數論與計算幾何

hansonyu123

2016 年 10 月 31 日

1 數論

1.1 最大公因數和歐幾里德算法

如果 d 同時整除 a, b，就稱 d 是 a, b 的公因數。顧名思義，最大公因數就是最大的正公因數，記為 $\gcd(a, b)$。最大公因數有許多性質，條列如下：

1. 結合律與交換律

2. $\gcd(a, 0) = |a|$

3. $\gcd(a, b) = \gcd(a, b - a) = \gcd(a, b \bmod a)$

4. (Bezout's Theorem) 對於所有整數 a, b，存在兩個整數 x, y 使得 $ax + by = \gcd(a, b)$

5. 若 $\gcd(n, b) = 1$ 且 $n|ab$，則 $n|a$

利用 2.、3. 可以輕鬆地算出最大公因數，稱為「歐幾里德算法」，實際上就是我們常聽到的「輾轉相除法」，複雜度 $O(\log(\min(a, b)))$，最差情況發生在兩數是費氏數列的相鄰項時。實作上請注意正負號，因為 C++ 的奇怪運算規則導致對負數模運算會得到負數。

第 4 條性質告訴我們存在那樣的 x, y。如果要對於一組 a, b 求得任一組符合條件的 x, y，可以將上述算法擴展一下，所以它就叫作「擴展歐幾里德算法」了。基本的想法是得到 $(b, a \bmod b)$ 的答案之後設法算出 (a, b) 的答案。可以發現若 $bx' + (a \bmod b)y' = d$，那麼 $ay' + b(x' - \frac{(a-(a \bmod b))y'}{b}) = d$，化簡就得到 $ay' + b(x' - \lfloor \frac{a}{b} \rfloor y') = d$。

Algorithm 1: Extended Euclid Algorithm in C++

```
1 tuple<int, int, int> extended_gcd(int a, int b) {
2     if (!b) return make_tuple(a, 1, 0);
3     int d, x, y;
4     tie(d, x, y) = extended_gcd(b, a % b);
5     return make_tuple(d, y, x - (a / b) * y);
6 }
```

複雜度仍然是 $O(\log(\min(a,b)))$。另外，因為同時回傳三個值有點麻煩，所以一個方法是寫成迴圈版本，或者用 reference 把 x, y 丟回來。

1.2 快速冪

既然是數論，我們常常需要求 a^n 這樣的東西。但是 $O(n)$ 的做法看起來很暴力，所以「快速冪」就是要快速地求出冪次。

想法就是如果把 n 進行二進位分拆，那就挑那些 1 的位乘起來就好，而 $a, a^2, a^4, \ldots$ 可以透過自己乘自己很快地求出來，因此複雜度是 $O(\log n)$。實際寫起來大概長得像這樣：

Algorithm 2: Fast Power in C++

```
1 int power(int a, int n) {
2     int ans = 1;
3     for (; n; n /= 2) {
4         if (n & 1) ans = ans * a;
5         a = a * a;
6     }
7     return ans;
8 }
```

因為結果數字會很大，通常會要模 m，那就在每個乘法的地方模一次 m 就可以了。

另一種實作方法是 $a^n = ((a^2)^{\lfloor \frac{n}{2} \rfloor}) \times a^{n \bmod 2}$，寫起來大同小異。

1.3 同餘

如果有兩個數 a, b 滿足 $m|a-b$，那我們說這兩個數模 m 同餘，記為 $a \equiv b \pmod{m}$。

同餘有一些很基本的性質，例如加、減、乘法都可以如常運作。因為實在是太基本了，所以這裡只列出可能會用到又不太基本的性質 (?)：

1. (費馬小定理) 如果 p 是質數，那麼 $a^p \equiv a \pmod{p}$。

2. (歐拉定理) 如果 a, n 互質，那麼 $a^{\phi(n)} \equiv 1 \pmod{n}$，這裡 $\phi(n)$ 是小於等於 n 且和 n 互質的數的個數（歐拉函數）。

3. (歐拉判別法) 如果 p 是質數且 d 不是 p 的倍數。那麼存在 $x^2 \equiv d \pmod{p}$ 若且唯若 $d^{\frac{p-1}{2}} \equiv 1 \pmod{p}$。

