数论

区间筛

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN = 1e5;
bool isPrime[MAXN]; //根据isPrime[i]判断i+a是不是质数，isPrime[i-a]=true 表示i是素数（下标偏移了a）
bool isSqrtPrime[MAXN]; //根据isSqrtPrime[i]判断i是不是质数
vector<int> prime;
//对区间[a,b)内的整数执行筛法
void SegmentSieve(ll a, ll b)   //对区间[a,b)进行筛选
{
    for (ll i = 0; i * i < b; i++) //初始化[2,sqrt(b))的全为质数
        isSqrtPrime[i] = true;
    for (ll i = 0; i < b - a; i++) //初始化偏移后的[a,b)全为质数
        isPrime[i] = true;

    for (ll i = 2; i * i < b; i++) //埃氏筛选
    {
        if (isSqrtPrime[i]) {
            for (ll j = 2 * i; j  <= sqrt(b); j += i)//筛选上限变为sqrt(b)
                isSqrtPrime[j] = false;
            //筛除[a,b)里的非素数
            //(a+i-1)/i 得到最接近（a/i）这个数，也即（a的i倍数），最低是2LL是表示最低2倍
            for (ll j = max(2LL, (a + i - 1) / i) * i; j < b; j += i)
                isPrime[j - a] = false;
        }
    }
    for (ll i = a; i < b; i++){
        if (isPrime[i - a]) prime.push_back(i);
    }
}


typedef long long ll;
const int MAXN = 1e6 + 5;
vector<int> primes;
void segment_sieve(ll a, ll b) {
    vector<bool> is_prime_small(sqrt(b) + 1, true); // 筛[2, sqrt(b)]
    vector<bool> is_prime(b - a, true);              // 筛[a, b)

    if (a == 1) is_prime[0] = false;

    for (ll p = 2; p * p < b; p++) {
        if (is_prime_small[p]) {
            // 筛小素数
            for (ll j = p * p; j * j < b; j += p)
                is_prime_small[j] = false;
            // 筛大区间
            ll start = max(p * p, (a + p - 1) / p * p);
            for (ll j = start; j < b; j += p)
                is_prime[j - a] = false;
        }
    }

    for (ll i = a; i < b; i++)
        if (is_prime[i - a]) primes.push_back(i);
}
int main(){
    segment_sieve(1000000000, 1001000000);
    for (auto v : primes){
        cout << v << " ";
    }
    return 0;
}



int main(){
    SegmentSieve(100, 10000);
    for (auto v : prime){
        cout << v << " ";
    }
    return 0;
}
```

FTT 普通循环版

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

class Polynomial {
private:
    static const double PI;
    struct Complex {
        double x, y;
        Complex(double _x = 0.0, double _y = 0.0) : x(_x), y(_y) {}
        Complex operator+(const Complex& rhs) const { return Complex(x + rhs.x, y
+ rhs.y); }
        Complex operator-(const Complex& rhs) const { return Complex(x - rhs.x, y
- rhs.y); }
        Complex operator*(const Complex& rhs) const {
            return Complex(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x);
        }
    };
    // 位逆序置换（迭代FFT核心）
    static void bitReverse(vector<Complex>& a) {
        int n = a.size();
        for (int i = 1, j = n / 2; i < n - 1; i++){
            if (i < j) swap(a[i], a[j]);
            int k = n / 2;
            while (j >= k){
                j -= k;
                k /= 2;
            }
            if (j < k) j += k;
        }
    }

    // 迭代FFT
    static void fft(vector<Complex>& a, int opt) {
        int n = a.size();
        bitReverse(a);
        for (int h = 2; h <= n; h <<= 1) {
            Complex wn(cos(2 * PI / h), opt * sin(2 * PI / h));
            for (int j = 0; j < n; j += h) {
                Complex w(1, 0);
                for (int k = j; k < j + h / 2; k++) {
                    Complex u = a[k], t = w * a[k + h / 2];
                    a[k] = u + t;
                    a[k + h / 2] = u - t;
                    w = w * wn;
                }
            }
        }
        if (opt == -1) {
            for (auto& x : a) x.x /= n;
        }
    }

public:
    // 计算多项式乘法 a * b，返回系数向量（从低次到高次）
    static vector<int> multiply(const vector<int>& a, const vector<int>& b) {
        int n = a.size(), m = b.size();
        int len = 1;
        while (len < n + m) len <<= 1;

        vector<Complex> fa(len), fb(len);
        for (int i = 0; i < n; i++) fa[i] = Complex(a[i], 0);
        for (int i = 0; i < m; i++) fb[i] = Complex(b[i], 0);

        fft(fa, 1);  // 正变换
        fft(fb, 1);
        for (int i = 0; i < len; i++) fa[i] = fa[i] * fb[i];
        fft(fa, -1); // 逆变换

        vector<int> res(n + m - 1);
        for (int i = 0; i < n + m - 1; i++)
```

```
            res[i] = round(fa[i].x);
        return res;
    }
};

const double Polynomial::PI = acos(-1.0);

/* 使用示例 */
int main() {
    vector<int> a = {3, 2, 1}; // 多项式 3 + 2x + 1x^2
    vector<int> b = {4, 3};     // 多项式 4 + 3x
    vector<int> c = Polynomial::multiply(a, b);

    for (int coeff : c) cout << coeff << " "; // 输出: 12 17 10 3
    return 0;
}
```

