計算幾何

sam571128

2022-07-15

目錄

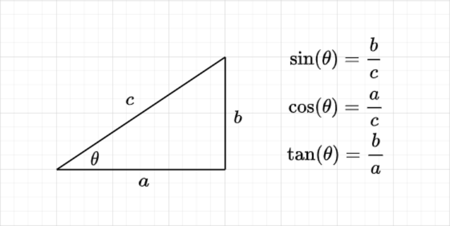
先備知識

三角函數 (Trigonometric Functions)

三個基本的三角函數

三角函數是被定義在直角三角形上的函數，最基本的有以下三種

$\sin(\theta)$ 表示對邊除以斜邊的值，$0 \le \sin(\theta) \le 1$
$\cos(\theta)$ 表示鄰邊除以斜邊的值，$0 \le \cos(\theta) \le 1$
$\tan(\theta)$ 表示對邊除以鄰邊的值

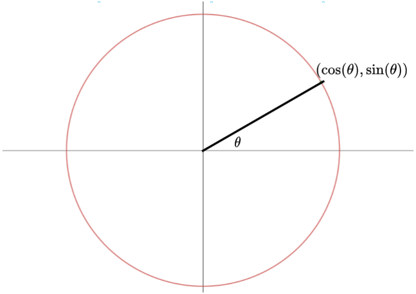


廣義角

廣義角

只被定義在直角三角形上的三角函數，並不方便。

我們用單位圓 (Unit Circle) 定義了除了 $0 \leq \theta \leq 90^\circ$ 的角度時，三角函數的值。



弧度 (Radian)

由於角度 (Degree) 的使用，會導致在畫圖時的困擾，定義了另一個度數單位 - 弧度

弧度 (Radian)

定義 π 為 180°，2π 為 360°

$$1 \text{ rad} = \frac{180^\circ}{\pi}$$

$$1^\circ = \frac{\pi}{180} \text{ rad}$$

在 C++ 內建的三角函數 `sin(theta)`, `cos(theta)`, `tan(theta)` 都必須要使用弧度

反三角函數 (Inverse Trigonometric Function)

有了 $\sin(\theta)$, $\cos(\theta)$, $\tan(\theta)$ 的值，但想要知道 θ

$$\arcsin(x) = \sin^{-1}(x):\ -1 \le x \le 1,\ -\frac{\pi}{2} \le \sin^{-1}(x) \le \frac{\pi}{2}$$

$$\arccos(x) = \cos^{-1}(x):\ -1 \le x \le 1,\ 0 \le \cos^{-1}(x) \le \pi$$

$$\arctan(x) = \tan^{-1}(x):\ -\frac{\pi}{2} \le \tan^{-1}(x) \le \frac{\pi}{2}$$

在 C++ 中，想要知道 π，可以直接呼叫 `acos(-1)`

順道一提，`atan2(y,x) = atan(y/x)`，而 $-\pi \le atan2(y,x) \le \pi$

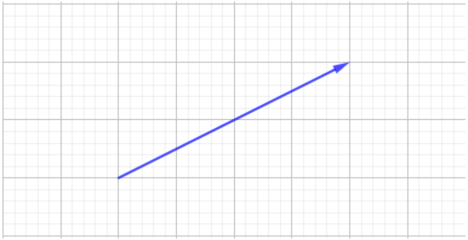
高中數學複習 - 向量

什麼是向量?

- 同時有方向和大小的量

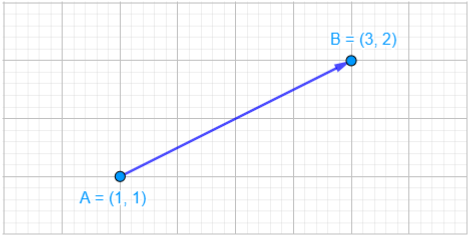
什麼是向量?

- 同時有方向和大小的量
- 表示方式:
  - 直接將向量畫出來



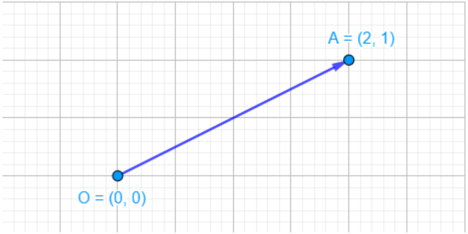
什麼是向量?

- 同時有方向和大小的量
- 表示方式:
  - 寫成從起點 $A(x_1, y_1)$ 指到終點 $B(x_2, y_2)$ 的向量，表示為 $\overrightarrow{AB} = (x_2 - x_1, y_2 - y_1)$



什麼是向量?

- 向量 (Vector)，同時有方向和大小的量
- 表示方式:
  - 寫成從原點 $O(0,0)$ 指到終點 $A(x_1, y_1)$ 的向量，表示為 $\overrightarrow{A} = (x_1, y_1)$

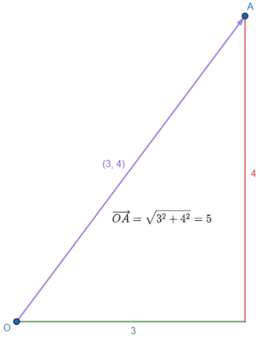


什麼是向量?