值得一提的是，這三個性質配著快速冪一起吃都格外地美味。

同餘的運算中有加法、減法、乘法，可惜除法不是那麼的完全。如果有一個數 $c^{-1}c \equiv 1 \pmod{m}$，那乘 c^{-1} 就可以當作除 c 來用。根據 Bezout's Theorem，c^{-1} 存在若且唯若 $\gcd(c, m) = 1$，也就是 c, m 互質。c^{-1} 即稱為 c 的模逆元。

模逆元可以直接由擴展歐幾里德求，複雜度是 $O(\log(\min(c, m)))$。不過如果 m 是質

數，由費馬小定理知道 c^{m-2} 其實就是 c 的模逆元。配著快速冪的話複雜度是 $O(\log m)$，複雜度差不多，而且快速冪更好寫（而且通常是需要寫），如果是要求質數的模逆元的話，不失為一個替代方案。

1.4 質數篩法

建質數表，判斷質數之類的東西已經考到快爛了，怒直接講最常用的 XD

要找出 $[2, N]$ 中的所有質數，國小就教過從 2 開始把數一個一個拿出來，把它的倍數全部畫掉，這樣的複雜度是 $O(n \log n)$，原因是 $\sum_{i=1}^{n} \frac{1}{i} = O(\log n)$。然而可以做幾個改進。第一個觀察是：只需要把質數拿出來，將它的倍數們篩掉就好，不需要將合數拿出來篩，可惜這只是個常數優化。第二個觀察是：如果現在要把 i 的倍數全都篩掉，只需要從 i^2 開始就好，因為對於比它小的 i，它的倍數一定會有其它更小的質因數，所以老早就被篩掉了。於是複雜度改進到 $O(n \log \log n)$。實作起來長得像這樣：

Algorithm 3: Sieve of Eratosthenes in C++

```
1 bool prime[N];
2 void sieve(int n) {
3     for (int i = 2; i <= n; i++) prime[i] = true;
4     for (int i = 2; i <= n; i++)
5         if (prime[i])
6             for (int j = i * i; j < n; j += i) prime[i] = false;
7 }
```

值得一提的是質數篩法最優可以達到 $O(n)$，其想法也不難：每次跑到 i 時，都把 ip 刪去，其中 p 是小於等於 i 的最小質因數的所有質數。如此一來，對於所有最小質因數是 q 的合數 n，它只會在 $i = \frac{n}{q}$ 時被篩掉，複雜度是 $O(n)$。

1.5 質數判斷

如果不依靠篩法，要判斷一個數 N 是否是質數，最簡單的想法就是從 2 到 $\sqrt{N}$ 判斷是否整除 N。複雜度是 $O(\sqrt{N})$。

有一個近似演算法稱為 Miller-Rabin 演算法。其精髓是費馬小定理以及一個質數才有的性質：$x^2 \equiv 1 \mod p$ 若且唯若 $x \equiv \pm 1 \mod p$。隨便給一個 a，要看看它是否通過了這兩個性質的考驗。如果沒通過，那麼它就是合數。如果通過了，代表它有 75% 的可能性是質數。

這演算法不保證正確性，但隨機用很多種 a 測試的話其正確性是頗高的。另外，如果已知數字範圍在 int 內，取 $a = 2, 7, 61$，如果三個都通過考驗，那麼它必定是質數。

整體的複雜度是 $O(\log n)$，相當快速。

Algorithm 4: Miller-Rabin in C++

```
bool miller_rabin(int n, int a) {
    if (n % 2 == 0) return n == 2;
    int u = n - 1, t = 0;
    for (; u & 1; u /= 2, t++);
    int a = power(a, u); //a^u 模 n 的快速冪
    if (a == 1 || a == n - 1) return true;
    for (i = 0; i < t; i++) {
        a = a * a % n;
        if (a == 1) return false;
        if (a == n - 1) return true;
    }
    return false;
}
```

順帶一提，目前最快的確定性質數判斷演算法稱為 AKS 演算法，複雜度沒有人知道，但已知在 $O(\log^6 n)$ 以內。

如果要質因數分解的話，可以直接從 2 到 $\sqrt{n}$ 找因數，複雜度 $O(\sqrt{n})$。也可以採用 Pollard's rho Algorithm，過程過於複雜，有興趣的請自行搜尋。複雜度應該是 $O(\sqrt{p})$，其中 p 是 n 最大的質因數，但是尚未被嚴謹證明。

如果要同時將 2 至 n 的數質因數分解只要在篩法時同時實行 DP 就好，複雜度不變。但是展開一個數的質因數分解式需要額外代價，複雜度會變為質因數個數，全部展開的話會到 $O(n \log n)$。

1.6 盧卡斯定理與倒數枚舉 (?

