基礎數學

sam571128

October 28, 2021

競賽中會用到的數學

競賽或演算法的題目當中，用到的數學有以下幾種

- 數論 - 最大公因數、模運算等等
- 排列組合
- 線性代數 - 矩陣、向量空間等
- 幾何 - 向量、外積等等

而我們今天會從數論開始講起

數論?

數論，基本上光看他的名字，就可以知道他是在討論數字之間的關係，而其實在我們從小到大的數學課中，也遇過很多與數論相關的題目。這裡就讓我們用「最大公因數」做開頭。

最大公因數 (Greatest Common Divisor)

最大公因數，是兩個數字 x, y 最大的共同因數。

國小時，應該有教過短除法可以找到兩個數字的最大公因數

最大公因數 (Greatest Common Divisor)

$$\begin{array}{r|rr} 5 & 25 & 20 \\ \hline & 5 & 4 \end{array}$$

最大公因數: 5

最小公倍數: 5*5*4

最大公因數 (Greatest Common Divisor)

不過，這樣的方式其實不是非常有效率，因為當我們遇到兩個數字如: $1469, 221$，你會做的事情就是從 2 開始不停地去試各種質數，看最後哪個數字能同時整除兩個數字。

最大公因數 (Greatest Common Divisor)

如果我們換成是用電腦算，那時間複雜度其實會是 $O(n)$，不夠快速!

最大公因數 (Greatest Common Divisor)

因此，上了國中之後，老師跟你說:「算最大公因數有另外一種方式，也就是輾轉相除法!」

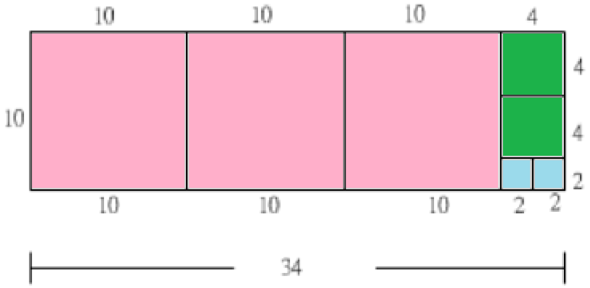
最大公因數 (Greatest Common Divisor)

6	1469	221	1
	1326	143	
1	143	78	1
	78	65	
5	65	13	
	65		
	0		

最大公因數 (Greatest Common Divisor)

而藉由這種方式，我們就可以靠著除法來計算最大公因數。而這種演算法，其實是歐幾里德 (寫幾何原本那個) 提出的一種演算法，英文被稱為 Euclid's Algorithm.

最大公因數 (Greatest Common Divisor)



最大公因數 (Greatest Common Divisor)

The proof is trivial and is left for the readers as an exercise

最大公因數 (Greatest Common Divisor)

可以發現這樣做可以很快速地找到兩個數字的最大公因數，時間複雜度最差會是 $O(\log n)$(可以很簡單的觀察到，因為每次都會除掉一個數字，最多只會有 $\log$ 次)，而實際上他的最差情況下只會發生在兩個費氏數列中

最大公因數 (Greatest Common Divisor)

寫法如下:

```
int gcd(int a, int b){
    if(a < b) swap(a,b);
    if(b == 0) return a;
    return gcd(b,a%b);
}
```

最大公因數 (Greatest Common Divisor)

不過 C++ 其實有內建 GCD 的函數，在 C++11 以後，可以用__gcd(a,b)，C++17 以後可以用 gcd(a,b)。而補充一下：兩個數字的 LCM (最小公倍數)，在 C++17 前必須用a*b/__gcd(a,b)，C++17 以後可以用 lcm(a,b)。

質因數分解

當我們要找一個數字 x 的質因數分解時，國中的時候，大概都會教大家使用短除法來找質因數分解，而老師應該都會告訴你，我們只要試除到根號即可!

質因數分解

```
for(int i = 2;i*i <= x;i++){
    while(x%i==0){
        //i 是 x 的因數
        x /= i;
    }
}
```

質因數分解

這樣的時間複雜度是 $O(\sqrt{n})$

質因數分解

不過我們其實有個更快速的做法可以去做質因數分解! 我們稱他為埃氏篩 (sieve of Eratosthenes)，而其實之前在時間複雜度的時候我有提過他可以判斷質數。

質因數分解

```
for(int i = 2;i < N;i++){
    if(prime[i]){
        for(int j = i*i; j < N; j += i){
            prime[j] = false;
        }
    }
}
```

判斷質數 $O(n \log \log n)$

質因數分解

不過，要怎麼做質因數分解呢?

