Advanced Competitive Programming

國立成功大學ACM-ICPC程式競賽培訓隊
nckuacm@imslab.org

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

質數

質數

- 質數判斷
- 質因數分解
- 質數篩檢 (生成)

質數判斷

- 若數 n>1 能被 1 與 n 以外的數整除，則 n 為合數

質數判斷

- 若數 n>1 能被 1 與 n 以外的數整除，則 n 為合數
- 假設 n 是**合數**，則 n=x·y 其中 1<x<y，
  顯然 x 的大小不超過 √n ;

質數判斷

- 若數 n>1 能被 1 與 n 以外的數整除，則 n 為合數
- 假設 n 是**合數**，則 n=x·y 其中 1<x<y，
  顯然 x 的大小不超過 √n ;
- 所以若有 x∈[2,√n] 滿足 x|n，則 n 不是質數。

質數判斷

- 若數 n>1 能被 1 與 n 以外的數整除，則 n 為合數
- 假設 n 是**合數**，則 n=x·y 其中 1<x<y，
  顯然 x 的大小不超過 √n；
- 所以若有 x∈[2,√n] 滿足 x|n，則 n 不是質數。

```
for (int i = 2; i <= sqrt(n); i++)
  if (n%i == 0) return false;
return true;
```

Number Theory

- 質數判斷
- 質因數分解
- 質數篩檢 (生成)

質因數分解

- 唯一分解定理：任何合數能分解成一些質數的積

質因數分解

- 唯一分解定理：任何合數能分解成一些質數的積
- 合數 $n = p_1^{t1} \cdot p_2^{t2} \cdots p_k^{tk}$ 為多個質數的乘積
  不失一般性的有 $p_1 < p_2 < \cdots < p_k < \sqrt{n}$

質因數分解

- 唯一分解定理：任何合數能分解成一些質數的積
- 合數 $n = p_1^{t1} \cdot p_2^{t2} \cdots p_k^{tk}$ 為多個質數的乘積
  不失一般性的有 $p_1 < p_2 < \cdots < p_k < \sqrt{n}$

- 所以在範圍 $[2, \sqrt{n}]$ **從小至大**找數試除即可。

質因數分解

```
for (int p = 2; p <= sqrt(n); p++) {
  int t = 0;
  while (n%p == 0) n /= p, t++;
  if (t) factors.push_back({p, t});
}

if (n != 1) factors.push_back({n, 1});
```

Number Theory

- 質數判斷
- 質因數分解
- 質數篩檢 (生成)

Sieve of Eratosthenes

- 從 2 開始，刪掉 2 的倍數

Sieve of Eratosthenes

- 從 2 開始，刪掉 2 的倍數
  找下一個未被刪掉的數，找到 3 ，刪掉 3 的倍數

Sieve of Eratosthenes

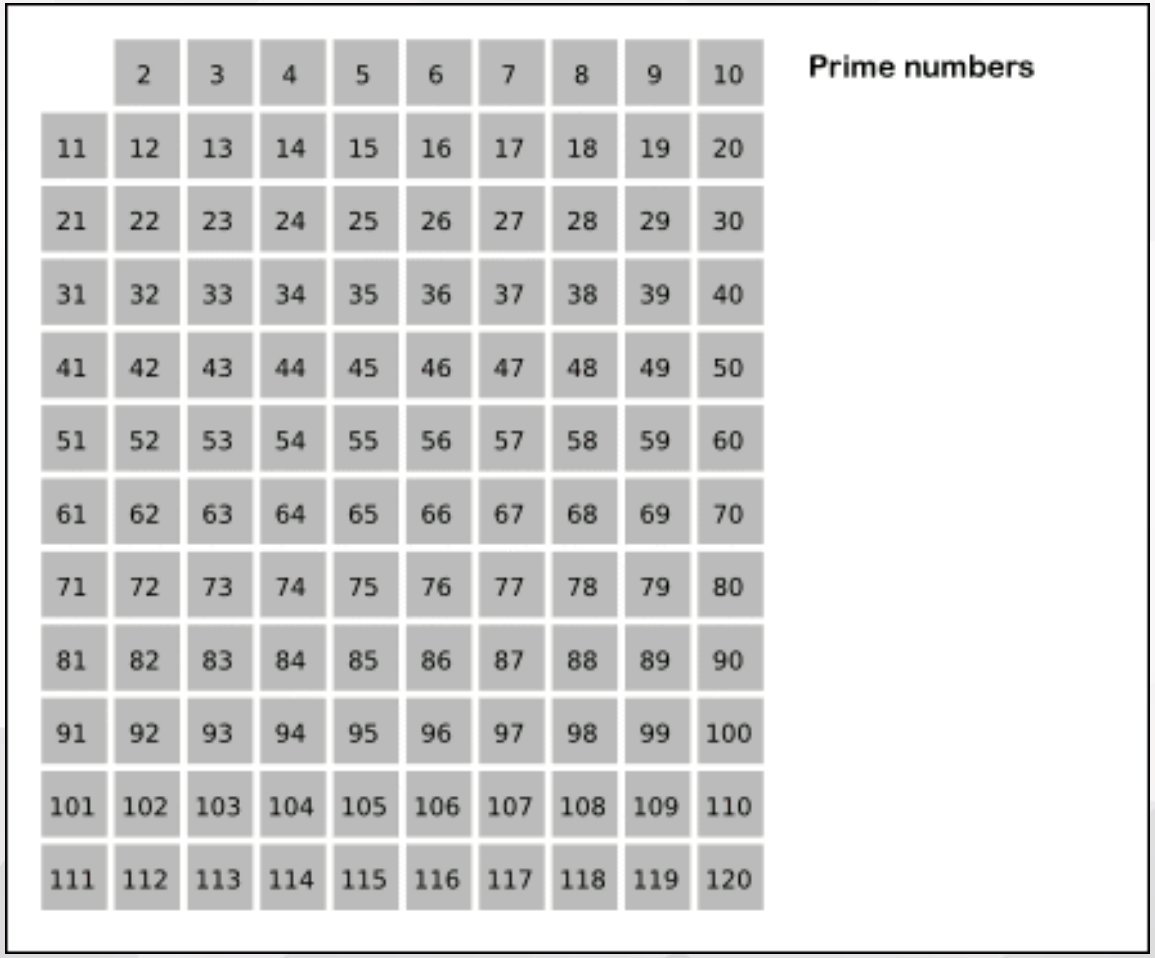
- 從 2 開始，刪掉 2 的倍數
  找下一個未被刪掉的數，找到 3 ，刪掉 3 的倍數
  找下一個未被刪掉的數，找到 5 ，刪掉 5 的倍數
  …