這裡再舉兩個和數論有關的東西，然而它們比較不常出現。

盧卡斯定理是說，對於所有的質數 p 和非負整數 m, m_0, n, n_0，都有

$$C_{pn+n_0}^{pm+m_0} \equiv C_n^m C_{n_0}^{m_0} \pmod{p}$$

這裡約定若 $m < n$，則 $C_n^m = 0$。如此一來就可以 $O(p \log p)$ 的預處理，$O(\log \min(m, n))$ 的時間內算出 C_n^m 除以 p 的餘數。

Algorithm 5: Lucas' Theorem in C++

```
int comb(int m, int n, int p) {
    if (m < n) return 0;
    if (m < p && n < p) return fac[m] * inv_fac[n] % p * inv_fac[m - n] % p;
    //這裡 fac 是階乘、inv_fac 是階乘在模 p 之下的模逆元，兩者皆需預處理好
    return comb(m / p, n / p) * comb(m % p, n % p) % p;
}
```

第二個是少見複雜度是 $O(\sqrt{n})$ 的演算法。

已知給定一個 n，$\lfloor \frac{n}{x} \rfloor$ 的取值只會有大約 $2\sqrt{n}$ 種。如何枚舉出這些值呢？當然可

以 x 從 1 到 n 直接掃一遍。不過這樣太費工了，事實上，從 $T_0 = 1$ 開始，令 $T_{i+1} = \lfloor n/\lfloor \frac{n}{T_i+1} \rfloor \rfloor$，那 T 數列就是所有的取值（$T_k = n$ 之後的項不算）。

1.7 習題

1. (No judge) 請試著實作篩法與質因數分解。

2. (ZJ a289) 求 a 模 m 的模逆元。

3. (No judge)(中國剩餘定理) 請設計一套演算法，能計算滿足 $x \equiv a_i \pmod{b}_i (i = 0, 1, \ldots, n-1)$ 的 x 的通式。

4. (TIOJ 1425) 令 $f(n)$ 代表 n 的相異質因數個數。給你一個 $n \leq 10^7$，請求出 $x + 16^{f(\lfloor \frac{n}{x} \rfloor)}$ 的最小值。多筆測資，共有 10^4 筆測資。

5. (Codeforces 547A) 有隻熊，牠養了個青蛙和花。青蛙和花一開始身長/高 h_1, h_2。每過一天，青蛙的身長會變為 $x_1 h_1 + y_1 \bmod m$，花的高會變為 $x_2 h_2 + y_2 \bmod m$。請問哪一天青蛙和花的身長/高第一次分別是 a_1, a_2，或者永遠不可能到達。

6. (No judge) 給你兩個數 d, n，請在 $O(\sqrt{n})$ 的時間內判斷是否存在 x 使得 $x^2 \equiv d \pmod{n}$。

2 矩陣

相信大家都會矩陣的加法以及乘法的定義，這裡就清描淡寫就好了。

1. 如果 A, B 都是 $m \times n$ 的矩陣，那麼 $(A + B)_{ij} = A_{ij} + B_{ij}$。

2. 如果 A 是 $m \times l$ 的矩陣，B 是 $l \times n$ 的矩陣，那麼 $(AB)_{ij} = \sum_{k=0}^{l} A_{ik} B_{kj}$。

3. 矩陣加法有交換律和結合律；乘法沒有交換律，但有結合律。

矩陣加法 trivial，而矩陣乘法樸素做法是 $O(n^3)$，而 $O(n^{\log_2 7})$ 的做法之前在 D&Q 的課程也講過了。矩陣的冪次，使用乘法搭配快速冪即可。

2.1 高斯消去法

高斯消去法的意思是說用以下的三種操作使得一個矩陣除了左上至右下的對角線是 1 之外，其它都是 0。

1. 第 i 列和第 j 列交換

2. 第 j 列同乘一個常數

3. 第 i 列加上 α 倍的第 j 列 (α 是一個常數)