- 向量的維度可以是很多維，但最常用的是二維 (平面) 與三維 (空間) 的形式
- 可以想成是兩點在空間中的位移
- 一個向量由其方向 (單位向量) 與大小所決定

向量的大小 (Magnitude)

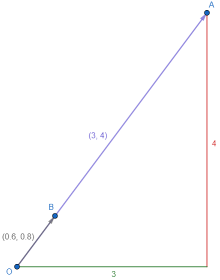
又稱長度、模長。一個 n 維向量 $\vec{v} = (x_1, x_2, \ldots, x_n)$ 的大小為 $|\vec{v}| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$



單位向量 (Unit Vector)

大小為 1 的向量。單位向量決定了向量的方向。

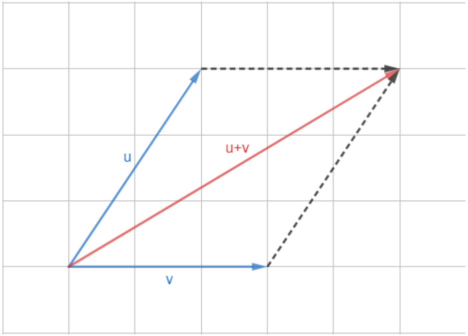
對於任一個 n 維向量 $\vec{v} = (x_1, x_2, \ldots, x_n)$，皆可以找到一個與其同向的單位向量 $\frac{\vec{v}}{|v|}$



向量的加法 (Vector Addition)

對於兩個 n 維向量 $\vec{a} = (a_1, a_2, \ldots, a_n),\ \vec{b} = (b_1, b_2, \ldots, b_n)$

定義 $\vec{a} + \vec{b} = (a_1 + b_1, a_2 + b_2, \ldots, a_n + b_n)$，可以使用平行四邊形法則畫出來

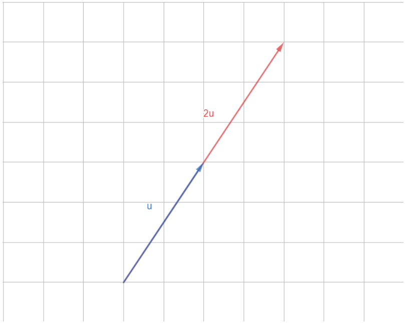


向量運算

純量乘法 (Scalar Multiplication)

對於一個 n 維向量 $\vec{a} = (a_1, a_2, \ldots, a_n)$，以及一個純量 c

定義 $c\vec{a} = (ca_1, ca_2, \ldots, ca_n)$，等同大小放大了 c 倍

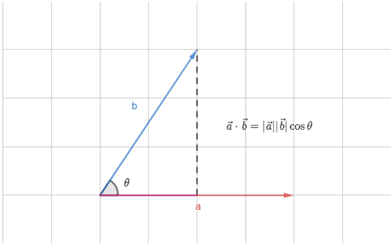


向量內積 (Dot Product)

對於兩個 n 維向量 $\vec{a} = (a_1, a_2, \ldots, a_n)$, $\vec{b} = (b_1, b_2, \ldots, b_n)$

定義 $\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$。等同於 $\vec{b}$ 投影到 $\vec{a}$ 的長度乘以 $\vec{a}$ 的長度

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



向量外積 (Cross Product)

對於兩個 3 維向量 $\vec{a} = (a_1, a_2, a_3)$, $\vec{b} = (b_1, b_2, b_3)$

定義 $\vec{a} \times \vec{b}$ 為一個與 $\vec{a}$ 和 $\vec{b}$ 同時垂直的向量 $\vec{c}$

計算方式為

$$\begin{vmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

特點是 $|\vec{a} \times \vec{b}| = |\vec{a}||\vec{b}| \sin\theta$ 等同於 $\vec{a}$ 與 $\vec{b}$ 所夾的平行四邊形面積

可以使用右手定則判斷外積出來的向量方向。

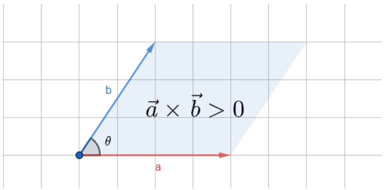
向量運算

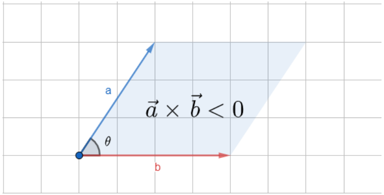
向量外積 (Cross Product)

儘管外積的定義只有在三維向量，在程式中，我們通常只會用到二維的向量。

因此在程式中，我們通常定義 $\vec{a} \times \vec{b}$ 是一個純量，而面積的正負表示方向

定義 $\vec{a} \times \vec{b} = |\vec{a}||\vec{b}| \sin\theta = a_1 b_2 - a_2 b_1$





向量運算

向量之間的夾角

我們可以根據內積的定義 $\vec{a} \cdot \vec{b} = |\overrightarrow{a}||\overrightarrow{b}|\cos\theta$ 來得到

$$\theta = \cos^{-1}(\frac{\vec{a} \cdot \vec{b}}{|\overrightarrow{a}||\overrightarrow{b}|})$$

用電腦存向量

怎麼在程式中實作向量?

以下是在打比賽時常見的作法

- 為了方便起見，將所有向量儲存成從原點開始的向量 (用點 A 的位置 (x, y) 表示 $\overrightarrow{OA}$)
- 常見實作方式是使用 `pair<int,int>`, `pair<double,double>` 或者 `struct`
- 通常會將向量寫成模板

怎麼在程式中實作向量?

```cpp
const double EPS = 1e-7;

struct point {
    double x, y;

    point operator * (int a) {return {a * x, a * y};}
    point operator / (int a) {return {x / a, y / a};}
    point operator + (point b){return {x + b.x, y + b.y};}
    point operator - (point b){return {x - b.x, y - b.y};}

    double operator * (point b){return x * b.x + y * b.y;}
    double operator ^ (point b){return x * b.y - y * b.x;}
};

double abs(point a){ return sqrt(a.x * a.x + a.y * a.y); }

int sign(double a){
    if(abs(a) < EPS) return 0;
    else return (a > 0 ? 1 : -1);
}

int ori(point a, point b, point c){
    return sign((b-a)^(c-a));
}
```

怎麼在程式中實作向量?

