常見數學方法

一、關於基礎數論

・**歐幾里得輾轉相除法**：若 a 除以 b 的餘數是 r，則 $(a,b)=(b,r)$。

・**篩法**：對所有$1 \le k \le n$，求出 (1)k 是不是質數 (2)k 的因數和、因數個數、質因數分解、φ(k)

・**模運算**

1.若 $a \equiv b \pmod m, c \equiv d \pmod m$, 則 $a+c \equiv b+d \pmod m$

2.若 $a \equiv b \pmod m, c \equiv d \pmod m$, 則 $ac \equiv bd \pmod m$

3.若 $ac \equiv bc \pmod m$, 則 $a \equiv b \left[\mathrm{mod}\,\frac{m}{(c,m)}\right]$

4.若 $(a,m)=1$, 則存在 c 使得 $ac \equiv 1 \pmod m$, 稱 c 為 a 對模 m 的逆(a^{-1})。

設 $ac = 1 - mq, q \in \mathbf{Z}$ 則 $ac + mq = 1 \Rightarrow$ 用輾轉相除法求出 c。也可用歐拉-費馬定理求模逆。

・**歐拉的 φ 函數**

1.定義：φ(n)為 $[1, n]$ 中與 n 互質的數的個數

2.若 p 為質數，則 φ(p)＝$p-1$，φ(p^k)＝$(p-1)\,p^{k-1}$，若 $(m,n)=1$，則 φ(mn)＝φ(m)φ(n)。

3.計算：若把 n 質因數分解為$n = \mathrm{p_1}^{\mathrm{e_1}}\mathrm{p_2}^{\mathrm{e_2}} \dots \mathrm{p_k}^{\mathrm{e_k}}$，則 $\varphi(n) = n\left(1-\frac{1}{\mathrm{p_1}}\right)\left(1-\frac{1}{\mathrm{p_2}}\right)\dots\left(1-\frac{1}{\mathrm{p_k}}\right)$

・**歐拉-費馬定理**

1.定理：若 $(a,m)=1$，則有 $a^{\varphi(m)} \equiv 1 \pmod m$。當 m 為質數時即為費馬小定理。

2.證明：令 $\mathrm{r_1, r_2, \dots, r_{\varphi(m)}}$為 m 的一個簡化剩餘系，因為$(a,m)=1$，所以 $a\mathrm{r}_1, a\mathrm{r}_2, \dots, a\mathrm{r}_{\varphi(m)}$也是 m 的簡化剩餘系 $\Rightarrow \mathrm{r_1 r_2 \dots r_{\varphi(m)}} \equiv (a\mathrm{r}_1)(a\mathrm{r}_2)\dots\left(a\mathrm{r}_{\varphi(m)}\right) \pmod m \Rightarrow 1 \equiv a^{\varphi(m)} \pmod m$

3.應用：RSA 加密：令 n＝pq, p 及 q 皆為質數，找出 e, d 使得 $ed \equiv 1\,[\mathrm{mod}\,\varphi(n)]$，則 e 為私密密鑰，d 為公開密鑰。加密時計算 $\mathrm{C} \equiv \mathrm{M}^e \pmod n$，解密時計算 $\mathrm{M} \equiv \mathrm{C}^d \pmod n$。

二、矩陣

數學上，一個 m×n 矩陣乃一 m 行 n 列的矩形陣列，由數組成，常見於線性代數、線性規劃、統計分析，以及組合數學等。其表示法為：

$A = A_{m\times \mathrm{n}} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}_{m\times n}$，或 $A = [a_{ij}]_{m\times n}$。定義單位矩陣 $I_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}_{n\times n}$。

加減法　：$A_{m\times n} \pm B_{m\times n} = C_{m\times n} = [a_{ij} \pm b_{ij}]_{m\times n}$

純量積　：$rA_{m\times n} = C_{m\times n} = \left[ra_{ij}\right]_{m\times n}$

乘法　：$A_{m\times n} \times B_{n\times p} = C_{m\times p} = \left[\sum_{k=1}^{m} a_{ik} \times b_{kj}\right]_{m\times p}$ **(※矩陣乘法不具交換律)**

左、右分配律：$A(B+C) = AB + AC$、$(B+C)A = BA + BC$

結合律　：$(AB)C = A(BC) = ABC$

反矩陣　：若對方陣 A，存在方陣 B 使得$AB = BA = I$，則稱 B 為 A 的反矩陣，記為 A^{-1}。

矩陣還可以用來加速特定 DP 的狀態轉移。

・高斯消元法

對於解方程組：$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + \ a_{22}x_2 + \ \dots + a_{2n}x_n = b_2 \\ \qquad\qquad\vdots \\ a_{n1}x_1 + \ a_{n2}x_2 + \ \dots + a_{nn}x_n = b_n \end{cases}$