想法是：對於某一行，挑某一列在那行不是 0，然後把用第三種操作將其它列的那行消成 0，必要的時候可以交換兩列。至於實作的方法因人而異，而且有點繁雜，這裡就不詳細寫了。複雜度是 $O(n^2m)$。

然而如果是整數運算的話，直接使用上面的想法是無法將左上至右下的對角線都消成 1 的。這個時候有個替代方案：允許對角線上的數字可以不是 1，而可以是任何非零整數。

註：有些矩陣沒辦法經由高斯消去法達成條件（例如在主對角線上有 0），這個時候代表它的「秩」小於 n；如果是方陣，就代表它不可逆。

高斯消去法求聯立方程的解是高中數學範圍，就不多提了。

2.2 計算反矩陣

如果 A 是一個方陣，在它行列式不為 0 的時候都存在一個反矩陣 A^{-1}（可以 google「餘因子矩陣」）。高斯消去法也可以幫助我們找出反矩陣。想法就是實作的時候在 A 右邊放一個單位矩陣，一起施行高斯消去法。假設過程中左邊的矩陣變為 X，右邊的矩陣變為 Y，可以證明 $X = YA$ 永遠都會保持住。當高斯消去法結束時，$X = I$，所以 $Y = A^{-1}$，複雜度是 $O(n^3)$（如果採用輾轉相除的話，複雜度是 $O(n^3 \log c)$）。

2.3 計算行列式

高斯消去法也可以幫助我們算行列式。每次將兩列交換時行列式變號，進行第 2 個操作時行列式也乘上相同的常數。而高斯消去法終止時，行列式就是對角線元素的乘積。

然而如果要算行列式，只要讓整個矩陣是上三角矩陣就好了，也就是說高斯消去法可以不用做得那麼澈底，可以稍微加快效率。複雜度同樣是 $O(n^3)$。

2.4 快速求線性遞迴

給定 $k_1, k_2, \ldots, k_n$ 與初始項 $a_0, a_1, \ldots, a_{n-1}$，之後的每一項都滿足 $a_i = \sum_{j=1}^{n} k_j a_{i-j} (i \geq n)$，這種東西我們稱為 n 階的「線性遞迴」。雖然可以用 DP 求出 $a_0, \ldots, a_N$ 的值，不過矩陣可以幫助我們更快地求出單一個 a_N 的值。

以下以最常見的費波那契數列為例。$a_0 = a_1 = 1$,$a_i = a_{i-1} + a_{i-2}$。一個神奇的發現是：

$$\begin{bmatrix} a_{i+1} & a_i \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} a_{i+2} & a_{i+1} \end{bmatrix}$$

所以

$$\begin{bmatrix} a_1 & a_0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^N = \begin{bmatrix} a_{N+1} & a_N \end{bmatrix}$$

因此只要利用快速冪算出那個 N 次方就好了。複雜度 $O(\log N)$。也可以用相同的方法將其推廣到 N 階線性遞迴。

2.5 矩陣的其它用途

矩陣除了是一個「壓縮聯立方程組」的好工具以及快速求線性遞迴的好工具外，它還有其它較為少見的用途。

其一是作為優化 DP 的工具。如果一個滾動的二維 DP 轉移式恰恰好跟矩陣乘法一樣的話，那麼可以利用快速冪或者 Strassen 演算法優化複雜度。不過如果只是跟矩陣乘法「類似」的話（加法和乘法換成兩個有交換以及結合律的運算），則有可能會因為失去某些性質而無法直接套用；儘管如此，把它寫成矩陣後還是能稍微優化一點點的複雜度，比如說全點對最短路徑得以優化為 $O(\frac{n^3 \log\log n}{\log n})$（可以 google「(min,+) 矩陣」）。當然，在競賽中不太會有這種噁爛的東西，通常可以直接用快速冪或 Strassen 解決。

其二是作為一張圖的表示方法：鄰接矩陣。雖然聽到「鄰接矩陣」這四個字大家第一個想到的一定是「MLE」，但是在記憶體足夠的情況下，有時候鄰接矩陣可以告訴我們非常多的訊息。事實上，在數學的圖論中就常常使用這種方法。

