> **写在最前面：本文部分内容来自网上各大博客或是各类图书，由我个人整理，增加些许见解，仅做学习交流使用，无任何商业用途。因个人实力时间等原因，本文并非完全原创，请大家见谅。**

《算法竞赛中的初等数论》正文 0x00整除、0x10 整除相关（ACM / OI / MO）（十五万字符数论书）

《算法竞赛中的初等数论》（信奥 / 数竞 / ACM）前言、后记、目录索引（十五万字符的数论书）

全文目录索引链接： https://fanfansann.blog.csdn.net/article/details/113765056

0x00 整除

数论的旅程，从整除开始。整除这个基本概念将贯穿数论始末，理解并掌握它是基础中的基础。

- **整除**

定义：若整数 n 除以整数 d 的余数为 0，即 d 能整除 n ，则称 d 是 n 的约数，n 是 d 的倍数，记为 $d \mid n$。

- **整除的性质**

性质00.1.1： $a \mid b, b \mid c \Rightarrow a|c$

性质00.1.2： $a \mid b \Rightarrow a \mid bc$， c 为任意的整数。

性质00.1.3： $a \mid b, a \mid c \Rightarrow a \mid kb \pm lc$， k 与 l 均为任意的整数。（**都有公因子 a，正确性显然**）
拓展： k_1, k_2 互质，则 $k_1 + k_2$ 与 $k_1 \times k_2$ 互质。（仅有 $a = 1$ 能整除 k_1, k_2，故仅有 $a = 1$ 能同时整除 $k_1 + k_2$ 与 $k_1 \times k_2$）

性质00.1.4：$a \mid b, b \mid a \Rightarrow a = \pm b$

性质00.1.5： $a = kb \pm c \Rightarrow a, b$ 的公因数与 b, c 的公因数完全相同

性质00.1.6： 若 $a \mid bc$，且 a 与 c 互质，则 $a \mid b$

性质 00.1.1 ~ 00.1.4 的正确性可由定义得到，正确性显然，这里仅给出性质00.1.5的证明。**性质 00.1.6** 涉及到互质的概念，具体性质与概念详见本文 **0x14 互质与欧拉函数**。

> **性质00.1.5的证明：**
>
> 利用**性质00.1.3**，$(a|b, a|c \Rightarrow a|kb \pm lc)$，对于任意的 a, b 的公因数d：$a = kb \pm c \Rightarrow c = \pm(a - kb) \Rightarrow d|c$

这里给出一个1987年的初中数学联赛真题帮助大家理解整除的相关概念与性质。

- **1987年全国初中数学联赛**

x, y, z**均为整数，若**$11 \mid (7x + 2y - 5z)$**，求证：**$11 \mid (3x - 7y + 12z)$**。**

Solution

非常经典的一个问题，令 $a = (7x + 2y - 5z)$，$b = (3x - 7y + 12z)$，通过构造可以得到一个等式：
$3a + 4b = 11(3x - 2y + 3z)$，则$4b = 11(3x - 2y + 3z) - 3a$。

性质00.1.2 + **性质00.1.3**，得出 $11 \mid (11(3x - 2y + 3z) - 4a) = 11 \mid 3b$。

性质00.1.6：，由于 11 和 3 互素，得出 $11 \mid b$，证明完毕。

0x10 整除相关

在了解完整除的相关性质以后，我们将进入数论的第一章，最重要的章节之一，也是数论千百年研究突破的重点，整除相关：素数，约数。

0x11 素数（质数）

素数定义

素数（又称质数）是只有 1 和它本身两个因数的数。 规定 1 既非素数也非合数。特殊的，2 是唯一的**偶素数**。

素数定理

我们设 $\pi(x)$ 为 1 到 x 中素数的个数。

其中：

$$\pi(n) = -1 + \sum_{k=1}^{x} \lfloor \cos^2 \lfloor \pi \frac{(n-1)!+1}{n} \rfloor \rfloor$$

> 上述公式是由 **黎曼** 给出的精确公式，详细证明参见：https://mathworld.wolfram.com/RiemannPrimeCountingFunction.html

尝试求极限发现（其中 $\ln(x)$ 表示 x 的自然对数）：

$$\lim_{x->\infty} \frac{\pi(x)}{(\frac{x}{\ln(x)})} = 1$$

即可得到定理：

定理11.0.1： 在自然数集中，小于 n 的质数约有 $\frac{n}{\ln(n)}$ 个。

由该定理可知，是 `int` 范围内的素数的个数并不会很多，其中 `int` 范围内的素数间距大概是 10^2 的数量级。

定理11.0.2： （伯特兰 — 切比雪夫定理）若整数 $n > 3$，则至少存在一个质数 p，符合 $n < p < 2n - 2$。另一个稍弱说法是：对于所有大于 1的整数 n，至少存在一个质数 p，符合 $n < p < 2n$。

0x11.1 素数的判定

既然知道了什么是素数，我们肯定要考虑如何判定一个数是素数，我们往往直接从定义角度出发来判定是否为素数。

接下来介绍的四种算法均为单个素数的判定方法。

- **试除法**

试除法是最常用的判断素数的方法。

一个数如果不是素数，则一定能被一个小于它自己的数整除。假设一个数能整除 n，即 $a \mid n$，那么 $\frac{n}{a}$ 也必定能整除 n，不妨设 $a \leq \frac{n}{a}$，则有$a^2 \leq n$，即$a \leq \sqrt{n}$。

时间复杂度：$O(\sqrt{n})$

```
inline bool is_prime(int x){
    if(x < 2)
        return false;
    for(register int i = 2; i * i ≤ x; ++ i)
        if(x % i == 0)
            return false;
    return true;
}
```

如果想要追求更高的效率，可以考虑使用 $kn+i$ 法

- **$kn+i$ 法**

一个大于 1 的整数如果不是素数，那么一定有素因子，因此在枚举因子时只需要考虑可能为素数的因子即可。 $kn+i$ 法即枚举形如 $kn+i$ 的数，例如取 $k=6$，那么 $6n+2, 6n+3, 6n+4, 6n+6$ 都不可能为素数（显然它们分别有因子 $2, 3, 2, 6$ 一定不是素数），因此我们只需要枚举形如$6n+1, 6n+5$ 的数即可，这样整体的时间复杂度就会降低了$\frac{2}{3}$，也就是 $O(n^{\frac{1}{3}})$。

下面是$kn+i$ 法 $k=30$ 版本的模板：

```
bool isPrime(ll n){
    if(n == 2 || n == 3 || n == 5)return 1;
    if(n % 2 == 0 || n % 3 == 0 || n % 5 == 0 || n == 1) return 0;
    ll c = 7, a[8] = {4,2,4,2,4,6,2,6};
    while(c * c ≤ n) for(auto i : a){if(n % c == 0)return 0; c += i;}
    return 1;
}
```

- **预处理法**

对于多组数据，如果 n 是合数，那么它必然有一个小于等于 $\sqrt{n}$ 的素因子，只需要对 $\sqrt{n}$ 内的素数进行测试即可，也就是预处理求出 $\sqrt{n}$ 中的素数，假设该范围内素数的个数为 s （$s=\frac{n}{\ln n}$），那么时间复杂度为 $O(\frac{n}{\ln n})$。

- **Miller − Rabin 判定法**

对于一个很大的数 n（例如十进制表示有 100 位），如果还是采用试除法进行判定，时间复杂度必定难以承受，目前比较稳定的大素数测试算法是米勒-拉宾（Miller − Rabin）素数测试算法，该素数测试算法可以通过控制迭代次数来间接控制正确率。

Miller − Rabin 判定法是基于费马小定理的，即如果一个数 p 为素数的条件是对于所有和 p 互素的正整数 a 满足以下等式：$a^{p-1} \equiv 1 \pmod{p}$。（费马小定理的相关概念定理性质详见**本文 0x21.2 费马小定理**）

然而我们不可能试遍所有和 p 互素的正整数，这样的话复杂度反而更高，事实上我们只需要取比 p 小的几个素数进行测试就行了。

具体判断 n 是否为素数的算法如下：

1. 如果 `n==2` ，返回 `true`；如果 `n<2 || !(n & 1)`，返回 `false`；否则跳到 ii) 。
2. 令 `n = m *(2 ^ k) + 1`，其中 m 为奇数，则 `n - 1 = m * (2 ^ k)` 。
3. 枚举小于 n 的素数 p（至多枚举 10 个），对每个素数执行费马测试，费马测试如下：计算 `pre = p ^ m % n`，如果 pre 等于 1 ，则该测试失效，继续回到 iii) 测试下一个素数；否则进行 k 次计算 `next = pre ^ 2 % n`，如果 `next == 1 && pre ≠ 1 && pre ≠ n-1` 则n必定是合数，直接返回；k 次计算结束判断 pre 的值，如果不等于 1，必定是合数。
4. 10次判定完毕，如果 n 都没有检测出是合数，那么 n 为素数。

时间复杂度为 $O(k\log n)$

Code

```cpp
ll Rand() {//决定了程序的性能
    static ll x = (srand((int)time(0)), rand());
    x += 1000003;
    if (x > 1000000007)
        x -= 1000000007;

    return x;
}
bool Witness(ll a, ll n) {
    ll t = 0, u = n - 1;
    while (!(u & 1))
        u >>= 1, t++;
    ll x = qpow(a, u, n), y;//qpow为快速幂

    while (t--) {
        y = x * x % n;
        if (y == 1 && x != 1 && x != n - 1)
            return true;
        x = y;
    }
    return x != 1;
}
bool MillerRabin(ll n, ll s) {
    if (n == 2 || n == 3 || n == 5)
        return 1;

    if (n % 2 == 0 || n % 3 == 0 || n % 5 == 0 || n == 1)
        return 0;

    while (s--) {
        if (Witness(Rand() % (n - 1) + 1, n))
            return false;
    }
    return true;
}
```

有了大素数测试算法，那么就会有大数的质因子分解法，详见本文 **0x12.1唯一分解定理**

0x11.2 素数的筛法

通过上文的学习，我们了解了判断单个数是否为素数的四种常用方法，那么对于一个很大范围内的所有数的素数判定，若我们直接对于每一个数都使用一次素数判定法，如此庞大的时间复杂度我们是无法承担的，因此引入了对于大规模数的素数判定方法：**质数筛法**。

质数筛法一般分为**埃氏筛**和**线性筛**。

埃氏筛没有线性筛时间复杂度好，不常用，但是他的**时间复杂度分析方法**却比较常用。

首先我们来证明一下 $O(\sum_{i=1}^{n}\frac{1}{i})\approx O(\log n)$

显然有：

$$\begin{aligned}\sum_{i=1}^{n}\frac{1}{i}&=1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\frac{1}{7}+\cdots\\&=1+(\frac{1}{2}+\frac{1}{3})+(\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\frac{1}{7})+\cdots\end{aligned}$$

显然有

$$\sum_{j=i}^{i+k-1}\frac{1}{i}\leq k\times\frac{1}{i} \tag{1}$$

即：$\frac{1}{2}+\frac{1}{3}\leq\frac{1}{2}\times 2\leq 1$，$\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\frac{1}{7}\leq\frac{1}{4}\times 4\leq 1$

即：

$$\sum_{j=2^{i-1}}^{2^i}\frac{1}{j}\leq 2^{i-1}\times\frac{1}{j}=1 \tag{2}$$

所以 $1\sim n$ 就被分为了 $\log n$ 组，每组的和小于等于 1，故 $\sum_{i=1}^{n}\frac{1}{i}\approx\log n$

- **Eratosthenes 筛法　（埃拉托色尼筛法）**

显然如果 x 是合数，那么 x 的倍数也一定是合数。利用这个结论，我们可以避免很多次不必要的检测。
我们可以从小到大枚举分析每一个数，然后同时把当前这个数的所有（比自己大的）倍数记为合数，那么运行结束的时候没有被标记的数就是素数了。

```
int v[N];
void primes(int n{
    memset(v, 0, sizeof v);
    for(int i = 2;i ≤ n; ++ i){
        if(v[i])continue;
        cout << i << endl;
        for(int j = i;j ≤ n / i; ++ j)
            v[i * j] = 1;
    }
}
```