Sieve of Eratosthenes

- 從 2 開始，刪掉 2 的倍數
  找下一個未被刪掉的數，找到 3 ，刪掉 3 的倍數
  找下一個未被刪掉的數，找到 5 ，刪掉 5 的倍數
  …

- 最後就能刪掉所有合數，找到所有質數。

Sieve of Eratosthenes



	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Sieve of Eratosthenes

```
vector<bool> is_p(maxn, true);
is_p[1] = false;

for (int n = 2; n < sqrt(maxn); n++) {
  if (!is_p[n]) continue;
  for (int m = n; m < maxn; m+=n)
    is_p[m] = false;
}
```

Sieve of Eratosthenes

- 欲刪掉質數 n 的倍數之時
  早已刪掉其 2 倍到 n-1 倍了

- 所以可以直接從 n 倍開始。

Sieve of Eratosthenes

```
vector<bool> is_p(maxn, true);
is_p[1] = false;

for (int n = 2; n < sqrt(maxn); n++) {
  if (!is_p[n]) continue;
  for (int m = n*n; m < maxn; m+=n)
    is_p[m] = false;
}
```

Linear Sieve Algorithm

- 一邊製作質數表，一邊刪掉每個數的質數倍
- 如此每個合數就只會讀取一次

Linear Sieve Algorithm

```
vector<bool> is_p(maxn, true);
is_p[1] = false;

for (int n = 2; n < maxn; n++) {
  if (is_p[n]) prime.push_back(n);

  for (int p: prime) {
    if (p*n >= maxn) break; //超出篩檢範圍

    is_p[p*n] = false;
    if (n%p == 0) break;
  }
}
```

Calculation

Calculation

對於**大數字**的運算，普通的做法不夠快，

因此接下來將介紹快速的**乘法**及**冪**運算

Calculation

- Gauss's complex multiplication algorithm
- Karatsuba algorithm
- Fast exponentiation