透過 (1) 調換任兩個方程

(2) 把某一個方程同乘以一個非零的倍數

(3) 把某一個方程乘以一個倍數加到令一列

其複雜度為 $O(n^3)$。我們可以解出 $x_1, x_2, \dots, x_n$。該方程組也可以想成

$$\mathrm{AX} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} = B$$

所以只要求出 A^{-1}，就可以用 $A^{-1}AX = X = A^{-1}B$ 求出 X。求出 A^{-1} 可以用

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & 1 & 0 & \cdots & 0 \\ a_{21} & a_{22} & & a_{2n} & 0 & 1 & & 0 \\ \vdots & & \ddots & \vdots & \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

然後逐步用列運算把 A 消成單位矩陣

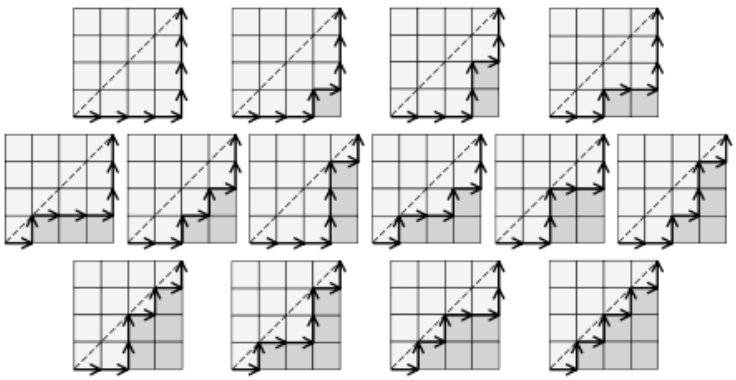
$$\begin{bmatrix} 1 & 0 & \cdots & 0 & b_{11} & b_{12} & \cdots & b_{1n} \\ 0 & 1 & & 0 & b_{21} & b_{22} & & b_{2n} \\ \vdots & & \ddots & \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

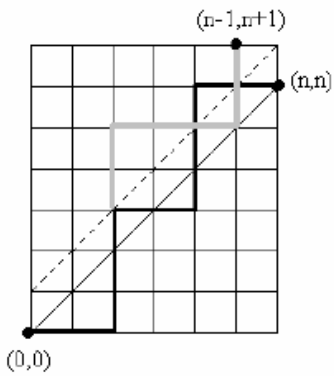
即可，時間複雜度也是 $O(n^3)$。因為每一列運算都相當於在 A 左邊乘上一個矩陣，若設那些矩陣為 $r_1, r_2, \dots, r_k$，既然 $r_k r_{k-1} \dots r_1 A = I$，所以 $r_k r_{k-1} \dots r_1 I = A^{-1}$。這個做法本質上跟高斯消元法一樣。當然，高斯消元法還可以用來求行列式值。高斯消元法也可以在模 m 下運算。

三、卡特蘭數

（以下皆取自 2009 年第七週 surwdkgo 學長講義）

考慮一系列網格如下圖，要從網格左下角的(0,0)走到右上角的(n,n)，只能往左或往上走而中途不跨越對角線，此種走法的方法數定義為卡特蘭數 Cn，其前幾項如下：{1, 1, 2, 5, 14, 42, 132, ⋯ }。





為了求出卡特蘭數的遞迴關係，我們考慮以下的方法：考慮第一次碰到對角線的點，我們可以把路徑拆成碰到前及碰到後共兩段，而在碰到這個點之前所有點均無碰到對角線，故可以將碰到前的那段刪去第一步及最後一步，則這兩條路徑均為符合中途不跨過對角線之路徑，且網格大小之和為 n-1，易檢驗其為一一對應關係，故可得遞迴式如下：$C_0 = 1, C_{n+1} = \sum_{k=0}^{n} C_k C_{n-k}$，這個遞迴式也提供了一個 $O(n^2)$的方法來計算。