埃氏筛的时间复杂度为 $O(n\log\log n)\approx O(n)$ （这里的log以10为底），因为我们这里外层循环 $O(n)$，内层循环上界为 $\frac{n}{i}$，随着 i 的增加，$\frac{n}{i}\in\{n,\frac{n}{2},\frac{n}{3},\frac{n}{4},\frac{n}{5}\ldots,\frac{n}{n}\}$，而调和级数$f(n)=1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}\ldots+\frac{1}{n}$ 当 n 趋近于 ∞ 时，调和级数极限为 $\ln n+C$ 其中 C 为欧拉常数 $C\approx 0.577218$ 当我们使用优化筛掉合数以后，这个调和级数就变成 $f(n)=\frac{1}{2}+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}\ldots+\frac{1}{n}\approx\log\log n$，所以整体的算法时间复杂度为 $O(n\log\log n)$。
这个**利用调和级数**计算时间复杂度的方法很常用。

> 更严谨的证明：
>
> 如果每一次对数组的操作花费 1 个单位时间，则时间复杂度为：
>
> $O\left(n\sum_{k=1}^{\pi(n)}\frac{1}{p_k}\right)$
>
> 其中 p_k 表示第 k 小的素数。根据 Mertens 第二定理，存在常数 B_1 使得：
>
> $\sum_{k=1}^{\pi(n)}\frac{1}{p_k}=\log\log n+B_1+O\left(\frac{1}{\log n}\right)$
>
> 所以 **Eratosthenes 筛法** 的时间复杂度为 $O(n\log\log n)$。接下来我们证明 Mertens 第二定理的弱化版本 $\sum_{k\leq\pi(n)}1/p_k=O(\log\log n)$：
>
> 根据 $\pi(n)=\Theta(n/\log n)$，可知第 n 个素数的大小为 $\Theta(n\log n)$。于是就有

$$\begin{aligned}\sum_{k=1}^{\pi(n)}\frac{1}{p_k}&=O\left(\sum_{k=2}^{\pi(n)}\frac{1}{k\log k}\right)\\&=O\left(\int_2^{\pi(n)}\frac{\mathrm{d}x}{x\log x}\right)\\&=O(\log\log\pi(n))=O(\log\log n)\end{aligned}$$

- **线性筛（欧拉筛）**

在欧拉筛我们可以保证每个数一定只会被它的最小质因子筛掉一次。

由于 `primes` 数组中的质数是递增的。

我们从小到大枚举 `primes` 数组，当第一次枚举到一个质数 $\text{primes}[j]$ 满足 $\text{primes}[j] \mid i$ 时， $\text{primes}[j]$ 一定是 i 的最小质因子， $\text{primes}[j]$ 也一定是 $i \times \text{primes}[j]$ 的最小质因子，而接下来的 $i \times \text{primes}[j+1]$ 的最小质因子应该是 $\text{primes}[j]$ 而不是 $\text{primes}[j+1]$ ，故此时直接 `break` 即可。

那么对于任意一个合数 x ，假设 x 的最小质因子为 y ，那么当枚举到 $\frac{x}{y} < x$ 的时候一定会把 x 筛掉，即在枚举到 x 之前一定能把合数 x 筛掉，所以一定能把所有的合数都筛掉。

由于 **保证每个合数只都被自己的最小质因子筛掉一遍** ，所以时间复杂度是 $O(n)$ 的。（注意到筛法求素数的同时也得到了每个数的最小质因子，这是后面筛法求欧拉函数的关键）

Code

```
const int N = 10005;
int n, primes[N], cnt;
bool vis[N];
inline void get_prime()
{
    for(register int i = 2;i ≤ n;i ++) {
        if(!vis[i]) primes[ ++ cnt] = i;
        for(register int j = 1;j ≤ cnt && i * primes[j] ≤ n; ++ j) {
            vis[i * primes[j]] = 1;
            if(i % primes[j] == 0) break;
        }
    }
}
```

- **竞赛例题选讲**

Problem A 夏洛克和他的女朋友（AcWing 1293）

有 n 件珠宝，第 i 件的价值是 $i+1$， 也就是说，珠宝的价值分别为 $2,3,\ldots,n+1$。请你来为这些珠宝染色，使得一件珠宝的价格是另一件珠宝的价格的质因子时，两件珠宝的颜色不同。求使用的颜色数的最小值以及染色方案。

Solution

因为一个数不能与自己的**质因子**同色，所以所有的质数可以同色。

又因为所有质因数已被染色，所以就不需要对其他数染色了。即所有质数染色为 1 ，所有合数染色为 2 即可。所以我们只需要找出 $[2,n+1]$的所有质数输出即可。注意特殊情况，不存在合数的时候，仅输出 1 即可。

Problem B 质数距离（AcWing 196）

给定两个整数 L 和 U，你需要在闭区间$[L,U]$内找到距离最接近的两个相邻质数 C_1 和 C_2（即 C_2-C_1 是最小的），如果存在相同距离的其他相邻质数对，则输出第一对。

同时，你还需要找到距离最远的两个相邻质数 D_1 和 D_2 （即 D_1-D_2 是最大的），如果存在相同距离的其他相邻质数对，则输出第一对。

$1 \le L \le U \le 2^{31}-1, R-L \le 10^6$

Solution

看数据范围做题。如果小的话我们可以直接枚举 $O(n)$ 即可。但是这里数据范围达到了 2^{31}，就算用线性筛也存不下这么大的数。

尽管数据范围大到不可做，我们发现区间$[L, R]$的差值不会超过 10^6，所以我们需要改进线性筛法，得到一个可以求得尽管数据很大但是实际区间长度不大的质数筛法，即**二次筛法**。

首先显然任何一个合数 n 必定包含一个不超过 $\sqrt{n}$ 的**因子** x，该因子 x 一定有一个小于 x 的质因子或者说 x 就是一个质数。

这样我们只需要求出 $2 \sim \sqrt{n}$ 的质数 p，也就是 50000 左右，对于每一个 $L \sim R$ 中的数，能被 p 整除的一定是合数，剩下的就是质数的，再对剩下的质数大概只有两两相比较就行了。

步骤
1、找出$1 \sim \sqrt{2^{31}-1}$ (< 50000)中的所有质因子
2、对于$1 \sim 50000$ 中每个质数 P，将 $[L, R]$ 中所有 P 的倍数筛掉(至少2倍)
3、找到大于等于 L 的最小的 P 的倍数P_0，显然 $P_0 = \max\{2 \times P, \left\lceil \frac{L}{P} \right\rceil\}$，找下一个倍数时只需要 += P 即可。

引理(分数的上取整转换下取整)

$$\left\lceil \frac{L}{P} \right\rceil = \left\lfloor \frac{L+P-1}{P} \right\rfloor$$

Code

```cpp
typedef long long LL;
const int N = 1000010;
int primes[N], cnt;
bool st[N];
void get_primes(int n);//线行筛代码略去
int main()
{
    int l, r;
    while (cin >> l >> r) {
        get_primes(50000);
        memset(st, 0, sizeof st);
        for (int i = 0; i < cnt; i ++ ) {
            LL p = primes[i];
            //j初始化为KP,使得KP>L,即找到最少P需要多少倍可以大于等于左边界L,并且至少需要是二倍。
            for (LL j = max(p * 2, (l + p - 1) / p * p); j ≤ r; j += p)
                st[j - l] = true;
        }
        cnt = 0;
        for (int i = 0; i ≤ r - l; i ++ )
            if (!st[i] && i + l ≥ 2)//特判，因为P可能为1
                primes[cnt ++ ] = i + l;

        if (cnt < 2) puts("There are no adjacent primes.");
        else {
            int minp = 0, maxp = 0;
            for (int i = 0; i + 1 < cnt; i ++ ) {//筛完之后暴力跑
                int d = primes[i + 1] - primes[i];
                if (d < primes[minp + 1] - primes[minp]) minp = i;
                if (d > primes[maxp + 1] - primes[maxp]) maxp = i;
            }

            printf("%d,%d are closest, %d,%d are most distant.\n",
                primes[minp], primes[minp + 1],
                primes[maxp], primes[maxp + 1]);
        }
    }
    return 0;
}
```

0x11.3 反素数

学完素数以后，最后我们再来介绍一下反素数。（建议在学完约数的相关概念以后再来学习本小节）

反素数定义

如果某个正整数 n 满足如下条件，则称为是反素数： 任何**小于** n 的数的正约数个数都**小于** n 的正约数个数，即 n 是 $1 \dots n$ 中正约数个数最多的数。

素数就是因子只有两个的数，那么反素数，就是**因子最多的数**（并且**因子个数相同的时候值最小**），所以反素数是相对于一个集合来说的，也就是在一个集合中，因素最多并且值最小的数，就是反素数。

既然要求因子数，我们首先想到的就是素因子分解。把 n 分解成 $n = p_1^{k_1} p_2^{k_2} \cdots p_n^{k_n}$ 的形式，其中 p 是素数， k 是他的指数。这样的话总因子个数就是 $(k_1 + 1) \times (k_2 + 1) \times (k_3 + 1) \cdots \times (k_n + 1)$ 。

但是对于每一个数都质因数分解的代价太大，总时间复杂度 $O(n\sqrt{n})$，并且每个数都相对独立，比较浪费。

我们考虑一些反素数的性质：

若 $N \leq 2^{31}$

引理11.3.1： $1 \sim N$ 中的反素数，就是$1 \sim N$ 中约数个数最多的数中 **最小** 的一个。

引理11.3.2： $1 \sim N$ 中任何数的不同质因子都不会超过 10 个且所有质因子的质数都不会超过 30。

引理11.3.3： $\forall x \in [1, N]$，x 为反素数的必要条件是： x 分解质因数后可以写成 $2^{c_1} \times 3^{c_2} \times 5^{c_3} \times 7^{c_4} \times 11^{c_5} \times 13^{c_6} \times 17^{c_7} \times 19^{c_8} \times 23^{c_9} \times 29^{c_{10}}$ ，且 $c_1 \geq c_2 \geq c_3 \geq \cdots \geq c_{10} \geq 0$，换句话说， x 的质因子是连续的若干个最小的质数，并指数单调递减。

这三个引理非常显然，应该不需要证明。

根据上面的三个引理，我们可以直接DFS，一次确认前 10 个质数的指数，并满足指数单调递减，总成绩不超过 N，同时记录约数的个数即可。最后利用 **引理11.3.1** 找到约数个数最多的数里最小的那个数即可。

我们可以把当前走到每一个素数前面的时候列举成一棵树的根节点，然后一层层的去找。找到什么时候停止呢?

1. 当前走到的数字已经大于我们想要的数字了
2. 当前枚举的因子已经用不到了
3. 当前因子大于我们想要的因子了
4. 当前因子正好是我们想要的因子（此时判断是否需要更新最小 ans）

然后 dfs 里面不断一层一层枚举次数继续往下迭代

- **竞赛例题选讲**

Problem A Number With The Given Amount Of Divisors（CF27E）

给定一个正整数 n ，输出最小的整数，满足这个整数有 n 个因子，即求因子数一定的最小反素数。

Solution

对于这种题，我们只要以因子数为 dfs 的返回条件基准，不断更新找到的最小值就可以了

Code

```
#include <stdio.h>
#define ULL unsigned long long
#define INF ~0ULL
ULL p[16] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53};
ULL ans;
ULL n;
// depth: 当前在枚举第几个素数。num: 当前因子数。
// temp: 当前因子数量为 num
// 的时候的数值。up: 上一个素数的幂，这次应该小于等于这个幂次嘛
void dfs(ULL depth, ULL temp, ULL num, ULL up) {
    if (num > n || depth ≥ 16) return;
    if (num == n && ans > temp) {
```

```
        ans = temp;
        return;
    }
    for (int i = 1; i ≤ up; i++) {
        if (temp / p[depth] > ans) break;
        dfs(depth + 1, temp = temp * p[depth], num * (i + 1), i);
    }
}
int main() {
    while (scanf("%llu", &n) ≠ EOF) {
        ans = INF;
        dfs(0, 1, 1, 64);
        printf("%llu\n", ans);
    }
  return 0;
}
```

Problem B More Divisors（ZOJ 1562）

求 n 以内因子数最多的数

Solution

思就是求 $1 \sim n$ 内的反素数嘛，改改 `dfs` 的返回条件就行了。注意题目的数据范围， 用 `int` 会溢出，在循环里可能就出不来了就超时了。

Code

```
#include <cstdio>
#include <iostream>
#define ULL unsigned long long
int p[16] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53};
ULL n;
ULL ans, ans_num;  // ans 为 n 以内的最大反素数（会持续更新），ans_sum 为 ans
                   // 的因子数。
void dfs(int depth, ULL temp, ULL num, int up) {
    if (depth ≥ 16 || temp > n) return;
    if (num > ans_num) {
        ans = temp;
        ans_num = num;
    }
    if (num == ans_num && ans > temp) ans = temp;
    for (int i = 1; i ≤ up; i++) {
        if (temp * p[depth] > n) break;
      dfs(depth + 1, temp *= p[depth], num * (i + 1), i);
    }
    return;
}
int main() {
    while (scanf("%llu", &n) ≠ EOF) {
        ans_num = 0;
        dfs(0, 1, 1, 60);
        printf("%llu\n", ans);
    }
    return 0;
}
```

