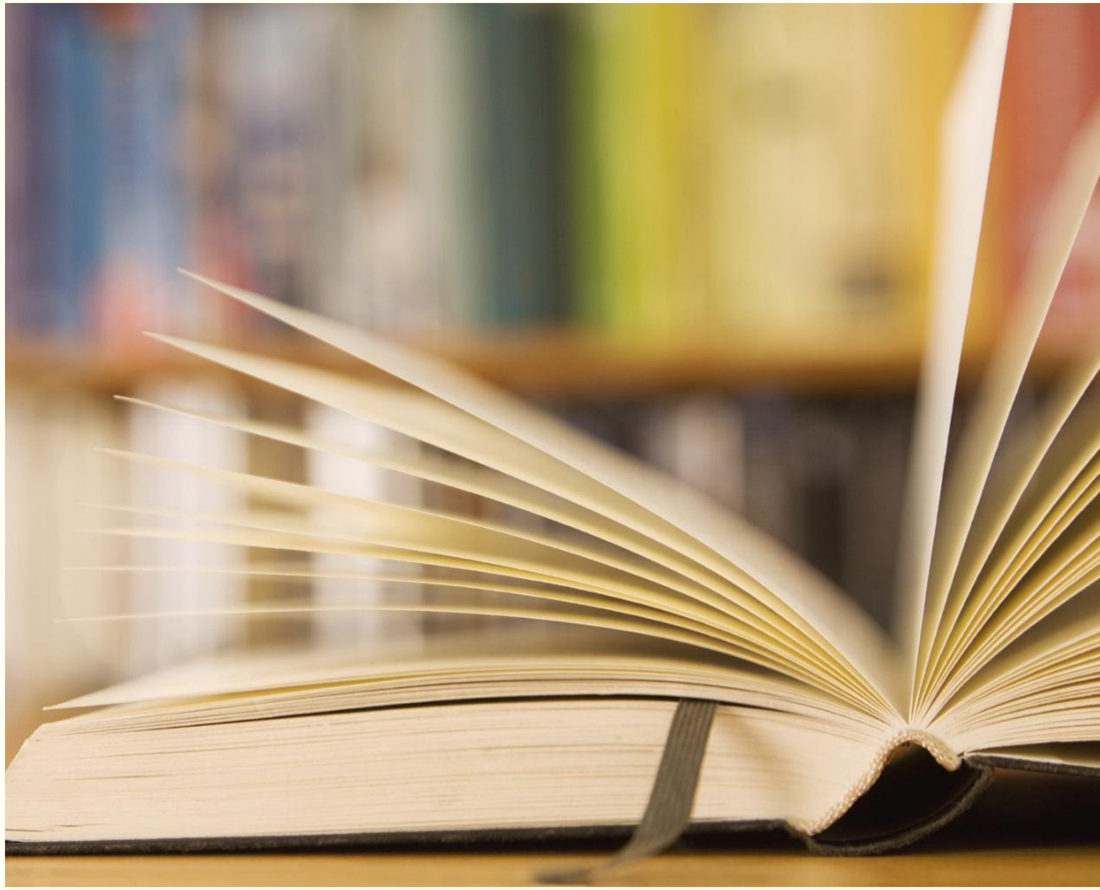
数论1：整除相关



目录

数论

- 数论，是专门研究整数的纯数学的分支，而整数的基本元素是素数（也称质数），所以数论的本质是对素数性质的研究。
- 数论被高斯誉为“数学中的皇冠”。按研究方法来看，数论大致可分为初等数论和高等数论。而初等数论是用初等方法研究的数论，它的研究方法本质上说，就是利用整除性质，主要包括整除理论、同余理论、连分数理论。
- 信息学竞赛中的数论主要涉及素数、约数、同余和数论函数等相关知识。

素数

基本概念

- **整数集合**：$Z = \{..., -2, -1, 0, 1, 2, ...\}$
- **自然数集合**：$N = \{0, 1, 2, ...\}$
- **整除**：若 $a = bk$，其中 a, b, k 都是整数，则 b 整除 a，记做 $b|a$，否则记做 $b \nmid a$ 。
- **约数**：如 $b|a$ 且 $b \geq 0$，则称 b 是 a 的约数（因数），a 是 b 的倍数。
  - 1 整除任何数，任何数都整除 0 。
  - 若 $a|b, a|c$，则 $a|(b+c), a|(b-c)$ 。
  - 若 $a|b$，则对任意整数 c，$a|(bc)$ 。
  - 传递性：若 $a|b, b|c$，则 $a|c$ 。
- **因子**：正整数 a 的平凡约数为 1 和 a 本身，a 的非平凡约数称为 a 的因子。如 20 的因子有 2、4、5、10 。

素数与合数

- **素数：** $a > 1$ 且只能被平凡约数整除的数。
- **合数：** $a > 1$ 且不是素数的数称为合数。
- 其他整数（0,1,负整数）既不是素数也不是合数。
- 素数有无穷多个，但分布比较稀疏，不大于 n 的素数约有 $\frac{n}{\ln(n)}$ 个。

基本概念

- x、y、z均为整数，且11|7x+2y-5z，求证：11|3x-7y+12z
- 证明:
- 分别构造11|2(7x+2y-5z)和11|11(x+y-2z)
- 将两式相减得到11|2(7x+2z-5z)-11(x+y-2z)
- 所以11|3x-7y+12z

素数判定方法1

- 若 n 是一个合数，则 n 至少有 1 个素因子。因此其中最小的素因子一定不大于 $\sqrt{n}$ 。
- 如果 n 是合数，则一定可以为分解 $a \times b$ 的形式，其中 $a \leq b, a \neq 1, b \neq n$，如 $18 = 2 \times 9, 18 = 3 \times 6$。因 $a \times a \leq a \times b = n$ ，则可得：$a \leq \sqrt{n}$ 。
- 可得判断依据：如果 $2 \sim \sqrt{n}$ 中有 n 的约数，则 n 是合数，否则 n 是素数。

```
bool isPrime(int n){
    if (n==1) return false;
    else{
        for(int i=2;i*i<=n;i++)
            if (n % i == 0) return false;
        return true;
    }
}
```

