数学知识-质数

一、AcWing 866. 试除法判定质数

【题目描述】

给定n个正整数a_i，判定每个数是否是质数。

【输入格式】

第一行包含整数n。

接下来n行，每行包含一个正整数a_i。

【输出格式】

共n行，其中第i行输出第i个正整数a_i是否为质数，是则输出`Yes`，否则输出`No`。

【数据范围】

$1 \leq n \leq 100$

$1 \leq a_i \leq 2^{31} - 1$

【输入样例】

```
2
2
6
```

【输出样例】

```
Yes
No
```

【分析】

质数的定义：在大于1的整数中，如果只包含1和它本身这两个约数，就被称为质数，或者称为素数。

根据定义，可以很容易的写出试除法判断质数的代码，时间复杂度为$O(n)$。

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

观察以上代码，有没有可以优化的地方呢？肯定是有的。首先要知道一个性质：若$d|n$（d能整除n），则$\frac{n}{d}|n$。例如$3|12$，则$4|12$。因此我们只需要枚举到$\sqrt{n}$即可，这样时间复杂度就优化到了$O(\sqrt{n})$。

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

注意：枚举时的判断条件最好不要写成`i <= sqrt(x)`，因为每次都计算一次$\sqrt{x}$效率会变低，也最好不要写成`i * i <= n`，因为存在`i * i`越界的情况。

【代码】

```
#include <iostream>
using namespace std;

bool is_prime(int x)
{
    if (x < 2) return false;
    for (int i = 2; i <= x / i; i++)
        if (x % i == 0) return false;
    return true;
}

int main()
{
    int n;
    cin >> n;
    while (n--)
    {
```

```
        int x;
        cin >> x;
        if (is_prime(x)) puts("Yes");
        else puts("No");
    }
    return 0;
}
```

二、AcWing 867. 分解质因数

【题目描述】

给定n个正整数a_i，将每个数分解质因数，并按照质因数从小到大的顺序输出每个质因数的底数和指数。

【输入格式】

第一行包含整数n。

接下来n行，每行包含一个正整数a_i。

【输出格式】

对于每个正整数a_i，按照从小到大的顺序输出其分解质因数后，每个质因数的底数和指数，每个底数和指数占一行。

每个正整数的质因数全部输出完毕后，输出一个空行。

【数据范围】

$1 \leq n \leq 100$

$1 \leq a_i \leq 2 \times 10^9$

【输入样例】

```
2
6
8
```

【输出样例】

```
2 1
3 1

2 3

```

【分析】

分解质因数的方法：从小到大枚举x的所有因数i（$i \geq 2$），如果$x \% i == 0$，那么将x中的i除干净，并且记录i的次数，时间复杂度为$(O(n))$。

```
void divide(int x)
{
    for (int i = 2; i <= x; i++)
        if (x % i == 0)
        {
            int s = 0;//质因数的个数
            while (x % i == 0)
            {
                x /= i;
                s++;
            }
            cout << i << ' ' << s;
        }
}
```

同样的，考虑一下以上代码是否有优化的空间呢？不难证明，n中最多只可能包含一个大于$\sqrt{n}$的质因数，因此只需要枚举到$\sqrt{n}$即可，若最后$x > 1$，说明此时的x即为那个大于$\sqrt{n}$的质因数。优化后的时间复杂度为$(O(\sqrt{n}))$。

```
void divide(int x)
{
    for (int i = 2; i <= x / i; i++)
        if (x % i == 0)
        {
            int s = 0;//质因数的个数
            while (x % i == 0)
            {
                x /= i;
                s++;
            }
            cout << i << ' ' << s << endl;
```

```
        }
    if (x > 1) cout << x << ' ' << 1 << endl;
}
```

【代码】

```
#include <iostream>
using namespace std;

void divide(int x)
{
    for (int i = 2; i <= x / i; i++)
        if (x % i == 0)
        {
            int s = 0;//质因数的个数
            while (x % i == 0)
            {
                x /= i;
                s++;
            }
            cout << i << ' ' << s << endl;
        }
    if (x > 1) cout << x << ' ' << 1 << endl;
    cout << endl;
}

int main()
{
    int n;
    cin >> n;
    while (n--)
    {
        int x;
        cin >> x;
        divide(x);
    }
    return 0;
}
```

三、AcWing 868. 筛质数

【题目描述】

给定一个正整数n，请你求出$1 \sim n$中质数的个数。

【输入格式】

共一行，包含整数n。

【输出格式】

共一行，包含一个整数，表示$1 \sim n$中质数的个数。

【数据范围】

$1 \leq n \leq 10^6$

【输入样例】

```
8
```

【输出样例】

```
4
```

【分析】

（1）朴素筛法$O(nlogn)$

```
void get_primes()
{
    for (int i = 2; i <= n; i++)
    {
        if (!st[i]) prime[cnt++] = i;//如果没被筛过，说明是质数
        for (int j = i + i; j <= n; j += i)//筛掉所有i的倍数
            st[j] = true;
    }
}
```

（2）埃氏筛法$O(nloglogn)$

```
void get_primes()
{
    for (int i = 2; i <= n; i++)
    {
        if (st[i]) continue;
        prime[cnt++] = i;
        for (int j = i + i; j <= n; j += i)//只筛所有质数的倍数
            st[j] = true;
    }
}
```

（3）线性筛法$O(n)$

```
void get_primes()
{
    for (int i = 2; i <= n; i++)
    {
        if (!st[i]) prime[cnt++] = i;
        for (int j = 0; prime[j] <= n / i; j++)
        {
            st[i * prime[j]] = true;
            if (i % prime[j] == 0) break;//prime[j]一定是i的最小质因子
        }
    }
}
```

【代码】

```
#include <iostream>
using namespace std;

const int N = 1000010;
int prime[N];
int n, cnt;
bool st[N];

void get_primes()
{
    for (int i = 2; i <= n; i++)
    {
        if (!st[i]) prime[cnt++] = i;
        for (int j = 0; prime[j] <= n / i; j++)
```

```
        {
            st[i * prime[j]] = true;
            if (i % prime[j] == 0) break;//prime[j]一定是i的最小质因子
        }
    }
}

int main()
{
    cin >> n;
    get_primes();
    cout << cnt << endl;
    return 0;
}
```

数学知识-约数

一、AcWing 869. 试除法求约数

【题目描述】

给定n个正整数a_i，对于每个整数a_i，请你按照从小到大的顺序输出它的所有约数。

【输入格式】

第一行包含整数n。

接下来n行，每行包含一个整数a_i。

【输出格式】

输出共n行，其中第i行输出第i个整数a_i的所有约数。

【数据范围】

$1 \leq n \leq 100$

$2 \leq a_i \leq 2 \times 10^9$

【输入样例】

```
2
6
8
```

【输出样例】

```
1 2 3 6
1 2 4 8
```

【分析】

试除法求约数的原理十分简单，因此直接看代码即可，算法时间复杂度为$O(\sqrt{n})$。

【代码】

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

vector<int> get_divisors(int x)
{
    vector<int> res;
    for (int i = 1; i <= x / i; i++)
        if (x % i == 0)
        {
            res.push_back(i);
            if (i != x / i) res.push_back(x / i);
        }
    sort(res.begin(), res.end());
    return res;
}

int main()
{
    int n;
    cin >> n;
    while (n--)
    {
        int a;
        cin >> a;
        auto res = get_divisors(a);
        for (auto x : res) cout << x << ' ';
        cout << endl;
    }
    return 0;
```

```
}
```

二、AcWing 870/871. 约数个数/约数之和

【题目描述】

给定n个正整数a_i，请你输出这些数的乘积的约数个数（之和），答案对10^9+7取模。

【输入格式】

第一行包含整数n。

接下来n行，每行包含一个整数a_i。

【输出格式】

输出一个整数，表示所给正整数的乘积的约数个数（之和），答案需对10^9+7取模。

【数据范围】

$1 \leq n \leq 100$

$1 \leq a_i \leq 2 \times 10^9$

【输入样例】

