2整除

- **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 ： $a = kb \pm c \Rightarrow c = \pm(a - kb) \Rightarrow d|c$

定理11.0.1： 在自然数集中，小于 n 的质数约有 $\dfrac{n}{\ln(n)}$ 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 $n < p < 2n$。

质数

判断质数

- $kn + i$ **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;
}
```

线性筛

- **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;
        }
    }
}
```

上取整转下取整

引理(分数的上取整转换下取整)

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

反素数

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 的正约数个数都**小于** n 的约数个数，即 n 是 $1\ldots n$ 中约数个数最多的数。

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：

引理11.3.1：$1\sim N$ 中最大的反素数，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综合不超过 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 $2^{c_1}\times3^{c_2}\times5^{c_3}\times7^{c_4}\times11^{c_5}\times13^{c_6}\times17^{c_7}\times19^{c_8}\times23^{c_9}\times29^{c_{10}}$，且 $c_1\geq c_2\geq c_3\geq\cdots\geq c_{10}\geq0$，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。

引理证明待更（其实特别显然，自己看看就能看懂）

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

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

复制

```
#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

思路同上，只不过要改改 `dfs` 的返回条件。注意这样的题目的数据范围，用 `int` 会溢出，在循环里可能就出不来了就超时了

```cpp
#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;
}
```

质因数分解

Pollard Rho算法

模板

```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;
}
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;
}
```

例题：

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 $O(N\log N)$

```
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;
}
```

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：$1+2+3+\cdots+\alpha+1=\dfrac{(\alpha+1)(\alpha+2)}{2}$。（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：$\prod_{i=1}^k\dfrac{(\alpha_i+1)(\alpha_i+2)}{2}$。

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。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define IOS ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
ll n;
const int mod = 1e7+7;
int p[500050],cnt;
bool vis[1000050];
void init(int n)
{
    vis[1]=1;
    for(int i=2;i<=n;i++){
        if(vis[i]==0) p[++cnt]=i;
        for(int j=1;j<=cnt&&i*p[j]<=n;j++){
            vis[i*p[j]]=1;
            if(i%p[j]==0) break;
```

```
        }
    }
}
ll divide(ll x)
{
    ll ans=0,tmp=n;
    while(tmp){
        ans+=tmp/x; tmp/=x;
    }
    return ans;
}
int main(){
init(1e6);
while(1){
    cin>>n;
    if(n==0) break;
    ll ans=1;
    for(int i=1;i<=cnt&&p[i]<=n;i++){
        ll tmp=divide(p[i]);
        ans*=(tmp+1)*(tmp+2)/2;
        ans%=mod;
    }
    cout<<ans<<'\n';
}
    return 0;
}
```

整除分块

整除分块

整除分块是用于快速处理形似

$$\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor$$

的式子的方法

很显然，这个可以$O(n)$得到答案。但是，在某些题目中，毒瘤出题人将数据加强到了10^{10}以上，这个时候我们就无法通过$O(n)$的解法来得到答案了。我们需要一个$O(\sqrt{n})$的更为优秀的解法

首先观察这个式子，找几个特殊值代入

`n=5时，sum=5+2+1+1+1`

可以发现的是：（这里给的例子并不明显，其实应该找一个大的n来代入才直观，读者可以自行尝试）

对于单一的$\lfloor\frac{n}{i}\rfloor$，某些地方的值是相同的，并且**呈块状分布**

通过进一步的探求规律与推理~~以及打表与瞎猜~~，我们可以惊喜的发现一个规律，这些**块状分布的值是有规律的**

对于一个块，**假设它的起始位置的下标为l，那么可以得到的是，它的结束位置的下标为**$\lfloor\frac{n}{\lfloor\frac{n}{l}\rfloor}\rfloor$

如果实在看的有点懵逼，可以继续采用代入特殊值的方法，验证一下上方的规律，用程序表现出来即为

```
//l为块的左端点，r为块的右端点
r=n/(n/l)
```

在实际应用中，需要注意的就是**除法除0**的问题（一般都需要特判一下n/l）

```
for(int l=1,r;l<=n;l=r+1){
    r=n/(n/l);
    //do something
}
```

$\sum_{i=1}^{n}\lfloor\frac{n}{i}\rfloor * i$的处理方法

例题：[BZOJ1257: [CQOI2007]余数之和]

这题其实就是求

$$\sum_{i=1}^{n} k \ mod \ i$$

这题和整除分块又有什么关系呢?
mod没有什么特殊的性质，所以我们将它展开来，就变成了

$$\sum_{i=1}^{n} k \ - \lfloor\frac{k}{i}\rfloor * i$$

于是我们就看到了一个熟悉的形式，也就是整除分块的一般形式

再次改一下这个式子

$$n * k - \sum_{i=1}^{n}\lfloor\frac{k}{i}\rfloor * i$$

那么$\sum_{i=1}^{n}\lfloor\frac{k}{i}\rfloor * i$和普通的整除分块有什么差别呢?

~~其实就是多了一个i~~

确实，就是多了一个i而已，只需要简单的化简一下，这个i就对我们的处理没有什么影响了

因为我们知道，对于一个整除分块$\sum_{i=l}^{r}\lfloor\frac{k}{i}\rfloor$，其中的每个值都是相同的，于是我们可以设$T = \lfloor\frac{k}{i}\rfloor$

式子就化为了

$$\sum_{i=l}^{r} T * i$$
$$= \sum_{i=l}^{r} T * \sum_{i=l}^{r} i$$

也就是说，其实这个式子前半段是一个整除分块，后半段是一个首项为l，公差为1的等差数列

至此，我们就圆满的解决了这个问题，可以在$O(\sqrt{n})$的时间内解决本题

这是整除分块中最基础的应用，就是单纯的利用整除分块来加速递推的实现，而实际上，整除分块更多的与其他函数结合在一起来使用，优化问题的求解

约数

随机数据下，约数个数的期望为$O(\ln \ n)$

最大公约数

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）。

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

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 （$0\leq k<b$），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。

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$G=\gcd(C_n^1,C_n^2,\ldots C_n^{n-1})$

- 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： $(n+1)\operatorname{lcm}(C_n^0, C_n^1, \cdots, C_n^n) = \operatorname{lcm}(1, 2, \cdots, n+1)$

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

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

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

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： $f_n = \dfrac{f_{n+2} + f_{n-2}}{3}$

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

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

互质与欧拉函数

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

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

由于欧拉函数是积性函数，由 **性质14.2.3** 得：

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

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

```
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;
}
```