素数判定方法2

- 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2/3。
- 下面是kn+i法 k=30版本的模板：

素数判定方法2

```
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;
}
```

素数判定方法3

- 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($\frac{n}{\ln n}$)。

素数判定方法4

- 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：$a^{p-1} \equiv 1\ (mod\ p)$。
- 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。

素数判定方法4

- 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。

素数判定方法4

```
bool Miller_Rabbin(ll a,ll n){
    ll s=n-1,r=0;
    while((s&1)==0){
        s>>=1;r++;
    }
    ll k=pow_mod(a,s,n);
    if(k==1)return true;
    for(int i=0;i<r;i++,k=k*k%n){
        if(k==n-1)return true;
    }
    return false;
}
bool isprime(ll n){
    ll times=7;
    ll prime[100]={2,3,5,7,11,233,331};
    for(int i=0;i<times;i++){
        if(n==prime[i])return true;
        if(Miller_Rabbin(prime[i],n)==false)return false;
    }
    return true;
```

素因数分解

例题：[NOIP 2012] 素因数分解

- 已知正整数 n 是两个不同素因数的乘积，试求出较大的那个素数。比如 21，较大的素因数是 7。对于 60% 的数据，$6 \le n \le 1000$，对于 100% 的数据，$6 \le n \le 2 \times 10^9$。
- 分析：
- 既然已经明确此合数为两个素数的乘积，即可以得出 $O(\sqrt{n})$ 的算法

```
    for(int i=2;i*i <= n;i++)
        if(n%i == 0) {
            cout << n/i << endl;
            break;
        }
```

整数惟一分解定理

- 表述：若整数 $N \geq 2$，那么 N 一定可以惟一地表示为若干素数的乘积。形如

$$N = p_1^{r_1} p_2^{r_2} \dots p_k^{r_k} \ (p_i\text{为素数}, r_i \geq 0)$$

```cpp
vector<int> factor(int x) {
    vector<int> ret;
    for (int i = 2; i * i <= x; ++i)
        while (x % i == 0) {
            ret.push_back(i);
            x /= i;
        }
    if (x > 1) ret.push_back(x);
    return ret;
}
```

素数筛法

素数筛法

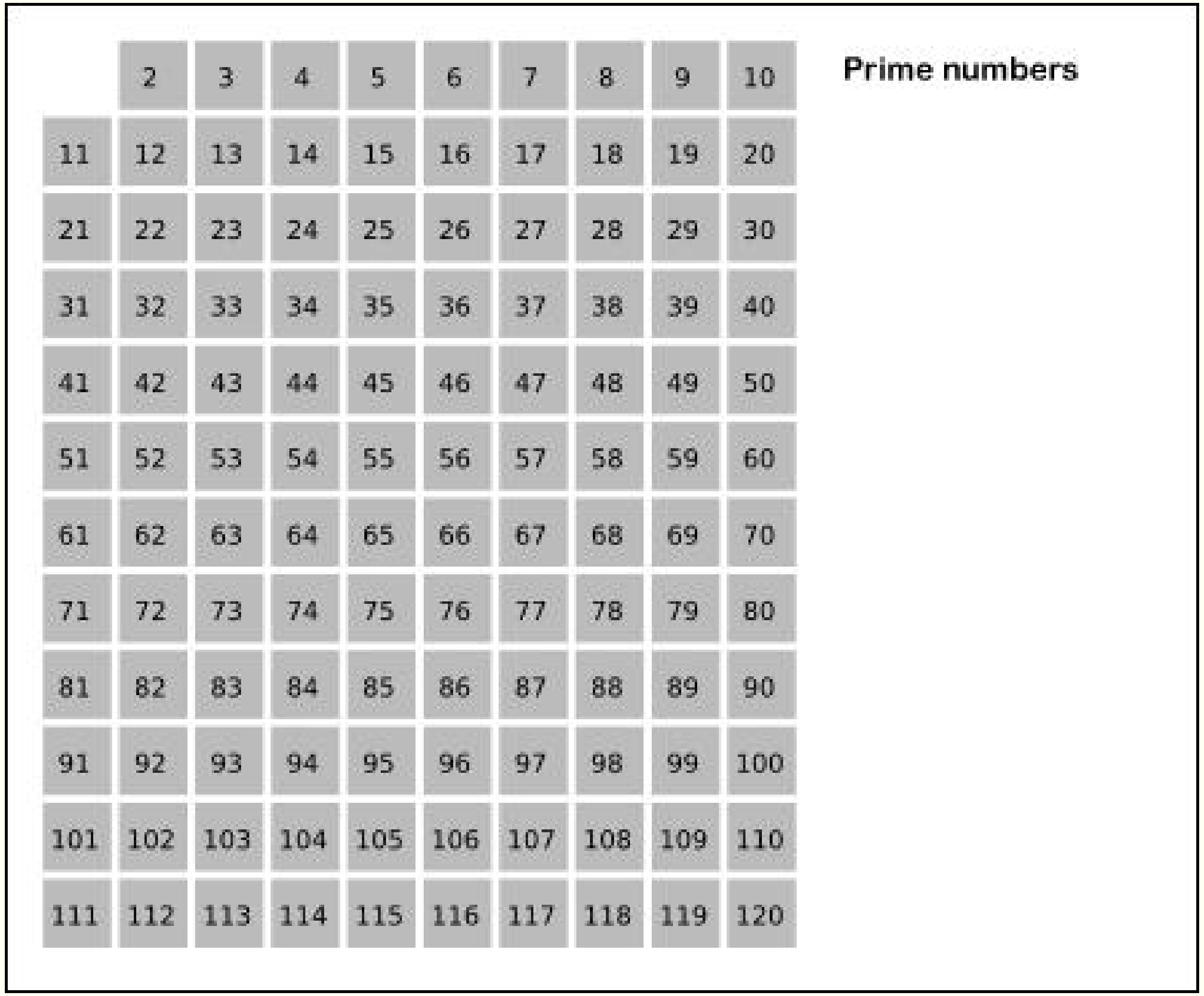
- 质数筛法一般分为**埃氏筛**和**线性筛**。
- 埃氏筛没有线性筛时间复杂度好，不常用，但是它的**时间复杂度分析方法**却比较常用。
- 首先我们来证明一下 $O(\sum_{i=1}^{n}\frac{1}{i}) \approx O(\log n)$

