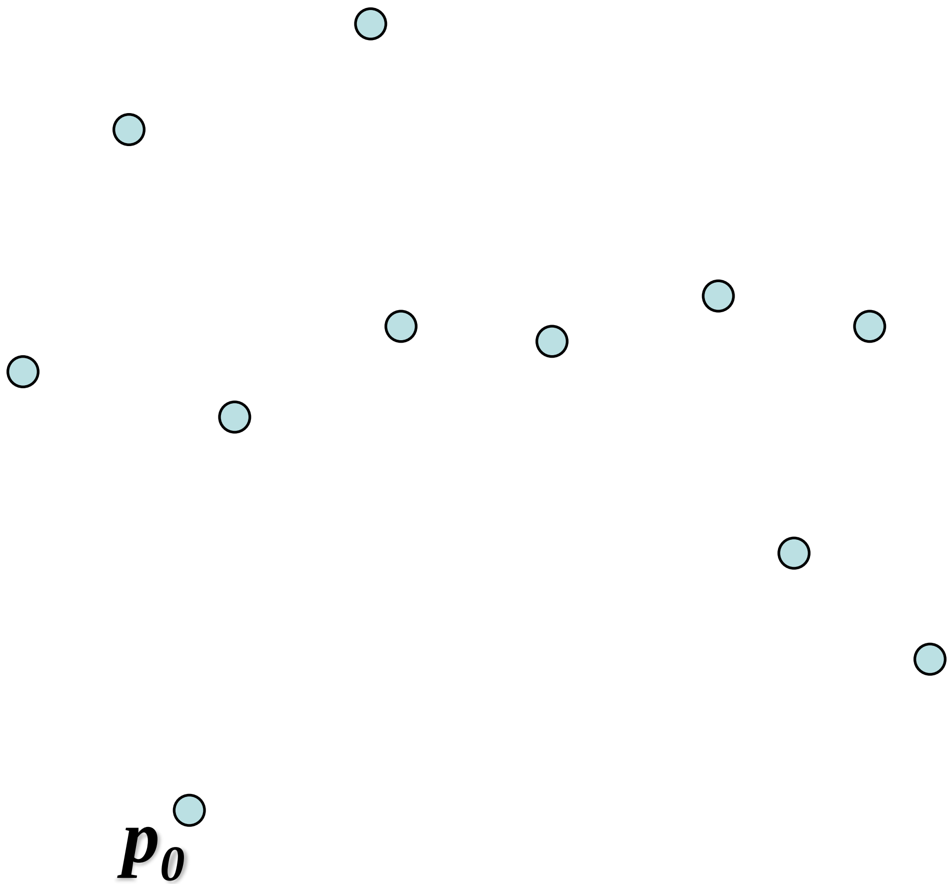
凸多边形P是具有如下性质多边形：连接P内任意两点的边都在P内

Graham-Scan算法

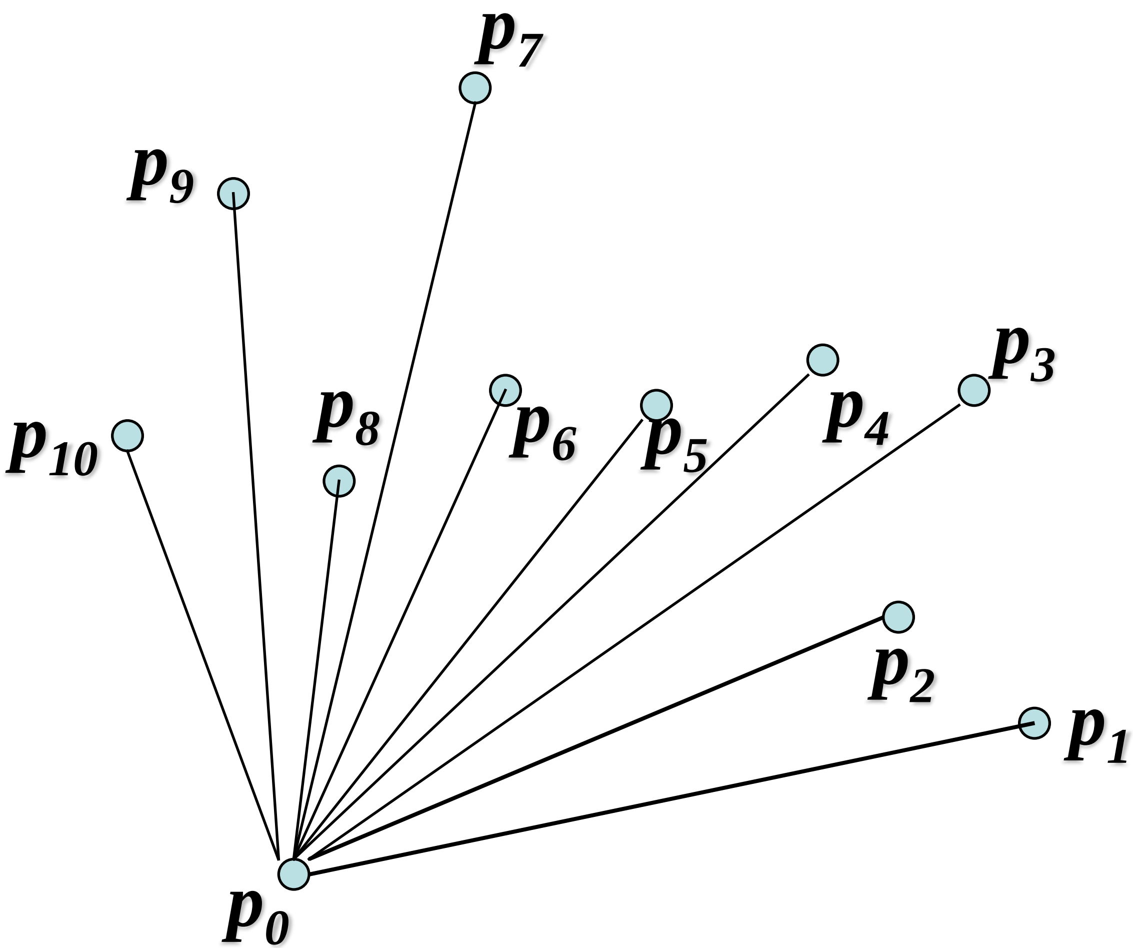
- 基本思想
  - 当沿着Convex hull逆时针漫游时，总是向左转
  - 在极坐标系下按照极角大小排列，然后逆时针方向漫游点集，去除非Convex hull顶点(非左转点)。