线性筛求欧拉函数

```
void init(int n)
{
    vis[1]=1; phi[1]=1;
    for(int i=2;i<=n;i++){
        if(vis[i]==0){p[++cnt]=i; phi[i]=i-1;}
        for(ll j=1;j<=cnt&&i*p[j]<=n;j++){
            vis[i*p[j]]=1;
            if(i%p[j]==0){
                phi[i*p[j]]=phi[i]*p[j]; break;
            }
```

```
            else phi[i*p[j]]=phi[i]*(p[j]-1);
        }
    }
}
```

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$n\times\dfrac{\varphi(n)}{2}$

> 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$p^2\not\mid n$,则 $\varphi(n)=\varphi(\frac{n}{p})\times(p-1)$。

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

推论14.2.7： $p>2\Longrightarrow[\varphi(p)\mod 2=0]$

推论14.2.8： $p\in\{Prime\}\Longrightarrow\varphi(p^k)=p^k-p^{k-1}$

推论14.2.9： $\sum_{i=1}^{n}i[\gcd(i,n)=1]=\dfrac{n\varphi(n)+[n=1]}{2}$ （例题）

推论14.2.10： $f(n)=\sum_{i=1}^{n}[\gcd(i,k)=1]=\dfrac{n}{k}\varphi(k)+f(n\mod k)$

推论14.2.11： 若 i,j 不互质，则 $\varphi(i\times j)=\dfrac{\varphi(i)\varphi(j)\gcd(i,j)}{\varphi(\gcd(i,j))}$

容斥

奇加偶减

待更

乘法

快速幂

```
ll qpow(ll a, ll b, ll q)
{
    ll res = 1;//因为是用乘法模拟乘方，所以res要是1
    while(b) {
        if(b & 1)res = (res * a) % q;
        a = (a * a) % q;//视情况将 * 换成Mul(龟速乘)
        b >>= 1;
    }
    return res % q;
}
```

龟速乘

在模意义下计算乘法，如果 c 较大（但是不超过 $long\ long$ 范围），进行乘法的两个数同样很大，直接**乘会爆掉**（例如快速幂里的乘法），我们可以用类似快速幂的快速乘计算，时间复杂度为 $O(logb)$。因为慢于 $O(1)$ 的乘法运算符，所以我们常常把这个叫做龟速乘。经常用与快速幂中代替普通乘法。

```
ll Mul(ll a, ll b, ll p)
{
    if(b < 0) a = - a, b = - b;
    ll res = 0;//因为是加法模拟乘法，所以res开始为0
    while(b) {
        if(b & 1) res = (res + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return res;
}
```

```
ll Mul(ll a, ll b, ll p)
{
    if(b < 0) a = - a, b = - b;
    ll res = 0;//因为是加法模拟乘法，所以res开始为0
    while(b) {
        if(b & 1) res = (res + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return res;
}
```

快速乘

```
ll Mul(ll x,ll y,ll p)
{
    if(y < 0) x = - x, y = - y;
    ll z = (long double)x / p * y;
    ll res = (unsigned long long)x * y - (unsigned long long)z * p;
    return (res + p) % p;
}
```

同余

同余的基本性质：

性质21.1.1： (自反性)$a \equiv a(\mod m)$

性质21.1.2： (对称性)：若$a \equiv b(\mod m)$，则$b \equiv a(\mod m)$

性质21.1.3： (传递性)：若$a \equiv b(\mod m), b \equiv c(\mod m)$，则$a \equiv c(\mod m)$

性质21.1.4： (同加性)：若$a \equiv b(\mod m)$，则$a \pm c \equiv b \pm c(\mod m)$

性质21.1.5： (同乘性)：若$a \equiv b(\mod m)$，则$a \times c \equiv b \times c(\mod m)$，若$a \equiv b(\mod m), c \equiv d(\mod m)$，则$a \times c \equiv b \times d(\mod m)$

性质21.1.6： (同幂性)：若$a \equiv b(\mod m)$，则$a^c \equiv b^c(\mod m)$

性质21.1.7： (不满足同除性)：若$a \equiv b(\mod m)$不满足$a \div c \equiv b \div c(\mod m)$

性质21.1.8： (满足同除性)：若 $c|a$，$c|b$，则$\frac{a}{c} \equiv \frac{b}{c}(\mod \frac{m}{\gcd(m,c)})$

推论21.1.9： 若 $da \equiv db(\mod m)$ 则$a \equiv b(\mod \frac{m}{\gcd(m,d)})$ (这在取遍剩余系会用到)

推论21.1.10： 若 $a \equiv b(\mod m)$ $m' \mid m$，$a \equiv b \pmod{m'}$

推论21.1.11： $a \equiv b \pmod{m_i} (i = 1..k)$ 等价于 $a \equiv b \pmod{M} = [m1, m2, ..mk]$

推论21.1.12： $\begin{cases} a \equiv b \pmod m \\ c \equiv d \pmod m \end{cases} \Longrightarrow a + c \equiv b + d \pmod m$

推论21.1.13： $\begin{cases} a \equiv b \pmod m \\ c \equiv d \pmod m \end{cases} \Longrightarrow a - c \equiv b - d \pmod m$

推论21.1.14： $ka \equiv kb \pmod m, \gcd(k, m) = 1 \Longrightarrow a \equiv b \pmod m$

注意：`-4 % 5 = -4，-6 % 5 = -1`，所以如果题目要求的是最小正整数那么我们就需要对答案x：`(x % b + b ) % b`

费马小定理

0x21.2 费马小定理

若 p 是质数，则对于任意整数 a 都有 $a^p \equiv a(\mod p)$。也可以转换为 $a^{p-1} \equiv 1(\mod p)$ (a 与 p 互质)

> 证明
> 因为 p 是质数，且$(a, p) = 1$，所以 $\varphi(p) = p - 1$。
> 由欧拉定理可得 $a^{p-1} \equiv 1 \pmod p$。证毕。
> 对于该式又有 $a^p \equiv a \pmod p$，而且此式不需要 $(a, p) = 1$，
> 所以，费马小定理的另一种表述为：假如 p 是质数，a 是整数，那么$a^p \equiv a \pmod p$。

费马小定理降幂：$a^k \equiv a^{k \mod (p-1)} (\mod p)$ (a 与 p 互质)

费马大定理：

- $m > 2$时，$x^m + y^m = z^m$ 无正整数解
- 当$m = 2$，对于式子$a^2 + b^2 = c^2$ (n为任意正整数)：
  - 当 a 为奇数时：$a = 2n + 1, c = n^2 + (n + 1)^2, b = c - 1$
  - 当 a 为偶数时：$a = 2n + 2, c = 1 + (n - 1)^2, b = c - 2$

