计算科学中的数学问题
-数论篇

By supermaker

概述

数论是基础数学的分支，主要研究整数的性质，大致可分为初等数论和高等数论两类，其中高等数论中还分为解析、代数、几何、计算、组合和超越数论等几类

我们在程序设计中遇到的问题基本上都属于初等数论范围，比如：

如何求解一个二元一次不定方程的通解？

如何求解$666666^{666666^{666666}}\ \%\ 12345$的值？

目录

1.整除理论与素性分析

2.不定方程与同余理论

3.同余式定理与积性函数

整除理论与素性分析

初等数论的基础知识

目录

整除理论

1.1整除理论基础知识

1.2最大公约数和最小公倍数

1.3辗转相除法(欧几里得算法)

1.4扩展欧几里得算法

素性分析

1.5素数基础知识

1.6算数基本定理

1.7素性分析

1.8整数素因子分解

1.1 整除理论的基础知识

1.1整除理论的基础知识

整除、因子和倍数： 设a、b两个整数，且满足$b \neq 0$。如果存在c，满足$a = b \cdot c$，则称b**整除**a，或a被b整除，记作$b \mid a$，此时也称b为a的**因子(约数)**，a为b的**倍数**

1.1整除理论的基础知识

整除、因子和倍数： 设a、b两个整数，且满足$b \neq 0$。如果存在c，满足$a = b \cdot c$，则称b**整除**a，或a被b整除，记作$b \mid a$，此时也称b为a的**因子(约数)**，a为b的**倍数**

整除的性质：

(i)如果$2 \mid n$, 则称n为**偶数**，否则称n为**奇数**

(ii)(传递性)若$a \mid b$，且$b \mid c$，则$a \mid c$

(iii)(线性)若$a \mid b$，且$a \mid c$，则$a \mid (x \cdot b + y \cdot c)$，其中x、y是整数

(iiii)(带余除法)(证明略)设a、b两个整数，满足$b \neq 0$，则存在唯一的整数k、q，使得

$$a = k \cdot b + q,\ 0 \leq q < |b|$$

1.2最大公约数与最小公倍数

1.2最大公约数和最小公倍数

公因子(约数)：设a、b为两个整数，如果有d | a，且d | b，则称d为a和b的**公因子(约数)**

最大公因子(约数)：设a、b为两个不全为0的数，则a和b的**公因子(约数)**中最大的叫做a和b的**最大公因子(约数)**，记作**gcd(a, b)**，常简记作**(a, b)**

1.2最大公约数和最小公倍数

公因子(约数)：设a、b为两个整数，如果有d | a，且d | b，则称d为a和b的**公因子(约数)**

最大公因子(约数)：设a、b为两个不全为0的数，则a和b的**公因子(约数)**中最大的叫做a和b的**最大公因子(约数)**，记作**gcd(a, b)**，常简记作**(a, b)**

公倍数：设a、b为两个整数，如果有a | d，且b | d，则称d为a和b的**公倍数**

最大公倍数：设a、b为两个不全为0的数，则a和b的**公倍数**中最小的叫做a和b的**最小公倍数**，记作**lcm(a, b)**，常简记作**[a, b]**

1.2最大公约数和最小公倍数

公因子(约数)：设a、b为两个整数，如果有d | a，且d | b，则称d为a和b的**公因子(约数)**

最大公因子(约数)：设a、b为两个不全为0的数，则a和b的**公因子(约数)**中最大的叫做a和b的**最大公因子(约数)**，记作**gcd(a, b)**，常简记作**(a, b)**

公倍数：设a、b为两个整数，如果有a | d，且b | d，则称d为a和b的**公倍数**

最大公倍数：设a、b为两个不全为0的数，则a和b的**公倍数**中最小的叫做a和b的**最小公倍数**，记作**lcm(a, b)**，常简记作**[a, b]**

重要性质：(i)lcm(a, b) · gcd (a, b) = a · b

(ii)设a、b为两个整数，若a | b，则gcd (a, b) = a

1.2最大公约数和最小公倍数

互素：设a、b为两个整数，若gcd (a, b) = 1，则称a与b互素，也就说a和b没有公因子

1.2最大公约数和最小公倍数

互素： 设a、b为两个整数，若gcd (a, b) = 1，则称a与b**互素**，也就是说a和b没有**公因子**

推论： 设a、b为两个整数，若gcd (a, b) = k, k > 1，则gcd $(\frac{a}{k}, \frac{b}{k}) = 1$，即此时$\frac{a}{k}$和$\frac{b}{k}$**互素**

1.2最大公约数和最小公倍数

互素： 设a、b为两个整数，若gcd (a, b) = 1，则称a与b**互素**，也就说a和b没有**公因子**

推论： 设a、b为两个整数，若gcd (a, b) = k, k > 1，则gcd $(\frac{a}{k}, \frac{b}{k}) = 1$，即此时$\frac{a}{k}$和$\frac{b}{k}$**互素**

互素的直接应用： 规约分数

1.2最大公约数和最小公倍数

互素： 设a、b为两个整数，若gcd (a, b) = 1，则称a与b互素，也就说a和b没有公因子

推论： 设a、b为两个整数，若gcd (a, b) = k, k > 1，则gcd $(\frac{a}{k}, \frac{b}{k}) = 1$，即此时$\frac{a}{k}$和$\frac{b}{k}$互素

互素的直接应用： 规约分数

最简分数定义为分子和分母没有公因子的分数，即分子和分母要互素。规约分数直接按照推论的方式将分子分母同除以它们的**最大公约数**就可以了

1.2求解最大公约数的算法

1.辗转相除法(欧几里得算法)

2.*更相减损术

1.2求解最大公约数的算法

1.辗转相除法(欧几里得算法)

2.*更相减损术

辗转相除法是求解最大公约数的最常用方法，而更相减损术也是一种常用的方法，只是辗转相除法以除法运算为主，计算次数较少。而更相减损术则以减法运算为主，计算次数较多，不太适用于求解两个相差较大数的最大公约数

1.3辗转相除法

1.3辗转相除法

定理：设a、b和q都是不全为0的整数，则满足$a = k \cdot b + q$，则有

$$\gcd(a, b) = \gcd(b, q)$$

1.3辗转相除法

定理：设a、b和q都是不全为0的整数，则满足a = k · b + q，则有

$$\gcd(a, b) = \gcd(b, q)$$

证明：

只要证明a和b的公因子与b和q的公因子相同即可。设d是a与b的公因子，即d | a，且d | b。由题目可得q = a - k · b，可得d | q(由整除的线性性质得到)。则有d | q，且d | b，此时可得d是b和q的公因子，反之设d是b和q的公因子，则可以按照上面的方法反推得到d是a和b的公因子，综上，原命题得证

1.3辗转相除法

定理：设a、b和q都是不全为0的整数，则满足$a = k \cdot b + q$，则有

$$gcd(a, b) = gcd(b, q)$$

证明：

只要证明a和b的公因子与b和q的公因子相同即可。设d是a与b的公因子，即$d \mid a$，且$d \mid b$。由题目可得$q = a - k \cdot b$，可得$d \mid q$(由整除的线性性质得到)。则有$d \mid q$，且$d \mid b$，此时可得d是b和q的公因子，反之设d是b和q的公因子，则可以按照上面的方法反推得到d是a和b的公因子，综上，原命题得证

通过上述定理，使得原问题等价转化成规模更小的问题，进而加速求解过程。辗转相除法常使用递归实现，只要设定好一个递归边界即可

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

解析：设a、b为两个整数，则有

$a = k_0 \cdot b + q_0$

$b = k_1 \cdot q_0 + q_1$

$q_0 = k_2 \cdot q_1 + q_2$

$q_1 = k_3 \cdot q_2 + q_3$

…

$q_{n-1} = k_{n+1} \cdot q_n + q_{n+1}$

$q_n = k_{n+2} \cdot q_{n+1} + q_{n+2}$

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

解析：设a、b为两个整数，则有

$a = k_0 \cdot b + q_0$

$b = k_1 \cdot q_0 + q_1$

$q_0 = k_2 \cdot q_1 + q_2$

$q_1 = k_3 \cdot q_2 + q_3$

…

$q_{n-1} = k_{n+1} \cdot q_n + q_{n+1}$

$q_n = k_{n+2} \cdot q_{n+1} + q_{n+2}$

此时由于满足$0 \le q_{i+1} < |q_i|$，则可得$0 \le |q_n| < |q_{n-1}| < \ldots < |q_1| < |q_0|$
即意味着余数越来越小，直到余数等于0为止。这时最大公约数就是除数(由前面的最大公约数的性质ii可得)

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

比如：求gcd(125, 17)
则有：
$125 = 7 \cdot 17 + 6$

解析：设a、b为两个整数，则有
$a = k_0 \cdot b + q_0$
$b = k_1 \cdot q_0 + q_1$
$q_0 = k_2 \cdot q_1 + q_2$
$q_1 = k_3 \cdot q_2 + q_3$
…
$q_{n-1} = k_{n+1} \cdot q_n + q_{n+1}$
$q_n = k_{n+2} \cdot q_{n+1} + q_{n+2}$

此时由于满足$0 \le q_{i+1} < |q_i|$，则可得$0 \le |q_n| < |q_{n-1}| < \ldots < |q_1| < |q_0|$
即意味着余数越来越小，直到余数等于0为止。这时最大公约数就是除数
(由前面的最大公约数的性质ii可得)

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

比如：求gcd(125, 17)
则有：
$125 = 7 \cdot 17 + 6$
$17 = 2 \cdot 6 + 5$

解析：设a、b为两个整数，则有
$a = k_0 \cdot b + q_0$
$b = k_1 \cdot q_0 + q_1$
$q_0 = k_2 \cdot q_1 + q_2$
$q_1 = k_3 \cdot q_2 + q_3$
…
$q_{n-1} = k_{n+1} \cdot q_n + q_{n+1}$
$q_n = k_{n+2} \cdot q_{n+1} + q_{n+2}$

此时由于满足$0 \le q_{i+1} < |q_i|$，则可得$0 \le |q_n| < |q_{n-1}| < \dots < |q_1| < |q_0|$
即意味着余数越来越小，直到余数等于0为止。这时最大公约数就是除数
(由前面的最大公约数的性质ii可得)

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

比如：求gcd(125, 17)
则有：
$125 = 7 \cdot 17 + 6$
$17 = 2 \cdot 6 + 5$

解析：设a、b为两个整数，则有
$a = k_0 \cdot b + q_0$
$b = k_1 \cdot q_0 + q_1$
$q_0 = k_2 \cdot q_1 + q_2$
$q_1 = k_3 \cdot q_2 + q_3$
…
$q_{n-1} = k_{n+1} \cdot q_n + q_{n+1}$
$q_n = k_{n+2} \cdot q_{n+1} + q_{n+2}$

此时由于满足$0 \le q_{i+1} < |q_i|$，则可得$0 \le |q_n| < |q_{n-1}| < \ldots < |q_1| < |q_0|$
即意味着余数越来越小，直到余数等于0为止。这时最大公约数就是除数
(由前面的最大公约数的性质ii可得)

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

比如：求gcd(125, 17)
则有：
$125 = 7 \cdot 17 + 6$
$17 = 2 \cdot 6 + 5$
$6 = 1 \cdot 5 + 1$

解析：设a、b为两个整数，则有
$a = k_0 \cdot b + q_0$
$b = k_1 \cdot q_0 + q_1$
$q_0 = k_2 \cdot q_1 + q_2$
$q_1 = k_3 \cdot q_2 + q_3$
…
$q_{n-1} = k_{n+1} \cdot q_n + q_{n+1}$
$q_n = k_{n+2} \cdot q_{n+1} + q_{n+2}$

此时由于满足$0 \le q_{i+1} < |q_i|$，则可得$0 \le |q_n| < |q_{n-1}| < \dots < |q_1| < |q_0|$
即意味着余数越来越小，直到余数等于0为止。这时最大公约数就是除数(由前面的最大公约数的性质ii可得)

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

比如：求gcd(125, 17)
则有：
125 = 7 · 17 + 6
17 = 2 · 6 + 5
6 = 1 · 5 + 1

解析：设a、b为两个整数，则有
a = k_0 · b + q_0
b = $k_1 \cdot q_0 + q_1$
$q_0 = k_2 \cdot q_1 + q_2$
$q_1 = k_3 \cdot q_2 + q_3$
…
$q_{n-1} = k_{n+1} \cdot q_n + q_{n+1}$
$q_n = k_{n+2} \cdot q_{n+1} + q_{n+2}$

此时由于满足$0 \le q_{i+1} < |q_i|$，则可得$0 \le |q_n| < |q_{n-1}| < \ldots < |q_1| < |q_0|$
即意味着余数越来越小，直到余数等于0为止。这时最大公约数就是除数
(由前面的最大公约数的性质ii可得)

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

比如：求gcd(125, 17)
则有：
$125 = 7 \cdot 17 + 6$
$17 = 2 \cdot 6 + 5$
$6 = 1 \cdot 5 + 1$
$5 = 5 \cdot 1 + 0$

解析：设a、b为两个整数，则有
$a = k_0 \cdot b + q_0$
$b = k_1 \cdot q_0 + q_1$
$q_0 = k_2 \cdot q_1 + q_2$
$q_1 = k_3 \cdot q_2 + q_3$
…
$q_{n-1} = k_{n+1} \cdot q_n + q_{n+1}$
$q_n = k_{n+2} \cdot q_{n+1} + q_{n+2}$

此时由于满足$0 \le q_{i+1} < |q_i|$，则可得$0 \le |q_n| < |q_{n-1}| < \ldots < |q_1| < |q_0|$
即意味着余数越来越小，直到余数等于0为止。这时最大公约数就是除数
(由前面的最大公约数的性质ii可得)

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

解析：设a、b为两个整数，则有

$a = k_0 \cdot b + q_0$
$b = k_1 \cdot q_0 + q_1$
$q_0 = k_2 \cdot q_1 + q_2$
$q_1 = k_3 \cdot q_2 + q_3$
…
$q_{n-1} = k_{n+1} \cdot q_n + q_{n+1}$
$q_n = k_{n+2} \cdot q_{n+1} + q_{n+2}$

此时由于满足$0 \le q_{i+1} < |q_i|$，则可得$0 \le |q_n| < |q_{n-1}| < \dots < |q_1| < |q_0|$
即意味着余数越来越小，直到余数等于0为止。这时最大公约数就是除数
(由前面的最大公约数的性质ii可得)

比如：求gcd(125, 17)
则有：
$125 = 7 \cdot 17 + 6$
$17 = 2 \cdot 6 + 5$
$6 = 1 \cdot 5 + 1$
$5 = 5 \cdot 1 + 0$
此时最大公约数就是
最后一个等式中除数1

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

比如： 求gcd(125, 17)
则有：
125 = 7 · 17 + 6
17 = 2 · 6 + 5
6 = 1 · 5 + 1
5 = 5 · 1 + 0
此时最大公约数就是
最后一个等式中除数1

解析： 设a、b为两个整数，则有

$a = k_0 \cdot b + q_0$
$b = k_1 \cdot q_0 + q_1$
$q_0 = k_2 \cdot q_1 + q_2$
$q_1 = k_3 \cdot q_2 + q_3$
…
$q_{n-1} = k_{n+1} \cdot q_n + q_{n+1}$
$q_n = k_{n+2} \cdot q_{n+1} + q_{n+2}$

思考： 该算法需要进行带余除法， 显然需要保证输入参数 $a > b$， 那么如果输入参数 $a < b$ 会发生什么？

此时由于满足 $0 \le q_{i+1} < |q_i|$， 则可得 $0 \le |q_n| < |q_{n-1}| < \ldots < |q_1| < |q_0|$
即意味着余数越来越小， 直到余数等于0为止。 这时最大公约数就是除数
(由前面的最大公约数的性质ii可得)

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

比如：求gcd(125, 17)
则有：
125 = 7 · 17 + 6
17 = 2 · 6 + 5
6 = 1 · 5 + 1
5 = 5 · 1 + 0
此时最大公约数就是
最后一个等式中除数1

解析：设a、b为两个整数，则有

$a = k_0 \cdot b + q_0$
$b = k_1 \cdot q_0 + q_1$
$q_0 = k_2 \cdot q_1 + q_2$
$q_1 = k_3 \cdot q_2 + q_3$
…
$q_{n-1} = k_{n+1} \cdot q_n + q_{n+1}$
$q_n = k_{n+2} \cdot q_{n+1} + q_{n+2}$

思考：该算法需要进行带余除法，显然需要保证输入参数 $a > b$，那么如果输入参数 $a < b$会发生什么？

解析：如果 $a < b$，则该算法的第一步其实将两个数进行交换，这是因为 $a < b$，则a除b的商为0，余数为a。在下次计算时，当前的除数b会成为被除数，当前的余数a会成为除数，进而可以看做是交换了a和b这两个数

此时由于满足$0 \leq q_{i+1} < |q_i|$，则可得$0 \leq |q_n| < |q_{n-1}| < \ldots < |q_1| < |q_0|$
即意味着余数越来越小，直到余数等于0为止。这时最大公约数就是除数(由前面的最大公约数的性质ii可得)

1.3辗转相除法

```
long long gcd(long long a, long long b)
{
    return (b == 0)? a: gcd(b, a % b);
}
```

比如：求gcd(125, 17)
则有：
$125 = 7 \cdot 17 + 6$
$17 = 2 \cdot 6 + 5$
$6 = 1 \cdot 5 + 1$
$5 = 5 \cdot 1 + 0$
此时最大公约数就是最后一个等式中除数1

解析：设a、b为两个整数，则有
$a = k_0 \cdot b + q_0$
$b = k_1 \cdot q_0 + q_1$
$q_0 = k_2 \cdot q_1 + q_2$
$q_1 = k_3 \cdot q_2 + q_3$
…
$q_{n-1} = k_{n+1} \cdot q_n + q_{n+1}$
$q_n = k_{n+2} \cdot q_{n+1} + q_{n+2}$

注意：由于整除是两个自然数之间(不含0)的关系，所以一个正整数和负整数之间不存在公约数，两个负整数之间也不存在公约数！！！

此时由于满足$0 \leq q_{i+1} < |q_i|$，则可得$0 \leq |q_n| < |q_{n-1}| < \dots < |q_1| < |q_0|$
即意味着余数越来越小，直到余数等于0为止。这时最大公约数就是除数(由前面的最大公约数的性质ii可得)

1.4扩展欧几里得算法

1.4扩展欧几里得算法

贝祖定理(Bezouts identity)：设a、b为不全为零的数，则存在唯一的整数x、y，使得

$$a \cdot x + b \cdot y = \gcd(a, b)$$

1.4扩展欧几里得算法

贝祖定理(Bezouts identity)：设a、b为不全为零的数，则存在唯一的整数x、y，使得

$$a \cdot x + b \cdot y = \gcd(a, b)$$

推论：a与b互素的充要条件是存在唯一的整数x、y，使得$a \cdot x + b \cdot y = 1$

1.4扩展欧几里得算法

贝祖定理(Bezouts identity)：设a、b为不全为零的数，则存在唯一的整数x、y，使得

$$a \cdot x + b \cdot y = \gcd(a, b)$$

推论：a与b互素的充要条件是存在唯一的整数x、y，使得$a \cdot x + b \cdot y = 1$

上述定理表示a与b的最大公约数可以表示成a与b的线性组合

1.4扩展欧几里得算法

贝祖定理(Bezouts identity)：设a、b为不全为零的数，则存在唯一的整数x、y，使得

$$a \cdot x + b \cdot y = \gcd(a, b)$$

推论：a与b互素的充要条件是存在唯一的整数x、y，使得$a \cdot x + b \cdot y = 1$

上述定理表示a与b的最大公约数可以表示成a与b的线性组合

思考：现在给出两个符合的数a、b，如何计算出上面的x、y?

1.4扩展欧几里得算法

贝祖定理(Bezouts identity)：设a、b为不全为零的数，则存在唯一的整数x、y，使得

$$a \cdot x + b \cdot y = \gcd(a, b)$$

推论：a与b互素的充要条件是存在唯一的整数x、y，使得$a \cdot x + b \cdot y = 1$

上述定理表示a与b的最大公约数可以表示成a与b的线性组合

思考：现在给出两个符合的数a、b，如何计算出上面的x、y?

解析：只需在欧几里得算法上进行一些改动即可，改动后的结果就是扩展欧几里得算法。现在先看下面的一个例子

1.4扩展欧几里得算法

比如：现有gcd (125, 17) = 1且满足：

$$125 = 7 \cdot 17 + 6 \cdots\cdots(1)$$

$$17 = 2 \cdot 6 + 5 \cdots\cdots\cdots(2)$$

$$6 = 1 \cdot 5 + 1 \cdots\cdots\cdots(3)$$

$$5 = 5 \cdot 1 + 0 \cdots\cdots\cdots(4)$$

1.4扩展欧几里得算法

比如：现有gcd (125, 17) = 1且满足：

$$125 = 7 \cdot 17 + 6 \cdots\cdots(1)$$

$$17 = 2 \cdot 6 + 5 \cdots\cdots\cdots(2)$$

$$6 = 1 \cdot 5 + 1 \cdots\cdots\cdots(3)$$

$$5 = 5 \cdot 1 + 0 \cdots\cdots\cdots(4)$$

(i)先将(3)式变形变成$1 = 6 - 1 \cdot 5$

1.4扩展欧几里得算法

比如：现有gcd (125, 17) = 1且满足：

$$125 = 7 \cdot 17 + 6 \cdots\cdots(1)$$

$$17 = 2 \cdot 6 + 5 \cdots\cdots(2)$$

$$6 = 1 \cdot 5 + 1 \cdots\cdots(3)$$

$$5 = 5 \cdot 1 + 0 \cdots\cdots(4)$$

(i)先将(3)式变形变成$1 = 6 - 1 \cdot 5$

(ii)将(2)式变形成$5 = 17 - 2 \cdot 6$并代入到(3)式中将5替换掉，化简得到$1 = 3 \cdot 6 - 1 \cdot 17$，并记为(5)式

1.4扩展欧几里得算法

比如：现有gcd (125, 17) = 1且满足：

$$125 = 7 \cdot 17 + 6 \cdots\cdots(1)$$

$$17 = 2 \cdot 6 + 5 \cdots\cdots\cdots(2)$$

$$6 = 1 \cdot 5 + 1 \cdots\cdots\cdots(3)$$

$$5 = 5 \cdot 1 + 0 \cdots\cdots\cdots(4)$$

(i)先将(3)式变形变成$1 = 6 - 1 \cdot 5$

(ii)将(2)式变形成$5 = 17 - 2 \cdot 6$并代入到(3)式中将5替换掉，化简得到$1 = 3 \cdot 6 - 1 \cdot 17$，并记为(5)式

(iii)将(1)式变形成$6 = 125 - 7 \cdot 17$并代入到(5)式中将6替换掉，化简得到$1 = 3 \cdot 125 - 22 \cdot 17$

1.4扩展欧几里得算法

比如：现有gcd (125, 17) = 1且满足：

$$125 = 7 \cdot 17 + 6 \cdots\cdots(1)$$

$$17 = 2 \cdot 6 + 5 \cdots\cdots\cdots(2)$$

$$6 = 1 \cdot 5 + 1 \cdots\cdots\cdots(3)$$

$$5 = 5 \cdot 1 + 0 \cdots\cdots\cdots(4)$$

(i)先将(3)式变形变成$1 = 6 - 1 \cdot 5$

(ii)将(2)式变形成$5 = 17 - 2 \cdot 6$并代入到(3)式中将5替换掉，化简得到$1 = 3 \cdot 6 - 1 \cdot 17$，并记为(5)式

(iii)将(1)式变形成$6 = 125 - 7 \cdot 17$并代入到(5)式中将6替换掉，化简得到$1 = 3 \cdot 125 - 22 \cdot 17$

此时得到答案：$3 \cdot 125 - 22 \cdot 17 = \gcd(125, 17) = 1$，此时$x = 3$，$y = 22$

1.4扩展欧几里得算法

观察上面的求解过程不难发现，我们不断地将后面的式子代入到前面的式子中并将一些中间量替换掉，直到式子全部由a和b所表示

1.4扩展欧几里得算法

观察上面的求解过程不难发现，我们不断地将后面的式子代入到前面的式子中并将一些中间量替换掉，直到式子全部由a和b所表示

还记得欧几里得算法的停止状态是：a = gcd，b = 0吗？那么是否给我们求解提供了一种思路？

1.4扩展欧几里得算法

观察上面的求解过程不难发现，我们不断地将后面的式子代入到前面的式子中并将一些中间量替换掉，直到式子全部由a和b所表示

还记得欧几里得算法的停止状态是：$a = gcd$，$b = 0$吗？那么是否给我们求解提供了一种思路？

解析：因为此时只要满足$x = 1$，$y = 0$，则会得到：

$$a \cdot x + b \cdot y = gcd$$

1.4扩展欧几里得算法

观察上面的求解过程不难发现，我们不断地将后面的式子代入到前面的式子中并将一些中间量替换掉，直到式子全部由a和b所表示

还记得欧几里得算法的停止状态是：$a = gcd$，$b = 0$吗？那么是否给我们求解提供了一种思路？

解析：因为此时只要满足$x = 1$，$y = 0$，则会得到：

$$a \cdot x + b \cdot y = gcd$$

这是最终的状态，能否仿照前面的求解过程反推出最初的状态？

1.4扩展欧几里得算法

假设我们当前要处理的是求出a、b的最大公约数，并求出x、y使得a · x + b · y = gcd，而我们已经求出下一个状态：b和a % b的最大公约数，并且求出一组x_1、y_1使得b · x_1 + a % b · y_1 = gcd，那么这两个状态之间是否存在一种关系？

