1. 同余

若 a, b 为两个整数，且它们的差 $a-b$ 能被某个自然数 m 所整除，则称 a 就模 m 来说同余于 b，或者说 a 和 b 关于模 m 同余，记为：$a \equiv b(mod\ m)$。它意味着：$a-b=m \times k$（k 为某一个整数）。

例如$32 \equiv 2(mod\ 5)$，此时 k 为 6。

对于整数 a, b, c 和自然数 m, n，对模 m 同余具有以下一些性质：

1. 自反性：$a \equiv a(mod\ m)$
2. 对称性：若 $a \equiv b(mod\ m)$，则 $b \equiv a(mod\ m)$
3. 传递性：若 $a \equiv b(mod\ m), b \equiv c(mod\ m)$，则 $a \equiv c(mod\ m)$
4. 同加性：若 $a \equiv b(mod\ m)$，则 $a+c \equiv b+c(mod\ m)$
5. 同乘性：若 $a \equiv b(mod\ m)$，则 $a*c \equiv b*c(mod\ m)$

若 $a \equiv b(mod\ m), c \equiv d(mod\ m)$，则 $a * c \equiv b * d(mod\ m)$

6. **不满足同除性**，但是若 $gcd(c, m) = 1$，则当 $a \times c \equiv b \times c\ (mod\ m)$时，有 $a \equiv b\ (mod\ m)$。
7. 同幂性：若 $a \equiv b(mod\ m)$，则 $a^n \equiv b^n(mod\ m)$
8. 推论1：$a * b(mod\ k) = (a\ mod\ k) * (b\ mod\ k)mod\ k$
9. 推论2：若 p, q 互质，$a\ mod\ p = x, a\ mod\ q = x$，则 $a\ mod\ (p * q) = x$

> 证明：
>
> 因为 p, q 互质，$a\ mod\ p = x, a\ mod\ q = x$
>
> 所以一定存在整数 s, t，使得 $a = s * p + x, b = t * q + x$
>
> 所以 $s * p = t * q$
>
> 又因为 t 为整数，p, q 互质，将 q 移到左边来看
>
> 则 $q|s$，即存在整数 r，使得 $s = r * q$
>
> 所以 $a = r * q * p + x$，即 $a\ mod\ (p * q) = x$。

1.1 例题

这个就没什么例题了，记住就行了

2 素数

2.1 素数的定义

一个大于 1 的自然数，除了 1 和它自身外，不能被其他自然数整除的数叫做**素数或质数。**

注意：

- **1** 既不是素数也不是合数；
- **2** 是最小的素数，也是**唯一**的偶素数；
- 素数的个数是**无限**的（可以用反证法证明）：

- 证明：

  假设素数是有限的，假设素数只有有限的 n 个，最大的一个素数是 p。设 q 为所有素数之积加上 1，即 $q = (2 \times 3 \times 5 \times \ldots \times p) + 1$ 不是素数。那么，q 可以被 2、3、$\ldots$、p 中的若干个数整除（因为合数一定可以分解成若干质因子的乘积）。而 q 被这 2、3、$\ldots$、p 中任意一个整除都会余1，与之矛盾。所以，素数是无限的。

2.2 有关素数的一个定理

2.2.1 算术基本定理（唯一分解定理）

任何一个大于 1 的正整数都能被唯一分解为**有限个**素数的乘积，可写作：

$N = p_1^{c_1} p_2^{c_2} p_3^{c_3} \cdots p_m^{c_m}$

其中 c_i 都是**正整数**，p_i 都是**素数**且满足 $p_1 < p_2 < p_3 \cdots < p_m$。

2.3 单个素数的判定

单个素数判定复杂度是 $O(\sqrt{n})$。

```
bool isPrime(int x) {
    if (x < 2) return false;
    for (int i = int(sqrt(x+0.5)); i >= 2; --i) {
        if (x % i == 0) return false;
    }
    return true;
}
```