複數乘法

對於 a + bi, c + di

複數乘法

對於 a + bi, c + di

相乘得 (ac – bd) + (bc + ad)i

複數乘法

對於 a + bi, c + di

相乘得 (ac – bd) + (bc + ad)i

一般需要計算 ac, bd, bc, ad 共四次乘法
才能計算出兩數相乘

複數乘法

$(ac - bd) + (bc + ad)i$

對於 $(ac - bd) = ac + bc - bc - bd$

複數乘法

$(ac - bd) + (bc + ad)i$

對於 $(ac - bd) = \underline{ac + bc} - \underline{bc - bd}$

令

$$k_1 = ac + bc = c(a+b)$$
$$k_2 = bc + bd = b(c+d)$$

複數乘法

$(ac - bd) + (bc + ad)i$

對於 $(bc + ad) = ac + bc + ad - ac$

複數乘法

$(ac - bd) + (bc + ad)i$

對於 $(bc + ad) = \underline{ac + bc} + \underline{ad - ac}$

令

$$k_1 = ac + bc = c(a+b)$$
$$k_3 = ad - ac = a(d-c)$$

複數乘法

$(ac - bd) + (bc + ad)i$

$= (k_1 - k_2) + (k_1 + k_3)i$

$$k_1 = ac + bc = c(a+b)$$
$$k_2 = bc + bd = b(c+d)$$
$$k_3 = ad - ac = a(d-c)$$

複數乘法

$(ac - bd) + (bc + ad)i$

$= (k_1 - k_2) + (k_1 + k_3)i$

總共只需要三次乘法

似乎變快了一點

Calculation

- Gauss's complex multiplication algorithm
- Karatsuba algorithm
- Fast exponentiation

Karatsuba algorithm

對於剛剛的複數乘法

似乎對於整數 x, y 相乘有些啟示

Karatsuba algorithm

對於剛剛的複數乘法

似乎對於整數 x, y 相乘有些啟示

也就是令

$$x = am + b$$
$$y = cm + d$$

Karatsuba algorithm

$$x = am + b$$

$$y = cm + d$$

$$xy = acm^2 + (ad + bc)m + bd$$

Karatsuba algorithm

$$xy = acm^2 + \underline{(ad + bc)}m + bd$$

其中

$$(ad + bc) = ad + ac + bc + bd - bd - ac$$
$$= (a + b)(c + d) - bd - ac$$

Karatsuba algorithm

$$xy = acm^2 + \underline{(ad + bc)}m + bd$$

$$(ad + bc) = ad + ac + bc + bd - bd - ac$$
$$= (a + b)(c + d) - bd - ac$$

$z_1 = ac$

$z_2 = bd$

$z_3 = (a + b)(c + d)$

Karatsuba algorithm

$$xy = acm^2 + \underline{(ad + bc)}m + bd$$

也就是說

$$xy = z_1m^2 + (z_3-z_1-z_2)m + z_2$$

$z_1 = ac$

$z_2 = bd$

$z_3 = (a + b)(c + d)$

Karatsuba algorithm

$$xy = acm^2 + \underline{(ad + bc)}m + bd$$

也就是說

$$xy = z_1m^2 + (z_3-z_1-z_2)m + z_2$$

$z_1 = ac$

$z_2 = bd$

$z_3 = (a + b)(c + d)$

Karatsuba algorithm

$$xy = z_1 m^2 + (z_3 - z_1 - z_2)m + z_2$$

假設數字長度為 n

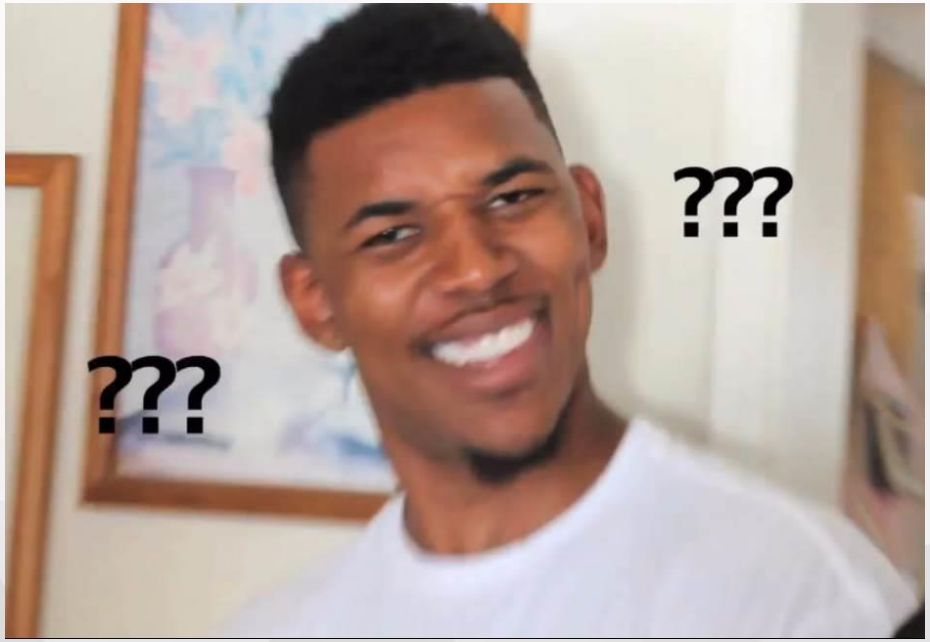
總共只需要三次乘法

相較於直式乘法複雜度 $O(n^2)$

這個算法有複雜度 $3 \cdot O(n^2)$

Karatsuba algorithm

到底在幹三小?