0x12 Z^* 与 $(Z_p^*, \cdot)$结构

Z^*，即正整数集。

$(Z_p^*, \cdot)$， Z_p 的剩余类群，即 $(1, 2, \ldots, p-1)$，与 p 互素的数，即 $U（Z_p）$

0x12.1 唯一分解定理（算数基本定理）

任何一个大于 1 的数都可以被分解成有限个质数乘积的形式
$n=\prod_{i=1}^{m}p_i^{C^i}=p_1^{C_1}\times p_2^{C_i}\times\cdots\times P_n^{C_n}$
其中 $p_1<p_2<\cdots<p_m$ 为质数，C_i 为正整数。

> **唯一分解定理证明：**
> （1）**大于1的自然数必可写成质数之积**
> 使用**反证法**：假设存在大于1的自然数不能写成质数的乘积，把最小的那个称为 n。
>
> 自然数可以根据其可除性（是否能表示成两个不是自身的自然数的乘积）分成3类：质数、合数和1。首先，按照定义，n 大于1。其次，n 不是质数，因为质数 p 可以写成质数乘积：$p=p$，这与假设不相符合。因此n只能是合数，但每个合数都可以分解成两个严格小于自身而大于1的自然数的积。设 $n=a\times b$，其中 a 和 b 都是介于 1 和 n 之间的自然数，因此，按照 n 的定义，a 和 b 都都可以写成质数的乘积。从而 $n=a\times b$ 也可以写成质数的乘积。由此产生矛盾。因此大于 1 的自然数必可写成质数的乘积。
> （2）**证明质数分解唯一性**
> 同样使用 **反证法**：设存在一个能分解为两种根本不同的质数乘积的正整数，则其中必有一最小的（最小数原理），设 $m=p_1p_2\dots p_r=q_1q_2\dots q_s$，从小到大排序。$p_1$ 不等于 q_1，否则可消去，不满足最小假设。设 $p_1<q_1$。构造一整数
> $m'=m-(p_1q_2\dots q_s)=(p_1p_2\dots p_r)-(p_1q_2\dots q_s)=p_1(p_2\dots p_r-q_2\dots q_s)=(q_1q_2\dots q_s)-(p_1q_2\dots q_s)=(q_1-p_1)q_2\dots q_s$
>
> 由于 $p_1<q_1$，m'是一个正整数，且$<m$。因此 m' 的质数分解，除了因子次序外，必须是唯一的（m 是最小数假设）。从上面表达式可看出，p_1是 m' 的因子，再根据 m' 质数分解的唯一性得出（推论、用反证法很容易得到），p_1必须是 (q_1-p_1) 或者 $q_2\dots q_s$ 的因子，由于 q都比 p_1 大，这后一情形是不可能的，这样就有某个整数 h，使 $q_1-p_1=p_1*h$ 或 $q_1=p_1(h+1)$。这表明 p_1 是 q_1 的一个因子，与 q_1 是质数相矛盾。故得证。

显然 n 最多仅有一个大于 $\sqrt{n}$ 的质因子（若有两个的话，他们的乘积就大于 n 了）。

- **试除法**

类似埃式筛，我们直接枚举因子然后把当前因子全部除尽即可，时间复杂度 $O(\sqrt{n})$。

```
int c[N],p[N];
void divide(int n) {
    cnt = 0;
    for(int i = 2; i * i ≤ n; ++ i) {
        if(n % i == 0) {
            p[ ++ cnt] = i,c[cnt] = 0;
            while(n % i == 0) n /= i,c[cnt] ++ ;
        }
    }
    if(n > 1)//如果n是质数
        p[ ++ cnt] = n,c[cnt] = 1;
    for(int i = 1;i ≤ cnt; ++ i)
        cout << p[i] << "^" << c[i] << endl;
}
```

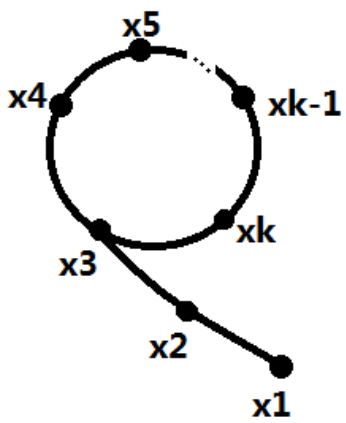
- **Pollard Rho 算法**

对于数据较大的情况，如 $n\geq 10^{18}$，有用来分解其因数的 Pollard Rho 算法。

`Pollard-rho` 算法是一个大数分解的**随机算法**，能够在 $O(n^{\frac{1}{4}})$ 的时间内找到 n 的一个素因子 p，然后再递归计算 $n'=\frac{n}{p}$，直到 n 为素数为止，通过这样的方法将 n 进行素因子分解。

`Pollard-rho` 的策略为：从 $[2,n)$ 中随机选取 k 个数x_1、x_2、x_3、$\dots$、x_k，求任意两个数x_i、x_j的差和 n 的最大公约数，即 $d=\gcd(x_i-x_j,n)$，如果 $1<d<n$，则 d 为 n 的一个因子，直接返回 d 即可。

然后来看如何选取这 k 个数，我们采用生成函数法，令 $x_1=rand()\%(n-1)+1$，$xi=(x_i-1^2+1)\mod n$，很明显，这个序列是有循环节的，如图所示：



我们需要做的就是在它进入循环的时候及时跳出循环，因为 x_1 是随机选的，x_1 选的不好可能使得这个算法永远都找不到 n 的一个范围在 $(1,n)$ 的因子，这里采用步进法，保证在进入环的时候直接跳出循环。

【模板】Pollard-Rho算法（Luogu P4718）

T组数据，每组数据输入一个数 n ，检验 n 是否是质数，是质数就输出 `Prime` 。如果不是质数，输出它的最大质因子。

$1 \leq T \leq 350, 1 \leq n \leq 10^{18}$

Solution

模板题，大数质因数分解后，比较每一个分解出来的质因子，找到最大质因子即可。

```cpp
typedef long long ll;
const int N = 1e5 + 7;
ll x, y, a[N];
ll max_factor;
struct BigIntegerFactor {
    const static int N = 1e6 + 7;
    ll prime[N], p[N], fac[N], sz, cnt; //多组输入注意初始化cnt = 0
    inline ll mul(ll a, ll b, ll mod) {          //WA了尝试改为__int128或慢速乘
        if (mod ≤ 1000000000)
            return a * b % mod;
        return (a * b - (ll)((long double)a / mod * b + 1e-8) * mod + mod) % mod;
    }
    void init(int maxn) {
        int tot = 0;
        sz = maxn - 1;
        for (int i = 1; i ≤ sz; ++i)
            p[i] = i;
        for (int i = 2; i ≤ sz; ++i) {
            if (p[i] == i)
                prime[tot++] = i;
            for (int j = 0; j < tot && 1ll * i * prime[j] ≤ sz; ++j) {
                p[i * prime[j]] = prime[j];
                if (i % prime[j] == 0)
                    break;
            }
        }
    }
    ll qpow(ll a, ll x, ll mod) {
        ll res = 1ll;
        while (x) {
            if (x & 1)
                res = mul(res, a, mod);
            a = mul(a, a, mod);
            x >>= 1;
        }
        return res;
    }
```

```cpp
bool check(ll a, ll n) {                        //二次探测原理检验n
    ll t = 0, u = n - 1;
    while (!(u & 1))
        t++, u >>= 1;
    ll x = qpow(a, u, n), xx = 0;
    while (t--) {
        xx = mul(x, x, n);
        if (xx == 1 && x ≠ 1 && x ≠ n - 1)
            return false;
        x = xx;
    }
    return xx == 1;
}
bool miller(ll n, int k) {
    if (n == 2)
        return true;
    if (n < 2 || !(n & 1))
        return false;
    if (n ≤ sz)
        return p[n] == n;
    for (int i = 0; i ≤ k; ++i) {               //测试k次
        if (!check(rand() % (n - 1) + 1, n))
            return false;
    }
    return true;
}
inline ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}
inline ll Abs(ll x) {
    return x < 0 ? -x : x;
}
ll Pollard_rho(ll n) {                          //基于路径倍增的Pollard_Rho算法
    ll s = 0, t = 0, c = rand() % (n - 1) + 1, v = 1, ed = 1;
    while (1) {
        for (int i = 1; i ≤ ed; ++i) {
            t = (mul(t, t, n) + c) % n;
            v = mul(v, Abs(t - s), n);
            if (i % 127 == 0) {
                ll d = gcd(v, n);
                if (d > 1)
                    return d;
            }
        }
        ll d = gcd(v, n);

        if (d > 1)
            return d;
        s = t;
        v = 1;
        ed <<= 1;
    }
}
void getfactor(ll n) {                          //得到所有的质因子(可能有重复的)
    if (n ≤ sz) {
        while (n ≠ 1)
            fac[cnt ++ ] = p[n], n /= p[n];
        max_factor = max_factor > p[n] ? max_factor : p[n];
        return;
```

```
        }
        if (miller(n, 6)) {
            fac[cnt ++ ] = n;
            max_factor = max_factor > n ? max_factor : n;
        }
        else {
            ll d = n;
            while (d ≥ n)
                d = Pollard_rho(n);
            getfactor(d);
            getfactor(n / d);
        }
        return ;
    }
} Q;
int main() {
    //Q.init(N - 1);//如果代码超时且仅需要分解大数的质因数可以用这句话，否则不要用
    ll T, n;
    scanf("%lld", &T);
    while (T--) {
        max_factor = -1;
        scanf("%lld", &n);
        Q.getfactor(n);
        if(max_factor == n)
            puts("Prime");
        else printf("%lld\n", max_factor);
    }
    return 0;
}
```

- **竞赛例题选讲**

Problem A 阶乘分解（AcWing 197）

给定整数 N ，试把阶乘 $N!$ 分解质因数，按照算术基本定理的形式输出分解结果中的 p_i 和 c_i 即可。如：$5! = 120 = 2^3 \times 3 \times 5$

Solution

我们发现 $N!$ 中质数因子 p 的个数,就是 $1 \sim N$ 中每个数含有的质因数 p 个数之和。既然如此的话,那么我们发现， $1 \sim N$ 中，p 的倍数，即至少有一个质因子 p 的数显然有$\lfloor \frac{N}{p} \rfloor$个，而$p^2$的倍数，即至少有两个质因子$p$数的显然是有$\lfloor \frac{N}{p^2} \rfloor$，不过由于这两个质因子 p 中有一个已经在$\lfloor \frac{N}{p} \rfloor$统计过了，所以只需要再统计第二个质因子，也就是直接累加$\lfloor \frac{N}{p^2} \rfloor$，而不是累乘。

即：

$$\lfloor \frac{N}{p} \rfloor + \lfloor \frac{N}{p^2} \rfloor + \cdots + \lfloor \frac{N}{p^{\log_p n}} \rfloor = \sum_{p^k \le N} \lfloor \frac{N}{p^k} \rfloor$$

时间复杂度 $O(N \log N)$

```cpp
int main()
{
    int n;
    cin >> n;
    init(n);//线性筛代码略去
    for (int i = 0; i < cnt; i ++ ) {
        int p = primes[i];
        int s = 0;
        for (int j = n; j; j /= p) s += j / p;
        printf("%d %d\n", p, s);
    }
    return 0;
}
```

由**Problem A 阶乘分解** 就可以引申出一道趣味题：

Problem B 0的个数

n 的阶乘 $n!$ 中的末尾有多少个 0？

Solution

通过了Problem A 阶乘分解我们已经知道了如何求得 $n!$ 中的质因子个数，所以这道题我们考虑如何应用一些。我们可以从 **“哪些质因子相乘可以得到10”** 的角度出发，发现只有 2 和 5 相乘的时候才会得到 10 。假设 $n!=2^a\times3^b\times5^c\cdots$ 。我们发现每一对 2 和 5 相乘就可以得到一个 10，所以 10 的个数 $num=min\{a,c\}$ 。由于 a 一定小于 c ，所以 $num=c$ 。

所以我们直接求出来 $n!$ 质因数分解后 5 的次数即可。

Code

```cpp
int number_0(int n)
{
    int num = 0;

    while(n) {
        num += n / 5;
        n /= 5;
    }
    return num;
}
```