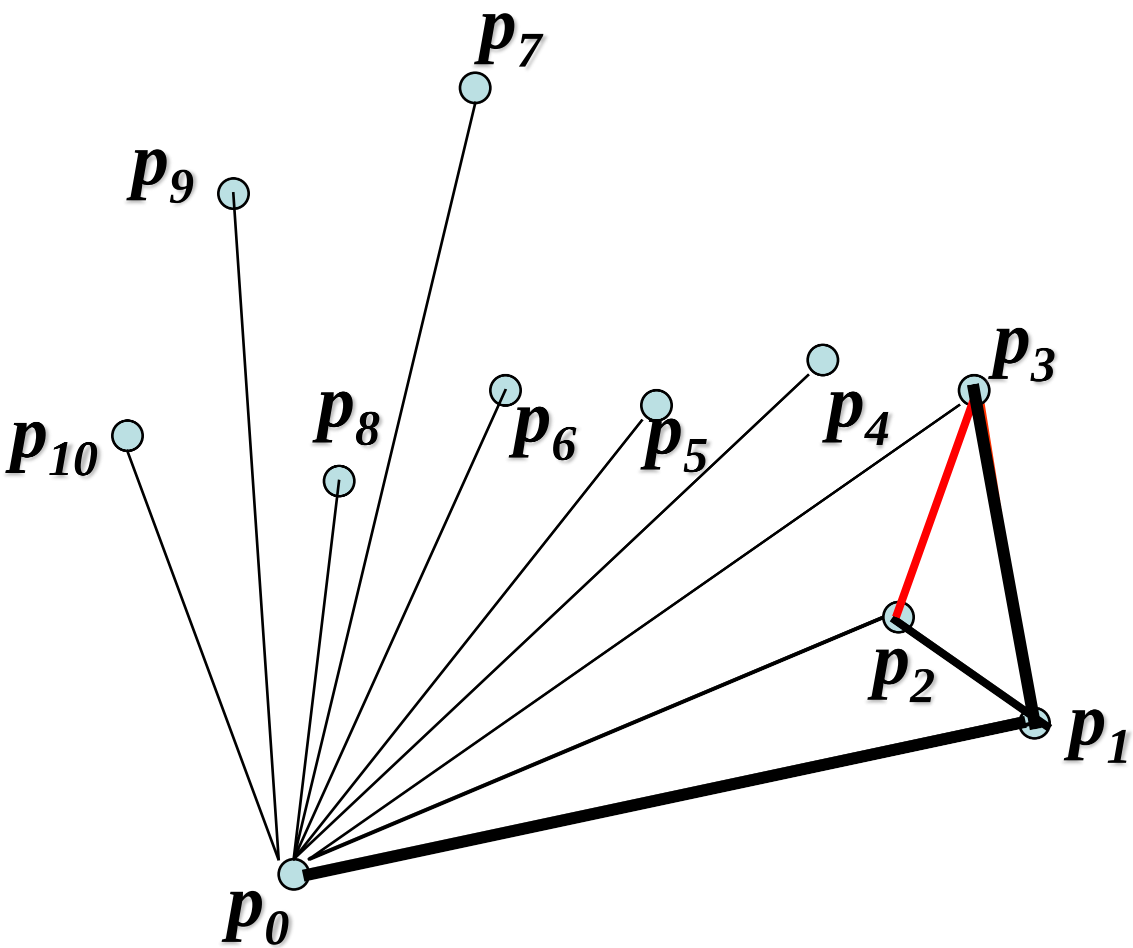
第1步

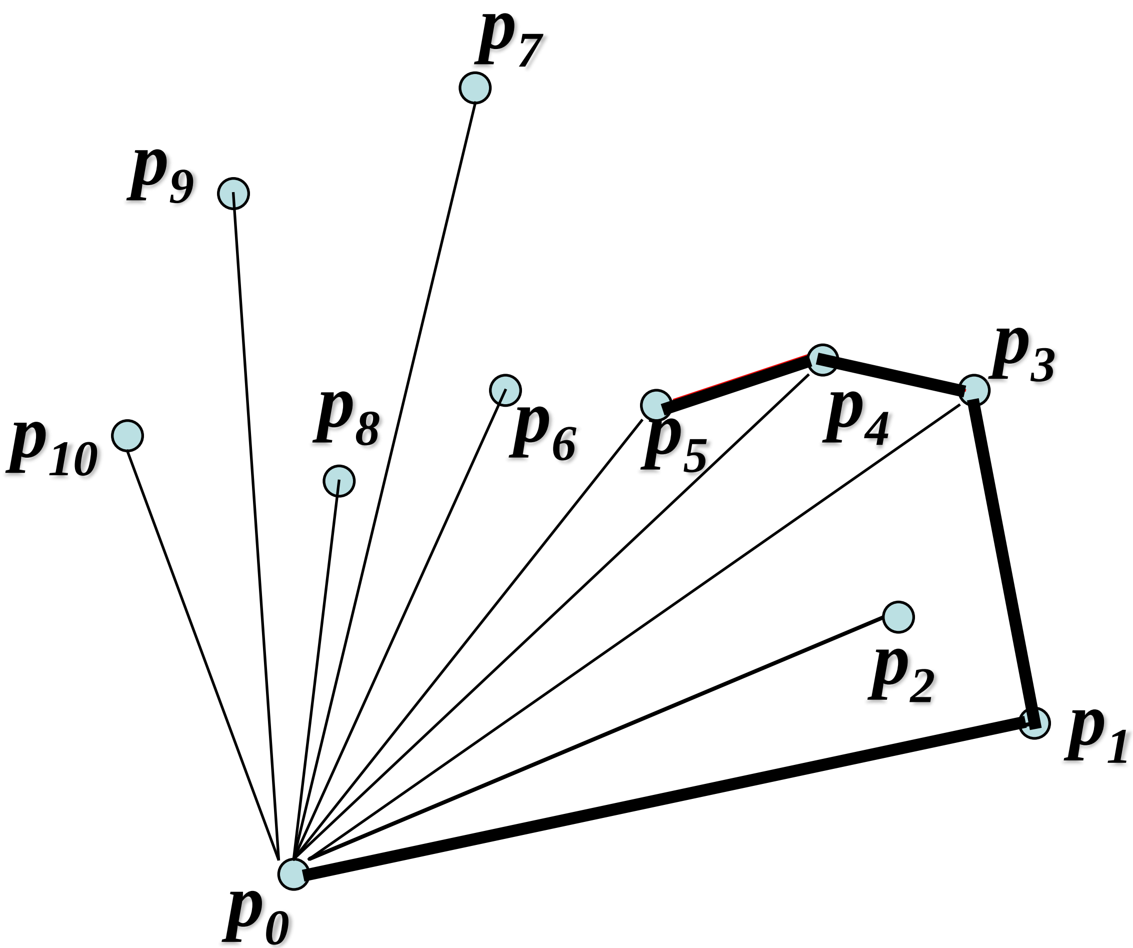


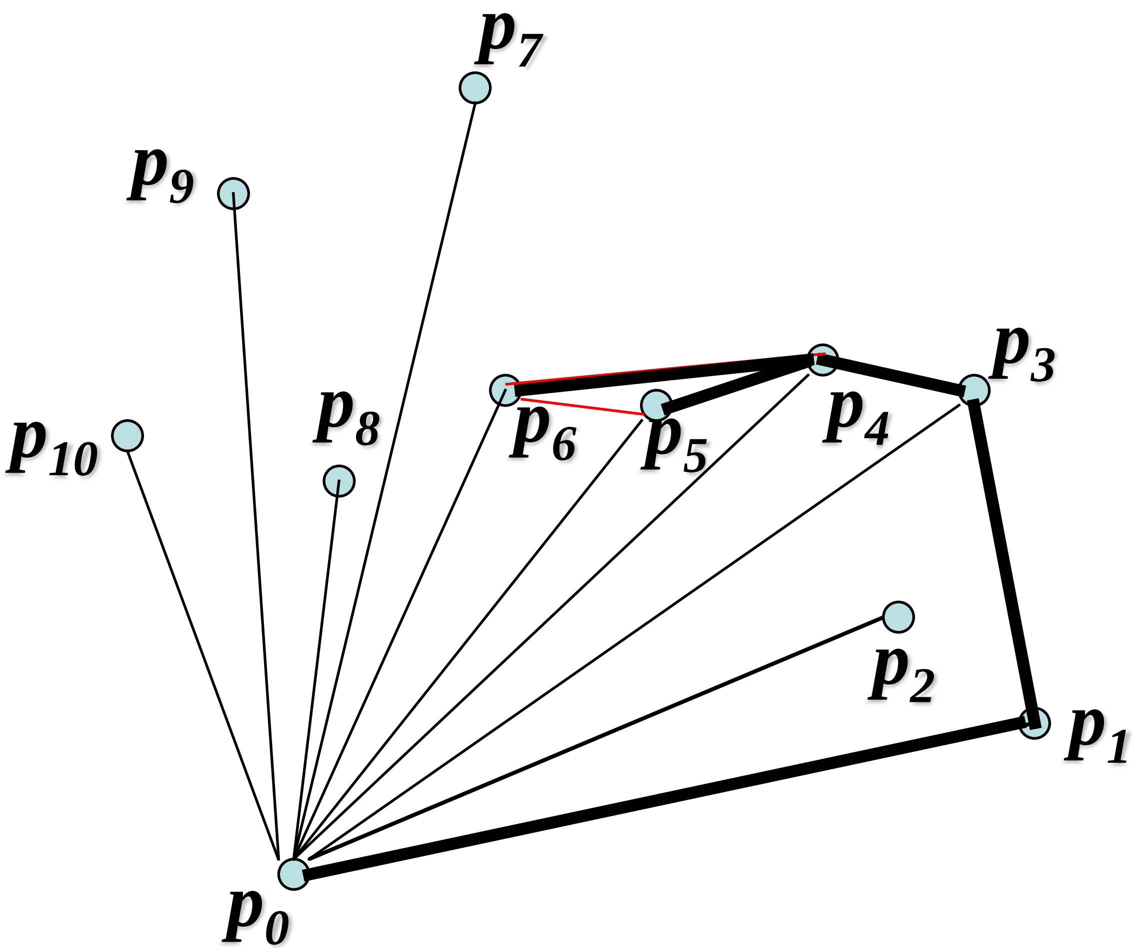
第2步



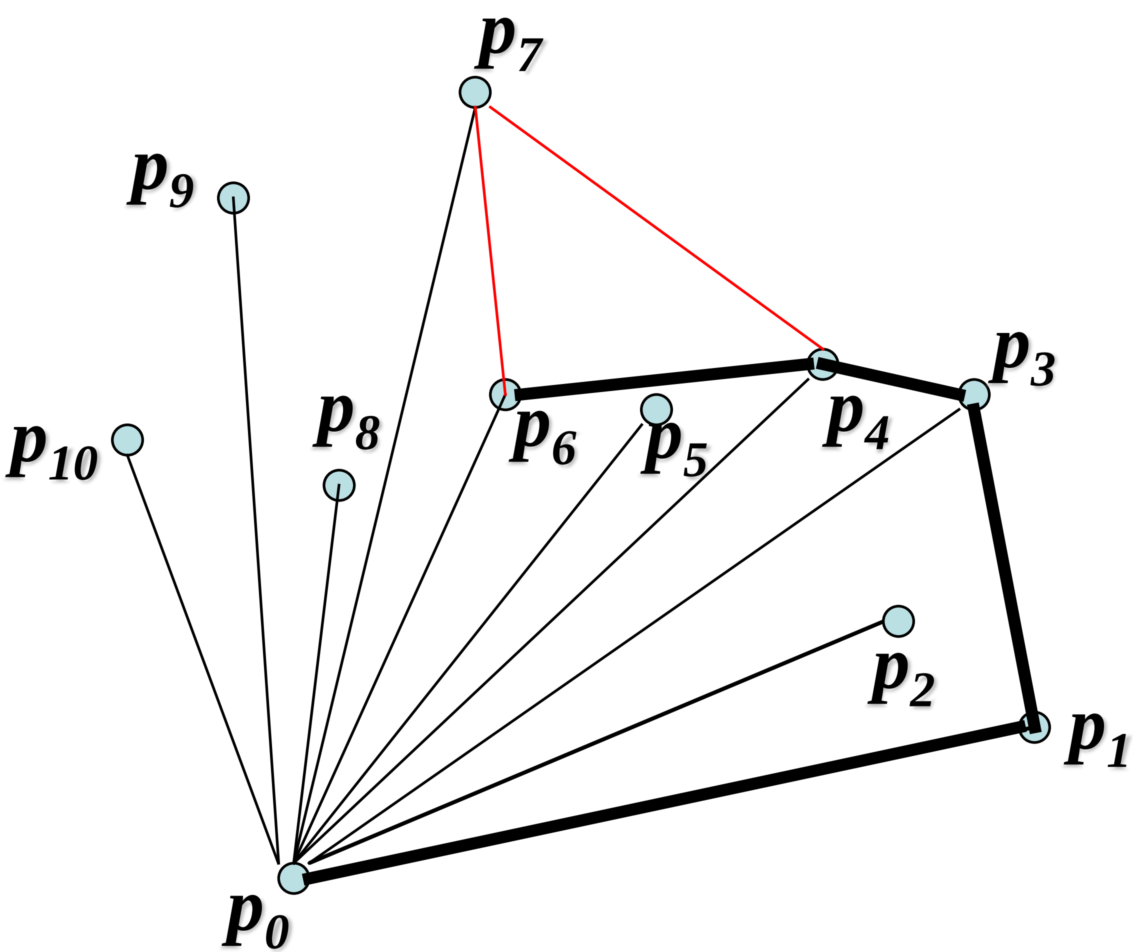
第3步



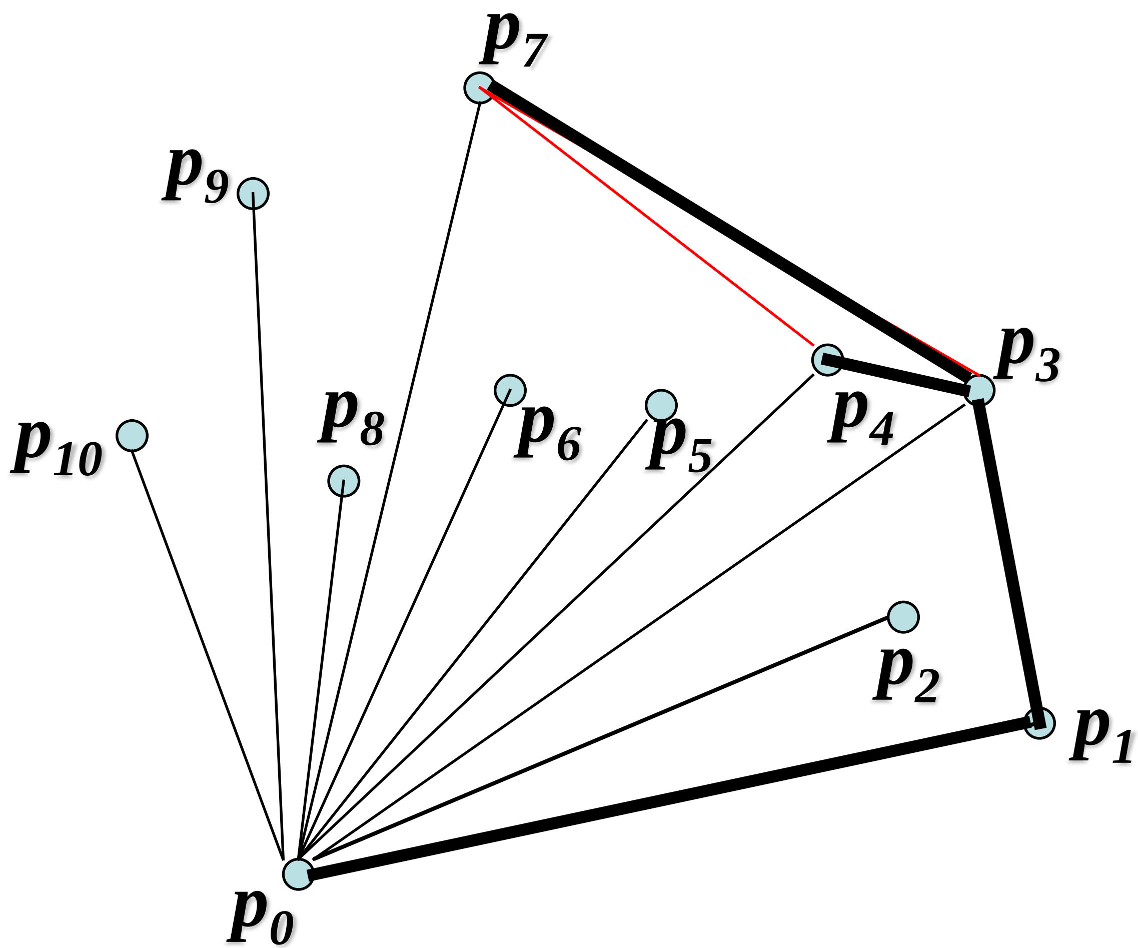

p_7
p_9
p_3
p_{10}
p_8
p_6
p_5
p_4
p_2
p_1
p_0

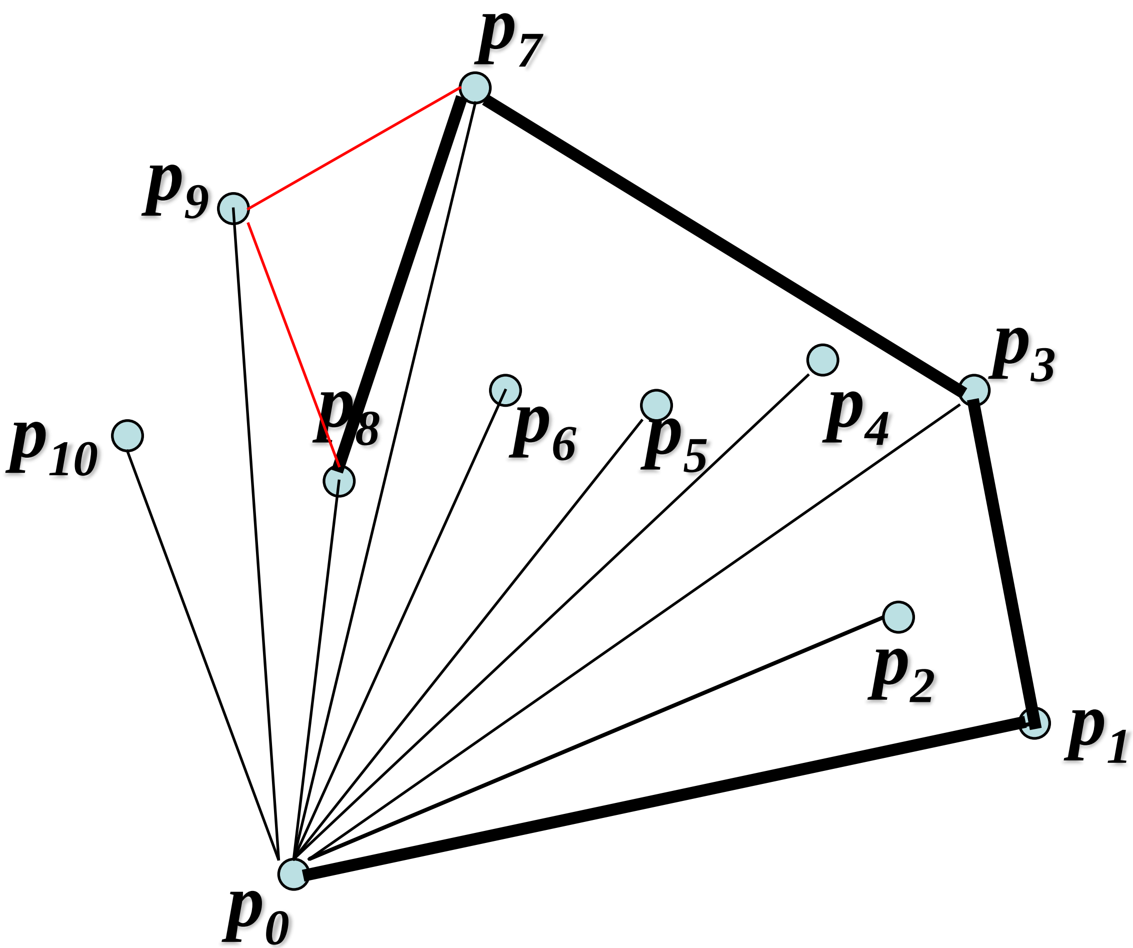

p_7
p_9
p_8
p_{10}
p_6
p_5
p_4
p_3
p_2
p_1
p_0

HITWH
SE

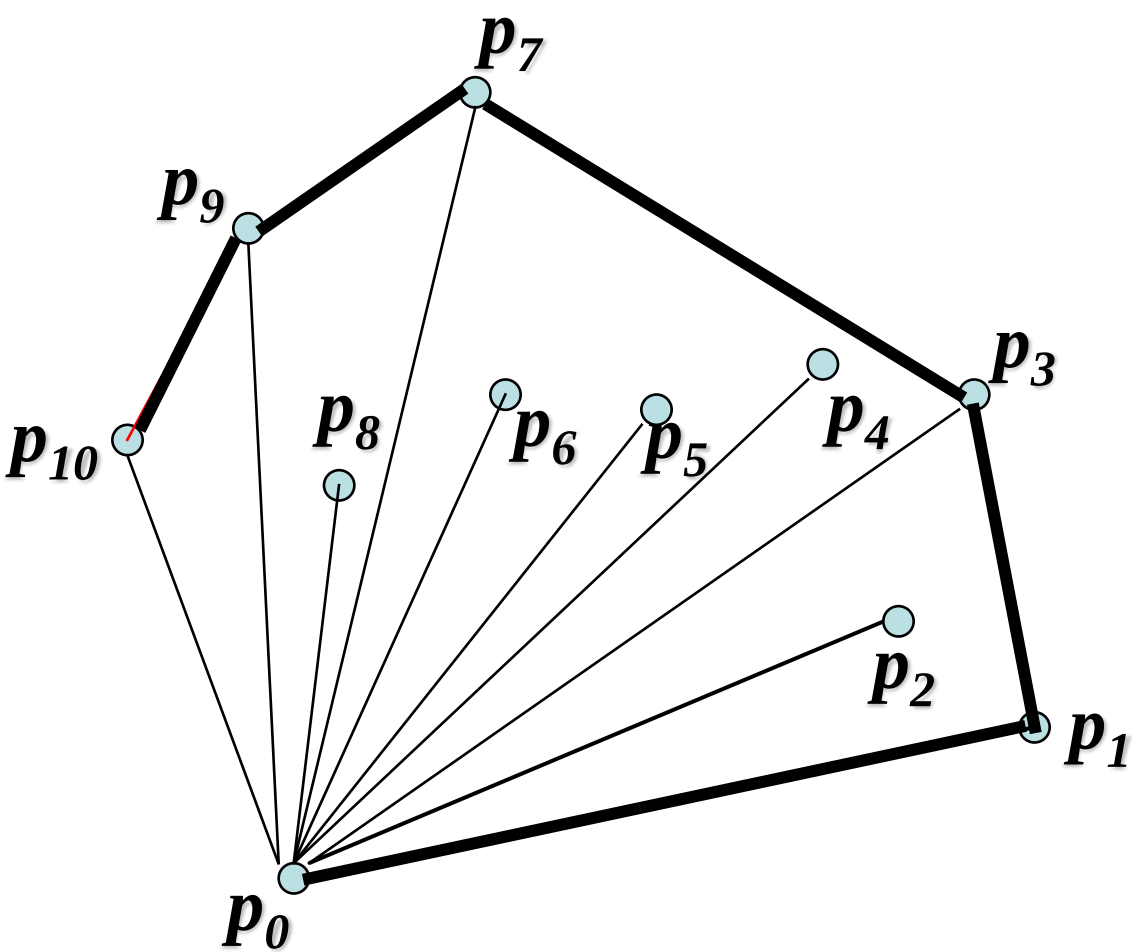


HITWH
SE




p7
p9