```
3
2
6
8
```

【输出样例】

```
12(252)
```

【分析】

基本公式：

如果$N = p_1^{\alpha_1} * p_2^{\alpha_2} * \ldots * p_k^{\alpha_k}$（分解质因数）

约数个数：$(\alpha_1 + 1) * (\alpha_2 + 1) * \ldots * (\alpha_k + 1)$

约数之和：$(p_1^0 + p_1^1 + \cdots + p_1^{\alpha_1}) * \ldots * (p_k^0 + p_k^1 + \cdots + p_k^{\alpha_k})$

求$p^0 + p^1 + \cdots + p^\alpha$的方式：

首先令$s=1$

$\Rightarrow s=(s*p+1)=p+1$

$\Rightarrow s=(s*p+1)=p^2+p+1$

$\Rightarrow s=(s*p+1)=p^3+p^2+p+1$

$\Rightarrow \dots$

$\Rightarrow s=(s*p+1)=p^\alpha+p^{\alpha-1}+\cdots+p+1$

即$while(\alpha--)\ s=(s*p+1)$。

证明：

因为任何一个约数d可以表示成$d=p_1^{\beta_1}\ *\ p_2^{\beta_2}\ *\ \dots\ *\ p_k^{\beta_k}(0\le\beta_i\le\alpha_i)$。

每一项的β_i如果不同，那么约数d就不相同（算数基本定理，每个数的因式分解是唯一的）所以n的约数就跟β_i的选法是一一对应的。

β_1一共有$0\sim\alpha_1$种选法

β_2一共有$0\sim\alpha_2$种选法

$\dots$

β_k一共有$0\sim\alpha_k$种选法

根据乘法原理，一共有$(\alpha_1+1)\ *\ (\alpha_2+1)\ *\ \dots\ *\ (\alpha_k+1)$个约数。

【约数个数代码】

```cpp
#include <iostream>
#include <unordered_map>
using namespace std;

typedef long long LL;
const int MOD = 1e9 + 7;
unordered_map<int, int> cnt;
int n;

int main()
{
    cin >> n;

    while (n--)
```

```
    {
        int x;
        cin >> x;
        for (int i = 2; i <= x / i; i++)
            while (x % i == 0)
            {
                x /= i;
                cnt[i]++;
            }
        if (x > 1) cnt[x]++;
    }
    LL res = 1;
    for (auto t : cnt) res = res * (t.second + 1) % MOD;
    cout << res << endl;
    return 0;
}
```

【约数之和代码】

```
#include <iostream>
#include <unordered_map>
using namespace std;

typedef long long LL;
const int MOD = 1e9 + 7;
unordered_map<int, int> cnt;
int n;

int main()
{
    cin >> n;
    while (n--)
    {
        int x;
        cin >> x;
        for (int i = 2; i <= x / i; i++)
            while (x % i == 0)
            {
                x /= i;
                cnt[i]++;
            }
        if (x > 1) cnt[x]++;

    }
```

```
    LL res = 1;
    for (auto t : cnt)
    {
        int p = t.first, a = t.second;
        LL s = 1;
        while (a--) s = (s * p + 1) % MOD;//计算p^0 + p^1 + ... + p^a
        res = res * s % MOD;
    }
    cout << res << endl;
    return 0;
}
```

三、AcWing 872. 最大公约数

【题目描述】

给定n对正整数a_i, b_i，请你求出每对数的最大公约数。

【输入格式】

第一行包含整数n。

接下来n行，每行包含一个整数对a_i, b_i。

【输出格式】

输出共n行，每行输出一个整数对的最大公约数。

【数据范围】

$1 \leq n \leq 10^5$

$1 \leq a_i, b_i \leq 2 \times 10^9$

【输入样例】

```
2
3 6
4 6
```

【输出样例】

```
3
2
```

【代码】

```cpp
#include <iostream>
using namespace std;

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

int main()
{
    int n;
    cin >> n;
    while (n--)
    {
        int a, b;
        cin >> a >> b;
        cout << gcd(a, b) << endl;
    }
    return 0;
}
```

数学知识-欧拉函数

欧拉函数是什么？

定义：对于正整数n，欧拉函数是小于或等于n的正整数中与n互质的数的个数，记作$\varphi(n)$。$\varphi(1)=1$。

如何求N的欧拉值？

首先，欧拉函数是一个积性函数，当m,n互质时，$\varphi(mn)=\varphi(m)*\varphi(n)$;

根据唯一分解定理知：$n=p_1^{a_1}*p_2^{a_2}*\cdots*p_x^{a_x}$;

因此$\varphi(n)=\varphi(p_1^{a_1})*\cdots*\varphi(p_x^{a_x})$。

对于任意一项$p_s^{a_s}$，从定义出发$\varphi(p_s^{a_s})$为小于或等于$p_s^{a_s}$的正整数中与$p_s^{a_s}$互质的数的个数。

从1到$p_s^{a_s}$中共有$p_s^{a_s}$个数字

其中与$p_s^{a_s}$不互质的有$p_s,2p_s,3p_s,\ldots,p_s^{a_s-1}*p_s$共$p_s^{a_s-1}$项

所以$\varphi(p_s^{a_s})=p_s^{a_s}-p_s^{a_s-1}=p_s^{a_s}(1-\frac{1}{p_s})$

因此

$\varphi(n) = \varphi(p_1^{a_1}) * \cdots * \varphi(p_x^{a_x})$

$= (p_1^{a_1} - p_1^{a_1-1}) * \cdots * (p_x^{a_x} - p_x^{a_x-1})$

$= p_1^{a_1} * (1 - \frac{1}{p_1}) * p_2^{a_2} * (1 - \frac{1}{p_2}) * \cdots * p_x^{a_x} * (1 - \frac{1}{p_x})$

$= p_1^{a_1} * p_2^{a_2} * \cdots * p_x^{a_x} * (1 - \frac{1}{p_1}) * (1 - \frac{1}{p_2}) * \cdots * (1 - \frac{1}{p_x})$

$= n * \prod_{i=1}^{x}(1 - \frac{1}{p_i})$

一、AcWing 873. 欧拉函数

【题目描述】

给定n个正整数a_i，请你求出每个数的欧拉函数。

【输入格式】

第一行包含整数n。

接下来n行，每行包含一个正整数a_i。

【输出格式】

输出共n行，每行输出一个正整数a_i的欧拉函数。

【数据范围】

$1 \leq n \leq 100$

$1 \leq a_i \leq 2 \times 10^9$

【输入样例】

```
3
3
6
8
```

【输出样例】

```
2
2
4
```

【代码】

```cpp
#include <iostream>
using namespace std;

int main()
{
    int n;
    cin >> n;
    while (n--)
    {
        int a;
        cin >> a;
        int res = a;//先令res = N
        for (int i = 2; i <= a / i; i++)//因式分解
            if (a % i == 0)
            {
                res = res / i * (i - 1);//由于1 / i为小数，因此将公式等价变形
                while (a % i == 0) a /= i;
            }
        if (a > 1) res = res / a * (a - 1);
        cout << res << endl;
    }
    return 0;
}
```

二、AcWing 874. 筛法求欧拉函数

【题目描述】

给定一个正整数n，求$1 \sim n$中每个数的欧拉函数之和。

【输入格式】

共一行，包含一个整数n。

【输出格式】

共一行，包含一个整数，表示$1 \sim n$中每个数的欧拉函数之和。

【数据范围】

$1 \leq n \leq 10^6$

【输入样例】

```
6
```

【输出样例】

```
12
```

【分析】

质数i的欧拉函数即为`phi[i] = i - 1`，因为$1 \sim i-1$均与i互质，共$i-1$个。

`phi[prime[j] * i]`分为两种情况：

1. `i % prime[j] == 0`时：$prime[j]$是i的最小质因子，也是$prime[j]*i$的最小质因子，因此$1-\frac{1}{prime[j]}$这一项在$phi[i]$中计算过了，只需将基数N修正为$prime[j]$倍，最终结果为`phi[i] * prime[j]`。
2. `i % prime[j] != 0`时：$prime[j]$不是i的质因子，只是$prime[j]*i$的最小质因子，因此不仅需要将基数N修正为$prime[j]$倍，还需要补上$1-\frac{1}{prime[j]}$这一项，因此最终结果`phi[i] * (prime[j] - 1)`。

【代码】

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>
using namespace std;

typedef long long LL;
const int N = 1000010;
int prime[N];
bool st[N];
int phi[N];//欧拉函数φ
int n, cnt;

LL get_eulers(int n)
{
    phi[1] = 1;//定义φ(1)=1
    for (int i = 2; i <= n; i++)
    {
        if (!st[i])
        {

            prime[cnt++] = i;
```

```
            phi[i] = i - 1;//如果i是质数，那么1~i-1都与i互质，因此ϕ(i)=i-1
        }
        for (int j = 0; prime[j] <= n / i; j++)
        {
            st[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                //若prime[j]是i的质因数，则i*prime[j]的质因数种类与i的质因数一样，答案要乘上prime[j]
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            //若prime[j]不是i的质因数，则i*prime[j]将会多出一个质因数prime[j]，答案要乘上prime[j]*(1-1/prime[j])
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
        }
    }
    LL res = 0;
    for (int i = 1; i <= n; i++) res += phi[i];
    return res;
}

int main()
{
    cin >> n;
    LL res = get_eulers(n);
    cout << res << endl;
    return 0;
}
```