接著我們考慮卡特蘭數的生成函數 $f(x)$，觀察其遞迴式，可得 $f(x) = x\,[f(x)]^2 + 1$，解得 $f(x) = \frac{1-\sqrt{1-4x}}{2x}$。其實除了遞迴關係外，卡特蘭數也是可以求出封閉表達式的，方法如下：不跨越對角線的走法數 = 所有走法數- 跨越對角線的走法數。而跨越對角線的走法數可以用一個對應來計算(如右上圖)，考慮第一次跨越對角線的點(即第一次碰到虛線的點)，將該點之後的路徑對虛線反射(灰線)，則這是一個從 (0,0) 到 $(n-1,n+1)$ 的走法，可驗證此為一一對應，故

$$C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n}$$

在組合計數的很多問題中都會遇到卡特蘭數，譬如以下幾個都與卡特蘭數等價：

(1) n 個節點可產生的二元樹個數

(2)在正 n 邊形中，以三角形分割多邊形的方法數

(3) n 對括號可產生的合法匹配數

~~(4)族繁不及備載~~

而在這些問題中，有時把 2 改成 3 或其他的數字可以得到另一個數列，稱為廣義卡特蘭數，其公式為 $\frac{1}{(k-1)n+1}\binom{kn}{n}$，在這裡不多加介紹。

四、生成函數

（以下皆取自 2009 年第七週 surwdkgo 學長講義）

生成函數在計數問題上，是一個強力的工具，其意義即將計數的問題，轉換成函數再使用代數的工具去解決。對一個數列$\{a_0,a_1,a_2,\cdots\}$，定義其一般型生成函數(ordinary generatingfunction)為

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots$$

而其指數型生成函數(exponential generating function)為

$$f(x) = a_0 + \frac{a_1 x}{1!} + \frac{a_2 x^2}{2!} + \frac{a_3 x^3}{3!} + \ \ldots$$

，其中指數型生成函數主要用於排列問題。

$f(x) = \frac{1}{1-ax}$	$f(x) = \sum_{n=0}^{\infty} a^n x^n$
$f(x) = \frac{1}{(1-x)^{k+1}}$	$f(x) = \sum_{n=0}^{\infty} \binom{n+k}{k} x^n$
$f(x) = e^x$	$f(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}$

$f(x)=(1+x)^r$	$f(x)=\sum_{n=0}^{\infty}\binom{r}{n}x^n$ ，其中 $\binom{r}{n}=\frac{r(r-1)(r-2)\dots(r-n+1)}{n!}$
$f(x)=g(x)h(x)$	$f(x)=\sum_{n=0}^{\infty}\left(\sum_{k=0}^{n}a_k b_{n-k}\right)x^n$
$f(x)=g'(x)$	$f(x)=\sum_{n=0}^{\infty}(n+1)g_{n+1}x^n$
$f(x)=\int g(x)dx$	$f(x)=\sum_{n=0}^{\infty}\frac{g_{n-1}x^n}{n}$

以下以費氏數列為例，簡單的介紹怎麼使用生成函數。

設 $f(x)$ 為費氏數列之一般型生成函數，則

$$f(x)=\sum_{n=0}^{\infty}F_n x^n=1+x+\sum_{n=2}^{\infty}(F_{n-1}+F_{n-2})x^n$$

$$=1+x\sum_{n=0}^{\infty}F_n x^n+x^2\sum_{n=0}^{\infty}F_n x^n=1+xf(x)+x^2f(x)$$

$$\Rightarrow f(x)=\frac{1}{1-x-x^2}=\frac{1}{(1-\varphi_1 x)(1-\varphi_2 x)}$$

$$=\frac{1}{\sqrt{5}}\left(\frac{\varphi_1}{1-\varphi_1 x}-\frac{\varphi_2}{1-\varphi_2 x}\right)$$

$$=\sum_{n=0}^{\infty}\frac{1}{\sqrt{5}}(\varphi_1^{n+1}-\varphi_2^{n+1})x^n$$ ，即

$$F_n=\frac{1}{\sqrt{5}}(\varphi_1^{n+1}-\varphi_2^{n+1})$$ 。

五、練習題

〈TIOJ 1501 Assssss!!〉

$x_1, x_2, \dots, x_n$ 為整數。是否存在一種括弧方式，使得 $x_1 / x_2 / \dots / x_n$ 為整數 ？

〈NPSC 2007 初賽 pB. 完全子圖〉TIOJ 1459

對於 $1 \leqq k \leqq n$，已知 $x \equiv f_k \pmod{w_k}$，求 x 通解或判斷出無解。

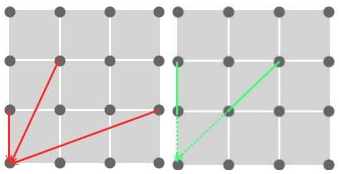
〈TIOJ 1135 止不住的迴圈〉

```
for (int i = A; i != B; i = (i + C) % (1 << k) ) do_something();
```