FTT (返回数组${a_i = sum_{j = 0}^{n - i - 1} b_j * c_{i+j}}$)

```
#include <iostream>
#include <complex>
#include <vector>
#include <cmath>
using namespace std;
typedef complex<double> Complex;
const double PI = acos(-1.0);

class FFT_Convolution {
private:
    // 位逆序置换（非递归FFT关键步骤）
    static void bitReverse(vector<Complex>& a) {
        int n = a.size();
        for (int i = 1, j = 0; i < n; i++) {
            int bit = n >> 1;
            for (; j >= bit; bit >>= 1) j -= bit;
            j += bit;
            if (i < j) swap(a[i], a[j]);
        }
    }

    // 迭代FFT核心
    static void fft(vector<Complex>& a, bool invert) {
        int n = a.size();
        bitReverse(a);  // 预处理位逆序

        for (int len = 2; len <= n; len <<= 1) {
            double angle = 2 * PI / len * (invert ? -1 : 1);
            Complex wn(cos(angle), sin(angle));
```

```cpp
            for (int i = 0; i < n; i += len) {
                Complex w(1);
                for (int j = 0; j < len / 2; j++) {
                    Complex u = a[i + j];
                    Complex v = w * a[i + j + len / 2];
                    a[i + j] = u + v;
                    a[i + j + len / 2] = u - v;
                    w *= wn;
                }
            }
        }

        if (invert) {
            for (Complex& x : a) x /= n;
        }
    }

public:
    // 计算卷积结果，返回数组a_i = sum_{j} b_j * c_{i+j}
    static vector<int> compute(const vector<int>& b, const vector<int>& c) {
        int n = 1;
        while (n < b.size() + c.size()) n <<= 1;

        vector<Complex> fb(n), fc(n);
        // 反转b并补零
        for (int i = 0; i < b.size(); i++) fb[i] = b[b.size() - 1 - i];
        for (int i = 0; i < c.size(); i++) fc[i] = c[i];

        fft(fb, false);
        fft(fc, false);
        for (int i = 0; i < n; i++) fb[i] *= fc[i];
        fft(fb, true);

        vector<int> result;
        for (int i = b.size() - 1; i < (int)c.size() + (int)b.size() - 1; i++)
            result.push_back(round(fb[i].real()));
        return result;
    }
};

/* 使用示例 */
int main() {
    vector<int> b = {1, 2, 3};
    vector<int> c = {4, 5, 6, 7};
    vector<int> a = FFT_Convolution::compute(b, c);
    for (int val : a) cout << val << " ";  // 输出: 32 38 20 7
    return 0;
}
```

NTT（标准版）

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN = 1 << 20;  // 支持长度1e6级别的多项式
const int MOD = 998244353; // 常用模数
const int G = 3;           // 原根
const int Gi = 332748118;  // 逆原根: G^(MOD-2) % MOD

class NTT {
private:
    // 快速幂取模
    ll qpow(ll a, ll b) {
        ll res = 1;
        while (b) {
            if (b & 1) res = res * a % MOD;
            a = a * a % MOD;
            b >>= 1;
        }
        return res;
    }