剛剛的程式碼中，出現了這個函式 `sign()`，而這個函式是幫助我們在浮點數時判斷正負的

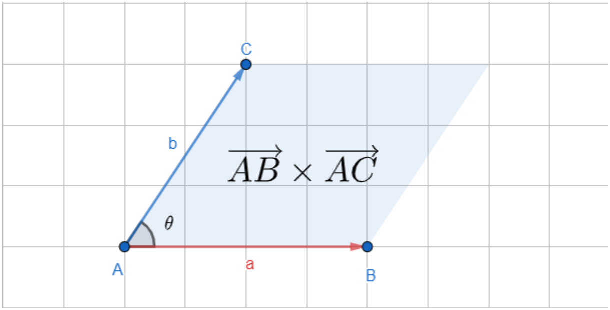
```
int sign(double a){
    if(abs(a) < EPS) return 0;
    else return (a > 0 ? 1 : -1);
}
```

EPS 的估計，我們會在這堂課後面與大家介紹，不過通常取 $10^{-6}, 10^{-7}$ 就足夠了

怎麼在程式中實作向量?

判斷順時針與逆時針，我們可以使用剛剛的 `ori(a,b,c)` 來做到

概念是，我們想要判斷 $\overrightarrow{AB}$ 轉到 $\overrightarrow{BC}$ 是順時針還是逆時針



直接使用外積即可

浮點數誤差分析

IEEE 754

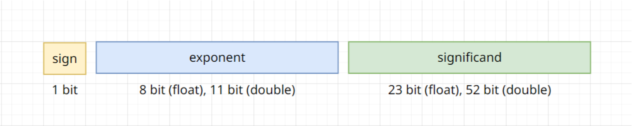
浮點數在電腦中的儲存被分成三個部分，$\mathtt{sign}$, $\mathtt{exponent}$, $\mathtt{significand}$

$$\mathtt{value} = -1^{\mathtt{sign}} \times 2^{\mathtt{exponent}} \times \mathtt{significand}$$

$\mathtt{sign}$ 表示了浮點數的正負號

$\mathtt{exponent}$ 與 $\mathtt{significand}$ 表示了科學記號的形式

$\mathtt{significand}$ 是一個介於 1 到 2 之間的數字，例如: $1.0101 = 2^0 + 2^{-2} + 2^{-4} = 1.3125$



浮點數在電腦中的儲存

浮點數誤差

正如同我們無法使用十進位的小數完整表示 $\frac{1}{3}$, $\frac{1}{7}$ 等數字

在電腦中，我們也只能想辦法使用二進位來逼近一個數字

因此，在大部分的浮點數運算，數字一定會出現誤差

浮點數在電腦中的儲存

以下是 C++ 不同型態在儲存浮點數時，會產生的相對誤差

Type	Size	Relative Precision
`float`	4 Bytes	$2^{-23} \approx 10^{-7}$
`double`	8 Bytes	$2^{-52} \approx 10^{-16}$
`long double`	10 Bytes	$2^{-63} \approx 10^{-19}$

絕對誤差與相對誤差

通常在競程中會遇到的題目，都會出現「絕對誤差或相對誤差不超過 ϵ 就算正確」

絕對誤差 (Absolute Error)

計算出的答案 x 與正確答案 $\mathtt{ans}$ 的差 $|x - \mathtt{ans}|$

相對誤差 (Relative Error)

絕對誤差與正確答案的比例 $\frac{\mathtt{absolute\ error}}{\mathtt{ans}}$

誤差的傳遞

當我們在進行浮點數運算時，儘管數字一開始的誤差很小，經過運算後，誤差會逐漸累計

加減法運算

在加減法運算時，最差的情況，絕對誤差會變成兩數絕對誤差之和

$$(x + \Delta x) \pm (y + \Delta y) = (x \pm y) + (\Delta x \pm \Delta y)$$

誤差的傳遞

當我們在進行浮點數運算時，儘管數字一開始的誤差很小，經過運算後，誤差會逐漸累計

乘除法運算

在乘運算時，最差的情況，相對誤差會變成兩數相對誤差之和

$$(x+\Delta x)(y+\Delta y) = xy(1+\frac{\Delta x}{x})(1+\frac{\Delta y}{y}) \approx xy(1+\frac{\Delta x}{x}+\frac{\Delta y}{y})$$

誤差的傳遞

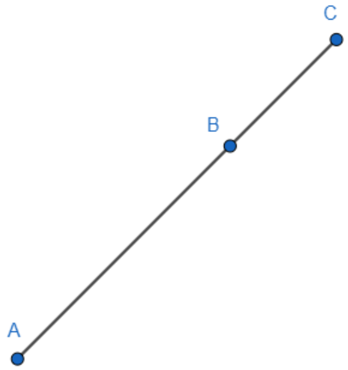
根據剛剛的兩個結論，通常如果我們要估計誤差，如果我們做了 N 次的運算

其相對誤差不會超過 $N\epsilon$，因此可以使用這樣的方式去估計誤差，並決定 EPS 的大小

向量的應用

三點共線

現在你有三個點 $A,\ B,\ C$，要如何判斷這三個點是否共線呢?