素数筛法

- $\sum_{i=1}^{n}\frac{1}{i}=1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\frac{1}{7}+\ldots\ldots$
- $=1+(\frac{1}{2}+\frac{1}{3})+(\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\frac{1}{7})+\ldots\ldots$
- $\leq 1+\frac{1}{2}*2+\frac{1}{4}*4+\ldots\ldots$
- 所以1~n 就被分成了logn组，每组的和小于等于1，故$\sum_{i=1}^{n}\frac{1}{i}\approx \log n$。

Eratosthenes筛法

- 除了能够检验给定整数 x 是否为素数的函数之外，如果能够事先准备好素数表就可以帮助我们更有效地求解素数的相关问题。埃拉托色尼筛选法（Sieve of Eratosthenes）可以快速列举出给定范围内的所有素数。



埃氏筛法思想

- 每个合数 a 一定可以写成 $p*x$ 的形式，其中 p 是素数，x 是倍数($x \neq 1$)，对于每一个 $1 \sim n$ 内的素数 p，枚举倍数 x ,把 $p*x$ 标记为合数，这就是埃氏筛法。
- 筛选时做一个改进：对于素数 p，只筛倍数 $x \geq p$ 的数，因为如果 $x < p$,则 x 中一定有比 p 小的素因子，$p*x$ 会在前面筛选过程中被筛出。
- 因此只需考虑 $2 \sim \sqrt{n}$ 范围的素数。
- 时间复杂度：$O(\frac{n}{2}+\frac{n}{3}+\frac{\mathrm{n}}{5}+\ldots)=O(n\mathrm{loglog}n)$

埃氏筛法

```
#define maxn 1000000
bool isPrime[maxn+1];/* isPrime[i]true表示i为素数*/
void eratos(int n){
    int i,j;
    isPrime[0] = isPrime[1] = false;
    for(i = 2;i <= n; ++i)
        isPrime[i] = true;
    for(i = 2;i * i <= n; ++i)
        if(isPrime[i]){
            for(j = i * i;j <= n;j += i)
                isPrime[j] = false;
        }
}
```

埃氏筛法思想

- 埃氏筛的时间复杂度为O(nloglogn)，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，$\frac{n}{i} \in \left\{n, \frac{n}{2}, \frac{n}{3} \ldots\ldots \frac{n}{n}\right\}$,而调和级数f(n)=1+$\frac{1}{2}$+$\frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \ldots\ldots + \frac{1}{n}$，当 n 趋近于 ∞ 时，极限为lnn+C， 其中 C 为欧拉常数 ,$\mathrm{C} \approx 0.577218$。
- 当我们使用优化筛掉合数以后，这个调和级数就变成 f(n)=$\frac{1}{2}$+$\frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \ldots\ldots + \frac{1}{n} \approx \mathrm{lo}glogn$, 所以整体的算法时间复杂度为O(nloglogn)。
- 利用调和级数计算时间复杂度的方法很常用。

例题：[POJ 2689]Prime Distance

- 给定两个整数 L, R ($1 \leq L \leq R \leq 2^{31}, R - L \leq 10^6$)，求闭区间 $[L, R]$ 中相邻两个素数的差最大是多少?
- 分析:
- 由于数据范围很大，无法生成 $[1, R]$ 中的所有素数。
- 使用筛法求出 $\left[2, \sqrt{R}\right]$ 之间的所有素数，对于每个素数 p ，把 $[L, R]$ 中能被 p 整除的数标记，即标记 $i \times p \left(\left\lceil\frac{L}{p}\right\rceil \leq i \leq \left\lfloor\frac{R}{p}\right\rfloor\right)$ 为合数。
- 将筛出的素数进行相邻两两比较，找出差最大的即可。

筛法优化素因数分解-1

- 利用埃氏筛法可以快速实现素因数分解，只要在判定质数时记录下每个数值的最小素因数即可。算法实现如下：

```
#define maxn 1000000
bool isPrime[maxn+1];
int minFactor[maxn+1];   //记录每个数的最小素因数的数组

void eratos(int n){
    int i,j;
    isPrime[0] = isPrime[1] = false;
    minFactor[0] = minFactor[1] = -1;
    for(i = 2;i <= n; ++i) {
        isPrime[i] = true;
        minFactor[i] = i; //初始化，表示还未找到最小的素因数
    }
```

筛法优化素因数分解-2

```
    for(i = 2;i * i <= n; ++i) {
          if(isPrime[i]){
                for(j = i * i;j <= n;j += i){
                      isPrime[j] = false;
                      if(minFactor[j]==j) //如果此前尚未找到j的素因数,
                            //那么将其设为i
                            minFactor[j] = i;
                }
          }
    }
}
```

筛法优化素因数分解-3

```
vector<int> factor(int x) {
    vector<int> ret;
    while(x > 1){
        ret.push_back(minFactor[x]);
        x /= minFactor[x];
        }
    return ret;
}
```

例题：阶乘分解

- 给定整数 N（$1 \le N \le 10^6$），试将阶乘 $N!$ 分解素因数，以惟一分解形式给出 p_i 和 r_i。
- 分析：
- 如果对 $1 \sim N$ 中的每个数进行素因数分解，再将结果合并，时间复杂度为$O(N\sqrt{N})$，超时。可知 $N!$ 的每个素因数都不超过 N，可用筛法先求出 $1 \sim N$ 的每个素数 N，然后再考虑 $N!$ 中包含哪些素因数 p。
- $N!$ 中包含的素因数 p 的个数等于 $1 \sim N$ 每个数包含素因数 p 的个数之和。在 $1 \sim N$ 中，p 的倍数有 $\left\lfloor\frac{N}{p}\right\rfloor$ 个，p^2 的倍数有 $\left\lfloor\frac{N}{p^2}\right\rfloor$ 个，综上可得 $N!$ 中 p 的个数为

$$\left\lfloor\frac{N}{p}\right\rfloor + \left\lfloor\frac{N}{p^2}\right\rfloor + \left\lfloor\frac{N}{p^3}\right\rfloor + \dots + \left\lfloor\frac{N}{p^{\lfloor\log_p N\rfloor}}\right\rfloor = \sum_{p^k \le N} \left\lfloor\frac{N}{p^k}\right\rfloor$$

