# 数论

# 1.gcd(最大公约数)

- 头文件algorithm,库函数: __gcd(a,b).

# 2.lcm(最小公倍数)

- 可由gcd函数推导:

$$lcm(a,b) = a/gcd(a,b) * b$$

### 考虑到gcd函数只在linux下可用，使用辗转相除定义gcd

```
int gcd (int a,int b){
        int tmp;
        while(b > 0){
               tmp = a % b;
               a = b;
               b =tmp;
                }
        return a;
}
```

# 3.素数筛法

## 1.埃氏筛法

```
bitset<maxn> pri;//全部初始化为0，令0为素数，1为合数（被筛除）
int cnt = 0,N = 1e7;
int aishi()
{
        for(int i = 2;i <= N / i;++i)
        {
                if(!pri[i])
                        for(int j = i * i;j <= N;j++)pri[j] = 1;
        }
        for(int i = 2;i <= N;++i)if(!pri[i])cnt ++;
        return cnt;
}
```

## 2.欧拉筛法

```
bitset<maxn> pri;
int primes[maxn];//按序储存被筛出来的素数
int pp=0,cnt =0;
int ola()
{
        for(int i = 2;i<=N;++i)
        {
                if(!pri[i])primes[++pp] = i;
                for(int j = 1;j<= pp;++j)
                {
                        pri[primes[j] *i] = 1;
                        if(i % primes[j] == 0)break;
                }
        }
        for(int i = 2;i <= N;++i)if(!pri[i])cnt ++;
        return cnt;

}
```

# 4.唯一分解定理

## 1.定理内容

**任何一个大于1的整数n 都可以分解成若干个素因数的连乘积，如果不计各个素因数的顺序，那么这种分解是惟一的。**

```
for(int i = 1; i <= len; i++)
{
       while(n % prime[i] == 0)//3 % 2  = 1 循环停止
       {
            ans[k++] = prime[i];
            n / = prime[i];
        }
}
```

### ans数组记录下每一个约数，在此之前需要先筛出因数,下面贴一个题，找出逻辑顺序唯一分解的各个素数

```
#include<iostream>
#include<cstdio>
#include<algorithm>
using namespace std;
const int N = 1e8;
int n;
int prime[10000], ans[10000];
bool pri[10000];
```

```
int pp = 0;
void ola()
{
        for (int j = 2; j <= n; j++)
        {
                if (pri[j] == false)prime[++pp] = j;
                for (int ii = 1; ii <= pp; ii++)
                {
                        pri[prime[ii] * j] = true;
                        if (j % prime[ii] == 0)
                        {
                                break;
                        }
                }
        }

}
void find(int m)
{
        int cnt = 0;
        for (int ii = 1; ii <= pp; ii++)
        {

                while (m % prime[ii] == 0)
                {
                        ans[cnt++] = prime[ii];
                        m = m / prime[ii];
                }
        }

}
int main()
{
        cin >> n;
        ola();
        find(n);
        int q;
        cin >> q;
        std::ios::sync_with_stdio(0);
        while (q--)
        {
                int k;
                cin >> k;
                cout << ans[k-1]<< endl;
        }
        return 0;
}
```

## 2.定理应用

# 1.求出数n的因子个数(求出n的约数个数)

$$\text{因子个数} = (1+a_1)\cdot(1+a_2)\cdots(1+a_n)(1+a1)\cdot(1+a2)\cdots(1+an)$$

# 2.求所有因子之和

$$\text{因子之和} = (P_1^0+P_1^1+P_1^2+\cdots+P_1^{a1})\cdot(P_2^0+P_2^1+P_2^2+\cdots+P_2^{a_2})\cdots(P_n^0+P_n^1+P_n^2+\cdots+P_n$$

# 3.求gcd和lcm

- 设两个数a,b

$$a = P_1^{a_1}\cdot P_2^{a_2}\cdot P_3^{a_3}\cdots P_n^{a_n}$$

$$b = P_1^{b_1}\cdot P_2^{b_2}\cdot P_3^{b_3}\cdots P_n^{b_n}$$

$$gcd(a,b) = P_1^{min(a_1,b_1)}\cdot P_2^{min(a2,b2)}\cdot P_3^{min(a_3,b_3)}\cdots P_n^{min(a_n,b_n)}$$