2.4 筛素数

对于求出某个范围内的所有素数，我们可以从 2 开始，将 2 的所有倍数都去掉，剩下的第一个未被去掉的数 3 为第二个素数。再讲 3 的所有的倍数去掉，以此来推……我们可以筛选出 n 以内的所有的素数。

但是这种方法会造成**重复筛除合数**，影响效率。比如 $n = 30$，在 $i = 2$ 的时候，$k = 2 \times 15$ 筛了一次；在 $i = 3$ 的时候，$k = 3 \times 10$，筛了一次；在 $i = 5$ 的时候，$k = 5 \times 6$ 又筛了一次。因此也就有了"快速线性筛法"。

2.4.1 线性筛

每个合数只被它**最小的质因子**筛掉。时间复杂度为 $O(n)$

```
int prime[MAXN]; // 保存素数
bool is_not_prime[MAXN] = {1, 1}; // 0和1都不是素数

// 筛选 n 以内的所有素数
void xxs(int n) {
    for (int i = 2; i <= n; ++i) {
        if (!is_not_prime[i]) { // 如果i是素数
            prime[++prime[0]] = i;
        }
        for (int j = 1; j <= prime[0] && i * prime[j] <= n; ++j) {
            is_not_prime[i*prime[j]] = 1;
            // 如果i中包含了该质因子，则停止
            if (i % prime[j] == 0) break;
        }
    }
}
```

上面代码中的 `if (i % prime[j] == 0) break;` 用来控制每个合数只被筛掉一次。

直观的举个例子：$i = 2 \times 3 \times 5$，此时能筛除 $2 \times i$，不能筛除 $3 \times i$。如果此时筛除 $3 \times i$ 的话，当 $i' = 3 \times 3 \times 5$ 时，筛除 $2 \times i'$ 就和前面重复筛了。也就是说，如果 $i \mod prime[j] == 0$，说明 i 自身有一个最小的质因子 $prime[j]$，根据线性筛的原理，到此终止即可。

2.5 欧拉函数

2.5.1 定义

- 对于正整数 n，其欧拉函数是指小于等于 n 的数中与 n 互质的数的个数，用字母 φ 表示。

- 通项公式为：$\varphi(n)=n\times\prod_{i=1}^{k}(1-\frac{1}{p_i})$，其中 $p_1,p_2,\ldots,p_k$ 为 n 的所有质因数，n 为正整数。
- 例如：$\varphi(12)=4$，即 12 以内与 12 互质的数的个数为 4，有 $1,5,7,11$。
- 利用通项公式计算：$\varphi(12)=12\times(1-\frac{1}{2})\times(1-\frac{1}{3})=4$ (因为 12 的不同的质因数只有 $2,3$)。

2.5.2 性质

1. $\varphi(1)=1$。
2. 若 p 是一个素数，则 $\varphi(p)=p-1$。
3. 若 p 是一个素数，则 $\varphi(p^k)=(p-1)\times p^{k-1}$。
4. 欧拉函数为积性函数：对于任意两个正整数 a,b，且 $gcd(a,b)=1$，则 $\varphi(a\times b)=\varphi(a)\times\varphi(b)$。特别的，对于奇数 n，$\varphi(2n)=\varphi(n)$。

证明：

1. 显然成立。
2. 也很显然。
3. 对于 $n=p^k$，比 n 小的正整数有 p^k-1个。其中，所有能被 p 整除的那些数可以表示成 $p\times t$ 的形式（$t=1,2,3,\ldots,p^{k-1}-1$），共有 $p^{k-1}-1$ 个数能被 p 整除，也就是说这些数不与 p^k 互质。所以 $\varphi(p^k)=p^k-1-(p^{k-1}-1)=p^k-p^{k-1}=(p-1)\times p^{k-1}$。
4. 证明略，需要用到中国剩余定理......