欧拉定理

0x21.3 欧拉定理

- **欧拉定理**

定理21.3.1： 若正整数 a, n 互质，则 $a^{\varphi(n)} \equiv 1(mod\ n)$ 其中 $\varphi(n)$ 是欧拉函数。

> 证明
> 设 $x_1, x_2, \cdots, x_{\varphi(n)}$是一个以 n 为模的简化剩余系，则 ax_1，$ax_2, \cdots, ax_{\varphi(n)}$ 也是一个以 n 为模的简化剩余系（因为 $(a，n) = 1$）。于是有ax_1，$ax_2, \cdots, ax_{\varphi(n)} \equiv x_1, x_2, \cdots, x_{\varphi(n)}\ (\bmod\ n)$，所以 $a^{\varphi(n)} \equiv 1\ (\bmod\ n)$。
> 证毕。

（费马小定理是欧拉定理的一种特殊情况）

推论21.3.2： $\exists x \in N^*, a^x \equiv 1(\mod m) \iff \gcd(a, m) = 1$（证明）（例题）

- **欧拉降幂（拓展欧拉定理）**

若 a 与 m 互质：

$$a^b \equiv a^{b \mod \varphi(m)} (\mod m)$$

> **证明**：
> 设 $b = q \times \varphi(m) + r$，其中 $0 \le r \le \varphi(m)$，即 $r = b \mod \varphi(m)$。
> $a^b \equiv a^{q \times \varphi(m) + r} \equiv (a^{\varphi(m)})^q \times a^r = 1^q \times a^r \equiv a^r \equiv a^{b \mod \varphi(m)} (\mod m)$
> 定理得证 □

若不保证 a 与 m 互质：$b > \varphi(m)$ 时：$a^b \equiv a^{b \mod \varphi(m) + \varphi(m)} (\mod m)$

> 太长不证，证明详见：https://www.cnblogs.com/1024th/p/11349355.html

在一些计数的问题中，常常要求对结果取模，但是在计算非常庞大的次幂的时候，无法直接取模，可以先把底数对 p 取模，指数对 $\varphi(p)$ 取模，再计算次幂，有效地降低时间复杂度。

威尔逊定理

0x21.4 威尔逊定理

威尔逊定理

定理21.4.1： 当 p 为质数时有：$(p-1)! \equiv p-1 \equiv -1(\mod p)$，$(p-2)! \equiv 1(\mod p)$

p可整除(p-1)!+1是p为质数的充要条件

其中 **定理21.4.1** 实际上就等价于：若 p 是质数，则 $(p-1)! + 1$ 能够被 p 整除。

n 为素数时：$(n-1)! \mod n = 1$
n 为合数时：除 $n = 4$ 以外，$(n-1)! \mod n = 0$

威尔逊定理的逆命题

定理21.4.2： 若一个数 x，满足条件 $(x-1)! + 1$可以被 x 整除，那么 x 是素数。

0x22.1 裴蜀（Bézout）定理

定理22.1.1：设 a, b 是不全为零的整数，存在无穷多组整数对 (x, y)，满足不定方程 $ax+by=d$，其中 $d=\gcd(a,b)$ 即：$ax+by=\gcd(a,b)$。

推论22.1.2：$\gcd(a,b) \mid c \iff \exists x, y \in Z, ax+by=c$

> 方程 $ax+by=d(d=\gcd(a,b))$ 即为丢番图方程。

推论21.1.3：$\forall a, b, z \in \mathbb{N}^*, \gcd(a,b)=1, \exists x, y \in \mathbb{N}, ax+by=ab-a-b+z$，即两互质的数 a, b，表示不出的最大的数为 $ab-a-b$。

> **推论21.1.3证明**：不妨设 $a<b$，假设答案为 x。
> 若$x \equiv ma \pmod b (1 \le m \le b-1)$（若$m \ge b$，$m \equiv 0 \pmod b$）
> 即$x=ma+nb(1 \le m \le b-1)$
> 显然当 $n \ge 0$ 时 x 可以用 a, b 表示出来，不合题意。
> 因此当 $n=-1$ 时 x 取得最大值，此时$x=ma-b$。
> 显然当 m 取得最大值 $b-1$ 时 x 最大，此时 $x=(b-1)a-b=ab-a-b$
> 即 a, b 所表示不出的最大的数是 $ab-a-b$

拓展欧几里得算法

0x22.2 扩展欧几里德算法

我们可以利用欧几里得算法求解 $ax+by=\gcd(a,b)$ 中的 x 和 y 。

我们知道欧几里得算法利用的核心性质为 $\gcd(a,b)=\gcd(b, a \bmod b)=\gcd(b, a-b\lfloor\frac{a}{b}\rfloor)$

根据裴蜀定理，我们一定可以找到四个整数 x, y, x', y'，使得 $ax+by=\gcd(a,b)$ 且 $bx'+(a-b\lfloor\frac{a}{b}\rfloor)y'=\gcd(b, a-b\lfloor\frac{a}{b}\rfloor)$

由于 $\gcd(a,b)=\gcd(b, a \bmod b)=\gcd(b, a-b\lfloor\frac{a}{b}\rfloor)$

有：

$$
\begin{gathered}
ax+by=bx'+(a-b\lfloor\frac{a}{b}\rfloor)y' \\
a(x-y')+b(y-(x'-\lfloor\frac{a}{b}\rfloor y'))=0
\end{gathered}
$$

我们希望这个等式对一切 a，b 都成立，于是 $x=y'$ 且 $y=x'-\lfloor\frac{a}{b}\rfloor y'$ 。

显然，我们想求 x 和 y，只需要求出 x'，y'，由于 x'，y' 对应的问题相同且规模更小，所以可以进行递归的运算。边界条件为：当 $b=0$ 时，$x=1, y=0$，$a \times 1+0 \times 0=\gcd(a,0)$ 。

由于算法思想与欧几里得相同，我们称之为拓展欧几里得算法。

实现：

在用欧几里德算法求 $d=\gcd(a,b)$ 的过程中求方程 $ax+by=d$ 的一组整数解 (x, y)
若 $d \mid c$,不妨设 $c=kd$，则有$a(kx)+b(ky)=c$，否则原方程无整数解。

```
//在gcd的过程上增加了拓展
inline int exgcd(int a, int b, int &x, int &y)
{
    if(b == 0){
        x = 1, y = 0, return a;
    }
    int d = exgcd(b, a % b, x, y);
    int z = x;x = y, y = z - y * (a / b);
    return d;
}
```

`exgcd` 可得到 $ax+by=\gcd(a,b)$ 的解，对于 $ax+by=c$ 的解，我们只需要根据**定理22.3.3**构造即可。

```
//在gcd的过程上增加了拓展
ll exgcd(ll a, ll b, ll &x, ll &y)
{
    if(b == 0){
        x = 1, y = 0; return a;
    }
    int d = exgcd(b, a % b, x, y);
    int z = x;x = y, y = z - y * (a / b);
    return d;
}
```