    // NTT核心变换
    void ntt(vector<ll>& a, int lim, int type) {
        // 蝴蝶变换
        for (int i = 0; i < lim; ++i)
            if (i < rev[i]) swap(a[i], a[rev[i]]);

        for (int mid = 1; mid < lim; mid <<= 1) {
            ll wn = qpow(type == 1 ? G : Gi, (MOD - 1) / (mid << 1));
            for (int j = 0; j < lim; j += (mid << 1)) {
                ll w = 1;
                for (int k = 0; k < mid; ++k, w = w * wn % MOD) {
                    ll x = a[j + k], y = w * a[j + k + mid] % MOD;
                    a[j + k] = (x + y) % MOD;
                    a[j + k + mid] = (x - y + MOD) % MOD;
                }
            }
        }
    }

public:
    vector<int> rev;

    // 计算多项式乘法
    vector<ll> multiply(const vector<ll>& a, const vector<ll>& b) {
        int n = a.size(), m = b.size();
        int lim = 1, l = 0;
        while (lim <= n + m) lim <<= 1, l++;
        rev.resize(lim);

        // 初始化蝴蝶变换索引
        for (int i = 0; i < lim; ++i)
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (l - 1));
        vector<ll> fa(lim), fb(lim);
        copy(a.begin(), a.end(), fa.begin());
        copy(b.begin(), b.end(), fb.begin());

        ntt(fa, lim, 1);  // 正变换
        ntt(fb, lim, 1);
        for (int i = 0; i < lim; ++i)
            fa[i] = fa[i] * fb[i] % MOD;
        ntt(fa, lim, -1); // 逆变换

        // 归一化并截取有效长度
        ll inv_lim = qpow(lim, MOD - 2);
        vector<ll> res(n + m - 1);
        for (int i = 0; i < n + m - 1; ++i)
            res[i] = fa[i] * inv_lim % MOD;
        return res;
    }
};

/* 使用示例 */
int main() {
    vector<ll> a = {1, 2, 3};       // 多项式1的系数
    vector<ll> b = {4, 5, 6, 7};    // 多项式2的系数
    NTT ntt_solver;
    vector<ll> c = ntt_solver.multiply(a, b);

    for (ll x : c) cout << x << " "; // 输出卷积结果: 4 13 28 34 32 21
    return 0;
}
```

NTT (返回数组${a_i = sum_{j = 0}^{n - i - 1} b_j * c_{i+j}}$)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN = 1 << 20;  // 支持长度1e6级别的多项式
const int MOD = 998244353; // 常用模数
const int G = 3;           // 原根
const int Gi = 332748118;  // 逆原根: G^(MOD-2) % MOD

class NTT_Convolution {
private:
    // 快速幂取模
    static ll qpow(ll a, ll b) {
        ll res = 1;
        while (b) {
            if (b & 1) res = res * a % MOD;
            a = a * a % MOD;
            b >>= 1;
        }
        return res;
    }

    // 位逆序置换
    static void bitReverse(vector<ll>& a, const vector<int>& rev) {
        int n = a.size();
        for (int i = 0; i < n; i++)
            if (i < rev[i]) swap(a[i], a[rev[i]]);
    }

    // NTT核心变换
    static void ntt(vector<ll>& a, int lim, int type, const vector<int>& rev) {
        bitReverse(a, rev);
        for (int mid = 1; mid < lim; mid <<= 1) {
            ll wn = qpow(type == 1 ? G : Gi, (MOD - 1) / (mid << 1));
            for (int j = 0; j < lim; j += (mid << 1)) {
                ll w = 1;
                for (int k = 0; k < mid; k++, w = w * wn % MOD) {
                    ll x = a[j + k], y = w * a[j + k + mid] % MOD;
                    a[j + k] = (x + y) % MOD;
                    a[j + k + mid] = (x - y + MOD) % MOD;
                }
            }
        }
        if (type == -1) {
            ll inv_lim = qpow(lim, MOD - 2);
            for (int i = 0; i < lim; i++)
                a[i] = a[i] * inv_lim % MOD;
        }
    }

public:
    // 计算卷积结果 a_i = sum_j b_j * c_{i+j}
    static vector<ll> compute(const vector<ll>& b, const vector<ll>& c) {
        int n = b.size(), m = c.size();
        int lim = 1, l = 0;
        while (lim < n + m) lim <<= 1, l++;

        vector<int> rev(lim);
        for (int i = 0; i < lim; i++)
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (l - 1));

        vector<ll> fb(lim), fc(lim);
        // 反转数组b并补零
        for (int i = 0; i < n; i++) fb[i] = b[n - 1 - i];
        for (int i = 0; i < m; i++) fc[i] = c[i];

        ntt(fb, lim, 1, rev);  // 正变换
        ntt(fc, lim, 1, rev);
        for (int i = 0; i < lim; i++) fb[i] = fb[i] * fc[i] % MOD;
        ntt(fb, lim, -1, rev); // 逆变换
        // 截取有效部分（a_i对应卷积结果的第n-1到n+m-2项）
        vector<ll> res(m);
        for (int i = 0; i < m; i++)
            res[i] = fb[n - 1 + i];
        return res;
    }
};

/* 使用示例 */
int main() {
    vector<ll> b = {1, 2, 3};       // 数组b
    vector<ll> c = {4, 5, 6, 7};    // 数组c
    vector<ll> a = NTT_Convolution::compute(b, c);

    for (ll x : a) cout << x << " ";  // 输出：32 38 20 7
    return 0;
}
```