分治法

對於上述演算法，
凡是遇到乘法運算，都使用同樣的演算法

感恩分治 讚嘆分治

分治法

對於上述演算法，
凡是遇到乘法運算，都使用同樣的演算法

並且對於數字的分割，總是分成均等的**兩半**

例如 6789 分成 67 和 89
$6789 = 67 \cdot 100 + 89$

分治法

```
int k(int x, int y) {
  if(x < 10 || y < 10) return x*y;

  int len = min(log10(x), log10(y));
  int m = pow(10, len/2 + 1);

  auto [a, b] = div(x, m); // since c++17
  auto [c, d] = div(y, m);

  int z1 = k(a, c), z2 = k(b, d), z3 = k(a+b, c+d);
  return z1*m*m + (z3-z1-z2)*m + z2;
}
```

分治法

時間成本 T(n)

$T(n) = 3 \cdot T(n/2) + c_n$

$T(1) = 1 + c_1$

複雜度為 $O(3^{\lg n}) = O(n^{\lg 3})$

Calculation

- Gauss's complex multiplication algorithm
- Karatsuba algorithm
- Fast exponentiation

Exponentiating by Squaring

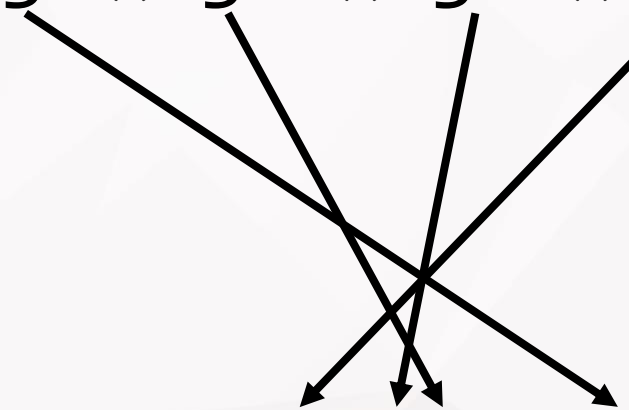
快速冪以及矩陣快速冪

如何計算 $3^{987654321} \% 1000007$

- O(n)：反正就把 3 一直乘
- O(lg n)：快速冪

快速冪

- 觀察 $3^n \times 3^n = 3^{2n}$
- 可以分解$3^{987654321} = 3^1 \times 3^{16} \times 3^{32} \times 3^{128} \times \cdots\cdots$



987654321

$= 111010110111100110100010110001_{(2)}$// 抱歉很亂

= 1 + 16 + 32 + 128 + …

快速冪

- 只要 n 是 2 的冪次 3^n 就能很快求出來
- $3^n = 3^{n/2} \times 3^{n/2}$

快速冪

```
int x = 1, a = 3;
while (n) {
  if (n&1) x *= a; // 二進位尾數是 1
  x %= 1000007;
  a *= a;
  a %= 1000007;
  n >>= 1;
} // x 就是答案
```

矩陣快速冪

- 矩陣也有類似性質
- 假設現在有個方陣 A，$A^n \times A^n = A^{2n}$
- 對於費氏數列：

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} f_{n+1} \\ f_n \end{bmatrix}$$

練習 Zero Judge b525

- b525: 先別管這個了，你聽過turtlebee嗎？

Questions?

Floating-Point Precision

浮點數誤差以及競程處理技巧

形成原因：IEEE 754 的浮點數的儲存

- 用 0 跟 1 表示浮點數
- 表示方式：$1.1101010_{(2)} \times 2^{4}{}_{(10)}$
- 優點：在精度內可以表達好，通常精度夠用
- 缺點：超出精度就無法表示
- 直接 WA

舉例

- $0.69_{(\text{double})} \times 10 \neq 6.9_{(\text{double})}$
- 結果可能是 6.89999……
- 在 print 的時候不容易出錯，但在比較大小或判斷相等時常會出問題

解決方法

- 假設今天有個閾值是 `6.9`，`6.899999` 這樣的表示會有問題
- 解決方案一：四捨五入
- 解決方案二：自己用 `int` 做運算
- 解決方案三：乘上 `1.000…001`(數量根據精度調整) -> 較推薦

解決方法(比較大小)

```
if (0.69 * 10 * 1.000…1 >= 6.9) {
    do something
}
```

這裡比較 `0.69*10` 是否大於等於 `6.9`
所以直接將左邊乘大一些，給他一點誤差空間

解決方法(比較相等)

```
if (abs((0.69*10) - 6.9) <= 6.9*1e-14) {
    do something
}
```

這裡比較 `0.69*10` 是否等於 `6.9`
`1e-14` 是一個很小的數，需大於資料型態的精度
代表 `0.69*10` 跟 `6.9` 的差值是否小於自身的某個比例

AC Get