如果你真的那麼怕 MLE 的話，就用 bitset 吧！（當然前提是這張圖沒有重邊，而且你沒有要對這個矩陣做非 0/1 的運算。）

2.6 習題

1. (No judge) 實作高斯消去法和輾轉相除的高斯消去法。

2. (ZJ a410) 解二元一次聯立方程式。

3. (ZJ d639) 有一個數列 $a_1 = a_2 = a_3 = 1$，且 $a_i = a_{i-1} + a_{i-2} + a_{i-3}, \forall i \geq 4$，請求出 a_n 模 10007 後的結果。$n < 2^{31}$。

4. (TIOJ 1428) 有一個 $N \leq 150$ 個點的圖（不一定是簡單圖）以及 $Q \leq 20000$ 個詢問。給定對於每筆詢問 (i, j)，請計算從某個點 i 花 k 步走到 j（可走重複的點）的方法數模 $10^9 + 9$ 的結果。

3 組合

3.1 組合計數

以下提供幾個比較常用到的東西。大部分都是高中數學範圍。

1. 加法原理、乘法原理、排容原理

2. m 個東西中取 n 個東西排列的方法數 $P_n^m = \frac{m!}{(m-n)!}$

3. m 個東西中取 n 個東西的方法數 $C_n^m = \frac{m!}{n!(m-n)!}$

值得一提的是 P 和 C 的大小都誇張到了一個極點（C 是指數等級，P 是階乘等級），所以通常不是要模一個數字就是要用浮點數運算（ex. 算機率）。排容原理的複雜度也是指數，所以通常會在測資範圍很小的時候出沒。也千萬不要小看排容原理的 coding 複雜度，只要是用到它的暴搜題，實作難度都不是普通的高。

3.2 組合恆等式

以下列了幾個，不過其實也沒有很常用。為了以防萬一，知道一下比較好。

1. $C_n^m = C_{n-1}^{m-1} C_n^{m-1}$

2. $\sum_{i=0}^{m-n} C_n^{n+i} = C_{n+1}^{m+1}$

3. (二項式定理)$(1+x)^n = \sum_{i=0}^{n} C_i^n x^i$

另外，遞迴與組合計數直接理解成 DP 你會比較好過。不過如果遞迴式可以用上面那三條化簡的話就化簡吧。除此之外，如果它是線性遞迴，請配合矩陣快速冪解決。

3.3 卡特蘭數

有一個 $n \times n$ 的網格紙，被一個左下到右上的對角線切成兩半。有一個小螞蟻從 $(0,0)$ 出發，要到 (n,n)。請問小螞蟻在只往右或往上走的條件下，不跨越對角線的走法有幾個？

我們假設這樣的答案叫作 C_n。一開始你會發現

$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$$

結合 $C_0 = 1$ 你可以用 $O(n^2)$ 的時間算出 C_n。然而這樣還不夠快。事實上可以證明

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

如此複雜度降低到 $O(n)$。卡特蘭數十分重要，以下的題目答案都是卡特蘭數：

1. n 個節點的二元樹個數

2. n 對括號合法匹配的方法數

3. 凸 $n+2$ 邊形三角化的方法數

4. 著名 stack 練習題 Rails (UVa 514) 中答案是 Yes 的排列數

3.4 忽略足夠小的數

通常是在算機率的時候會用到。因為通常輸出浮點數的時候要求的精確度有限，所以可以將絕對不會影響到精確度的部分捨去以降低複雜度。這要求非常高的估計技巧，所以如果你不會的話……自己調個常數看看會不會既不 TLE 又不 WA 就好了 (?