0x22.3 解二元模线性方程

二元模线性方程（二元一次不定方程）：形如 $ax \equiv c \pmod b$ 或 $ax+by=c$ 。其中 a,b,c,x,y 均为整数。

定理22.3.1：上述方程有解的充要条件是 $\gcd(a,b) \mid c$

可以理解为 $\gcd(a,b)$ 是 $ax+by$ 可以表示出来的最小的正整数。

定理22.3.2：方程$ax+by=d, d=\gcd(a,b)$的所有解为：

$$\begin{cases} x = x_0 + k\dfrac{b}{d} \\ y = y_0 - k\dfrac{a}{d} \end{cases}$$

其中 x_0，y_0 是一组特解，$k \in \mathbb{Z}$ 。

> **证明：**
> 方程的特解 (x_0,y_0)，任取另一组解 (x,y)，则 $ax_0+by_0=ax+by=\gcd(a,b)$。变形得 $a(x_0-x)=b(y-y_0)$ 。设 $\gcd(a,b)=d$，方程左右两边同时除以 d（如果 $d=0$ ，说明 a 或 b 等于 0），得 $a'(x-x_0)=b'(y_0-y)$，其中 $a'=\frac{a}{d}$，$b'=\frac{b}{d}$。显然此时 a' 和 b' 互质，因此 $x-x_0$ 一定是 b' 的整数倍（因为 a' 中不包含 b' ，所以 x_0-x 一定包含 b' ）。设它为 kb' ，即 $x-x_0=k\frac{b}{d}$，同理，有 $y_0-y=k\frac{a}{d}$。

定理22.3.3：方程 $ax+by=c, \gcd(a,b) \mid c$ 的所有解为

$$\begin{cases} x = \dfrac{c}{d}x_0 + k\dfrac{b}{d} \\ y = \dfrac{c}{d}y_0 - k\dfrac{a}{d} \end{cases}$$

其中 x_0，y_0 是方程$ax+by=d, d=\gcd(a,b)$ 的一组特解，$k \in \mathbb{Z}$。

Z 是整数集，也就意味着k 可以为负数，即最小正整数解：

若 $x>0 \to x \mod \frac{b}{d}$

乘法逆元

费马小定理要求p必须为质数 扩欧可以不需要

费马小定理

```
ll qpow(ll a, ll b, ll q)
{
    ll res = 1;//因为是用乘法模拟乘方，所以res要是1
    while(b) {
        if(b & 1)res = (res * a) % q;
        a = (a * a) % q;//视情况将 * 换成Mul(龟速乘)
        b >>= 1;
    }
    return res % q;
}
ll inv(ll x,ll p) {return qpow(x, mod - 2, p) % p;}
```

扩展欧几里得

```
ll exgcd(ll a, ll b, ll &x, ll &y)
{
    if(b == 0){
        x = 1, y = 0; return a;
    }
    int d = exgcd(b, a % b, x, y);
    int z = x;x = y, y = z - y * (a / b);
    return d;
}
扩展欧几里得用于在已知a,b的情况下，求解出一组解x,y，使之满足ax+by=gcd(a,b)=d。
```

线性递推

```
int n, m;
int inv[N], p;
int main()
{
    scanf("%d%d", &n, &p);
    inv[1] = 1;
    puts("1");
    for(int i = 2; i <= n; ++ i) {
        inv[i] = (ll)(p - p / i) * inv[p % i] % p;
        printf("%d\n", inv[i]);
    }
    return 0;
}
```

0x23.5 求阶乘的逆元

定义 $\text{inv}[i]$ 为 $i!$ 的逆元

我们知道 $\text{inv}[i+1]=\dfrac{1}{i+1!}$

两边同时乘上 $i+1$ 得 $\text{inv}[i+1]\times(i+1)=\dfrac{1}{i!}=\text{inv}[i]$。

我们只需要先求出 $\text{inv}[n]$然后往回递推即可。

当然我们也可以先 $O(n)$ 求出 $1\sim n$ 的所有数的逆元，然后求阶乘即可。

0x22.4 类欧几里德算法（一个求和技巧）

- **竞赛例题选讲**

Problem A 类欧几里德算法 1

求

$$\sum_{i=0}^{n}\left[\frac{ai+b}{c}\right]$$

其中$[\ ]$表示取整。

Solution

```
ll sum_pow(ll n, ll k) {
    if (k == 0) return n;
    else if (k == 1) return n * (n + 1) / 2;
    else if (k == 2) return n * (n + 1) * (2 * n + 1) / 6;
    else if (k == 3) return n * n * (n + 1) * (n + 1) / 4;
    else if (k == 4) return n * (2 * n + 1) * (n + 1) * (3 * n * n + 3 * n - 1) / 30;
    else assert(false);
}
ll EuclidLike1(ll a, ll b, ll c, ll n) {
    if (a == 0) return b / c * (n + 1);
    else if (a >= c || b >= c)
        return (a / c) * sum_pow(n, 1) + (b / c) * (n + 1) + EuclidLike1(a % c, b % c, c, n);
    else
        return (a * n + b) / c * n - EuclidLike1(c, c - b - 1, a, (a * n + b) / c - 1);
}
```

Problem B 类欧几里德算法 2

给定 n,a,b,c，求

$$\sum_{i=0}^{n}\left\lfloor\frac{ai+b}{c}\right\rfloor,\sum_{i=0}^{n}\left\lfloor\frac{ai+b}{c}\right\rfloor^2,\sum_{i=0}^{n}i\left\lfloor\frac{ai+b}{c}\right\rfloor$$