給定 A, B, C, k，請求出 do_something();會執行的次數，或是無限次。

〈TIOJ 1607 山稜線〉
請在 $O(n \log n)$ 的時間內求出卡特蘭數列第 1 ~ n 項除以 1,000,000,007 的餘數。

〈TIOJ 1514 *Skyly & Shik* Contest #1 pD. 橫掃射擊場〉
如右圖，你在 (0, 0)，給定正方形的大小，有多少個看不到的點 ？
兩個點看得到的條件是它們之間沒有其他點。如左圖就看得到。



〈TIOJ 1607 山稜線〉
請在 $O(n \log n)$ 的時間內求出卡特蘭數列第 1 ~ n 項除以 1,000,000,007 的餘數。

〈TIOJ 1219 發糖果囉〉(Source：POI XIV Stage I. Queries)
請問對於 $1 \leqq a \leqq x$, $1 \leqq b \leqq y$ 中有多少個數對 (a, b) 互質 ？ $x, y \leqq 50{,}000$ 且為多筆詢問。

〈UVa 10689 Yet another Number Sequence〉
定義 $f_0 = a, f_1 = b, f_n = f_{n-1} + f_{n-2}$，求 $f_n \bmod m$。

〈UVa 10518 How many calls〉
定義 $F_0 = 1, F_1 = 1, F_n = F_{n-1} + F_{n-2}$，請問遞迴計算 F_n 需要多少次函式呼叫 ? (mod m)

〈NTUJ 1023 Brute-force Algorithm〉(ACM/ICPC 某 Regional)
給定 a, b, n, P，令 F_n 代表費氏數列第 n 項，$a^{F_n} b^{F_{n-1}} \bmod P$ 為 ？
※若 $(a, P)=1$, $(b, P)=1$ 呢 ？ 但若不互質呢 ? (P 不一定為質數)

〈NTUJ 1014 Rating Hazard〉
給定實數 a, b，求分母最小的一個最簡分數 p/q 滿足 $a \leqq p/q < b$。輸出分母。

〈NPSC 2006 初賽 pC. 幼稚國王的獎賞〉TIOJ 1094
給定 N 個非負整數，請問它們可以 Xor 出的最大數字是多少 ？

〈UVa 10237 Bishops〉
若要在一個 n*n 的棋盤中擺上 k 個主教，使得他們互相無法攻擊，有幾種擺法 ? (主教攻擊斜的)

・以下程式的時間複雜度為 ？

```
procedure XD( n )
    for i←2 to n do
        for j←1 to 因數個數( i ) do something;
```

計算幾何

・比大小(浮點數誤差)

需要注意，因為浮點數運算有誤差，因此浮點數比大小應寫成類似

```
DOUBLE-COMPARISON (a, b)
    d ← a－b
    if d＞EPS then return 1
    if d＜-EPS then return -1
    return 0
```

其中 EPS 為一極小的常數。

・內積 給定兩個向量$\vec{a}=(x_1,y_1),\vec{b}=(x_2,y_2)$，且其夾角為 θ，則

$$\vec{a}\cdot\vec{b}=x_1x_2+y_1y_2=|\vec{a}||\vec{b}|\cos\theta$$

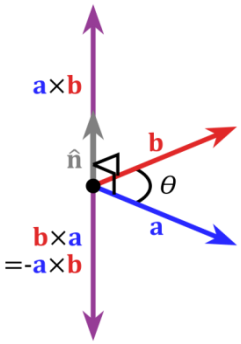
・平面上的「外積」 給定兩個向量$\vec{a}=(x_1,y_1),\vec{b}=(x_2,y_2)$，且其夾角為 θ，則

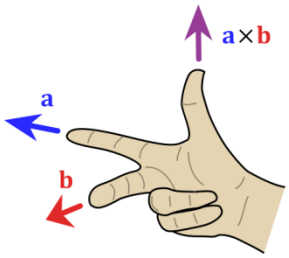
$$\vec{a}\times\vec{b}=\begin{vmatrix}x_1 & y_1\\ x_2 & y_2\end{vmatrix}=x_1y_2-x_2y_1=|\vec{a}||\vec{b}|\sin\theta$$