三點共線

一個沒學過向量的人，常用的方法可能是分別找出兩點之間直線的斜率。

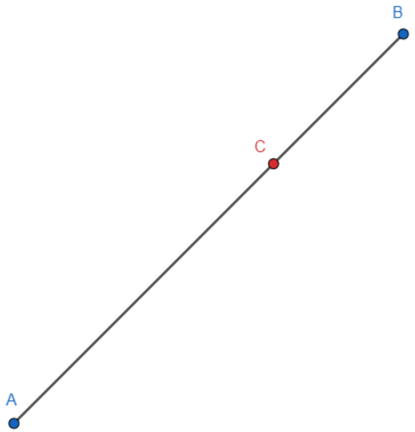
不過，我們現在有了向量的知識，其實只要 $\overrightarrow{AB}$ 與 $\overrightarrow{AC}$ 同向即可

```
bool colinear(point a, point b, point c){
    return sign((b-a)^(c-a)) == 0;
}
```

可以使用交叉相乘，或者直接用外積判斷 $\overrightarrow{AB} \times \overrightarrow{AC} = 0$ ($\overrightarrow{AB}$ 與 $\overrightarrow{AC}$ 夾成的面積為 0)

點是否在線段上

現在你有三個點 A, B, C，要如何判斷 C 是否在 $\overline{AB}$ 上呢?



點是否在線段上

只要滿足兩個條件，C 就會在 $\overline{AB}$ 上了

1. A, B, C 三點共線
2. $\overrightarrow{CA}, \overrightarrow{CB}$ 反向

點是否在線段上

只要滿足兩個條件，C 就會在 $\overline{AB}$ 上了

1. A, B, C 三點共線
2. $\overrightarrow{CA}, \overrightarrow{CB}$ 反向

如何判斷兩個向量是否反向呢?

點是否在線段上

只要滿足兩個條件，C 就會在 $\overline{AB}$ 上了

1. A, B, C 三點共線
2. $\overrightarrow{CA}$, $\overrightarrow{CB}$ 反向

如何判斷兩個向量是否反向呢? 內積 < 0

點是否在線段上

只要滿足兩個條件，C 就會在 $\overline{AB}$ 上了

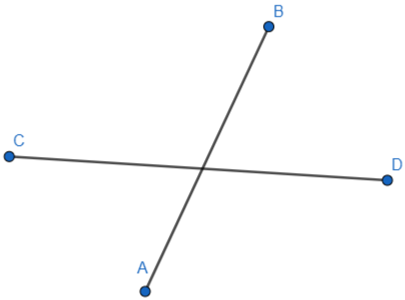
1. A, B, C 三點共線
2. $\overrightarrow{CA}$, $\overrightarrow{CB}$ 反向

如何判斷兩個向量是否反向呢? 內積 < 0

```
bool between(point a, point b, point c){
    if(!colinear(a,b,c)) return false;
    return sign((a-c) * (b-c)) <= 0;
}
```

線段相交

現在你有兩個線段 $\overline{AB}$ 與 $\overline{CD}$，要怎麼判斷兩線段是否相交?



線段相交

C, D 對於 $\overline{AB}$ 必須在兩側，A, B 對於 $\overline{CD}$ 也必須在兩側

滿足 $(\overrightarrow{AB} \times \overrightarrow{AC})(\overrightarrow{AB} \times \overrightarrow{AD}) < 0$ 且 $(\overrightarrow{CD} \times \overrightarrow{CA})(\overrightarrow{CD} \times \overrightarrow{CB}) < 0$

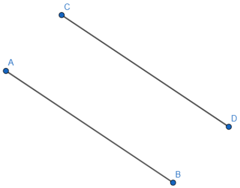
但是，有沒有別的可能性呢?

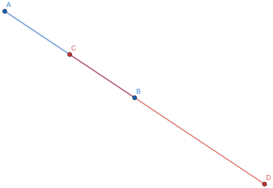
線段相交

$\overline{AB}$ 與 $\overline{CD}$ 兩線段平行怎麼辦?

線段相交

$\overline{AB}$ 與 $\overline{CD}$ 兩線段平行怎麼辦?





會發生 $(\overrightarrow{AB} \times \overrightarrow{AC}) = 0$ 且 $(\overrightarrow{AB} \times \overrightarrow{AD}) = 0$ 的情況，判斷點是否互相在對方的線段上即可

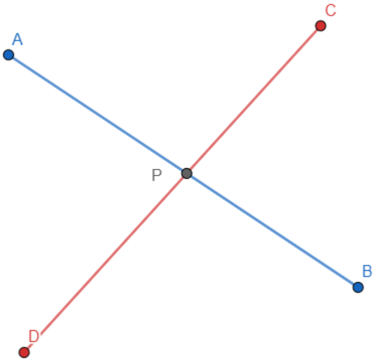
線段相交

這裡的函數有人會叫 intersect() 也有人會叫香蕉 banana()

```
bool intersect(point a, point b, point c, point d){
    int abc = ori(a,b,c);
    int abd = ori(a,b,d);
    int cda = ori(c,d,a);
    int cdb = ori(c,d,b);
    if(abc==0 && abd==0)
        return between(a,b,c) || between(a,b,d) || between(c,d,a) ||
            between(c,d,b);
    return abc * abd <= 0 && cda * cdb <= 0;
}
```

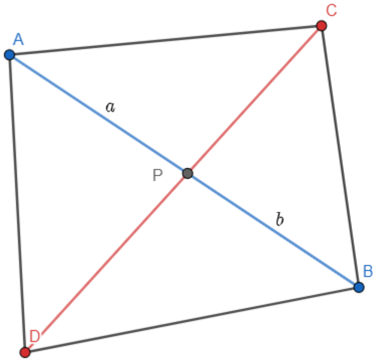
線段交點

現在你有兩個線段 $\overline{AB}$ 與 $\overline{CD}$，要怎麼計算兩線段交點 P?