这样又可以引申出一道题目：求 $n!$ 的二进制表示中最低位 1 中的位置。

Solution

求 $n!$ 的二进制表示中最低位 1 中的位置，我们发现实际上就是求二进制下最低位 1 后面 0 的个数。在所有小于 n 的数中，2 的倍数贡献一个 0，4 的倍数再贡献一个 0，所以答案就是 $n!$ 质因数分解后 2 的次数。

Problem C Divisors of the Divisors of an Integer（2018-2019 ACM-ICPC, Asia Dhaka Regional ContestC）

给出 n，问 $n!$ 的因子的因子的个数和。

Solution

学会上面的阶乘分解之后，我们能一眼看出来这道题也一定跟它有关系，所以我们按照惯例先对 $n!$ 进行质因数分解。

$n!=p_1^{\alpha_1}\times p_2^{\alpha_2}\times\cdots\times p_k^{\alpha_k}$ 。

我们单独考虑每一中质数，我们假设一个素数为 p ，它的幂次 α ，因为我们要求的是因子的因子个数，因为它一共有幂次 α ，那么该因子的因子就有 $0,1,2,\cdots,\alpha$ 的 $\alpha+1$ 种选择。

即：$p^0,p^1,p^2,p^3,\cdots,p^\alpha$，对于 p^0 而言，因子个数为 1 ，对于 p^1 而言，因子个数为 2 ，对于 p^α 而言，因子个数为 $\alpha+1$ ，总的因子个数为： $1+2+3+\cdots+\alpha+1=\frac{(\alpha+1)(\alpha+2)}{2}$。（等差数列求和公式）

对于一个素数的贡献如上，那么总的贡献为：$\prod_{i=1}^{k} \frac{(\alpha_i+1)(\alpha_i+2)}{2}$。

我们由 **Problem A 阶乘分解** 知道了求解 α 的方法，所以使用公式计算答案即可。

Code

```cpp
const ll mod = 1e7 + 7;
int n, m;
int primes[N], cnt;
bool vis[N];
void init(int n)
{
    for(int i = 2; i ≤ n; ++ i) {
        if(vis[i] == 0) primes[ ++ cnt] = i;
        for(int j = 1; j ≤ cnt && primes[j] * i ≤ n; ++ j) {
            vis[i  * primes[j]] = true;
            if(i % primes[j] == 0) break;
        }
    }
}
ll f(ll x)
{
    ll res = 0;
    ll tmp = n;
    while(tmp) {
        res += tmp / x;
        tmp /= x;
    }
    return res;
}
int main()
{
    init(N - 1);
    while(scanf("%d", &n) ≠ EOF && n) {
        ll res = 1;
        for(int i = 1; i ≤ cnt && primes[i] ≤ n; ++ i) {
            ll alpha = f(primes[i]);
            res = (res * ((alpha + 1) * (alpha + 2)) / 2) % mod;
        }
        printf("%lld\n", res);
    }
    return 0;
}
```

Problem D Multiply（2019 ACM- ICPC Asia Xuzhou Regional Contest E）

给 n 个数的序列 a，定义 $Z = \prod_{i=1}^{n}(a_i!)$。给出 X，Y，若 $b_i = Z \times X^i$，求出最大的 i 使得 $b_i \mid Y!$。

Solution

首先按照套路先将 X 质因数分解得 $X = p_1^{\alpha_1}, p_2^{\alpha_2}, p_3^{\alpha_3} \cdots p_n^{\alpha_n}$。对于某质因子 p_i ，设 Z 中有 a 个， X 中有 b 个， $Y!$ 中有 c 个。因为 $b_i = Z \times X^i$ ， Z 是一个常数，所以我们很自然地想到先去掉这个常数。

$$b_i \mid Y! \Rightarrow Z \times X^i \mid Y! \Rightarrow X^i \mid \frac{Y!}{Z}$$

即该质因子 p_i 最多对应 $\frac{c-a}{b}$ 个，我们只需要贪心地求出所有质因子中最小的那个 i，即为答案。

Code

```cpp
typedef long long ll;
const int N = 1e5 + 7;
ll x, y, a[N];
ll max_factor;
```

```cpp
struct BigIntegerFactor {
    const static int N = 1e6 + 7;
    ll prime[N], p[N], fac[N], sz, cnt; //多组输入注意初始化cnt = 0
    inline ll mul(ll a, ll b, ll mod) {          //WA了尝试改为__int128或慢速乘
        if (mod ≤ 1000000000)
            return a * b % mod;
        return (a * b - (ll)((long double)a / mod * b + 1e-8) * mod + mod) % mod;
    }
    void init(int maxn) {
        int tot = 0;
        sz = maxn - 1;
        for (int i = 1; i ≤ sz; ++i)
            p[i] = i;
        for (int i = 2; i ≤ sz; ++i) {
            if (p[i] == i)
                prime[tot++] = i;
            for (int j = 0; j < tot && 1ll * i * prime[j] ≤ sz; ++j) {
                p[i * prime[j]] = prime[j];
                if (i % prime[j] == 0)
                    break;
            }
        }
    }
    ll qpow(ll a, ll x, ll mod) {
        ll res = 1ll;
        while (x) {
            if (x & 1)
                res = mul(res, a, mod);
            a = mul(a, a, mod);
            x >>= 1;
        }
        return res;
    }
    bool check(ll a, ll n) {                    //二次探测原理检验n
        ll t = 0, u = n - 1;
        while (!(u & 1))
            t++, u >>= 1;
        ll x = qpow(a, u, n), xx = 0;
        while (t--) {
            xx = mul(x, x, n);
            if (xx == 1 && x ≠ 1 && x ≠ n - 1)
                return false;
            x = xx;
        }
        return xx == 1;
    }
    bool miller(ll n, int k) {
        if (n == 2)
            return true;
        if (n < 2 || !(n & 1))
            return false;
        if (n ≤ sz)
            return p[n] == n;
        for (int i = 0; i ≤ k; ++i) {           //测试k次
            if (!check(rand() % (n - 1) + 1, n))
                return false;
        }
        return true;
    }
```

```cpp
inline ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}
inline ll Abs(ll x) {
    return x < 0 ? -x : x;
}
ll Pollard_rho(ll n) {                  //基于路径倍增的Pollard_Rho算法
    ll s = 0, t = 0, c = rand() % (n - 1) + 1, v = 1, ed = 1;
    while (1) {
        for (int i = 1; i ≤ ed; ++i) {
            t = (mul(t, t, n) + c) % n;
            v = mul(v, Abs(t - s), n);
            if (i % 127 == 0) {
                ll d = gcd(v, n);
                if (d > 1)
                    return d;
            }
        }
        ll d = gcd(v, n);

        if (d > 1)
            return d;
        s = t;
        v = 1;
        ed <<= 1;
    }
}
void getfactor(ll n) {                          //得到所有的质因子(可能有重复的)
    if (n ≤ sz) {
        while (n ≠ 1)
            fac[cnt ++ ] = p[n], n /= p[n];
        max_factor = max_factor > p[n] ? max_factor : p[n];
        return;
    }
    if (miller(n, 6)) {
        fac[cnt ++ ] = n;
        max_factor = max_factor > n ? max_factor : n;
    }
    else {
        ll d = n;
        while (d ≥ n)
            d = Pollard_rho(n);
        getfactor(d);
        getfactor(n / d);
    }
    return ;
}
ll cal(ll n, ll x) {                            //计算 n! 中质因子 x 的数量
    ll num = 0;
    while (n) {
        num += n / x;
        n = n / x;
    }
    return num;
}
ll solve(int n, ll x, ll y) {
    map<ll, ll> mp;
    ll ans = 4e18;
    cnt = 0;
```

```
        getfactor(x);
        for (int i = 0; i < cnt; ++i)
            mp[fac[i]]++;
        map<ll, ll>::iterator it = mp.begin();
        while (it ≠ mp.end()) {
            ll num = 0;
            for (int i = 1; i ≤ n; ++i) {
                num += cal(a[i], it→first);
            }
            ans = min(ans, (cal(y, it→first) - num) / it→second);
            it ++ ;
        }
        return ans;
    }
} Q;
int main() {
    //Q.init(N - 1);//如果代码超时且仅需要分解大数的质因数可以用这句话，否则不要用
    int T, n;
    scanf("%d", &T);
    while (T--) {
        scanf("%d %lld %lld", &n, &x, &y);
        for (int i = 1; i ≤ n; ++i)
            scanf("%lld", a + i);
        printf("%lld\n", Q.solve(n, x, y));
    }
    return 0;
}
```

0x12.2 Z^*结构中的一些定理

推论12.2.1:

若

$n = p_1^{\alpha_1} \times p_2^{\alpha_2} \times p_3^{\alpha_3} \times \cdots \times p_k^{\alpha_k}$

$m = p_1^{\beta_1} \times p_2^{\beta_2} \times p_3^{\beta_3} \times \cdots \times p_k^{\beta_k}$

则

$n \times m = p_1^{\alpha_1+\beta_1} \times p_2^{\alpha_2+\beta_2} \times p_3^{\alpha_3+\beta_3} \times \cdots \times p_k^{\alpha_k+\beta_k}$

$\gcd(n, m) = p_1^{\min\{\alpha_1,\beta_1\}} \times p_2^{\min\{\alpha_1,\beta_1\}} \times \cdots \times p_k^{\min\{\alpha_k,\beta_k\}}$

$\operatorname{lcm}(n, m) = p_1^{\max\{\alpha_1,\beta_1\}} \times p_2^{\max\{\alpha_1,\beta_1\}} \times \cdots \times p_k^{\max\{\alpha_k,\beta_k\}}$

定理12.2.2: ： $(p-1)! + 1 \equiv 0 \ (\mod p)$

定理12.2.3: $((n+1)(n+2)\ldots(n+k))\%k \neq 0$

0x12.3 $(Z_p^*, \cdot)$结构

定理12.3.1: $(Z_p^*, \cdot)$是循环群，即存在$a \in Z_p^*$，使得$Z_p^* = \{a^n | n = 1, 2, \cdots, p-1\}$

这样的 a 称为 p 的原根。

素数一定有原根，原根不唯一，部分合数也有**原根**

- 1000000007的原根为5
- 998244353的原根为 3

原根详见**0x60原根**。

0x13 最大公因数与最小公倍数

0x13.1 约数

约数，又称因数。整数a除以整数$b(b \neq 0)$除得的商正好是整数而没有余数，我们就说a能被b整除，或b能整除a。a称为b的倍数，b称为a的约数。

唯一分解定理，任何一个大于 1 的数都可以被分解成有限个质数乘积的形式
$N = \prod_{i=1}^{m} p_i^{C^i}$
其中 $p_1 < p_2 < \cdots < p_m$ 为质数，C_i 为正整数

N 的正约数个数为：

$(c_1 + 1) \times (c_2 + 1) \times \cdots (c_m + 1) = \prod_{i=1}^{m}(c_i + 1)$

N^M 的正约数个数为$(M \times c_1 + 1) \times (M \times c_2 + 1) \times \cdots (M \times c_m + 1) = \prod_{i=1}^{m}(M \times c_i + 1)$

N 的所有正约数和为：

$(1 + p_1 + p_1^2 + \cdots + p_1^{c_1}) \times \cdots \times (1 + p_m + p_m^2 + \cdots + p_m^{c_m}) = \prod_{i=1}^{m}(\sum_{j=0}^{c_i}(p_i)^j)$

随机数据下，约数个数的期望是 $O(\ln\ n)$

- **试除法 - 求 n 的正约数集合**

  显然约数总是成对出现（除了完全平方数，只有一个$\sqrt{n}$），所以只需要枚举到$\sqrt{n}$ 即可。

```
vector<int>factor;
int m;
int main()
{
    scanf("%d", &n);
    for(int i = 1; i * i ≤ n; ++ i){
        if(n % i == 0){
            factor.push_back(i);
            if(i =≠ n / i)
                factor.push_back(n / i);
        }
    }
    for(int i = 0; i < factor.size(); ++ i){
        printf("%d\n", factor[i]);
    }
    return 0;
}
```

推论：一个整数 n 的约数个数上界为 $2\sqrt{n}$。

- **倍数法 - 求 $1 \sim n$ 中每个数的正约数集合**

按照埃氏筛的形式枚举倍数，时间复杂度为 $O(n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}) \approx n \times \log n = O(n \log n)$

时间复杂度为$O(n \log n)$

可以求出求 $1 \sim n$ 中每个数的正约数集合，但并不能求出具体某个数的因子是谁，常用于一些于因子有关的计算，如计算 $\sum_{i=1}^{n}\sum_{d|n} d$