p3
p8
p6
p5
p4
p10
p2
p1
p0

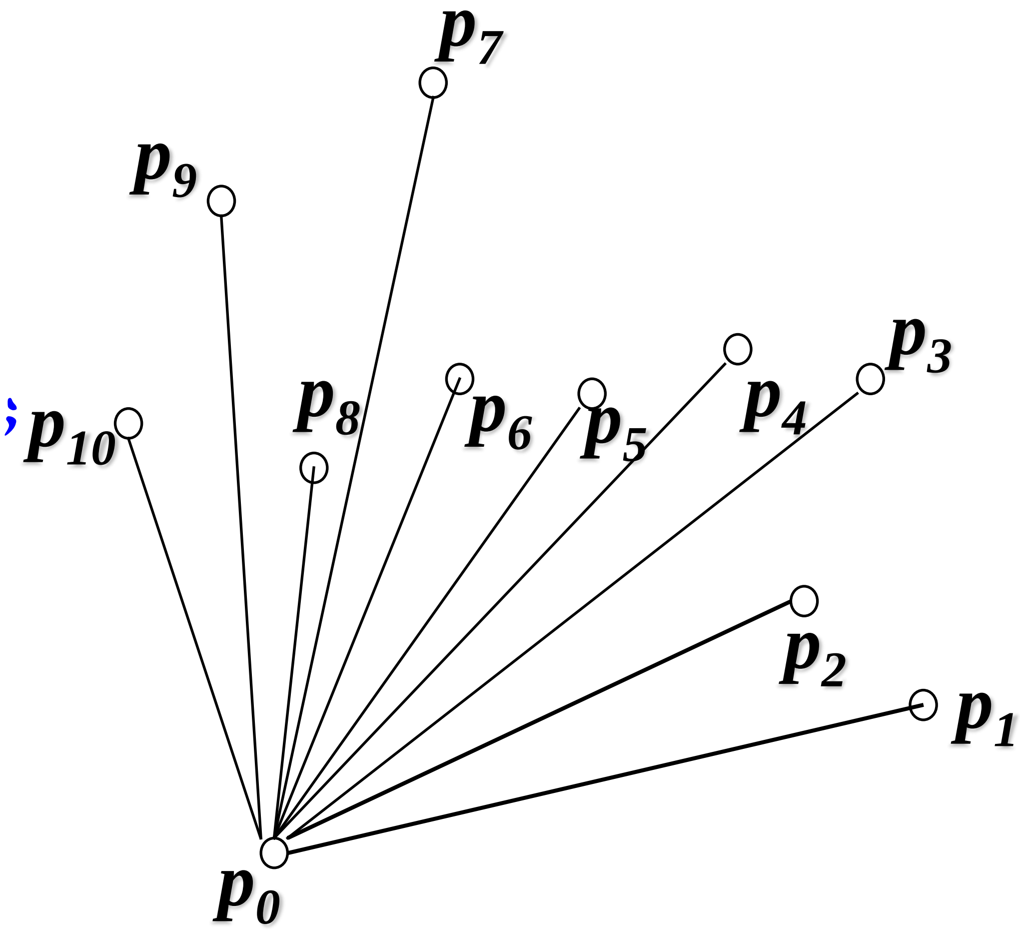
HITWH
SE



算法Graham-Scan(Q)

/* 栈S从底到顶存储按逆时针方向排列的CH(Q)顶点 */

1. 求Q中y-坐标值最小的点p_0;
2. 按照与p_0极角(逆时针方向)大小排序Q中其余点，结果为$<p_1, p_2, \ldots, p_n>$;
3. Push p_0, p_1, p_2 into S;
4. FOR i=3 TO n DO
5. While Next-to-top(S)、Top(S)和p_i形成非左移动 Do
6. Pop(S);
7. Push(p_i, S);
8. Rerurn S.



- 时间复杂性$T(n)$

1. 求Q中y-坐标值最小的点p_0;
2. 按照与p_0极角(逆时针方向)大小排序Q中其余点，结果为$<p_1, p_2, \ldots, p_n>$;