我們可以在做埃氏篩時，順便維護一個數字 x 的最大質因數 (Largest Prime Factor)，藉此在 $O(n \log \log n)$ 的預處理時間完成這個問題

質因數分解

```
for(int i = 2;i < N;i++){
    if(lpf[i]==1){
        //lpf 表示一個數字的最大質因數
        for(int j = i*i; j < N; j += i){
            lpf[j] = i;
        }
    }
}

while(lpf[x]!=1){
    //這些 lpf[x] 就是 x 的質因數
    x /= lpf[x];
}
```

質因數分解 $O(n \log\log n)$ 預處理/ $O(\log n)$ 分解

質因數分解

不過，事實上有另外一種更快的埃氏篩法，可以在線性的時間完成預處理，我們稱其為**線性篩**

質因數分解

```cpp
vector<int> primes; //存所有質數
for(int i = 2;i < N;i++){
    if(lpf[i]==1){
        //lpf 表示一個數字的最大質因數
        primes.push_back(i);
    }

    for(int p : primes){
        if(i*p >= N) break;
        lpf[i*p] = p;
    }
}
```

質因數分解 $O(n)$ 預處理

模運算與同餘 (Modular Arithmetic & Congruence)

模運算對於大家應該是一個新的概念，而什麼是模 (mod) 呢?

模運算與同餘 (Modular Arithmetic & Congruence)

其實他就是程式中我們使用的 $\%$，也就是當 $a\%b$ 時，答案是 a 除以 b 的餘數，通常在數學式子當中會寫成 $a \pmod{b}$，而在程式中使用到這個概念時，通常會是在答案很大時，題目會要你輸出答案 $\pmod{p}$。

模運算與同餘 (Modular Arithmetic & Congruence)

所以說，餘數有什麼性質呢?

模運算與同餘 (Modular Arithmetic & Congruence)

模運算，通常我們會用一種稱為同餘的寫法表達，寫法如下:

$$a \equiv b \pmod{p}$$

表示 a 除以 p 的餘數與 b 除以 p 的元素相同

模運算與同餘 (Modular Arithmetic & Congruence)

模運算主要有以下性質:

- $a + b \pmod p \equiv a \pmod p + b \pmod p$ (加法)
- $a - b \pmod p \equiv a \pmod p - b \pmod p$ (減法)
- $a \cdot b \pmod p \equiv a \pmod p \cdot b \pmod p$ (乘法)
- $a/b \pmod p \not\equiv a \pmod p / b \pmod p$ (**不滿足**除法)

加法的證明:

我們可以將一個數字 a 除以另外一個數字 b 寫成 $a = bq + r$

因此當 $a_1 = bq_1 + r_1$ 與 $a_2 = bq_2 + r_2$ 時，$a_1 + a_2 = b(q_1 + q_2) + (r_1 + r_2)$

所以我們可以得到 $a_1 + a_2 \equiv r_1 + r_2 \pmod{b}$

得證。

模運算與同餘 (Modular Arithmetic & Congruence)

減法的證明:

我們可以將一個數字 a 除以另外一個數字 b 寫成 $a = bq + r$

因此當 $a_1 = bq_1 + r_1$ 與 $a_2 = bq_2 + r_2$ 時，$a_1 - a_2 = b(q_1 - q_2) + (r_1 - r_2)$

所以我們可以得到 $a_1 - a_2 \equiv r_1 - r_2 \pmod{b}$

得證。

模運算與同餘 (Modular Arithmetic & Congruence)

乘法的證明:

我們可以將一個數字 a 除以另外一個數字 b 寫成 $a = bq + r$

因此當 $a_1 = bq_1 + r_1$ 與 $a_2 = bq_2 + r_2$ 時，

$a_1 a_2 = (bq_1 + r_1)(bq_2 + r_2) = b(bq_1 q_2 + q_1 r_2 + r_1 q_2) + r_1 r_2$

所以我們可以得到 $a_1 \cdot a_2 \equiv r_1 \cdot r_2 \pmod{b}$

得證。

模運算與同餘 (Modular Arithmetic & Congruence)

而模運算無法直接使用兩個數字除出來的餘數去做除法。因此，我們會需要另外一種東西來處理這件事。

模運算與同餘 (Modular Arithmetic & Congruence)

當我們在做兩個數字 a, b 的除法時，七年級應該教過，我們可以將其寫為 $a \cdot b^{-1}$，而一個數字 x 的 -1 會被稱為 x 的**反元素**。而在模運算系統中，我們沒有除法，但是同樣有反元素，我們稱其為 **模反元素**或 **模逆元**

模運算與同餘 (Modular Arithmetic & Congruence)

在 mod 不是質數時，可能會遇到一個數字沒有模反元素的情況，但我們接下來會以 mod 是質數為主。

模運算與同餘 (Modular Arithmetic & Congruence)

而如何找一個數字的模反元素呢? 這裡我們要先回到如何快速找一個數字的次方開始。

快速冪

假設今天題目問你 $x^{1}00$，你應該會想到，那麼我們就直接暴力乘開，反正對電腦來說，乘 100 次不是什麼大問題。

快速冪

但要是今天問你 x^{10^9} 之類的很大的數字呢?