1.4扩展欧几里得算法

假设我们当前要处理的是求出a、b的最大公约数，并求出x、y使得a · x + b · y = gcd，而我们已经求出下一个状态：b和a % b的最大公约数，并且求出一组x_1、y_1使得b · x_1 + a % b · y_1 = gcd，那么这两个状态之间是否存在一种关系？

我们可得：

$$\begin{aligned} gcd &= b \cdot x_1 + a \% b \cdot y_1 \\ &= b \cdot x_1 + (a - b \cdot \lfloor a / b \rfloor) \cdot y_1 \\ &= b \cdot x_1 + a \cdot y_1 - b \cdot \lfloor a / b \rfloor \cdot y_1 \\ &= a \cdot y_1 + b \cdot (x_1 - b \cdot \lfloor a / b \rfloor \cdot y_1) \quad (1) \end{aligned}$$

1.4扩展欧几里得算法

假设我们当前要处理的是求出a、b的最大公约数，并求出x、y使得a · x + b · y = gcd，而我们已经求出下一个状态：b和a % b的最大公约数，并且求出一组x_1、y_1使得b · x_1 + a % b · y_1 = gcd，那么这两个状态之间是否存在一种关系？

我们可得：

$$
\begin{aligned}
\mathrm{gcd} &= \mathrm{b} \cdot x_1 + \mathrm{a} \,\%\, \mathrm{b} \cdot y_1 \\
&= \mathrm{b} \cdot x_1 + (\mathrm{a} - \mathrm{b} \cdot \lfloor \mathrm{a} / \mathrm{b} \rfloor) \cdot y_1 \\
&= \mathrm{b} \cdot x_1 + \mathrm{a} \cdot y_1 - \mathrm{b} \cdot \lfloor \mathrm{a} / \mathrm{b} \rfloor \cdot y_1 \\
&= \mathrm{a} \cdot y_1 + \mathrm{b} \cdot (x_1 - \mathrm{b} \cdot \lfloor \mathrm{a} / \mathrm{b} \rfloor \cdot y_1) \quad (1)
\end{aligned}
$$

对比待求解的式子gcd = a · x + b · y和(1)式，由相同项系数值对应相等可得：

$$
\begin{cases}
x = y_1 \\
y = \mathrm{b} \cdot (x_1 - \mathrm{b} \cdot \lfloor \mathrm{a} / \mathrm{b} \rfloor \cdot y_1)
\end{cases}
$$

1.4扩展欧几里得算法

假设我们当前要处理的是求出a、b的最大公约数，并求出x、y使得a · x + b · y = gcd，而我们已经求出下一个状态：b和a % b的最大公约数，并且求出一组x_1、y_1使得b · x_1 + a % b · y_1 = gcd，那么这两个状态之间是否存在一种关系？

我们可得：

$$\begin{aligned} gcd &= b \cdot x_1 + a \% b \cdot y_1 \\ &= b \cdot x_1 + (a - b \cdot \lfloor a / b \rfloor) \cdot y_1 \\ &= b \cdot x_1 + a \cdot y_1 - b \cdot \lfloor a / b \rfloor \cdot y_1 \\ &= a \cdot y_1 + b \cdot (x_1 - b \cdot \lfloor a / b \rfloor \cdot y_1) \quad (1) \end{aligned}$$

对比待求解的式子gcd = a · x + b · y和(1)式，由相同项系数值对应相等可得：

$$\begin{cases} x = y_1 \\ y = b \cdot (x_1 - b \cdot \lfloor a / b \rfloor \cdot y_1) \end{cases}$$

得到上面的递推式，且前面已经设定了x、y的初始值(x = 1、y = 0)，则按照递推式反向求解即可

1.4扩展欧几里得算法

```
long long gcd_ex (long long a, long long b, long long &x, long long &y)
{
    if (b == 0) { x = 1; y = 0; return a; }
    long long d = gcd_ex (b, a % b, y, x);
    y = y - a / b * x;
    return d;
}
```

1.4扩展欧几里得算法

```
long long gcd_ex (long long a, long long b, long long &x, long long &y)
{
    if (b == 0) { x = 1; y = 0; return a; }
    long long d = gcd_ex (b, a % b, y, x);
    y = y - a / b * x;
    return d;
}
```

注意：这里使用C++中的引用来交换前后两个状态之间的x、y值，即当前状态的y是前一个状态的x，当前状态的x是前一个状态的y！！！

1.4扩展欧几里得算法

```
long long gcd_ex (long long a, long long b, long long &x, long long &y)
{
    if (b == 0) { x = 1; y = 0; return a; }
    long long d = gcd_ex (b, a % b, y, x);
    y = y - a / b * x;
    return d;
}
```

注意：这里使用C++中的引用来交换前后两个状态之间的x、y值，即当前状态的y是前一个状态的x，当前状态的x是前一个状态的y！！！

可以简单地给出欧几里得算法和扩展欧几里得算法的时间复杂度是O(logn)的

1.5 素数基础知识

1.5素数基础知识

素数定义：一个数N如果除了1和N之外，没有其他的因子，则称N为素数，否则称其为合数

素数的性质：

1.素数存在无穷多个

2.合数N一定有小于等于$\sqrt{N}$的素因子(重要)

素数分布

小于正整数N的素数大致有$\pi(N)$ 个，且满足

$$\pi(N) \sim \frac{N}{\ln N}$$

1.6算数基本定理

1.6算数基本定理

算数基本定理：任意大于1的正整数N都可以唯一表示成素因子幂乘积的形式，令乘积中的素因子按照非递减序排列，则可得到标准分解式

$$N = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot {p_3}^{\alpha_3} \cdot {p_4}^{\alpha_4} \cdot \ldots\ldots \cdot {p_k}^{\alpha_k} \quad \alpha_{\mathrm{i}} \in N^* , 1 \le \mathrm{i} \le \mathrm{k}$$

其中p_i $(1 \le \mathrm{i} \le \mathrm{k})$是N的素因子，且$p_i < p_{i+1}$

1.6算数基本定理

算数基本定理： 任意大于1的正整数N都可以**唯一**表示成**素因子幂乘积**的形式，令乘积中的素因子按照**非递减序**排列，则可得到**标准分解式**

$$\mathrm{N} = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot {p_3}^{\alpha_3} \cdot {p_4}^{\alpha_4} \cdot \ldots\ldots \cdot {p_k}^{\alpha_k} \quad \alpha_{\mathrm{i}} \in N^* \text{ , } 1 \le \mathrm{i} \le \mathrm{k}$$

其中p_i $(1 \le \mathrm{i} \le \mathrm{k})$是N的**素因子**，且$p_i < p_{i+1}$

比如

24的算数分解式为：$24 = 2^3 \cdot 3^1$

60的算数分解式为：$60 = 2^2 \cdot 3^1 \cdot 5^1$

1.6算数基本定理

算数基本定理：任意大于1的正整数N都可以唯一表示成素因子幂乘积的形式，令乘积中的素因子按照非递减序排列，则可得到标准分解式

$$\mathrm{N} = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot {p_3}^{\alpha_3} \cdot {p_4}^{\alpha_4} \cdot \ldots\ldots \cdot {p_k}^{\alpha_k} \quad \alpha_{\mathrm{i}} \in N^* \text{ , } 1 \le \mathrm{i} \le \mathrm{k}$$

其中p_i $(1 \le \mathrm{i} \le \mathrm{k})$是N的素因子，且$p_i < p_{i+1}$

为什么标准分解式中的每一项都是素因子，有没有可能是一个合数因子？

1.6算数基本定理

算数基本定理：任意大于1的正整数N都可以唯一表示成素因子幂乘积的形式，令乘积中的素因子按照非递减序排列，则可得到标准分解式

$$N = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot {p_3}^{\alpha_3} \cdot {p_4}^{\alpha_4} \cdot \ldots\ldots \cdot {p_k}^{\alpha_k} \quad \alpha_i \in N^*, 1 \le i \le k$$

其中p_i $(1 \le i \le k)$是N的素因子，且$p_i < p_{i+1}$

为什么标准分解式中的每一项都是素因子，有没有可能是一个合数因子？

不可能，这是因为合数因子可以继续分解因子，而如果分解出的因子还是合数，则递归地分解，直到分解出的因子是素因子时分解过程才会结束

1.6算数基本定理

算数基本定理：任意大于1的正整数N都可以唯一表示成素因子幂乘积的形式，令乘积中的素因子按照非递减序排列，则可得到标准分解式

$$N = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot {p_3}^{\alpha_3} \cdot {p_4}^{\alpha_4} \cdot \ldots\ldots \cdot {p_k}^{\alpha_k} \quad \alpha_i \in N^* , 1 \le i \le k$$

其中p_i $(1 \le i \le k)$是N的素因子，且$p_i < p_{i+1}$

为什么标准分解式中的每一项都是素因子，有没有可能是一个合数因子？

不可能，这是因为合数因子可以继续分解因子，而如果分解出的因子还是合数，则递归地分解，直到分解出的因子是素因子时分解过程才会结束

这表明素数是表示正整数的一种单位，算数基本定理对于分析后续问题很有帮助

1.6算数基本定理

算数基本定理：任意大于1的正整数N都可以唯一表示成素因子幂乘积的形式，令乘积中的素因子按照非递减序排列，则可得到标准分解式

$$N = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot {p_3}^{\alpha_3} \cdot {p_4}^{\alpha_4} \cdot \ldots\ldots \cdot {p_k}^{\alpha_k} \quad \alpha_i \in N^* , 1 \le i \le k$$

其中p_i $(1 \le i \le k)$是N的素因子，且$p_i < p_{i+1}$

重要性质：

若$a = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot \ldots \cdot {p_k}^{\alpha_k}$，$b = {p_1}^{\beta_1} \cdot {p_2}^{\beta_2} \cdot \ldots \cdot {p_k}^{\beta_k}$，则有

$$\gcd(a, b) = {p_1}^{\min(\alpha_1,\beta_1)} \cdot {p_2}^{\min(\alpha_2,\beta_2)} \cdot \ldots \cdot {p_k}^{\min(\alpha_k,\beta_k)}$$

$$\mathrm{lcm}(a, b) = {p_1}^{\max(\alpha_1,\beta_1)} \cdot {p_2}^{\max(\alpha_2,\beta_2)} \cdot \ldots \cdot {p_k}^{\max(\alpha_k,\beta_k)}$$

1.6算数基本定理

算数基本定理：任意大于1的正整数N都可以唯一表示成素因子幂乘积的形式，令乘积中的素因子按照非递减序排列，则可得到标准分解式

$$N = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot {p_3}^{\alpha_3} \cdot {p_4}^{\alpha_4} \cdot \ldots\ldots \cdot {p_k}^{\alpha_k} \quad \alpha_i \in N^*, 1 \le i \le k$$

其中p_i $(1 \le i \le k)$是N的素因子，且$p_i < p_{i+1}$

重要性质：

若$a = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot \ldots \cdot {p_k}^{\alpha_k}$，$b = {p_1}^{\beta_1} \cdot {p_2}^{\beta_2} \cdot \ldots \cdot {p_k}^{\beta_k}$，则有

$$\gcd(a, b) = {p_1}^{\min(\alpha_1,\beta_1)} \cdot {p_2}^{\min(\alpha_2,\beta_2)} \cdot \ldots \cdot {p_k}^{\min(\alpha_k,\beta_k)}$$

$$\mathrm{lcm}(a, b) = {p_1}^{\max(\alpha_1,\beta_1)} \cdot {p_2}^{\max(\alpha_2,\beta_2)} \cdot \ldots \cdot {p_k}^{\max(\alpha_k,\beta_k)}$$

这个性质可以用来快速求解和多个数的最大公约数、最小公倍数相关的问题！！！

1.7素性分析

1.7素性分析的引入

分析一个数是否是素数称作对该数进行的素性分析，也常叫做素数测试。通俗来讲就是给一个数N，判断N是否是素数

1.7素性分析的引入

分析一个数是否是素数称作对该数进行的素性分析，也常叫做素数测试。通俗来讲就是给一个数N，判断N是否是素数

常用的方法：

1.试除法(定义)

2.改进的试除法

3.Eratosthenes筛法

4.Euler筛法

1.7素性分析-试除法

给定一个数N，然后从2～N-1依次判断是否能够整除N，如果能整除，则表示N是合数，否则表示N是素数，由于要尝试整除每个数，所以这种方法常叫做试除法

1.7素性分析-试除法

给定一个数N，然后从2 ~ N - 1依次判断是否能够整除N，如果能整除，则表示N是合数，否则表示N是素数，由于要尝试整除每个数，所以这种方法常叫做试除法

```
bool IsPrime (long long val)
{
    for (long long i = 2; i <= val - 1; i++)
        if (val % i == 0)
            return false;

    return true;
}
```

1.7素性分析-试除法

给定一个数N，然后从2 ~ N - 1依次判断是否能够整除N，如果能整除，则表示N是合数，否则表示N是素数，由于要尝试整除每个数，所以这种方法常叫做试除法

```
bool IsPrime (long long val)
{
    for (long long i = 2; i <= val - 1; i++)
        if (val % i == 0)
            return false;

    return true;
}
```

时间复杂度是O(n)的，其中n表示待进行素性分析的数

1.7素性分析-试除法

给定一个数N，然后从2 ~ N - 1依次判断是否能够整除N，如果能整除，则表示N是合数，否则表示N是素数，由于要尝试整除每个数，所以这种方法常叫做试除法

```
bool IsPrime (long long val)
{
    for (long long i = 2; i <= val - 1; i++)
        if (val % i == 0)
            return false;

    return true;
}
```

时间复杂度是O(n)的，其中n表示待进行素性分析的数

简单分析可知，其实没有必要枚举所有数，由素数的性质可将复杂度优化到$O(\sqrt{n})$

1.7素性分析-改进的试除法

由素数性质可知合数N一定有小于等于$\sqrt{N}$的素因子，则直接使用这些素数去尝试可以减少判断的次数，优化复杂度

1.7素性分析-改进的试除法

由素数性质可知合数N一定有小于等于$\sqrt{N}$的素因子，则直接使用这些素数去尝试可以减少判断的次数，优化复杂度

想一想我们该如何设计算法，真的需要先求出所有小于$\sqrt{N}$的素数吗？

1.7素性分析-改进的试除法

由素数性质可知合数N一定有小于等于$\sqrt{N}$的素因子，则直接使用这些素数去尝试可以减少判断的次数，优化复杂度

想一想我们该如何设计算法，真的需要先求出所有小于$\sqrt{N}$的素数吗?

实际上如果使用算数分解定理并结合上面的性质可以不需要预先知道小于$\sqrt{N}$的所有素因子

1.7素性分析-改进的试除法

先给出代码

```
bool IsPrime (long long val)
{
    for (long long i = 2; i * i <= val; i++)
        if (val % i == 0)
            return false;

    return true;
}
```

1.7素性分析-改进的试除法

先给出代码

```
bool IsPrime (long long val)
{
    for (long long i = 2; i * i <= val; i++)
        if (val % i == 0)
            return false;

    return true;
}
```

结论：上面的代码保证如果val是一个合数，则val会被其最小素因子整除，也就是说当return false时，此时的i一定是val的最小素因子

1.7素性分析-改进的试除法

先给出代码

```
bool IsPrime (long long val)
{
    for (long long i = 2; i * i <= val; i++)
        if (val % i == 0)
            return false;

    return true;
}
```

疑问：左边的代码是如何进行素数测试的？是如何保证val是合数时，会被其最小素因子整除的？

结论：上面的代码保证如果val是一个合数，则val会被其最小素因子整除，也就是说当return false时，此时的i一定是val的最小素因子

1.7素性分析-改进的试除法

先给出代码

```
bool IsPrime (long long val)
{
    for (long long i = 2; i * i <= val; i++)
        if (val % i == 0)
            return false;

    return true;
}
```

分析：

(i)如果val是合数，则val一定有小于等于$\sqrt{val}$的素因子，则循环枚举所有小于$\sqrt{val}$的数一定可以找到这个素因子。这说明了val如果是合数，则一定能够找到符合条件的素因子

结论：上面的代码保证如果val是一个合数，则val会被其最小素因子整除，也就是说当return false时，此时的i一定是val的最小素因子

1.7素性分析-改进的试除法

先给出代码

```
bool IsPrime (long long val)
{
    for (long long i = 2; i * i <= val; i++)
        if (val % i == 0)
            return false;

    return true;
}
```

分析：

(i)求证：如果val是一个合数，则会被其最小素因子所整除

结论：上面的代码保证如果val是一个合数，则val会被其最小素因子整除，也就是说当return false时，此时的i一定是val的最小素因子

1.7素性分析-改进的试除法

先给出代码

```
bool IsPrime (long long val)
{
    for (long long i = 2; i * i <= val; i++)
        if (val % i == 0)
            return false;

    return true;
}
```

分析：

(i) 求证：如果val是一个合数，则会被其最小素因子所整除

反证：假设val可以被其一个合数因子所整除

结论：上面的代码保证如果val是一个合数，则val会被其最小素因子整除，也就是说当return false时，此时的i一定是val的最小素因子

1.7素性分析-改进的试除法

先给出代码

```
bool IsPrime (long long val)
{
    for (long long i = 2; i * i <= val; i++)
        if (val % i == 0)
            return false;

    return true;
}
```

结论：上面的代码保证如果val是一个合数，则val会被其最小素因子整除，也就是说当return false时，此时的i一定是val的最小素因子

分析：

(i)求证：如果val是一个合数，则会被其最小素因子所整除

反证：假设val可以被其一个合数因子所整除

设i是一个合数，且满足i | val，由算数分解定理可知，i可以被分解成素因子幂乘积的形式，即一定存在能够整除i的素因子p_i，即有p_i | i，而由条件可知有i | val，则可推出p_i | val。而由于p_i < i，且是从小到大循环判断每个数的，则这就意味着val会被素因子p_i整除，不会被i所整除，与假设矛盾。这证明val一定会被其素因子整除

1.7素性分析-改进的试除法

先给出代码

```
bool IsPrime (long long val)
{
    for (long long i = 2; i * i <= val; i++)
        if (val % i == 0)
            return false;

    return true;
}
```

结论：上面的代码保证如果val是一个合数，则val会被其最小素因子整除，也就是说当return false时，此时的i一定是val的最小素因子

分析：

(i)求证：如果val是一个合数，则会被其最小素因子所整除

证明：

由于程序是从小到大枚举每个数的，这就保证一旦找到val的一个素因子，则该素因子一定是最小的。综上所述，原命题得证

1.7素性分析-改进的试除法

先给出代码

```
bool IsPrime (long long val)
{
    for (long long i = 2; i * i <= val; i++)
        if (val % i == 0)
            return false;

    return true;
}
```

分析：

(ii)如果val是素数，则val没有小于等于于$\sqrt{val}$的素因子，则循环完直接return true

结论：上面的代码保证如果val是一个合数，则val会被其最小素因子整除，也就是说当return false时，此时的i一定是val的最小素因子

1.7素性分析-改进的试除法

使用素数性质优化后的试除法的时间复杂度是O($\sqrt{n}$)的，这对于测试一个数是否是素数算是很不错的了，但是如果要测试一个范围之内的数是否是素数时，其复杂度是O(M*$\sqrt{n}$)的，其中M表示待测试数的个数，n表示待测试数的平均取值。**则很难满足程序竞赛的要求**

1.7素性分析-改进的试除法

使用素数性质优化后的试除法的时间复杂度是O($\sqrt{n}$)的，这对于测试一个数是否是素数算是很不错的了，但是如果要测试一个范围之内的数是否是素数时，其复杂度是O(M*$\sqrt{n}$)的，其中M表示待测试数的个数，n表示待测试数的平均取值。则很难满足程序竞赛的要求

想一想，是判断一个数是素数简单还是判断是合数简单？

1.7素性分析-改进的试除法

使用素数性质优化后的试除法的时间复杂度是$O(\sqrt{n})$的，这对于测试一个数是否是素数算是很不错的了，但是如果要测试一个范围之内的数是否是素数时，其复杂度是$O(M*\sqrt{n})$的，其中M表示待测试数的个数，n表示待测试数的平均取值。则很难满足程序竞赛的要求

想一想，是判断一个数是素数简单还是判断是合数简单？

当然判断是合数更简单，因为只要找到一个能够整除自己的素因子就可以了。而前面的试除法在判断出val是素数这个过程上会比判断出val是合数更花时间

1.7素性分析-改进的试除法

使用素数性质优化后的试除法的时间复杂度是O($\sqrt{n}$)的，这对于测试一个数是否是素数算是很不错的了，但是如果要测试一个范围之内的数是否是素数时，其复杂度是O(M*$\sqrt{n}$)的，其中M表示待测试数的个数，n表示待测试数的平均取值。则很难满足程序竞赛的要求

想一想，是判断一个数是素数简单还是判断是合数简单?

当然判断是合数更简单，因为只要找到一个能够整除自己的素因子就可以了。而前面的试除法在判断出val是素数这个过程上会比判断出val是合数更花时间

联想到大于等于2的自然数要么是素数，要么是合数，且判断合数更简单，则不难想到：如果先将给定范围内的所有合数找出来，然后在总范围内的所有数中除去这些合数，则剩下的就是我们想要的素数。基于上述的优化就是筛法

素性分析-埃氏筛法(Sieve of Eratosthenes)

埃氏筛法最早由古希腊数学家Eratosthenes提出，可以将2~N的所有素数找到

素性分析-埃氏筛法(Sieve of Eratosthenes)

埃氏筛法最早由古希腊数学家Eratosthenes提出，可以将2~N的所有素数找到

埃氏筛法依次将当前最小素数的倍数删去(素数的倍数一定是合数)，最后留下来的就是素数，比如先将2的倍数删去，则4、6、8、...依次被删去了，2之后的最小的没有被删去的数是3，此时可以确定3是素数(想想这是为什么？)，然后将3的倍数删去，则6、9...依次被删去

素性分析-埃氏筛法(Sieve of Eratosthenes)

埃氏筛法最早由古希腊数学家Eratosthenes提出，可以将2~N的所有素数找到

埃氏筛法依次将当前最小素数的倍数删去(素数的倍数一定是合数)，最后留下来的就是素数，比如先将2的倍数删去，则4、6、8、...依次被删去了，2之后的最小的没有被删去的数是3，此时可以确定3是素数(想想这是为什么？)，然后将3的倍数删去，则6、9...依次被删去

这里的删去操作不是真的将数删去，而是仅仅将合数标记

素性分析-埃氏筛法(Sieve of Eratosthenes)

埃氏筛法最早由古希腊数学家Eratosthenes提出，可以将2~N的所有素数找到

埃氏筛法依次将当前最小素数的倍数删去(素数的倍数一定是合数)，最后留下来的就是素数，比如先将2的倍数删去，则4、6、8、...依次被删去了，2之后的最小的没有被删去的数是3，此时可以确定3是素数(想想这是为什么？)，然后将3的倍数删去，则6、9...依次被删去

比如：

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 ...

↑

素数

素性分析-埃氏筛法(Sieve of Eratosthenes)

埃氏筛法最早由古希腊数学家Eratosthenes提出，可以将2~N的所有素数找到

埃氏筛法依次将当前最小素数的倍数删去(素数的倍数一定是合数)，最后留下来的就是素数，比如先将2的倍数删去，则4、6、8、...依次被删去了，2之后的最小的没有被删去的数是3，此时可以确定3是素数(想想这是为什么？)，然后将3的倍数删去，则6、9...依次被删去

比如：

1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ ...

素数

素性分析-埃氏筛法(Sieve of Eratosthenes)

埃氏筛法最早由古希腊数学家Eratosthenes提出，可以将2~N的所有素数找到

埃氏筛法依次将当前最小素数的倍数删去(素数的倍数一定是合数)，最后留下来的就是素数，比如先将2的倍数删去，则4、6、8、...依次被删去了，2之后的最小的没有被删去的数是3，此时可以确定3是素数(想想这是为什么？)，然后将3的倍数删去，则6、9...依次被删去

比如：

1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ …

素数

素性分析-埃氏筛法(Sieve of Eratosthenes)

埃氏筛法最早由古希腊数学家Eratosthenes提出，可以将2~N的所有素数找到

埃氏筛法依次将当前最小素数的倍数删去(素数的倍数一定是合数)，最后留下来的就是素数，比如先将2的倍数删去，则4、6、8、...依次被删去了，2之后的最小的没有被删去的数是3，此时可以确定3是素数(想想这是为什么？)，然后将3的倍数删去，则6、9...依次被删去

比如：

1 2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ...