3. Push p_0, p_1, p_2 into S;
4. **FOR** i=3 TO n DO
5. **While** Next-to-top(S)、Top(S)和p_i形成非左移动 **Do**
6. Pop(S);
7. Push(p_i, S);
8. Rerurn S.

- 第1步需要$O(n)$时间
- 第2步需要$O(n\log n)$时间
- 第3步需要$O(1)$时间
- 第4-7步需要$O(n)$时间
  - 因为每个点至多进栈一次出栈一次，每次需要常数计算时间
- $T(n)= O(n\log n)$

- 正确性分析

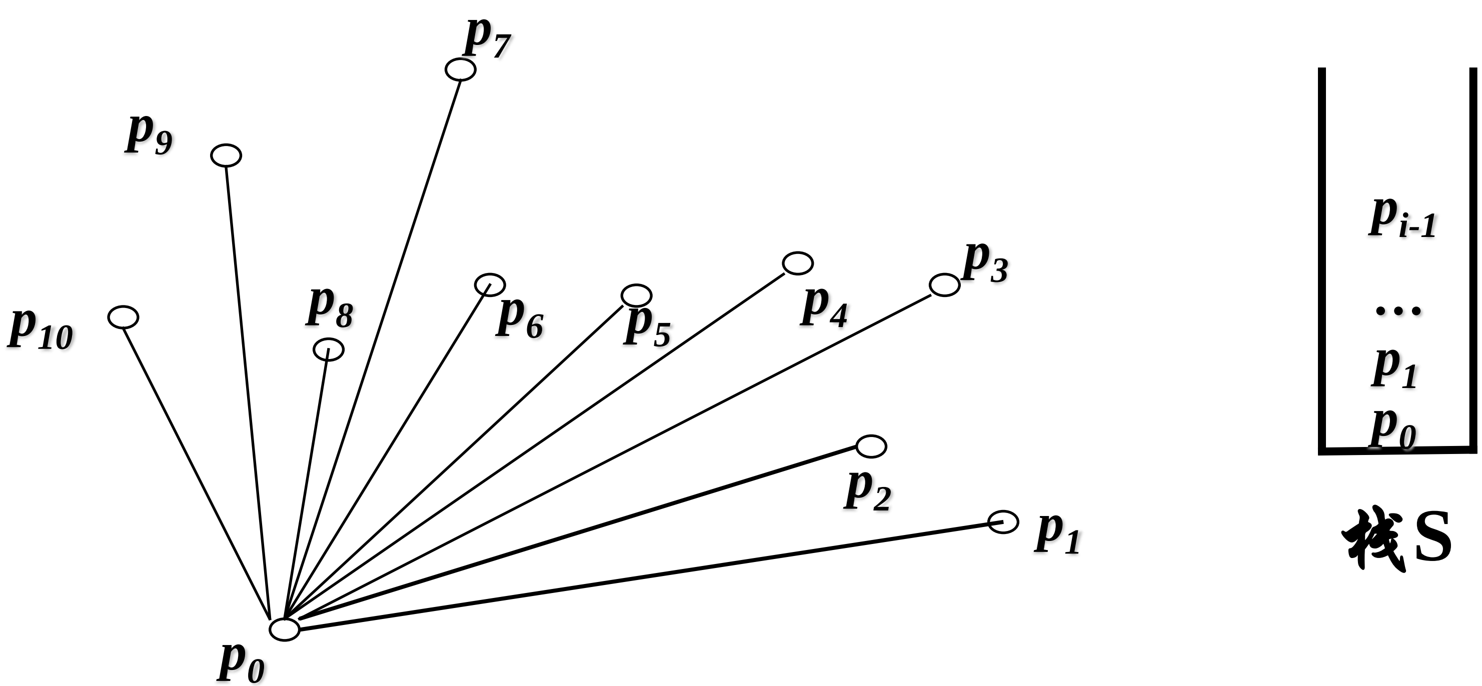
定理. 设n个二维点的集合Q是Graham-Scan算法的输入，$|Q|\geq 3$，算法结束时，栈S中自底到顶存储CH(Q)的顶点（按照逆时针顺序）.

证明: 使用循环不变量方法

```
Push p0, p1, p2 into S;