- 每个 p 需要 $O(\log N)$，对于 $N!$ 分解素因数的时间复杂度为$O(N\log N)$。

欧拉筛法（线性筛）

- 埃氏筛法中,以 $n = 50$ 为例，30 这个数被筛了 3 次，分别是:

$$2 * 15(p = 2)$$
$$3 * 10(p = 3)$$
$$5 * 6(p = 5)$$

- 如何$O(n)$求$1 \sim n$的素数?

- 如果每个合数只被它的最小素因数筛除，那么每个数最多只被筛一次。

欧拉筛法（线性筛）$n = 50$的筛选过程图示

$i =$	素数表	筛除的数	$i =$	素数表	筛除的数
$\underline{2}$	$\{2\}$	$\{4\}$	$\underline{13}$	$\{2, 3, 5, 7, 11, 13\}$	$\{26, 39\}$
$\underline{3}$	$\{2, 3\}$	$\{6, 9\}$	$\underline{14}$	$\{2, 3, 5, 7, 11, 13\}$	$\{28\}$
$\underline{4}$	$\{2, 3\}$	$\{8\}$	$\underline{15}$	$\{2, 3, 5, 7, 11, 13\}$	$\{30, 45\}$
$\underline{5}$	$\{2, 3, 5\}$	$\{10, 15, 25\}$	$\underline{16}$	$\{2, 3, 5, 7, 11, 13\}$	$\{32\}$
$\underline{6}$	$\{2, 3, 5\}$	$\{12\}$	$\underline{17}$	$\{2, 3, 5, 7, 11, 13, 17\}$	$\{34\}$
$\underline{7}$	$\{2, 3, 5, 7\}$	$\{14, 21, 35, 49\}$	$\underline{18}$	$\{2, 3, 5, 7, 11, 13, 17\}$	$\{36\}$
$\underline{8}$	$\{2, 3, 5, 7\}$	$\{16\}$	$\underline{19}$	$\{2, 3, 5, 7, 11, 13, 17, 19\}$	$\{38\}$
$\underline{9}$	$\{2, 3, 5, 7\}$	$\{18, 27\}$	$\underline{20}$	$\{2, 3, 5, 7, 11, 13, 17, 19\}$	$\{20\}$
$\underline{10}$	$\{2, 3, 5, 7\}$	$\{20\}$	$\underline{21}$	$\{2, 3, 5, 7, 11, 13, 17, 19\}$	$\{42\}$
$\underline{11}$	$\{2, 3, 5, 7, 11\}$	$\{22, 33\}$	$\underline{22}$	$\{2, 3, 5, 7, 11, 13, 17, 19\}$	$\{44\}$
$\underline{12}$	$\{2, 3, 5, 7, 11\}$	$\{24\}$	$\underline{\ldots}$	$\ldots$	$\ldots$

欧拉筛法（线性筛）

- 枚举 `2~n` 中的每一个数 `i`:
  - 如果 `i` 是素数则保存到素数表中;
  - 利用 `i` 和素数表中的素数 `prime[j]` 去筛除 `i*prime[j]` ，为了确保 `i*prime[j]` 只被素数 `prime[j]` 筛除过这一次，要确保 `prime[j]` 是 `i*prime[j]` 中最小的素因子，即 `i` 中不能有比 `prime[j]` 还要小的素因子。

欧拉筛法（线性筛）

```
void Euler_sieve(int n){
    memset(isprime,true,sizeof(isprime));
    prime[0]=0; //记录当前素数个数
    for(int i=2;i<=n;i++){
        if (isprime[i])prime[++prime[0]]=i;//把素数保存到素数表prime中，并更新素数个数
        for(int j=1;j<=prime[0] && i*prime[j]<=n;j++){
            isprime[i*prime[j]]=false;//筛除i*prime[j]
            if (i % prime[j]==0) break;
            //当i中含有素因子prime[j]时中断循环,确保每个数只被它的最小素因子筛除
        }
    }
}
```

约数

整数惟一分解定理的推论

- 若整数$N \geq 2$，那么 $N = p_1^{r_1} p_2^{r_2} \ldots p_k^{r_k}$ (p_i为素数，$r_i \geq 0$)
- N 的正约数集合为：$\{p_1^{b_1} p_2^{b_2} \ldots p_k^{b_k}\}$ $(0 \leq b_i \leq r_i)$
- N 的正约数个数为：

$$(r_1 + 1) \times (r_2 + 1) \times \ldots \times (r_k + 1) = \prod_{i=1}^{k} (r_i + 1)$$

- 除了完全平方数，约数总是成对出现的，即 $d \leq \sqrt{N}$ 和 $\frac{N}{d} \leq \sqrt{N}$ 都是 N 的约数。
- N 的约数个数上界为 $2\sqrt{N}$ ，时间复杂度为 $O(\sqrt{N})$ 。
- N 的所有正约数的和为：

$$\left(1 + p_1 + p_1^2 + \ldots + p_1^{r_1}\right) \times \ldots \times \left(1 + p_k + p_k^2 + \ldots + p_k^{r_k}\right) = \prod_{i=1}^{k} \left(\sum_{j=0}^{r_i} (p_i)^j \right)$$

例题： [BZOJ 1053] 反素数

- 对于任何正整数 x ，其约数的个数计作 $g(x)$。例如 $g(1) = 1, g(6) = 4$ 。
- 如果某个正整数 x 满足：对于任意的 $0 < i < x$ ，都有 $g(x) > g(i)$，那么称 x 为反素数。例如 1,2,4,6 都是反素数。
- 现在给定一个数 $N(1 \le N \le 2 * 10^9)$，求出不超过 N 的最大的反素数。