素数

素性分析-埃氏筛法(Sieve of Eratosthenes)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

//Eratosthenes筛法
void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
        if (prime[i])
        {
            primelist[prime_len++] = i;
            for (long long j = 2; j * i <= maxn; j++)
                prime[j*i] = false;
        }
}
```

素性分析-埃氏筛法(Sieve of Eratosthenes)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

//Eratosthenes筛法
void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
        if (prime[i])
        {
            primelist[prime_len++] = i;
            for (long long j = 2; j * i <= maxn; j++)
                prime[j*i] = false;
        }
}
```

解析：使用prime数组将合数进行标记，先初始化为true，然后依次将素数的倍数全部标记成false。使用primelist数组将素数存起来

素性分析-埃氏筛法(Sieve of Eratosthenes)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

//Eratosthenes筛法
void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
        if (prime[i])
        {
            primelist[prime_len++] = i;
            for (long long j = 2; j * i <= maxn; j++)
                prime[j*i] = false;
        }
}
```

解析：使用prime数组将合数进行标记，先初始化为true，然后依次将素数的倍数全部标记成false。使用primelist数组将素数存起来

还记得前面说的最小还没有删去的数就是素数吗？

素性分析-埃氏筛法(Sieve of Eratosthenes)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

//Eratosthenes筛法
void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
        if (prime[i])
        {
            primelist[prime_len++] = i;
            for (long long j = 2; j * i <= maxn; j++)
                prime[j*i] = false;
        }
}
```

解析：使用prime数组将合数进行标记，先初始化为true，然后依次将素数的倍数全部标记成false。使用primelist数组将素数存起来

还记得前面说的最小还没有删去的数就是素数吗？

这是因为一个数如果是合数，就一定有小于等于其根号值的素因子，则一定是其素因子的倍数进而被删去。如果当前值没有被删掉，且满足所有小于其值的数在判断当前数之前已经被判断过时(从小到大循环判断)。则此时的数才是一个素数

素性分析-埃氏筛法(Sieve of Eratosthenes)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

//Eratosthenes筛法
void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
        if (prime[i])
        {
            primelist[prime_len++] = i;
            for (long long j = 2; j * i <= maxn; j++)
                prime[j*i] = false;
        }
}
```

解析：使用prime数组将合数进行标记，先初始化为true，然后依次将素数的倍数全部标记成false。使用primelist数组将素数存起来

还记得前面说的最小还没有删去的数就是素数吗？

注意：只有最小还没有删去的数满足所有小于其值的数在之前都被判断过！！这是因为一个非最小还没有删去的数可能会被最小还没有删去的数删去，这就表明小于非最小还没有删去的数没有被全部判断过

素性分析-埃氏筛法(Sieve of Eratosthenes)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

//Eratosthenes筛法
void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
        if (prime[i])
        {
            primelist[prime_len++] = i;
            for (long long j = 2; j * i <= maxn; j++)
                prime[j*i] = false;
        }
}
```

解析：使用prime数组将合数进行标记，先初始化为true，然后依次将素数的倍数全部标记成false。使用primelist数组将素数存起来

时间复杂度：

$$\frac{n}{2} + \frac{n}{3} + \ldots \frac{n}{\frac{n}{2}} = n \cdot \sum_{k=2}^{\frac{n}{2}} \frac{1}{k} < n \cdot (\ln n + c)$$

$\Rightarrow n \cdot \ln n$，其中c表示Euler常数

素性分析-埃氏筛法(Sieve of Eratosthenes)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

//Eratosthenes筛法
void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
        if (prime[i])
        {
            primelist[prime_len++] = i;
            for (long long j = 2; j * i <= maxn; j++)
                prime[j*i] = false;
        }
}
```

解析：使用prime数组将合数进行标记，先初始化为true，然后依次将素数的倍数全部标记成false。使用primelist数组将素数存起来

时间复杂度：

$$\frac{n}{2} + \frac{n}{3} + \ldots \frac{n}{\frac{n}{2}} = \mathrm{n} \cdot \sum_{k=2}^{\frac{n}{2}} \frac{1}{k} < \mathrm{n} \cdot (\ln n + \mathrm{c})$$

$\Rightarrow \mathrm{n} \cdot \ln n$，其中c表示Euler常数

则一个不太精确的估计是$O(n \cdot \ln n)$，一个更好的估计是$O(n\log\log n)$

素性分析-埃氏筛法(Sieve of Eratosthenes)

由前面的分析可得，埃氏筛法保证一个数是合数，则一定会被其素因子删去，但是这个数可能会被其多个素因子所重复删去，比如给出一个合数6：

$$6 = 2^1 \cdot 3^1$$

则6会被2(2 · 3 = 6)删去一次，然后又会被3(3 · 2 = 6)删去一次，造成冗余的操作

素性分析-埃氏筛法(Sieve of Eratosthenes)

由前面的分析可得，埃氏筛法保证一个数是合数，则一定会被其素因子删去，但是这个数可能会被其多个素因子所重复删去，比如给出一个合数6：

$$6 = 2^1 \cdot 3^1$$

则6会被2($2 \cdot 3 = 6$)删去一次，然后又会被3($3 \cdot 2 = 6$)删去一次，造成冗余的操作

一个显然的想法就是：能不能进行一些优化，使得如果一个数是合数，就只会被其最小素因子所删去？

素性分析-埃氏筛法(Sieve of Eratosthenes)

由前面的分析可得，埃氏筛法保证一个数是合数，则一定会被其素因子删去，但是这个数可能会被其多个素因子所重复删去，比如给出一个合数6：

$$6 = 2^1 \cdot 3^1$$

则6会被2($2 \cdot 3 = 6$)删去一次，然后又会被3($3 \cdot 2 = 6$)删去一次，造成冗余的操作

一个显然的想法就是：能不能进行一些优化，使得如果一个数是合数，就只会被其最小素因子所删去？

还记得前面讲的改进的试除法吗？使用试除法的话，合数不就是被其最小素因子所整除的吗？

素性分析-埃氏筛法(Sieve of Eratosthenes)

由前面的分析可得，埃氏筛法保证一个数是合数，则一定会被其素因子删去，但是这个数可能会被其多个素因子所重复删去，比如给出一个合数6：

$$6 = 2^1 \cdot 3^1$$

则6会被2($2 \cdot 3 = 6$)删去一次，然后又会被3($3 \cdot 2 = 6$)删去一次，造成冗余的操作

一个显然的想法就是：能不能进行一些优化，使得如果一个数是合数，就只会被其最小素因子所删去？

还记得前面讲的改进的试除法吗？使用试除法的话，合数不就是被其最小素因子所整除的吗？

将试除法的思想类似地应用到埃氏筛法上优化得到的就是欧拉筛法

素性分析-欧拉筛法(Sieve of Euler)

欧拉筛法是一种优化过的埃氏筛法，所以其很多地方和埃氏筛法是一样的，但是为了保证每个合数只被其最小素因子所删去，需要在删数操作上面有所不同，这也是优化的地方：

素性分析-欧拉筛法(Sieve of Euler)

欧拉筛法是一种优化过的埃氏筛法，所以其很多地方和埃氏筛法是一样的，但是为了保证每个合数只被其最小素因子所删去，需要在删数操作上面有所不同，这也是优化的地方：

埃氏筛法：在删数时，需要先判断当前数是否是素数，如果当前数是素数，则将其所有的倍数都标记为合数

素性分析-欧拉筛法(Sieve of Euler)

欧拉筛法是一种优化过的埃氏筛法，所以其很多地方和埃氏筛法是一样的，但是为了保证每个合数只被其最小素因子所删去，需要在删数操作上面有所不同，这也是优化的地方：

埃氏筛法：在删数时，需要先判断当前数是否是素数，如果当前数是素数，则将其所有的倍数都标记为合数

欧拉筛法：在删数时，对于当前数来说，无论其是素数还是合数，都将其乘上已经找到的所有素数(需满足一定的条件)，并将得到的值标记为合数

素性分析-欧拉筛法(Sieve of Euler)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
    {
        if (prime[i])
            primelist[prime_len++] = i;

        for (long long j = 0; j < prime_len; j++)
        {
            if (i * primelist[j] > maxn)
                break;

            prime[i*primelist[j]] = false;
            if (i % primelist[j] == 0)
                break;
        }
    }
}
```

素性分析-欧拉筛法(Sieve of Euler)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
    {
        if (prime[i])
            primelist[prime_len++] = i;

        for (long long j = 0; j < prime_len; j++)
        {
            if (i * primelist[j] > maxn)
                break;

            prime[i*primelist[j]] = false;
            if (i % primelist[j] == 0)
                break;
        }
    }
}
```

Euler筛法是如何保证每个合数只被其最小素因子所删去的呢?

素性分析-欧拉筛法(Sieve of Euler)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
    {
        if (prime[i])
            primelist[prime_len++] = i;

        for (long long j = 0; j < prime_len; j++)
        {
            if (i * primelist[j] > maxn)
                break;

            prime[i*primelist[j]] = false;
            if (i % primelist[j] == 0)
                break;
        }
    }
}
```

Euler筛法是如何保证每个合数只被其最小素因子所删去的呢?

设primelist[j]为p_j , j ∈ [0, primelen).并将当前数i进行素因子分解表示成:

i = $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots p_j^{\alpha_j} \cdot \ldots\ldots \cdot p_n^{\alpha_n}$ $\alpha_k \in N$, k ∈[0,primelen)

素性分析-欧拉筛法(Sieve of Euler)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
    {
        if (prime[i])
            primelist[prime_len++] = i;

        for (long long j = 0; j < prime_len; j++)
        {
            if (i * primelist[j] > maxn)
                break;

            prime[i*primelist[j]] = false;
            if (i % primelist[j] == 0)
                break;
        }
    }
}
```

Euler筛法是如何保证每个合数只被其最小素因子所删去的呢?

设primelist[j]为p_j , j ∈ [0, primelen). 并将当前数i进行素因子分解表示成:

$i = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots p_j^{\alpha_j} \cdot \ldots\ldots \cdot p_n^{\alpha_n} \quad \alpha_k \in N$, k ∈[0,primelen)

(1)若$p_j \mid i$, 则可得$\alpha_j \geq 1$, 如果此时继续判断，则会得到

$i \cdot p_k = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots p_j^{\alpha_j} \cdot \ldots p_k^{\alpha_k+1} \ldots \cdot p_n^{\alpha_n}$

k > j, k ∈ [0, primelen)

但由于$\alpha_j \geq 1$,则此时总有$p_j \mid i \cdot p_k$, 且j < k

$\Longrightarrow p_j < p_k$

$\Longrightarrow p_k$不是能够整除$i \cdot p_k$的最小素因子

素性分析-欧拉筛法(Sieve of Euler)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
    {
        if (prime[i])
            primelist[prime_len++] = i;

        for (long long j = 0; j < prime_len; j++)
        {
            if (i * primelist[j] > maxn)
                break;

            prime[i*primelist[j]] = false;
            if (i % primelist[j] == 0)
                break;
        }
    }
}
```

Euler筛法是如何保证每个合数只被其最小素因子所删去的呢?

设primelist[j]为p_j , j ∈ [0, primelen). 并将当前数i进行素因子分解表示成:

$$i = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots p_j^{\alpha_j} \cdot \ldots\ldots \cdot p_n^{\alpha_n} \quad \alpha_k \in N,$$

k ∈[0,primelen)

(2)若$p_j \nmid$ i, 则会继续循环查找

素性分析-欧拉筛法(Sieve of Euler)

```
//定义素数判断表和素数表
bool prime[maxn+10];
long long primelist[maxn+10], prime_len;

void GetPrime ()
{
    memset (prime, true, sizeof (prime));
    prime_len = 0;
    for (long long i = 2; i <= maxn; i++)
    {
        if (prime[i])
            primelist[prime_len++] = i;

        for (long long j = 0; j < prime_len; j++)
        {
            if (i * primelist[j] > maxn)
                break;

            prime[i*primelist[j]] = false;
            if (i % primelist[j] == 0)
                break;
        }
    }
}
```

Euler筛法是如何保证每个合数只被其最小素因子所删去的呢?

设primelist[j]为p_j , j ∈ [0, primelen). 并将当前数i进行素因子分解表示成:

i = $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots p_j^{\alpha_j} \cdot \ldots\ldots \cdot p_n^{\alpha_n}$ $\alpha_k \in N$, k ∈[0,primelen)

这就意味着一旦出现 i % primelist[j] == 0，则后面的素数都不需要再判断了，直接break就可以了，这就保证了每个合数都是被其最小素因子所删去的

素性分析-四种算法的总结

算法	类型	时间复杂度	描述
试除法(定义)	单素数测试	O (n)	
改进的试除法	单素数测试	O $(\sqrt{n})$	
Eratosthenes筛法	区间素数测试	O $(n\log\log n)$	k是一个足够小的数
Euler筛法	区间素数测试	O $(k \cdot n)$(亚线性)	

素性分析-四种算法的总结

算法	类型	时间复杂度	描述
试除法(定义)	单素数测试	O (n)	
改进的试除法	单素数测试	O ($\sqrt{n}$)	
Eratosthenes筛法	区间素数测试	O ($n\log\log n$)	k是一个足够小的数
Euler筛法	区间素数测试	O (k · n)(亚线性)	

总结来说就是，如果要求解的是与单素数相关的问题，则使用O ($\sqrt{n}$)的试除法求解就可以了，而且试除法也是后面对整数进行素因子分解、求解欧拉函数的方法之一

素性分析-四种算法的总结

算法	类型	时间复杂度	描述
试除法(定义)	单素数测试	O (n)	
改进的试除法	单素数测试	O $(\sqrt{n})$	
Eratosthenes筛法	区间素数测试	O $(n\log\log n)$	k是一个足够小的数
Euler筛法	区间素数测试	O (k · n)(亚线性)	

总结来说就是，如果要求解的是与单素数相关的问题，则使用O $(\sqrt{n})$的试除法求解就可以了，而且试除法也是后面对整数进行素因子分解、求解欧拉函数的方法之一

如果问题涉及大量、大范围的素数操作，此时就需要使用筛法预打表处理了。如果n小于1e5，则两种筛法都可以使用。但是如果n大于1e5的话，此时建议使用Euler筛法。当n近似等于7e5时，Eratosthenes筛法就已经比Euler筛法慢上10倍多了

素性分析

(POJ2689)题目大意：先给出两个整数L和U$(1 \le L < U \le 2^{31} - 1)$，要找出两个相邻的素数$C_1$和$C_2$ $(L \le C_1 < C_2 \le U)$，并满足C_1和C_2的距离最小。如果具有最小距离的相邻素数不只有一对，则输出最初的。还需要找出两个相邻的素数D_1和D_2是最大的(同样是有多对时选择最初的)，样例保证每次给出的L和U的差不超过1e6

素性分析

(POJ2689)题目大意：先给出两个整数L和U($1 \le L < U \le 2^{31}-1$)，要找出两个相邻的素数C_1和C_2 ($L \le C_1 < C_2 \le U$)，并满足C_1和C_2的距离最小。如果具有最小距离的相邻素数不只有一对，则输出最初的。还需要找出两个相邻的素数D_1和D_2是最大的(同样是有多对时选择最初的)，样例保证每次给出的L和U的差不超过1e6

解析：很显然需要使用筛法预处理出素数表，如果我能将$[1, 2^{31}-1]$范围内的素数都找到的话，直接可以从小到大扫一遍素数表，同时维护并更新相邻素数之间的最小、最大值，复杂度是O(n)的，n为素数表中元素的个数。但现在问题的关键是$[1, 2^{31}-1]$范围太大，其中的素数总数不可能用数组都存下来

素性分析

(POJ2689)题目大意：先给出两个整数L和U($1 \le L < U \le 2^{31} - 1$)，要找出两个相邻的素数$C_1$和$C_2$ ($L \le C_1 < C_2 \le U$)，并满足C_1和C_2的距离最小。如果具有最小距离的相邻素数不只有一对，则输出最初的。还需要找出两个相邻的素数D_1和D_2是最大的(同样是有多对时选择最初的)，样例保证每次给出的L和U的差不超过1e6

解析：此时注意题目中最后给出的一个条件：$U - L \le$ 1e6，则想到我不先把所有范围内的素数都找到，而是对每次所给的区间[L,U]使用筛法将给定区间的素数找出来就可以了，而1e6的数据规模是可以开数组存下的

素性分析

(POJ2689)题目大意：先给出两个整数L和U($1 \le L < U \le 2^{31} - 1$)，要找出两个相邻的素数$C_1$和$C_2$ ($L \le C_1 < C_2 \le U$)，并满足C_1和C_2的距离最小。如果具有最小距离的相邻素数不只有一对，则输出最初的。还需要找出两个相邻的素数D_1和D_2是最大的(同样是有多对时选择最初的)，样例保证每次给出的L和U的差不超过1e6

解析：此时注意题目中最后给出的一个条件：$U - L \le 1e6$，则想到我不先把所有范围内的素数都找到，而是对每次所给的区间[L,U]使用筛法将给定区间的素数找出来就可以了，而1e6的数据规模是可以开数组存下的

具体实现是：想要筛掉[L,U]区间内的合数，需要知道$[1,\sqrt{U}]$内的素因子。则预先将$[1,\sqrt{2^{31}-1}]$内的素数筛出来，其中$\sqrt{2^{31}-1} \approx 46341$，完全可以开数组存下。然后对于读入的每个区间，用筛好的素数的倍数删去给定区间中的数并标记。最后遍历一下区间内的素数更新最值即可

1.8整数素因子分解

1.8整数素因子分解

那么现在给出一个正整数n，如何将n分解成前面所述的素因子幂乘积的形式？

1.8整数素因子分解

那么现在给出一个正整数n，如何将n分解成前面所述的素因子幂乘积的形式？

常用的方法：

1.试除法

2.筛法(Eratosthenes筛和Euler筛)

3.*Pollard rho算法(会在后面同余式定理部分介绍)

1.8整数素因子分解

那么现在给出一个正整数n，如何将n分解成前面所述的素因子幂乘积的形式?

常用的方法：

1.试除法

2.筛法(Eratosthenes筛和Euler筛)

3.*Pollard rho算法(会在后面同余式定理部分介绍)

用于进行素性分析的试除法、筛法都是从小到大对数进行测试，而且都可以"构造"出素数(即判断当前数是否是素数)，这和素因子分解中从小到大，由素数幂乘积组成的标准分解式的求解过程不谋而合，所以使用试除法和筛法进行整数素因子分解是十分自然的

1.8整数素因子分解

1.试除法

对于待分解的数n来说，从小到大枚举2 ~ $\sqrt{n}$中的所有数，记当前枚举的数是i，则如果i | n，则i就是n的一个素因子，并用当前数n除尽i。如果循环完后的n > 1，则此时的n就是一个素因子(想想这是为什么？)

1.8整 数 素 因 子 分 解

```
long long factor[maxn+10], fac_len;//素因子表和表的长度

void GetFactor (long long val)
{
    fac_len = 0;
    for (long long i = 2; i * i <= val; i++)
    {
        while (val % i == 0)
        {
            factor[fac_len++] = i;
            val /= i;
        }
    }

    if (val != 1)
        factor[fac_len++] = val;
}
```

1.8整数素因子分解

```
long long factor[maxn+10], fac_len;//素因子表和表的长度

void GetFactor (long long val)
{
    fac_len = 0;
    for (long long i = 2; i * i <= val; i++)
    {
        while (val % i == 0)
        {
            factor[fac_len++] = i;
            val /= i;
        }
    }

    if (val != 1)
        factor[fac_len++] = val;
}
```

解析：注意这里factor数组中会重复出现同一个素因子，比如：

$$12 = 2^2 \cdot 3^1$$

则factor数组中分别存储的是2、2和3。当然也可以改一下代码将每个素因子只存储一个，这多半取决于题目的要求

1.8整数素因子分解

```
long long factor[maxn+10], fac_len;//素因子表和表的长度

void GetFactor (long long val)
{
    fac_len = 0;
    for (long long i = 2; i * i <= val; i++)
    {
        while (val % i == 0)
        {
            factor[fac_len++] = i;
            val /= i;
        }
    }

    if (val != 1)
        factor[fac_len++] = val;
}
```

解析：注意这里factor数组中会重复出现同一个素因子，比如：

$$12 = 2^2 \cdot 3^1$$

则factor数组中分别存储的是2、2和3。当然也可以改一下代码将每个素因子只存储一个，这多半取决于题目的要求

1.8整数素因子分解

```
long long factor[maxn+10], fac_len;//素因子表和表的长度

void GetFactor (long long val)
{
    fac_len = 0;
    for (long long i = 2; i * i <= val; i++)
    {
        while (val % i == 0)
        {
            factor[fac_len++] = i;
            val /= i;
        }
    }

    if (val != 1)
        factor[fac_len++] = val;
}
```

解析：注意这里factor数组中会重复出现同一个素因子，比如：

$$12 = 2^2 \cdot 3^1$$

则factor数组中分别存储的是2、2和3。当然也可以改一下代码将每个素因子只存储一个，这多半取决于题目的要求

想一想，为什么只要满足 val % i == 0，则i就是val的素因子？

1.8整数素因子分解

```
long long factor[maxn+10], fac_len;//素因子表和表的长度

void GetFactor (long long val)
{
    fac_len = 0;
    for (long long i = 2; i * i <= val; i++)
    {
        while (val % i == 0)
        {
            factor[fac_len++] = i;
            val /= i;
        }
    }

    if (val != 1)
        factor[fac_len++] = val;
}
```

解析：注意这里factor数组中会重复出现同一个素因子，比如：

$$12 = 2^2 \cdot 3^1$$

则factor数组中分别存储的是2、2和3。当然也可以改一下代码将每个素因子只存储一个，这多半取决于题目的要求

想一想，为什么只要满足 val % i == 0，则i就是val的素因子？

前面讲试除法时已经证明过了

1.8整数素因子分解

```
long long factor[maxn+10], fac_len;//素因子表和表的长度

void GetFactor (long long val)
{
    fac_len = 0;
    for (long long i = 2; i * i <= val; i++)
    {
        while (val % i == 0)
        {
            factor[fac_len++] = i;
            val /= i;
        }
    }

    if (val != 1)
        factor[fac_len++] = val;
}
```

解析：注意这里factor数组中会重复出现同一个素因子，比如：

$$12 = 2^2 \cdot 3^1$$

则factor数组中分别存储的是2、2和3。当然也可以改一下代码将每个素因子只存储一个，这多半取决于题目的要求

看代码发现val的大小在循环过程中会减小(val /= i)，而val又是判断循环是否终止的条件，那么val的减小，会不会导致循环少判断了一些素数？

1.8整数素因子分解

```
long long factor[maxn+10], fac_len;//素因子表和表的长度

void GetFactor (long long val)
{
    fac_len = 0;
    for (long long i = 2; i * i <= val; i++)
    {
        while (val % i == 0)
        {
            factor[fac_len++] = i;
            val /= i;
        }
    }

    if (val != 1)
        factor[fac_len++] = val;
}
```

解析：注意这里factor数组中会重复出现同一个素因子，比如：

$$12 = 2^2 \cdot 3^1$$

则factor数组中分别存储的是2、2和3。当然也可以改一下代码将每个素因子只存储一个，这多半取决于题目的要求

看代码发现val的大小在循环过程中会减小(val /= i)，而val又是判断循环是否终止的条件，那么val的减小，会不会导致循环少判断了一些素数?