啊對了我從來都沒有成功辦到這種事……如果有興趣的話拜 CBD 為師吧 XD

3.5 習題

1. (ZJ b229)(2009 TOI 初選第一題) 從原點開始，每次只能往右，往上或往左走一單位，且不能走重複的邊。請問長度為 n 的路徑總數是多少？$n \leq 50$。

2. (No judge) 承上題，請輸出模 $10^9 + 7$ 後的答案，$n \leq 10^8$。

3. (TIOJ 1607) 計算寬度為 n 的山梭線總數模 $10^9 + 7$。$n \leq 10^6$，10^5 筆測資。

4. (TIOJ 1594) 承上題，把答案改模 11。$n < 2^{31}$，10^6 筆測資。

5. (Codeforces 559D) 給你一個凸多邊形（頂點座標依逆時鐘方向給）。從凸多邊形的頂點中隨機挑個子集使得圍出的面積不是零（也就是不退化），求所挑出的多邊形嚴格地包住的格點數的期望值。

(提示：忽略小數字以節省時間。)

4 計算幾何

4.1 浮點數誤差

先看看這段 C++ code：

Algorithm 6: BOOM!!!

```
1 #include <cstdio>
2 int main() {
3     for (double i = 0; i != 1; i += 0.0001)
4         if (i > 1) puts("BOOM");
5 }
```

它會陷入無窮迴圈。為什麼呢？不是迴圈執行了 10000 次之後 i 就會變成 1，就會跳離迴圈了嗎？

要記住，浮點數運算是有誤差的。當它加了 10000 次 0.0001 之後，並不會變成 1，而會跟 1 差一點點。所以當我們要判斷兩個浮點數是否相等的時候，事實上是問說這兩個數夠近嗎？所謂夠不夠近的「標準」，我們通常叫它 ϵ 或 eps（也就是微小誤差的意思）。如

果兩個數的差值小於 eps，我們就說它們是相等的。

eps 的取值同樣來自高超的估計技巧，不會的話就自己亂試常數吧。

Algorithm 7: Correct code using eps

```
#include <cstdio>
#include <cmath>
int main() {
    int cnt = 0;
    double eps = 1e-6;
    for (double i = 0; abs(i - 1.0) > eps; i += 0.0001) cnt++;
    printf("%d\n", cnt);
}
```

現在它會乖乖地吐出 10000 給你了。其實最好的方法就是，如果題目不要求浮點數，那就不要用浮點數運算。

4.2 內積

兩個向量 $\vec{v_1}(x_1, y_1), \vec{v_2}(x_2, y_2)$ 的內積定義為 $\vec{v_1} \cdot \vec{v_2} = x_1x_2 + y_1y_2$。事實上它也是 $|\vec{v_1}||\vec{v_2}| \cos\theta$（這裡 θ 是兩向量的夾角）。利用這件事實和 arccos 不難算出兩向量間的夾角。

比較常見的用法是判斷兩向量是否垂直，如果垂直的話內積會是 0。

4.3 外積

兩個向量 $\vec{v_1}(x_1, y_1), \vec{v_2}(x_2, y_2)$ 的外積 $\vec{v_1} \times \vec{v_2} = x_1y_2 - x_1y_2$。事實上它也是 $|\vec{v_1}||\vec{v_2}| \sin\theta$（這裡 θ 是兩向量的夾角）。

外積有負交換律。這是因為 θ 的意思是由 $\vec{v_1}$「逆時鐘」轉到 $\vec{v_2}$ 的夾角。所以外積通常是用來判斷兩向量之間的夾角是順時鐘還是逆時鐘（看它的正負號），在凸包算法中扮演著極重要的角色。除此之外，因為長度為 a 和 b 夾角為 θ 的三角形面積是 $\frac{1}{2}ab\sin\theta$，所以外積也常來計算面積。

其實外積是個向量，但是對於二維空間的向量，它的方向不是向上就是向下，所以可以用正負號代表一切。

4.4 判斷各種線相交

判斷相交算是基礎又惹人厭的一個東西。如果你同時要判斷線段相交，半線和線段相交，半線和半線相交……等等。勸你還是直接寫求交點的函數判斷吧……

想法是 $(x_1, y_1)(x_2, y_2)$ 可以定出一條直線 $(x_1 + t(x_2 - x_1), y_1 + t(y_2 - y_1))$（$t$ 是參數，這叫做直線的參數式）。直線代表 t 可以是任意數，射線代表 $t \geq 0$（或者 $t \leq 1$），半線代表 $t \leq 0$ 或 $t \geq 1$，線段代表 $0 \leq t \leq 1$。好處是可以同時解決一堆 case 而不用重寫函式，