数学知识-快速幂

一、什么是快速幂？

快速幂能够快速求出$a^b \% p$的问题，时间复杂度为$O(logb)$，对于n组数据，那么时间复杂度为$O(nlogb)$。

先来看一下朴素求解$a^b \% p$的代码，基本思路是对于n组数据，分别循环b次求出$a^b \% p$，时间复杂度为$O(nb)$，当n和b较大时会TLE！！！

【朴素代码】

```
while (n--)
{
    int a, b, p;
    long long res = 1;
    cin >> a >> b >> p;
    while (b--) res = res * a % p;
    cout << res << endl;
}
```

那么再来看一下快速幂的基本思路：

首先预处理出$a^{2^0}, a^{2^1}, a^{2^2}, \dots, a^{2^{logb}}$，这$logb+1$个数；

将a^b用$a^{2^0}, a^{2^1}, a^{2^2}, \dots, a^{2^{logb}}$这$logb+1$种数来组合，即组合成$a^b = a^{2^{x_1}} * a^{2^{x_2}} * \cdots * a^{2^{x_t}} = a^{2^{x_1}+2^{x_2}+\cdots+2^{x_t}}$，即用二进制来表示；

为什么b可用$a^{2^0}, a^{2^1}, a^{2^2}, \dots, a^{2^{logb}}$这$logb+1$个数来表示？是因为二进制可以表示所有数，且用单一二进制表示时，b单一表示最大可表示为二进制形式的2^{logb}。

【快速幂代码】

```
typedef long long LL;

//返回a ^ b % p的结果
LL quickMi(LL a, LL b, LL p)
{
    LL res = 1;
    while (b)
    {
        if (b & 1) res = res * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return res % p;//此处%p是处理b = 0, p = 1的情况
}
```

二、AcWing 876. 快速幂求逆元

【题目描述】

给定n组a_i, p_i，其中p_i是质数，求a_i模p_i的乘法逆元，若逆元不存在则输出`impossible`。

注意：请返回在$0 \sim p-1$之间的逆元。

【输入格式】

第一行包含整数n。

接下来n行，每行包含一个数组a_i, p_i，数据保证p_i是质数。

【输出格式】

输出共n行，每组数据输出一个结果，每个结果占一行。

若a_i模p_i的乘法逆元存在，则输出一个整数，表示逆元，否则输出`impossible`。

【数据范围】

$1 \leq n \leq 10^5$

$1 \leq a_i, p_i \leq 2 \times 10^9$

【输入样例】

```
3
4 3
8 5
6 3
```

【输出样例】

```
1
2
impossible
```

【分析】

乘法逆元的定义：若整数b, m互质，并且对于任意的整数a，如果满足$b|a$，则存在一个整数x，使得$a/b \equiv a * x (mod\ m)$，则称$x$为$b$的模$m$乘法逆元，记为$b^{-1} (mod\ m)$。

当p为质数时，才可以用快速幂求逆元！

a有逆元的充要条件是a与p互质

$a/b \equiv a * x (mod\ p)$

两边同乘b可得$a \equiv a * b * x (mod\ p)$

即$1 \equiv b * x (mod\ p)$

同$b * x \equiv 1 (mod\ p)$

由费马小定理可知：当p为质数时，$b^{(p-1)} \equiv 1 (mod\ p)$

拆一个b出来可得$b * b^{(p-2)} \equiv 1(mod\ p)$

故当n为质数时，b的乘法逆元$x = b^{(p-2)}$

【代码】

```
#include <iostream>
using namespace std;

typedef long long LL;
int n;

LL quickMi(LL a, LL b, LL p)
{
    LL res = 1;
    while (b)
    {
        if (b & 1) res = res * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return res % p;
}

int main()
{
    cin >> n;
    while (n--)
    {
        LL a, p;
        cin >> a >> p;
        //a有逆元的充要条件是a与p互质，由于p为质数，因此a不能是p的倍数
        if (a % p == 0) puts("impossible");
        else cout << quickMi(a, p - 2, p) << endl;
    }
    return 0;
}
```

数学知识-扩展欧几里得算法

一、AcWing 877. 扩展欧几里得算法

【题目描述】

给定n对正整数a_i,b_i，对于每对数，求出一组x_i,y_i，使其满足$a_i \times x_i + b_i \times y_i = gcd(a_i,b_i)$。

【输入格式】

第一行包含整数n。

接下来n行，每行包含两个整数a_i,b_i。

【输出格式】

输出共n行，对于每组a_i,b_i，求出一组满足条件的x_i,y_i，每组结果占一行。

本题答案不唯一，输出任意满足条件的x_i,y_i均可。

【数据范围】

$1 \leq n \leq 10^5$

$1 \leq a_i,b_i \leq 2 \times 10^9$

【输入样例】

```
2
4 6
8 18
```

【输出样例】

```
-1 1
-2 1
```

【分析】

扩展欧几里得算法用于求解方程$ax+by=gcd(a,b)$的解。

- 当$b=0$时，$ax+by=a$，故而$x=1,y=0$。
- 当$b \neq 0$时，$gcd(a,b)=gcd(b,a\%b)$

调换两者的系数，即$by+(a\%b)x=gcd(a,b)$

又因为$a\%b=a-\lfloor \frac{a}{b} \rfloor \times b$，带入上式得：$by+(a-\lfloor \frac{a}{b} \rfloor \times b)x=gcd(a,b)$

整理得：$ax+b(y-\lfloor \frac{a}{b} \rfloor x)=gcd(a,b)$

因此可递归求出下层的x,y，在回溯的时候x不变，y修改为$y-\lfloor \frac{a}{b} \rfloor x$即可。

【代码】

```cpp
#include <iostream>
using namespace std;

int exgcd(int a, int b, int &x, int &y)
{
    if (!b)//b = 0时, gcd(a, 0) = a, 此时ax + by = a, 则x = 1, y = 0
    {
        x = 1, y = 0;
        return a;
    }
    int res = exgcd(b, a % b, y, x);
    y -= a / b * x;//by + (a % b)x = gcd(a, b)
    return res;
}

int main()
{
    int n;
    cin >> n;
    while (n--)
    {
        int a, b, x, y;
        cin >> a >> b;
        exgcd(a, b, x, y);
        cout << x << ' ' << y << endl;
    }
    return 0;
}
```

二、AcWing 878. 线性同余方程

【题目描述】

给定n组数据a_i, b_i, m_i，对于每组数求出一个x_i，使其满足$a_i \times x_i \equiv b_i (mod\ m_i)$，如果无解则输出`impossible`。

【输入格式】

第一行包含整数n。

接下来n行，每行包含一组数据a_i, b_i, m_i。

【输出格式】

输出共n行，每组数据输出一个整数表示一个满足条件的x_i，如果无解则输出`impossible`。

每组数据结果占一行，结果可能不唯一，输出任意一个满足条件的结果均可。

输出答案必须在int范围之内。

【数据范围】

$1 \le n \le 10^5$

$1 \le a_i, b_i, m_i \le 2 \times 10^9$

【输入样例】

```
2
2 3 6
4 3 5
```

【输出样例】

```
impossible
-3
```

【分析】

首先题意要求$ax \equiv b(mod\ m)$，那么一定存在一个y，使得$ax = my + b$，即$ax - my = b$。

令$y' = -y$，得$ax + my' = b$，因此可用扩展欧几里得算法求出x, y使得$ax + my = gcd(a, m)$，再将x扩大$b/gcd(a, m)$倍即可。若b不是$gcd(a, m)$的倍数，则x会乘上一个小数，而题意要求x必须为整数，因此若$b\%gcd(a, m)! = 0$则无解。