NTT

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

template <int T>
struct MInt {
    int x;
    MInt() : x(0) {}
    MInt(int y) : x(y % T) { if (x < 0) x += T; }
    MInt operator+(const MInt &y) const { return MInt((x + y.x) % T); }
    MInt operator-(const MInt &y) const { return MInt((x - y.x + T) % T); }
    MInt operator*(const MInt &y) const { return MInt(1LL * x * y.x % T); }
    MInt& operator+=(const MInt &y) { x = (x + y.x) % T; return *this; }
    MInt& operator-=(const MInt &y) { x = (x - y.x + T) % T; return *this; }
    MInt& operator*=(const MInt &y) { x = 1LL * x * y.x % T; return *this; }
    bool operator==(const MInt &y) const { return x == y.x; }
    bool operator!=(const MInt &y) const { return x != y.x; }
    friend ostream& operator<<(ostream &os, const MInt &m) { return os << m.x; }
};
constexpr int mod=1e9+7;

using Z=MInt<mod>;
// mod  一定要满足 mod = len * k + 1; len 为 2 的次幂长度  一定要注意模数的选取！！！
// ntt 从 0 - n 的位置依次存放权值从小到大的
struct Polynomial
{
    vector<Z> z;
    vector<int> r;
    Polynomial(vector<int> &a)
    {
        int n = a.size();
        z.resize(n);
        r.resize(n);
        for (int i = 0; i < n; i++)
        {
            z[i] = a[i];
            r[i] = (i & 1) * (n / 2) + r[i / 2] / 2;
        }
        ntt(z, n, 1);
    }
    LL power(LL a, int b)
    {
        LL res = 1;
        for (; b; b /= 2, a = a * a % mod)
        {
            if (b % 2)
            {
                res = res * a % mod;
            }
        }
        return res;
    }
    void ntt(vector<Z> &a, int n, int opt)
    {
        for (int i = 0; i < n; i++)
        {
            if (r[i] < i)
            {
                swap(a[i], a[r[i]]);
            }
        }
        for (int k = 2; k <= n; k *= 2)
        {
            Z gn = power(3, (mod - 1) / k);
            for (int i = 0; i < n; i += k)
            {
                Z g = 1;
                for (int j = 0; j < k / 2; j++, g *= gn)
                {
                    Z t = a[i + j + k / 2] * g;
                    a[i + j + k / 2] = a[i + j] - t;
                    a[i + j] = a[i + j] + t;
                }
            }
        }
        if (opt == -1)
        {
            reverse(a.begin() + 1, a.end());
            Z inv = power(n, mod - 2);
            for (int i = 0; i < n; i++)
            {
                a[i] *= inv;
            }
        }
    }
};
/*输入:
5
0 1
10 10
10 11
101 101
110 110
*/