而且如果座標都是整數的話其實可以只用整數運算。

另有一種判斷線段相交的方法，如果只有線段的 case 的話是不錯的選擇。想法就是：如果要 $\overline{AB}$ 和 $\overline{CD}$ 相交的話，A, B 要在 $\overline{CD}$ 的異側，C, D 要在 $\overline{AB}$ 的異側。所以我們只需要判斷兩個點是否在直線的兩側。這時外積就出動了！不難發現 A, B 在 $\overline{CD}$ 的異側若且唯若 $\overrightarrow{AC}, \overrightarrow{AD}$ 的夾角和 $\overrightarrow{BC}, \overrightarrow{BD}$ 的夾角一個是逆時鐘的，一個是順時鐘的。也就是說 $\overrightarrow{AC} \times \overrightarrow{AD}$ 和 $\overrightarrow{BC} \times \overrightarrow{BD}$ 一正一負。

4.5 計算面積

剛剛講過外積可以計算面積。如果 $P_1 = P_{n+1}, \ldots, P_n$ 是一個凸 n 邊形，那麼隨便取一個原點 O，這個凸 n 邊形的面積會是

$$\frac{1}{2}\left|\sum_{i=1}^{n} \overrightarrow{OP_i} \times \overrightarrow{OP_{i+1}}\right|$$

可以用有向面積推得這個結果。你也可以把 O 取在凸 n 邊形內驗證看看。

4.6 凸包

最好寫的凸包演算法應該就是 Andrew's monotone chain 了。想法是將所有點依 x 軸排序之後，再從左到右掃描，同時維護它的「下凸包」以及「上凸包」。另外一種實作方式是從左往右找下凸包，再從右往左找上凸包。只需要用外積判斷這個點會不會在當前的凸包上即可。（其實這個過程有點像是 DP 優化裡套用單調隊列的斜率優化。）

4.7 掃描線與旋轉卡尺

掃描線（sweep line）可以想像成一條前進的直線，遇到某些點的時候會停留並維護它所需要維護的東西，一旦掃完整個平面，演算法即完成。也可以理解成把 n 維中的其中一維視為「時間」後，開始用各種資料結構維護剩下 n-1 維的答案。

而旋轉卡尺（rotating calipers）可以想像成兩條平行直線（卡尺）緊靠著一些東西，而把兩條線旋轉一周，每當碰到的點改變的時候就維護它要維護的東西。

這兩個都是計算幾何中很經典的想法，可以解決很多經典的計算幾何問題。掃描線的應用比較廣泛。

4.8 習題

1. (No judge) 實作判斷各種線相交的函數。

2. (No judge)(2016 北市賽) 判斷平面上 N 個線段是否有任何交點。時間複雜度 $O(N \log N)$。

3. (TIOJ 1178) 裸凸包。

4. (TIOJ 1500) 裸最近點對。請使用非 D&Q 的方法實作，複雜度 $O(n \log n)$。

5. (No judge) 請實作 d 維空間的最近點對，複雜度 $O(n \log^{d-1} n)$。
（註：事實上對於任一固定維度 d，最近點對都有 $O(n \log n)$ 演算法。）

6. (TIOJ 1105) 裸最遠點對。測資很小，但請寫複雜度 $O(n \log n)$ 的方法。

7. (Codeforces 559A) 給你一個內角全都是 120° 的六邊形還有它的六個邊長（都是正整數），請算出這個六邊形可以被幾個邊長為 1 的正三角形填滿。
（註：想想不用面積怎麼做？用面積怎麼做？）

8. (HOJ 404)(目前潰爛中) 平面上有一些紅點和一些藍點。任三點不共線。兩個紅點連線構成紅線段，兩個藍點連線構成藍線段。請算出紅線段與藍線段相交的次數。（也就是說如果有兩個紅線段跟一個藍線段共點的話，還是算兩次）。請嘗試在 $O(n^2 \log n)$ 的時間內完成。
（註：這題極度血腥請小心 XD）

9. (TIOJ 1224) 求平面上 $N \leq 10^5$ 個矩形覆蓋的總面積。
（提示：聽說這在線段樹的習題裡出現過了。）

10. (TIOJ 1041) 平面上有兩組點，請判斷可不可以用兩條平行直線把這兩組點分開，如果可以，請輸出這兩條線最遠的距離。測資很小，但請寫複雜度 $O(n \log n)$ 的方法。
（提示：小心浮點數。）