肯定不会，因为我们是从小到大判断的素因子，这就意味着当前被除尽的素因子之前没有还没判断的素因子。所以不会出现越过某些素因子进而判断其之后的数，导致漏判的情况

1.8整数素因子分解

```
long long factor[maxn+10], fac_len;//素因子表和表的长度

void GetFactor (long long val)
{
    fac_len = 0;
    for (long long i = 2; i * i <= val; i++)
    {
        while (val % i == 0)
        {
            factor[fac_len++] = i;
            val /= i;
        }
    }

    if (val != 1)
        factor[fac_len++] = val;
}
```

解析：注意这里factor数组中会重复出现同一个素因子，比如：

$$12 = 2^2 \cdot 3^1$$

则factor数组中分别存储的是2、2和3。当然也可以改一下代码将每个素因子只存储一个，这多半取决于题目的要求

想一想，为什么最后如果$val \neq 1$（或者$val > 1$），则val就是原来数的一个素因子？

1.8整数素因子分解

```
long long factor[maxn+10], fac_len;//素因子表和表的长度

void GetFactor (long long val)
{
    fac_len = 0;
    for (long long i = 2; i * i <= val; i++)
    {
        while (val % i == 0)
        {
            factor[fac_len++] = i;
            val /= i;
        }
    }

    if (val != 1)
        factor[fac_len++] = val;
}
```

解析：注意这里factor数组中会重复出现同一个素因子，比如：

$$12 = 2^2 \cdot 3^1$$

则factor数组中分别存储的是2、2和3。当然也可以改一下代码将每个素因子只存储一个，这多半取决于题目的要求

想一想，为什么最后如果$val \neq 1$(或者$val > 1$)，则val就是原来数的一个素因子？

如果提示一下此时val在标准分解式中的指数是1，能想到这是为什么吗？

1.8整数素因子分解

```
long long factor[maxn+10], fac_len;//素因子表和表的长度

void GetFactor (long long val)
{
    fac_len = 0;
    for (long long i = 2; i * i <= val; i++)
    {
        while (val % i == 0)
        {
            factor[fac_len++] = i;
            val /= i;
        }
    }

    if (val != 1)
        factor[fac_len++] = val;
}
```

解析：注意这里factor数组中会重复出现同一个素因子，比如：

$$12 = 2^2 \cdot 3^1$$

则factor数组中分别存储的是2、2和3。当然也可以改一下代码将每个素因子只存储一个，这多半取决于题目的要求

想一想，为什么最后如果val $\neq$ 1(或者val $>$ 1)，则val就是原来数的一个素因子？

如果提示一下此时val在标准分解式中的指数是1，能想到这是为什么吗？

假设最后剩下的数为*val $= {p_k}^{\alpha_k}$

若$\alpha_k \geq 2$，则当i $= p_k$时，i · i $= {p_k}^2 \leq$ val，满足循环条件，此时*val会在while循环中除尽素因子p_k，*val $\Longrightarrow$ 1

1.8整数素因子分解

```
long long factor[maxn+10], fac_len;//素因子表和表的长度

void GetFactor (long long val)
{
    fac_len = 0;
    for (long long i = 2; i * i <= val; i++)
    {
        while (val % i == 0)
        {
            factor[fac_len++] = i;
            val /= i;
        }
    }

    if (val != 1)
        factor[fac_len++] = val;
}
```

解析：注意这里factor数组中会重复出现同一个素因子，比如：

$$12 = 2^2 \cdot 3^1$$

则factor数组中分别存储的是2、2和3。当然也可以改一下代码将每个素因子只存储一个，这多半取决于题目的要求

想一想，为什么最后如果$val \neq 1$(或者$val > 1$)，则val就是原来数的一个素因子？

如果提示一下此时val在标准分解式中的指数是1，能想到这是为什么吗？

假设最后剩下的数为*$val = {p_k}^{\alpha_k}$

若$\alpha_k = 1$，则当$i = p_k$时，$i \cdot i = {p_k}^2 > val$，不满足循环条件，此时*$val = {p_k}^1$中的素因子$p_k$还没有被判断就退出了循环。所以此时需要额外判断一下

1.8整数素因子分解

2.筛法

由于合数a一定有小于等于其$\sqrt{a}$的素因子，所以对于小于maxn的待分解的数n来说，直接试除小于等于$\sqrt{n}$的所有素数即可。可以先将2~$\sqrt{maxn}$中的所有素数预打表，然后将n对应素数表中的素数一一试除

可以使用Eratosthenes筛法打素数表，也可以使用Euler筛法打素数表

1.8整数素因子分解

```
//在调用GetFactor之前需要先打出素数表
void GetFactor (long long val)
{
    fac_len = 0;
    for (long long i = 0; i < prime_len && primelist[i] * primelist[i] <= val; i++)
    {
        while (val % primelist[i] == 0)
        {
            factor[fac_len++] = primelist[i];
            val /= primelist[i];
        }
    }

    if (val != 1)
        factor[fac_len++] = val;
}
```

解析：这里没有给出打素数表的代码(不熟悉的可以参考前面介绍筛法时的代码)。代码总体上和直接使用试除法的代码相同

1.8整数素因子分解

(LIGHTOJ1215)题目大意：现有整数a、b、c和L，其中a、b和L已知，要求满足[a, b, c] = L的最小c的值，如果没有解则输出impossible

1.8整数素因子分解

(LIGHTOJ1215)题目大意：现有整数a、b、c和L，其中a、b和L已知，要求满足[a, b, c] = L的最小c的值，如果没有解则输出impossible

解析：求解与多个数的gcd或者是lcm相关的问题自然首先想到素因子分解。前面说若a = $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$，b = $p_1^{\beta_1} \cdot p_2^{\beta_2} \cdot \ldots \cdot p_k^{\beta_k}$，则有

$$\text{lcm (a, b)} = p_1^{\max(\alpha_1,\beta_1)} \cdot p_2^{\max(\alpha_2,\beta_2)} \cdot \ldots \cdot p_k^{\max(\alpha_k,\beta_k)}$$

1.8整数素因子分解

(LIGHTOJ1215)题目大意：现有整数a、b、c和L，其中a、b和L已知，要求满足[a, b, c] = L的最小c的值，如果没有解则输出impossible

解析：求解与多个数的gcd或者是lcm相关的问题自然首先想到素因子分解。前面说若a = $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot … \cdot p_k^{\alpha_k}$，b = $p_1^{\beta_1} \cdot p_2^{\beta_2} \cdot … \cdot p_k^{\beta_k}$，则有

$$\text{lcm (a, b)} = p_1^{\max(\alpha_1,\beta_1)} \cdot p_2^{\max(\alpha_2,\beta_2)} \cdot … \cdot p_k^{\max(\alpha_k,\beta_k)}$$

对于三个数以上仍满足：

$$\text{lcm } (a_1, a_2, …a_n) = p_1^{\max(\alpha_1,\beta_1,…\gamma_1)} \cdot p_2^{\max(\alpha_2,\beta_2,…\gamma_2)} \cdot … \cdot p_k^{\max(\alpha_k,\beta_k,…\gamma_k)}$$

1.8整数素因子分解

(LIGHTOJ1215)题目大意：现有整数a、b、c和L，其中a、b和L已知，要求满足[a, b, c] = L的最小c的值，如果没有解则输出impossible

解析：求解与多个数的gcd或者是lcm相关的问题自然首先想到素因子分解。前面说若a = $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$，b = $p_1^{\beta_1} \cdot p_2^{\beta_2} \cdot \ldots \cdot p_k^{\beta_k}$，则有

$$\text{lcm}(a, b) = p_1^{\max(\alpha_1,\beta_1)} \cdot p_2^{\max(\alpha_2,\beta_2)} \cdot \ldots \cdot p_k^{\max(\alpha_k,\beta_k)}$$

对于三个数以上仍满足：

$$\text{lcm}(a_1, a_2, \ldots a_n) = p_1^{\max(\alpha_1,\beta_1,\ldots\gamma_1)} \cdot p_2^{\max(\alpha_2,\beta_2,\ldots\gamma_2)} \cdot \ldots \cdot p_k^{\max(\alpha_k,\beta_k,\ldots\gamma_k)}$$

直接判断对应的素因子的指数是否合理就可以了

1.8整数素因子分解

(LIGHTOJ1215)题目大意：现有整数a、b、c和L，其中a、b和L已知，要求满足[a, b, c] = L的最小c的值，如果没有解则输出impossible

解析：求解与多个数的gcd或者是lcm相关的问题自然首先想到素因子分解。则先将a、b和L进行素因子分解，然后从小到大枚举判断L的每个素因子的指数值k，若a和b对应的指数值的最大值小于k，则c的对应的指数值就是k。若a和b对应的指数值的最大值等于k，则不需要进行任何操作。若大于k，则不可能出现这种情况，直接输出”impossible”即可。枚举完L的所有素因子之后，注意要查看此时a和b是否还有没在前面判断中出现的素因子。若还有，则也输出”impossible”。反之，则输出c的值

1.8整数素因子分解

(LIGHTOJ1215)题目大意：现有整数a、b、c和L，其中a、b和L已知，要求满足[a, b, c] = L的最小c的值，如果没有解则输出impossible

解析：求解与多个数的gcd或者是lcm相关的问题自然首先想到素因子分解。则先将a、b和L进行素因子分解，然后从小到大枚举判断L的每个素因子的指数值k，若a和b对应的指数值的最大值小于k，则c的对应的指数值就是k。若a和b对应的指数值的最大值等于k，则不需要进行任何操作。若大于k，则不可能出现这种情况，直接输出”impossible”即可。枚举完L的所有素因子之后，注意要查看此时a和b是否还有没在前面判断中出现的素因子。若还有，则也输出”impossible”。反之，则输出c的值

当然也可以先将a和b的最小公倍数k = [a,b]求出来，然后再求解[k,c] = L，此时只需要对k和L进行素因子分解就可以了

不定方程与同余理论

初等数论的核心知识

目录

不定方程

2.1同余理论的基础知识

2.2二元一次不定方程

2.3特殊的不定方程

同余理论

2.4线性同余方程

2.5乘法逆元

2.6线性同余方程组

2.7*二次同余式与平方剩余

2.8*整数的阶和原根

2.9*离散对数

2.1同余理论的基础概念

2.1同余理论的基础知识

同余：设a、b和m为3个整数，如果a和b除以m的余数相同，即满足

$$a = k_1 \cdot m + r$$

$$b = k_2 \cdot m + r$$

则称a和b对(关于)模m同余，记作$a \equiv b \pmod{m}$，并称该式为同余式

2.1同余理论的基础知识

同余：设a、b和m为3个整数，如果a和b除以m的余数相同，即满足

$$a = k_1 \cdot m + r$$

$$b = k_2 \cdot m + r$$

则称a和b对(关于)模m同余，记作$a \equiv b \pmod{m}$，并称该式为同余式

重要性质：

(i)(自反性)：$a \equiv a \pmod{m}$

(ii)(对称性)：若$a \equiv b \pmod{m}$，则$\Longrightarrow b \equiv a \pmod{m}$

(iii)(传递性)：若$a \equiv b \pmod{m}$，且$b \equiv c \pmod{m}$，则$\Longrightarrow a \equiv c \pmod{m}$

2.1同余理论的基础知识

同余：设a、b和m为3个整数，如果a和b除以m的余数相同，即满足

$$a = k_1 \cdot m + r$$

$$b = k_2 \cdot m + r$$

则称a和b对(关于)模m同余，记作$a \equiv b \pmod{m}$，并称该式为同余式

重要性质：

(i)(自反性)：$a \equiv a \pmod{m}$

(ii)(对称性)：若$a \equiv b \pmod{m}$，则$\Longrightarrow b \equiv a \pmod{m}$

(iii)(传递性)：若$a \equiv b \pmod{m}$，且$b \equiv c \pmod{m}$，则$\Longrightarrow a \equiv c \pmod{m}$

同余关系是一种等价关系，则所有适用于线性运算的基本运算律都适用于模算术

2.1同余理论的基础知识

设a、b、c、d、m和n都是整数，则满足$a \equiv b \pmod{m}$和$c \equiv d \pmod{m}$

模算术：	整数算术：
$a + c \equiv b + c \pmod{m}$	$a + c = b + c$
$a - c \equiv b - c \pmod{m}$	$a - c = b - c$
$a \cdot c \equiv b \cdot c \pmod{m}$	$a \cdot c = b \cdot c$
$a \cdot x + c \cdot y \equiv b \cdot x + d \cdot y \pmod{m}$	$a \cdot x + c \cdot y = b \cdot x + d \cdot y$
$a \cdot c \equiv b \cdot d \pmod{m}$	$a \cdot c = b \cdot d$
$a^n \equiv b^n \pmod{m}$	$a^n = b^n$
$f(a) \equiv f(b) \pmod{m}$, f(x)为一个多项式函数	$f(a) = f(b)$,f(x)为一个多项式函数

2.1同余理论的基础知识

合并性质：若a ≡ b (mod m_i) (1 ≤ i ≤ n)同时成立，当且仅当：

$$a \equiv b \ (\mathrm{mod}\ [m_1, m_2, m_3, \ldots m_n])$$

在后面介绍线性同余方程组合并求解时是会用到

2.1同余理论的基础知识

合并性质：若a ≡ b (mod m_i) (1 ≤ i ≤ n)同时成立，当且仅当：

$$a \equiv b \pmod{[m_1, m_2, m_3, \ldots m_n]}$$

在后面介绍线性同余方程组合并求解时是会用到

剩余类：一个整数可以通过模m进而被分到m个集合中，这些集合称作模m的剩余类，也就是说每个剩余类中的两个数都是模m同余的，其中每个集合中的数都称为该集合的代表元

2.1同余理论的基础知识

合并性质：若a ≡ b (mod m_i) (1 ≤ i ≤ n)同时成立，当且仅当：

$$a \equiv b \pmod{[m_1, m_2, m_3, \ldots m_n]}$$

在后面介绍线性同余方程组合并求解时是会用到

剩余类：一个整数可以通过模m进而被分到m个集合中，这些集合称作模m的剩余类，也就是说每个剩余类中的两个数都是模m同余的，其中每个集合中的数都称为该集合的代表元

完全剩余系：包含每个模m的剩余类中的代表元的集合称作模m的完全剩余系

2.1同余理论的基础知识

合并性质：若$a \equiv b \pmod{m_i}$ $(1 \le i \le n)$同时成立，当且仅当：

$$a \equiv b \pmod{[m_1, m_2, m_3, \ldots m_n]}$$

在后面介绍线性同余方程组合并求解时是会用到

剩余类：一个整数可以通过模m进而被分到m个集合中，这些集合称作模m的剩余类，也就是说每个剩余类中的两个数都是模m同余的，其中每个集合中的数都称为该集合的代表元

完全剩余系：包含每个模m的剩余类中的代表元的集合称作模m的完全剩余系

最小非负完全剩余系：模m的完全剩余系中的{0, 1, 2, …, m - 1}

2.1同余理论的基础知识

合并性质：若a ≡ b (mod m_i) (1 ≤ i ≤ n)同时成立，当且仅当：

$$a \equiv b \pmod{[m_1, m_2, m_3, \ldots m_n]}$$

在后面介绍线性同余方程组合并求解时是会用到

剩余类：一个整数可以通过模m进而被分到m个集合中，这些集合称作模m的剩余类，也就是说每个剩余类中的两个数都是模m同余的，其中每个集合中的数都称为该集合的代表元

完全剩余系：包含每个模m的剩余类中的代表元的集合称作模m的完全剩余系

最小非负完全剩余系：模m的完全剩余系中的{0, 1, 2, …, m - 1}

简化剩余系：从所有与模m互素的剩余类中各取一个数所组成的集合

2.1同余理论的基础知识

合并性质：若a ≡ b (mod m_i) (1 ≤ i ≤ n)同时成立，当且仅当：

$$a \equiv b \pmod{[m_1, m_2, m_3, \ldots m_n]}$$

在后面介绍线性同余方程组合并求解时是会用到

剩余类：一个整数可以通过模m进而被分到m个集合中，这些集合称作模m的剩余类，也就是说每个剩余类中的两个数都是模m同余的，其中每个集合中的数都称为该集合的代表元

完全剩余系：包含每个模m的剩余类中的代表元的集合称作模m的完全剩余系

最小非负完全剩余系：模m的完全剩余系中的{0, 1, 2, ..., m - 1}

简化剩余系：从所有与模m互素的剩余类中各取一个数所组成的集合

易知模m的简化剩余系中的元素个数是φ(m)的

2.1同余理论的基础知识

(1)$a + b \equiv (a \% m + b \% m) \pmod m$

(2)$a - b \equiv (a \% m - b \% m) \pmod m$

(3)$a * b \equiv (a \% m * b \% m) \pmod m$

2.1同余理论的基础知识

(1)$a + b \equiv (a \% m + b \% m) \pmod{m}$

(2)$a - b \equiv (a \% m - b \% m) \pmod{m}$

(3)$a * b \equiv (a \% m * b \% m) \pmod{m}$

$a \div b \not\equiv (a \% m \div b \% m) \pmod{m}$ 切记除法不满足！！！除法的模算术需要使用后面讲的乘法逆元来求

2.1同余理论的基础知识

(1)$a + b \equiv (a \% m + b \% m) \pmod{m}$

(2)$a - b \equiv (a \% m - b \% m) \pmod{m}$

(3)$a * b \equiv (a \% m * b \% m) \pmod{m}$

$a \div b \not\equiv (a \% m \div b \% m) \pmod{m}$ 切记除法不满足！！！除法的模算术需要使用后面讲的乘法逆元来求

推论(重要)：$a \equiv (a + km) \pmod{m}\ k \in Z$

2.1同余理论的基础知识

(1)$a + b \equiv (a \% m + b \% m) \pmod m$

(2)$a - b \equiv (a \% m - b \% m) \pmod m$

(3)$a * b \equiv (a \% m * b \% m) \pmod m$

$a \div b \not\equiv (a \% m \div b \% m) \pmod m$ 切记除法不满足！！！除法的模算术需要使用后面讲的乘法逆元来求

推论(重要)：$a \equiv (a + km) \pmod m \quad k \in Z$

证明：使用(1)性质可得：

$$(a + km) \equiv (a \% m + km \% m) \equiv (a \% m + 0) \equiv a \pmod m$$

2.1同余理论的基础知识

(1)$a + b \equiv (a \% m + b \% m) \pmod{m}$

(2)$a - b \equiv (a \% m - b \% m) \pmod{m}$

(3)$a * b \equiv (a \% m * b \% m) \pmod{m}$

$a \div b \not\equiv (a \% m \div b \% m) \pmod{m}$ 切记除法不满足！！！除法的模算术需要使用后面讲的乘法逆元来求

推论(重要)：$a \equiv (a + km) \pmod{m}$ $k \in Z$

想一想，如何求一个数模上m的最小非负值？

2.1同余理论的基础知识

(1)$a + b \equiv (a \% m + b \% m) \pmod{m}$

(2)$a - b \equiv (a \% m - b \% m) \pmod{m}$

(3)$a * b \equiv (a \% m * b \% m) \pmod{m}$

$a \div b \not\equiv (a \% m \div b \% m) \pmod{m}$ 切记除法不满足！！！除法的模算术需要使用后面讲的乘法逆元来求

推论(重要)：$a \equiv (a + km) \pmod{m}\ k \in Z$

想一想，如何求一个数模上m的最小非负值？

比如：对于$-5 \equiv 3 \pmod{4}$该式，3就是-5在模4的条件下的最小非负值，实际上利用推论可以知道这个值总是存在的，求解方法是令：

$$-5 \equiv -5 + 4k \pmod{4}\ k \in Z$$

2.1同余理论的基础知识

(1)$a + b \equiv (a \% m + b \% m) \pmod m$

(2)$a - b \equiv (a \% m - b \% m) \pmod m$

(3)$a * b \equiv (a \% m * b \% m) \pmod m$

$a \div b \not\equiv (a \% m \div b \% m) \pmod m$ 切记除法不满足！！！除法的模算术需要使用后面讲的乘法逆元来求

推论(重要)：$a \equiv (a + km) \pmod m \quad k \in Z$

想一想，如何求一个数模上m的最小非负值？

比如：对于$-5 \equiv 3 \pmod 4$该式，3就是-5在模4的条件下的最小非负值，实际上利用推论可以知道这个值总是存在的，求解方法是令：

$$-5 \equiv -5 + 4k \pmod 4 \quad k \in Z$$

易知此时当$k = 2$时，得到的值3为最小非负值

2.1同余理论的基础知识

(1)$a + b \equiv (a \% m + b \% m) \pmod m$

(2)$a - b \equiv (a \% m - b \% m) \pmod m$

(3)$a * b \equiv (a \% m * b \% m) \pmod m$

$a \div b \not\equiv (a \% m \div b \% m) \pmod m$ 切记除法不满足！！！除法的模算术需要使用后面讲的乘法逆元来求

推论(重要)：$a \equiv (a + km) \pmod m \quad k \in Z$

想一想，如何求一个数模上m的最小非负值？

如何快速求出最小非负值？

2.1同余理论的基础知识

(1)$a + b \equiv (a \% m + b \% m) \pmod{m}$

(2)$a - b \equiv (a \% m - b \% m) \pmod{m}$

(3)$a * b \equiv (a \% m * b \% m) \pmod{m}$

$a \div b \not\equiv (a \% m \div b \% m) \pmod{m}$ 切记除法不满足！！！除法的模算术需要使用后面讲的乘法逆元来求

推论(重要)：$a \equiv (a + km) \pmod{m}\ k \in Z$

想一想，如何求一个数模上m的最小非负值?

如何快速求出最小非负值?

解析：易知有如下等价式：

$$a \equiv a \% m \equiv (a \% m + km) \pmod{m}$$

2.1同余理论的基础知识

(1)$a + b \equiv (a \% m + b \% m) \pmod{m}$

(2)$a - b \equiv (a \% m - b \% m) \pmod{m}$

(3)$a * b \equiv (a \% m * b \% m) \pmod{m}$

$a \div b \not\equiv (a \% m \div b \% m) \pmod{m}$ 切记除法不满足！！！除法的模算术需要使用后面讲的乘法逆元来求

推论(重要)：$a \equiv (a + km) \pmod{m}\ k \in Z$

想一想，如何求一个数模上m的最小非负值？

如何快速求出最小非负值？

解析：易知有如下等价式：

$$a \equiv a \% m \equiv (a \% m + km) \pmod{m}$$

2.1同余理论的基础知识

(1)$a + b \equiv (a \% m + b \% m) \pmod m$

(2)$a - b \equiv (a \% m - b \% m) \pmod m$

(3)$a * b \equiv (a \% m * b \% m) \pmod m$

$a \div b \not\equiv (a \% m \div b \% m) \pmod m$ 切记除法不满足！！！除法的模算术需要使用后面讲的乘法逆元来求

推论(重要)：$a \equiv (a + km) \pmod m \quad k \in Z$

想一想，如何求一个数模上m的最小非负值？

如何快速求出最小非负值？

解析：易知有如下等价式：

$$a \equiv a \% m \equiv (a \% m + km) \pmod m$$

由于$0 \le |a \% m| < m$，则此时$0 < a \% m + m \le 2m - 1$。即$k = 1$满足非负且最小

2.1同余理论的基础知识

(1)$a + b \equiv (a \% m + b \% m) \pmod m$

(2)$a - b \equiv (a \% m - b \% m) \pmod m$

(3)$a * b \equiv (a \% m * b \% m) \pmod m$

$a \div b \not\equiv (a \% m \div b \% m) \pmod m$ 切记除法不满足！！！除法的模算术需要使用后面讲的乘法逆元来求

推论(重要)：$a \equiv (a + km) \pmod m \quad k \in Z$

以后直接使用同余式$a \equiv (a \% m + m) \pmod m$来求解a模上m的最小非负解，后面介绍的线性同余方程(组)等的通解总是表示成一个特解模上m的形式，有时候题目要求的是最小非负特解，此时就需要求特解模上m的的最小非负值

2.2 二元一次不定方程

2.2二元一次不定方程

不定方程(Indeterminate equation)：指未知元的个数多于方程的个数，且未知元受到某些限制的方程

2.2二元一次不定方程

不定方程(Indeterminate equation)：指未知元的个数多于方程的个数，且未知元受到某些限制的方程

古希腊的丢番图(Diophantus)在公元3世纪开始研究不定方程，今天我们将整系数的不定方程称为丢番图方程，主要探究的是其整数解或有理数解

2.2二元一次不定方程

不定方程(Indeterminate equation)：指未知元的个数多于方程的个数，且未知元受到某些限制的方程

古希腊的丢番图(Diophantus)在公元3世纪开始研究不定方程，今天我们将整系数的不定方程称为丢番图方程，主要探究的是其整数解或有理数解

注意：我们在数论中谈及的不定方程都是丢番图方程，即方程所有的系数都是整数

2.2二元一次不定方程

不定方程(Indeterminate equation)：指未知元的个数多于方程的个数，且未知元受到某些限制的方程

古希腊的丢番图(Diophantus)在公元3世纪开始研究不定方程，今天我们将整系数的不定方程称为丢番图方程，主要探究的是其整数解或有理数解

注意：我们在数论中谈及的不定方程都是丢番图方程，即方程所有的系数都是整数

二元一次不定方程：设a、b和c是整数，且$a \cdot b \neq 0$，则形如：

$$a \cdot x + b \cdot y = c$$

的方程叫做二元一次不定方程

2.2二元一次不定方程

思考：如何求解二元一次不定方程？

2.2二元一次不定方程

思考：如何求解二元一次不定方程？

定理1(证明略)： 对于不定方程$a \cdot x + b \cdot y = c$来说，如果$\gcd(a, b) \mid c$，则该方程存在无数组解，否则方程不存在解