FOR  i=3  TO  n  DO
    While  Next-to-top(S)、Top(S)
          和pi形成非左移动  Do
              Pop(S);
    Push(pi, S);
```

Loop invariant

在处理第i个顶点之前，栈S中自底到顶存储$CH(Q_{i-1})$的顶点.



```
3.  Push p0, p1, p2 into S;
4.  FOR  i=3  TO  n  DO
5.     While  Next-to-top(S)、Top(S)
              和pi形成非左移动  Do
6.                Pop(S);
7.     Push(pi, S);
```

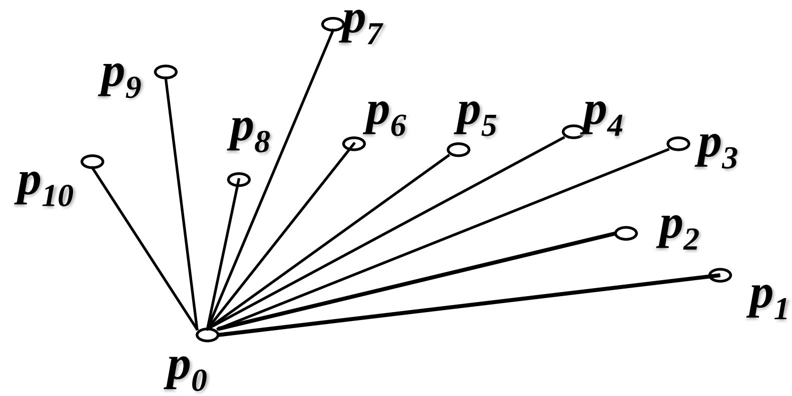
循环不变量

在处理第i个顶点之前，栈S自底到顶存储CH(Q_{i-1})的顶点.