Solution

```
#define Int register int
#define mod 998244353ll
#define int long long
int inv2 = 499122177ll,inv6 = 166374059ll;
struct Ans{int f,g,h;};
Ans Solve (int a,int b,int c,int n)
{
    if (!a) {
        int f = (n + 1) * (b / c) % mod;
        int g = (n + 1) * (b / c) % mod * (b / c) % mod;
        int h = n * (n + 1) % mod * inv2 % mod * (b / c) % mod;
        return Ans {f % mod,g % mod,h % mod};
    }
    else if (a >= c || b >= c) {
        Ans fucker = Solve (a % c,b % c,c,n);
        int F = fucker.f + n * (n + 1) / 2 % mod * (a / c) % mod + (n + 1) * (b / c) % mod;
        int G = fucker.g + 2 * (a / c) % mod * fucker.h % mod + 2 * (b / c) % mod * fucker.f +
        n % mod * (n + 1) % mod * (2 * n % mod + 1) % mod * inv6 % mod * (a / c) % mod * (a / c) % mod +
        n % mod * (n + 1) % mod * (a / c) % mod * (b / c) % mod + (n + 1) * (b / c) % mod * (b / c) % mod;
        int H = fucker.h + n % mod * (n + 1) % mod * (2 * n + 1) % mod * inv6 % mod * (a / c) % mod + n % mod * (n +
        return Ans {F % mod,G % mod,H % mod};
    }
    else {
        int M = (a * n + b) / c;
        Ans fucker = Solve (c,c - b - 1,a,M - 1);
        int F = n * M % mod - fucker.f;
        int G = n * M % mod * (M + 1) % mod - 2 * fucker.h % mod + mod - 2 * fucker.f % mod + mod - F % mod;
        int H = (M * n % mod * (n + 1) % mod - fucker.g + mod - fucker.f) % mod * inv2 % mod;
        return Ans {F % mod,G % mod,H % mod};
    }
}
int read ()
{
    int x = 0;char c = getchar();int f = 1;
```

```
    while (c < '0' || c > '9'){if (c == '-') f = -f;c = getchar();}
    while (c >= '0' && c <= '9'){x = (x << 3) + (x << 1) + c - '0';c = getchar();}
    return x * f;
}
void write (int x)
{
    if (x < 0){x = -x;putchar ('-');}
    if (x > 9) write (x / 10);
    putchar (x % 10 + '0');
}
signed main()
{
    int times = read ();
    while (times --)
    {
        int n = read (),a = read (),b = read (),c = read ();
        Ans Putout = Solve (a,b,c,n);
        write ((Putout.f + mod) % mod),putchar (' '),write ((Putout.g + mod) % mod),putchar (' '),write ((Putout.h +
    }
    return 0;
}
```

0x22.5 整式方程

整式方程就是方程中所有的未知数均在分子上，分母只是常数且无未知数。

通常情况下，常年用字母 x, y, z 来表示未知数，方程中含有几个不同的未知数就叫做几元，未知数的最高次数是几就叫做几次。

例如：$ax + b = c$ 就是一个一元一次整式方程

- **一元一次整式方程**

对于方程 $ax + b = c$，有：$x = \frac{c-b}{a}$

```
double calculate(double a, double b, double c){
    return (c - b) / a;
}
```

- **一元二次整式方程**

```
double x1, x2;
bool calculate(double a, double b, double c){
    double delta = b * b - 4 * a * c;
    if(delta < 0) {
        return false;
    }
    else if(delta == 0) {
        x1 = ( - 1 * b + sqrt(delta) / (2 * a) );
        x2 = x1;
    }
    else {
        x1 = ( - 1 * b + sqrt(delta) / (2 * a) );
        x2 = ( - 1 * b - sqrt(delta) / (2 * a) );
    }
    return true;
}
```

中国剩余定理

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define IOS ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
const int N = 20;
int a[N],m[N],n;
//在gcd的过程上增加了拓展
ll exgcd(ll a, ll b, ll &x, ll &y)
{
    if(b == 0){
```

```
        x = 1, y = 0; return a;
    }
    int d = exgcd(b, a % b, x, y);
    int z = x;x = y, y = z - y * (a / b);
    return d;
}
int main(){
cin>>n;
ll M=1;
for(int i=1;i<=n;i++){
    cin>>m[i]>>a[i]; M*=m[i];
}
ll ans=0;
for(int i=1;i<=n;i++){
    ll tmp=M/m[i]; ll ti,y;
    ll d=exgcd(tmp,m[i],ti,y);
    // ti是tmp在m[i]下的逆元 因为m[i]不保证为质数 所以不能用费马小定理
    ti=(ti%m[i]+m[i])%m[i];
    ans+=a[i]*ti*tmp;
}
cout<<(ans%M+M)%M<<'\n';
    return 0;
}
```

- **同余方程组**

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ \dots \\ x \equiv a_n \pmod{m_n} \end{cases}$$

其中 $a_1, a_2, \dots, a_n$ 是整数，$m_1, m_2, \dots, m_n$是正整数且两两互质。

- **中国剩余定理**

我们设：

$$M = m_1 \times m_2 \times \cdots \times m_n，\ M_i = \frac{M}{m_i}$$

$$M_i \times M_i^{-1} \equiv 1 \quad (\mathrm{mod}\ \ ) m_i$$

，其中M_i^{-1}是M_i的逆元。

我们可以构造出一个解 $x = \sum_{i=1}^{k} a_i M_i M_i^{-1}$

由此，任意解x_0 即为$x + k \times M$

最小正整数解 $x_{\min} = x_0 \,\%\, M$

如何证明构造出来的解 x 对于所有的同余方程都成立呢?
我们首先对第一个式子，$x \bmod m_1$，其中 $i \in 2 \sim k$ 中的 M_i 里面都是 m_i 的倍数，$M_i \bmod m_i$ 都等于 0，$M_1 \times M_1^{-1} \equiv 1 \pmod{m_i}$，也就是说只有第一项 $\bmod m_i$ 不为 0，等于 a_i，那么也就是说满足第一个同余方程，同理也满足所有的同余方程。

Code

```
const ll N = 5e5 + 7;
int n, a[N], m[N];
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if(b == 0){
        x = 1; y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, x, y);
    ll z = x;x = y;y = z - (a / b) * x;
    return d;
}
int main()
{
    ll M = 1;
    scanf("%d", &n);
    for(int i = 1; i <= n; ++ i) {
        scanf("%d%d", &m[i], &a[i]);
        M *= m[i];
    }
    ll res = 0;
    for(int i = 1; i <= n; ++ i) {
        ll Mi = M / m[i];
        ll ti, y;
        //exgcd求逆元：解同余方程：ax + my = 1;(ax ≡ 1 mod m)
        ll d = exgcd(Mi, m[i], ti, y);
        ti = (ti % m[i] + m[i]) % m[i];
        res += a[i] * ti * Mi;
    }
    printf("%lld\n", (res % M + M) % M);//可能为负数，所以需要处理一下
    return 0;
}
```

0x24.3 拓展中国剩余定理

仍然是上面的问题，只是 $m_1 c \dots m_n$ 不保证两两互质了。

考虑如果只有两个方程如何求解：

$$\begin{cases} x \equiv a_1 \pmod{m_1} \qquad (1) \end{cases}$$

$$\lfloor x \equiv a_2 \pmod{m_2} \qquad (2)$$

对于第一个方程而言，解的形式是 $x = a_1 + km_1$ 。带入第二个方程，得到：$a_1 + km_1 = a_2 \pmod{m_2}$

这个方程中只有 k 是未知的。用扩展欧几里得解出来一个 k_0 ，则任意 $km_1 = k_0m_1 + zm_2$ 对于等式 (2) 都是合法的，为了满足等式 (1) 所以要求 $zm_2 = u \times \operatorname{lcm}(m_1, m_2)$ 。现在，x 的表现形式为 $a_1 + k_0m_1 + u \times \operatorname{lcm}(m_1, m_2)$ ，我们合并出了一个新的方程：

$$x \equiv a_1 + k_0m_1 \pmod{\operatorname{lcm}(m_1, m_2)}$$

$z, u \in \mathbb{Z}$。

然后再合并 n 次即可。

Code