/*输出: 进制为 sqrt(-2)时
0
100
110
1
10110100
*/

void solve(){
    string s1, s2;
    cin >> s1 >> s2;
    int n = s1.size(), m = s2.size();
    int len = 1;
    while (len < n * 2 || len < m * 2) len <<= 1;
    vector<int> a(n), b(m);
    for (int i = 0; i < n; i++){
        a[i] = s1[i] - '0';
    }
    for (int i = 0; i < m; i++){
        b[i] = s2[i] - '0';
    }
    reverse(a.begin(), a.end());
    reverse(b.begin(), b.end());
    a.resize(len);
    b.resize(len);
    Polynomial f1(a), f2(b);
    for (int i = 0; i < len; i++){
        f1.z[i] = f1.z[i] * f2.z[i];
    }
    f1.ntt(f1.z, len, -1);
    len = n + m - 1;
    vector<int> ans(len + 100);
    for (int i = 0; i < len; i++){
        // cout << f1.z[i].x << " ";
        ans[i] = f1.z[i].x;
    }
    // cout << endl;
    for (int i = 0; i <= len; i++){
        // cout << ans[i] << " ";
        if (i == ans.size()) break;
        if (ans[i] > 1){
            ans[i + 2] -= ans[i] / 2;
```

```
                ans[i] %= 2;
                len = max(len, i + 2);
            }
            if(ans[i] < 0){
                ans[i + 2] += -ans[i] / 2;
                ans[i] %= 2;
                if (ans[i] < 0){
                    ans[i] = 1;
                    ans[i + 2]++;
                }
                len = max(len, i + 2);
            }
        }
        // cout << endl;
        while (ans[len] == 0) len--;
        len = max(len, 0);
        for (int i = len; i >= 0; i--) cout << ans[i];
        cout << endl;
    }
    int main()
    {
        ios::sync_with_stdio(false);
        cin.tie(0), cout.tie(0);
        int t = 1;
        cin >> t;
        while (t--) solve();
        return 0;
    }
```

高斯消元

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

const int N = 110;
const double eps = 1e-8;
LL n;
vector<vector<double>> a(N , vector<double>(N));
LL gauss(){
    LL c, r;
    for (c = 0, r = 0; c < n; c++){
        LL 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 j = c; j < n + 1; j++){
            swap(a[t][j], a[r][j]);
        }
        for (int j = n; j >= c; j--) a[r][j] /= a[r][c];
        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[r][j] * a[i][c];
                }
            }
        }
        r++;
    }
    if (r < n){
        for (int i = r; i < n; i++){
            if (fabs(a[i][n]) > eps) 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;
}
/* 最后第n列存储等式结果， 前0 - (n - 1) 列为未知量系数
a11 * x1 + a12 * x2 + ... + a1n * xn = k1
a21 * x1 + ... = k2
*/
int main(){
    cin >> n;
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n + 1; j++){
            cin >> a[i][j];
        }
    }
    LL t = gauss();
    if (t == 0){
        for (int i = 0; i < n; i++){
            if (fabs(a[i][n]) < eps) a[i][n] = abs(a[i][n]);
            cout << a[i][n] << endl;
        }
    }
    else if (t == 1) cout << "多个解" << endl;
    else cout << "无解" << endl;
}
```

Min25筛

```cpp
#include <bits/stdc++.h>
#define ll long long
using namespace std;
namespace min25{
    // 求解 1 - n 的质数的和, n 最大为1e10
    const int N = 1e6 + 10;
    int prime[N], id1[N], id2[N], flag[N], ncnt, m;
    ll g[N], sum[N], a[N], T;
    ll n, mod;
    inline ll ps(ll n, ll k){
        ll r = 1;
        for (; k; k >>= 1){
            if (k & 1){
                r = r * n % mod;
            }
            n = n * n % mod;
        }
        return r;
    }
    void finit(){
        // 最开始清 0
        memset(g, 0, sizeof(g));
        memset(a, 0, sizeof(a));
        memset(sum, 0, sizeof(sum));
        memset(prime, 0, sizeof(prime));
        memset(id1, 0, sizeof(id1));
        memset(id2, 0, sizeof(id2));
        memset(flag, 0, sizeof(flag));
        ncnt = m = 0;
    }
    int ID(ll x){
        return x <= T ? id1[x] : id2[n / x];
    }
    ll calc(ll x){
        return x * (x + 1) / 2 - 1;
    }
    ll init(ll x){
        T = sqrt(x + 0.5);
        for (int i = 2; i <= T; i++){
            if (!flag[i]) prime[++ncnt] = i, sum[ncnt] = sum[ncnt - 1] + i;
            for (int j = 1; j <= ncnt && i * prime[j] <= T; j++){
                flag[i * prime[j]] = 1;
                if (i % prime[j] == 0) break;
            }
        }
        for (ll L = 1; L <= x; L = x / (x / L) + 1){
            a[++m] = x / L;
            if (a[m] <= T) id1[a[m]] = m;
            else id2[x / a[m]] = m;
            g[m] = calc(a[m]);
        }
        for (int i = 1; i <= ncnt; i++){
            for (int j = 1; j <= m && (ll) prime[i] * prime[i] <= a[j]; j++){
                g[j] = g[j] - (ll) prime[i] * (g[ID(a[j] / prime[i])] - sum[i - 
1]);
            }
        }
    }
    ll solve(ll x){
        if (x <= 1) return x;
        return n = x, init(n), g[ID(n)];
    }
}

using namespace min25;

int main(){
    int tt;
    cin >> tt;
    while (tt--){
        finit();
        cin >> n >> mod;
        // cout << n << " " << mod << endl;
        // ll ans = (n + 3) % mod * n % mod * ps(2, mod - 2) % mod + solve(n + 1) 
- 4;
        // ans = (ans + mod) % mod;
        cout << solve(n) << endl;
    }
}
```