- Proof by induction
  - Initialization: (第3步)
    - 处理i=3之前．栈S中包含了$Q_{i-1}=Q_2=\{p_0,p_1,p_2\}$中的顶点，这三个点形成了一个CH. 循环不变量为真.
  - Maintenance:
    - 设在处理第$i(i\geq 3)$个顶点之前，循环不变量为真，即：栈S中自底到顶存储CH(Q_{i-1})的顶点.
    - 往证：

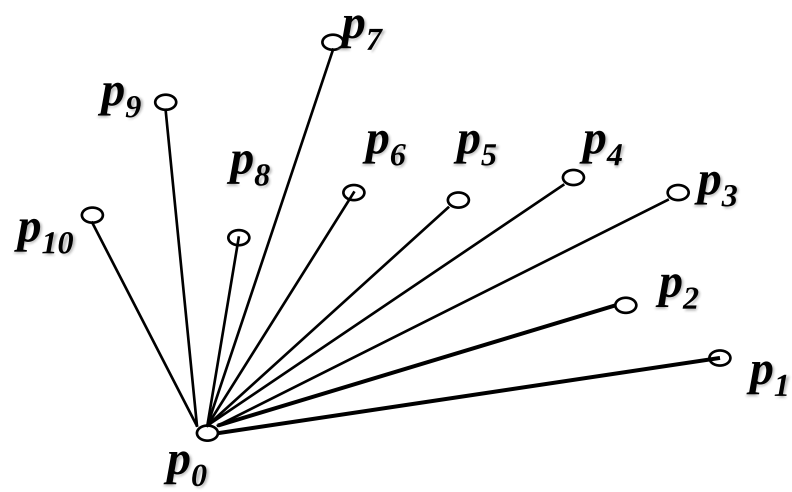
      算法执行5~7步之后，栈S中自底到顶存储CH(Q_i)的顶点.

3. Push p_0, p_1, p_2 into S;
4. **FOR** i=3 TO n DO
5. **While** Next-to-top(S)、Top(S) 和p_i形成非左移动 **Do**
6. Pop(S);
7. Push(p_i, S);



- 往证：算法执行5~7步后，栈S中自底到顶存储CH(Q_i)的顶点
  - 5~6步while循环执行结束后，第7步将p_i压入栈之前，设栈顶元素为p_j，次栈顶元素为p_k，则此时，栈中包含了与for循环的第j轮迭代后相同的顶点，即CH(Q_j)，因为第j轮迭代后循环不变量为真.
  - 执行第7步之后，p_i入栈，则栈S中包含了CH(Q_j) $\cup\{p_i\}$=CH($Q_j\cup\{p_i\}$)中的顶点，且这些点仍按逆时针顺序，自底向上出现在栈中. CH($Q_j\cup\{p_i\}$)= CH(Q_i)?
  - 对于任意一个在第i轮迭代中被弹出的栈顶点p_t，设p_r为紧靠p_t的次栈顶点，p_t被弹出当且仅当p_r、p_t、p_i构成非左移动。因此，p_t不是CH(Q_i)的一个顶点，即CH($Q_i-\{p_t\}$)= CH(Q_i)，以此类推，可知第i轮迭代中被弹出的点都不是CH(Q_i)的一个顶点。
  - 设P_t为for循环第t轮迭代中被弹出的所有点的集合，则有CH(Q_i-P_i)= CH(Q_i)。
  - 对于$j+1 \le t \le i-1$，由于第t轮迭代后循环不变量为真，可知P_t不在CH(Q_t)中，继而必然不在CH(Q_i)中，因此$\mathrm{CH}(Q_i) = \mathrm{CH}(Q_i - P_i) = CH\left(Q_i - P_i - \bigcup_{t=j+1}^{i-1} P_t\right)$。
  - 又 $Q_i - P_i - \bigcup_{t=j+1}^{i-1} P_t = Q_j \cup \{p_i\}$，故有CH($Q_j\cup\{p_i\}$)= CH($Q_i - P_i - \bigcup_{t=j+1}^{i-1} P_t$)= CH($Q_i$)
  - 即得到：一旦将p_i压入栈后，栈S中恰包含CH(Q_i)中的顶点，且按照逆时针顺序，自底向上排列。

3. Push p_0, p_1, p_2 into S;
4. **FOR** i=3 TO n DO
5. **While** Next-to-top(S)、Top(S) 和p_i形成非左移动 **Do**
6. Pop(S);
7. Push(p_i, S);



– Termination:

- $i=n+1$,栈S中自底到顶存储CH(Q_n)的顶点,算法正确.

证毕.

Divide-and-conquer算法

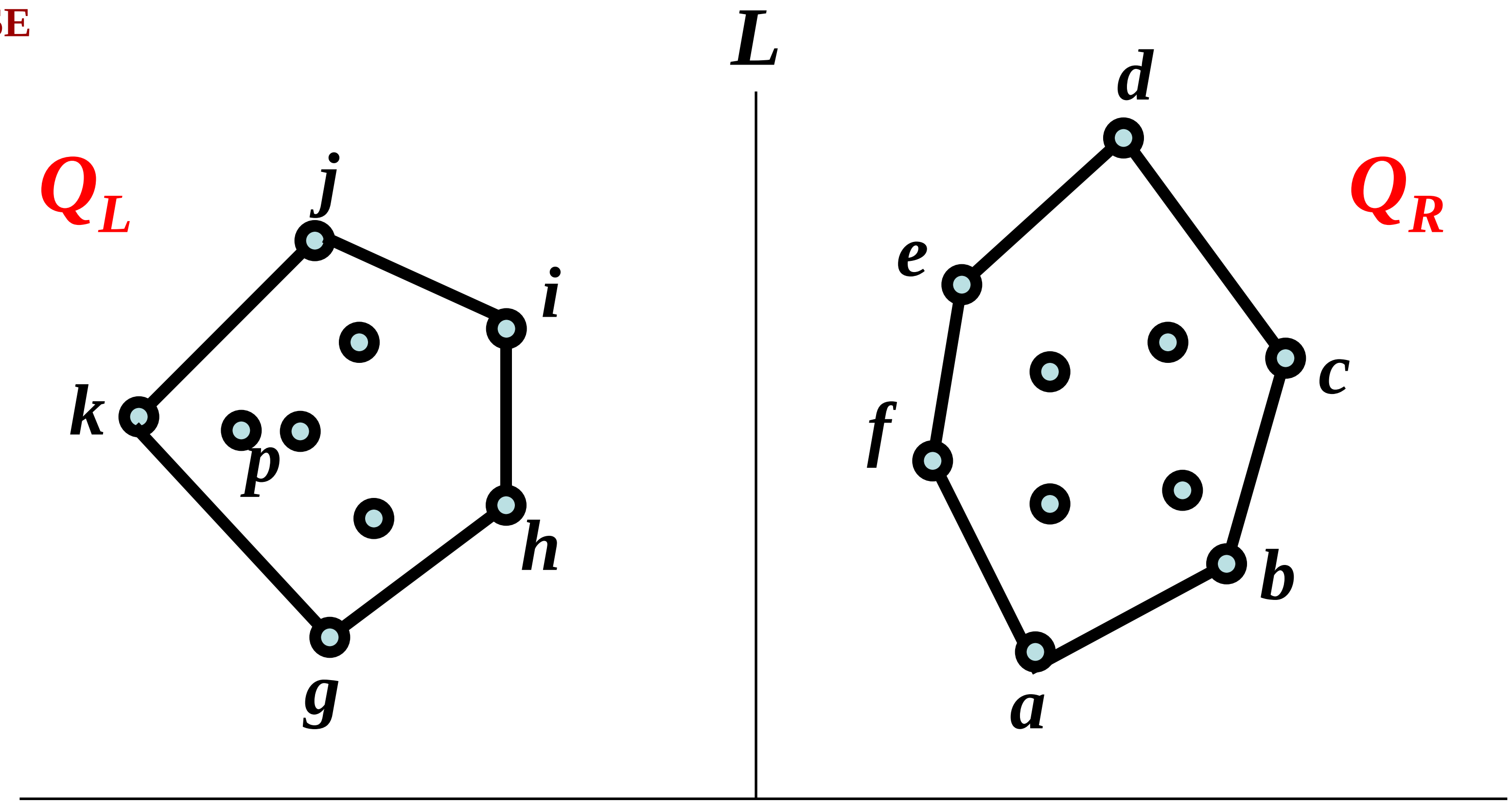
边界条件：(时间复杂性为$O(1)$)