或是 $\sum_{i=1}^{n}\sum_{d|n} f(d)$，可以使用倍数法在 $O(n \log n)$ 的复杂度下计算。

```
int main()
{
    scanf("%d", &n);
    for(int i = 1; i ≤ n; ++ i){
        for(int j = 1 ;j * i ≤ n; ++ j){
            factor[i * j].push_back(j);
```

```
        }
    }
    for(int i = 1; i ≤ n; ++ i){
        cout << i << ": ";
        for(int j = 0; j < factor[i].size(); ++ j)
            printf("%d ", factor[i][j]);
        puts("");
    }
    return 0;
}

```

推论：$1 \sim n$ 中每个数的约数的总和大概为 $n \log n$。

- **竞赛例题选讲**

Problem A 约数之和（AcWing 97）

给定正整数 A，B，求 A^B 的所有约数之和 $\mod 9901 (1 \leq A, B, \leq 5 \times 10^7)$。

Solution

根据唯一分解定理将A进行因式分解可得:$A = p_1^{a_1} \times p_2^{a_2} \times \cdots \times p_n^{a_n}$.
$A^B = p1^{a_1 \times B} \times p_2^{a_2 \times B} \times \cdots \times p_n^{a_n \times B}$

$A^{B的所有约数之和sum} = (1 + p_1 + p_1^2 + \cdots + p_1^{a_1 \times B}) \times (1 + p_2 + p_2^2 + \cdots + p_2^{a_2 \times B}) \times \cdots \times (1 + p_n + p_n^2 + \cdots + p_n^{a_n \times B})$

上式中每一项都是一个等比数列

根据等比数列求和公式：

$S_n = n \times a_1 \ (q = 1)$

$S_N = a_1 \times \frac{1 - q^n}{1 - q} = \frac{a_1 - a_n \times q}{1 - q} \ (q \neq 1)$

$(1 + p_1 + p_1^2 + \cdots + p_1^{a_1 \times B}) = \frac{1 - p_1^{a_1 \times B + 1}}{1 - p_1} = \frac{p_1^{a_1 \times B + 1} - 1}{p_1 - 1}$

我们可以用快速幂计算分子 $p_1^{a_1 \times B + 1} - 1 \ (\mod 9901)$，以及分母 $p_1 - 1 \ (mod 9901)$。

因为 9901 是质数，所以只要 $p_1 - 1$ 不是 9901 的倍数，我们就可以求分母的逆元用乘逆元的形式来代替除法，这样我们直接暴力计算整个式子即可。

特别地，若 $p_1 - 1$ 是 9901 的倍数，则 $p_1 - 1 \equiv 0 \ (\mod 9901)$，此时的乘法逆元不存在，但是我们可以得到 $p_1 \equiv 1 \ (\mod 9901)$，所以 $(1 + p_1 + p_1^2 + \cdots + p_1^{a_1 \times B}) \equiv 1 + 1 + 1^2 + \cdots + 1^{a_1 \times B} \equiv a_1 \times B + 1 \ (\mod 9901)$。

Code

```
const int N = 507, M = 500007, INF = 0x3f3f3f3f;
const double eps = 1e-6;
ll a, b, m, mod = 9901;
ll ans = 1;
int p[N], c[N];
int cnt, n;
void divide(int n)//质因子分解
{
    cnt = 0;
    for(int i = 2; i * i ≤ n; ++ i) {
        if(n % i == 0){
            p[ ++ cnt] = i, c[cnt] = 0;
            while(n % i == 0)n /= i, c[cnt] ++ ;
        }
    }
    if(n > 1)
        p[++ cnt] = n, c[cnt] = 1;
```

```cpp
}
ll qpow(ll a, ll b)
{
    ll res = 1;
    while(b) {
        if(b & 1)res = (res * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return res;
}
int main()
{
    scanf("%lld%lld", &a, &b);
    if(a == 0){puts("0");return 0;}//数据有毒啊
    divide(a);
    for(int i = 1; i ≤ cnt; ++ i) {
        //cout << ans << endl;
        if((p[i] - 1) % mod == 0){//没有逆元，需要特判
            ans = (b * c[i] + 1) % mod * ans % mod;
            continue;
        }
        ll x = qpow(p[i], (ll)b * c[i] + 1);//分子
        x = (x - 1 + mod) % mod;
        ll y = p[i] - 1;//分母
        y = qpow(y, mod - 2);//费马小定理求乘法逆元
        ans = ans * x % mod * y % mod;
    }
    printf("%lld\n", ans);
}
```

Problem B. A New Function (LightOJ-1098)

定义约数和 $\sigma(n)$ 为对于每个数求除 1 和它本身的约数和， 求 $S(n) = \sum_{i=1}^{n} \sigma(i)$。

$n \leq 2 \times 10^9$

Solution

显然有：

$$S(n) = \sum_{i=2}^{n} i \times (\left\lfloor \frac{n}{i} \right\rfloor - 1)$$

即对于 $1 \sim n$ 内的数 i，含有因子 i 的数的个数为 $\left\lfloor \frac{n}{i} \right\rfloor$，除去它本身，显然有 $\left\lfloor \frac{n}{i} \right\rfloor - 1$ 个，因为要除去 1，从 2 开始枚举，整除分块即可。

$O(\sqrt{n})$

Code

```cpp
#define int long long
int n, m, s, t, k, ans, a[N], kcase;
void solve()
{
    ans = 0;
    scanf("%lld", &n);
    for (int l = 2, r; l ≤ n; l = r + 1) {
        r = n / (n / l);
        ans += (l + r) * (r - l + 1) / 2 * (n / l - 1);
    }
    cout << ans << endl;
}
```

```cpp
signed main()
{
    scanf("%lld", &t);
    while(t -- ) {
        printf("Case %lld: ", ++ kcase);
        solve();
    }
    return 0;
}
```

0x13.2 最大公约数

两个数 a 和 b 的**最大公约数 (GreatestCommonDivisor)** 是指同时整除 a 和 b 的最大因数，记为 $\gcd(a,b)$。

> **一个约定俗成的定理**：任何非零整数和零的最大公约数为它本身。

有如下基本性质：

性质13.2.1：$\gcd(a,b)=\gcd(b,a)$

性质13.2.2：$\gcd(a,b)=\gcd(a-b,b)(a\geq b)$

性质13.2.3：$\gcd(a,b)=\gcd(a \bmod b,b)$

性质13.2.4：$\gcd(a,b,c)=\gcd(\gcd(a,b),c)$

性质13.2.5：$\gcd(ka,kb)=k\ \gcd(a,b)$

性质13.2.6：$\gcd(k,ab)=1 \iff \gcd(k,a)=1\ \&\&\ \gcd(k,b)=1$

特别地，如果 a,b 的$\gcd(a,b)=1$，则称这两个数互质（互素）。

- **辗转相除法（又称欧几里德算法）**

$\forall a,b\in\mathbb{N},b\neq 0,\gcd(a,b)=\gcd(b,a \bmod b)$

> **欧几里德算法证明**
>
> $a=kb+r=kb+a\%b$，则$a\%b=a-kb$。令 d 为 a 和 b 的公约数，则 $d\mid a$ 且 $d\mid b$ 根据整除的组合性原则，有 $d\mid(a-kb)$，即 $d\mid(a\%b)$。
>
> 这就说明如果 d 是 a 和 b 的公约数，那么 d 也一定是 b 和 $a\%b$ 的公约数，即两者的公约数是一样的，所以最大公约数也必定相等。

时间复杂度 $O(\log n)$

最坏情况：斐波那契数列相邻的两项，因为斐波那契数列相邻的两项一定互质。

欧几里德算法由于存在大量的取模运算，对于大整数耗时较大。

Code

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

- **更相减损术**：$\gcd(n,m)=\gcd(n,n-m)$

> **证明1：**
>
> 设 $\gcd(n,n-m)=d\quad$ 令 $n=k_1\times d,\ n-m=k_2\times d,\gcd(k_1,k_2)=1\quad\Rightarrow m=(k_1-k_2)\times d$
> $\therefore \gcd(n,m)=\gcd(k_1\times d,(k_1-k_2)\times d)=d=\gcd(n,n-m)$
> 性质得证 □
>
> **证明2：**
> 由 **性质00.1.5** 可知，n,m 的公约数集合与 $n,n-m$ 的公约数集合完全一样，故最大公约数自然也相等。
> 性质得证 □

- **Stein 算法（可看作更相减损术的应用）**

渐近时间，空间复杂度均与欧几里德算法相同。
原理：$\gcd(ka, kb) = k \times \gcd(a, b)$
最大特点：只有移位和加减法计算，避免了大整数的取模运算。

- **递归版**

```cpp
int stein(int a, int b) {
    if (a < b)
        a ^= b, b ^= a, a ^= b; //交换，使a为较大数；
    if (b == 0)
        return a;                    //当相减为零，即两数相等时，gcd=a；
    if ((!(a & 1)) && (!(b & 1)))
        return stein(a >> 1, b >> 1) << 1; //s1,注意最后的左移，在递归返回过程中将2因子乘上；
    else if ((a & 1) && (!(b & 1)))
        return stein(a, b >> 1);         //s2;
    else if ((!(a & 1)) && (b & 1))
        return stein(a >> 1, b);
    else
        return stein(a - b, b);                                    //s3;
}
```

- **迭代版**

```cpp
int stein(int a, int b) {
    int k = 1;
    while ((!(a & 1)) && (!(b & 1))) { //s1;
        k <<= 1;                          //用k记录全部公因子2的乘积 ；
        a >>= 1;
        b >>= 1;
    }
    while (!(a & 1)) a >>= 1;            //s2;
    while (!(b & 1)) b >>= 1;
    if (a < b) a ^= b, b ^= a, a ^= b; //交换，使a为较大数；
    while (a != b) {                       //s3;
        a -= b;
        if (a < b)
            a ^= b, b ^= a, a ^= b;
    }
    return k * a;
}
```

- **竞赛例题选讲**

Problem A 永远永远

$f[0] = 0$，当 $n > 1$ 时，$f[n] = (f[n-1] + a)\%b$，给定 a 和 b，问是否存在一个自然数 k （$0 \le k < b$），是 $f[n]$ 永远都取不到的。

Solution

我们发现这里的 $f[\cdots]$ 一定是有循环节的，如果在某个循环节内都无法找到那个自然数 k，那么必定是永远都找不到了。

求出 $f[n]$ 的通项公式，为 $f[n] = an\%b$，令 $an = kb + r$，那么这里的 $r = f[n]$，如果 $t = \gcd(a, b)$，$r = an - kb = t((\frac{a}{t})n - (\frac{b}{t})k)$，则有 $t \mid r$，要满足所有的 r 使得 $t \mid r$，只有当 $t = 1$ 的时候，于是这个问题的解也就出来了，只要求 a 和 b 的 $\gcd$，如果 $\gcd(a, b) > 1$，则存在一个 k 使得 $f[n]$ 永远都取不到，直观的理解是当 $\gcd(a, b) > 1$，那么 $f[n]$ 不可能是素数。

一个简单的应用就是本文 **0x22.1** 的 **Problem A Fox And Jumping**。

0x13.3 最小公倍数

两个数 a 和 b 的**最小公倍数**（LeatestCommonMultiple）是指同时被 a 和 b 整除的最小倍数，记为 $\text{lcm}(a,b)$。特殊的，当 a 和 b 互素时，$\text{lcm}(a,b)=ab$。

性质13.3.1：$\forall a,b\in N,\gcd(a,b)\times\text{lcm}(a,b)=a\times b$

> 可以使用 $\gcd,\text{lcm}$ 的定义证明 **性质13.3.1** ，证明略。

```cpp
int lcm(int a,int b){
    return a / gcd(a,b) * b;//先除后乘，以免溢出64位整数
}
```

- **重要性质：$\gcd$ 与 lcm 的指数最值表示法**

由唯一分解定理得，若

$n=p_1^{\alpha_1}\times p_2^{\alpha_2}\times p_3^{\alpha_3}\times\cdots\times p_k^{\alpha_k}$

$m=p_1^{\beta_1}\times p_2^{\beta_2}\times p_3^{\beta_3}\times\cdots\times p_k^{\beta_k}$

则

$n\times m=p_1^{\alpha_1+\beta_1}\times p_2^{\alpha_2+\beta_2}\times p_3^{\alpha_3+\beta_3}\times\cdots\times p_k^{\alpha_k+\beta_k}$