事實上，(高中)數學上定義的外積是空間中向量的運算，設兩個向量

$\vec{a}=(x_1,y_1,z_1)$、$\vec{b}=(x_2,y_2,z_2)$，其夾角為 θ，則

$\vec{a}\times\vec{b}=\left(\begin{vmatrix}y_1 & z_1\\ y_2 & z_2\end{vmatrix},\begin{vmatrix}z_1 & x_1\\ z_2 & x_2\end{vmatrix},\begin{vmatrix}x_1 & y_1\\ x_2 & y_2\end{vmatrix}\right)=\vec{n}$ 為 $\vec{a}$, $\vec{b}$ 的公垂向量，大小為$|\vec{a}||\vec{b}|\sin\theta$，方向如下





所以用外積可以方便的處理面積、判斷點的左手/右手方向等，而用內積可以判斷是否垂直、夾角是否為銳角/鈍角等。注意計算幾何中一個重要的想法就是「左/右手螺旋方向」。如，設 $P(1,2)$、$P_1(5,6)$、$P_2(2,1)$，則 $\overrightarrow{PP_1}\times\overrightarrow{PP_2}=8>0$，因此 P_2 在 P_1 相對於 P 的右手螺旋(逆時針)方向。

・判斷線段相交

給定四個點 $A(x_1,y_1)$、$B(x_2,y_2)$、$C(x_3,y_3)$、$D(x_4,y_4)$，請問 $\overline{AB}$、$\overline{CD}$ 是否相交 ？若是，則交點為 ？
最直接的想法當然就是算出直線方程式後解交點。但單純判斷線段是否相交有誤差更小的方法：

※跨立試驗：若 C、D 在 $\overline{AB}$ 異側，且 A、B 在 $\overline{CD}$ 異側，則 $\overline{AB}$、$\overline{CD}$ 相交。

因此判斷式可以簡寫如下：$(\overrightarrow{AC}\times\overrightarrow{AB})(\overrightarrow{AD}\times\overrightarrow{AB})<0\wedge(\overrightarrow{CA}\times\overrightarrow{CD})(\overrightarrow{CB}\times\overrightarrow{CD})<0$
注意要先處理掉三點共線的情況，非規範相交(交點為端點)的情況則要例外處理。
至於判斷交點可以求出面積比，再用分點公式，比直接解方程式稍微精確一點。

・簡單多邊形面積

對於簡單多邊形 $P_1(x_1, y_1), P_2(x_2, y_2), P_3(x_3, y_3), \ldots, P_n(x_n, y_n)$，其面積為(假設 $P_{n+1}=P_1$)

$$\frac{1}{2}\left|\sum_{k=1}^{n}\begin{vmatrix} x_k & y_k \\ x_{k+1} & y_{k+1}\end{vmatrix}\right| = \frac{1}{2}\left|\sum_{k=1}^{n}(x_k y_{k+1} - x_{k+1} y_k)\right|$$

其中一種證明法是把每個點都對 x 軸做垂線，求梯形面積和化簡即可。

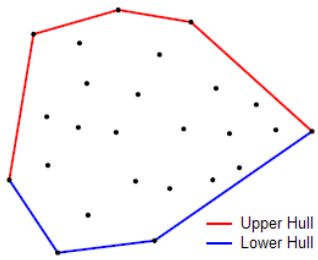
・判斷點是否在簡單多邊形內

設給定的多邊形頂點為 $P_1(x_1, y_1), P_2(x_2, y_2), P_3(x_3, y_3), \ldots, P_n(x_n, y_n)$，要判斷的點為 $A(x, y)$ 若所給定的多邊形是凸多邊形，可以求 $\Sigma\ |\Delta\ AP_kP_{k+1}|$，若面積和等於凸多邊形的面積，則 A 在多邊形內(或邊上)。或者若 A 在多邊形內，則對於 $\overrightarrow{P_1P_2}$、$\overrightarrow{P_2P_3}$、…、$\overrightarrow{P_kP_{k+1}}$、…、$\overrightarrow{P_nP_1}$，$A$ 在同一側。對任意簡單多邊形，一般的做法是從 A 往任意方向做射線。若與多邊形的交奇數次，則 A 在多邊形內；否則 A 在多邊形外。若射線剛好交在多邊形的頂點(或貼在邊上)上會造成判斷困難，一種做法是把射線稍微偏移一點角度，避開交點。