2.2二元一次不定方程

思考：如何求解二元一次不定方程？

定理1(证明略)： 对于不定方程a · x + b · y = c来说，如果gcd (a, b) | c，则该方程存在无数组解，否则方程不存在解

定理2(证明略)：对于满足gcd (a, b) = 1的不定方程a · x + b · y = c，如果有一组特解x_0、y_0，则此方程的所有整数解都可以表示成：

$$\begin{cases} x = x_0 + bt \\ y = y_0 - at \end{cases}$$ 其中t为任意整数

2.2二元一次不定方程

思考：如何求解二元一次不定方程？

定理1(证明略)： 对于不定方程a · x + b · y = c来说，如果gcd (a, b) | c，则该方程存在无数组解，否则方程不存在解

定理2(证明略)：对于满足gcd (a, b) = 1的不定方程a · x + b · y = c，如果有一组特解x_0、y_0，则此方程的所有整数解都可以表示成：

$$\begin{cases} x = x_0 + bt \\ y = y_0 - at \end{cases}$$ 其中t为任意整数

有了定理2，则求解二元一次不定方程的关键就是求出它的一组特解

2.2二元一次不定方程

思考：如何求解二元一次不定方程的一组特解？

2.2二元一次不定方程

思考：如何求解二元一次不定方程的一组特解？

解析：对于不定方程a · x + b · y = c，使用扩展欧几里得算法求出一组解x_0、y_0，也就是该等式的解a · x_0+ b · y_0 = gcd (a, b)。然后判断此时的gcd (a, b) | c是否成立(也即判断方程是否有解)。如果有解，则方程两边同时除上gcd(a, b)，构造互素：

$$a \cdot \frac{x_0}{\gcd(a,b)} + b \cdot \frac{y_0}{\gcd(a,b)} = 1$$

2.2二元一次不定方程

思考：如何求解二元一次不定方程的一组特解？

解析：对于不定方程a · x + b · y = c，使用扩展欧几里得算法求出一组解x_0、y_0，也就是该等式的解a · x_0+ b · y_0 = gcd (a, b)。然后判断此时的gcd (a, b) | c是否成立(也即判断方程是否有解)。如果有解，则方程两边同时除上gcd(a, b)，构造互素：

$$a \cdot \frac{x_0}{\gcd(a,b)} + b \cdot \frac{y_0}{\gcd(a,b)} = 1$$

接着方程两边同时乘上c，构造对应相等的方程：

$$a \cdot \frac{x_0}{\gcd(a,b)} \cdot c + b \cdot \frac{y_0}{\gcd(a,b)} \cdot c = c$$

2.2二元一次不定方程

思考：如何求解二元一次不定方程的一组特解？

解析：对于不定方程a · x + b · y = c，使用扩展欧几里得算法求出一组解x_0、y_0，也就是该等式的解a · x_0+ b · y_0 = gcd (a, b)。然后判断此时的gcd (a, b) | c是否成立(也即判断方程是否有解)。如果有解，则方程两边同时除上gcd(a, b)，构造互素：

$$a \cdot \frac{x_0}{\gcd(a,b)} + b \cdot \frac{y_0}{\gcd(a,b)} = 1$$

接着方程两边同时乘上c，构造对应相等的方程：

$$a \cdot \frac{x_0}{\gcd(a,b)} \cdot c + b \cdot \frac{y_0}{\gcd(a,b)} \cdot c = c$$

则对比原方程a · x + b · y = c和$a \cdot \frac{x_0}{\gcd(a,b)} \cdot c + b \cdot \frac{y_0}{\gcd(a,b)} \cdot c = c$，可得此时原方程的一组特解就是：

$$\begin{cases} x = \dfrac{x_0}{\gcd(a,\ b)} \cdot c \\ y = \dfrac{y_0}{\gcd(a,\ b)} \cdot c \end{cases}$$

2.2二元一次不定方程

通过上面的特解，我们可以将原方程的所有解都表示出来(使用定理2)。但在实际题目中，我们常常需要去求最小非负特解，或者是用最小非负特解来表示其他的整数解，那么如何求最小非负特解？

2.2二元一次不定方程

通过上面的特解，我们可以将原方程的所有解都表示出来(使用定理2)。但在实际题目中，我们常常需要去求最小非负特解，或者是用最小非负特解来表示其他的整数解，那么如何求最小非负特解？

2.2二元一次不定方程

通过上面的特解，我们可以将原方程的所有解都表示出来(使用定理2)。但在实际题目中，我们常常需要去求最小非负特解，或者是用最小非负特解来表示其他的整数解，那么如何求最小非负特解?

解析：直接使用定理2中的结论，设最小非负特解分别是x_p、y_p。则此时有：

$$\begin{cases} x = \dfrac{x_0}{\gcd(a,\ b)} \cdot c = x_p + \dfrac{b}{\gcd(a,\ b)} t \\ y = \dfrac{y_0}{\gcd(a,\ b)} \cdot c = y_p + \dfrac{a}{\gcd(a,\ b)} t \end{cases}$$

2.2二元一次不定方程

通过上面的特解，我们可以将原方程的所有解都表示出来(使用定理2)。但在实际题目中，我们常常需要去求最小非负特解，或者是用最小非负特解来表示其他的整数解，那么如何求最小非负特解?

解析：直接使用定理2中的结论，设最小非负特解分别是x_p、y_p。则此时有：

$$\begin{cases} x = \dfrac{x_0}{\gcd(a,\ b)} \cdot \mathrm{c} = x_p + \dfrac{\mathrm{b}}{\gcd(a,\ b)} t \\ y = \dfrac{y_0}{\gcd(a,\ b)} \cdot \mathrm{c} = y_p + \dfrac{a}{\gcd(a,\ b)} t \end{cases}$$

$$\begin{cases} x_p \equiv x \,\%\, \dfrac{\mathrm{b}}{\gcd(a,\ b)} + \dfrac{\mathrm{b}}{\gcd(a,\ b)} \left(mod\, \dfrac{\mathrm{b}}{\gcd(a,\ b)}\right) \\ y_p \equiv y \,\%\, \dfrac{a}{\gcd(a,\ b)} + \dfrac{a}{\gcd(a,\ b)} \left(mod\, \dfrac{a}{\gcd(a,\ b)}\right) \end{cases}$$

2.2二元一次不定方程

通过上面的特解，我们可以将原方程的所有解都表示出来(使用定理2)。但在实际题目中，我们常常需要去求最小非负特解，或者是用最小非负特解来表示其他的整数解，那么如何求最小非负特解？

解析：直接使用定理2中的结论，设最小非负特解分别是x_p、y_p。则此时有：

$$\begin{cases} x = \dfrac{x_0}{\gcd(a,\ b)} \cdot c = x_p + \dfrac{b}{\gcd(a,\ b)} t \\ y = \dfrac{y_0}{\gcd(a,\ b)} \cdot c = y_p + \dfrac{a}{\gcd(a,\ b)} t \end{cases}$$

$$\begin{cases} x_p \equiv x \% \dfrac{b}{\gcd(a,\ b)} + \dfrac{b}{\gcd(a,\ b)} \left(mod \dfrac{b}{\gcd(a,\ b)}\right) \\ y_p \equiv y \% \dfrac{a}{\gcd(a,\ b)} + \dfrac{a}{\gcd(a,\ b)} \left(mod \dfrac{a}{\gcd(a,\ b)}\right) \end{cases}$$

注意：此时的x_p和y_p分别是原方程中x、y的最小非负特解，两者是独立求解的。即意味着$a \cdot x_p + b \cdot y_p = c$不一定成立！！！但是只其一可以求出另一个

2.2二元一次不定方程

```
long long gcd_ex (long long a, long long b, long long &x, long long &y)
{
    if (b == 0) { x = 1; y = 0; return a; }
    long long d = gcd_ex (b, a % b, y, x);
    y = y - a / b * x;
    return d;
}

long long mod_equ (long long a, long long b, long long c)
{
    long long x, y;
    long long d = gcd_ex (a, b, x, y);

    if (c % d)
        return -1;

    x = (x * c / d) % (b / d);
    if (x < 0)
        x += b / d;

/*    y = (y * c / d) % (a / d);
    if (y < 0)
        y += a / d;
*/
    return x;
}
```

注意：这里mod_equ函数用来求解不定方程，如果方程不存在解，则函数返回-1。如果方程存在解，则返回x的最小非负特解

2.2二元一次不定方程

(POJ2142)题目大意：现有一个天平、质量为a和b两种砝码。已知两种砝码的数量不限且天平左右都可以放砝码，现要求天平上称出质量为c的物品并给出一个方案。这个方案需满足：放置的砝码应尽量少；当放置的砝码的数量相同时，砝码的总质量应尽量少$(0 < a, b \leq 10000, a \neq b, c \leq 50000)$

2.2二元一次不定方程

(POJ2142)题目大意：现有一个天平、质量为a和b两种砝码。已知两种砝码的数量不限且天平左右都可以放砝码，现要求天平上称出质量为c的物品并给出一个方案。这个方案需满足：放置的砝码应尽量少；当放置的砝码的数量相同时，砝码的总质量应尽量少$(0 < a, b \le 10000, a \ne b, c \le 50000)$

解析：其实就是求解$a \cdot x + b \cdot y = c$的一个整数解，要满足$|x| + |y|$的和尽量小，而且如果相等的话，则要求$a|x| + b|y|$要尽量小。此时假设$a > b$(如果不满足就交换两个值)，则$|x| + |y| = \left|x_0 + \frac{b}{gcd}t\right| + \left|y_0 - \frac{a}{gcd}t\right|$，此时已知$\left|x_0 + \frac{b}{gcd}t\right|$是递增的，而$\left|y_0 - \frac{a}{gcd}t\right|$是先递减后递增的，且斜率$\frac{a}{gcd} > \frac{b}{gcd}$，则$|x| + |y|$是先递减后递增的，可知一定会在$y_0 - \frac{a}{gcd}t = 0$的t附近取得最小值，然后简单枚举相邻的几个整数即可

2.3特殊的不定方程

2.3特殊的不定方程-费马大定理

费马大定理：三元n次不定方程$x^n + y^n = z^n$没有非零的整数解，其中$n \in N^*$, $n \geq 3$

2.3特殊的不定方程-费马大定理

费马大定理：三元n次不定方程$x^n + y^n = z^n$没有非零的整数解，其中$n \in N^*$, $n \geq 3$

费马大定理作为数论中的一个十分棘手的问题遗留上百年，然而该定理于1994年被美国普林斯顿大学的英国数学家安德鲁·怀尔斯证明

2.3特殊的不定方程-毕达哥拉斯三元组

毕达哥拉斯定理(勾股定理)：直角三角形的的两个直角边的平方和等于斜边的平方和，即直角三角形的三条边满足一个三元二次不定方程：

$$x^2 + y^2 = z^2$$

2.3特殊的不定方程-毕达哥拉斯三元组

毕达哥拉斯定理(勾股定理)：直角三角形的的两个直角边的平方和等于斜边的平方和，即直角三角形的三条边满足一个三元二次不定方程：

$$x^2 + y^2 = z^2$$

满足上述不定方程的，由正整数x、y和z所构成的三元组叫做毕达哥拉斯三元组，记作triples(x, y, z)

2.3特殊的不定方程-毕达哥拉斯三元组

毕达哥拉斯定理(勾股定理)：直角三角形的的两个直角边的平方和等于斜边的平方和，即直角三角形的三条边满足一个三元二次不定方程：

$$x^2 + y^2 = z^2$$

满足上述不定方程的，由正整数x、y和z所构成的三元组叫做毕达哥拉斯三元组，记作triples(x, y, z)

本原毕达哥拉斯三元组：如果一个毕达哥拉斯三元组triples(x, y, z)满足gcd(x, y, z) = 1，则称该三元组为本原三元组

2.3特殊的不定方程-毕达哥拉斯三元组

如何去构造一个本原毕达哥拉斯三元组?

2.3特殊的不定方程-毕达哥拉斯三元组

如何去构造一个本原毕达哥拉斯三元组？

定理：如果正整数x、y和z可以构成一个本原毕达哥拉斯三元组，当且仅当满足y为偶数，且存在两个正整数m和n互素(m > n)，m和n奇偶互异，则按照如下构造：

$$\begin{cases} x = m^2 - n^2 \\ y = 2 \cdot m \cdot n \\ z = m^2 + n^2 \end{cases}$$

此时的本原毕达哥拉斯三元组为triples (x, y, z)

2.3特殊的不定方程-毕达哥拉斯三元组

(POJ1350)题目大意：求出1~N范围内的本原毕达哥拉斯三元组的数量，以及N以内不含有毕达哥拉斯三元组的数的个数(N <= 1e6)

2.3特殊的不定方程-毕达哥拉斯三元组

(POJ1350)题目大意：求出1~N范围内的本原毕达哥拉斯三元组的数量，以及N以内不含有毕达哥拉斯三元组的数的个数(N <= 1e6)

解析：直接暴力的方法对于1e6规模的数据肯定会TLE的

2.3特殊的不定方程-毕达哥拉斯三元组

(POJ1350)题目大意：求出1~N范围内的本原毕达哥拉斯三元组的数量，以及N以内不含有毕达哥拉斯三元组的数的个数(N <= 1e6)

解析：由于N只有1e6，则可以从小到大分别枚举n和m。此时m最多枚举到$\sqrt{N}$，然后判断两者是否互素且奇偶互异。满足条件，则本原的三元组数量自加1，并将三元组每个分量都倍增来构造所有的毕达哥拉斯三元组，复杂度是O (K· N)的

2.3特殊的不定方程-佩尔方程(Pell)

佩尔方程：设一个正整数d，满足d > 1，且d不为完全平方数，则形如：

$$x^2 - dy^2 = 1$$

的不定方程叫做佩尔方程，佩尔方程一定有无穷组正整数解

2.3特殊的不定方程-佩尔方程(Pell)

佩尔方程：设一个正整数d，满足d > 1，且d不为完全平方数，则形如：

$$x^2 - dy^2 = 1$$

的不定方程叫做佩尔方程，佩尔方程一定有无穷组正整数解

若佩尔方程的$x^2 - dy^2 = 1$的最小非负特解是(x_1, y_1)，则该方程的所有解(x_n, y_n)可由递推公式得到：

$$\begin{cases} x_n = x_{n-1}x_1 + dy_{n-1}y_1 \\ y_n = x_{n-1}y_1 + y_{n-1}x_1 \end{cases}$$

2.3特殊的不定方程-佩尔方程(Pell)

佩尔方程：设一个正整数d，满足d > 1，且d不为完全平方数，则形如：

$$x^2 - dy^2 = 1$$

的不定方程叫做佩尔方程，佩尔方程一定有无穷组正整数解

若佩尔方程的$x^2 - dy^2 = 1$的最小非负特解是(x_1, y_1)，则该方程的所有解(x_n, y_n)可由递推公式得到：

$$\begin{cases} x_n = x_{n-1}x_1 + dy_{n-1}y_1 \\ y_n = x_{n-1}y_1 + y_{n-1}x_1 \end{cases}$$

为了加速求解(矩阵快速幂)，常写成矩阵形式：

$$\begin{pmatrix} x_n \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 & dy_1 \\ y_1 & x_1 \end{pmatrix}^{n-1} \begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$$

2.3特殊的不定方程-佩尔方程(Pell)

由前面可知，若最小非负特解已经求出，则可以使用矩阵快速幂来快速求解任意一个解，那么如何求出最小非负特解呢?

2.3特殊的不定方程-佩尔方程(Pell)

由前面可知，若最小非负特解已经求出，则可以使用矩阵快速幂来快速求解任意一个解，那么如何求出最小非负特解呢？

解析：介绍两种常用的方法：暴力枚举法和连分数法

2.3特殊的不定方程-佩尔方程(Pell)

由前面可知，若最小非负特解已经求出，则可以使用矩阵快速幂来快速求解任意一个解，那么如何求出最小非负特解呢？

解析：介绍两种常用的方法：暴力枚举法和连分数法

1.暴力枚举法：由于一般情况下佩尔方程的最小非负特解不是特别大，所以可以从小到大枚举。具体做法是：

2.3特殊的不定方程-佩尔方程(Pell)

由前面可知，若最小非负特解已经求出，则可以使用矩阵快速幂来快速求解任意一个解，那么如何求出最小非负特解呢？

解析：介绍两种常用的方法：暴力枚举法和连分数法

1.暴力枚举法：由于一般情况下佩尔方程的最小非负特解不是特别大，所以可以从小到大枚举。具体做法是：

(i) 由$x^2 - dy^2 = 1 \Rightarrow x = \sqrt{dy^2 + 1}$

(ii) 令y = 1，并使用上面的公式求出x，并检验此时是否满足$x^2 - dy^2 = 1$。如果满足，则此时的x、y就是最小非负特解，否则y自加1，继续判断

2.3特殊的不定方程-佩尔方程(Pell)

由前面可知，若最小非负特解已经求出，则可以使用矩阵快速幂来快速求解任意一个解，那么如何求出最小非负特解呢?

解析：介绍两种常用的方法：暴力枚举法和连分数法

2.连分数法

2.3特殊的不定方程-佩尔方程(Pell)

(POJ1320)题目大意：求出两个不等的正整数n、m(n < m)，使得

$$1 + 2 + \ldots + n = (n + 1) + (n + 2) + \ldots m$$

并从小到大输出前十组的n和m

2.3特殊的不定方程-佩尔方程(Pell)

(POJ1320)题目大意：求出两个不等的正整数n、m(n < m)，使得

$$1 + 2 + \ldots + n = (n + 1) + (n + 2) + \ldots m$$

并从小到大输出前十组的n和m

解析：上面的等式可以化简得$\frac{(1+n)n}{2} = \frac{(m-n)(m+n+1)}{2}$，最终化简得$(2m+1)^2 - 8n^2 = 1$，令x = $(2m+1)^2$，y = n，则原式等于

$$x^2 - 8y^2 = 1$$

2.3特殊的不定方程-佩尔方程(Pell)

(POJ1320)题目大意：求出两个不等的正整数n、m(n < m)，使得

$$1 + 2 + \ldots + n = (n + 1) + (n + 2) + \ldots m$$

并从小到大输出前十组的n和m

解析：上面的等式可以化简得$\frac{(1+n)n}{2} = \frac{(m-n)(m+n+1)}{2}$，最终化简得$(2m+1)^2 - 8n^2 = 1$，令$x = (2m+1)^2$，$y = n$，则原式等于

$$x^2 - 8y^2 = 1$$

直接暴力求解出最小非负特解后，通过递推公式构造并输出前十组解即可

2.4线性同余方程

2.4线性同余方程

一元线性同余方程： 设a、b和m都是整数，$m > 0$，则形如：

$$a \cdot x \equiv b \pmod{m},\ x \in N^*$$

的**同余式**叫做**一元线性同余方程**

2.4线性同余方程

一元线性同余方程：设a、b和m都是整数，$m>0$，则形如：

$$a \cdot x \equiv b \pmod{m},\ x \in N^*$$

的同余式叫做一元线性同余方程

定理(证明略)：如果一元线性同余方程满足gcd (a, m) | b，则方程恰好有gcd (a, m)个模m不同余的解，否则方程无解

2.4线性同余方程

一元线性同余方程：设a、b和m都是整数，m > 0，则形如：

$$a \cdot x \equiv b \pmod{m},\ x \in N^*$$

的同余式叫做一元线性同余方程

定理(证明略)：如果一元线性同余方程满足gcd (a, m) | b，则方程恰好有gcd (a, m)个模m不同余的解，否则方程无解

由同余式的定义可知$a \cdot x \equiv b \pmod{m}$满足

$$a \cdot x = k_1 \cdot m + r \quad (1)$$

$$b = k_2 \cdot m + r \quad (2)$$

2.4线性同余方程

一元线性同余方程：设a、b和m都是整数，m > 0，则形如：

$$a \cdot x \equiv b \pmod{m},\ x \in N^*$$

的同余式叫做一元线性同余方程

定理(证明略)：如果一元线性同余方程满足gcd (a, m) | b，则方程恰好有gcd (a, m)个模m不同余的解，否则方程无解

由同余式的定义可知$a \cdot x \equiv b \pmod{m}$满足

$$a \cdot x = k_1 \cdot m + r \quad (1)$$

$$b = k_2 \cdot m + r \quad (2)$$

(1)+(2)式得：$a \cdot x + b = (k_1 + k_2) \cdot m + 2 \cdot r \Longrightarrow a \cdot x - m \cdot (k_1 + k_2) = 2 \cdot r - b$

此时令$y = -(k_1 + k_2)$，$c = 2 \cdot r - b$，则线性同余方程最终转化成二元一次不定方程：

$$a \cdot x + m \cdot y = c$$

2.4线性同余方程

一元线性同余方程：设a、b和m都是整数，m > 0，则形如：

$$a \cdot x \equiv b \pmod{m},\ x \in N^*$$

的同余式叫做一元线性同余方程

定理(证明略)：如果一元线性同余方程满足gcd (a, m) | b，则方程恰好有gcd (a, m)个模m不同余的解，否则方程无解

由同余式的定义可知$a \cdot x \equiv b \pmod{m}$满足

$$a \cdot x = k_1 \cdot m + r \quad (1)$$

$$b = k_2 \cdot m + r \quad (2)$$

则直接使用前面求解二元一次不定方程的方法求解即可

2.4线性同余方程

(POJ1061)题目大意：有两个青蛙，在一个首尾相连的数轴(环)上同向行走，青蛙A的起始坐标是x，速度是m，青蛙B的起始坐标是y，速度是n，环的总长度的是L，现问多久之后两个青蛙会见面？其中$0 < x, y, m, n, L < 2e9$，且$x \neq y$

2.4线性同余方程

(POJ1061)题目大意：有两个青蛙，在一个首尾相连的数轴(环)上同向行走，青蛙A的起始坐标是x，速度是m，青蛙B的起始坐标是y，速度是n，环的总长度的是L，现问多久之后两个青蛙会见面？其中$0 < x, y, m, n, L < 2e9$，且$x \neq y$

解析：设所用时间为t，则当A的坐标等于B的坐标时，两者相遇，即满足：

$$x + mt = y + nt + kL$$

如果这个二元一次不定方程有解的话，输出最小的t即可，否则输出impossible

2.4线性同余方程

(ZOJ3593)题目大意：在一个数轴上给出一个起点A和终点B，每次有6种可以走的步数：向左或向右走a步，向左或向右走b步，向左或向右走(a + b)步。问最小的步数，如果无解，则直接输出-1，($-2^{31} < A，B < 2^{31}，0 < a，b < 2^{31}$)

2.4线性同余方程

(ZOJ3593)题目大意：在一个数轴上给出一个起点A和终点B，每次有6种可以走的步数：向左或向右走a步，向左或向右走b步，向左或向右走(a + b)步。问最小的步数，如果无解，则直接输出-1，($-2^{31} < A$，$B < 2^{31}$，$0 < a$，$b < 2^{31}$)

解析：显然不同的操作之间的先后顺序对最后的结果没有影响，此时设走a步这个操作进行了x次(x为负表示向左走了$|x|$步)，同理可以设走b步这个操作进行了y次。则问题就转换成为求ax + by = $|B - A|$的解，使得步数最少。此时如果x、y异号，则结果就是$|x| + |y|$。如果x、y同号，则结果为max ($|x|, |y|$)，因为此时可以将同号的a和b操作合并成c操作。此时将$|x|$和$|y|$之间的差值逐步缩小(会使得两种情况下的值都变小)，找到的最小差值时的$|x|$和$|y|$就是最优值

2.5 乘法逆元

2.5乘法逆元

乘法逆元：设a、m都是整数，$m > 0$，则形如：

$$a \cdot x \equiv 1 \pmod{m},\ x \in N^*$$

的同余式中的一个解x叫做a模m的乘法逆元，常记作$a \equiv x^{-1} \pmod{m}$

2.5乘法逆元

乘法逆元：设a、m都是整数，$m > 0$，则形如：

$$a \cdot x \equiv 1 \pmod m, x \in N^*$$

的同余式中的一个解x叫做a模m的乘法逆元，常记作$a \equiv x^{-1} \pmod m$

如何求解乘法逆元?

简单来说就是求线性同余方程$a \cdot x \equiv b \pmod m$ $b = 1$时的解x，使用前面所讲的知识求解即可

2.5乘法逆元

为什么要引入乘法逆元呢？

2.5乘法逆元

为什么要引入乘法逆元呢？

当我们求解$\frac{a}{b}$ % p (b | a)时，往往a是很大的，无法直接求出$\frac{a}{b}$再取模，这里就要用到乘法逆元，有：

$$\frac{a}{b} \equiv a \cdot k \pmod{p}，其中 k \equiv b^{-1} \pmod{p}$$

2.5乘法逆元

为什么要引入乘法逆元呢?

当我们求解$\frac{a}{b}$ % p (b | a)时，往往a是很大的，无法直接求出$\frac{a}{b}$再取模，这里就要用到乘法逆元，有：

$$\frac{a}{b} \equiv a \cdot k \pmod{p}，其中k \equiv b^{-1} \pmod{p}$$

证明比较简单，直接将$k \equiv b^{-1} \pmod{p}$的值代入到 $a \cdot k \% p$化简看最后能不能整除$\frac{a}{b}$即可

2.5乘法逆元

为什么要引入乘法逆元呢?