欧拉函数求解

```
#include <bits/stdc++.h>
using namespace std;
int phik(int n){
    // 求解 n 的欧拉函数值
    int ans = n;
    for (int i = 2; 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;
}

const int N = 1e5 + 7;
int v[N], prime[N], phi[N];
void euler(int n){
    // 求解 1 - n 区间内每个数的欧拉函数值
    int m = 0;
    for (int i = 2; i <= n; i++){
        if (v[i] == 0){
            v[i] = i;
            prime[++m] = i;
            phi[i] = i - 1;
        }
        for (int j = 1; j <= m; j++){
            if (prime[j] > v[i] || prime[j] > n / i) break;
            v[i * prime[j]] = prime[j];
            phi[i * prime[j]] = phi[i] * (i % prime[j] ? prime[j] - 1 : prime[j]);
        }
    }
}
```

欧拉降幂

f(x) 为欧拉函数

若 m 为质数，则gcd(a, m) = 1 : ${a^x % m = a^{x % f(m)} % m}$

若 gcd(a, m) != 1 时：

$$ a^x % m = \begin{cases} a^x % m & x < f(m) \ a^{x % f(m) + f(m)} % m & x >= f(m) \end{cases} $$

```
#include <iostream>
#include <algorithm>
#include <math.h>
using namespace std;
typedef long long ll;

// 计算欧拉函数φ(m) [质因数分解法]
ll euler_phi(ll m) {
    ll res = m;
    for (ll i = 2; i * i <= m; i++) {
        if (m % i == 0) {
            res = res / i * (i - 1);  // φ(m) = m * Π(1 - 1/p)
            while (m % i == 0) m /= i;
        }
    }
    if (m > 1) res = res / m * (m - 1);  // 处理剩余的质因数
    return res;
}

// 快速幂取模
ll qpow(ll a, ll b, ll mod) {
    ll res = 1;
    a %= mod;  // 防止a过大
    while (b) {
        if (b & 1) res = res * a % mod;
        a = a * a % mod;
        b >>= 1;
    }
    return res;
}

// 欧拉降幂核心函数：计算 a^b mod m（b为整数）
ll euler_pow(ll a, ll b, ll m) {
    if (m == 1) return 0;  // 任何数模1为0
    ll phi_m = euler_phi(m);
    // 扩展欧拉定理：若a与m不互质且b≥ϕ(m)，则指数需加ϕ(m)
    if (b >= phi_m && __gcd(a, m) != 1) {
        b = b % phi_m + phi_m;
    } else {
        b %= phi_m;  // 否则直接取模
    }
    return qpow(a, b, m);
}

/* 使用示例 */
int main() {
    ll a = 2, b = 1000, m = 10007;
    cout << euler_pow(a, b, m) << endl;  // 输出 2^1000 mod 10007
    return 0;
}
```