$\gcd(n,m)=p_1^{min\{\alpha_1,\beta_1\}}\times p_2^{min\{\alpha_1,\beta_1\}}\times\cdots\times p_k^{min\{\alpha_k,\beta_k\}}$

$\text{lcm}(n,m)=p_1^{max\{\alpha_1,\beta_1\}}\times p_2^{max\{\alpha_1,\beta_1\}}\times\cdots\times p_k^{max\{\alpha_k,\beta_k\}}$

- **竞赛例题选讲**

Problem A GCD and LCM（hdu 4497）

三个未知数 x,y,z，它们的 $\gcd$ 为 G，lcm 为 L，G 和 L 已知，求 (x,y,z) 三元组的个数。

Solution

三个数的 $\gcd$ 可以参照两个数 $\gcd$ 的指数最值表示法，只不过每个素因子的指数上是三个数的最值（即$\min\{x_1,y_1,z_1\}$），那么这个问题首先要做的就是将 G 和 L 分别进行素因子分解，然后轮询 L 的每个素因子，对于每个素因子单独处理。

假设素因子为 p，L 分解式中 p 的指数为 l，G 分解式中 p 的指数为 g，那么显然 $l<g$ 时不可能存在满足条件的三元组，所以只需要讨论 $l\geq g$ 的情况，对于单个 p 因子，问题转化成了求三个数 x_1,y_1,z_1，满足 $\min\{x_1,y_1,z_1\}=g$ 且 $\max\{x_1,y_1,z_1\}=l$，更加通俗的意思就是三个数中最小的数是 g，最大的数是 l，另一个数在 $[g,l]$ 范围内，这是一个排列组合问题，三元组 x_1,y_1,z_1 的种类数当 $l==g$ 时只有 1 种，否则答案就是 $6(l-g)$。

最后根据乘法原理将每个素因子对应的种类数相乘就是最后的答案了。

Code

```cpp
int m, n, ans;
int t, num[1010], cnt;
int main()
{
    scanf("%d", &t);
    while(t -- ) {
        scanf("%d%d", &n, &m);
        if(m % n ≠ 0) {
            printf("0\n");
            continue;
        }
        m = m / n;
        cnt = 0;
        for(int i = 2; i * i ≤ m; i ++ ) {
            if(m % i == 0) {
                num[cnt] = 0;
                while(m % i == 0) {
                    m = m / i;
```

```
                    num[cnt] ++ ;
                }
                cnt ++ ;
            }
        }
        if(m != 1) num[cnt ++ ] = 1;
        ans = 1;
        for(int i = 0; i < cnt; i ++ ) ans = ans * num[i] * 6;
        printf("%d\n", ans);
    }
    return 0;
}
```

0x13.4 GCD 与 LCM 的一些性质与定理

性质13.4.1： $\gcd(F(n),F(m))=F(\gcd(n,m))$

性质13.4.2： $\gcd(a^m-1,a^n-1)=a^{\gcd(n,m)}-1\ (a>1,n>0,m>0)$（证明待更...）

性质13.4.3： $\gcd(a^m-b^m,a^n-b^n)=a^{\gcd(m,n)}-b^{\gcd(m,n)}\ (gcd(a,b)=1)$

性质13.4.4： $\gcd(a,b)=1,\gcd(a^m,b^n)=1$

性质13.4.5： $(a+b)\mid ab\Longrightarrow \gcd(a,b)\neq 1$

a,b不互质，因为互质就提不出来公因子了。例题

性质13.4.6： 设$G=\gcd(C_n^1,C_n^2,\dots C_n^{n-1})$

- n 为素数，$G=n$
- n 非素且有一个素因子 p，$G=p$
- n 有多个素因子，$G=1$

性质13.4.7： $(n+1)\mathrm{lcm}(C_n^0,C_n^1,\cdots,C_n^n)=\mathrm{lcm}(1,2,\cdots,n+1)$

性质13.4.8： $\sum_{i=1}^{n}\gcd(i,n)=\sum_{d|n}d\varphi(\frac{n}{d})$

性质13.4.8： 在 `Fibonacc` 数列中求相邻两项的 gcd 时，**辗转相减**次数等于**辗转相除**次数。

性质13.4.8： $\gcd\left(fib_n,fib_m\right)=fib_{\gcd(n,m)}$（证明）

详见推论12.2.1

0x13.5 补充知识： *Fibonacc* 数列及其推论

- **基本性质定理**：

$$fib_n=\begin{cases}0 & n=0\\ 1 & n=1\\ fib_{n-1}+fib_{n-2} & n>1\end{cases}$$

- **推导结论**：

性质13.5.1： $\sum_{i=1}^{n}f_i=f_{n+2}-1$

性质13.5.2： $\sum_{i=1}^{n}f_{2i-1}=f_{2n}$

性质13.5.3： $\sum_{i=1}^{n}f_{2i}=f_{2n+1}-1$

性质13.5.4： $\sum_{i=1}^{n}(f_n)^2=f_nf_{n+1}$

性质13.5.5： $f_{n+m}=f_{n-1}f_{m-1}+f_nf_m$

性质13.5.6： $(f_n)^2=(-1)^{(n-1)}+f_{n-1}f_{n+1}$

性质13.5.7： $f_{2n-1}=(f_n)^2-(f_{n-2})^2$

性质13.5.8： $f_n=\frac{f_{n+2}+f_{n-2}}{3}$

性质13.5.9： $\frac{f_i}{f_{i-1}} \approx \frac{\sqrt{5}-1}{2} \approx 0.618$

性质13.5.10： $f_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$ （证明）

0x14 互质与欧拉函数

0x14.1 欧拉函数

定义

$\forall a, b \in N$ 若 $\gcd(a,b) = 1$,则称 a, b **互质**。

对于三个数或更多的数，把$\gcd(a,b,c) = 1$称之为 a, b, c 互质。
把 $\gcd(a,b) = \gcd(a,c) = \gcd(b,c) = 1$ 称之为 a, b, c 两两互质。显然 a, b, c 两两互质是优于 a, b, c 互质的。

性质14.1.1： int 范围内的数 n 中，$1 \sim n$ 中与 n 互质的个数最多只有1600（$\max\{\varphi(1 \sim 2147483647)\} = 1600$）。

欧拉函数

$1 \cdots N$ 中与 N 互质的数的**个数**，被称为欧拉函数，记作 $\varphi(N)$， `phi`。

如果 n 是一个素数，那么 $\varphi(n) = n - 1$ （所有小于 n 的都互素）

如果 n 是素数的 k 次幂，即 $n = p^k$，那么 $\varphi(p^k) = p^k - p^{k-1}$ （除了 p 的倍数以外，与 $1 \sim n$ 中的任意数都互素）

故我们可以的到下列结论：

由算数基本定理（唯一分解定理）得

$N = p_1^{k_1} \times p_2^{k_2} \times p_3^{k_3} \times \cdots p_m^{k_m}$

则有：

$$\varphi(N) = N \times \prod_{p|N}(1 - \frac{1}{p}) \tag{3}$$

证明见：**0x15 Problem A Co-prime**

由于欧拉函数是积性函数，由 **性质14.2.3** 得：

$$\begin{aligned}
\varphi(N) &= \varphi(p_1^{k_1} \times p_2^{k_2} \times p_3^{k_3} \times \cdots p_m^{k_m}) \\
&= \varphi(p_1^{k_1}) \times \varphi(p_2^{k_2}) \times \varphi(p_3^{k_3}) \times \cdots \times \varphi(\ p_m^{k_m}) \\
&= (p_1^{k_1} - p_1^{k_1-1}) \times (p_2^{k_2} - p_2^{k_2-1}) \times (p_3^{k_3} - p_3^{k_3-1}) \times \cdots (p_m^{k_m} - p_m^{k_m-1}) \\
&= (p_1^{k_1} \times p_2^{k_2} \times p_3^{k_3} \times \cdots p_m^{k_m}) \times (1 - \frac{1}{p_1}) \times (1 - \frac{1}{p_2}) \times \cdots \times (1 - \frac{1}{p_m}) \\
&= N \times (1 - \frac{1}{p_1}) \times (1 - \frac{1}{p_2}) \times \cdots \times (1 - \frac{1}{p_m}) \\
&= N \times \prod_{p|N}(1 - \frac{1}{p})
\end{aligned} \tag{4}$$

其中，如果 p 是素数,则 $\varphi(p) = p \times (1 - \frac{1}{p}) = p - 1$。

我们可以利用这个性质在分解质因数的同时 $O(\sqrt{n})$ 使用公式求解欧拉函数

```cpp
inline int euler_one(int n)
{
    int ans = n;
    for(int i = 2;  i * i ≤ n; ++ i){
        if(n % i == 0){
            ans = ans / i * (i - 1);
            while(n % i == 0)n /= i;
        }
    }
    if(n > 1)ans = ans / n * (n - 1);
    return ans;
}
```

显然我们也可以筛出质数之后用小于 $\sqrt{n}$ 的质数再去分解质因数求欧拉函数时间复杂度为 $\frac{\sqrt{n}}{\log n}$。

我们也可以利用容斥原理得到欧拉函数的计算公式，这里不再展开，详见 **0x15 容斥原理初探**

0x14.2 欧拉函数的性质

- **积性函数**

如果当 a,b 互质时，满足$f(ab)=f(a)\times f(b)$ 的函数 f 称为积性函数。积性函数实际上是由欧拉函数推广到一般的函数上得到的概念，我们将在本文**0x30积性函数**中详细探讨这类问题。

- **欧拉函数性质**

性质14.2.0：当 $n>2$ 时， $\varphi(n)$ 是偶数。

> 证明
> 由于更相减损术，$\gcd(n,m)=\gcd(n,n-m).$
>
> 若 n,m 互质，则有：$\gcd(n,m)=1$ ，$\gcd(n,n-m)=1(n>m)$
>
> 所以每一个与 n 互质的数 m 都对应一个 $n-m$ 与之互质，所以 $\varphi(n)$ 是偶数。

性质14.2.1：$\forall n>1,1\cdots n$ 中与 n 互质的数的和为$n\times\frac{\varphi(n)}{2}$

> 证明
> 因为$\gcd(n,x)=\gcd(n,n-x)$，所以与 n 不互质的数 $x,n-x$ 一定成对出现，平均值为 $\frac{n}{2}$ ，因此与 n 互质的数的平均值也是 $\frac{n}{2}$ ，进而得到性质14.2.1。

性质14.2.2：若 a,b 互质，则 $\varphi(ab)=\varphi(a)\times\varphi(b)$。

> 证明
> 根据欧拉函数的计算式，对 a，b 分解质因数，直接可得性质14.2.2。

性质14.2.3：若 f 是积性函数，且在算数基本定理中 $n=\prod_{i=1}^{m}p_i^{c_i}$，则$f(n)=\prod_{i=1}^{m}f(p_i^{c_i})$。

性质14.2.4：设 p 为质数，若 $p\mid n$ 且 $p^2\mid n$, 则 $\varphi(n)=\varphi(\left\lfloor\frac{n}{p}\right\rfloor)\times p$。（$p\mid n$，即 p 是 n 的因数）

性质14.2.5：设 p 为质数，若 $p\mid n$ 且$p^2\nmid n$,则 $\varphi(n)=\varphi(\frac{n}{p})\times(p-1)$。

性质14.2.6：$\sum_{d\mid n}\varphi(d)=n$。

> 证明
> 如果 $\gcd(k,n)=d$，那么 $\gcd(\frac{k}{d},\frac{n}{d})=1$，（$k<n$）。
>
> 如果我们设 $f(x)$ 表示 $\gcd(k,n)=x$ 的数的个数，那么 $n=\sum_{i=1}^{n}f(i)$。（显然包含 $1\sim n$ 中所有的数）

> 根据上面的证明，我们发现，$f(x)=\varphi(\frac{n}{x})$，从而 $n=\sum_{d|n}\varphi(\frac{n}{d})$。注意到约数 d 和 $\frac{n}{d}$ 具有对称性，所以上式化为 $n=\sum_{d|n}\varphi(d)$。

推论14.2.7： $\sum_{i=1}^{n}i[\gcd(i,n)=1]=\frac{n\varphi(n)+[n=1]}{2}$ （例题）

推论14.2.8： $f(n)=\sum_{i=1}^{n}[\gcd(i,k)=1]=\frac{n}{k}\varphi(k)+f(n \mod k)$

推论14.2.9： 若 i,j 不互质，则 $\varphi(i\times j)=\frac{\varphi(i)\varphi(j)\gcd(i,j)}{\varphi(\gcd(i,j))}$