例题： [BZOJ 1053] 反素数

- 性质1：1-N中的反素数，就是1-N中约数个数最多的数中最小的一个。
- 性质2：1-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。
- 性质3：x的质因子是连续的若干个最小的质数，并指数单调递减。
- 这三个性质非常显然，应该不需要证明。
- 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。

Pollard-Rho算法

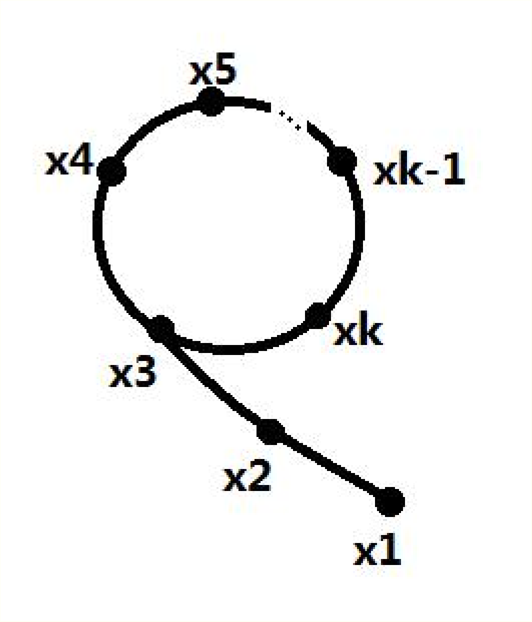
- 对于数据较大的情况，如 n>=10^18，有用来分解其因数的 Pollard-Rho算法。

- 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’=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x1,x2,x3…xk ，求任意两个数xi,xj的差和n的最大公约数，如果 1<d<n，则 d为n 的一个因子，直接返回d 即可。

Pollard-Rho算法

- 然后来看如何选取这 k个数，我们采用生成函数法，令 x1=rand()%(n-1)+c,xi=(x_{i-1}^2+c)%n,很明显，这个序列是有循环节的，如图所示



- 我们在这里使用**Floyd判环算法**（也叫**龟兔赛跑算法**），设置两个变量 t,r，每次判断是否有 gcd(|t−r|,N)>1 ，如果没有，就令 t=f(t) ， r=f(f(r)) 。因为 r 跑得更快，如果没有找到答案，最终会与 t 在环上相遇，这时退出，换一个 c 重新生成伪随机数。

Pollard-Rho算法

```
ll Pollard_Rho(ll N){
    if (is_prime(N)) return N;// 特判质数
    while (1){
        ll c = randint(1, N - 1); // 生成随机的c
        auto f = [=](ll x) { return ((lll)x * x + c) % N; }; // lll表示__int128，防溢出
        ll t = f(0), r = f(f(0));
        while (t != r){
            ll d = gcd(abs(t - r), N);
            if (d > 1)return d;
            t = f(t), r = f(f(r));
        }
    }
}
```

最大公约数

- 设 a, b 是不都为 0 的整数，c 为满足 $c|a$ 且 $c|b$ 的最大整数，则称 c 是 a, b 的**最大公约数**，记为 $\gcd(a, b)$ 或 (a, b) 。
- 最大公约数有如下性质：
  - $\gcd(a, b) = \gcd(b, a)$
  - $\gcd(a, b) = \gcd(-a, b)$
  - $\gcd(a, b) = \gcd(|a|, |b|)$
  - 若$d|a$且$d|b$，则$d|\gcd(a, b)$
  - $\gcd(a, 0) = a$

最大公约数

- 最大公约数有如下性质：
  - $\gcd(a, ka) = a$
  - $\gcd(an, bn) = n\gcd(a, b)$
  - $\gcd(a, b) = \gcd(a, ka + b)$

计算gcd(a, b)：枚举法

- 从 $\min(a, b)$ 到 1 枚举 x，并判断 x 是否能同时整除 a 和 b 。
- 如果可以则输出 x 退出循环。
- 时间复杂度为$O(\min(a, b))$ 。

计算gcd(a, b)：分解素因数

- 可得求 gcd(a, b) 的一般公式：

$$当\ a = p_1^{r1} p_2^{r2} \ldots p_k^{rk}\ ，\ b = p_1^{s1} p_2^{s2} \ldots p_k^{sk}\ ，\ ri, si \geq 0\ 且不会同时为\ 0\ (1 \leq i \leq n)$$

$$则\ \gcd(a, b) = p_1^{\min(r1,\ s1)} p_2^{\min(r2,\ s2)} \ldots p_k^{\min(rk,\ sk)}$$

```
int Decompose(int a,int b){
    int ans = 1;
    for(int x = 2; x * x <= min(a,b); x++){
        while(a % x == 0 && b % x == 0){a /= x;b /= x;ans *= x;}
        while(a % x == 0) a /= x;
        while(b % x == 0) b /= x;
    }
    if(a % b == 0) ans *= b;
    else if(b % a == 0) ans *= a;
    return ans;
}
```

计算gcd(a,b)：欧几里得算法

- “两个整数 $a,b(a \geq b)$ 的公约数集合与 $a-b$ 和 b 的公约数集合相同”，可得：

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

- 又称“辗转相除法”

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

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

计算gcd(a, b)：欧几里得算法-复杂度分析

- 根据 $(a, b) => (b, a \bmod b)$，设$a > b$：

  ①当 $a \geq 2b$时，$b \leq a/2$，b 的规模至少缩小一半；

  ②当 $a < 2b$时，$a \bmod b < a/2$，余数的规模至少缩小一半；

  所以时间复杂度为 $\mathrm{O}(\log(a + b))$ 。

例题：[BZOJ 1876] Super GCD

- 求$\gcd(A, B)$，$0 < A, B \leq 10^{10000}$ 。

计算gcd(a,b)：适用于高精度数的二进制法

- 适合大整数（高精度）情况下求gcd

1. $a < b$时，$\gcd(a,b) = \gcd(b,a)$
2. $a = b$时，$\gcd(a,b) = a$
3. a,b 同为偶数时，$\gcd(a,b) = 2 * \gcd(a/2, b/2)$
4. a 为偶数，b 为奇数时，$\gcd(a,b) = \gcd(a/2, b)$
5. a 为奇数，b 为偶数时，$\gcd(a,b) = \gcd(a, b/2)$
6. a,b 同为奇数时，$\gcd(a,b) = \gcd(a-b, b)$