还有一些变形：

1. p 为素数，若 $n\%p=0$，则 $\varphi(n\times p)=p\times\varphi(n)$。（这条一些资料都要求 p 为素数，但是 p 不是素数的时候也成立，大家可以自己推一下）。
2. p 为素数，若 $n\%p\neq0$，则 $\varphi(n\times p)=(p-1)\times\varphi(n)$。
3. 当 n 为奇数时，$\varphi(2n)=\varphi(n)$。
4. 与 n 互质的数都是成对出现的，且每对的和为 n，所以大于 2 的数的 $\varphi(n)$ 都为偶数。

证明：（反证法）

假设 $gcd(n,x)=1,x<n,n>2$，但是 $gcd(n,n-x)=k(k>1)$，则可以改写成 $n=a\times k,n-x=b\times k$，那么移项可得 $x=n-b\times k=a\times k-b\times k=(a-b)\times k$，则 $gcd(n,x)=gcd(ak,(a-b)k)$，它们至少有一个公约数 k，与假设矛盾。

2.5.3 求欧拉函数

1. 如果只要求一个数的欧拉函数值，那么直接根据定义，在质因数分解的同时求就好了

```
int euler_phi(int n) {
  // 如果存在大于根号n的质因子，至多有一个
  int m = int(sqrt(n + 0.5));
  int ans = n;
  // 跟判定素数很像
  for (int i = 2; i <= m; i++) {
    // 如果i能整除n，则i是n的因子，也会质因子（看下面的while）
    if (n % i == 0) {
      ans = ans / i * (i - 1); // 根据定义计算
      // 根据唯一分解定理，去掉i的幂
      while (n % i == 0) n /= i;
    }
  }
  // 如果最后n>1，则为一个大于根号n的一个质因子
```

```
    if (n > 1) ans = ans / n * (n - 1);
    return ans;
}
```

2. 如果求 1 到 n 的所有欧拉函数值，利用线性筛即可求出

```
// 整体框架为线性筛素数的代码，中间根据欧拉函数的性质加了几条语句而已
int n, phi[N], prime[N], tot;
bool not_prime[N]; // true表示不是素数，false表示是素数
void getPhi() {
    int i, j, k;
    phi[1] = 1;
    for (i = 2; i <= n; ++i) {
        if (!not_prime[i]) {
            prime[++tot] = i;
            phi[i] = i-1; // 根据性质2
        }
        for (j = 1; j <= tot; ++j) {
            k = i * prime[j];
            if (k > n) break;
            not_prime[k] = true;
            if (i % prime[j] == 0) { // 变形1
                phi[k] = prime[j] * phi[i];
                break;
            } else { // 变形2
                phi[k] = (prime[j]-1) * phi[i];
            }
        }
    }
}
```

例题：POJ 2478

2.6 费马小定理

2.6.1 先科普两个概念：

1. 剩余类：

   一个整数被正整数 n 除后，余数有 n 种情形：0，1，2，3，$\ldots$，$n-1$，它们彼此对模 n 不同余。这表明，每个整数恰与这 n 个整数中某一个对模 n 同余。这样一来，按模 n 是否同余对整数集进行分类，可以将整数集分成 n 个两两不相交的子集。我们把（所有）对模 n 同余的整数构成的一个集合叫做模 n 的一个**剩余类**。

2. 完全剩余系：

   任取整数 n，那么 0，1，$\ldots$，$n-1$ 这 n 个数称为模 n 的一个**完全剩余系**。每个数称为相应剩余类类的代表元。最常用的完全剩余系是 $\{0$，1，$\ldots$，$n-1\}$。

2.6.2 费马小定理（要求模数为素数）

若 p 为素数，整数 a 不是 p 的倍数（即$\gcd(a,p)=1$，等价吗？），则 $a^{p-1}\equiv 1(mod\,p)$。

- 先来证明一个性质：

  设一个素数为 p，我们取一个不为 p 的倍数的数 a。

  构造一个序列：$A=1,2,3,\ldots,p-1$，这个序列有这样一个性质：

  $\prod_{i=1}^{n} A_i \equiv \prod_{i=1}^{n}(A_i\times a)\ (mod\ p)$

- 证明：

  因为$gcd(A_i,p)=1, gcd(A_i \times a, p)=1$（有疑问吗？）

  又因为每一个 $A_i \times a \ (mod\ p)$ 都是独一无二的（反证法证明假设存在同余的，如果我还记得就讲一下），且 $A_i \times a \ (mod\ p) < p$，