> **推论14.2.9证明：**
>
> 由欧拉函数计算式：$\varphi(n)=n\times\prod_{p|n}(1-\frac{1}{p})$
>
> $$\begin{aligned}\varphi(i\times j)&=i\times j\times\prod_{p|ij}(1-\frac{1}{p})\\&=\frac{i\times\prod_{p|i}(1-\frac{1}{p})\times j\times\prod_{p|j}(1-\frac{1}{p})}{\prod_{p|i且p|j}(1-\frac{1}{p})}\\&=\frac{\gcd(i,j)\times i\times\prod_{p|i}(1-\frac{1}{p})\times j\times\prod_{p|j}(1-\frac{1}{p})}{\gcd(i,j)\times\prod_{p|i且p|j}(1-\frac{1}{p})}\\&=\frac{\gcd(i,j)\times i\times\prod_{p|i}(1-\frac{1}{p})\times j\times\prod_{p|j}(1-\frac{1}{p})}{\gcd(i,j)\times\prod_{p|\gcd(i,j)}(1-\frac{1}{p})}\\&=\frac{\varphi(i)\varphi(j)\gcd(i,j)}{\varphi(\gcd(i,j))}\end{aligned}$$

推论14.2.10： 设 $n=i\times d^k$，若 $k>2$，则 $\varphi(n)=d\times\varphi(\frac{n}{d})$。

> 若 i,j 互质，$\varphi(ijd^{k+2})=d^k\varphi(ijd^2)=d^k\times\frac{\varphi(id)\times\varphi(jd)\times d}{\varphi(d)}$

推论14.2.10证明：

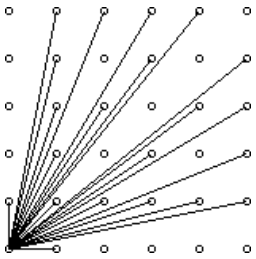
> 设 $n=i\times d^k$，若 $k>2$，显然 n 与 $\frac{n}{d}$ 具有相同的质因子，设$n=\prod_{i=1}^{m}p_i^{c_i}$
>
> 故 $\frac{\varphi(n)}{\varphi(\frac{n}{d})}=\frac{n\times\prod_{i=1}^{m}(1-\frac{1}{p}_i)}{\frac{n}{d}\times\prod_{i=1}^{m}(1-\frac{1}{p}_i)}=\frac{n}{\frac{n}{d}}=d$
>
> 即：$\varphi(n)=d\times\varphi(\frac{n}{d})$

欧拉函数的筛法详见本文 **0x51.1 线性筛求欧拉函数。**

- **竞赛例题宣讲**

Problem A. 仪仗队（[Luogu P2158 [SDOI2008]](https://www.luogu.com.cn/problem/P2158)）

作为体育委员，C君负责这次运动会仪仗队的训练。仪仗队是由学生组成的 $N \times N$ 的方阵，为了保证队伍在行进中整齐划一，C君会跟在仪仗队的左后方，根据其视线所及的学生人数来判断队伍是否整齐(如下图)。 现在，C君希望你告诉他队伍整齐时能看到的学生人数。



Solution

首先我们将原图从 $(1,1)$ 到 (n,n) 重新标号 $(0,0)$ 到 $(n-1，n-1)$

分析题目可知，当 $n=1$ 时，显然答案是 0，特判断即可。

我们考虑 $n \neq 1$的情况，首先$(0,1),(1,0),(1,1)$这三个点是一定能看到的

考虑剩余的点

对于任意的点 (x,y)，$2 \leq x,y \leq n-1$，若不被其它的点挡住必定满足 $\gcd(x,y)=1$。

证明：

> 若点(x,y) 不满足$gcd(x,y)=1$，令$d=gcd(x,y)$
> 则(x,y)与$(0,0)$连线的斜率$k=\frac{y}{x}=\frac{y/d}{x/d}$
> 所以(x,y)会被$(x/d,y/d)$挡住

然后我们把这个图按照对角线分成两个等腰直角三角形

若点(x,y)在一个三角形中，并满足$gcd(x,y)=1$

则必有一个点(y,x)在另一个三角形中,并满足$gcd(y,x)=1$

所以只统计其中一个三角形即可

现在我们在$y<x$的三角形中考虑

对于任意一个x,满足条件的y的个数就是$\phi(x)$

所以答案就是

$$3+2\times\sum_{i=2}^{n-1}\phi(i)$$

我们使用线性筛 $O(n)$ 筛出欧拉函数然后计算答案即可。

Time

$O(n)$

Code

```cpp
#include<cstdio>
#include<cmath>
#include<algorithm>
#include<iostream>
#include<cstring>

using namespace std;
typedef long long ll;
const int N = 50007, M = 50007, INF = 0x3f3f3f3f;
const double eps = 1e-6;

int n, m;

int vis[N];
int primes[N], phi[N], cnt;
```

```cpp
int sum;

void get_euler(int n)
{
    phi[1] = 1;
    for(int i = 2; i ≤ n; ++ i) {
        if(!vis[i]) {
            primes[ ++ cnt] = i;
            phi[i] = i - 1;
        }
        for(int j = 1; j ≤ cnt && i * primes[j] ≤ n; ++ j) {
            vis[i * primes[j]] = true;
            if(i % primes[j] == 0) {//最小的质因子
                phi[i * primes[j]] = phi[i] * primes[j];
                break;
            }
            else phi[i * primes[j]] = phi[i] * (primes[j] - 1);
        }
    }
}

int main()
{
    scanf("%d", &n);
    if(n == 1) puts("0") , exit(0);
    n --;
    get_euler(n);
    sum = 0;
    for(int i = 2; i ≤ n; ++ i)
        sum += phi[i];
    int ans = 3 + 2 * sum;
    printf("%d\n", ans);
    return 0;
}
```

0x15 容斥原理初探

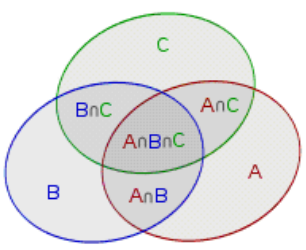
> **容斥原理虽然是组合数学的内容，但是与数论配合较为密切，因此在这里简单介绍一下容斥原理以及它与数论的一些简单结合。**

容斥原理是一种应用在集合上的较常用的计数方法，其基本思想是：先不考虑重叠的情况，把包含于某内容中的所有对象的数目先计算出来（**容**），然后再把计数时重复计算的数目排斥出去（**斥**），使得计算的结果既无遗漏又无重复。

容斥原理核心的计数规则可以归为一句话：**奇加偶减**。

假设被计数的有 A、B、C 三类，那么，A、B、C 类元素个数总和 $=$ A 类元素个数 $+$ B 类元素个数 $+$ C 类元素个数 $-$ 既是 A 又是 B 的元素个数 $-$ 既是 A 又是 C 的元素个数 $-$ 既是 B 又是 C 的元素个数 $+$ 既是 A 又是 B 且是 C 的元素个数。

即：$A \cup B \cup C = A + B + C - AB - BC - AC + ABC$



当被计数的种类被推到 n 类时，其统计规则遵循**奇加偶减**。

- **竞赛例题选讲**

Problem A Co-prime（HDU 4135）

求区间 $[a, b]$ 中与 n 互质的数的个数，其中 $1 \leq a, b, n \leq 2^{31}$。

Solution

容斥定理最常用于**求解 $[a,b]$ 区间与 n 互质的数的个数问题**，该问题可以视为求 $[1,b]$ 区间与 n 互质的个数减去 $[1,a-1]$ 区间内与 n 互质的个数。

那么我们这里只需要考虑它的一个子问题：求小于等于 m 且 n 互素的数的个数。

我们知道当 m 等于 n ，就是一个简单的欧拉函数问题，但是一般 m 都不等于 n ，我们考虑将 n 质因数分解。

我们首先分析最简单的情况：当 n 为素数的幂，即 $n=p^k$ 时，那么显然答案就等于 $m-\left\lfloor\frac{m}{p}\right\rfloor$ （其中 $\left\lfloor\frac{m}{p}\right\rfloor$ 表示的是 p 的倍数，$1\sim m$ 中去掉 p 的倍数，剩下的就都是与 n 互素的数了）

然后再来讨论 n 是两个素数的幂的乘积，即 $n=p_1^{k_1}\times p_2^{k_2}$，那么我们需要做的就是找到 p_1 的倍数和 p_2 的倍数，并且要减去 p_1 和 p_2 的公倍数，我们发现这实际上就是容斥原理，所以这种情况下答案为：$m-(\frac{m}{p_1}+\frac{m}{p_2}-\frac{m}{p_1\times p_2})$。

（拓展到 n 的所有质因子即可得到欧拉函数的计算式）

这里的 $+$ 就是 **容**， $-$ 就是 **斥**，并且 **容** 和 **斥** 总是交替进行的（一个的加上，两个的减去，三个的加上，四个的减去），而且可以推广到 n 个元素的情况,如果 n 分解成 s 个素因子，也同样可以用容斥原理求解。

容斥原理其实是枚举子集的过程，常见的枚举方法为 dfs ，也可以采用二进制法（0 表示取，1 表示不取）。

例如我们求 $[1,9]$ 中和 6 互素的数的个数，这时 6 分解的素因子为 2 和 3 。

$ans=9-(\frac{9}{2}+\frac{9}{3})+\frac{9}{6}=3$，其中，$ans$ 分为三部分， 0 个数的组合， 1 个数的组合，2 个数的组合。

答案 = （$1\sim b$ 的元素个数） $-$ （$1\sim a-1$ 的元素个数） $-$ （$1\sim b$ 中与 n 不互质的数的个数） $+$ （$1\sim a-1$ 中与 n 不互质的数的个数）

Code

```cpp
const int N = 10005;
ll n, primes[N], cnt, factor[N], num;
bool vis[N];
inline void get_primes(int n)
{
    for(register int i = 2;i ≤ n;i ++) {
        if(!vis[i]) primes[ ++ cnt] = i;
        for(register int j = 1;j ≤ cnt && i * primes[j] ≤ n; ++ j) {
            vis[i * primes[j]] = 1;
            if(i % primes[j] == 0) break;
        }
    }
}
void get_factor(int n) {
    num = 0;
    for (ll i = 1; primes[i] * primes[i] ≤ n && i ≤ cnt; i ++ ) {
        if (n % primes[i] == 0) { //记录n的因子
            factor[num ++ ] = primes[i];
            while (n % primes[i] == 0)
                n /= primes[i];
        }
    }
    if (n ≠ 1) //1既不是素数也不是合数
        factor[num ++ ] = n;
}
ll solve(ll m, ll num) {
    ll res = 0;
    for (ll i = 1; i < (1 << num); i++) {
        ll sum = 0;
        ll temp = 1;
        for (ll j = 0; j < num; j++) {
            if (i & (1 << j)) {
                sum ++ ;
```

```
                temp *= factor[j];
            }
        }
        if (sum % 2) res += m / temp;
        else res -= m / temp;
    }
    return res;
}
int kcase, t;
int main() {
    get_primes(N - 1);
    scanf("%d", &t);
    while(t -- ) {
        ll a, b, n;
        scanf("%lld%lld%lld", &a, &b, &n);
        get_factor(n);
        //容斥定理，奇加偶减，
        ll res = (b - (a - 1) - solve(b, num)) + solve(a - 1, num);
        printf("Case #%d: %lld\n", ++ kcase, res);
    }
    return 0;
}
```

Problem B Helping Cicada（LightOJ - 1117）

给定 m 个数。 求 $[1,n]$ 中不能被这 m 个数整除的数的个数。

Solution

我们知道对于任意一个数 k，在 $[1,n]$ 中有 $\left\lfloor \frac{n}{k} \right\rfloor$ 个数是 k 的倍数，也就是能被 k 整除，故 $ans = n - \sum_{i=1}^{n} \left\lfloor \frac{n}{a[i]} \right\rfloor$ 。

我们再来考虑两个数 a,b 的情况，因为 a,b 的公倍数 $\operatorname{lcm}(a,b)$，即被 a 整除，又被 b 整除，所以减了两次。所以最后的答案要加上 $\left\lfloor \frac{n}{\operatorname{lcm}(a,b)} \right\rfloor$ 。对于三个数 a,b,c 来说，答案就要减去 $\left\lfloor \frac{n}{\operatorname{lcm}(a,b,c)} \right\rfloor$ ，四个数就要加上，五个就要减去，以此类推。

最后拓展到 m 个数的情况，我们发现需要使用容斥定理。

根据容斥定理的**奇加偶减**，对于 m 个数来说，其中的任意 $2,4,\cdots,2k$ 个数就要**减去**他们最小公倍数能组成的数，$1,3,\cdots,2k+1$ 个数就要**加上**他们的最小公倍数，对于每一个数来说，我们都有选或不选两种情况，因此 m 个数就有 2^m 种情况，也就是从0（啥也不选）到 2^m-1，对应二进制就是 m 个 1，也就意味着 我们选择了 m 个数。这种方法叫做状态压缩，我们接用状态压缩来枚举所有的可能状态，依次使用位运算判断当前的状态选择了多少个数，然后再根据容斥原理进行奇加偶减即可。

$sum =$ （从 m 中选 1 个数得到的倍数的个数） $-$ （从 m 中选 2 个数得到的倍数的个数） $+$ （从 m 中选 3 个数得到的倍数的个数） $-$ （从 m 中选 4 个数得到的倍数的个数）.....