- 又称“更相减损术”。

计算gcd(a,b)：适用于高精度数的二进制法

```
int gcd(int m,int n){
    if (m == n) return m;
    if (m < n) return gcd(n,m);
    if (m & 1 == 0)
        return (n & 1 == 0)? 2*gcd(m>>1,n>>1):gcd(m>>1,n);
    return (n & 1 == 0)? gcd(m,n>>1): gcd(n,m-n);
}
```

- 以上代码仅作为算法说明，大整数如为数组存储方式，则上述运算符都需要实现重载。

例题：[NOIP 2009] Hankson的趣味题

- 有 n 个询问，在每个询问中，先给定四个自然数 a, b, c, d ，然后求有多少个 x 满足 $\gcd(a, x) = c$ 并且 $\operatorname{lcm}(b, x) = d$。$n \leq 2000, 1 \leq a, b, c, d \leq 2 * 10^9$ 。
- 分析:
- 从 $\operatorname{lcm}(b, x) = d$ 可知 x 一定是 d 的约数。可得朴素算法：用试除法求出 d 的所有约数，逐一判断是否满足这两个条件。时间复杂度为 $O\left(n\sqrt{d}\log d\right)$ 。
- 虽然 d 的约数个数上界大约是 $\sqrt{d}$ ，但是 $1 \sim d$ 中平均每个数的约数个数大约只有 $\log d$。
- 10^9 之内的自然数中，约数个数最多的自然数仅有 1536 个约数。
- 为了尽量避免试除法中不能整除时耗费的时间，预处理出 $1 \sim \sqrt{2 * 10^9}$之间的所有素数，用搜索算法组成 d 的所有约数，再判断题目的两个条件是否满足即可。

最小公倍数

- a 和 b 最小的正公倍数为 a 和 b 的**最小公倍数**，记作 $\mathrm{lcm}(a,b)$ 。
- 最小公倍数有如下性质：
  - $\mathrm{lcm}(a,b)=\frac{ab}{\gcd(a,b)}$
  - 若 $a|m$ 且 $b|m$，则 $\mathrm{lcm}(a,b)|m$
  - 若 m,a,b 是正整数，则 $\mathrm{lcm}(ma,mb)=m*\mathrm{lcm}(a,b)$ 。

计算n个整数$a_1, a_2, \ldots, a_n$的最小公倍数

- 两个数 a_1, a_2 的最小公倍数

$$\mathrm{lcm}(a_1, a_2) = a_1 a_2 / \mathrm{gcd}(a_1, a_2)$$

$$\mathrm{lcm}(a_1, a_2, a_3) = \mathrm{lcm}(\mathrm{lcm}(a_1, a_2), a_3)$$

- 以此类推，可以先求 a_1, a_2 的最小公倍数 b_1，再求 b_1 与 a_3 的最小公倍数 b_2，再求 b_2 与 a_4 的最小公倍数 $b_3 \ldots$

```
ans=1;
for(int i=1;i<=n;i++){
    scanf("%d",&a[i]);
    ans=ans*a[i]/gcd(ans,a[i]);
}
printf("%d",ans);
```

容斥原理

各集合的交集

- 现在有 $S = \{1,2,3,\ldots,600\}$，求其中可被 2,3,5 整除的数的数目。
- 令 A,B,C 分别表示 S 中被 2,3,5 整除的数的集合。可得：

$$|A| = \left\lfloor\frac{600}{2}\right\rfloor = 300, |B| = \left\lfloor\frac{600}{3}\right\rfloor = 200, |C| = \left\lfloor\frac{600}{5}\right\rfloor = 120$$

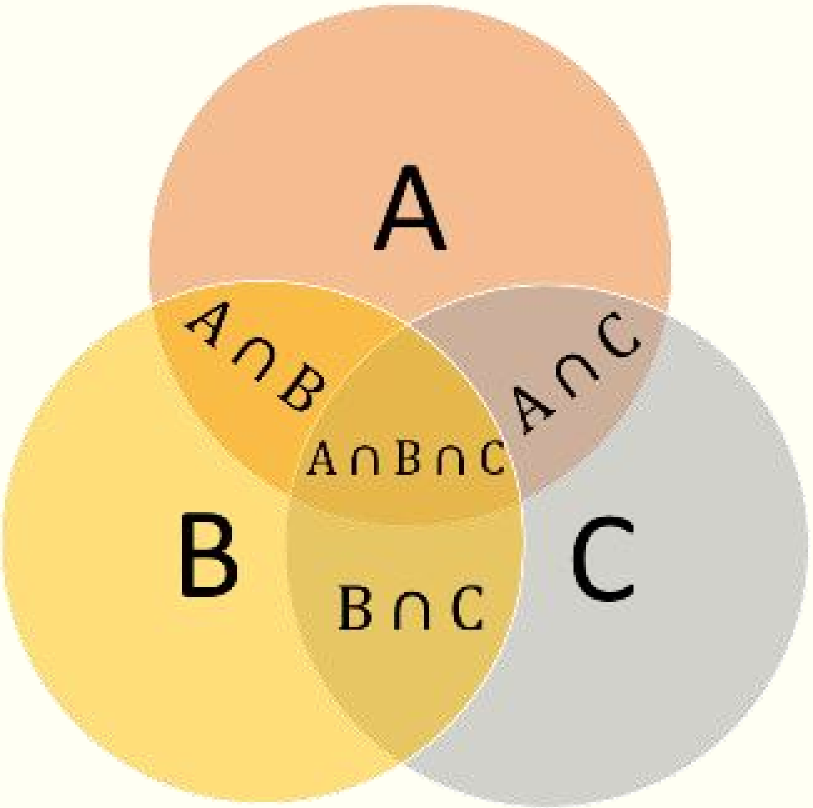
- 显然 A,B 集合中一定有相同的元素，比如 6,12…，可继续求 A,B,C 两两交集的情况：

$$|A \cap B| = \left\lfloor\frac{600}{2 \times 3}\right\rfloor = 100, |A \cap C| = \left\lfloor\frac{600}{2 \times 5}\right\rfloor = 60, |B \cap C| = \left\lfloor\frac{600}{3 \times 5}\right\rfloor = 40,$$