卢卡斯定理 (mod 必须为质数)

```
//P3807
#include<bits/stdc++.h>
using namespace std;
const int N=100010;
typedef long long ll;
// mod 为质数时， C(n, m) % mod = C(n / mod, m / mod) * C(n % mod, m % mod) % mod
// 当 mod 不为质数时使用扩展卢卡斯定理

// 快速幂计算 a^b mod p
ll qpow(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;
}

// 预处理阶乘和逆元（需保证 p 是质数）
vector<ll> fac, inv_fac;
void init(int p) {
    fac.resize(p + 1);
    inv_fac.resize(p + 1);
    fac[0] = 1;
    for (int i = 1; i <= p; i++)
        fac[i] = fac[i - 1] * i % p;
    inv_fac[p - 1] = qpow(fac[p - 1], p - 2, p); // 费马小定理求逆元
    for (int i = p - 2; i >= 0; i--)
        inv_fac[i] = inv_fac[i + 1] * (i + 1) % p;
}

// 计算组合数 C(n, m) mod p（需 n, m < p）
ll C(ll n, ll m, ll p) {
    if (m > n) return 0;
    return fac[n] * inv_fac[m] % p * inv_fac[n - m] % p;
}

// 卢卡斯定理递归实现
ll lucas(ll n, ll m, ll p) {
    if (m == 0) return 1;
    return C(n % p, m % p, p) * lucas(n / p, m / p, p) % p;
}

void solve1() {
    int T;
    cin >> T;
    while (T--) {
        ll n, m, p;
        cin >> n >> m >> p;
        init(p); // 预处理阶乘和逆元
        cout << lucas(n, m, p) << endl; // 示例：计算 C(n+m, m) mod p
    }
}
int main(){

}
```

扩展卢卡斯定理

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

// 快速幂计算 a^b mod p
ll qpow(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;
}
```

```
// 扩展欧几里得求逆元
void exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) { x = 1; y = 0; return; }
    exgcd(b, a % b, y, x);
    y -= a / b * x;
}

// 求逆元（适用于模数非质数）
ll inv(ll a, ll p) {
    ll x, y;
    exgcd(a, p, x, y);
    return (x % p + p) % p;
}

// 计算 n! 中除去质因子 p 后的部分 mod pk
ll fac(ll n, ll p, ll pk) {
    if (!n) return 1;
    ll ans = 1;
    // 循环节部分（每 pk 个数模 pk 结果相同）
    for (ll i = 1; i <= pk; i++)
        if (i % p) ans = ans * i % pk;
    ans = qpow(ans, n / pk, pk);
    // 剩余部分
    for (ll i = n / pk * pk + 1; i <= n; i++)
        if (i % p) ans = ans * (i % pk) % pk;
    return ans * fac(n / p, p, pk) % pk; // 递归处理 p 的倍数
}

// 计算 C(n, m) mod p^k（p 是质数）
ll C_pk(ll n, ll m, ll p, ll pk) {
    if (m > n) return 0;
    // 计算质因子 p 的指数
    ll cnt = 0;
    for (ll i = n; i; i /= p) cnt += i / p;
    for (ll i = m; i; i /= p) cnt -= i / p;
    for (ll i = n - m; i; i /= p) cnt -= i / p;
    // 计算组合数模 p^k
    ll a = fac(n, p, pk), b = fac(m, p, pk), c = fac(n - m, p, pk);
    return qpow(p, cnt, pk) * a % pk * inv(b, pk) % pk * inv(c, pk) % pk;
}

// 中国剩余定理合并同余方程
ll CRT(const vector<ll> &a, const vector<ll> &m, ll p) {
    ll ans = 0, M = 1;
    for (ll mi : m) M *= mi;
    for (int i = 0; i < a.size(); i++) {
        ll Mi = M / m[i];
        ans = (ans + a[i] * Mi % p * inv(Mi, m[i]) % p) % p;
    }
    return ans;
}

// 扩展卢卡斯定理主函数
```

```cpp
ll exLucas(ll n, ll m, ll p) {
    vector<ll> a, m_factors;
    ll tmp = p;
    // 质因数分解 p
    for (ll i = 2; i * i <= tmp; i++) {
        if (tmp % i == 0) {
            ll pk = 1;
            while (tmp % i == 0) tmp /= i, pk *= i;
            m_factors.push_back(pk);
            a.push_back(C_pk(n, m, i, pk));
        }
    }
    if (tmp > 1) {
        m_factors.push_back(tmp);
        a.push_back(C_pk(n, m, tmp, tmp));
    }
    // 中国剩余定理合并
    return CRT(a, m_factors, p);
}

int main() {
    ll n, m, p;
    cin >> n >> m >> p;
    cout << exLucas(n, m, p) << endl;
    return 0;
}
```