【代码】

```
#include <iostream>
using namespace std;

int exgcd(int a, int b, int &x, int &y)
{
    if (!b)
    {

        x = 1, y = 0;
```

```
        return a;
    }
    int res = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return res;
}

int main()
{
    int n;
    cin >> n;
    while (n--)
    {
        int a, b, m, x, y;
        cin >> a >> b >> m;//ax = my + b
        int g = exgcd(a, m, x, y);//ax + my = gcd(a, m)
        if (b % g) puts("impossible");//x为整数
        else cout << (long long)x * b / g % m << endl;
    }
    return 0;
}
```

数学知识-中国剩余定理

一、AcWing 204. 表达整数的奇怪方式

【题目描述】

给定$2n$个整数$a_1, a_2, \ldots, a_n$和$m_1, m_2, \ldots, m_n$，求一个最小的非负整数x，满足$\forall i \in [1, n], x \equiv m_i (mod\ a_i)$。

【输入格式】

第1行包含整数n。

第$2 \sim n+1$行：每$i+1$行包含两个整数a_i和m_i，数之间用空格隔开。

【输出格式】

输出最小非负整数x，如果x不存在，则输出-1。

如果存在x，则数据保证x一定在64位整数范围内。

【数据范围】

$1 \leq a_i \leq 2^{31} - 1$

$0 \leq m_i < a_i$

$1 \leq n \leq 25$

【输入样例】

```
2
8 7
11 9
```

【输出样例】

```
31
```

【分析】

（1）式子等价转换

对于：

$x \equiv m_1 (\% \ a_1)$

$x \equiv m_2 (\% \ a_2)$

等价于：

$x = k_1 * a_1 + m_1$

$x = k_2 * a_2 + m_2$

所以：

$k_1 * a_1 + m_1 = k_2 * a_2 + m_2$

移项得：

$k_1 * a_1 - k_2 * a_2 = m_2 - m_1$

即：

$k_1 * a_1 + k_2 * (-a_2) = m_2 - m_1$ ①

也就是我们需要找到一个最小的k_1, k_2，使得等式成立（因为要求x最小，而a和m都是正数）。

（2）用扩展欧几里得算法找出一组解

可以用扩展欧几里得算法算出一组k_1', k_2'使得

$$k_1' * a_1 + k_2' * (-a_2) = gcd(a_1, -a_2)$$

将$gcd(a_1, -a_2)$记为g，若$(m_2 - m_1)$不能整除g，说明无解。否则只需让k_1', k_2'分别扩大$\frac{m_2-m_1}{g}$倍，则可以找到一个k_1, k_2满足①式，即

$$k_1 = k_1' * \frac{m_2-m_1}{g}, k_2 = k_2' * \frac{m_2-m_1}{g}$$

（3）找到最小正整数解

首先有性质：

$$k_1 = k_1 + k * \frac{a_2}{g}$$ ②

$$k_2 = k_2 + k * \frac{a_1}{g}$$

k为任意整数，这时新的k_1, k_2仍满足①式。

要找一个最小的非负整数解，我们只需要让

$$k_1 = k_1 \% abs(\frac{a_2}{g})$$

$$k_2 = k_2 \% abs(\frac{a_1}{g})$$

即可找到当前最小的k_1, k_2的解，即此时的k为0。

Q：此处为什么要取绝对值呢？

A：因为不知道$\frac{a_2}{g}$的正负性，我们在原基础上要尽量减多个$abs(\frac{a_2}{g})$，使其为正整数且最小。

（4）等效替代

将②式带入原方程可得

$$x = (k_1 + k * \frac{a_2}{g})a_1 + m_1$$

$$= k * \frac{a_1 a_2}{g} + k_1 a_1 + m_1$$

令$a_1 = \frac{a_1 a_2}{g}, m_1 = k_1 a_1 + m_1$可得

$$x = k * a_1 + m_1$$

这个形式与一开始我们分解的形式是不是特别像呢？

没错！假设之后又来了一个a_3, m_3

我们只需要继续找：

$x = k * a_1 + m_1 = k_3 * a_3 + m_3$，那么问题又回到了第一步。

因此我们的做法相当于每次考虑合并两个式子，将这n个式子合并$n-1$次后变为一个式子。最后剩下的式子就满足我们的答案。

【代码】

```cpp
#include <iostream>
using namespace std;

typedef long long LL;

LL exgcd(LL a, LL b, LL &x, LL &y)
{
    if (!b)
    {
        x = 1, y = 0;
        return a;
    }
    LL res = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return res;
}

int main()
{
    int n;
    cin >> n;
    LL a1, m1;
    cin >> a1 >> m1;
    bool flag = true;
    for (int i = 1; i < n; i++)
    {
        LL a2, m2;
        cin >> a2 >> m2;
        LL k1, k2;
        LL g = exgcd(a1, a2, k1, k2);
        if ((m2 - m1) % g) flag = false;//无解
        k1 *= (m2 - m1) / g;//将k1扩大为k1*a1-k2*a2=m2-m1中的k1
        LL t = a2 / g;
```

```
        k1 = (k1 % t + t) % t;//取最小的k1
        m1 = a1 * k1 + m1;//因为更新m1时要用到a1因此先更新m1
        a1 = abs(a1 / g * a2);
    }
    if (flag) cout << (m1 % a1 + a1) % a1 << endl;
    else cout << -1 << endl;
    return 0;
}
```

数学知识-高斯消元

一、AcWing 883. 高斯消元解线性方程组

【题目描述】

输入一个包含n个方程n个未知数的线性方程组。

方程组中的系数为实数。

求解这个方程组。

一个包含m个方程n个未知数的线性方程组示例如下：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases}$$

【输入格式】

第一行包含整数n。

接下来n行，每行包含$n+1$个实数，表示一个方程的n个系数以及等号右侧的常数。

【输出格式】

如果给定线性方程组存在唯一解，则输出共n行，其中第i行输出第i个未知数的解，结果保留两位小数。

如果给定线性方程组存在无数解，则输出`Infinite group solutions`。

如果给定线性方程组无解，则输出`No solution`。

【数据范围】

$1 \leq n \leq 100$

所有输入系数以及常数均保留两位小数，绝对值均不超过**100**。

【输入样例】

```
3
1.00 2.00 -1.00 -6.00
2.00 1.00 -3.00 -9.00
-1.00 -1.00 2.00 7.00
```

【输出样例】

```
1.00
-2.00
3.00
```

【分析】

什么是高斯消元法？（$O(n^3)$）

- 通过初等行（列）变换把增广矩阵化为**阶梯型矩阵**并回代得到方程的解
- 适用于求解包含n个方程，n个未知数的多元线性方程组

什么是初等行（列）变换？

- 把某一行乘一个非零的数（方程的两边同时乘上一个非零数不改变方程的解）
- 交换某两行（交换两个方程的位置）
- 把某行的若干倍加到另一行上去（把一个方程的若干倍加到另一个方程上去）

高斯消元算法步骤：

1. 枚举每一列c
2. 找到当前列绝对值最大的一行
3. 把这一行换到最上面（未确定阶梯型的行，并不是第一行）
4. 将该行的第一个数变成**1**（其余所有的数字依次跟着变化）
5. 将下面所有行的当且列的值变成**0**（其余所有的数字依次跟着变化）

如果每一行都进行了如上的操作，说明方程组有唯一解，否则需要对未操作到的方程进行判断，既然是枚举每一列进行操作的，因此未被操作的行除了最右边一列（即b）其余的所有元素均为**0**，因此只需判断b那一列即可：

- 若出现了某一行的b不为**0**，即出现了$0 = C$（C为非零数）的情况，说明无解
- 若剩余行的b全为**0**，即剩下的方程均为$0 = 0$，是多余方程，说明有无穷多组解