線段交點

高中的時候有學過分點公式，我們可以使用面積比例套分點公式計算出交點位置



$$
\begin{aligned}
a : b &= \Delta ADC : \Delta BDC \\
&= (\overrightarrow{AD} \times \overrightarrow{AC}) : (\overrightarrow{BD} \times \overrightarrow{BC})
\end{aligned}
$$

$$
P = \frac{A \times \Delta BDC + B \times \Delta ADC}{\Delta ADC + \Delta BDC}
$$

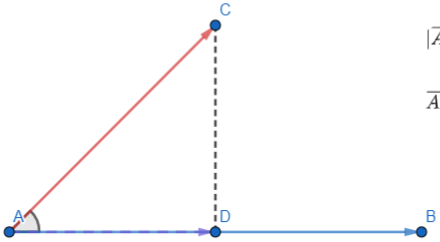
線段交點

這個方法只要在兩線段不平行時，都能計算出答案，包含兩線段延伸出去的交點

```
point intersect_point(point a, point b, point c, point d){
    int adc = (d-a) ^ (c-a);
    int bdc = (d-b) ^ (b-a);
    return (a * bdc + b * adc) / (adc + bdc);
}
```

投影

現在你有一個三個點 A, B, C，請找到 $\overrightarrow{AC}$ 投影在 $\overrightarrow{AB}$ 上的向量



$$|\overrightarrow{AD}| = \overrightarrow{AB} \cdot \overrightarrow{AC}$$

$$\overrightarrow{AD} = \frac{\overrightarrow{AB} \cdot \overrightarrow{AC}}{|\overrightarrow{AC}|} \overrightarrow{AC}$$

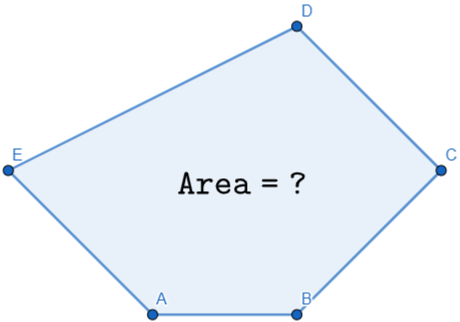
投影

直接套正射影公式即可

```
point projection(point a, point b, point c){
    return (c-a) * ((b-a) * (c-a)) / abs(c-a);
}
```

多邊形面積

照順序給你一個多邊形 $P_0P_1\cdots P_{n-1}$ 的面積，請問這個多邊形的面積是多少?

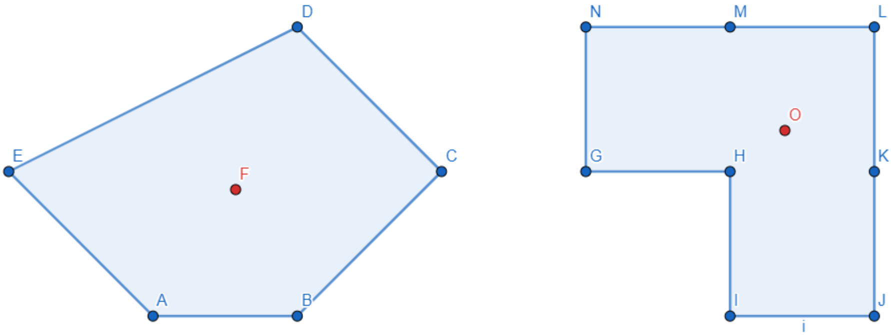


多邊形面積

- 首先，我們知道 $\Delta ABC = \frac{1}{2}|\overrightarrow{AB} \times \overrightarrow{AC}|$
- 則我們可以將多邊形想成是很多個三角形的面積相加 (面積可正可負)
- 令 $P_n = P_0$，面積就是 $\sum_{i=0}^{n-1} P_i \times P_{i+1}$
- 這個公式就是廣為人知的「測量師公式」或「鞋帶公式」
- 不只凸多邊形可以用，凹的也可以

點是否在多邊形內部

照順序給你一個多邊形 $P_0P_2\cdots P_{n-1}$，還有一個點 A，請問 A 是否在多邊形內部?