1. 如果|Q|<3, 算法停止;
2. 如果|Q|=3, 按照逆时针方向输出CH(Q)的顶点;
3. 如果|Q|<5, 使用Graham-Scan求$CH(Q)$;

Divide:(使用$O(n)$算法求中值)

1. 选择一个垂直于x-轴的直线把Q划分为基本相等的两个集合Q_L和Q_R, Q_L在Q_R的左边;

HITWH
SE



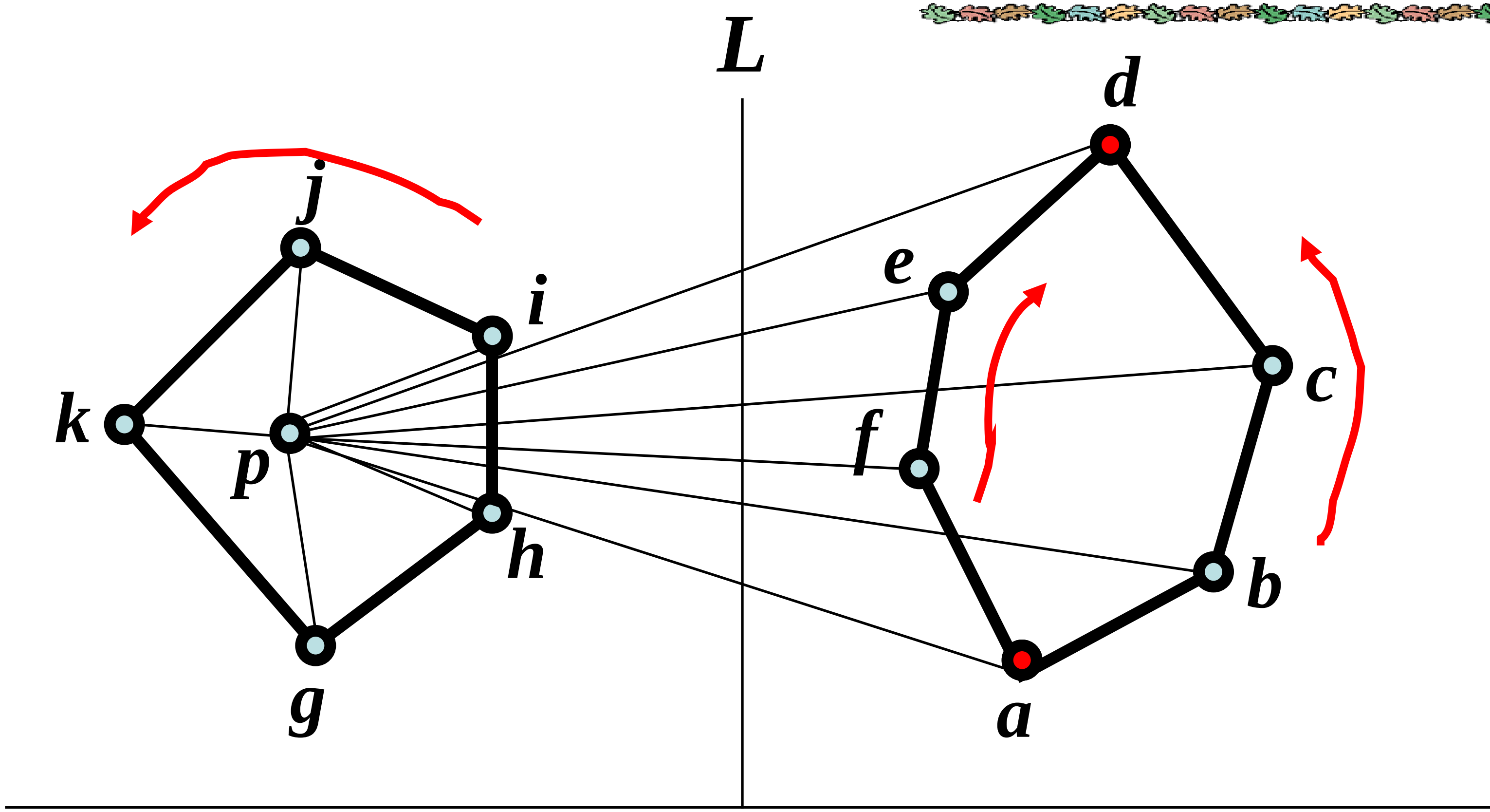
Conquer: (时间复杂性为$2T(n/2)$)

1. 递归地为Q_L和Q_R构造$CH(Q_L)$和$CH(Q_R)$;

Merge:

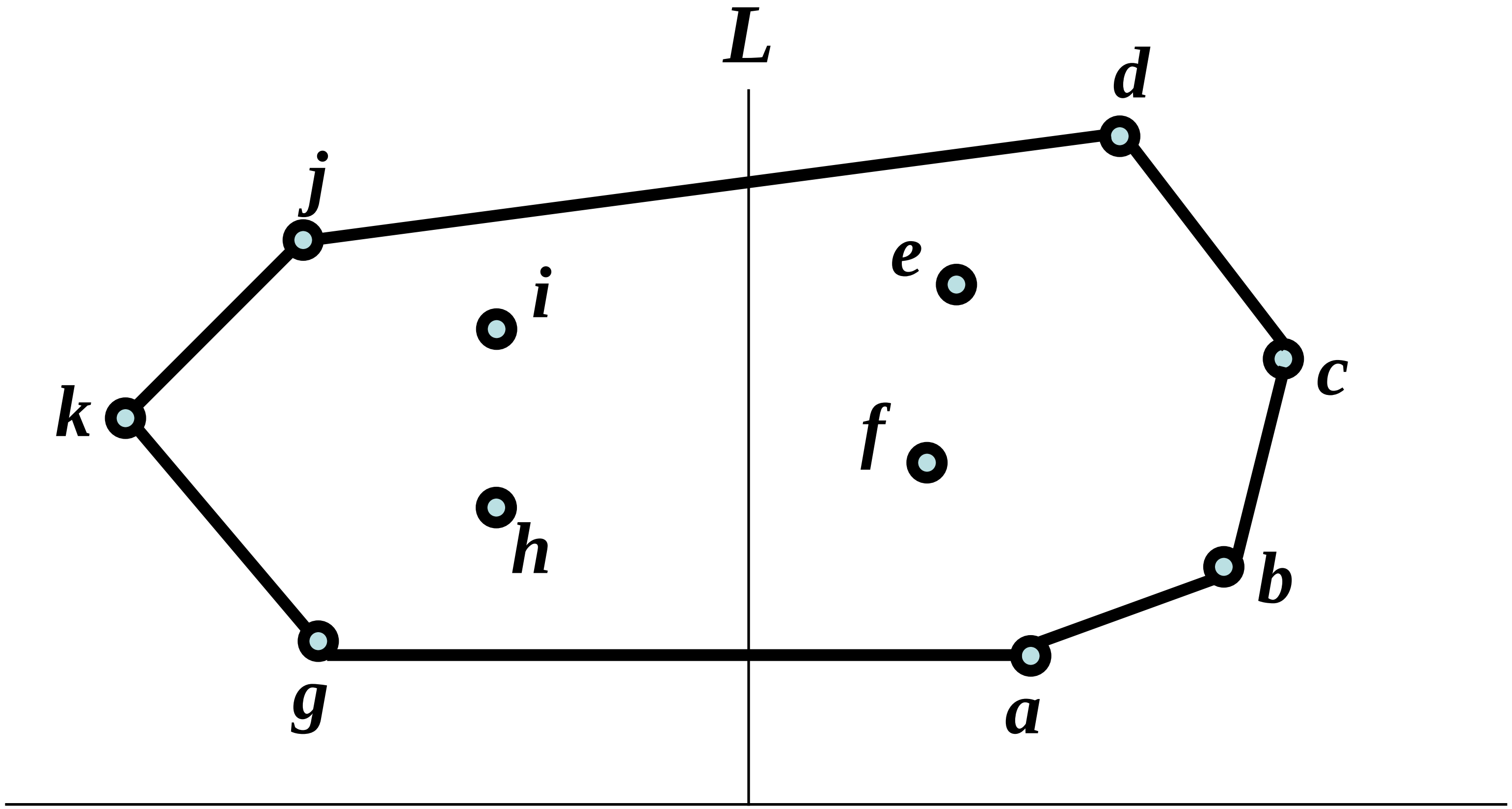
我们先通过一个例子来看Merge的思想

Merge实例



3个序列：<g, h, i, j, k>, <a, b, c, d>, <f, e>

合并以后：<g, h, a, b, f, c, e, d, i, j, k>



选择p点为y坐标最小的点

在合并的序列上运行Graham-Scan，排序部分合并三个有序序列即可

Merge:(时间复杂性为$O(n)$)

1. 找到Q_L和Q_R中y坐标最小的点p (假设在Q_L中);
2. 在CH(Q_R)中找与p的极角最大和最小顶点u和v;
3. 构造如下三个点序列:
   (1) 按逆时针方向排列的CH(Q_L)的所有顶点,
   (2) 按逆时针方向排列的CH(Q_R)从v到u的顶点,
   (3) 按顺时针方向排列的CH(Q_R)从v到u的顶点;
4. 合并上述三个序列;
5. 在合并的序列上应用Graham-Scan.