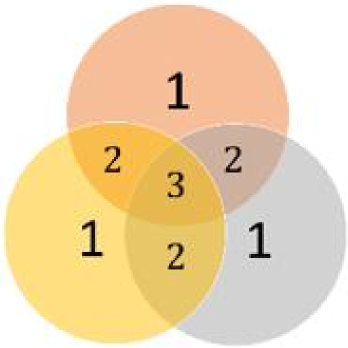
- 最后求 A,B,C 三个集合的交集情况：

$$|A \cap B \cap C| = \left\lfloor\frac{600}{2 \times 3 \times 5}\right\rfloor = 20$$

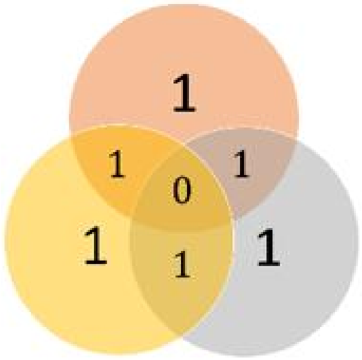
容斥原理



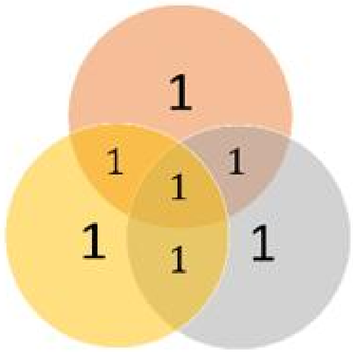
容斥原理



$|A| + |B| + |C|$



$|A| + |B| + |C|$
$-|A \cap B| - |A \cap C| - |B \cap C|$



$|A| + |B| + |C|$
$-|A \cap B| - |A \cap C| -$
$|B \cap C| + |A \cap B \cap C|$

容斥原理

- 具有性质 A 或者 B 的元素个数，等于具有性质 A 的元素个数与具有性质 B 的元素个数的和，减去同时具有性质 A 和 B 的元素的个数，使得计算的结果既无遗漏又无重复。这就是容斥原理，一般表示如下：

$$|\cup A_i|$$

$$= \sum_{1\le i\le m} |A_i| - \sum_{1\le i<j\le m} \left|A_i \cap A_j\right| + \sum_{1\le i<j<k\le m} \left|A_i \cap A_j \cap A_k\right| - \dots$$

$$+ (-1)^{m+1} \sum |A_1 \cap A_2 \cap \dots \cap A_m|$$

错排问题

- n 个有序的元素应有 $n!$ 种不同的排列。若一个排列使得所有的元素都不在自己原来的位置上，则称这个排列为错排。现任给一个 n，求出 $1,2,\ldots,n$ 的错排个数。
- 设集合 S 为 $1,2,\ldots,n$ 的所有全排列，可得 $|S|=n!$ 。S 的子集 S_i 为数字 i 排在 i 位置上的全排列，因为数字 i 不动，所以

$$|S_i|=(n-1)!\,(i=1,2,\ldots,n)$$

- 同理 $S_{i1}\cap S_{i2}\cap\ldots\cap S_{ik}$ 表示 $i_1,i_2,\ldots,i_k$ 位置上的全排列的集合，这就是说在 $1,2,\ldots,n$ 中除了 $i_1,i_2,\ldots,i_k$ 这 k 个数被固定外，其余 $n-k$ 个数可以任意排列。所以

$$|S_{i1}\cap S_{i2}\cap\ldots\cap S_{ik}|=(n-k)!\quad(1\le i_1\le i_2,\ldots,i_k\le n,k=1,2,\ldots,n)$$

错排问题

- 每个元素都不在自己位置上的排列数为D_n，

$$D_n = |\overline{S_1} \cap \overline{S_2} \cap \ldots \cap \overline{S_n}|$$

- 由容斥原理公式可得

$$D_n$$
$$= |S| - \sum_{1 \le i \le n} |S_i| + \sum_{1 \le i < j \le m} |S_i \cap S_j| - \sum_{1 \le i < j < k \le n} |S_i \cap S_j \cap S_k| + \ldots$$
$$+ (-1)^{n-1} \sum |S_1 \cap S_2 \cap \ldots \cap S_n|$$
$$= n!\left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \ldots \pm \frac{1}{n!}\right)$$

欧拉函数

欧拉函数

- 若 $(a,b)=1$，则称 a,b 互素（互质），记作 $a\perp b$ 。
- 欧拉函数 $\varphi(n)$（读作fai），定义为 $[1,\ n]$ 中与 n 互素的数的个数。
- $\varphi(8)=4$，小于 8 且与 8 互素的数是 1,3,5,7。
- 推论：若 p 为素数，则 $\varphi(p)=p-1$ 。

容斥原理求欧拉函数

- 若将 N 分解为不同素数的乘积，即：

$$N = p_1^{r_1} p_2^{r_2} \dots p_k^{r_k}$$

- 设 1 到 N 的 N 个数中为 p_i 倍数的集合为 A_i， $|A_i| = \left\lfloor \frac{N}{p_i} \right\rfloor$ （$i = 1,2, \dots, k$）。
- 对于$p_i \neq p_j$，$A_i \cap A_j$既是p_i的倍数也是p_j的倍数，即可得

$$\left|A_i \cap A_j\right| = \left\lfloor \frac{N}{p_i p_j} \right\rfloor \ (1 \leq i,j \leq k, i \neq j)$$

- 在去除 $|A_i|$ 和 $\left|A_j\right|$ 的时候，p_i和p_j的倍数被去除去了两次，需要再把 $\left|A_i \cap A_j\right|$ 加回来。