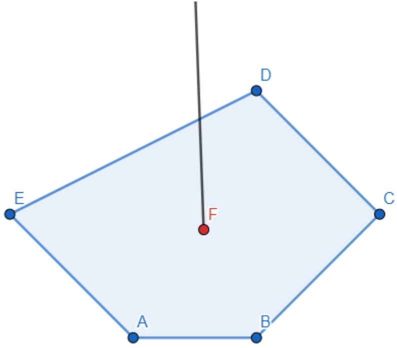


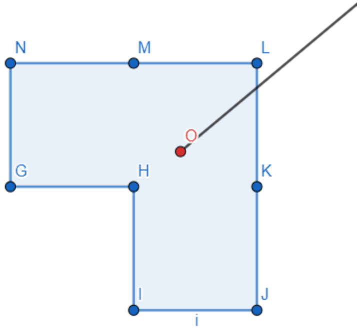
點是否在多邊形內部

這個問題有個很經典的處理方式，我們可以從 A 點開始，往外射出一條射線

一個點是否在多邊形內部 $\iff$ 射出去的射線與奇數個多邊形的邊相交

判斷時，要注意射線會不會打到頂點，否則答案可能會算錯





練習題

CSES - Point Location Test

檢查一個點在線段左邊、右邊、還是在上面

CSES - Line Segment Intersection

給你兩個線段，檢查兩個線段是否相交

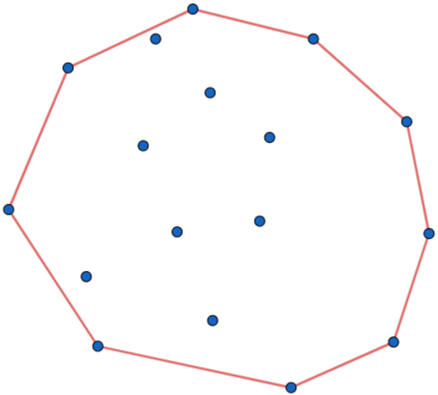
CSES - Polygon Area

給你多邊形的點，找到多邊形的面積

凸包 (Convex Hull)

什麼是凸包?

對於平面上的點集，找到一個包含所有點的最小凸多邊形，這個凸多邊形即為凸包



要怎麼找凸包?

比較廣為人知的兩種凸包演算法是「Monotone Chain」和「Graham Scan」

其中，第一種的 Monotone Chain 是在打競程比賽中較常被使用的演算法

而第二種的 Graham Scan 則是用到了等等會教的極角排序。

不過在此，我們只會介紹第一種方法。

Monotone Chain

首先，先將點依照 x 座標排序，相同的話，照 y 進行排序

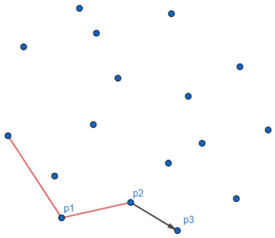
```
bool operator < (point b){
    return a.x == b.x ? a.y < b.y : a.x < b.x;
}
```

Monotone Chain

首先，先將點依照 x 座標排序，相同的話，照 y 進行排序

```
bool operator < (point b){
    return a.x == b.x ? a.y < b.y : a.x < b.x;
}
```

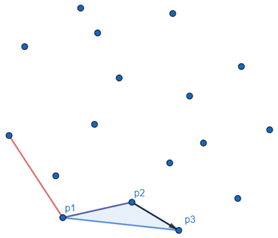
接著，我們依序去考慮每個點，類似在維護單調隊列的過程



Monotone Chain

如果目前，在單調隊列最後的兩個點 P_1, P_2，與目前要新加入的點 P_3

如果 $\overrightarrow{P_1P_3} \times \overrightarrow{P_1P_2} < 0$，則會導致多邊形不是凸的!

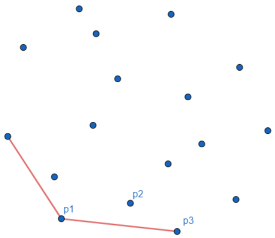


$\overrightarrow{P_1P_3} \times \overrightarrow{P_1P_2} < 0$

遇到這種情況時，我們就必須要把 P_2 給 pop 掉!

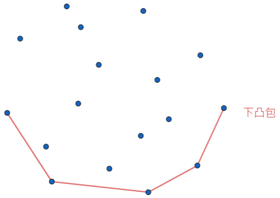
Monotone Chain

因此，原本的單調隊列就會將 P_2 給移除，再一路用同樣的方式去處理即可



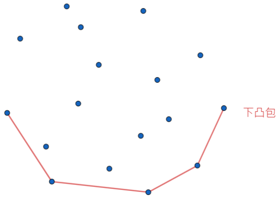
Monotone Chain

照著這樣的方式一路跑完，會建出 下凸包

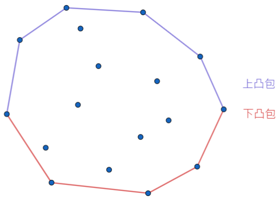


Monotone Chain

照著這樣的方式一路跑完，會建出 下凸包



只要把原來排序好的點，直接反轉過來，再做一次，就會得到上凸包!



Monotone Chain

而這樣做，時間複雜度會是 $O(n \log n)$ (排序的時間複雜度)

```
vector<point> convex_hull(vector<point> points){
    sort(points.begin(),points.end());

    vector<point> hull;
    for(int t = 0; t < 2; t++){
        int sz = hull.size();
        for(int i = 0; i < points.size(); i++){
            while(hull.size() >= sz+2 && ori(hull[hull.size()-2], hull.back(), points[i]) <= 0){
                hull.pop_back();
            }
            hull.push_back(points[i]);
        }
        hull.pop_back();
        reverse(points.begin(),points.end());
    }
    return hull;
}
```

旋轉卡尺

旋轉卡尺的概念

首先，我們先看一題最經典的例題

最遠點對 (USACO 2003 - Beauty Contest G)

給你平面上的 n 個點，請找到最遠點對之間的距離。

測資範圍: $1 \leq n \leq 5 \times 10^4$

旋轉卡尺的概念

最遠點對 (USACO 2003 - Beauty Contest G)

給你平面上的 n 個點，請找到最遠點對之間的距離。

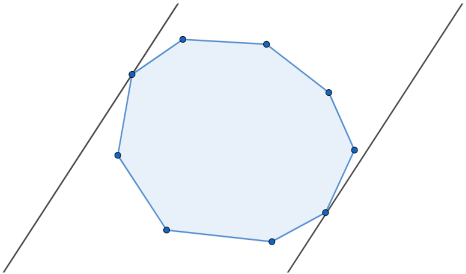
測資範圍: $1 \leq n \leq 5 \times 10^4$

最遠點對必定會出現在凸包的兩個點上，不過，我們要怎麼在凸包上找到最遠的兩個點呢?