你可能就會放棄這個想法了!

快速冪

不過，要找一個數字的 n 次方真的需要乘那麼多次嗎?

快速冪

答案是不用，假設我們今天想要算 x^4 次方時，你真的會去將 x 乘 4 次嗎? 還是就直接拿 x^2 去平方了呢?

同理，$x^8, x^{16}, x^{32}, \cdots$，你應該會想要使用平方的方式來做計算，因為這個數字設計得非常漂亮

快速冪

那假設今天是一個很奇怪的數字，像是 x^{25} 次方呢?

快速冪

我們其實可以把它拆成 $x^{25} = x \cdot x^8 \cdot x^{16}$，而使用這個方式，我們其實只需要乘三個數字，而乘法次數只要 $3 + 4$ 次! 比 25 次少了好幾倍!

快速冪

根據這個想法，我們可以將他寫成程式碼

快速冪

```
int fastpow(int n, int p){
    int res = 1;

    while(p){
        if(p % 2 == 0) res = res * n;
        n = n * n;

        p /= 2;
    }
    return res;
}
```

快速冪時間複雜度 $O(\log_2 p)$

快速冪

```cpp
int fastpow(int n, int p){
    int res = 1;

    while(p){
        if(p % 2 == 0) res = res * n % MOD;
        n = n * n % MOD;

        p /= 2;
    }
    return res;
}
```

快速冪 (帶有模運算的) 時間複雜度 $O(\log_2 p)$

回到模反元素

那麼我們又要怎麼求一個數字的模反元素呢?

這裡，我們要引用一個數學定理「費馬小定理」

費馬小定理

當 p 是質數時，對於任何非 0 整數有以下性質

$$a^p \equiv a \pmod{p} \rightarrow a^{p-2} \equiv a^{-1} \pmod{p}$$

費馬小定理

證明的話，我們這裡省略，有興趣的可以來問我

費馬小定理

因此，當我們的 mod 是質數時，我們可以直接用 $a \cdot b^{p-2}$ 來表示 a/b。

回到同餘

模運算主要有以下性質:

- $a + b \pmod p \equiv a \pmod p + b \pmod p$ (加法)
- $a - b \pmod p \equiv a \pmod p - b \pmod p$ (減法)
- $a \cdot b \pmod p \equiv a \pmod p \cdot b \pmod p$ (乘法)
- $a / b \pmod p \equiv a \pmod p \cdot b^{p-2} \pmod p$ (當 p 是質數時成立)

回到同餘

而根據這樣，模運算就變得很簡單了!

貝祖定理 (Bezout’s Theorem)

下一個我們要來談的是「貝祖定理」

貝祖定理 (Bezout's Theorem)

貝祖定理

對於一個式子 $ax + by = c$，(x, y) 有整數解，若且為若 $gcd(a, b)|c$

貝祖定理 (Bezout's Theorem)

而如果我們想要找一組整數解，我們可以使用「拓展歐幾里德算法」或「拓展輾轉相除法」(Extended Euclid's Algorithm) 或直接寫為 extgcd

Extended Euclid's Algorithm

```
pair<int,int> ext_gcd(int a, int b){
    if(b == 0) return {1,0};
    else if(a == 0) return {0,1};
    pair<int,int> s,t = ext_gcd(b,a%b);
    return {t,s-a/b*t};
    //a*s + b*t =gcd(a,b)
}
```

拓展歐幾里德算法

數論總結

今天大概就講到這裡，明天我們會講一些與矩陣有關的東西

矩陣

今天要講的內容，高二的同學應該都在數學課上學過了，不過我們要講的東西就是矩陣!

矩陣

對於一個矩陣，我們會將數字寫在 $n \times m$ (列 $\times$ 行) 的方格中，以下為一個 2×3 的矩陣

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

矩陣

而首先，我們要先來講線性代數的基礎，也就是「高斯消去法」！

高斯消去法

假設今天你有一組聯立方程式

$$\begin{cases} 2x + 3y & = & 5 \\ x - 2y & = & -1 \end{cases}$$

你會怎麼解呢?