```
typedef long long ll;const int N = 100007, M = 1000007, INF = 0x3f3f3f3f;
const double eps = 1e-8;
const int mod = 10007;
ll n, m;
ll bi[N], ai[N];
ll exgcd(ll a, ll b, ll &x, ll &y)
{
    if(b == 0){x = 1; y = 0; return a;}
    ll d = exgcd(b, a % b, x, y);
    ll z = x;x = y;y = z - a / b * y;
    return d;
}
ll mul(ll a, ll b, ll c)//注意数据范围可能会爆long long需要用到龟速乘
{
    if(b < 0) a = - a, b = - b;
    ll res = 0;
    while(b){
        if(b & 1)res  = (res + a) % c;
        a = (a + a) % c;
        b >>= 1;
    }
    return res;
}
ll excrt()//拓展中国剩余定理
{
    ll x, y, k;
    ll M = bi[1], ans = ai[1];//第一个方程的特解
    for(int i = 2; i <= n; ++ i) {
        ll a = M, b = bi[i], c = (ai[i] - ans % b + b) % b;
        ll d = exgcd(a, b, x, y);
        ll bg = b / d;//lcm
        if(c % d != 0)return -1; //判断是否无解，然而这题其实不用
        x = mul(x, c / d, bg);
        ans += x * M;//更新前k个方程组的答案
        M *= bg;//M为前k个m的lcm
        ans = (ans % M + M) % M;
    }
    ans = (ans % M + M) % M;
    //if(ans == 0) ans = M;//视情况而定，等于0的时候是因为给定的模数均为1，此时答案应该取任意值均可，而不是只有解 0 ，有时需
    return ans;
}

int main()
{
    scanf("%lld", &n);
    //bi -> m[i], ai -> a[i]
    for(int i = 1; i <= n; ++ i)
        scanf("%lld%lld", &bi[i], &ai[i]);
    printf("%lld\n", excrt());
    return 0;
}
```

多项式

fft

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define IOS ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
const double PI = acos(-1.0);
const int maxn = 3e6+10;
int n,m,l;
int r[maxn];
struct Complex
{
    double x,y;
    Complex operator+ (const Complex&t) const {return {x+t.x,y+t.y};}
    Complex operator- (const Complex&t) const {return {x-t.x,y-t.y};}
    Complex operator* (const Complex&t) const {return {x*t.x-
y*t.y,x*t.y+y*t.x};}
}a[maxn],b[maxn];
int rev[maxn],bit,tot;
void fft(Complex a[],int inv)
{
    for(int i=0;i<tot;i++){
        if(i<rev[i]) swap(a[i],a[rev[i]]);
    }
    for(int mid=1;mid<tot;mid<<=1){
        auto w1=Complex({cos(PI/mid),inv*sin(PI/mid)});
        for(int i=0;i<tot;i+=mid*2){
            auto wk=Complex({1,0});
            for(int j=0;j<mid;j++,wk=wk*w1){
                auto x=a[i+j],y=wk*a[i+j+mid];
                a[i+j]=x+y,a[i+j+mid]=x-y;
            }
        }
    }
}
int main(){
IOS
cin>>n>>m;
for(int i=0;i<=n;i++) cin>>a[i].x;
for(int i=0;i<=m;i++) cin>>b[i].x;
while((1<<bit)<n+m+1) bit++;
tot=1<<bit;
for(int i=0;i<tot;i++){
    rev[i]=(rev[i>>1]>>1)|((i&1)<<(bit-1));
}
fft(a,1); fft(b,1);
for(int i=0;i<tot;i++) a[i]=a[i]*b[i];
fft(a,-1);
for(int i=0;i<=n+m;i++){
    cout<<(int)(a[i].x/tot+0.5)<<" ";
}
    return 0;
}
```