・極角排序

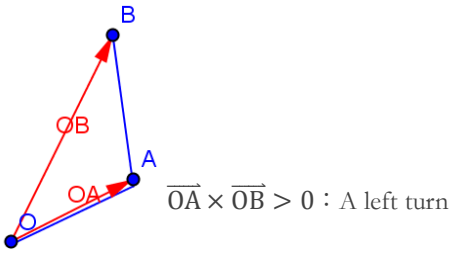
極角排序是指把一堆點相對於一個點依照逆(順)時針順序排序。通常的做法是用 `atan2(y,x)` 排序，或使用外積判斷方向。用外積判斷方向可以先把點依其在基準點的左/右分成兩堆分別排序。

・凸包 (Convex Hull)



對於平面上的一些點，我們想要找出一個最小的凸多邊形，使得所有的點都在這個凸多邊形內(或邊上)。我們稱這個凸多邊形為這些點的凸包，底下介紹 Andrew's Monotone Chain Convex Hull algorithm。其主要想法是：把凸包分兩次包，一次包下半部，一次包上半部；最左下、最右上的點一定都在凸包上，總時間複雜度為 $O(n \log n)$。

一個點集的凸包求出來後，可以用旋轉卡殼的概念在 $O(n)$ 的時間中求出點集中最遠的兩個點的距離，以及最小面積/周長包含矩形，兩凸多邊形的最遠/最近距離等等。(可以搜尋 Rotating Calipers)

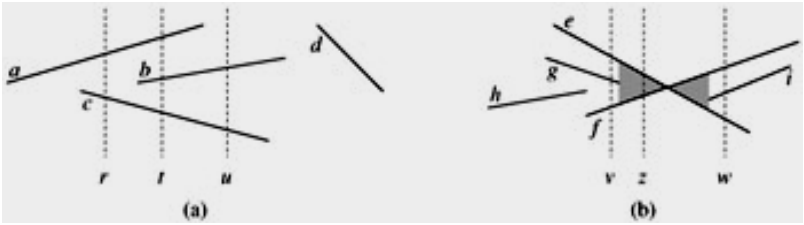



```
CROSS ( O , A , B )
    (x1 , y1) ← A − O    ▷ OA向量
    (x2 , y2) ← B − O    ▷ OB向量
    return x1×y2 − x2×y1

CONVEX-HULL ( P )
    sort P lexicographically (first by x-coordinate, and in the case of a tie, by y-coordinate)
    L ← empty , U ← empty
    for i ← 1 to |P| do
        while |L| ≧ 2 and CROSS( L[|L| − 1], L[|L|], P[ i ] ) ≦ 0 do L.pop_back()
        L.push_back( P[ i ] )    ▷ 維持左(逆時針)轉
    for i ← |P| down to 1 do
        while |U| ≧ 2 and CROSS( U[|U| − 1], U[|U|], P[ i ] ) ≦ 0 do U.pop_back();
        U.push_back( P[ i ] )    ▷ 維持左(逆時針)轉
    return L ∪ U
```

・判斷線段相交

判斷一堆線段 $L_1, L_2, \ldots, L_n$ 是否相交。
為了方便起見，我們假設沒有任何線段是垂直的，任意三條線段也不會在端點相交。實際寫的時候要把這些狀況處理一下。在這個演算法中，我們使用「掃描線」的概念：想像一條鉛直線從左邊掃到右邊。對於任意兩條線段 L_i、L_j，若他們同時與 $x=T$ 的掃描線相交，我們就說 L_i 和 L_j 在 T 是可比較的，且若此時 L_i 在 L_j 上面，那我們定義他們的關係為 $L_i <_T L_j$。



觀察後可發現，若兩個線段不相交，則在他們可比較的所有時刻 $[T_L, T_R]$ 中，他們的關係不會改變。若兩條線段相交，那必然存在一段時間 $[T_L, T_R]$，使得在這段時間內他們之中沒有其他線段。
在底下的演算法中，我們維護一個線段的集合 S (且 S 是基於「$<_T$」關係的偏序集)，保存所有與掃描線相交的線段，並支援以下操作：

(1) INSERT(S, s)：將 s 加入集合 S
(2) DELETE(S, s)：從集合 S 中刪除 s
(3) ABOVE(S, s)：傳回集合 S 中在 s 上面的線段
(4) BELOW(S, s)：傳回集合 S 中在 s 下面的線段

演算法如下：

```
SEGMENTS-INTERSECT( L[1…n] )
S ← ∅
move sweeping line from left to right
    if segment s enters sweeping line
        INSERT ( S, s )
        if ABOVE ( S, s ) or BELOW ( S, s ) exists and intersects s
            then return true                ▷ s 與 ABOVE ( S, s )、BELOW ( S, s ) 相鄰
    if segment s leaves sweeping line
        if both ABOVE ( S, s ) and BELOW ( S, s ) exist and ABOVE ( S, s ) intersects BELOW ( S, s )
            then return true                ▷ ABOVE ( S, s )與 BELOW ( S, s ) 相鄰
        DELETE ( S, s )
return false
```