时间复杂性

- Preprocessing阶段
  - $O(1)$
- Divide阶段(使用$O(n)$算法求中值)
  - $O(n)$
- Conquer阶段
  - $2T(n/2)$
- Merge阶段
  - $O(n)$

时间复杂性

- 总的时间复杂性

  $T(n)=2T(n/2)+O(n)$

- 使用Master定理

  $T(n)=O(n\log n)$

3.7 快速傅里叶变换

问题定义

输入：$a_0, a_1, \ldots, a_{n-1}$，$a_i$是实数，$(0 \le i \le n-1)$

输出：$A_0, A_1, \ldots, A_{n-1}$，使得，

$$A_j = \sum_{k=0}^{n-1} a_k e^{\frac{2\pi ijk}{n}}$$

其中：

(2) $0 \le j \le n-1$

(3) e是自然对数的底数

(4) $i = \sqrt{-1}$ 是虚数单位

蛮力法利用定义计算每个A_j，时间复杂度为$\Theta(n^2)$

算法的数学基础

$$A_j = \sum_{k=0}^{n-1} a_k e^{\frac{2\pi ijk}{n}}$$ 令 $w_n = e^{\frac{2\pi i}{n}}$， 有： $A_j = \sum_{k=0}^{n-1} a_k w_n^{jk}$

$$A_j = a_0 + a_1 w_n^{j} + a_2 w_n^{2j} + a_3 w_n^{3j} + a_4 w_n^{4j} + \ldots\ldots + a_{n-2} w_n^{(n-2)j} + a_{n-1} w_n^{(n-1)j}$$

$$= (a_0 + a_2 w_n^{2j} + a_4 w_n^{4j} + \ldots\ldots + a_{n-2} w_n^{(n-2)j})$$

$$+ (a_1 w_n^{j} + a_3 w_n^{3j} + a_5 w_n^{5j} + \ldots\ldots + a_{n-1} w_n^{(n-1)j})$$

$$= (a_0 + a_2 w_n^{2j} + a_4 w_n^{4j} + \ldots\ldots + a_{n-2} w_n^{(n-2)j})$$

$$+ w_n^{j} (a_1 + a_3 w_n^{2j} + a_5 w_n^{4j} + \ldots\ldots + a_{n-1} w_n^{(n-2)j})$$

HITWH
SE

算法的数学基础

$$A_j=(a_0+a_2w_n^{2j}+a_4w_n^{4j}+\ldots\ldots+a_{n-2}w_n^{(n-2)j})$$

奇数输入项的FFT

$$+w_n^j(a_1+a_3w_n^{2j}+a_5w_n^{4j}+\ldots\ldots+a_{n-1}w_n^{(n-2)j})$$

偶数输入项的FFT

由于$w_n^2=w_{n/2}$,以及$w_n^{n+k}=w_n^k$,

$$A_j=\underbrace{(a_0+a_2w_{n/2}^{j}+a_4w_{n/2}^{2j}+\ldots\ldots+a_{n-2}w_{n/2}^{\frac{(n-2)j}{2}})}_{B_j}$$

$$+w_n^j\underbrace{(a_1+a_3w_{n/2}^{j}+a_5w_{n/2}^{2j}+\ldots\ldots+a_{n-1}w_{n/2}^{\frac{(n-2)j}{2}})}_{C_j}$$

将问题划分为两个子问题

于是，$A_j=B_j+w_n^jC_j$　还可证明，$A_{j+n/2}=B_j+w_n^{j+n/2}C_j$

分治算法过程

划分：将输入析分成$a_0,a_2,\ldots,a_{n\text{-}2}$和$a_1,a_3,\ldots,a_{n\text{-}1}$.

递归求解：递归计算$a_0,a_2,\ldots,a_{n\text{-}2}$的变换$B_0,B_1,\ldots,B_{n/2\text{-}1}$
递归计算$a_1,a_3\ldots,a_{n\text{-}1}$的变换$C_0,C_1,\ldots,C_{n/2\text{-}1}$

合并：根据$A_j= B_j+C_j\cdot W_n^{\ j}\ (j<n/2)$ 和$A_j=B_{j\text{-}n/2}+C_{j\text{-}n/2}\cdot W_n^{\ j}$
$(n/2\le j<n\text{-}1)$依次求得$A_0, A_1, \ldots, A_{n\text{-}1}$.

分治算法过程

例如：计算8个点$a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7$的FFT.

计算4个点 a_0, a_2, a_4, a_6的FFT.

计算4个点 a_1, a_3, a_5, a_7的FFT.

计算2个点 a_0, a_4的FFT.

计算2个点 a_2, a_6的FFT.

计算2个点 a_1, a_5的FFT.

计算2个点 a_3, a_7的FFT.

算法FFT($a_0, a_2, \ldots, a_{n-1}, n$)

输入： $a_0, a_1, \ldots, a_{n-1}$， $n=2^k$

输出： $a_0, a_1, \ldots, a_{n-1}$ 的傅里叶变换 $A_0, \ldots, A_{n-1}$

1. $W \leftarrow \exp(2\pi i/n)$;
2. If ($n=2$) Then
3、 $A_0 \leftarrow a_0 + a_1$;
4. $A_1 \leftarrow a_0 - a_1$;
5. 输出 A_0, A_1, 算法结束；
6. $B_0, B_1, \ldots, B_{n/2-1} \leftarrow$ FFT($a_0, a_2, \ldots, a_{n-2}, n/2$)；
7. $C_0, C_1, \ldots, C_{n/2-1} \leftarrow$ FFT($a_1, a_3, \ldots, a_{n-1}, n/2$)；
8. For $j=0$ To $n/2-1$
9. $A_j \leftarrow B_j + C_j \cdot W^j$;
10. $A_{j+n/2} \leftarrow B_j + C_j \cdot W^{j+n/2}$;
11. 输出 $A_0, A_1, \ldots, A_{n-1}$, 算法结束；

$T(n)=\theta(1)$ If $n=2$

$T(n)=2T(n/2)+\theta(n)$ If $n>2$

$T(n)=\theta(n\log n)$

$$T(n)=\theta(1) \quad \text{if } n\le c$$
$$T(n)=aT(n/b)+D(n)+C(n) \quad \text{if } n>c$$

3.8 整数乘法

优化划分阶段，降低$T(n)=aT(n/b)+f(n)$中的a

问题定义

输入：n位二进制整数X和Y
输出：X和Y的乘积

通常，计算X*Y时间复杂性为$O(n^2)$，
我们给出一个复杂性为$O(n^{1.59})$的算法。

算法的思想

	n/2位	n/2位
X=	A	B

	n/2位	n/2位
Y=	C	D

$$XY = (A2^{n/2} + B)(C2^{n/2} + D)$$
$$= AC2^{n} + AD2^{n/2} + BC2^{n/2} + BD$$
$$= AC2^{n} + ((A-B)(D-C)+AC+BD)2^{n/2} + BD$$

如此计算需要

$T(n)=4T(n/2) +O(n)=O(n^2)$

时间复杂性

$$T(n)=\theta(1) \qquad \text{if } n=1$$
$$T(n)=3T(n/2)+O(n) \qquad \text{if } n>1$$

使用Master定理

$T(n)=O(n^{\log_2 3}) =O(n^{1.59})$