【代码】

```
#include <iostream>
#include <algorithm>
#include <cmath>
using namespace std;

const int N = 110;
const double EPS = 1e-6;
double a[N][N];
int n;

int gauss()//返回0表示有唯一解，1表示有无穷多组解，2表示无解
{
    int r, c;//分别表示行、列
    for (r = 0, c = 0; c < n; c++)
    {
        //第一步：找出第c列中绝对值最大的那一行
        int t = r;
        for (int i = r + 1; i < n; i++)
            if (fabs(a[i][c]) > fabs(a[t][c]))
                t = i;
        if (fabs(a[t][c]) < EPS) continue;//如果最大的数为0说明所有方程的这一列都为0

        //第二步：将这一行换至第一行(未确定方程的第一行)，即第r行
        for (int i = c; i <= n; i++) swap(a[t][i], a[r][i]);

        //第三步：将这一行第c列的数变为1(注意从后往前，最后更新a[r][c])
        for (int i = n; i >= c; i--) a[r][i] /= a[r][c];

        //第四步：将这一行之后的每一行的第c列的数变成0
        for (int i = r + 1; i < n; i++)
            if (fabs(a[i][c]) > EPS)
                for (int j = n; j >= c; j--)
                    a[i][j] -= a[i][c] * a[r][j];

        r++;//处理完当前行，进入下一行
    }
    if (r < n)//出现了0 = ?的情况，没有唯一解
    {

        for (int i = r; i < n; i++)
```

```
            if (fabs(a[i][n]) > EPS) return 2;//0 = 非零，无解
        return 1;//否则后面的方程全为0 = 0，无穷多解
    }
    //如果有唯一解，那么从后往前更新每个b
    for (int i = n - 1; i >= 0; i--)
        for (int j = i + 1; j < n; j++)
            a[i][n] -= a[i][j] * a[j][n];
    return 0;
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j <= n; j++)
            cin >> a[i][j];
    int t = gauss();
    if (t == 0) for (int i = 0; i < n; i++) printf("%.2lf\n", a[i][n]);
    else if (t == 1) puts("Infinite group solutions");
    else puts("No solution");
    return 0;
}
```

二、AcWing 884. 高斯消元解异或线性方程组

【题目描述】

输入一个包含n个方程n个未知数的异或线性方程组。

方程组中的系数和常数为0或1，每个未知数的取值也为0或1。

求解这个方程组。

异或线性方程组示例如下：

$$\begin{cases} a_{11}x_1 \wedge a_{12}x_2 \wedge \cdots \wedge a_{1n}x_n = b_1 \\ a_{21}x_1 \wedge a_{22}x_2 \wedge \cdots \wedge a_{2n}x_n = b_2 \\ \cdots \\ a_{m1}x_1 \wedge a_{m2}x_2 \wedge \cdots \wedge a_{mn}x_n = b_m \end{cases}$$

【输入格式】

第一行包含整数n。

接下来n行，每行包含$n+1$个整数0或1，表示一个方程的n个系数以及等号右侧的常数。

【输出格式】

如果给定线性方程组存在唯一解，则输出共n行，其中第i行输出第i个未知数的解。

如果给定线性方程组存在多组解，则输出`Multiple sets of solutions`。

如果给定线性方程组无解，则输出`No solution`。

【数据范围】

$1 \leq n \leq 100$

【输入样例】

```
3
1 1 0 1
0 1 1 0
1 0 0 1
```

【输出样例】

```
1
0
0
```

【分析】

高斯消元求解异或线性方程组的基本思想和求解普通线性方程组基本一样，甚至还更简单：

1. 枚举每一列c
2. 找到当前列为1的任意一行
3. 把这一行换到最上面（未确定阶梯型的行，并不是第一行）
4. 将下面所有行的当且列的值变成0，即和这一行异或即可（其余所有的数字依次跟着变化）

【代码】

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

const int N = 110;
int a[N][N];
```

```cpp
int n;

int gauss()//返回0表示有唯一解，1表示有无穷多组解，2表示无解
{
    int r, c;//分别表示行、列
    for (r = 0, c = 0; c < n; c++)
    {
        //第一步：找出第c列中为1的任意一行
        int t = r;
        for (int i = r; i < n; i++)
            if (a[i][c]) { t = i; break; }
        if (!a[t][c]) continue;//如果所有方程的这一列都为0那么不进行操作

        //第二步：将这一行换至第一行(未确定方程的第一行)，即第r行
        for (int i = c; i <= n; i++) swap(a[t][i], a[r][i]);

        //第三步：将这一行之后的每一行的第c列的数变成0
        for (int i = r + 1; i < n; i++)
            if (a[i][c])
                for (int j = c; j <= n; j++)
                    a[i][j] ^= a[r][j];

        r++;//处理完当前行，进入下一行
    }
    if (r < n)
    {
        for (int i = r; i < n; i++)
            if (a[i][n]) return 2;
        return 1;
    }
    for (int i = n - 1; i >= 0; i--)
        for (int j = i + 1; j < n; j++)
            a[i][n] ^= a[i][j] & a[j][n];
    return 0;
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j <= n; j++)
            cin >> a[i][j];

    int t = gauss();
```

```
    if (t == 0) for (int i = 0; i < n; i++) cout << a[i][n] << endl;
    else if (t == 1) puts("Multiple sets of solutions");
    else puts("No solution");
    return 0;
}
```

数学知识-求组合数

组合数学概述

组合是数学的重要概念之一。从n个不同元素中，任取$m(m \leq n)$个元素并成一组，叫做从n个不同元素中取出m个元素的一个组合；从n个不同元素中取出$m(m \leq n)$个元素的所有组合的个数，叫做从n个不同元素中取出m个元素的组合数，记为C_n^m。

计算组合数的一些基本公式：

- $C_n^m = \frac{n!}{m!(n-m)!}, C_n^0 = 1$
- $C_n^m = \frac{A_n^m}{A_m^m} = \frac{n(n-1)(n-2)\dots(n-m+1)}{m!}$
- $C_n^m = C_n^{n-m} = C_{n-1}^{m-1} + C_{n-1}^m$

在算法题中求解组合数的方式有很多种，需要根据题目的数据范围特点选择不同的方式求解，接下来分别对每种情况进行讨论。

一、AcWing 885. 求组合数 I

【题目描述】

给定n组询问，每组询问给定两个整数a, b，请你输出$C_a^b \ mod (10^9 + 7)$的值。

【输入格式】

第一行包含整数n。

接下来n行，每行包含一组a和b。

【输出格式】

共n行，每行输出一个询问的解。

【数据范围】

$1 \leq n \leq 100000$

$1 \leq b \leq a \leq 2000$

【输入样例】

```
3
3 1
5 3
2 2
```

【输出样例】

```
3
10
1
```

【分析】

对于多组询问，如果每次都进行计算，那么会TLE，因此可利用前言所说的第三个基本公式（$C_n^m = C_n^{n-m} = C_{n-1}^{m-1} + C_{n-1}^m$）进行预处理算出每组$a, b$的$C_a^b$值，对于每次查询所需要的时间为（$O(1)$）。

该公式可用动态规划的思想进行分析：

状态表示：$c[n][m]$表示从n个物品中选m个的方案集合

状态转移：假设要从i个物品中选j个，当前考虑第i个物品，那么有两种情况：

- 不选第i个物品，那么就相当于从$i-1$个物品中选j个，即$c[i-1][j]$
- 选第i个物品，那么就相当于从$i-1$个物品中选$j-1$个，即$c[i-1][j-1]$

因此我们的状态转移方程为：$c[i][j] = c[i-1][j] + c[i-1][j-1], c[i][0] = 1$

【代码】

```
#include <iostream>
using namespace std;

const int N = 2010, MOD = 1e9 + 7;
int c[N][N];
int n;

int main()
{
    for (int i = 0; i < N; i++)

        for (int j = 0; j <= i; j++)
```

```
            if (!j) c[i][j] = 1;
            else c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % MOD;
    cin >> n;
    while (n--)
    {
        int a, b;
        cin >> a >> b;
        cout << c[a][b] << endl;
    }
    return 0;
}
```

二、AcWing 886. 求组合数 II（乘法逆元）

【题目描述】

与问题一相同

【数据范围】

$1 \leq n \leq 10000$

$1 \leq b \leq a \leq 10^5$

【分析】

当a, b较大时，无法预处理所有不同组合的a, b的组合数，因此可以预处理出阶乘$fact[i]$的值，对于每次询问，使用公式$C_n^m = \frac{n!}{m!(n-m)!}$计算即可，由于$\frac{n!}{m!(n-m)!}\ mod(10^9+7)$与$\frac{n!\ mod(10^9+7)}{m!(n-m)!\ mod(10^9+7)}$不同，因此需要将除法转变为乘法，预处理出阶乘的逆元$infact[i]$即可。

【代码】