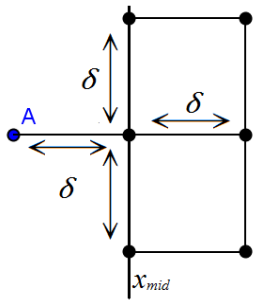
若使用平衡樹等資料結構，則可以做到 $O(n \log n)$ 的時間複雜度。

・最近點對問題 (Closest pair of points problem)

給定平面上 n 個點，$P_1, P_2, \ldots, P_n$，請找出最近的兩個點之間的距離。
若我們將這 n 個點從中間分成 L、R(左右)兩個點集，那最短的距離 δ 只有三種情況：

(1) 距離最近的點對位於 L 集合，則 $\delta = \delta_L$
(2) 距離最近的點對位於 R 集合，則 $\delta = \delta_R$
(3) 距離最近的點對恰巧一個點在 L、一個點在 R

因此可以先得到大略的演算法如下：




```
FIND-DISTANCE ( P )
    split P into two sets L and R by x = x_mid   ▷ so that ||L| − |R|| ≤ 1
    δ ← min( FIND-DISTANCE( L ) , FIND-DISTANCE( R ) )
    return min(δ , CLOSEST-POINT( L, R, δ ) )   ▷ Case (3)
```

時間複雜度為 $T(n)=2T(n/2)+f(n)$，其中 $f(n)$ 是 CLOSEST-POINT(L, R, δ) 所需時間。所以現在的問題就在 Case (3) 如何處理。觀察到：若 Case (3) 發生，那最近的點 P_i、P_j 的 x 座標一定都會在區間 $(x_{mid}-\delta, x_{mid}+\delta)$ 中。更進一步的，對於一個在左邊的點 $P_i(x_i, y_i)$，點 $P_j(x_j, y_j)$一定滿足限制 $x_j \in (x_i-\delta, x_i+\delta)$、$y_j \in (y_i-\delta, y_i+\delta)$，又任意兩個滿足條件的點 P_j、P_k 一定有 $\overline{P_jP_k} \geq \delta$，因此對 L 中每個點只需要考慮不超過 6 個點。若 L、R 皆已照 y 座標排序則可以用 Queue 做到 $O(n)$。

```
CLOSEST-POINT( L, R, δ )
    sort L, R by y-coordinate
    Queue ← ∅
    j ← 1
    for i ← 1 to |L| do
        if |x[L[ i ]] − x_mid| ≧ δ then continue
        while j ≦ |R| and y[R[ j ]] < ( y[L[ i ]] + δ ) do
            if |x[R[ j ]] − x_mid| < δ then do ENQUEUE( Queue, R[ j ] )
            j ← j + 1
        while Queue is not empty and y[Queue.front] ≦ ( y[L[ i ]] − δ ) do
            DEQUEUE( Queue )
        for each point P in Queue do
            δ ← min( δ, distance of L[ i ] and P )
    return δ
```

假如我們在 CLOSEST-POINT(L, R, δ) 的第一步用 $O(n \log n)$ 的排序，演算法整體的時間複雜度就是 $O(n \log n \log n)$。然而，其實可以邊找最小值、邊用 Merge Sort，整體時間複雜度 $O(n \log n)$。最後，當 P 中的點數夠少(如三個點)時，可以直接枚舉所有點對，減少遞迴花費。

・最小包含圓 (Smallest Enclosing Circle/Disk)

給定平面上 n 個點，$P_1, P_2, \ldots, P_n$，請找出包含他們的最小圓。
直接枚舉的時間複雜度是 $O(n^4)$，透過逐步把圓縮小的想法，我們可以得到一個 $O(n^2)$的做法：

```
1  MIN-CIRCLE( P )
2      draw a circle enclosing all points in P
3      make the circle smaller until it contains at least two points on its boundary
4      while we can find a point-free arc AB which is greater than half of the circle's circumference
5          while keeping A and B on the circle's boundary, reduce the diameter of the circle until
6              (1) the diameter is the distance AB
7            or (2) the circle touches another point C
```