当我们求解$\frac{a}{b}$ % p (b | a)时，往往a是很大的，无法直接求出$\frac{a}{b}$再取模，这里就要用到乘法逆元，有：

$$\frac{a}{b} \equiv \mathrm{a} \cdot \mathrm{k} \pmod{\mathrm{p}}，其中\mathrm{k} \equiv b^{-1} \pmod{\mathrm{p}}$$

证明比较简单，直接将$\mathrm{k} \equiv b^{-1} \pmod{\mathrm{p}}$的值代入到 $\mathrm{a} \cdot \mathrm{k}$ % p化简看最后能不能整除$\frac{a}{b}$即可

乘法逆元常常是求解其他数论问题的中间步骤，直接求解乘法逆元的题目基本上都属于简单题

2.6线性同余方程组

2.6线性同余方程组

一元线性同余方程组： 设a_i和m_i $(1 \le i \le k)$均为整数，则形如：

$$x \equiv a_1 \pmod{m_1}$$

$$x \equiv a_2 \pmod{m_2}$$

$$\cdots$$

$$x \equiv a_k \pmod{m_k}$$

的同余式组叫做**一元线性同余方程组**

2.6线性同余方程组

一元线性同余方程组：设a_i和m_i $(1 \leq \mathrm{i} \leq \mathrm{k})$均为整数，则形如：

$$\mathrm{x} \equiv a_1 \ (\mathrm{mod}\ m_1)$$
$$\mathrm{x} \equiv a_2 \ (\mathrm{mod}\ m_2)$$
$$\cdots$$
$$\mathrm{x} \equiv a_k \ (\mathrm{mod}\ m_k)$$

的同余式组叫做一元线性同余方程组

一元线性同余方程组可以看作是多个一元线性同余方程的联立

2.6线性同余方程组

一元线性同余方程组：设a_i和m_i $(1 \le i \le k)$均为整数，则形如：

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$
$$\cdots$$
$$x \equiv a_k \pmod{m_k}$$

的同余式组叫做一元线性同余方程组

一元线性同余方程组可以看作是多个一元线性同余方程的联立

想想该如何求解？

2.6线性同余方程组

一元线性同余方程组：设a_i和m_i $(1 \le \mathrm{i} \le \mathrm{k})$均为整数，则形如：

$$x \equiv a_1 \ (\mathrm{mod}\ m_1)$$
$$x \equiv a_2 \ (\mathrm{mod}\ m_2)$$
$$\cdots$$
$$x \equiv a_k \ (\mathrm{mod}\ m_k)$$

的同余式组叫做一元线性同余方程组

一元线性同余方程组可以看作是多个一元线性同余方程的联立

想想该如何求解？

如果m_1、m_2、$m_3 \ldots m_k$两两互素，则可以使用后面介绍的中国剩余定理求解

2.6线性同余方程组

中国剩余定理(Chinese Remainder Theorem, CRT)：又称孙子定理，先设m_1、m_2、m_3...m_k两两互素，则同余方程组：

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$
$$\cdots$$
$$x \equiv a_k \pmod{m_k}$$

有整数解，并且在模$M = m_1 \cdot m_2 \cdot m_3 \cdot \ldots \cdot m_k$下的解是唯一的，解为：

$$x \equiv (a_1 M_1 {M_1}^{-1} + a_2 M_2 {M_2}^{-1} + \ldots + a_k M_k {M_k}^{-1}) \pmod{M}$$

其中$M_i = \frac{M}{m_i}$，${M_i}^{-1}$为M_i模上m_i的乘法逆元

2.6线性同余方程组

中国剩余定理(Chinese Remainder Theorem, CRT)：又称孙子定理，先设m_1、m_2、m_3...m_k两两互素，则同余方程组：

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$
$$\cdots$$
$$x \equiv a_k \pmod{m_k}$$

有整数解，并且在模M = $m_1 \cdot m_2 \cdot m_3 \cdot \ldots \cdot m_k$下的解是唯一的，解为：

$$x \equiv (a_1 M_1 {M_1}^{-1} + a_2 M_2 {M_2}^{-1} + \ldots + a_k M_k {M_k}^{-1}) \pmod{M}$$

其中$M_i = \frac{M}{m_i}$，${M_i}^{-1}$为M_i模上m_i的乘法逆元

中国剩余定理(CRT)直接给出了模值两两互素的线性同余方程组有解的判定条件，并同时给出了有解情况下解的具体形式

2.6线性同余方程组

```
//CRT中方程组的组数、剩余值和模值
long long n, a[maxn+10], m[maxn+10];

long long gcd_ex (long long a, long long b, long long &x, long long &y)
{
    if (b == 0) { x = 1; y = 0; return a; }
    long long d = gcd_ex (b, a % b, y, x);
    y = y - a / b * x;
    return d;
}

long long CRT ()
{
    long long M = 1;
    long long Mi, x, y, d, res = 0;
    for (long long i = 1; i <= n; i++)
        M *= m[i];

    for (long long i = 1; i <= n; i++)
    {
        Mi = M / m[i];
        d = gcd_ex (Mi, m[i], x, y);
        res = (res + a[i] * Mi * x) % M;
    }
    res = (res % M + M) % M;
    return res;
}
```

解析：由于Mi和m[i]互素(因为Mi = M / m[i])，则在使用扩展欧几里得求乘法逆元时不需要判断1 % d == 0是否满足。

一般我们要求解的是最小非负解res，所以如果res % M < 0，则将其加上M，也就是(res % M + M) % M

2.6线性同余方程组

(POJ1006)题目大意：人体有三个周期：身体周期、情感周期和智力周期。长度分别为23天、28天和33天。人体的最佳状态日定义为身体周期、情感周期和智力周期同时出现的日子。现在给出各自周期在最佳状态日之后经过的天数a、b和c。再给出年内已经经过的天数d，现在求d天所代表的日期之后的多少天，又一次出现最佳状态日

2.6线性同余方程组

(POJ1006)题目大意：人体有三个周期：身体周期、情感周期和智力周期。长度分别为23天、28天和33天。人体的最佳状态日定义为身体周期、情感周期和智力周期同时出现的日子。现在给出各自周期在最佳状态日之后经过的天数a、b和c。再给出年内已经经过的天数d，现在求d天所代表的日期之后的多少天，又一次出现最佳状态日

解析：设最佳状态日的日期为x，则由题意可得同余方程组：

$$x \equiv a \pmod{23}$$

$$x \equiv b \pmod{28}$$

$$x \equiv c \pmod{33}$$

由于模值23、28和33两两互素，则直接使用中国剩余定理求解即可

2.6线性同余方程组

模值两两互素的线性同余方程组问题最早出现于南北朝时期的《孙子算经》中，其全文为：有物不知其数，三三数之剩二，五五数之剩三，七七数之剩二。问物几何？

2.6线性同余方程组

模值两两互素的线性同余方程组问题最早出现于南北朝时期的《孙子算经》中，其全文为：有物不知其数，三三数之剩二，五五数之剩三，七七数之剩二。问物几何？

南宋数学家秦九韶于《数书九章》“大衍求一术”中对“物不知数”问题给出了系统完整的解答，比西方数学家高斯早了554年。被国际数学界称为中国剩余定理，该定理代表了中世纪世界数学的最高成就，也是初等数论四大定理之一(另外三个是威尔逊定理、费马小定理和欧拉定理)

2.6线性同余方程组

由前面可知，中国剩余定理只能求解模值两两互素的线性同余方程组，那么对于模值不是两两互素的线性同余方程组，该如何计算？

2.6线性同余方程组

由前面可知，中国剩余定理只能求解模值两两互素的线性同余方程组，那么对于模值不是两两互素的线性同余方程组，该如何计算？

使用两两合并的思想，对于两个线性同余方程来说：

$$x \equiv a_1 \pmod{m_1} \Longrightarrow x = a_1 + m_1 k_1 \quad (1)$$

$$x \equiv a_2 \pmod{m_2} \Longrightarrow x = a_2 + m_2 k_2 \quad (2)$$

2.6线性同余方程组

由前面可知，中国剩余定理只能求解模值两两互素的线性同余方程组，那么对于模值不是两两互素的线性同余方程组，该如何计算？

使用两两合并的思想，对于两个线性同余方程来说：

$$x \equiv a_1 \pmod{m_1} \Longrightarrow x = a_1 + m_1k_1 \quad (1)$$

$$x \equiv a_2 \pmod{m_2} \Longrightarrow x = a_2 + m_2k_2 \quad (2)$$

联立消去x，化简得$m_1k_1 + m_2k_2 = a_2 - a_1$，这是二元一次不定方程，使用扩展欧几里得求出$k_1$的最小非负解后，将$k_1$代入到(1)式可以求出此时的x值，记作$x_s$。将得到的方程化成同余方程：

$$x_s \equiv a_1 \pmod{m_1} \equiv x \pmod{m_1}$$

$$x_s \equiv a_2 \pmod{m_2} \equiv x \pmod{m_2}$$

2.6线性同余方程组

由前面可知，中国剩余定理只能求解模值两两互素的线性同余方程组，那么对于模值不是两两互素的线性同余方程组，该如何计算？

使用两两合并的思想，对于两个线性同余方程来说：

$$x \equiv a_1 \pmod{m_1} \Longrightarrow x = a_1 + m_1 k_1 \quad (1)$$

$$x \equiv a_2 \pmod{m_2} \Longrightarrow x = a_2 + m_2 k_2 \quad (2)$$

联立消去x，化简得$m_1 k_1 + m_2 k_2 = a_2 - a_1$，这是二元一次不定方程，使用扩展欧几里得求出$k_1$的最小非负解后，将$k_1$代入到(1)式可以求出此时的x值，记作$x_s$。将得到的方程化成同余方程：

$$x_s \equiv a_1 \pmod{m_1} \equiv x \pmod{m_1}$$

$$x_s \equiv a_2 \pmod{m_2} \equiv x \pmod{m_2}$$

由同余的合并性质可得：$x \equiv x_s \pmod{[m_1, m_2]}$

2.6线性同余方程组

由前面可知，中国剩余定理只能求解模值两两互素的线性同余方程组，那么对于模值不是两两互素的线性同余方程组，该如何计算？

使用两两合并的思想，对于两个线性同余方程来说：

$$x \equiv a_1 \pmod{m_1} \Longrightarrow x = a_1 + m_1k_1 \quad (1)$$

$$x \equiv a_2 \pmod{m_2} \Longrightarrow x = a_2 + m_2k_2 \quad (2)$$

联立消去x，化简得$m_1k_1 + m_2k_2 = a_2 - a_1$，这是二元一次不定方程，使用扩展欧几里得求出$k_1$的最小非负解后，将$k_1$代入到(1)式可以求出此时的x值，记作$x_s$。将得到的方程化成同余方程：

$$x \equiv x_s \pmod{[m_1, m_2]}$$

如果还有同余方程，则按照上面的步骤继续合并，直到只剩下一个同余方程，此时直接使用前面求解同余方程的方法计算即可

2.6线性同余方程组

```
//n表示方程组的组数，a[i], m[i]分别表示x = ai (mod mi)中ai和mi的值
long long n, m[maxn+10], a[maxn+10];

long long gcd_ex (long long a, long long b, long long &x, long long &y)
{
    if (b == 0) { x = 1; y = 0; return a; }
    long long d = gcd_ex (b, a % b, y, x);
    y = y - a / b * x;
    return d;
}
long long mod_equ_set ()
{
    long long M = m[1], A = a[1], x, y, d;
    for (long long i = 2; i <= n; i++)
    {
        d = gcd_ex (M, m[i], x, y);

        if ((a[i] - A) % d) return -1;

        x = (a[i] - A) / d * x % (m[i] / d);
        x = (x + (m[i] / d)) % (m[i] / d);
        A += x * M;
        M = M * m[i] / d;
        A %= M;
    }
    return (A + M) % M;
}
```

解析：如果合并后得到的不定方程没有解，则直接返回-1。

注意使用扩展欧几里得求出的X可能是负数，回代会造成解的大小没法控制，所以需要求出X的最小非负解

2.6线性同余方程组

注意我们前面介绍的求解的线性同余方程的形式如下：

$$
\begin{gathered}
x \equiv a_1 \pmod{m_1} \\
x \equiv a_2 \pmod{m_2} \\
\cdots \\
x \equiv a_k \pmod{m_k}
\end{gathered}
$$

2.6线性同余方程组

注意我们前面介绍的求解的线性同余方程的形式如下：

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$
$$\cdots$$
$$x \equiv a_k \pmod{m_k}$$

上面介绍的方法不能直接来求解形式如下的线性同余方程组：

$$a_1 \cdot x \equiv b_1 \pmod{m_1}$$
$$a_2 \cdot x \equiv b_2 \pmod{m_2}$$
$$\cdots$$
$$a_k \cdot x \equiv b_k \pmod{m_k}$$

2.6线性同余方程组

注意我们前面介绍的求解的线性同余方程的形式如下：

$$x \equiv a_1 \pmod{m_1}$$
$$x \equiv a_2 \pmod{m_2}$$
$$\cdots$$
$$x \equiv a_k \pmod{m_k}$$

上面介绍的方法不能直接来求解形式如下的线性同余方程组：

$$a_1 \cdot x \equiv b_1 \pmod{m_1}$$
$$a_2 \cdot x \equiv b_2 \pmod{m_2}$$
$$\cdots$$
$$a_k \cdot x \equiv b_k \pmod{m_k}$$

想要求解必须求a_i关于m_i的乘法逆元，将方程转换成前面介绍的标准形式

*2.7 二次同余式与平方剩余

2.7二次同余式与平方剩余

2.7二次同余式与平方剩余

二次同余式：设a、b、c和m都是整数，则形如：

$$ax^2 + bx + c \equiv 0 \pmod{m}$$

的同余式叫做**二次同余式**，上式总可以转化成$x^2 \equiv a \pmod{m}$的形式

2.7二次同余式与平方剩余

二次同余式：设a、b、c和m都是整数，则形如：

$$ax^2 + bx + c \equiv 0 \pmod{m}$$

的同余式叫做二次同余式，上式总可以转化成$x^2 \equiv a \pmod{m}$的形式

平方剩余：假设a、m是整数(m > 0)，且gcd (a, m) = 1。若二次同余式

$$x^2 \equiv a \pmod{m}$$

有解，那么称a是模m的平方剩余。若无解，则称a是模m的平方非剩余

2.7二次同余式与平方剩余

欧拉判定条件：设p是奇素数，若gcd (a, p) = 1，则

(i)a是模p的平方剩余的充要条件是

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

(ii)a是模p的平方非剩余的充要条件是

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

2.7二次同余式与平方剩余

欧拉判定条件：设p是奇素数，若gcd (a, p) = 1，则

(i)a是模p的平方剩余的充要条件是

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

(ii)a是模p的平方非剩余的充要条件是

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

勒让德符号：设p是一个给定的奇素数，对于整数a定义勒让德符号：

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & a\text{是平方剩余} \\ -1 & a\text{是平方非剩余} \\ 0 & p \mid a \end{cases}$$

2.7二次同余式与平方剩余

勒让德符号的性质：

(1) $\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}}$ (mod p)

(2) a ≡ b (mod p) $\Longrightarrow \left(\frac{a}{p}\right) = \left(\frac{b}{p}\right)$

(3) $\left(\frac{a_1 a_2 \ldots a_k}{p}\right) = \left(\frac{a_1}{p}\right)\left(\frac{a_2}{p}\right)\ldots\left(\frac{a_k}{p}\right)$

(4) $\left(\frac{1}{p}\right) = 1$

(5) $\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}$

(6) $\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$

二次互反律：设p、q分别是不同的奇素数，则有：

$$\left(\frac{p}{q}\right) = (-1)^{\frac{p-1}{2}\cdot\frac{q-1}{2}}\left(\frac{q}{p}\right)$$

2.7二次同余式与平方剩余

雅可比符号：给定正奇数m = $p_1p_2...p_k$，对任意的整数a定义

$$\left(\frac{a}{m}\right)=\left(\frac{a}{p_1}\right)\left(\frac{a}{p_2}\right)...\left(\frac{a}{p_k}\right)$$

则称$\left(\frac{a}{m}\right)$为雅可比符号

2.7二次同余式与平方剩余

雅可比符号：给定正奇数m = $p_1 p_2 \ldots p_k$，对任意的整数a定义

$$\left(\frac{a}{m}\right) = \left(\frac{a}{p_1}\right)\left(\frac{a}{p_2}\right)\ldots\left(\frac{a}{p_k}\right)$$

则称$\left(\frac{a}{m}\right)$为雅可比符号

雅可比符号是勒让德符号的推广，由于使用勒让德符号需要模上奇素数，而雅可比符号将一个正整数分解成素数乘积，然后对于每个素数(其实奇素数，因为分解的是正奇数)使用勒让德符号计算，最后将每个结果相乘就是对应的雅可比符号的值

雅可比符号的运算性质和勒让德符号的基本相同

*2.8 整数的阶与原根

2.8整数的阶与原根

整数的阶：设a、m都是整数，且m ≥ 1，gcd (a, m) = 1，则使得：

$$a^x \equiv 1 \pmod{m}$$

成立的最小正整数x叫做a模m的阶(指数)，记作ord_m(a)

2.8整数的阶与原根

整数的阶：设a、m都是整数，且m ≥ 1，gcd (a, m) = 1，则使得：

$$a^x \equiv 1 \pmod{m}$$

成立的最小正整数x叫做a模m的阶(指数)，记作ord_m(a)

原根：若ord_m(a) = φ(m)，则将a叫做模m的一个原根，φ(m)表示在m处的欧拉函数值

原根个数定理：若模m的原根存在，则它一共有$\varphi(\varphi(m))$个原根

原根存在定理：若p是奇素数，则模p的原根一定存在，且模p的原根个数是φ(p-1)

2.8整数的阶与原根

整数的阶：设a、m都是整数，且m ≥ 1，gcd (a, m) = 1，则使得：

$$a^x \equiv 1 \pmod{m}$$

成立的最小正整数x叫做a模m的阶(指数)，记作ord_m(a)

原根：若ord_m(a) = φ(m)，则将a叫做模m的一个原根，φ(m)表示在m处的欧拉函数值

原根个数定理：若模m的原根存在，则它一共有$\varphi(\varphi(m))$个原根

原根存在定理：若p是奇素数，则模p的原根一定存在，且模p的原根个数是φ(p-1)

原根在密码学上的应用就是离散对数

2.8整数的阶与原根

重要性质：若p是一个素数，a是模p的一个原根，则下列数：

$$a^1 \% p，a^2 \% p，a^3 \% p，...a^{p-1} \% p$$

各不相同(即两两不同余)，并且组成了从1到p-1的所有整数

2.8整数的阶与原根

重要性质：若p是一个素数，a是模p的一个原根，则下列数：

$$a^1 \,\%\, p，a^2 \,\%\, p，a^3 \,\%\, p，\ldots a^{p-1} \,\%\, p$$

各不相同(即两两不同余)，并且组成了从1到p-1的所有整数

反证(证明两两不同余)：假设$0 < i, j \le p - 1$，使得$a^j \equiv a^i \pmod p$

$\because \gcd(a^1, p) = 1$，且由算数基本定理易知$\gcd(a^i, p) = 1$

$\therefore$可得$a^{j-i} \equiv 1 \pmod p$ (1)

又$\because$易知存在一个原根满足$ord_p(a) = \varphi(p)$，且p是素数 $\Longrightarrow \varphi(p) = p - 1 = ord_p(a)$

而对于(1)式来说，$j - i < p - 1 = ord_p(a)$，与阶的定义矛盾，假设不成立

$\therefore$原命题得证

2.8整数的阶与原根

重要性质：若p是一个素数，a是模p的一个原根，则下列数：

$$a^1\ \%\ p，a^2\ \%\ p，a^3\ \%\ p，\ldots a^{p-1}\ \%\ p$$

各不相同(即两两不同余)，并且组成了从1到p-1的所有整数

说明：由于此时已经得到$a^1\ \%\ p$，$a^2\ \%\ p$，$a^3\ \%\ p$，$\ldots a^{p-1}\ \%\ p$这p - 1个数模p两两不同余，且$a^i\ \%\ p \in [0, p-1], (0 < i \le p-1)$。则意味着这p - 1个数不重复地取遍1~p - 1这些值

*2.9离散对数

2.9离散对数

离散对数(指数)：设a，b，m是整数，且m > 1，gcd (b, m) = 1，a是模m的一个原根，则存在唯一一个整数x，使得：

$$b \equiv a^x \pmod{m}$$

成立，则x叫做以a为底的b对模m的一个离散对数(指数)，记作$x = ind_a b$

同余式定理与积性函数

求解数论问题的利器

目录

同余式定理

3.1威尔逊定理

3.2费马小定理和欧拉定理

3.3*大素数测试

3.4*大素数分解

积性函数

3.5积性函数的基础知识

3.6欧拉函数

3.7因子个数与因子和

3.1威尔逊定理(Wilson)

3.1威尔逊定理

威尔逊定理(Wilson)：若p是素数，则(p - 1)! ≡ -1 (mod p)

3.1威尔逊定理

威尔逊定理(Wilson)：若p是素数，则$(p-1)! \equiv -1 \pmod p$

重要结论：若$p(p \neq 4)$是一个合数，则$(p-1)! \equiv 0 \pmod p$。

若$p = 4$，则$(p-1)! \equiv 2 \pmod p$

3.1威尔逊定理

威尔逊定理(Wilson)：若p是素数，则$(p - 1)! \equiv -1 \pmod p$

重要结论：若$p(p \neq 4)$是一个合数，则$(p - 1)! \equiv 0 \pmod p$。

若$p = 4$，则$(p - 1)! \equiv 2 \pmod p$

(HDU5391 BC#51Div2)：这道题直接使用上面的结论进行简单的素数判定即可

3.1威尔逊定理

威尔逊定理(Wilson)：若p是素数，则(p - 1)! ≡ -1 (mod p)

重要结论：若p(p ≠ 4)是一个合数，则(p - 1)! ≡ 0 (mod p)。

若p = 4，则(p - 1)! ≡ 2 (mod p)

(HDU2973)题目大意：现给出Q次查询，每次查询给出一个n，每次计算S_n = $\sum_{k=1}^{n}\left(\frac{(3k+6)!+1}{3k+7}-\left\lfloor\frac{(3k+6)!}{3k+7}\right\rfloor\right)$的值 (Q ≤ 1e6，1 ≤ n ≤ 1e6)

3.1威尔逊定理

威尔逊定理(Wilson)：若p是素数，则(p - 1)! ≡ -1 (mod p)

重要结论：若p(p ≠ 4)是一个合数，则(p - 1)! ≡ 0 (mod p)。

若p = 4，则(p - 1)! ≡ 2 (mod p)

(HDU2973)题目大意：现给出Q次查询，每次查询给出一个n，每次计算S_n = $\sum_{k=1}^{n}\left(\frac{(3k+6)!+1}{3k+7}-\left\lfloor\frac{(3k+6)!}{3k+7}\right\rfloor\right)$的值 (Q ≤ 1e6，1 ≤ n ≤ 1e6)

提示：令p = 3k + 7，将和式的一般项化简

3.1威尔逊定理

威尔逊定理(Wilson)：若p是素数，则(p - 1)! ≡ -1 (mod p)

重要结论：若p(p ≠ 4)是一个合数，则(p - 1)! ≡ 0 (mod p)。

若p = 4，则(p - 1)! ≡ 2 (mod p)

(HDU2973)题目大意：现给出Q次查询，每次查询给出一个n，每次计算S_n = $\sum_{k=1}^{n}\left(\frac{(3k+6)!+1}{3k+7}-\left\lfloor\frac{(3k+6)!}{3k+7}\right\rfloor\right)$的值 (Q ≤ 1e6，1 ≤ n ≤ 1e6)

提示：令p = 3k + 7，将和式的一般项化简

一旦问题中出现大数阶乘或者是大数阶乘+1形式的公式，就应该考虑Wilson定理了

3.2费马小定理和欧拉定理

3.2费马小定理和欧拉定理

费马小定理：若p是一个素数，且gcd (a, p) = 1，则有

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

欧拉定理：设m是一个正整数，a是一个整数，且满足gcd(a, m) = 1，则有

$$a^{\varphi(m)} \equiv 1 \pmod{m}$$, φ(m)表示m处的欧拉函数值

其实费马小定理是欧拉定理当m是素数时的一个特例

3.2费马小定理和欧拉定理

费马小定理：若p是一个素数，且gcd (a, p) = 1，则有

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

欧拉定理：设m是一个正整数，a是一个整数，且满足gcd(a, m) = 1，则有

$a^{\varphi(m)} \equiv 1 \pmod{m}$，$\varphi(m)$表示m处的欧拉函数值

其实费马小定理是欧拉定理当m是素数时的一个特例

重要结论：对于同余式$a^{x} \equiv 1 \pmod{m}$，其中gcd (a, m) = 1，m > 1，x为满足条件的最小正整数，则有：

$$x \mid \varphi(m)$$

3.2费马小定理和欧拉定理

费马小定理：若p是一个素数，且gcd (a, p) = 1，则有

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

欧拉定理：设m是一个正整数，a是一个整数，且满足gcd(a, m) = 1，则有

$$a^{\varphi(m)} \equiv 1 \pmod{m}$$, φ(m)表示m处的欧拉函数值

其实费马小定理是欧拉定理当m是素数时的一个特例

重要结论：对于同余式$a^x \equiv 1 \pmod{m}$，其中gcd (a, m) = 1，m > 1，x为满足条件的最小正整数，则有：

$$x \mid \varphi(m)$$

费马小定理和欧拉定理是求解指数循环节问题、构建Miller-Rabin和Pollard rho算法的关键

3.2费马小定理和欧拉定理

费马小定理和欧拉定理一个重要的应用是求解指数循环节类问题

3.2费马小定理和欧拉定理

费马小定理和欧拉定理一个重要的应用是求解指数循环节类问题

设a、b、q都是正整数，则我们把形如：

$$a^b \% p$$

的式子叫做指数循环式，由于这个式子要模上p，可以理解为寻找循环节，所以求解上面这个式子的问题就叫做指数循环节问题

3.2费马小定理和欧拉定理

费马小定理和欧拉定理一个重要的应用是求解指数循环节类问题

设a、b、q都是正整数，则我们把形如：

$$a^b \ \% \ \mathrm{p}$$

的式子叫做指数循环式，由于这个式子要模上p，可以理解为寻找循环节，所以求解上面这个式子的问题就叫做指数循环节问题

首先，对于$0 < b < 2^{1e6}$，可以使用学过的整数快速幂取模在O(logb)的时间下将上面的式子求解出来，只不过若b超过long long的话，需要高精度手动模拟

3.2费马小定理和欧拉定理

费马小定理和欧拉定理一个重要的应用是求解指数循环节类问题

设a、b、q都是正整数，则我们把形如：

$$a^b \text{ \% p}$$

的式子叫做指数循环式，由于这个式子要模上p，可以理解为寻找循环节，所以求解上面这个式子的问题就叫做指数循环节问题

首先，对于$0 < b < 2^{1e6}$，可以使用学过的整数快速幂取模在O(logb)的时间下将上面的式子求解出来，只不过若b超过long long的话，需要高精度手动模拟

如果b = 10^{1e7}，该如何计算？此时就算是使用高精度+快速幂都会TLE！！！

因为此时仍直接使用快速幂求解的话复杂度是$\log_2 10^{1e7} = \frac{\log_{10} 10^{1e7}}{\log_{10} 2} \approx$ 3e7，这还不算额外的计算开销

3.2费马小定理和欧拉定理

那该如何求解呢?