- 反之求lcm就是把上面的min全部换成max

# 5. 扩展欧几里得

## 前置知识：裴蜀定理

- 对于给定的整数a,b，方程ax+by = gcd(a,b)总有整数解（x,y）
- 即gcd(a,b) = gcd(b,a%b)

## 扩展欧几里得定义

- 用于求解线性同余方程的ax + by = gcd(x,y）的一组正整数特解x,y

```
int gcd(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 t = y;
        y= x - a/b*y;
        x= t;
        return d;
}
int main()
```

```
{
        int n;
        scanf("%d",&n);
        while(n--)
        {
                int a,b;
                cin >> a >> b;
                int x ,y;
                exgcd(a,b,x,y);
                printf("%d %d/n", x,y);
        }
        return 0;
}
```

# 6.欧拉函数

## 作用是

- 求出小于等于该数的正整数中与n互质的数的个数
- 当n为质数时

$$\varphi(n) = n - 1$$

- 当n为合数时

$$\varphi(n) = \varphi(p^k) = p^k - p^{k-1} = (p-1) \times p^{k-1}$$

```
int oula(int n)//计算N的欧拉函数
{
        int res = n; //保存 原n及结果
        for(int i = 2; i <=n / i;i++)
        {
                if(n %i == 0)
                {
                        res = res / n*(n-1);
                        while(n % i == 0)
                        n=n/i;
                }
        }
        if(n > 1)res = res/n *(n-1);
        return res;
}
```

# 7.快速幂

## 意义

- 通过计算机取模运算的原理(ab)%p = [(a % p)(b %p)]%p;
- 得到超高指数的模

```
typedef long long ll;
ll fast_power(ll a,ll b,ll c){
        ll ans = 1;
        a %= c;
        while(b)
        {
                if(b&1)
                {
                         ans = (ans*a)  %c;

                }
                a = (a*a)%c;
                b /= 2;
        }
        return ans;
}
```

# 8.乘法逆元

## 定义

- 对于a*b≡1(mod p),则b是a在模m下a的逆元。
- 只有a与b互质时存在逆元。

## 作用

- 余数的基本性质中只包括加减乘，当面对除法时，逆元就相当于是个倒数，可以将除法变为乘法，方便进行模运算。

## 余数基本性质

(a+-b)%c = ((a%c) +- (b%c)) % c;
(ab)%c = ((a%c)(b%c)) % c;

## 应用

### 1.当p为质数且p值较大时由费马小定理$a^{(p-1)}$≡ 1(mod p)得，逆元b为$a^{(p-2)}$;

- b * $b^{(p-2)}$≡ 1(mod p)
- 取

$$b_i = b^{p-2}$$

```
typedef long long ll;//快速幂计算b的p-2次方，即binv
ll fast_pow(ll a,ll b,ll c)
{
        ll ans = 1;a %= p;
        while(b){
                if(b & 1) ans = (ans * a )% p;
                a = (a * a) % p;
                b >>= 1;
        }
        return ans;
}
ll get_inv(ll x,ll p)//费马小定理传值
{
        return fast_pow (x,p - 2,p);
}
```

### 2.当p不为质数时，使用扩展欧几里得定理

```
void exgcd(int a,int b,int &x,int &y)
{
        if(b == 0){x = 1;y = 0;}
        int gcd = exgcd(b, a%b, y ,x);
        y = y-(a / b) * x;
}
int get_inv(int a,int p)
{
        int x = 1,y = 0;
        exgcd(a, p, x, y);
        return (x%p + p) % p;//防止出现负数
}
// a*a_i + py = 1;—>对应exgcd中的a,p
```

### 3.线性求逆元

- 对于一个数x，由于p > x,设 p= kx + r, k = [p/k], r= p % x;
- 本题要求x的逆元

```
int inv[MAXN];
inv[1] = 1;
for(int i = 2;i ≤n;++i)
{
        inv[i] = -(p / i) * inv[p % i];//x_inv = -k * r_inv;
        inv[i] = (inv[i] % p + p) % p;// x_inv = (x_inv % p + p)%p;
}
```