那么能被整除的数的个数就是 sum，不能被整除的数的个数就是 $n - sum$ 。

Code

```
const int N = 10005;
ll LCM(ll a, ll b) {
    return a / __gcd(a, b) *b;
}
int m, kcase;
ll n, a[N];
int main()
{
    int t;
    scanf("%d", &t);
    while(t -- ) {
        scanf("%lld%d", &n, &m);
        for(int i = 0; i < m; ++ i)
            scanf("%lld", &a[i]);
        ll sum = 0;
```

```
        for(int i = 0; i < (1 << m); ++ i) {
            ll lcm = 1;
            ll cnt = 0;
            for(int j = 0; j < m; ++ j) {
                if(i >> j & 1) {//如当前的状态i选择了第j个数
                    lcm = LCM(lcm, a[j]);//那就选第j个数
                    cnt ++ ;
                }
            }
            if(cnt =≠ 0) {
                if(cnt & 1) sum += n / lcm;//奇加
                else sum -= n / lcm;//偶减
            }
        }
        printf("Case %d: %lld\n", ++ kcase, n - sum);
    }
    return 0;
}
```

Problem C 硬币购物 ([P1450 [HAOI2008]](https://www.luogu.com.cn/problem/P1450))

共有 4 种硬币。面值分别为 c_1, c_2, c_3, c_4 。

某人去商店买东西，去了 n 次，对于每次购买，他带了 d_i 枚 i 种硬币，想购买 s 的价值的东西。请问每次有多少种付款方法。

Solution

看起来像是一个多重背包求方案数的模板题，但是由于有 n 组，所以每次都求一次多重背包绝对会超时。

显然多重背包即有限制的方案数，直接计算有限制的方案数比较难，考虑**正难则反**：

有限制的方案数 = 无限的方案数 - 超过限制方案数

首先无限制的方案数显然就是完全背包，我们可以先用完全背包预处理出所有的方案数，即 `f[i]` 表示购买价值为 `i` 的方案数。

考虑减去超过显示的即不合法的部分，考虑使用容斥原理。

首先考虑一种硬币超额使用的方案数怎么计算。若第 j 种硬币超额使用，即超过了原定的 d_j 个硬币的限制，我们可以先强制选了 d_j+1个第j 种硬币，这样剩下的价值里， 4 种硬币随便选，这样就能保证第 j 种硬币一定超额使用，第 j 种硬币超额的不合法的方案数就等于 $f[s-(d_j+1)\times c_j]$。

然后我们只需要使用容斥原理奇加偶减即可。加上一种硬币不合法的方案数，减去两种硬币不合法的方案数，加上三种硬币不合法的方案数 $\cdots$ 二进制枚举子集即可。

Code

```
// Problem: P1450 [HAOI2008]硬币购物
// Contest: Luogu
// URL: https://www.luogu.com.cn/problem/P1450
// Memory Limit: 125 MB
// Time Limit: 1000 ms
//
// Powered by CP Editor (https://cpeditor.org)

#include <bits/stdc++.h>
#define int long long
using namespace std;
const int N = 1e5 + 7;
int n, m, s, t, k, ans;
int f[N];
int c[50], d[50];

void pre_work(int n)
{
    f[0] = 1;
```

```
    for (int i = 1; i ≤ 4; ++ i)
        for (int j = c[i]; j < n; ++ j)
                f[j] += f[j - c[i]];
}

void solve()
{
    scanf("%lld%lld%lld%lld%lld", &d[1], &d[2], &d[3], &d[4], &s);

    int ans = f[s], now;
    for (int i = 1; i ≤ (1 ≪ 4) - 1 ; ++ i) {
        now = s;

        int tmp, j, k;
        for (tmp = i, j = 1, k = 0; tmp; tmp >>= 1, ++ j) {
            if(tmp & 1) {
                k ^= 1;
                now -= (d[j] + 1) * c[j];
            }
        }
        if(now ≥ 0)
            k ? ans -= f[now] : ans += f[now];
    }
    printf("%lld\n", ans);
}

signed main()
{
    scanf("%lld%lld%lld%lld%lld", &c[1], &c[2], &c[3], &c[4], &n);
    pre_work(N - 7);
    while (n -- ){
        solve();
    }
    return 0;
}
```

Problem D. Coprime Subsequences（CF803F）

给定一个 n 个数的序列，问你有多少个子序列的 $\gcd = 1$ 。

$n \leq 10^5$

Solution

序列一共有 n 个数，显然一共有 $2^n - 1$ 个子序列（每个数选或不选减去空集）

考虑容斥。显然答案就是 $2^n - 1$ 减去 $\gcd > 1$ 的子序列个数，设所有含有大于 1 的因子的序列中的个数为 x ，显然 $\gcd > 1$ 的子序列的个数为 $2^x - 1$。显然只与点的权值有关，而 $a[i] \leq 10^5$，考虑维护权值。设序列中的数的最大值为 m。

- 设 cnt_i 表示权值为 i 的序列中的数的个数，可以在输入的时候处理一下。
- 设 sum_i 表示含有因子 i 的数的个数，显然 $sum_i = \sum_{i|j} cnt_j$，即序列中 i 的倍数的个数。我们可以通过枚举倍数在 $O(m \log m)$ 的复杂度下计算。
- 设 f_i 表示含有因子 i 的子序列的个数，显然 $f_i = 2^{sum_i} - 1 = 2^{\sum_{i|j} cnt_j} - 1$，显然 $sum < m \leq 10^5$，我们可以 $O(m)$ 预处理一下 2 的次幂。

对于 $\gcd > 1$ 的子序列个数，根据奇加偶减的容斥原理，显然为：含有因子 2 的子序列的个数（f_2） $+$ 含有因子 3 的子序列的个数（f_3） $+$ 含有因子 5 的子序列的个数（f_5） $+ \cdots -$ 含有因子 $2, 3$ 的子序列的个数（f_6） $-$ 含有因子 $2, 5$ 的子序列的个数（f_{10}） $- \cdots +$ 含有因子 $2, 3, 5$（f_{30}） 的子序列的个数 $+ \cdots$

最终的答案为 $2^n - 1$ 减去 $\gcd > 1$ 的子序列个数，即变为奇减偶加的形式。

但是如果我们继续使用二进制枚举选取状态，复杂度为 $O(2^n), n \leq 10^5$，显然不可做。

因此我们引入莫比乌斯函数：

$$\mu(n) = \begin{cases} 0 & \exists i \in [1,m], C_i > 1 \\ (-1)^m & \forall i \in [1,m], C_i = 1 \end{cases}$$

然后我们可以发现前面 f_x 的系数实际上就是 $\mu(x)$（莫比乌斯函数本身就是一个容斥的映射）。

即答案为

$$2^n - 1 + \sum_{i=2}^{m} \mu(i) \times f_i \tag{5}$$

Time

$O(m \log m), m = \max\{a[i]\}$

Code

```cpp
// Problem: CF803F Coprime Subsequences
// Contest: Luogu
// URL: https://www.luogu.com.cn/problem/CF803F
// Memory Limit: 250 MB
// Time Limit: 2000 ms
// Powered by CP Editor (https://cpeditor.org)

#include <bits/stdc++.h>
using namespace std;
const int N = 500007, mod = 1e9 + 7;

typedef long long ll;
int n, m, t;
int a[N], mu[N], cnt[N];
bool vis[N];
int primes[N], tot;
int pow2[N];
ll ans;

int add(int a, int b)
{
    return 1ll * a + b ≥ mod ? 1ll * a + b - mod : 1ll * a + b;
}

int sub(int a, int b)
{
    return a - b < 0 ? a - b + mod : a - b;
}

void init(int n)
{
    pow2[0] = 1, pow2[1] = 2, mu[1] = 1;
    for(int i = 2; i ≤ n; ++ i) {
        pow2[i] = add(pow2[i - 1], pow2[i - 1]);
        if(vis[i] == 0) {
            primes[ ++ tot] = i;
            mu[i] = -1;
        }
        for(int j = 1; j ≤ tot && i * primes[j] ≤ n; ++ j) {
            vis[i * primes[j]] = true;
            if(i % primes[j] == 0) {
                mu[i * primes[j]] = 0;
                break;
            }
            mu[i * primes[j]] -= mu[i];
```

```
        }
    }
}

int main()
{
    scanf("%d", &n);
    for(int i = 1; i ≤ n; ++ i) {
        scanf("%d", &a[i]);
        m = max(m, a[i]);
        cnt[a[i]] ++ ;
    }

    init(N - 7);
    ans = pow2[n] - 1;
    for(int i = 2; i ≤ m; ++ i) {
        int sum = 0;
        for(int j = i; j ≤ m; j += i)
            sum = (1ll * sum + cnt[j]) % mod;
        ans = (ans + 1ll * mu[i] * (pow2[sum] - 1) % mod + mod) % mod;
    }
    printf("%lld\n", ans);
    return 0;
}
```

0x16 RSA原理

> 本小节的内容涉及到后面章节的内容，且竞赛不会考察（~~谁知道呢~~），建议学本书之后再做了解

如果要问我哪一种算法是最重要的，那么答案一定会包含公钥加密算法，因为它是计算机通信安全的基石，保证了加密数据不会被破解，想想你的**加密资源**被别人破解，那将是一个多么可怕的事情😋。

RSA加密算法是非对称加密算法中的一种，在1977年由罗纳德·李维斯特（Ron Rivest）、阿迪·萨莫尔（Adi Shamir）和伦纳德·阿德曼（Leonard Adleman）一起提出的，并取三人名字的首字母命名该算法。

RSA加密算法因其可靠的安全性（目前看来是十分安全的），得到了广泛的认可和使用，ISO（国际标准化组织）、ITU（国际电信联盟）及SWIFT（环球同业银行金融电讯协会）等国际标准组织均采用RSA作为加密标准，PGP等协议也采用RSA算法来传输会话密钥和数字签名等等。

RSA加密算法用到的数学知识有：素数，互质，欧拉函数，费马小定理，斐蜀定理，欧拉定理。（有没有发现全是数论的知识嘻嘻嘻）

RSA原理

RSA 算法有三个参数，n，pub, pri，其中 n 等于两个大素数 p 和 q 的乘积 $(n = p \times q)$，pub 可以任意取，但是要求与 $(p-1)*(q-1)$ 互素，$\text{pub} \times \text{pri}\%() = 1$ (可以理解为 pri 是 pub 的逆元)，那么这里的 (n, pub) 称为公钥，(n, pri) 称为私钥。$(p-1)*(q-1)$

RSA算法的加密和解密是一致的，令 x 为明文，y 为密文，则：

- 加密：$y = x^{\text{pub}}\ \%n$ (利用公钥加密，$y = \text{encode}(x)$)
- 解密：$x = y^{\text{pri}}\ \%n$ (利用私钥解密，$x = \text{decode}(y)$)

那么我们来看看这个算法是如何运作的。

假设你得到了一个密文 y，并且手上只有公钥，如何得到明文 x，从 decode 的情况来看，只要知道私钥貌似就可以了，而私钥的获取方式只有一个，就是求公钥对 $(p-1)*(q-1)$ 的逆元，如果 $(p-1)*(q-1)$ 已知，那么可以利用扩展欧几里德定理进行求解，问题是 $(p-1)*(q-1)$ 是未知的，但是我们有 $n = p*q$，于是问题归根结底其实是难在了对n进行素因子分解上了，$\texttt{Pollard}-\texttt{rho}$ 的分解算法时间复杂度只能达到$O(n^{\frac{1}{4}})$，对 `int_64` 范围内的整数可以在几十毫秒内出解，而当 n 是几百位、几千位的大数的时候，计算时间就是一个天文数字了，以此达到加密的作用。

以上です。

全文目录索引链接： https://fanfansann.blog.csdn.net/article/details/113765056