```
#include <iostream>
using namespace std;

typedef long long LL;
const int N = 100010, MOD = 1e9 + 7;
LL fact[N], infact[N];
int n;

LL quickMi(LL a, LL b, LL p)

{
```

```
    LL res = 1;
    while (b)
    {
        if (b & 1) res = res * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return res % p;
}

int main()
{
    fact[0] = infact[0] = 1;
    for (int i = 1; i < N; i++)
    {
        fact[i] = fact[i - 1] * i % MOD;
        infact[i] = infact[i - 1] * quickMi(i, MOD - 2, MOD) % MOD;
    }
    cin >> n;
    while (n--)
    {
        int a, b;
        cin >> a >> b;
        cout << fact[a] * infact[b] % MOD * infact[a - b] % MOD << endl;
    }
    return 0;
}
```

三、AcWing 887. 求组合数 III（Lucas定理）

【题目描述】

给定n组询问，每组询问给定三个整数a,b,p，其中p是质数，请你输出$C_a^b \mod p$的值。

【输入格式】

第一行包含整数n。

接下来n行，每行包含一组a,b,p。

【输出格式】

共n行，每行输出一个询问的解。

【数据范围】

$1 \leq n \leq 20$

$1 \leq b \leq a \leq 10^{18}$

$1 \leq p \leq 10^5$

【输入样例】

```
3
5 3 7
3 1 5
6 4 13
```

【输出样例】

```
3
3
2
```

【分析】

对于a,b特别大的情况，可使用卢卡斯定理（$Lucas$），该定理为：

$$C_a^b \ (mod\ p) = C_{a\%p}^{b\%p} \times C_{a/p}^{b/p} \ (mod\ p)$$

即当a,b大于p时，递归求解$C(a\%p, b\%p)$，当a,b小于p时，直接返回$C(a,b)$，求解C的时候可用定义进行计算。

【代码】

```cpp
#include <iostream>
using namespace std;

typedef long long LL;
LL a, b, p;
int n;

LL quickMi(LL a, LL b)
{
    LL res = 1;
    while (b)
    {
        if (b & 1) res = res * a % p;
```

```
        a = a * a % p;
        b >>= 1;
    }
    return res % p;
}

LL C(LL a, LL b)
{
    LL res = 1;
    for (int i = 1, j = a; i <= b; i++, j--)
        res = res * j % p * quickMi(i, p - 2) % p;
    return res;
}

LL lucas(LL a, LL b)
{
    if (a < p && b < p) return C(a, b);
    return C(a % p, b % p) * lucas(a / p, b / p) % p;
}

int main()
{
    cin >> n;
    while (n--)
    {
        cin >> a >> b >> p;
        cout << lucas(a, b) << endl;
    }
    return 0;
}
```

四、AcWing 888. 求组合数 IV（高精度，分解质因数）

【题目描述】

输入a, b，求C_a^b的值。

注意结果可能很大，需要使用高精度计算。

【输入格式】

共一行，包含两个整数a和b。

【输出格式】

共一行，输出C_a^b的值。

【数据范围】

$1 \leq b \leq a \leq 5000$

【输入样例】

```
5 3
```

【输出样例】

```
10
```

【分析】

首先有公式：$C_a^b = \frac{a!}{b!(a-b)!}$，对分子和分母分解质因数，最终某个质因数$p$出现的次数为分子$a!$中$p$的次数减去分母$b!(a-b)!$中$p$的次数，那么如何快速求出$n!$中$p$的次数呢？

$cnt_{(n)} = \lfloor \frac{n}{p} \rfloor + \lfloor \frac{n}{p^2} \rfloor + \lfloor \frac{n}{p^3} \rfloor + \dots$

比如：$8! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8$，则$1 \sim 8$之间含$2^1$的数有$2, 4, 6, 8$，含$2^2$的数有$4, 8$（但其中的$2^1$已经被记过一次了因此这次只需要记一次），含$2^3$的数有$8$（同理只记一次）。

最后将化简后的所有的质因数进行高精乘即为答案。

【代码】

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

const int N = 5010;
int prime[N], cnt;
bool st[N];
int sum[N];//记录每个质数的最终次数

//筛出1~n中的质数
void get_primes(int n)
{
```

```
    for (int i = 2; i <= n; i++)
    {
        if (!st[i]) prime[cnt++] = i;
        for (int j = 0; prime[j] <= n / i; j++)
        {
            st[i * prime[j]] = true;
            if (i % prime[j] == 0) break;
        }
    }
}

//返回n!中p的次数
int get(int n, int p)
{
    int res = 0;
    while (n)
    {
        res += n / p;
        n /= p;
    }
    return res;
}

//高精乘
vector<int> mul(vector<int> &A, int b)
{
    vector<int> res;
    int t = 0;
    for (int i = 0; i < A.size() || t; i++)
    {
        if (i < A.size()) t += A[i] * b;
        res.push_back(t % 10);
        t /= 10;
    }
    while (res.size() > 1 && res.back() == 0) res.pop_back();
    return res;
}

int main()
{
    int a, b;
    cin >> a >> b;

    get_primes(a);
```

```
    for (int i = 0; i < cnt; i++)
        sum[i] += get(a, prime[i]) - get(a - b, prime[i]) - get(b, prime[i]);
    vector<int> res = { 1 };
    for (int i = 0; i < cnt; i++)
        for (int j = 0; j < sum[i]; j++)
            res = mul(res, prime[i]);
    for (int i = res.size() - 1; i >= 0; i--) cout << res[i];
    return 0;
}
```

数学知识-容斥原理

一、AcWing 890. 能被整除的数

【题目描述】

给定一个整数n和m个不同的质数$p_1,p_2,\ldots,p_m$。

请你求出$1 \sim n$中能被$p_1,p_2,\ldots,p_m$中的至少一个数整除的整数有多少个。

【输入格式】

第一行包含整数n和m。

第二行包含m个质数。

【输出格式】

输出一个整数，表示满足条件的整数的个数。

【数据范围】

$1 \leq m \leq 16$

$1 \leq n,p_i \leq 10^9$

【输入样例】

```
10 2
2 3
```

【输出样例】

```
7
```

【分析】

本题如果使用暴力枚举的话时间复杂度为（$O(nm)$），必定TLE，因此需要使用**容斥原理**求解。

什么是容斥原理？

假设有集合：S_1, S_2, S_3，那么有如下式子成立：

$|S_1 \cup S_2 \cup S_3|$

$= |S_1| + |S_2| + |S_3| - |S_1 \cap S_2| - |S_1 \cap S_3| - |S_2 \cap S_3| + |S_1 \cap S_2 \cap S_3|$

即枚举所有集合的不同交集组合，**若组合中集合的个数为奇数那么系数为1否则为-1**，所有集合并集的元素个数即为以上不同交集组合的元素个数之和（注意系数正负）。

回到本题，对于$1 \sim n$，假设能被p_i整除的数的集合为S_{p_i}，那么求解的答案即为：$|S_{p_1} \cup S_{p_2} \cup \cdots \cup S_{p_m}|$。

每个集合实际上并不需要知道具体元素是什么，只要知道这个集合的大小，大小为$|S_i| = n/p_i$，比如题目中$|S_1| = 10/2 = 5, |S_2| = 10/3 = 3$。

交集的大小如何确定？因为p_i均为质数，这些质数的乘积就是他们的最小公倍数，n除这个最小公倍数就是交集的大小，故$|S_1 \cap S_2| = n/(p_1 * p_2) = 10/(2 * 3) = 1$。

如何表示每个集合的状态？考虑到集合数最多为$m(m \leq 16)$，因此可用一个16位二进制数的每一位来分别表示每个集合的状态，1表示选择该集合，0表示不选，枚举$1 \sim 2^m - 1$即可表示所有不同集合的组合方式。