高斯消去法

$$\begin{cases} 2x + 3y & = & 5 \\ x - 2y & = & -1 \end{cases}$$

根據國中的經驗，你應該會使用「加減消去法」，也就是將第二式乘上 2 之後，減掉第一式。

高斯消去法

$$\begin{cases} 2x + 3y &= 5 \\ x - 2y &= -1 \end{cases} \Rightarrow \begin{cases} 2x + 3y &= 5 \\ 2x - 4y &= -2 \end{cases} \Rightarrow \begin{cases} 2x + 3y &= 5 \\ -7y &= -7 \end{cases}$$

高斯消去法

而這樣的想法，我們可以使用矩陣，來整理出一個更統整的做法。

高斯消去法

回到剛剛一開始的聯立式，我們可以將他寫成「增廣矩陣 (Augmented Matrix)」

$$\begin{cases} 2x + 3y & = & 5 \\ x - 2y & = & -1 \end{cases} \Rightarrow \begin{bmatrix} 2 & 3 & 5 \\ 1 & -2 & -1 \end{bmatrix}$$

而 $\begin{bmatrix} 2 & 3 \\ 1 & -2 \end{bmatrix}$ 被稱為「係數矩陣 (Coefficient Matrix)」

高斯消去法

對於一個矩陣，我們可以對他做以下三種操作，稱為「矩陣列運算」

1. 將兩列交換
2. 將某一列乘上 c 加到另一列
3. 將某一列乘上 c

高斯消去法

而我們會發現，其實當我們在做這件事情時，跟加減消去法做的事情非常像，因此，只要我們能用「矩陣列運算」將矩陣化簡為

$$\begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \end{bmatrix}$$

答案就出來了!

高斯消去法

實際做看看吧!

$$\begin{bmatrix} 2 & 3 & 5 \\ 1 & -2 & -1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & -2 & -1 \\ 2 & 3 & 5 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & -2 & -1 \\ 0 & 7 & 7 \end{bmatrix}$$

正常來說，在用手算的時候，算到這個時候就結束了

高斯消去法

不過我們繼續下去吧!

$$\begin{bmatrix} 1 & -2 & -1 \\ 0 & 7 & 7 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & -2 & -1 \\ 0 & 1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

因此答案為 $\begin{cases} x & = & 1 \\ y & = & 1 \end{cases}$

高斯消去法

而這個方式，不只能用在兩條方程式，很多條方程式時也可以用一樣的方式解決

高斯消去法

而要在電腦上做到這件事情非常簡單，我們只要照著這個方式有系統性地去做即可!

1. 枚舉第 i 行第 i 列的數字
2. 若該位置為 0，找同一行的第 i 個位置非 0 的列，並交換兩列
3. 將該位置的數字化為 1 (對整列乘上某個倍數)
4. 用這列乘上某個數字去削掉其他列的第 i 個位置

高斯消去法

這樣的寫法實際寫起來的時間複雜度會是 $O(n^3)$，而這裡不附上 code，但照著上面的做法去寫，應該能很容易的寫出來，而大家可以去寫寫看 TIOJ 2170 或 TIOJ 2012，

矩陣

讓我們接著來看看矩陣的運算

矩陣

而矩陣之間可以做的運算有以下幾種

1. 加法 (只能對維度相同的矩陣進行)
2. 純量乘法
3. 矩陣乘法 (若 $A \cdot B$，則 A 的 m 要等於 B 的 n)

矩陣

加法:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} \end{bmatrix}$$

矩陣

純量乘法:

$$c \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} = \begin{bmatrix} ca_{11} & ca_{12} & ca_{13} \\ ca_{21} & ca_{22} & ca_{23} \end{bmatrix}$$

矩陣

矩陣乘法:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} =$$

$$\begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix}$$

矩陣乘法: 看起來很複雜嗎? 沒關係! 我們可以寫成數學算式

$$c_{ij} = \sum_{k=1}^{A_m} a_{ik} \times b_{kj}$$

矩陣

矩陣乘法:

$$c_{ij} = \sum_{k=1}^{A_m} a_{ik} \times b_{kj}$$

而這個算式，我們可以很明顯看到，矩陣乘法可以在 $O(n^3)$ 完成

矩陣

矩陣乘法:

```
for(int i = 0;i < A.n;i++){
    for(int j = 0;j < B.m;j++){
        for(int k = 0;k < A.m;k++){
            c[i][j] += a[i][k] * b[k][j];
        }
    }
}
```