ntt

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
```

```
#define IOS ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
int n,m;
const int p=998244353,g=3,gi=332748118;
const int maxn = 3e6+50;
ll a[maxn],b[maxn],bit,tot;
int rev[maxn];
ll qpow(ll a, ll b, ll q)
{
    ll res = 1;//因为是用乘法模拟乘方，所以res要是1
    while(b) {
        if(b & 1)res = (res * a) % q;
        a = (a * a) % q;//视情况将 * 换成Mul(龟速乘)
        b >>= 1;
    }
    return res % q;
}
void ntt(ll a[],int inv)
{
    for(int i=0;i<tot;i++){
        if(i<rev[i]) swap(a[i],a[rev[i]]);
    }
    for(int mid=1;mid<tot;mid<<=1){
        ll w1=qpow(inv==1?g:gi,(p-1)/(mid<<1),p);
        for(int j=0;j<tot;j+=(mid<<1)){
            ll w=1;
            for(int k=0;k<mid;k++,w=(w*w1)%p){
                int x=a[j+k],y=w*a[j+k+mid]%p;
                a[j+k]=(x+y)%p; a[j+k+mid]=(x-y+p)%p;
            }
        }
    }
}
int main(){
cin>>n>>m;
for(int i=0;i<=n;i++){
    cin>>a[i]; a[i]=(a[i]+p)%p;
}
for(int i=0;i<=m;i++){
    cin>>b[i]; b[i]=(b[i]+p)%p;
}
while((1<<bit)<n+m+1) bit++;
tot=1<<bit;
for(int i=0;i<tot;i++){
    rev[i]=(rev[i>>1]>>1)|((i&1)<<(bit-1));
}
ntt(a,1); ntt(b,1);
for(int i=0;i<tot;i++) a[i]=a[i]*b[i]%p;
ntt(a,-1);
ll inv=qpow(tot,p-2,p);
for(int i=0;i<=n+m;i++){
    cout<<(a[i]*inv)%p<<" ";
}
cout<<'\n';
    return 0;
}
```

fwt

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

using namespace std;
typedef long long ll;
typedef int itn;
const int N = (1 << 18) + 7, mod = 998244353;
const ll INF = 4e18;

int n, m;
ll A[N], B[N], a[N], b[N];

ll qpow(ll a, ll b)
{
    ll res = 1;
    while(b) {
        if(b & 1) res = 1ll * res * a % mod;
        a = 1ll * a * a % mod;
        b >>= 1;
    }
    return res;
}

ll inv2 = qpow(2, mod - 2);

inline void in()
{
    for(int i = 0; i < n; ++ i)
        a[i] = A[i], b[i] = B[i];
}

inline void get()
{
    for(int i = 0; i < n; ++ i)
        a[i] = a[i] * b[i] % mod;
}

inline void out()
{
    for(int i = 0; i < n; ++ i)
        printf("%lld%s", (a[i] % mod + mod) % mod, i == (n - 1) ? "\n" : " ");
}


inline void OR(ll *f, int x = 1)//前半部分 f[i + j], 后半部分 f[i + j + k]
{
    for(int o = 2; o <= n; o <<= 1)
        for(int i = 0, k = o >> 1; i < n; i += o)
            for(int j = 0; j < k; ++ j)
                f[i + j + k] = (f[i + j] * x + f[i + j + k] + (x == 1 ? 0 :
mod)) % mod;

}
```

```
inline void AND(ll *f, int x = 1)//前半部分 f[i + j],后半部分 f[i + j + k]
{
    for(int o = 2; o <= n; o <<= 1)
        for(int i = 0, k = o >> 1; i < n; i += o)
            for(int j = 0; j < k; ++ j)
                f[i + j] = (f[i + j] + f[i + j + k] * x + (x == 1 ? 0 : mod)) %
mod;

}

inline void XOR(ll *f, int x = 1)//前半部分 f[i + j],后半部分 f[i + j + k]
{
    for(int o = 2; o <= n; o <<= 1)
        for(int i = 0, k = o >> 1; i < n; i += o)
            for(int j = 0; j < k; ++ j) {
                int X = f[i + j], Y = f[i + j + k];
                f[i + j] = (X + Y) % mod;
                f[i + j + k] = (X - Y % mod + mod) % mod;
                if(x != 1) {
                    f[i + j] = f[i + j] * inv2 % mod;
                    f[i + j + k] = f[i + j + k] * inv2 % mod;
                }
            }
}

int main()
{
    scanf("%d", &m);
    n = 1 << m;
    for(int i = 0; i < n; ++ i)
        scanf("%lld", &A[i]);
    for(int i = 0; i < n; ++ i)
        scanf("%lld", &B[i]);
    in(), OR(a), OR(b), get(), OR(a, -1), out();
    in(), AND(a), AND(b), get(), AND(a, -1), out();
    in(), XOR(a), XOR(b), get(), XOR(a, -1), out();
    return 0;
}
```

mtt

拆系数FFT

```
//luoguP4245 【模板】MTT
#include<cstdio>
#include<cmath>
#include<algorithm>
const int N = 262144 + 10, M = 32767;
const double pi = acos(-1.0);
typedef long long LL;
int read() {
    char ch = getchar(); int f = 1, x = 0;
    for(;ch < '0' || ch > '9'; ch = getchar()) if(ch == '-') f = -1;
    for(;ch >= '0' && ch <= '9'; ch = getchar()) x = (x << 1) + (x << 3) - '0' +
ch;
    return x * f;
```

```
}
struct cp {
    double r, i;
    cp(double _r = 0, double _i = 0) : r(_r), i(_i) {}
    cp operator * (const cp &a) {return cp(r * a.r - i * a.i, r * a.i + i *
a.r);}
    cp operator + (const cp &a) {return cp(r + a.r, i + a.i);}
    cp operator - (const cp &a) {return cp(r - a.r, i - a.i);}
}w[N], nw[N], da[N], db[N];
cp conj(cp a) {return cp(a.r, -a.i);}
int L, n, m, a[N], b[N], c[N], R[N], P;
void Pre() {
    int x = 0; for(L = 1; (L <<= 1) <= n + m; ++x) ;
    for(int i = 1;i < L; ++i) R[i] = (R[i >> 1] >> 1) | (i & 1) << x;
    for(int i = 0;i < L; ++i) w[i] = cp(cos(2 * pi * i / L), sin(2 * pi * i /
L));
}
void FFT(cp *F) {
    for(int i = 0;i < L; ++i) if(i < R[i]) std::swap(F[i], F[R[i]]);
    for(int i = 2, d = L >> 1;i <= L; i <<= 1, d >>= 1)
        for(int j = 0;j < L; j += i) {
            cp *l = F + j, *r = F + j + (i >> 1), *p = w, tp;
            for(int k = 0;k < (i >> 1); ++k, ++l, ++r, p += d)
                tp = *r * *p, *r = *l - tp, *l = *l + tp;
        }
}
void Mul(int *A, int *B, int *C) {
    for(int i = 0;i < L; ++i) (A[i] += P) %= P, (B[i] += P) %= P;
    static cp a[N], b[N], Da[N], Db[N], Dc[N], Dd[N];
    for(int i = 0;i < L; ++i) a[i] = cp(A[i] & M, A[i] >> 15);
    for(int i = 0;i < L; ++i) b[i] = cp(B[i] & M, B[i] >> 15);
    FFT(a); FFT(b);
    for(int i = 0;i < L; ++i) {
        int j = (L - i) & (L - 1); static cp da, db, dc, dd;
        da = (a[i] + conj(a[j])) * cp(0.5, 0);
        db = (a[i] - conj(a[j])) * cp(0, -0.5);
        dc = (b[i] + conj(b[j])) * cp(0.5, 0);
        dd = (b[i] - conj(b[j])) * cp(0, -0.5);
        Da[j] = da * dc; Db[j] = da * dd; Dc[j] = db * dc; Dd[j] = db * dd; //顺
便区间反转，方便等会直接用DFT代替IDFT
    }
    for(int i = 0;i < L; ++i) a[i] = Da[i] + Db[i] * cp(0, 1);
    for(int i = 0;i < L; ++i) b[i] = Dc[i] + Dd[i] * cp(0, 1);
    FFT(a); FFT(b);
    for(int i = 0;i < L; ++i) {
        int da = (LL) (a[i].r / L + 0.5) % P; //直接取实部和虚部
        int db = (LL) (a[i].i / L + 0.5) % P;
        int dc = (LL) (b[i].r / L + 0.5) % P;
        int dd = (LL) (b[i].i / L + 0.5) % P;
        C[i] = (da + ((LL)(db + dc) << 15) + ((LL)dd << 30)) % P;
    }
}
int main() {
    n = read(); m = read(); P = read();
    for(int i = 0;i <= n; ++i) a[i] = read();
    for(int j = 0;j <= m; ++j) b[j] = read();
    Pre(); Mul(a, b, c);
    for(int i = 0;i <= n + m; ++i) printf("%d ", (c[i] + P) % P); puts("");
```

```
    return 0;
}
```