  所以每一个 $A_i \times a$ 都对应一个 A_i。

  设 $f=(p-1)!$，则 $f \equiv a \times A_i \times a \times A_2 \times a \times A_3 \times \cdots \times A_{p-1} \ (mod\ p)$

  所以 $a^{p-1} \times f \equiv f \ (mod\ p)$

  证毕。

- 由上面的性质证明中的 $a^{p-1} \times f \equiv f \ (mod\ p)$，又因为 $gcd(f,p)=1$，所以 $a^{p-1} \equiv 1 \ (mod\ p)$ 。从而证明了费马小定理。
- 另一个形式：若 p 为素数，对于任意整数 a，有 $a^p \equiv a (mod\ p)$。
  - 证明：

    因为 p 为质数，所以对于任意整数 a 只有两种情况：

    当 a 为 p 的倍数时，显然成立；

    当 a 不为 p 的倍数时，则必有 $gcd(a,p)=1$，那么费马小定理直接拿来用，有 $a^{p-1} \equiv 1 \ (mod\ p)$。

    由同余的同乘性，显然成立。

2.6.3 欧拉定理

费马小定理是用来阐述在素数模下，指数的同余性质。当模是合数的时，就要应用范围更广的欧拉定理了。

若 $gcd(a,m)=1$，则 $a^{\varphi(m)} \equiv 1 \ (mod\ m)$。

特别的，当 m 为质数时，与费马小定理一致。

证明略吧……

2.6.3 扩展欧拉定理

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(p)}, & \gcd(a,p)=1 \\ a^b, & \gcd(a,p) \neq 1, b < \varphi(p) \\ a^{b \bmod \varphi(p) + \varphi(p)}, & \gcd(a,p) \neq 1, b \geq \varphi(p) \end{cases} \pmod p$$

具体证明可参考 https://oi-wiki.org/math/fermat/

2.6.4 例题

1. [SPOJ #4141 "Euler Totient Function"Difficulty: CakeWalk]
2. [UVA #10179 "Irreducible Basic Fractions"Difficulty: Easy]
3. [UVA #10299 "Relatives"Difficulty: Easy]
4. [UVA #11327 "Enumerating Rational Numbers"Difficulty: Medium]
5. [TIMUS #1673 "Admission to Exam"Difficulty: High]

3 最大公约数

一组数的公约数，是指同时是这组数中每一个数的约数的数。而最大公约数，则是指所有公约数里面最大的一个，常缩写为 gcd（Greatest Common Divisor）。

求 gcd 常用的方法为**辗转相除法（欧几里得算法）**，还有一种为**更相减损法**，其实本质是一样的。

3.1 辗转相除法

3.1.1 辗转相除法用来求两个数的最大公约数

其原理为：$gcd(x,y)=gcd(y,x\%y)$。

```
// 递归形式
int gcd(int x, int y) {
    return (y == 0 ? x : gcd(y, x%y));
}

// 非递归形式
int gcd(int x, int y) {
    int r = x % y; // 取余数
    while (r) { // 余数不为0，交换变量，继续做除法
        x = y;
        y = r;
        r = x % y;
    }
    return y; // 余数为0时，除数为gcd
}
```

证明：

3.1.2 多个数的最大公约数

怎么求多个数的最大公约数呢？

显然答案一定是每个数的约数，那么也一定是每相邻两个数的约数。我们每次取出两个数求出答案后再放回去，直到取完所有的数即可。

3.1.3 最小公倍数

对于两个整数 a 和 b，我们根据唯一分解定f理将其表示出来。

可以看出，最小公倍数 $lcm(a,b)$ 与最大公约数 $gcd(a,b)$ 的乘积正好为 $a\times b$，即：$gcd(a,b)\times lcm(a,b)=a\times b$。