3.2费马小定理和欧拉定理

那该如何求解呢?

使用费马小定理或者是欧拉定理进行降幂处理，就是通过公式将原本十分巨大的b缩小成较小的数，然后再调用快速幂求解

3.2费马小定理和欧拉定理

那该如何求解呢?

使用费马小定理或者是欧拉定理进行降幂处理，就是通过公式将原本十分巨大的b缩小成较小的数，然后再调用快速幂求解

(i)使用费马小定理降幂

设a、b、p都是整数，且p是素数，gcd (a, p) = 1，则此时有：

$$a^b \ \% \ p = a^{b \ \% \ (p-1)} \ \% \ p$$

3.2费马小定理和欧拉定理

那该如何求解呢?

使用费马小定理或者是欧拉定理进行降幂处理，就是通过公式将原本十分巨大的b缩小成较小的数，然后再调用快速幂求解

(i)使用费马小定理降幂

设a、b、p都是整数，且p是素数，gcd (a, p) = 1，则此时有：

$$a^b \% p = a^{b \% (p-1)} \% p$$

证明： ∵由于b是一个整数，则有b = k · (p - 1) + r，且有$a^{p-1} \equiv 1 \pmod p$

$$\therefore a^b \equiv a^{k\cdot(p-1)+r} \equiv a^r a^{k\cdot(p-1)} \equiv a^r a^{(p-1)^k} \equiv a^r 1^k \equiv a^r \pmod p$$

而r = b % (p - 1)，则原命题得证

3.2费马小定理和欧拉定理

那该如何求解呢？

使用费马小定理或者是欧拉定理进行降幂处理，就是通过公式将原本十分巨大的b缩小成较小的数，然后再调用快速幂求解

(i)使用费马小定理降幂

设a、b、p都是整数，且p是素数，gcd (a, p) = 1，则此时有：

$$a^b \% p = a^{b \% (p-1)} \% p$$

易知b % (p - 1)的大小不会超过p - 2，显然之后直接调用快速幂求解就可以了

上面的这种方法只能适用于模值是素数的情况，如果模值是合数该如何求解？

3.2费马小定理和欧拉定理

如果模值有可能是合数，则使用欧拉定理来降幂

3.2费马小定理和欧拉定理

如果模值有可能是合数，则使用欧拉定理来降幂

(ii)使用欧拉定理降幂

设a、b、m是整数，若gcd (a, m) = 1，则此时有：

$$a^b \ \% \ \mathrm{m} = a^{b \ \% \ \varphi(m)} \ \% \ \mathrm{m}$$

证明的方法和前面使用费马小定理的类似，而且这里不管m是素数还是合数，只要满足gcd (a, m) = 1，上面的等式恒成立，从这里也可以看出欧拉定理是费马小定理的一般形式

3.2费马小定理和欧拉定理

如果模值有可能是合数，则使用欧拉定理来降幂

(ii)使用欧拉定理降幂

设a、b、m是整数，若gcd (a, m) = 1，则此时有：

$$a^b \ \% \ m = a^{b \ \% \ \varphi(m)} \ \% \ m$$

证明的方法和前面使用费马小定理的类似，而且这里不管m是素数还是合数，只要满足gcd (a, m) = 1，上面的等式恒成立，从这里也可以看出欧拉定理是费马小定理的一般形式

再考虑一个更一般的情况，如果gcd (a, m) ≠ 1，则此时欧拉定理也失效了，那么此时该如何求解？

3.2费马小定理和欧拉定理

若gcd (a, m) $\neq$ 1，且m奇偶任意，则此时使用广义欧拉定理来降幂

3.2费马小定理和欧拉定理

若gcd (a, m) $\neq$ 1，且m奇偶任意，则此时使用广义欧拉定理来降幂

(iii)使用广义欧拉定理降幂

设a、b、m是正整数，则此时有：

$$a^b \ \% \ m = a^{b \ \% \ \varphi(m) + \varphi(m)} \ \% \ m$$

证明这个等式需要使用鸽巢定理，而且理论性较强，有兴趣的可以下去自己看一下

3.2费马小定理和欧拉定理

求解指数循环节问题总结

(i)如果模值p是素数，且gcd (a, p) = 1，则有a^b % p = $a^{b\,\%\,(p-1)}$ % p

(ii)如果模值m奇偶任意，且gcd (a, m) = 1，则有a^b % m = $a^{b\,\%\,\varphi(m)}$ % m

(iii)如果模值m奇偶任意，则有a^b % m = $a^{b\,\%\,\varphi(m)+\varphi(m)}$ % m

3.2费马小定理和欧拉定理

求解指数循环节问题总结

(i)如果模值p是素数，且gcd (a, p) = 1，则有a^b % p = $a^{b\,\%\,(p-1)}$ % p

(ii)如果模值m奇偶任意，且gcd (a, m) = 1，则有a^b % m = $a^{b\,\%\,\varphi(m)}$ % m

(iii)如果模值m奇偶任意，则有a^b % m = $a^{b\,\%\,\varphi(m)+\varphi(m)}$ % m

事实上(iii)是求解指数循环节问题最一般的方法，前面两种方法都需要满足额外的条件才能使用

但是上面的三种方法都需要处理高精度取模问题(即b % k)，而高精度取模直接可以从高位到低位对待取模数按位取模

3.2费马小定理和欧拉定理

求解指数循环节问题总结

(i)如果模值p是素数，且gcd (a, p) = 1，则有a^b % p = $a^{b\,\%\,(p-1)}$ % p

(ii)如果模值m奇偶任意，且gcd (a, m) = 1，则有a^b % m = $a^{b\,\%\,\varphi(m)}$ % m

(iii)如果模值m奇偶任意，则有a^b % m = $a^{b\,\%\,\varphi(m)+\varphi(m)}$ % m

事实上(iii)是求解指数循环节问题最一般的方法，前面两种方法都需要满足额外的条件才能使用

但是上面的三种方法都需要处理高精度取模问题(即b % k)，而高精度取模直接可以从高位到低位对待取模数按位取模

有了上面的介绍，看会不会求解这个问题：

$$a^{1000000^{1000000^{1000000}}}\ \%\ \mathrm{m}$$

3.2费马小定理和欧拉定理

求解指数循环节问题总结

(i)如果模值p是素数，且gcd (a, p) = 1，则有a^b % p = $a^{b\,\%\,(p-1)}$ % p

(ii)如果模值m奇偶任意，且gcd (a, m) = 1，则有a^b % m = $a^{b\,\%\,\varphi(m)}$ % m

(iii)如果模值m奇偶任意，则有a^b % m = $a^{b\,\%\,\varphi(m)+\varphi(m)}$ % m

事实上(iii)是求解指数循环节问题最一般的方法，前面两种方法都需要满足额外的条件才能使用

但是上面的三种方法都需要处理高精度取模问题(即b % k)，而高精度取模直接可以从高位到低位对待取模数按位取模

有了上面的介绍，看会不会求解这个问题：

$$a^{1000000^{1000000^{1000000}}}\ \%\ m$$

这个问题叫做求解迭代幂问题，求解方法是先递归求出最上一层的值，然后回溯时分别降幂

3.2费马小定理和欧拉定理

(HDU4549)题目大意：定义一个序列F[n]，满足F[0] = a, F[1] = b, F[n] = F[n-1] · F[n-2] (n > 1)，现在给出a、b和n，求出F[n] % (1e9 + 7)的值，其中$0 \le a, b, n \le 1e9$

3.2费马小定理和欧拉定理

(HDU4549)题目大意：定义一个序列F[n]，满足F[0] = a, F[1] = b, F[n] = F[n-1] · F[n-2] (n > 1)，现在给出a、b和n，求出F[n] % (1e9 + 7)的值，其中$0 \leq a, b, n \leq 1e9$

解析：写出前几项找规律：F[0] = a^1，F[1] = b^1，F[2] = a^1b^1，F[3] = a^1b^2，F[4] = a^2b^3，F[5] = a^3b^5。发现a的指数和b的指数分别构成一个斐波那契数列，则 F[n] = $a^{\alpha(n)}b^{\beta(n)}$ % p，其中

$$\alpha(n) = \alpha(n-1) + \alpha(n-2)\ n > 3 \qquad \alpha(2) = \alpha(3) = 1$$

$$\beta(n) = \beta(n-1) + \beta(n-2)\ n > 2 \qquad \beta(1) = \beta(2) = 1$$

3.2费马小定理和欧拉定理

(HDU4549)题目大意：定义一个序列F[n]，满足F[0] = a, F[1] = b, F[n] = F[n-1] · F[n-2] (n > 1)，现在给出a、b和n，求出F[n] % (1e9 + 7)的值，其中0 ≤ a, b, n ≤ 1e9

解析：写出前几项找规律：F[0] = a^1，F[1] = b^1，F[2] = a^1b^1，F[3] = a^1b^2，F[4] = a^2b^3，F[5] = a^3b^5。发现a的指数和b的指数分别构成一个斐波那契数列，则 F[n] = $a^{\alpha(n)}b^{\beta(n)}$ % p，其中

$$\alpha(n) = \alpha(n-1) + \alpha(n-2)\ n > 3 \qquad \alpha(2) = \alpha(3) = 1$$

$$\beta(n) = \beta(n-1) + \beta(n-2)\ n > 2 \qquad \beta(1) = \beta(2) = 1$$

由于α(n)和β(n)的值会很大，则此时需要降幂，由于模值p = 1e9 + 7是素数，且a、b的值都小于p，则(a, p) = (b, p) = 1，可以使用费马小定理降幂

3.2费马小定理和欧拉定理

(HDU4549)题目大意：定义一个序列F[n]，满足F[0] = a, F[1] = b, F[n] = F[n-1] · F[n-2] (n > 1)，现在给出a、b和n，求出F[n] % (1e9 + 7)的值，其中$0 \le a, b, n \le 1e9$

解析：写出前几项找规律：F[0] = a^1，F[1] = b^1，F[2] = $a^1 b^1$，F[3] = $a^1 b^2$，F[4] = $a^2 b^3$，F[5] = $a^3 b^5$。发现a的指数和b的指数分别构成一个斐波那契数列，则 F[n] = $a^{\alpha(n)} b^{\beta(n)}$ % p

则结果res = (($a^{\alpha(n)\%(p-1)}$ % p) · ($b^{\beta(n)\%(p-1)}$ % p)) % p

其中α(n)和β(n)是递推得来的，此时可以使用矩阵快速幂取模来加速求解α(n)和β(n)

*3.3大素数测试(Miller-Rabin算法)

3.3大素数测试

*3.4大素数分解(Pollard rho算法)

3.4大 素 数 分 解

3.5积性函数基础知识

3.5积性函数-积性函数基础知识

算数函数：定义在所有整数上的函数

积性函数：如果算术函数F满足对任意两个互素的正整数x和y，有F(xy) = F(x)F(y)，则称算数函数F为积性(乘性)函数

完全积性函数：如果算数函数对于任意两个正整数x和y，满足F(xy) = F(x)F(y)，则称算数函数F为完全积性(乘性)函数

3.5积性函数-积性函数基础知识

算数函数：定义在所有整数上的函数

积性函数：如果算术函数F满足对任意两个互素的正整数x和y，有F(xy) = F(x)F(y)，则称算数函数F为积性(乘性)函数

完全积性函数：如果算数函数对于任意两个正整数x和y，满足F(xy) = F(x)F(y)，则称算数函数F为完全积性(乘性)函数

性质：若F是一个积性函数，且对任意正整数n都有素因子分解n = ${p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot \ldots \cdot {p_k}^{\alpha_k}$，则有：

$$F(n) = F({p_1}^{\alpha_1}) \cdot F({p_2}^{\alpha_2}) \cdot \ldots \cdot F({p_k}^{\alpha_k})$$

3.6欧拉函数

3.6积性函数-欧拉函数

欧拉函数：欧拉函数φ定义为不超过n且与n互素的正整数个数，记作$\varphi(n)$

比如：

由于1、2、3和4都分别和5互素，即与5互素的正整数个数为4，则$\varphi(5)=4$

3.6积性函数-欧拉函数

欧拉函数：欧拉函数φ定义为不超过n且与n互素的正整数个数，记作$\varphi(n)$

比如：

由于1、2、3和4都分别和5互素，即与5互素的正整数个数为4，则$\varphi(5)=4$

重要性质：

(1)若p是素数，则$\varphi(p)=p-1$，反之也成立

(2)若p是素数，且a是整数，则$\varphi(p^a)=p^a-p^{a-1}$

3.6积性函数-欧拉函数

欧拉函数：欧拉函数φ定义为不超过n且与n互素的正整数个数，记作$\varphi(n)$

比如：

由于1、2、3和4都分别和5互素，即与5互素的正整数个数为4，则$\varphi(5)$ = 4

重要性质：

(1)若p是素数，则$\varphi(p)$ = p - 1，反之也成立

(2)若p是素数，且a是整数，则$\varphi(p^a) = p^a - p^{a-1}$

说明：对于(1)来说，由于p是素数，则只有1和p两个因子。则易知p和1~p - 1这p - 1个数都没有公因子，则p和1~p - 1这p - 1个数都互素，自然$\varphi(p)$ = p - 1

对于(2)来说，小于等于p^a和p^a不互素的数为p、$2p$、$3p$、… $p^{a-1}p$，这些数一共有p^{a-1}个，则小于等于p^a和p^a互素的数的个数为总数p^a减去p^{a-1}

3.6积性函数-欧拉函数

欧拉函数：欧拉函数φ定义为不超过n且与n互素的正整数个数，记作$\varphi(n)$

比如：

由于1、2、3和4都分别和5互素，即与5互素的正整数个数为4，则$\varphi(5)$ = 4

重要性质：

(3)若n是奇数，则$\varphi(2n)$ =$\varphi(n)$

(4)若n是大于2的正整数，则$\varphi(n)$是偶数

(5)若n是正整数，则有：

$$\sum_{d\,|\,n}\varphi(d) = n$$

性质(3)的推论：若n是偶数，则$\varphi(n) \le \frac{n}{2}$

3.6积性函数-欧拉函数

如何求解欧拉函数$\varphi(n)$在n处的函数值呢?

3.6积性函数-欧拉函数

如何求解欧拉函数$\varphi(n)$在n处的函数值呢?

设n = $p_1{}^{\alpha_1} \cdot p_2{}^{\alpha_2} \cdot \ldots \cdot p_k{}^{\alpha_k}$为正整数n的素因子分解，则有：

$$\varphi(n) = n \cdot (1 - \frac{1}{p_1}) \cdot (1 - \frac{1}{p_2}) \cdot \ldots \cdot (1 - \frac{1}{p_k})$$

上式给出了求解$\varphi(n)$的一般方法

3.6积性函数-欧拉函数

如何求解欧拉函数$\varphi(n)$在n处的函数值呢?

设n = $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$为正整数n的素因子分解，则有：

$$\varphi(n) = n \cdot (1 - \frac{1}{p_1}) \cdot (1 - \frac{1}{p_2}) \cdot \ldots \cdot (1 - \frac{1}{p_k})$$

上式给出了求解$\varphi(n)$的一般方法

证明：设n = $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$，且$\varphi(n)$为积性函数，则可得：

$$\varphi(n) = \varphi(p_1^{\alpha_1}) \cdot \varphi(p_2^{\alpha_2}) \cdot \ldots \cdot \varphi(p_k^{\alpha_k})$$

3.6积性函数-欧拉函数

如何求解欧拉函数$\varphi(n)$在n处的函数值呢?

设n = $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$为正整数n的素因子分解，则有：

$$\varphi(n) = n \cdot (1 - \frac{1}{p_1}) \cdot (1 - \frac{1}{p_2}) \cdot \ldots \cdot (1 - \frac{1}{p_k})$$

上式给出了求解$\varphi(n)$的一般方法

证明：设n = $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$，且$\varphi(n)$为积性函数，则可得：

$$\varphi(n) = \varphi(p_1^{\alpha_1}) \cdot \varphi(p_2^{\alpha_2}) \cdot \ldots \cdot \varphi(p_k^{\alpha_k})$$

且$\varphi(p_i^{\alpha_i}) = p_i^{\alpha_i} - p_i^{\alpha_i - 1}$，则可将原式化简得：

$$\varphi(n) = (p_1^{\alpha_1} - p_1^{\alpha_1 - 1}) \cdot (p_2^{\alpha_2} - p_2^{\alpha_2 - 1}) \cdot \ldots \cdot (p_k^{\alpha_k} - p_k^{\alpha_k - 1})$$

3.6积性函数-欧拉函数

如何求解欧拉函数$\varphi(n)$在n处的函数值呢?

设$n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$为正整数n的素因子分解，则有：

$$\varphi(n) = n \cdot (1 - \frac{1}{p_1}) \cdot (1 - \frac{1}{p_2}) \cdot \ldots \cdot (1 - \frac{1}{p_k})$$

上式给出了求解$\varphi(n)$的一般方法

证明：设$n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$，且$\varphi(n)$为积性函数，则可得：

$$\varphi(n) = \varphi(p_1^{\alpha_1}) \cdot \varphi(p_2^{\alpha_2}) \cdot \ldots \cdot \varphi(p_k^{\alpha_k})$$

且$\varphi(p_i^{\alpha_i}) = p_i^{\alpha_i} - p_i^{\alpha_i - 1}$，则可将原式化简得：

$$\varphi(n) = (p_1^{\alpha_1} - p_1^{\alpha_1 - 1}) \cdot (p_2^{\alpha_2} - p_2^{\alpha_2 - 1}) \cdot \ldots \cdot (p_k^{\alpha_k} - p_k^{\alpha_k - 1})$$

将上式中每个括号中的$p_i^{\alpha_i}$提出来得到：

$$\varphi(n) = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k} \cdot (1 - p_1^{-1}) \cdot (1 - p_2^{-1}) \cdot \ldots \cdot (1 - p_k^{-1})$$

将$n = p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$代入到上式，即可证明原命题

3.6积性函数-欧拉函数

现在有了求解欧拉函数值的公式，那么该如何具体实现呢？

3.6积性函数-欧拉函数

现在有了求解欧拉函数值的公式，那么该如何具体实现呢?

由于在证明求解欧拉函数值公式的时候使用了素因子分解，所以可以边分解素因子，边计算欧拉函数值，使用的算法就是素因子分解时介绍的试除法和筛法。至于递推求解算法，后面会详细介绍

3.6积性函数-欧拉函数

现在有了求解欧拉函数值的公式，那么该如何具体实现呢？

由于在证明求解欧拉函数值公式的时候使用了素因子分解，所以可以边分解素因子，边计算欧拉函数值，使用的算法就是素因子分解时介绍的试除法和筛法。至于递推求解算法，后面会详细介绍

常用方法：

1.试除法

2.筛法

3.递推求解(重要)

3.6积性函数-欧拉函数

```
long long GetEuler (long long val)
{
    long long ans = val;
    for (long long i = 2; i * i <= val; i++)
    {
        if (val % i == 0)
        {
            ans -= ans / i;
            while (val % i == 0)
            {
                val /= i;
            }
        }
    }

    if (val > 1)
        ans -= ans / val;

return ans;
}
```

解析：这是使用试除法求解在val处的欧拉函数值的代码。

想一下，代码中为什么会有ans -= ans / i这个式子？

3.6积性函数-欧拉函数

```
long long GetEuler (long long val)
{
    long long ans = val;
    for (long long i = 2; i * i <= val; i++)
    {
        if (val % i == 0)
        {
            ans -= ans / i;
            while (val % i == 0)
            {
                val /= i;
            }
        }
    }

    if (val > 1)
        ans -= ans / val;

return ans;
}
```

解析：这是使用试除法求解在val处的欧拉函数值的代码。

想一下，代码中为什么会有ans -= ans / i这个式子？

解析：可知求解欧拉函数的公式为：

$$\varphi(n) = n \cdot (1 - \frac{1}{p_1}) \cdot (1 - \frac{1}{p_2}) \cdot \ldots \cdot (1 - \frac{1}{p_k})$$

则先令ans = n，然后从小到大找到所有的素数，假设此时找的素数是p_1，则将ans乘上$(1 - \frac{1}{p_1})$即：

$$ans \cdot (1 - \frac{1}{p_1}) = ans - ans \cdot \frac{1}{p_1}$$

将此时的值赋给ans，即：

$$ans = ans - ans \cdot \frac{1}{p_1} \Leftrightarrow ans \mathrel{-}= ans \cdot \frac{1}{p_1}$$

然后重复上面的过程

3.6积性函数-欧拉函数

```
long long GetEuler (long long val)
{
    long long ans = val;
    for (long long i = 0; i < prime_len && primelist[i] * primelist[i] <= val; i++)
    {
        if (val % primelist[i] == 0)
        {
            ans -= ans / primelist[i];
            while (val % primelist[i] == 0)
            {
                val /= primelist[i];
            }
        }
    }

    if (val > 1)
        ans -= ans / val;

    return ans;
}
```

解析：这是使用筛法求解欧拉函数值的代码。基本上和前面使用试除法的代码一样

3.6积性函数-欧拉函数

如果遇到需要重复求解欧拉函数值的问题时，就需要打欧拉函数值表。可以使用前面介绍的试除法和筛法打表，但是直接使用递推的算法实现不仅效率更高，而且代码量也少

3.6积性函数-欧拉函数

如果遇到需要重复求解欧拉函数值的问题时，就需要打欧拉函数值表。可以使用前面介绍的试除法和筛法打表，但是直接使用递推的算法实现不仅效率更高，而且代码量也少

递推求解的思想是：先给每个数的欧拉函数值赋上自身值，然后通过其素因子将欧拉函数值缩小，当其素因子都被处理后，就得到了最终的结果，这个过程仔细一想和试除法(或筛法)求解过程是一样的。递推求解效率更好一些是因为能够将具有相同素因子的数的欧拉函数值同时修改，而不需要像试除法那样对于每个数独立修改，独立修改肯定会增加额外的查找素因子的开销，自然会慢不少

3.6积性函数-欧拉函数

```
long long euler[maxn+10];

void GetEuler ()
{
    for (long long i = 0; i < maxn; i++)
        euler[i] = i;

    for (long long i = 2; i < maxn; i += 2)
        euler[i] >>= 1;

    for (long long i = 3; i < maxn; i += 2)
    {
        if (euler[i] == i)
        {
            for (long long j = i; j < maxn; j += i)
                euler[j] = euler[j] - euler[j] / i;
        }
    }
}
```