高斯消元法

线性方程组（浮点型）

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define IOS ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
const int N = 1050;
const double eps = 1e-7;
double a[N][N];//增广矩阵
double x[N];//解集
bool freeX[N];//标记是否为自由变元
int Gauss(int equ,int var){//返回自由变元个数
    /*初始化*/
    for(int i=0;i<=var;i++){
        x[i]=0;
        freeX[i]=true;
    }
    /*转换为阶梯阵*/
    int col=0;//当前处理的列
    int row;//当前处理的行
    for(row=0;row<equ&&col<var;row++,col++){//枚举当前处理的行
        int maxRow=row;//当前列绝对值最大的行
        for(int i=row+1;i<equ;i++){//寻找当前列绝对值最大的行
            if(abs(a[i][col])>abs(a[maxRow][col]))
                maxRow=i;
        }
        if(maxRow!=row){//与第row行交换
            for(int j=row;j<var+1;j++)
                swap(a[row][j],a[maxRow][j]);
        }
        if(fabs(a[row][col])<1e-6){//col列第row行以下全是0，处理当前行的下一列
            row--;
            continue;
        }
        for(int i=row+1;i<equ;i++){//枚举要删去的行
            if(fabs(a[i][col])>1e-6){
                double temp=a[i][col]/a[row][col];
                for(int j=col;j<var+1;j++)
                    a[i][j]-=a[row][j]*temp;
                a[i][col]=0;
            }
        }
    }
    /*求解*/
    //无解
    for(int i=row;i<equ;i++)
        if(fabs(a[i][col])>1e-6)
            return -1;
    //无穷解: 在var*(var+1)的增广阵中出现(0,0,...,0)这样的行
    int temp=var-row;//自由变元有var-row个
    if(row<var)//返回自由变元数
```

```
        return temp;
    //唯一解: 在var*(var+1)的增广阵中形成严格的上三角阵
    for(int i=var-1;i>=0;i--){//计算解集
        double temp=a[i][var];
        for(int j=i+1;j<var;j++)
            temp-=a[i][j]*x[j];
        x[i]=temp/a[i][i];
    }
    return 0;
}
int main(){
    int n; cin>>n;
    for(int i=0;i<n;i++){
        for(int j=0;j<n+1;j++){
            cin>>a[i][j];
        }
    }
    //存入增广矩阵 equ为方程数 var为变元 都为n
    int ans= Gauss(n,n);
    if(ans!=0){
        cout<<"No Solution"<<'\n'; return 0;
    }
    // cout<<ans<<'\n';
    for(int i=0;i<n;i++){
        cout<<fixed<<setprecision(2)<<x[i]<<'\n';
    }
    return 0;
}
```

解异或方程组

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define IOS ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
const int N = 2050;
int n,m;
int a[N][N];//增广矩阵
int x[N];//解集
int freeX[N];//自由变元
int Gauss(int equ,int var){//返回自由变元个数
    /*初始化*/
    for(int i=0;i<=var;i++){
        x[i]=0;
        freeX[i]=0;
    }
    /*转换为阶梯阵*/
    int col=0;//当前处理的列
    int num=0;//自由变元的序号
    int row;//当前处理的行
    for(row=0;row<equ&&col<var;row++,col++){//枚举当前处理的行
        int maxRow=row;//当前列绝对值最大的行
        for(int i=row+1;i<equ;i++){//寻找当前列绝对值最大的行
            if(abs(a[i][col])>abs(a[maxRow][col]))
                maxRow=i;
        }
        if(maxRow!=row){//与第row行交换
```

```
            for(int j=row;j<var+1;j++)
                swap(a[row][j],a[maxRow][j]);
        }
        if(a[row][col]==0){//col列第row行以下全是0，处理当前行的下一列
            freeX[num++]=col;//记录自由变元
            row--;
            continue;
        }

        for(int i=row+1;i<equ;i++){
            if(a[i][col]!=0){
                for(int j=col;j<var+1;j++){//对于下面出现该列中有1的行，需要把1消掉
                    a[i][j]^=a[row][j];
                }
            }
        }
    }

    /*求解*/
    //无解：化简的增广阵中存在(0,0,...,a)这样的行，且a!=0
    for(int i=row;i<equ;i++)
        if(a[i][col]!=0)
            return -1;

    //无穷解: 在var*(var+1)的增广阵中出现(0,0,...,0)这样的行
    int temp=var-row;//自由变元有var-row个
    if(row<var)//返回自由变元数
        return temp;

    //唯一解: 在var*(var+1)的增广阵中形成严格的上三角阵
    for(int i=var-1;i>=0;i--){//计算解集
        x[i]=a[i][var];
        for(int j=i+1;j<var;j++)
            x[i]^=(a[i][j]&&x[j]);
    }
    return 0;
}
// int enumFreeX(int freeNum,int var){//枚举自由元，统计有解情况下1最少的个数
//     int sta=(1<<(freeNum));//自由元的状态总数
//     int res=INF;
//     for(int i=0;i<sta;++i){//枚举状态
//         int cnt=0;
//         for(int j=0;j<freeNum;j++){//枚举自由元
//             if(i&(1<<j)){
//                 cnt++;
//                 x[freeX[j]]=1;
//             }else
//                 x[freeX[j]]=0;
//         }
//         for(int k=var-freeNum-1;k>=0;k--){//没有自由元的最下面一行
//             int index=0;
//             for(index=k;k<var;index++){//在当前行找到第一个非0自由元
//                 if(a[k][index])
//                     break;
//             }
//             x[index]=a[k][var];
//             for(int j=index+1;j<var;++j){//向后依次计算出结果
//                 if(a[k][j])
```

```
//                      x[index]^=x[j];
//              }
//              cnt+=x[index];//若结果为1，则进行统计
//          }
//          res=min(res,cnt);
//      }
//      return res;
// }
int main(){
    IOS
    cin>>n>>m;
    for(int i=0;i<m;i++){
        string s; int x; cin>>s>>x;
        for(int j=0;j<n;j++){
            a[i][j]=s[j]-'0';
        }
        a[i][n]=x;
    }
    bool flag=0;
    for(int i=1;i<=m;i++){
        int res = Gauss(i, n);
        if(res==0){
            cout<<i<<'\n';
            for(int j=0;j<n;j++){
                if(x[j]&1) cout<<"?y7M#"<<'\n';
                else cout<<"Earth"<<'\n';
            }
            flag=1; break;
        }
    }
    if(!flag){
        cout<<"Cannot Determine"<<'\n';
    }
    return 0;
}
```