所以我们可以先求两者的最大公约数，就能方便得出最小公倍数了。

3.1.3 多个数的最小公倍数

可以发现，当我们求出两个数的 gcd 时，求最小公倍数是 $O(1)$ 的复杂度。

那么对于多个数，我们其实没有必要求一个共同的最大公约数再去处理。类似于求多个数的最大公约数，我们可以先算出两个数的最小公倍数，放入原序列，继续与第三个数求最小公倍数，以此类推即可。

3.2 扩展欧几里得

扩展欧几里德定理（Extended Euclidean algorithm, EXGCD），常用于求 $ax+by=gcd(a,b)$ 的一组整数解（这里 x,y 为未知数）。

证明略讲一下打打酱油……

$ax_1 + by_1 = gcd(a, b)$

$bx_2 + (a\%b)y_2 = gcd(b, a\%b)$

$\because gcd(a, b) = gcd(b, a\%b)$

$\therefore ax_1 + by_1 = bx_2 + (a\%b)y_2$

$\because a\%b = a - \lfloor \frac{a}{b} \rfloor \times b$

$\therefore ax_1 + by_1 = bx_2 + (a - \lfloor \frac{a}{b} \rfloor \times b)y_2 = ay_2 + b(x_2 - \lfloor \frac{a}{b} \rfloor y_2)$

因为系数相同，所以我们可以让 $x_1 = y_2, y_1 = x_2 - \lfloor \frac{a}{b} \rfloor y_2$

```
// 该函数求解 ax+by=1 的整数解
int exgcd(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return b; // b为最大公约数
    }
    // 下面的x、y交换是很直观的根据推到式子来的
    int ret = exgcd(b, a%b, x, y);
    int t = x;
    x = y;
    y = t - a/b*y;
    return ret;
}
```

3.3 例题

4 乘法逆元

乘法逆元是模意义下的乘法运算的逆元，应用比较广泛。

4.1 定义

若 $a \times x \equiv 1 \pmod b$，则称 x 为 a 在模 b 意义下的乘法逆元，记为 a^{-1}。

注意：并非所有的情况下都存在乘法逆元，但是当 $gcd(a, b) = 1$，即 a, b 互质时，存在乘法逆元。

4.2 逆元是干什么的

首先对于除法取模不成立，即 $(a \div b)\%p \neq ((a\%p) \div (b\%p))\%p$ 。

显然数学家们是不能忍受这种局面的，他们扔出了“逆元”来解决这个问题。

因为取模运算对于乘法来说是成立的，逆元就是把除法取模运算转化为乘法取模运算。

$$(a/b)\%p = m \tag{1}$$

假设存在一个数 x 满足

$$a \times x\%p = m \tag{2}$$

由模运算对乘法成立，对 (1) 式两边同时乘以 b，得到：

$$a\%p = m \times (b\%p)\%p;$$

如果 a 和 b 均小于模数 p 的话，上式可以改写为：

$$a = m \times b\%p$$

等式两边再同时乘以 x，联立 (2) 式比较得到：

$$a \times x\%p = m\%p = x \times m \times b\%p$$

因此可以得到：

$$b \times x\%p = 1$$

可以看出 x 是 b 在模 p 意义下的逆元。

由以上过程我们看到，**求取** $(a/b)\%p$ **等同于求取** $a \times b^{-1}\%p$。 因此，求模运算的除法问题就转化为求一个数的逆元问题了。

4.3 如何求乘法逆元

求一个数的逆元，有两种方法。

4.3.1 费马小定理求逆元

因为我们可能会遇到模数为一个素数 p，所以可以利用费马小定理来求。

例如：求整数 a 在模 p 意义下的逆元，如果 a 是 p 的倍数，显然不存在；如果 a 不是 p 的倍数，由 $a^{p-1} \equiv 1 \pmod p$，即 $a^{p-1}\%p = a \times a^{p-2}\%p = 1$。

所以 $a^{p-2}\%p$ 即为 a 在模 p 意义下的逆元，用快速幂求解即可。

```
typedef long long ll;
ll quickpow(ll a, ll n, ll p) { //快速幂求 a^n % p
ll ans = 1;
while(n) {
    if(n & 1) ans = ans * a % p;
    a = a * a % p;
    n >>= 1;
}
    return ans;
}

ll niyuan(ll a, ll p) { //费马小定理求逆元 a^(p-2)%p
    return quickpow(a, p - 2, p);
}
```