雙指針!

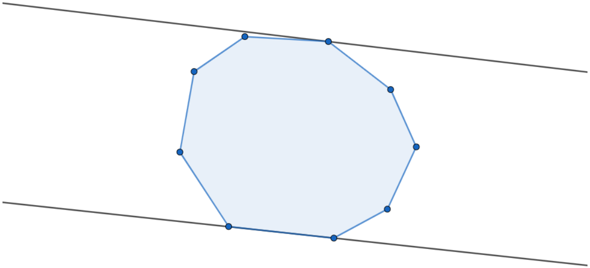
旋轉卡尺的概念

旋轉卡尺 (Rotating Caliper)，事實上可以想成兩個平行線，夾著一個凸多邊形進行旋轉



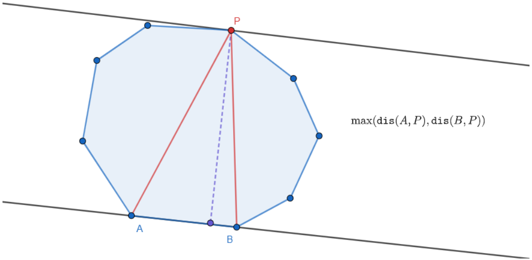
旋轉卡尺的概念

不過，我們無法完全模擬旋轉的每個時刻，因此只會去計算直線是多邊形上的邊的情形



旋轉卡尺的概念

當我們在枚舉平行線時，最遠點必定會出現在這條線段的最遠處



照著這樣的概念，我們只要對點找出凸包之後，再上面進行旋轉卡尺 (雙指針) 即可

旋轉卡尺的概念

以下是參考程式碼 (單純旋轉卡尺，複雜度是 $O(n)$)

```
double farthest_pair_of_points(vector<point> hull){
    double res = 0;
    if(hull.size() == 2){
        return abs(hull[0]-hull[1]);
    }

    hull.push_back(hull[0]);

    int j = 2;

    for(int i = 0;i < hull.size()-1;i++){
        while(area(hull[i],hull[i+1],hull[j]) < area(hull[i],hull[i+1],hull[(j+1)%hull.size()])){
            j = (j+1)%hull.size();
        }
        res = max(res,max(abs(hull[i]-hull[j]),abs(hull[i+1]-hull[j])));
    }

    return res;
}
```

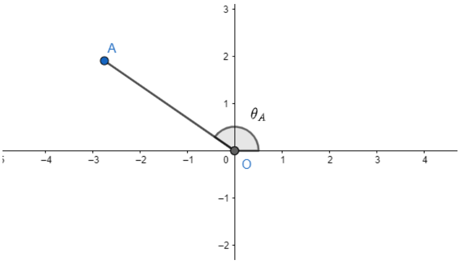
極角排序

極角排序

在座標平面上有 n 個點，要怎麼按照極角的順序將這些點排好呢?

極角

從正 x 方向開始，逆時針轉到 (x, y) 時所需的角度 (極座標系的角度)



極角排序

一個滿直覺的方式是使用 `atan2(y,x)` 的方式進行排序

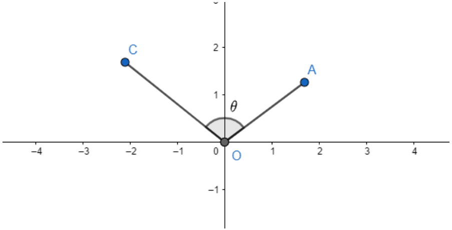
```
bool operator < (point b){
    return atan2(y,x) < atan2(b.y,b.x);
}
```

而這樣的方式，就可以快速的得到極角排序了 (不過象限順序是三四一二)

但由於 `atan2(y,x)` 會回傳浮點數，在某些情況，可能會導致精度產生問題

極角排序

因此，我們希望可以有一個更好的排序方式。而我們提過，外積可以判斷兩個點的順逆時針關係



以上圖為例，A 對 C 做外積結果為正，反之為負。

因此，我們可以分一二象限與三四象限 (分成上下兩平面) 分別用外積排序即可

極角排序

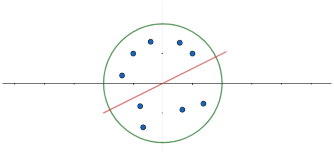
使用外積的極角排序

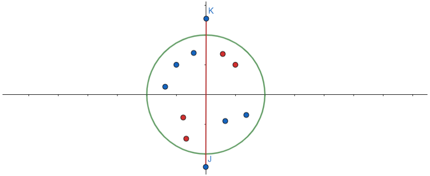
```
bool polar_sort(point a, point b){
    auto up = [&](point p){
        return p.y > 0 || p.y == 0 && p.x >= 0;
    };
    int A = up(a), B = up(b);

    if(A != B){
        return A < B;
    }

    if(sign(a^b) == 0)
        return abs(a) < abs(b);

    return sign(a^b) > 0;
}
```

例題

TOI 2020 初選 pD. 質感測試

在座標平面上有 n 個燈泡，每個點有一個質感係數。而在 $(0,0)$ 上，有一根棒狀感應器。你可以選擇這個感應器初始的方向，問你最多可以讓感應器掃過的燈泡的質感係數總和為多少?





例題

TOI 2020 初選 pD. 質感測試

在座標平面上有 n 個燈泡，每個點有一個質感係數。而在 $(0,0)$ 上，有一根棒狀感應器。你可以選擇這個感應器初始的方向，問你最多可以讓感應器掃過的燈泡的質感係數總和為多少?