矩陣乘法

矩陣

然後在矩陣的世界當中，有一種矩陣，他不管跟誰乘，都不會改變原本的矩陣的值，也就是「單位方陣 I_n」

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

一個單位方陣就是左上右下的對角線是 1 剩下都是 0 的矩陣

矩陣

可以觀察到，單位方陣與 1 很像

單位方陣的性質

對於任意一矩陣 A，$I_n A = A = A I_m$

矩陣

矩陣其實有更多的性質，不過那留給大家以後在高二或大學的時候再繼續學，我們競賽中會需要用到的就只有這樣。

矩陣快速冪

而矩陣搭配我們昨天教的「快速冪」，可以拿來優化**線性遞迴**，什麼意思呢?

矩陣快速冪

舉個例子:

費氏數列

請輸出費氏數列第 n 項。

矩陣快速冪

費氏數列

請輸出費氏數列第 n 項。

你可以使用我們之前教過的 DP，在 $O(n)$ 的時間找到答案

矩陣快速冪

不過，我們將轉移式寫出來，會得到 $f(n) = f(n-1) + f(n-2)$

矩陣快速冪

不過，我們將轉移式寫出來，會得到 $f(n) = f(n-1) + f(n-2)$ 那麼，我們可以將費氏數列寫成 $\begin{bmatrix} f(n+1) \\ f(n) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix}$

矩陣快速冪

而我們稱 $\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ 是費氏數列的轉移矩陣

矩陣快速冪

因此，想要找費氏數列第 n 項，我們就只要算以下式子

$$\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

矩陣快速冪

而一個數字的 n 次方，我們可以使用快速冪，在在 $\log n$ 的時間算出來，那矩陣呢?

矩陣快速冪

而一個矩陣的 n 次方，我們可以使用快速冪，在在 $k^3 \log n$ (k 是矩陣大小) 的時間算出來。因為矩陣相乘要花 $O(k^3)$ 的時間

矩陣快速冪

因此，我們就可以在 $O(logn)$ 的時間算出某個線性 dp 的第 n 項了。所以如果以後遇到一個看起來很 dp 的題目，就使用矩陣快速冪即可

例題

2020 北市賽 pC

有 n 個位置，每個位置你可以放 6 種不同的棋，而在這 6 種棋當中，包含了國王與皇后，你希望國王與皇后都只能有偶數個，問有幾種排列方式?

範圍: $1 \leq n \leq 10^9$

例題

2020 北市賽 pC

有 n 個位置，每個位置你可以放 6 種不同的棋，而在這 6 種棋當中，包含了國王與皇后，你希望國王與皇后都只能有偶數個，問有幾種排列方式?

範圍: $1 \leq n \leq 10^9$

這題看到 n 的數字很大，大概會想到我們無法輕易地使用 dp 等方式完成。

例題

2020 北市賽 pC

有 n 個位置，每個位置你可以放 6 種不同的棋，而在這 6 種棋當中，包含了國王與皇后，你希望國王與皇后都只能有偶數個，問有幾種排列方式?

範圍: $1 \leq n \leq 10^9$

不過如果 $n \leq 10^6$ 呢?

例題

2020 北市賽 pC

有 n 個位置，每個位置你可以放 6 種不同的棋，而在這 6 種棋當中，包含了國王與皇后，你希望國王與皇后都只能有偶數個，問有幾種排列方式?

範圍: $1 \leq n \leq 10^9$

我們可以使用 $dp[i][0/1/2/3]$ 表示國王與皇后有奇數還是偶數個，我們就可以推出轉移式了

例題

2020 北市賽 pC

有 n 個位置，每個位置你可以放 6 種不同的棋，而在這 6 種棋當中，包含了國王與皇后，你希望國王與皇后都只能有偶數個，問有幾種排列方式?

範圍: $1 \leq n \leq 10^9$

而稍微想一下就能得到轉移矩陣，並解出這題了

(題外話: 去年我用數學推出答案，沒有使用矩陣)

例題

路徑數量

給你一張 n 個點, m 條邊的圖，問總共有多少長度為 k 的路徑?

範圍: $1 \leq k \leq 10^{18}$

這題也同樣是矩陣快速冪，但因為我們還沒講到圖論，我們等講到圖論再談

例題

TIOJ 2053 - 費氏數列

給你一個數列的開頭 x_1, x_2，給定 $x_n = bx_{n-1} + ax_{n-2}$，問第 n 項是多少?

範圍: $1 \leq n \leq 10^{18}$

這題大家可以去練習看看。

矩陣總結

我們矩陣大概就講到這裡，我們下次應該會來談計算幾何。