## 9.矩阵快速幂

```
#include<algorithm>
#include<iostream>
#include<cstring>
#include<cstdio>
#include<cctype>
#define ll long long
#define gc() getchar()
#define maxn 105
#define mo 1000000007
using namespace std;

ll read(){
	ll a=0;int f=0;char p=gc();
	while(!isdigit(p)){f|=p=='-';p=gc();}
	while(isdigit(p)){a=(a<<3)+(a<<1)+(p^48);p=gc();}
	return f?-a:a;
}
int n;

struct ahaha{
	ll a[maxn][maxn];	//一定要用long long存矩阵，否则在过程中会爆掉
	ahaha(){
		memset(a,0,sizeof a);
	}
	inline void build(){	//建造单位矩阵
		for(int i=1;i<=n;++i)a[i][i]=1;
	}
}a;
ahaha operator *(const ahaha &x,const ahaha &y){	//重载运算符
	ahaha z;
	for(int k=1;k<=n;++k)
		for(int i=1;i<=n;++i)
			for(int j=1;j<=n;++j)
				z.a[i][j]=(z.a[i][j]+x.a[i][k]*y.a[k]
[j]%mo)%mo;
	return z;
}

ll k;
inline void init(){
	n=read();k=read();
	for(int i=1;i<=n;++i)
		for(int j=1;j<=n;++j)
			a.a[i][j]=read();
}

int main(){
	init();
	ahaha ans;ans.build();
```

```
        do{     //递推快速幂，与普通的递推快速幂无异，但*不能缩写为*=
                if(k&1)ans=ans*a;
                a=a*a;k>>=1;
        }while(k);
        for(int i=1;i<=n;putchar('\n'),++i)
                for(int j=1;j<=n;++j)
                        printf("%d ",ans.a[i][j]);
        return 0;
}
```

# 10.高斯消元

- 解包含n个方程n个未知数的线性方程组
- 例题中，n行，n + 1 个未知数
- 二维数组 `a` 去存储整个方程组，
- 因为精度问题，我们可以用一个特别小的数去当 `0`，当运算结果小于我们设定的这个十分小的数的时候，默认为 `0`
- 我们用 `const double eps = 1e-6;` 去表达这个无穷小的数.
- 我们用 `c` 去表示列，用 `r` 去表示行

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

using namespace std;

const int N = 110;
const double eps = 1e-6;

int n;
double a[N][N];


int gauss()
{
    int c, r;
    for (c = 0, r = 0; c < n; c ++ )
    {
        int t = r;
        for (int i = r; i < n; i ++ )
            if (fabs(a[i][c]) > fabs(a[t][c]))
                t = i;

        if (fabs(a[t][c]) < eps) continue;

        for (int i = c; i < n + 1; i ++ ) swap(a[t][i], a[r][i]);
        for (int i = n; i >= c; i -- ) a[r][i] /= a[r][c];
```

```
        for (int i = r + 1; i < n; i ++ )
            if (fabs(a[i][c]) > eps)//这个数不是0再进行操作
                for (int j = n; j >= c; j -- )
                    a[i][j] -= a[r][j] * a[i][c];

        r ++ ;
    }

    if (r < n)
    {
        for (int i = r; i < n; i ++ )
            if (fabs(a[i][n]) > eps)//即等式左边全部为0，但是等式右边的值不为0
                return 2;//证明无解
        return 1;//证明有无穷多组解
    }

    for (int i = n - 1; i >= 0; i -- )
        for (int j = i + 1; j < n; j ++ )
            a[i][n] -= a[j][n] * a[i][j];

    return 0;//证明有解
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i ++ )
        for (int j = 0; j < n + 1; 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;
}
```

# 11.求组合数

$$C_n^m = C_{n-1}^{m-1} + C_{n-1}^m$$

## 1.递归法求组合数

```
// c[a][b] 表示从a个苹果中选b个的方案数
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;
```

## 2.分解质因数法求组合数

- 当我们需要求出组合数的真实值，而非对某个数的余数时，分解质因数的方式比较好用：
    1. 筛法求出范围内的所有质数
    2. 通过

    $$C(a,b) = a!/b!/(a-b)!$$

    这个公式求出每个质因子的次数。 n! 中p的次数是

    $$n/p + n/p^2 + n/p^3 + \dots$$

    3. 用高精度乘法将所有质因子相乘