容斥原理求欧拉函数

- $\varphi(N) = |\overline{A_1} \cap \overline{A_2} \cap \ldots \cap \overline{A_k}|$

$$= N - \left(\frac{N}{p_1} + \frac{N}{p_2} + \ldots + \frac{N}{p_k}\right) + \left(\frac{N}{p_1 p_2} + \frac{N}{p_2 p_3} + \ldots + \frac{N}{p_1 p_k}\right) \ldots \pm \left(\frac{N}{p_1 p_2 \ldots p_k}\right)$$

$$= N\left(1 - \frac{1}{p_1}\right)\left(1 - \frac{1}{p_2}\right) \ldots \left(1 - \frac{1}{p_k}\right)$$

基于素因数分解求欧拉函数的算法

```
int euler_phi(int n){
    int res = n;
    for(int i = 2; i * i <= n; i++) {
        if(n % i == 0) {
            res = res / i * (i - 1);
            for (; n % i == 0; n /= i);
        }
    }
    if (n != 1) res = res / n * (n - 1);
    return res;
}
```

- 时间复杂度为$O(\sqrt{n})$

利用埃氏筛法，实现欧拉函数值的预处理

- 利用埃氏筛法，每次发现素因子时就把它的倍数的欧拉函数乘上 $(p-1)/p$，这样就可以一次性求出 $1\sim n$ 的欧拉函数值的表了。实现如下：

```
int euler[MAX_N];
void euler_phi2(){
    for (int i = 0; i < MAX_N; i++) euler[i] = i;
    for (int i = 2; i < MAX_N; i++) {
        if (euler[i] == i) {
            for (int j = i; j < MAX_N; j += i)
                euler[j] = euler[j] / i * (i - 1);
        }
    }
}
```

欧拉函数的性质

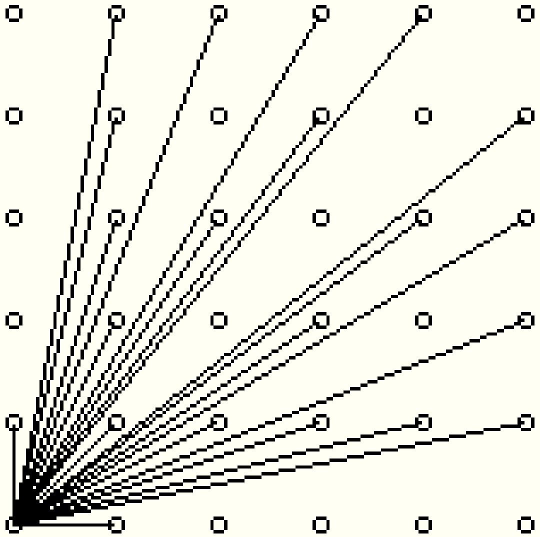
- 若 $a \perp b$ ，则:

$$\varphi(ab) = \varphi(a)\varphi(b)$$

- 将 a, b 各自分解素因数，利用欧拉函数计算式即可得证。
- 欧拉函数是积性函数

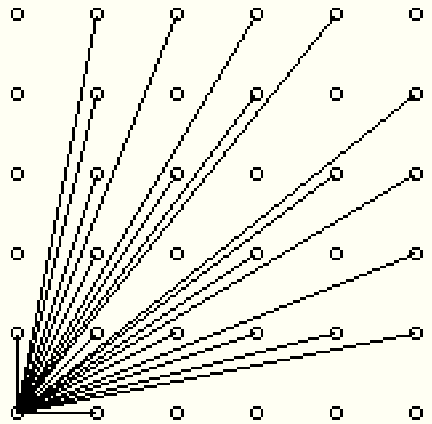
例题：[POJ 3090] Visible Lattice Points

- 在一个平面直角坐标系的以 $(0,0)$ 为左下角、 (N,N) 为右上角的矩形中，除了$(0,0)$ 之外，每个坐标上都插着一个钉子，如图所示。
- 求在原点向四周看去，能够看到多少个钉子。一个钉子能被看到，当且仅当连接它和原点的线段上没有其他钉子。下图也画出了所有能看到的钉子以及实现。
- $1 \leq N \leq 1000$



例题：[POJ 3090] Visible Lattice Points

- 分析题目容易发现，除了$(1,0)$, $(0,1)$ 和$(1,1)$ 这三个钉子外，一个钉子(x,y) 能被看到，当且仅当 $1 \le x,y \le N,\ x \ne y$ 并且 $\gcd(x,y)=1$ 。
- 在 $1 \le x,y \le N,\ x \ne y$ 中能看到的钉子关于过 $(0,0)$ 和(N,N) 的直线对称。考虑其中一半，即 $1 \le x < y \le N$ 。
- 换言之对于每个 $2 \le y \le N$ ，需要统计有多少个 x 满足 $1 \le x < y$ 并且 $\gcd(x,y)=1$ 。这样的 x 的数量恰好就是 $\varphi(y)$ 。
- 综上所述，本题的答案就是 $3+2*\ \sum_{i=2}^{N}\varphi(i)$ 。



例题：[BZOJ 2705] Longge的问题

- 给定一个整数 $N(N \leq 2^{32})$，求 $\sum_{i=1}^{N} \gcd(i,N)$
- 分析：
- 设 $\gcd(i,N) = d$，可得 $\gcd\left(\frac{i}{d},\frac{N}{d}\right) = 1$，即 $\gcd(i,N) = d \times \gcd\left(\frac{i}{d},\frac{N}{d}\right)$
- 因为 $\frac{i}{d} \leq \frac{N}{d}$，所以共有 $\varphi\left(\frac{N}{d}\right)$ 个 i 满足条件，因此：

$$\sum_{i=1}^{N} \gcd(i,N) = \sum_{d|N} d\varphi\left(\frac{N}{d}\right)$$

- 利用素因数分解求出每个$\varphi\left(\frac{N}{d}\right)$ 即可。
- 时间复杂度为 $\sum_{d|N} \sqrt{d}$