【代码】

```cpp
#include <iostream>
using namespace std;

typedef long long LL;
const int N = 20;
int p[N];
int n, m;

int main()
{
    cin >> n >> m;
    for (int i = 0; i < m; i++) cin >> p[i];

    LL res = 0;
```

```
    //枚举1~2^m-1，表示所有集合组合的状态
    for (int i = 1; i < 1 << m; i++)
    {
        LL t = 1, cnt = 0;//t表示选中的集合的最小公倍数，cnt表示组合中的集合数
        for (int j = 0; j < m; j++)//枚举i的每一位，即每一个集合的状态
            if (i >> j & 1)//如果第j位为1说明组合中有p[j]这个集合
            {
                cnt++;
                t *= p[j];
                if (t > n)//如果最小公倍数大于n就没必要继续计算了
                {
                    t = -1;
                    break;
                }
            }
        if (t != -1)
            if (cnt % 2) res += n / t;//如果是奇数个集合那么结果是加上集合中元素的个数
            else res -= n / t;//否则是减去集合中元素的个数
    }
    cout << res << endl;
    return 0;
}
```

数学知识-博弈论

一、AcWing 891. Nim游戏

【题目描述】

给定n堆石子，两位玩家轮流操作，每次操作可以从任意一堆石子中拿走任意数量的石子（可以拿完，但不能不拿），最后无法进行操作的人视为失败。

问如果两人都采用最优策略，先手是否必胜。

【输入格式】

第一行包含整数n。

第二行包含n个数字，其中第i个数字表示第i堆石子的数量。

【输出格式】

如果先手方必胜，则输出`Yes`。

否则，输出`No`。

【数据范围】

$1 \leq n \leq 10^5$

$1 \leq$ 每堆石子数 $\leq 10^9$

【输入样例】

```
2
2 3
```

【输出样例】

```
Yes
```

【分析】

首先给出结论：若$a_1 \wedge a_2 \wedge \cdots \wedge a_n = 0$（$a_i$表示第$i$堆石子的数量），则先手必败，否则先手必胜。

在讲Nim游戏之前，先了解一下什么是公平组合游戏(ICG)。若一个游戏满足以下条件，则该游戏称为公平组合游戏：

- 由两名玩家交替行动
- 在游戏进行的任意时刻，可以执行的合法行动与轮到哪位玩家无关
- 不能行动的玩家判负

Nim游戏属于公平组合游戏，但常见的棋类游戏，比如围棋就不是公平组合游戏，因为围棋交战双方分别只能落黑子和白子，胜负判定也比较复杂，不满足条件2和3。

什么是**先手必胜状态**和**先手必败状态**？

- 先手必胜状态：先手进行**某一个操作**，留给后手是一个必败状态时，对于先手来说是一个必胜状态。即先手可以走到**某一个**必败状态。
- 先手必败状态：先手**无论如何操作**，留给后手都是一个必胜状态时，对于先手来说是一个必败状态。即先手走不到**任何一个**必败状态。

为什么当$a_1 \wedge a_2 \wedge \cdots \wedge a_n = 0$时先手必败，否则先手必胜？

1. 当到达结束状态时，即无法再进行任何操作，此时$0 \wedge 0 \wedge \cdots \wedge 0 = 0$；

2. 如果其中某一个状态$a_1 \wedge a_2 \wedge \cdots \wedge a_n = x$（$x$不为$0$），那么一定有一种拿法能够使得剩下的数异或值为$0$。证明如下：

   假设x的二进制表示里最高的一位1在第k位，则$a_1 \sim a_n$中至少有一个数a_i的第k位是1（如果全为0那么x的第k位一定是0），那么显然$a_i \wedge x < a_i$，所以我们可以从中拿走$a_i - (a_i \wedge x)$个石子，那么第i堆石子就剩$a_i - (a_i - (a_i \wedge x)) = a_i \wedge x$个，即原式变为：$a_1 \wedge a_2 \wedge \cdots \wedge (a_i \wedge x) \wedge \cdots \wedge a_n$。由于$a_1 \wedge a_2 \wedge \cdots \wedge a_i \wedge \cdots \wedge a_n = x$，因此上式变为$x \wedge x = 0$。

3. 如果其中某一个状态$a_1 \wedge a_2 \wedge \cdots \wedge a_n = 0$，那么不管怎么拿，剩下所有数的异或值一定不为$0$，用反证法证明如下：

   假设第i堆拿走一部分石子后剩余石子数为a_i'，且$a_1 \wedge a_2 \wedge \cdots \wedge a_i' \wedge \cdots \wedge a_n = 0$，而由于已知原状态为$a_1 \wedge a_2 \wedge \cdots \wedge a_i \wedge \cdots \wedge a_n = 0$，将两个式子进行异或，由于除了$a_i$、$a_i'$以外其余数都是成对出现，异或值为$0$，因此结果为$a_i \wedge a_i' = 0$，所以$a_i = a_i'$，又由于假设第$i$堆已经拿走一部分石子了，与$a_i = a_i'$矛盾，因此假设不成立，原命题成立。

综上，当先手局面为$a_1 \wedge a_2 \wedge \cdots \wedge a_n = 0$时抛给后手的状态一定为$a_1 \wedge a_2 \wedge \cdots \wedge a_n \neq 0$，当先手局面为$a_1 \wedge a_2 \wedge \cdots \wedge a_n \neq 0$时抛给后手的状态一定是$a_1 \wedge a_2 \wedge \cdots \wedge a_n = 0$。所以当$a_1 \wedge a_2 \wedge \cdots \wedge a_n = 0$时先手必败，否则先手必胜。

【代码】

```cpp
#include <iostream>
using namespace std;

int main()
{
    int n, res = 0;
    cin >> n;
    while (n--)
    {
        int x;
        cin >> x;
        res ^= x;
    }
    if (res) puts("Yes");
    else puts("No");
    return 0;
}
```

二、AcWing 892. 台阶-Nim游戏

【题目描述】

现在，有一个n级台阶的楼梯，每级台阶上都有若干个石子，其中第i级台阶上有a_i个石子$(i \geq 1)$。

两位玩家轮流操作，每次操作可以从任意一级台阶上拿若干个石子放到下一级台阶中（不能不拿）。

已经拿到地面上的石子不能再拿，最后无法进行操作的人视为失败。

问如果两人都采用最优策略，先手是否必胜。

【输入格式】

第一行包含整数n。

第二行包含n个整数，其中第i个整数表示第i级台阶上的石子数a_i。

【输出格式】

如果先手方必胜，则输出`Yes`。

否则，输出`No`。

【数据范围】

$1 \leq n \leq 10^5$

$1 \leq a_i \leq 10^9$

【输入样例】

```
3
2 1 3
```

【输出样例】

```
Yes
```

【分析】

结论：当奇数级阶梯的石子数量异或值为0，即$a_1 \wedge a_3 \wedge \cdots \wedge a_{2n-1} = 0$时，先手必败，否则先手必胜。证明思路同经典$Nim$游戏相似。

【代码】

```cpp
#include <iostream>
using namespace std;

int main()
{
    int n, res = 0;
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        int x;
        cin >> x;
        if (i % 2) res ^= x;
    }
    if (res) puts("Yes");
    else puts("No");
    return 0;
}
```

三、AcWing 893. 集合-Nim游戏（SG函数）

【题目描述】

给定n堆石子以及一个由k个不同正整数构成的数字集合S。

现在有两位玩家轮流操作，每次操作可以从任意一堆石子中拿取石子，每次拿取的石子数量必须包含于集合S，最后无法进行操作的人视为失败。

问如果两人都采用最优策略，先手是否必胜。

【输入格式】

第一行包含整数k，表示数字集合S中数字的个数。

第二行包含k个整数，其中第i个整数表示数字集合S中的第i个数s_i。

第三行包含整数n。

第四行包含n个整数，其中第i个整数表示第i堆石子的数量h_i。

【输出格式】

如果先手方必胜，则输出`Yes`。

否则，输出`No`。

【数据范围】

$1 \leq n, k \leq 100$

$1 \leq s_i, h_i \leq 10000$

【输入样例】

```
2
2 5
3
2 4 7
```

【输出样例】

```
Yes
```

【分析】

（1）Mex运算：

设S表示一个非负整数集合，定义$mex(S)$为求出不属于集合S的**最小非负整数**运算，即：

$mex(S) = min\{x\}$;