```
int primes[N], cnt;     // 存储所有质数
int sum[N];     // 存储每个质数的次数
bool st[N];     // 存储每个数是否已被筛掉


void get_primes(int n)      // 线性筛法求素数
{
    for (int i = 2; i <= n; i ++ )
    {
        if (!st[i]) primes[cnt ++ ] = i;
        for (int j = 0; primes[j] <= n / i; j ++ )
        {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}


int get(int n, int p)       // 求n! 中的次数
{
    int res = 0;
    while (n)
    {
        res += n / p;
        n /= p;
```

```
    }
    return res;
}


vector<int> mul(vector<int> a, int b)       // 高精度乘低精度模板
{
    vector<int> c;
    int t = 0;
    for (int i = 0; i < a.size(); i ++ )
    {
        t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }

    while (t)
    {
        c.push_back(t % 10);
        t /= 10;
    }

    return c;
}

get_primes(a);  // 预处理范围内的所有质数

for (int i = 0; i < cnt; i ++ )     // 求每个质因数的次数
{
    int p = primes[i];
    sum[i] = get(a, p) - get(b, p) - get(a - b, p);
}

vector<int> res;
res.push_back(1);

for (int i = 0; i < cnt; i ++ )     // 用高精度乘法将所有质因子相乘
    for (int j = 0; j < sum[i]; j ++ )
        res = mul(res, primes[i]);
```

## 3.预处理逆元的方法求组合数

- 首先预处理出所有阶乘取模的余数$fact[N]$，以及所有阶乘取模的逆元$infact[N]$
- 如果取模的数是质数，可以用费马小定理求逆元

```
int qmi(int a, int k, int p)    // 快速幂模板
{
```

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

// 预处理阶乘的余数和阶乘逆元的余数
fact[0] = infact[0] = 1;
for (int i = 1; i < N; i ++ )
{
    fact[i] = (LL)fact[i - 1] * i % mod;
    infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
}
```

# 12.分数的加减乘除

## 分数的表示

```
struct Fraction{                  //分数
        int up, down;          //分子，分母
};
```

## 分数的化简

```
Fraction reduction (Fraction result)
{
        if (result.down < 0)
        {
                result.up = - result.up;
                result.down = - result.down;
        }
        if (result.up == 0) result.down = 1;
        else
        {
                int d = gcd(abs(result.up), abs(result.down));
                result.up /= d;
                result.down /= d;
        }
        return result;
}
```

## 分数的加法

```cpp
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <map>

using namespace std;

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

struct Fraction
{
        int up, down;
};

Fraction reduction (Fraction result)
{
        if (result.down < 0)
        {
                result.up = - result.up;
                result.down = - result.down;
        }
        if (result.up == 0) result.down = 1;
        else
        {
                int d = gcd(abs(result.up), abs(result.down));
                result.up /= d;
                result.down /= d;
        }
        return result;
}

Fraction add(Fraction f1, Fraction f2)
{
        Fraction result;
        result.up = f1.up * f2.down + f2.up * f1.down;
        result.down = f1.down * f2.down;
        return reduction(result);
}

void showResult (Fraction r)
{
        r = reduction(r);
        if (r.down == 1) printf("%d", r.up);
```

```
        else if (abs(r.up) > r.down)
                printf("%d %d/%d", r.up / r.down, abs(r.up) % r.down,
r.down);
        else printf("%d/%d", r.up, r.down);
}

int main()
{
        Fraction f1, f2;
        scanf("%d%d", &f1.up, &f1.down);
        scanf("%d%d", &f2.up, &f2.down);
        Fraction res = add(f1, f2);
        showResult(res);
        return 0;
}
```

## 分数的减法

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <map>

using namespace std;

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

struct Fraction
{
        int up, down;
};

Fraction reduction (Fraction result)
{
        if (result.down < 0)
        {
                result.up = - result.up;
                result.down = - result.down;
        }
        if (result.up == 0) result.down = 1;
        else
        {
                int d = gcd(abs(result.up), abs(result.down));
```

```
                result.up /= d;
                result.down /= d;
        }
        return result;
}