解析：euler[i]表示i处的欧拉函数值

第一个循环先将所有数的欧拉函数值置为这个数本身。

第二个循环将所有的偶数的欧拉函数值除以2，这是因为由前面性质(3)的推论euler[i] $\leq \frac{i}{2}$可知，如此操作可以将问题的规模缩小。

3.6积性函数-欧拉函数

```
long long euler[maxn+10];

void GetEuler ()
{
    for (long long i = 0; i < maxn; i++)
        euler[i] = i;

    for (long long i = 2; i < maxn; i += 2)
        euler[i] >>= 1;

    for (long long i = 3; i < maxn; i += 2)
    {
        if (euler[i] == i)
        {
            for (long long j = i; j < maxn; j += i)
                euler[j] = euler[j] - euler[j] / i;
        }
    }
}
```

解析：euler[i]表示i处的欧拉函数值

第一个循环先将所有数的欧拉函数值置为这个数本身。

第二个循环将所有的偶数的欧拉函数值除以2，这是因为由前面性质(3)的推论euler[i] $\leq \frac{i}{2}$可知，如此操作可以将问题的规模缩小。

由于若euler[i] = i - 1，则i一定是素数，如果在第三个循环过程中遇到欧拉函数值等于自身的情况(即euler[i] == i)，则说明i是素数，按照前面介绍的过程将当前数i的欧拉函数值进行如下修改：

euler[i] = euerl[i] - euler[i] / i

并且将i的所有倍数j(都含有i这个素因子)的欧拉函数值也进行同样的修改

3.6积性函数-欧拉函数

```
long long euler[maxn+10];

void GetEuler ()
{
    for (long long i = 0; i < maxn; i++)
        euler[i] = i;

    for (long long i = 2; i < maxn; i += 2)
        euler[i] >>= 1;

    for (long long i = 3; i < maxn; i += 2)
    {
        if (euler[i] == i)
        {
            for (long long j = i; j < maxn; j += i)
                euler[j] = euler[j] - euler[j] / i;
        }
    }
}
```

想想，为什么当euler[i] == i(即等于自身时)，i就是素数？ (i > 2)

解析：euler[i]表示i处的欧拉函数值

第一个循环先将所有数的欧拉函数值置为这个数本身。
第二个循环将所有的偶数的欧拉函数值除以2，这是因为由前面性质(3)的推论euler[i] $\leq \frac{i}{2}$可知，如此操作可以将问题的规模缩小。
由于若euler[i] = i - 1，则i一定是素数，如果在第三个循环过程中遇到欧拉函数值等于自身的情况(即euler[i] == i)，则说明i是素数，按照前面介绍的过程将当前数i的欧拉函数值进行如下修改：

euler[i] = euerl[i] - euler[i] / i

并且将i的所有倍数j(都含有i这个素因子)的欧拉函数值也进行同样的修改

3.6积性函数-欧拉函数

```
long long euler[maxn+10];

void GetEuler ()
{
    for (long long i = 0; i < maxn; i++)
        euler[i] = i;

    for (long long i = 2; i < maxn; i += 2)
        euler[i] >>= 1;

    for (long long i = 3; i < maxn; i += 2)
    {
        if (euler[i] == i)
        {
            for (long long j = i; j < maxn; j += i)
                euler[j] = euler[j] - euler[j] / i;
        }
    }
}
```

解析：首先所有偶数i的euler[i]已经在前面循环中除以2了，所以euler[i] != i。而剩下的奇数中，合数奇数i一定是其一个小于等于$\sqrt{i}$的一个素因子的倍数进而被修改(从小到大判断的)。则合数奇数i的euler[i] != i。则同理可知最终剩下的素数奇数i在判断euler[i]是否等于i之前不会被修改，且其初值置为i，则表明其euler[i] == i。说明这个等式对于奇数素数唯一成立。当然唯一一个偶数素数2的欧拉函数值已经在除以2时求出来了

想想，为什么当euler[i] == i(即等于自身时)，i就是素数？ (i > 2)

3.6积性函数-欧拉函数

```
long long euler[maxn+10];

void GetEuler ()
{
    for (long long i = 0; i < maxn; i++)
        euler[i] = i;

    for (long long i = 2; i < maxn; i += 2)
        euler[i] >>= 1;

    for (long long i = 3; i < maxn; i += 2)
    {
        if (euler[i] == i)
        {
            for (long long j = i; j < maxn; j += i)
                euler[j] = euler[j] - euler[j] / i;
        }
    }
}
```

将这里改成for (long long i = 2; i < maxn; i++)

在这里输出当前的euler[i]的值

3.6积性函数-欧拉函数

```
long long euler[maxn+10];

void GetEuler ()
{
    for (long long i = 0; i < maxn; i++)
        euler[i] = i;

    for (long long i = 2; i < maxn; i += 2)
        euler[i] >>= 1;

    for (long long i = 2; i < maxn; i++)
    {
        if (euler[i] == i)
        {
            for (long long j = i; j < maxn; j += i)
                euler[j] = euler[j] - euler[j] / i;
        }
        cout << euler[i] << endl;
        //此时euler[i]已经得到
    }
}
```

这样修改后和前面的没有本质的区别，只是为了完整输出euler[i]的值

再延伸一下，能不能想到每次输出的euler[i]是已经计算完成的？即意味着i每循环一次，就能够确定当前euler[i]的值？

3.6积性函数-欧拉函数

```
long long euler[maxn+10];

void GetEuler ()
{
    for (long long i = 0; i < maxn; i++)
        euler[i] = i;

    for (long long i = 2; i < maxn; i += 2)
        euler[i] >>= 1;

    for (long long i = 2; i < maxn; i++)
    {
        if (euler[i] == i)
        {
            for (long long j = i; j < maxn; j += i)
                euler[j] = euler[j] - euler[j] / i;
        }
        cout << euler[i] << endl;
        //此时euler[i]已经得到
    }
}
```

这样修改后和前面的没有本质的区别，只是为了完整输出euler[i]的值

再延伸一下，能不能想到每次输出的euler[i]是已经计算完成的？即意味着i每循环一次，就能够确定当前euler[i]的值？

说明：若此时euler[i] == i，则说明此时i是素数，则经过一次修改后，当前euler[i] = i - 1计算完成。若此时euler[i] != i，则说明i是合数，此时其已经通过其所有的素因子修改完值了，则说明此时euler[i]就是当前数i的欧拉函数值

3.6积性函数-欧拉函数

欧拉函数的应用

(1) 求解$\sum_{i=1}^{n}\gcd(i,n)$的值

(2) 求解$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gcd(i,j)$的值

3.6积性函数-欧拉函数

欧拉函数的应用

(1)求解$\sum_{i=1}^{n}\gcd(i,n)$的值

如果能够知道gcd (i, n) = k的个数，则将其乘以k累加起来就是答案。而

$$\gcd(i, n) = k \Longrightarrow \gcd(\frac{i}{k}, \frac{n}{k}) = 1$$

后者的个数是$\varphi\left(\frac{n}{k}\right)$，则所有gcd (i, n) = k的和是$k \cdot \varphi\left(\frac{n}{k}\right)$，则

$$\sum_{i=1}^{n}\gcd(i,n) = \sum_{k\,|\,n} k \cdot \varphi\left(\frac{n}{k}\right)$$

3.6积性函数-欧拉函数

欧拉函数的应用

(1) 求解$\sum_{i=1}^{n}\gcd(i,n)$的值

如果能够知道gcd (i, n) = k的个数，则将其乘以k累加起来就是答案。而

$$\gcd(i,n)=k \Longrightarrow \gcd\left(\frac{i}{k},\frac{n}{k}\right)=1$$

后者的个数是$\varphi\left(\frac{n}{k}\right)$，则所有gcd (i, n) = k的和是$k\cdot\varphi\left(\frac{n}{k}\right)$，则

$$\sum_{i=1}^{n}\gcd(i,n)=\sum_{k\mid n}k\cdot\varphi\left(\frac{n}{k}\right)$$

具体实现时只需要枚举小于等于$\sqrt{n}$的因子k，大于$\sqrt{n}$的因子可以通过$\frac{n}{k}$同时计算出来

3.6积性函数-欧拉函数

欧拉函数的应用

(1) 求解$\sum_{i=1}^{n}\gcd(i,n)$的值

如果能够知道gcd (i, n) = k的个数，则将其乘以k累加起来就是答案。而

$$\gcd(i, n) = k \Longrightarrow \gcd\left(\frac{i}{k}, \frac{n}{k}\right) = 1$$

后者的个数是$\varphi\left(\frac{n}{k}\right)$，则所有gcd (i, n) = k的和是$k \cdot \varphi\left(\frac{n}{k}\right)$，则

$$\sum_{i=1}^{n}\gcd(i,n) = \sum_{k\mid n} k \cdot \varphi\left(\frac{n}{k}\right)$$

注意要特判$k \cdot k = n$的情况，此时就不要计算两次了

3.6积性函数-欧拉函数

```
long long tot = 0;
for (long long i = 1; i * i <= n; i++)
{
    if (n % i == 0)
    {
        tot += i * GetEuler (n / i);
        if (i * i < n)
            tot += (n / i) * GetEuler (i);
    }
}
```

3.6积性函数-欧拉函数

欧拉函数的应用

(2) 求解$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gcd(i,j)$的值

3.6积性函数-欧拉函数

欧拉函数的应用

(2)求解$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gcd(i,j)$的值

由于$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gcd(i,j) = \sum_{i=1}^{1}\gcd(i,1) + \sum_{i=1}^{2}\gcd(i,2) + \ldots + \sum_{i=1}^{n}\gcd(i,n)$

3.6积性函数-欧拉函数

欧拉函数的应用

(2)求解$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gcd(i,j)$的值

由于$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gcd(i,j)=\sum_{i=1}^{1}\gcd(i,1)+\sum_{i=1}^{2}\gcd(i,2)+\ldots+\sum_{i=1}^{n}\gcd(i,n)$

每一项都可以用(1)式的方法求解，总的复杂度是O $(n\sqrt{n})$ (已预打好欧拉函数表)

但是还有一个复杂度更低的实现：

3.6积性函数-欧拉函数

欧拉函数的应用

(2)求解$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gcd(i,j)$的值

由于$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gcd(i,j) = \sum_{i=1}^{1}\gcd(i,1) + \sum_{i=1}^{2}\gcd(i,2) + \dots + \sum_{i=1}^{n}\gcd(i,n)$

每一项都可以用(1)式的方法求解，总的复杂度是O $(n\sqrt{n})$ (已预打好欧拉函数表)

但是还有一个复杂度更低的实现：

现设$F(n) = \sum_{i=1}^{n}\gcd(i,n)$，对于$\gcd(a, b) = 1, a < b$来说，$\gcd(2\cdot a, 2\cdot b) = 2$，$\gcd(3\cdot a, 3\cdot b) = 3$，… $\gcd(k\cdot a, k\cdot b) = k$，则每次从小到大求出b的欧拉函数值，之后直接

$$F(k\cdot b) \mathrel{+}= k\cdot\varphi(b)$$

3.6积性函数-欧拉函数

欧拉函数的应用

(2)求解$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gcd(i,j)$的值

由于$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\gcd(i,j) = \sum_{i=1}^{1}\gcd(i,1) + \sum_{i=1}^{2}\gcd(i,2) + \ldots + \sum_{i=1}^{n}\gcd(i,n)$

每一项都可以用(1)式的方法求解，总的复杂度是O ($n\sqrt{n}$) (已预打好欧拉函数表)

但是还有一个复杂度更低的实现：

现设F(n) = $\sum_{i=1}^{n}\gcd(i,n)$，对于gcd (a, b) = 1, a < b来说，gcd (2 · a, 2 · b) = 2，gcd (3 · a, 3 · b) = 3，… gcd (k · a, k · b) = k，则每次从小到大求出b的欧拉函数值，之后直接

$$F(k \cdot b) \mathrel{+}= k \cdot \varphi(b)$$

这一步骤可以和Euler筛素数同时完成，只需在每次循环之后加上上面的语句就可以了

3.6积性函数-欧拉函数

欧拉函数的应用

(POJ2480) 题目大意：有多组测试用例，每个测试用例给出一个整数n，现要求解$\sum_{i=1}^{n}\gcd(i,n)$的值，其中$1 < n < 2^{31}$

3.6积性函数-欧拉函数

欧拉函数的应用

(POJ2480) 题目大意：有多组测试用例，每个测试用例给出一个整数n，现要求解 $\sum_{i=1}^{n} \gcd(i,n)$ 的值，其中 $1 < n < 2^{31}$

解析：由前面的分析可知有 $res = \sum_{k\,|\,n} k \cdot \varphi\left(\frac{n}{k}\right)$。但是n比较大，且有多组数据，哪怕是先预打出欧拉函数表，此时也会TLE，此时需要对公式进行优化

3.6积性函数-欧拉函数

欧拉函数的应用

(POJ2480) 题目大意：有多组测试用例，每个测试用例给出一个整数n，现要求解$\sum_{i=1}^{n} \gcd(i,n)$的值，其中$1 < n < 2^{31}$

解析：由前面的分析可知有$res = \sum_{k\,|\,n} k \cdot \varphi\left(\frac{n}{k}\right)$。但是n比较大，且有多组数据，哪怕是先预打出欧拉函数表，此时也会TLE，此时需要对公式进行优化

设G(x) = x，F(x) = $\varphi(x)$，则可知G(x)和F(x)为完全积性函数，此时设

$H(x) = (F \cdot G)(x) = \sum_{k\,|\,x} x \cdot \varphi\left(\frac{n}{x}\right)$，即此时H(x)为G(x)和F(x)的狄利克雷卷积

3.6积性函数-欧拉函数

欧拉函数的应用

(POJ2480) 题目大意：有多组测试用例，每个测试用例给出一个整数n，现要求解 $\sum_{i=1}^{n} \gcd(i,n)$ 的值，其中 $1 < n < 2^{31}$

解析：由前面的分析可知有 $res = \sum_{k|n} k \cdot \varphi\left(\frac{n}{k}\right)$。但是n比较大，且有多组数据，哪怕是先预打出欧拉函数表，此时也会TLE，此时需要对公式进行优化

设G(x) = x，F(x) = $\varphi(x)$，则可知G(x)和F(x)为完全积性函数，此时设

$H(x) = (F \cdot G)(x) = \sum_{k|x} x \cdot \varphi\left(\frac{n}{x}\right)$，即此时H(x)为G(x)和F(x)的狄利克雷卷积

两个积性函数的狄利克雷卷积仍是积性函数，所以此时有

$$H(x) = H({p_1}^{\alpha_1}) \cdot H({p_2}^{\alpha_2}) \cdot \ldots H({p_k}^{\alpha_k})$$

3.6积性函数-欧拉函数

欧拉函数的应用

(POJ2480) 题目大意：有多组测试用例，每个测试用例给出一个整数n，现要求解 $\sum_{i=1}^{n}\gcd(i,n)$ 的值，其中$1 < n < 2^{31}$

解析：由前面的分析可知有res = $\sum_{k\,|\,n} k \cdot \varphi\left(\frac{n}{k}\right)$。但是n比较大，且有多组数据，哪怕是先预打出欧拉函数表，此时也会TLE，此时需要对公式进行优化

设G(x) = x，F(x) = $\varphi(x)$，则可知G(x)和F(x)为完全积性函数，此时设

H(x) = (F · G) (x) = $\sum_{k\,|\,x} x \cdot \varphi\left(\frac{n}{x}\right)$，即此时H(x)为G(x)和F(x)的狄利克雷卷积

两个积性函数的狄利克雷卷积仍是积性函数，所以此时有

$$H(x) = H({p_1}^{\alpha_1}) \cdot H({p_2}^{\alpha_2}) \cdot \ldots H({p_k}^{\alpha_k})$$

且可易知$H({p_i}^{\alpha_i}) = \sum_{i=0}^{\alpha_i} p_i\varphi\left({p_i}^{\alpha_i-i}\right)$，且有$\varphi({p_i}^{\alpha_i}) = {p_i}^{\alpha_i} - {p_i}^{\alpha_i-1}$

3.6积性函数-欧拉函数

欧拉函数的应用

(POJ2480) 题目大意：有多组测试用例，每个测试用例给出一个整数n，现要求解$\sum_{i=1}^{n}\gcd(i,n)$的值，其中$1 < n < 2^{31}$

解析：由前面的分析可知有res = $\sum_{k\,|\,n} k \cdot \varphi\left(\frac{n}{k}\right)$。但是n比较大，且有多组数据，哪怕是先预打出欧拉函数表，此时也会TLE，此时需要对公式进行优化

则可得到

$$\mathrm{H}({p_i}^{\alpha_i}) = \sum_{i=0}^{\alpha_i} p_i\varphi\left({p_i}^{\alpha_i - i}\right) = {p_i}^{\alpha_i} - {p_i}^{\alpha_i - 1} + p_1({p_i}^{\alpha_i - 1} - {p_i}^{\alpha_i - 2}) + \ldots + {p_i}^{\alpha_i}$$

$$= \alpha_i({p_i}^{\alpha_i} - {p_i}^{\alpha_i - 1}) + {p_i}^{\alpha_i}$$

$$= (\alpha_i + 1)\,{p_i}^{\alpha_i} - \alpha_i {p_i}^{\alpha_i - 1}$$

3.6积性函数-欧拉函数

欧拉函数的应用

(POJ2480) 题目大意：有多组测试用例，每个测试用例给出一个整数n，现要求解 $\sum_{i=1}^{n} \gcd(i,n)$ 的值，其中1 < n < 2^{31}

解析：由前面的分析可知有res = $\sum_{k\mid n} k \cdot \varphi\left(\frac{n}{k}\right)$。但是n比较大，且有多组数据，哪怕是先预打出欧拉函数表，此时也会TLE，此时需要对公式进行优化

则可得到

$$H({p_i}^{\alpha_i}) = \sum_{i=0}^{\alpha_i} p_i \varphi\left({p_i}^{\alpha_i - i}\right) = {p_i}^{\alpha_i} - {p_i}^{\alpha_i - 1} + p_1({p_i}^{\alpha_i - 1} - {p_i}^{\alpha_i - 2}) + \ldots + {p_i}^{\alpha_i}$$

$$= \alpha_i({p_i}^{\alpha_i} - {p_i}^{\alpha_i - 1}) + {p_i}^{\alpha_i}$$

$$= (\alpha_i + 1)\, {p_i}^{\alpha_i} - \alpha_i {p_i}^{\alpha_i - 1}$$

有了这个公式后，直接分解素因子求解即可

3.7因子个数与因子和

3.7积性函数-因子个数与因子和

因子和函数：因子和函数σ定义为正整数n的所有正因子之和，记作$\sigma(n)$

因子个数函数：因子个数函数τ定义为正整数n的所有正因子的个数，记作$\tau(n)$

3.7积性函数-因子个数与因子和

因子和函数：因子和函数σ定义为正整数n的所有正因子之和，记作$\sigma(n)$

因子个数函数：因子个数函数τ定义为正整数n的所有正因子的个数，记作$\tau(n)$

定理1：$\sigma(n)$和$\tau(n)$都是积性函数

定理2：若f是积性函数，则f的和函数F(n) = $\sum_{d\mid n} f(d)$也是积性函数(重要)

3.7积性函数-因子个数与因子和

因子和函数：因子和函数σ定义为正整数n的所有正因子之和，记作$\sigma(n)$

因子个数函数：因子个数函数τ定义为正整数n的所有正因子的个数，记作$\tau(n)$

定理1：$\sigma(n)$和$\tau(n)$都是积性函数

定理2：若f是积性函数，则f的和函数F(n) = $\sum_{d\mid n} f(d)$也是积性函数(重要)

那么该如何求解因子和函数$\sigma(n)$和个数函数$\tau(n)$呢？

3.7积性函数-因子个数与因子和

因子和函数：因子和函数σ定义为正整数n的所有正因子之和，记作$\sigma(n)$

因子个数函数：因子个数函数τ定义为正整数n的所有正因子的个数，记作$\tau(n)$

定理1：$\sigma(n)$和$\tau(n)$都是积性函数

定理2：若f是积性函数，则f的和函数F(n) = $\sum_{d\mid n} f(d)$也是积性函数(重要)

那么该如何求解因子和函数$\sigma(n)$和个数函数$\tau(n)$呢?

引理：设p是一个素数，a是一个正整数，则可得：

$$\sigma(p) = 1 + p^1 + p^2 + \ldots + p^a = \frac{p^{a+1}-1}{p-1},\quad \tau(p) = a + 1$$

3.7积性函数-因子个数与因子和

因子和函数：因子和函数σ定义为正整数n的所有正因子之和，记作$\sigma(n)$

因子个数函数：因子个数函数τ定义为正整数n的所有正因子的个数，记作$\tau(n)$

定理1：$\sigma(n)$和$\tau(n)$都是积性函数

定理2：若f是积性函数，则f的和函数F(n) = $\sum_{d\,|\,n} f(d)$也是积性函数(重要)

那么该如何求解因子和函数$\sigma(n)$和个数函数$\tau(n)$呢?

引理：设p是一个素数，a是一个正整数，则可得：

$$\sigma(p) = 1 + p^1 + p^2 + \dots + p^a = \frac{p^{a+1} - 1}{p - 1}, \quad \tau(p) = a + 1$$

直接对照定义就可以证明出来

3.7积性函数-因子个数与因子和

因子和函数： 因子和函数σ定义为正整数n的所有正因子之和，记作$\sigma(n)$

因子个数函数： 因子个数函数τ定义为正整数n的所有正因子的个数，记作$\tau(n)$

定理1：$\sigma(n)$和$\tau(n)$都是积性函数

定理2：若f是积性函数，则f的和函数$F(n) = \sum_{d \mid n} f(d)$也是积性函数(重要)

那么该如何求解因子和函数$\sigma(n)$和个数函数$\tau(n)$呢?

定理：设正整数$n = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot \ \ldots \cdot {p_k}^{\alpha_k}$，则可得：

$$\sigma(n) = \frac{{p_1}^{\alpha_1} - 1}{p_1 - 1} \cdot \frac{{p_2}^{\alpha_2} - 1}{p_2 - 1} \cdot \ldots \cdot \frac{{p_k}^{\alpha_k} - 1}{p_k - 1} = \prod_{i=1}^{k} \frac{{p_i}^{\alpha_i} - 1}{p_i - 1}$$

$$\tau(n) = (\alpha_1 + 1) \cdot (\alpha_2 + 1) \cdot \ldots \cdot (\alpha_k + 1)$$

3.7积性函数-因子个数与因子和

(POJ1845)题目大意：给出两个自然数A、B，求A^B的所有的因子和对9901取余后的值(0 ≤ A, B ≤ 5e7)

3.7积性函数-因子个数与因子和

(POJ1845)题目大意：给出两个自然数A、B，求A^B的所有的因子和对9901取余后的值$(0 \le A, B \le 5e7)$

解析：先把A分解得到$A = {p_1}^{\alpha_1} \cdot {p_2}^{\alpha_2} \cdot \ldots \cdot {p_k}^{\alpha_k}$，那么得到$A^B = {p_1}^{B\alpha_1} \cdot {p_2}^{B\alpha_2} \cdot \ldots \cdot {p_k}^{B\alpha_k}$，则使用因子和公式可以得到：

$$\sigma(n) = \frac{{p_1}^{B\alpha_1} - 1}{p_1 - 1} \cdot \frac{{p_2}^{B\alpha_2} - 1}{p_2 - 1} \cdot \ldots \cdot \frac{{p_k}^{B\alpha_k} - 1}{p_k - 1} = \prod_{i=1}^{k} \frac{{p_i}^{B\alpha_i} - 1}{p_i - 1}$$

3.7积性函数-因子个数与因子和

(POJ1845)题目大意：给出两个自然数A、B，求A^B的所有的因子和对9901取余后的值(0 ≤ A, B ≤ 5e7)

解析：先把A分解得到A = $p_1^{\alpha_1} \cdot p_2^{\alpha_2} \cdot \ \ldots \cdot p_k^{\alpha_k}$，那么得到$A^B = p_1^{B\alpha_1} \cdot p_2^{B\alpha_2} \cdot \ \ldots \cdot p_k^{B\alpha_k}$，则使用因子和公式可以得到：

$$\sigma(n) = \frac{p_1^{B\alpha_1} - 1}{p_1 - 1} \cdot \frac{p_2^{B\alpha_2} - 1}{p_2 - 1} \cdot \ldots \cdot \frac{p_k^{B\alpha_k} - 1}{p_k - 1} = \prod_{i=1}^{k} \frac{p_i^{B\alpha_i} - 1}{p_i - 1}$$

则此时$\sigma(n)$ % 9901就是答案，此时对于每项分子中的$p_i^{B\alpha_i}$使用快速幂取模，然后分母用前面介绍的乘法逆元处理