例如：$S = \{0, 1, 2, 4\}$，那么$mex(S) = 3$。

（2）SG函数

在有向图游戏中，对于每个节点x，设从x出发共有k条有向边，分别到达节点$y_1, y_2, \ldots, y_k$，定义$SG(x)$为x的后继节点$y_1, y_2, \ldots, y_k$的SG函数值构成的集合执行mex运算的结果，即：

$SG(x) = mex(\{SG(y_1), SG(y_2), \ldots, SG(y_k)\})$

特别地，整个有向图游戏G的SG函数值被定义为有向图游戏**起点**s的SG函数值，即：$SG(G) = SG(s)$。

定义终点的SG函数值为0。

（3）有向图游戏的和

设$G_1, G_2, \ldots, G_m$是m个有向图游戏。定义有向图游戏G，他的行动规则是任选某个有向图游戏G_i，并在G_i上行动一步，G被称为有向图游戏$G_1, G_2, \ldots, G_m$的和。

有向图游戏的和的SG函数值等于它包含的各个子游戏SG函数的**异或和**，即：

$SG(G) = SG(G_1) \wedge SG(G_2) \wedge \cdots \wedge SG(G_m)$。

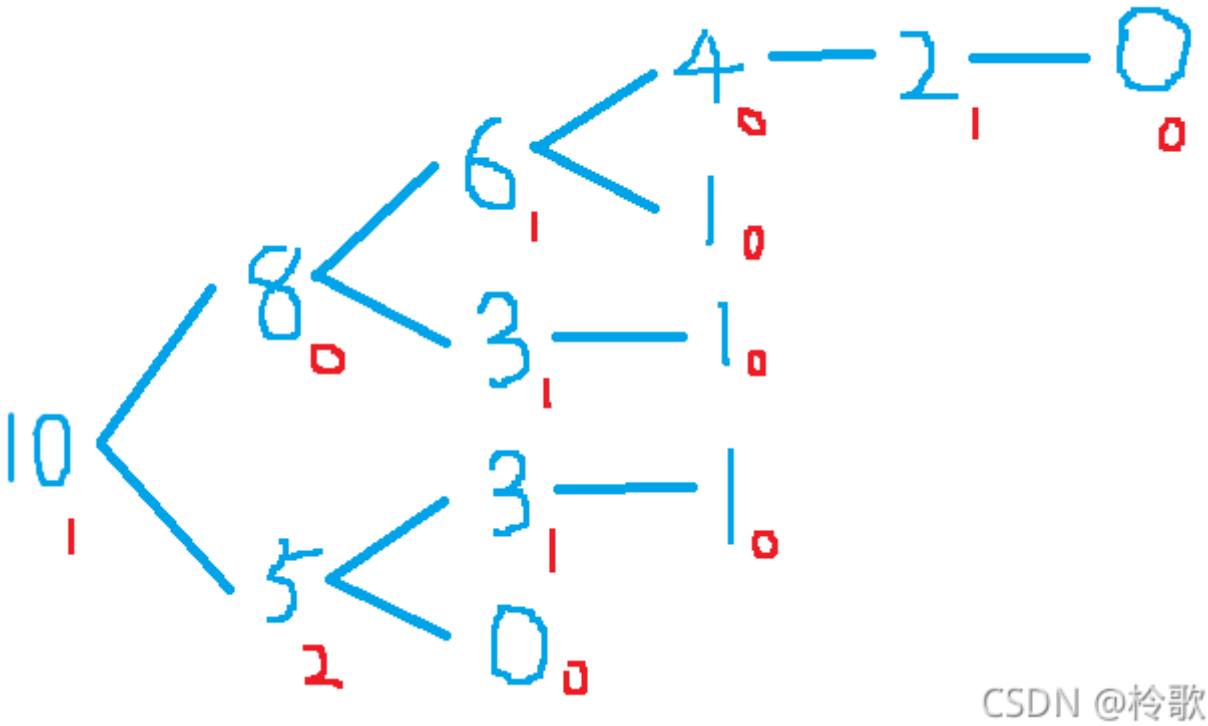
（4）结论

- 对于一个图G，如果$SG(G) \neq 0$，则先手必胜，反之必败。

  证明：根据mex运算的定义可知，如果$SG(G) \neq 0$，那么其连接的一点必为0（如果其后继结点的值均不为0，那么该点的mex结果必定为0），当走到SG值为0的这一点时，由于mex运算，该结点所连接的结点的SG值必定不为0。又因为终点状态的SG值为0，所以只要先手的SG值不为0，便可以一直走向SG值为0的状态，最终走向终点。

- 对于n个图，如果$SG(G_1) \wedge SG(G_2) \wedge \cdots \wedge SG(G_n) \neq 0$，则先手必胜，反之必败。

例子：假设有一堆石子，石子数量为10，每次可以拿走2个或5个，那么其有向图如下图所示：



红色标注的数字为该结点的SG函数值，由于起点10的SG函数值不为0，因此该游戏局面为先手必胜状态。

【代码】

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <unordered_set>
using namespace std;
```

```cpp

const int N = 110, M = 10010;
int s[N], f[M];
int n, k;

//记忆化搜索保存各结点的SG值
int sg(int x)
{
    if (~f[x]) return f[x];
    unordered_set<int> st;//保存x能到达的后继结点的SG值
    //枚举x能到达的所有状态
    for (int i = 0; i < k; i++)
        if (s[i] <= x) st.insert(sg(x - s[i]));
    //mex操作，找出x后继结点中未出现过的最小的非负整数
    for (int i = 0; ; i++)
        if (!st.count(i))
            return f[x] = i;
}

int main()
{
    cin >> k;
    for (int i = 0; i < k; i++) cin >> s[i];
    memset(f, -1, sizeof f);
    int res = 0;
    cin >> n;
    while (n--)
    {
        int x;
        cin >> x;
        res ^= sg(x);//对每个有向图的起始结点的SG值进行异或运算
    }
    if (res) puts("Yes");
    else puts("No");
    return 0;
}
```

四、AcWing 894. 拆分-Nim游戏

【题目描述】

给定n堆石子，两位玩家轮流操作，每次操作可以取走其中的一堆石子，然后放入两堆**规模更小**的石子（新堆规模可以为0，且两个新堆的石子总数可以大于取走的那堆石子数），最后无法进行操作的人视为失败。

问如果两人都采用最优策略，先手是否必胜。

【输入格式】

第一行包含整数n。

第二行包含n个整数，其中第i个整数表示第i堆石子的数量a_i。

【输出格式】

如果先手方必胜，则输出`Yes`。

否则，输出`No`。

【数据范围】

$1 \leq n, a_i \leq 100$

【输入样例】

```
2
2 3
```

【输出样例】

```
Yes
```

【分析】

相比于集合Nim游戏，这里的每一堆可以变成**小于原来那堆的任意大小的两堆**。

即$a[i]$可以拆分成$(b[i], b[j])$，为了避免重复规定$b[i] \geq b[j]$，即：$a[i] > b[i] \geq b[j]$。

相当于一个局面拆分成了两个局面，由SG函数理论，多个独立局面的SG值，**等于这些局面SG值的异或和**。

因此需要存储的状态就是$sg(b[i]) \wedge sg(b[j])$。

【代码】

```
#include <iostream>
#include <cstring>
```

```cpp
#include <algorithm>
#include <unordered_set>
using namespace std;

const int N = 110;
int f[N];
int n;

int sg(int x)
{
    if (~f[x]) return f[x];
    unordered_set<int> st;
    //枚举x可变成的所有状态(i, j)，防止重复j只用枚举到i即可
    for (int i = 0; i < x; i++)
        for (int j = 0; j <= i; j++)
            st.insert(sg(i) ^ sg(j));
    //mex操作
    for (int i = 0; ; i++)
        if (!st.count(i))
            return f[x] = i;
}

int main()
{
    cin >> n;
    memset(f, -1, sizeof f);
    int res = 0;
    while (n--)
    {
        int x;
        cin >> x;
        res ^= sg(x);
    }
    if (res) puts("Yes");
    else puts("No");
    return 0;
}
```