Fraction minu(Fraction f1, Fraction f2)
{
        Fraction result;
        result.up = f1.up * f2.down - f2.up * f1.down;
        result.down = f1.down * f2.down;
        return reduction(result);
}

void showResult (Fraction r)
{
        r = reduction(r);
        if (r.down == 1) printf("%d", r.up);
        else if (abs(r.up) > r.down)
                printf("%d %d/%d", r.up / r.down, abs(r.up) % r.down, r.down);
        else printf("%d/%d", r.up, r.down);
}

int main()
{
        Fraction f1, f2;
        scanf("%d%d", &f1.up, &f1.down);
        scanf("%d%d", &f2.up, &f2.down);
        Fraction res = minu(f1, f2);
        showResult(res);
        return 0;
}
```

## 分数的乘法

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <map>

using namespace std;

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

```
}

struct Fraction
{
        int up, down;
};

Fraction reduction (Fraction result)
{
        if (result.down < 0)
        {
                result.up = - result.up;
                result.down = - result.down;
        }
        if (result.up == 0) result.down = 1;
        else
        {
                int d = gcd(abs(result.up), abs(result.down));
                result.up /= d;
                result.down /= d;
        }
        return result;
}

Fraction multi(Fraction f1, Fraction f2)
{
        Fraction result;
        result.up = f1.up * f2.up;
        result.down = f1.down * f2.down;
        return reduction(result);
}

void showResult (Fraction r)
{
        r = reduction(r);
        if (r.down == 1) printf("%d", r.up);
        else if (abs(r.up) > r.down)
                printf("%d %d/%d", r.up / r.down, abs(r.up) % r.down,
r.down);
        else printf("%d/%d", r.up, r.down);
}

int main()
{
        Fraction f1, f2;
        scanf("%d%d", &f1.up, &f1.down);
        scanf("%d%d", &f2.up, &f2.down);
        Fraction res = multi(f1, f2);
        showResult(res);
        return 0;
```

```
}
```

## 分数的除法

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <map>

using namespace std;

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

struct Fraction
{
        int up, down;
};

Fraction reduction (Fraction result)
{
        if (result.down < 0)
        {
                result.up = - result.up;
                result.down = - result.down;
        }
        if (result.up == 0) result.down = 1;
        else
        {
                int d = gcd(abs(result.up), abs(result.down));
                result.up /= d;
                result.down /= d;
        }
        return result;
}

Fraction divide(Fraction f1, Fraction f2)
{
        Fraction result;
        result.up = f1.up * f2.down;
        result.down = f1.down * f2.up;
        return reduction(result);
}
```

```
void showResult (Fraction r)
{
        r = reduction(r);
        if (r.down == 1) printf("%d", r.up);
        else if (abs(r.up) > r.down)
                printf("%d %d/%d", r.up / r.down, abs(r.up) % r.down,
r.down);
        else printf("%d/%d", r.up, r.down);
}

int main()
{
        Fraction f1, f2;
        scanf("%d%d", &f1.up, &f1.down);
        scanf("%d%d", &f2.up, &f2.down);
        Fraction res = divide(f1, f2);
        showResult(res);
        return 0;
}
```

# 13.中国剩余定理

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 $x$ ， 满足 $\forall i \in [1, n], x \equiv mi(modai)$

```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;
LL exgcd(LL a, LL b, LL &x, LL &y){//扩展欧几里得求ax+by=gcd(a,b)的解
    if(b==0){
        x = 1, y = 0;
        return a;
    }
    LL x1, y1, gcd = exgcd(b,a%b,x1,y1);
    x = y1, y = x1-a/b*y1;
    return gcd;
}
int main(){
    int n,has_ans=1;
    LL a1,m1,t;//第一个方程的系数 备份数据
    cin>>n;
    cin>>a1>>m1;//先输入第一个方程
    for(int i = 2,a2,m2; i <= n; i++){//合并接下来的n-1个方程
        cin>>a2>>m2;
        LL k01,k02,gcd = exgcd(a1,a2,k01,k02);
        if((m2-m1)%gcd){//此时无解
            has_ans = 0;
```

```
            break;
        }
        k01 = k01*(m2-m1)/gcd;//特解
        k01 = (k01 % (a2/gcd) + a2/gcd) % (a2/gcd);//让特解k01取到最小正整数解
        t = a1*a2/gcd;
        m1 += a1*k01;
        a1 = t;
    }
    if(has_ans) cout<<(m1%a1+a1)%a1<<endl;
    else cout<<-1<<endl;
    return 0;
}
```