注意：该方法的前提条件：模数为素数，且 a 不是 p 的倍数。

4.3.2 欧拉定理求逆元

当模数不是素数时，我们可以用欧拉定理来求解，再回顾一下欧拉定理：若 $\gcd(a, n) = 1$，则 $a^{\varphi(n)} \equiv 1 \pmod n$。那么 $a^{\varphi(n)-1}$ 即为 a 在模 n 意义下的逆元，快速幂搞它。代码就略过吧……

4.3.3 扩展欧几里得求逆元

根据逆元的定义，若 $a \times x \equiv 1 \pmod b$，则称 x 为 a 在模 b 意义下的乘法逆元。那么，我们可以把原式转换成 $a \times x = b \times y + 1$，移项得：$a \times x - b \times y = 1$，因为 y 为商，可以变个符号，让原式变为 $a \times x + b \times y = 1$，这就变成了我们熟悉的形式。我们知道，当 $gcd(a, b) = 1$ 时，方程才有整数解，因此，利用扩展欧几里得求逆元，要求 a, b 互质。

```
// ax+by=1
int exgcd(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return b; // b为最大公约数
    }
    // 注意传参及下面的计算
    int ret = exgcd(b, a%b, y, x);
    y -= a/b*x;
    return ret;
}

// ax+by=1
int exgcd2(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return b; // b为最大公约数
    }
    // 下面的x、y交换是很直观的根据推到式子来的
    int ret = exgcd2(b, a%b, x, y);
    int t = x;
    x = y;
    y = t - a/b*y;
    return ret;
}
```

4.3.3 线性求逆元

要求 $1, 2, \ldots, p-1$ 中每个数关于 p 的逆元（p 为素数），用以上两种方法，就显得有点慢了，下面讲一下线性求逆元。

> 首先，$1^{-1} \equiv 1 \pmod p$，显然成立。
>
> 设 $p = k \times i + r$，其中 $1 < i < p, r < i$
>
> 再放到模 p 意义下，则有：$k \times i + r \equiv 0 \pmod p$。
>
> 两边同时乘以 $i^{-1} \times r^{-1}$，则：
>
> $$\begin{aligned} k \times i \times i^{-1} \times r^{-1} + r \times i^{-1} \times r^{-1} &\equiv 0 && \pmod p \\ k \times r^{-1} + i^{-1} &\equiv 0 && \pmod p \\ i^{-1} &\equiv -k \times r^{-1} && \pmod p \\ i^{-1} &\equiv -\lfloor \frac{p}{i} \rfloor \times r^{-1} && \pmod p \\ i^{-1} &\equiv -\lfloor \frac{p}{i} \rfloor \times (p \bmod i)^{-1} && \pmod p \end{aligned}$$
>
> 所以，我们可以用以上递推式来求解逆元，代码很简单。

```
ny[1] = 1;
for (int i = 2; i < p; ++i) {
    ny[i] = (long long)-(p / i) * ny[p % i] % p; // 注意最后的模 p 不要忘记
}
```

但是有时候上面的方法会得到负数，而我们往往需要的是不大于 p 的正整数，所以可以做一下改动：

```
// 因为 1<i<p，所以 p/i 一定小于 p
ny[1] = 1;
for (int i = 2; i < p; ++i) {
    ny[i] = (long long)(p - p / i) * ny[p % i] % p; // 注意最后的模 p 不要忘记
}
```

注意：求逆元往往涉及大量的乘法，所以运算的时候一定要注意是否需要用到 long long。

4.3.4 小结

方法	限定条件	时间复杂度	备注
费马小定理	模数为素数	$O(logn)$	
欧拉定理	a 与 p 互质	差不多是 $O(\sqrt{n})$	
扩展欧几里得	a 与 p 互质	据说为 $O(\ln n)$	可以在求解过程中判断是否互质
线性递推	模数为素数	$O(n)$	