首先，觀察到感應器掃過的燈泡經過極角排序後，一定是連續的

例題

TOI 2020 初選 pD. 質感測試

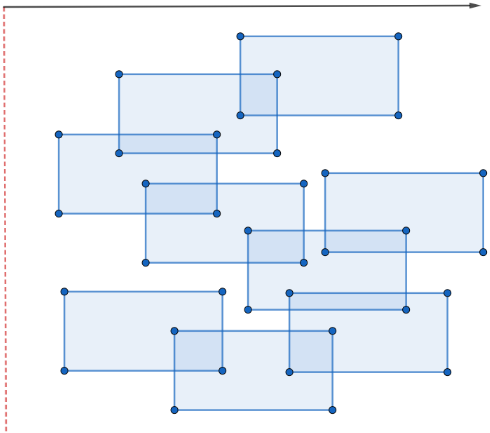
在座標平面上有 n 個燈泡，每個點有一個質感係數。而在 $(0,0)$ 上，有一根棒狀感應器。你可以選擇這個感應器初始的方向，問你最多可以讓感應器掃過的燈泡的質感係數總和為多少?

首先，觀察到感應器掃過的燈泡經過極角排序後，一定是連續的
我們只要將這些點進行極角排序之後
對陣列找「環狀最大連續和」即可 (環狀最大連續和 $=$ $\texttt{max}$(最大連續和, 總和-最小連續和))

掃描線

掃描線的概念

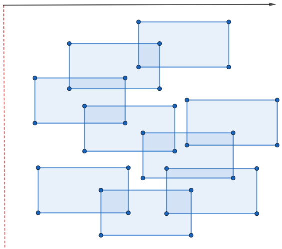
將幾何物件轉成不同的事件，使用掃描線的概念來進行枚舉，



掃描線的概念

TIOJ 1224 - 矩形覆蓋面積計算

給你很多平面上的矩形，請求出它們覆蓋的總表面積。

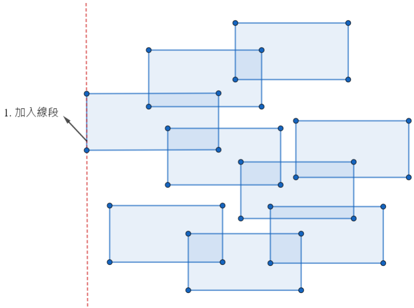


掃描線的概念

這題是一個滿經典的掃描線搭配線段樹的題目

我們可以使用一個鉛直的掃描線往右掃，考慮三種不同的情況

1. 矩形的左邊界 (區間加值)
2. 查詢的點 (查詢區間最小值數量)
3. 矩形的右邊界 (區間減值)

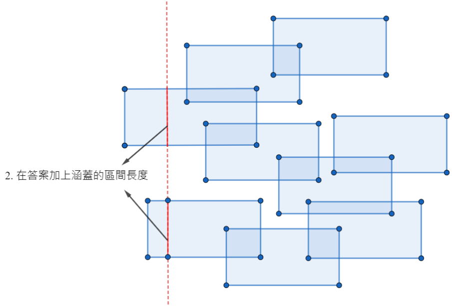


掃描線的概念

這題是一個滿經典的掃描線搭配線段樹的題目

我們可以使用一個鉛直的掃描線往右掃，考慮三種不同的情況

1. 矩形的左邊界 (區間加值)
2. 查詢的點 (查詢區間最小值數量)
3. 矩形的右邊界 (區間減值)

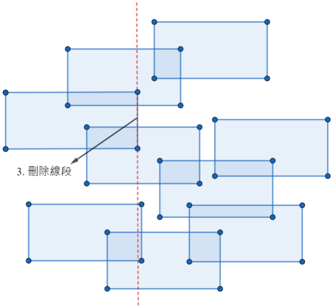


掃描線的概念

這題是一個滿經典的掃描線搭配線段樹的題目

我們可以使用一個鉛直的掃描線往右掃，考慮三種不同的情況

1. 矩形的左邊界 (區間加值)
2. 查詢的點 (查詢區間最小值數量)
3. 矩形的右邊界 (區間減值)



掃描線的概念

而這三個操作，我們可以使用一棵懶標線段樹來達成。

1. 矩形的左邊界 (區間加值)
2. 查詢的點 (查詢區間最小值數量)
3. 矩形的右邊界 (區間減值)

掃描線的概念

而這三個操作，我們可以使用一棵懶標線段樹來達成。

1. 矩形的左邊界 (區間加值)
2. 查詢的點 (查詢區間最小值數量)
3. 矩形的右邊界 (區間減值)

練習題

CSES - Intersection Points (線段交點數量)

給你 n 個鉛直或水平的線段，問這些線段總共有多少交點。

Codeforces 1401E - Divide Square

現在平面上有一個 $10^6 \times 10^6$ 的正方形以及 n 個鉛直或水平的線段，問這些線段總共將這個正方形切割成了幾塊不同的區塊。

更多例題

一些給大家思考看看的題目

通過最多點的直線

平面上有 n 個點，任意畫一條直線，這條直線最多能通過幾個點?

點到線段的最短距離

給你一個點 P 和一個線段 $\overrightarrow{AB}$，問點 P 距離線段 $\overrightarrow{AB}$ 的最短距離是多少?

最窄寬度

平面上有 n 個點，用兩條平行線將所有點夾住的話，兩條平行線的寬度最小可以是多窄?

三角形數量

平面上有 n 個點，你總共可以畫出幾個三角形