這個演算法做的事情非常簡潔。其中第 3 行的 while 最多會被執行 $n-2$ 次，每一次要花費 $O(n)$ 的時間把圓縮小，因此整體的時間複雜度為 $O(n^2)$。此外，還有一個使用 Voronoi 圖作到 $O(n \log n)$ 的做法，以及一個使用線性規劃作到 $O(n)$ 的演算法(用 prune-and-search 技巧)。
底下介紹另一個期望執行時間 $O(n)$的隨機增量算法：

```
MIN-CIRCLE( P )
    Compute a random permutation of P = { P_1, P_2, …, P_n }
    Let C_2 be the smallest enclosing circle of P_1 and P_2
    S ← { P_1 , P_2 }
    for i ← 3 to |P| do
        if P_i isn't in C_{i−1} then do C_i ← MIN-CIRCLE-WITH-POINT( S, P_i )   ▷ P_i is on C_i's boundary. Find
        else do C_i ←C_{i−1}                                                      minimum enclosing circle
        S ← S∪P_i                                                                 with P_i on its boundary.
    return C_{|P|}

MIN-CIRCLE-WITH-POINT( S, P )
    Let C_1 be the smallest enclosing circle of P_1 and P
    T ← { P_1 }
    for i ← 2 to |S| do
        if P_i isn't in C_{i−1} then do C_i ← MIN-CIRCLE-WITH-TWO-POINTS( T, P, P_i )
        else do C_i ←C_{i−1}                                   ▷ P and P_i are both on C_i's boundary. Find
        T ← T∪P_i                                                minimum enclosing circle with both P and P_i
    return C_{|S|}                                               on its boundary.

MIN-CIRCLE-WITH-TWO-POINTS( T, P_j, P_k )
    Let C_0 be the smallest enclosing circle of P_j and P_k
    for i ← 1 to |T| do
        if P_i isn't in C_{i−1} then do C_i ← the circumscribed circle of P_i, P_j and P_k
        else do C_i ←C_{i−1}
    return C_{|T|}
```

※正確性

我們保證演算法找出來的圓必包含所有點，再討論找出來的會是最小圓。在 MIN-CIRCLE 中，若 $P_i \notin C_{i-1}$，代表 P_i 在 C_{i-1} 的外面。在包含 $P_1, P_2, \ldots, P_{i-1}$ 的情況下，慢慢擴展直到 C_{i-1} 直到碰到 P_i。因此存在一個 P_i 在邊界上且包含 $P_1, P_2, \ldots, P_i$ 的圓。依此類推，在 MIN-CIRCLE-WITH-POINT 中也存在一個圓包含 $P_1, P_2, \ldots, P_i$ 且 P_i、P 皆在邊界上。在 MIN-CIRCLE-WITH-TWO-POINTS 中，若行 3 被執行，設 $P \in \{ P_1, P_2, \ldots, P_{i-1} \}$，若 P 與 P_i 在 $\overleftrightarrow{P_jP_k}$ 同側，則 $P \in C_i$。反之，若 P 與 P_i 在 $\overleftrightarrow{P_jP_k}$ 異側，因為存在同時包含 P、P_i 且 P_j、P_k 在邊界上的圓，因此 $\angle P_jPP_k + \angle P_jP_iP_k \leqq \pi \Rightarrow P \in C_i$。

接著來看找到的圓是否會是最小的。我們考慮把演算法的流程「倒過來」看，用移除點的方式討論。設 C(P,Q)代表包含 P 中所有點、Q 中所有點接在圓周上的圓，且 $P \cap Q = \emptyset$。設 $p \in P$，若 $p \in C(P,Q)$，那麼 $C(P,Q) = C(P - \{p\},Q)$。否則 $C(P,Q) = C(P - \{p\},Q \cup \{p\})$。

※時間複雜度

時間複雜度為 $O(n^3)$。

一樣把演算法倒過來看，一個一個點移除。先考慮 MIN-CIRCLE-WITH-POINT 的時間複雜度，因為演算法每一步找出來的都會是當前最小圓，所以若把點移除後，最小圓變更小的話，那圓周上必然會有 2 個點或 3 個點，而其中一個點是永遠不會被移除的 P 點。因為我們是隨機加入點的，所以每個點被移除的機會都均等，因此在總共已插入 i 個點時，把第 i 個點洽是圓周上的點的機率不大於 2/i。所以 MIN-CIRCLE-WITH-POINT 的期望時間複雜度就是

$$O(n) + \sum_{i=2}^{n} O(i) \times \frac{2}{i} = O(n)$$

類似的，MIN-CIRCLE-WITH-POINT 被呼叫的機率為 3/i，則 MIN-CIRCLE 的期望執行時間是 $O(n)$。