快的一b多项式NTT全家桶

const int N = 1 << 18, pG = 3, P = 998244353; // N = 2 ^ k

using ll = int64\_t;

#define inc(a, b) (((a) += (b)) >= P ? (a) -= P : 0)

#define dec(a, b) (((a) -= (b)) < 0 ? (a) += P : 0)

#define mul(a, b) (ll(a) \* (b) % P)

int fpow(ll a, int b = P - 2, ll x = 1) {

for (; b; b >>= 1, a = a \* a % P)

if (b & 1)

x = x \* a % P;

return x;

}

int inv[N], fac[N], ifac[N], W[N], \_ = [] {

fac[0] = fac[1] = ifac[0] = ifac[1] = inv[1] = 1;

for (ll i = 2; i < N; ++i) {

fac[i] = fac[i - 1] \* i % P;

inv[i] = (P - P / i) \* inv[P % i] % P;

ifac[i] = (ll)ifac[i - 1] \* inv[i] % P;

}

W[N / 2] = 1;

for (int i = N / 2 + 1, wn = fpow(pG, P / N); i < N; ++i)

W[i] = mul(W[i - 1], wn);

for (int i = N / 2 - 1; ~i; --i)

W[i] = W[i << 1];

return 0;

}();

namespace NTT {

void dft(int \*a, int n) {

for (int k = n >> 1; k; k >>= 1)

for (int i = 0; i < n; i += k << 1)

for (int j = 0; j < k; ++j) {

int &x = a[i + j], y = a[i + j + k];

a[i + j + k] = mul(x - y + P, W[k + j]);

inc(x, y);

}

}

void idft(int \*a, int n) {

for (int k = 1; k < n; k <<= 1)

for (int i = 0; i < n; i += k << 1)

for (int j = 0; j < k; ++j) {

int &x = a[i + j], y = mul(a[i + j + k], W[k + j]);

a[i + j + k] = x < y ? x - y + P : x - y;

inc(x, y);

}

for (int i = 0, in = P - (P - 1) / n; i < n; ++i)

a[i] = mul(a[i], in);

reverse(a + 1, a + n);

}

} // namespace NTT

int norm(int n) { return 1 << (\_\_lg(n - 1) + 1); }

struct Poly : public vector<int> {

#define T (\*this)

using vector<int>::vector;

int deg() const { return size(); }

Poly rev() const { return Poly(rbegin(), rend()); }

void append(const Poly &a) { insert(end(), a.begin(), a.end()); }

Poly operator-() const {

Poly a(T);

for (auto &x : a)

x = x ? P - x : 0;

return a;

}

Poly &operator+=(const Poly &a) {

if (a.deg() > deg())

resize(a.deg());

for (int i = 0; i < a.deg(); ++i)

inc(T[i], a[i]);

return T;

}

Poly &operator-=(const Poly &a) {

if (a.deg() > deg())

resize(a.deg());

for (int i = 0; i < a.deg(); ++i)

dec(T[i], a[i]);

return T;

}

Poly &operator^=(const Poly &a) {

if (a.deg() < deg())

resize(a.deg());

for (int i = 0; i < deg(); ++i)

T[i] = mul(T[i], a[i]);

return T;

}

Poly &operator\*=(int a) {

for (auto &x : T)

x = mul(x, a);

return T;

}

Poly operator+(const Poly &a) const { return Poly(T) += a; }

Poly operator-(const Poly &a) const { return Poly(T) -= a; }

Poly operator^(const Poly &a) const { return Poly(T) ^= a; }

Poly operator\*(int a) const { return Poly(T) \*= a; }

Poly &operator<<=(int k) { return insert(begin(), k, 0), T; }

Poly operator<<(int r) const { return Poly(T) <<= r; }

Poly operator>>(int r) const {

return r >= deg() ? Poly() : Poly(begin() + r, end());

}

Poly &operator>>=(int r) { return T = T >> r; }

Poly pre(int k) const { return k < deg() ? Poly(begin(), begin() + k) : T; }

friend void dft(Poly &a) { NTT::dft(a.data(), a.size()); }

friend void idft(Poly &a) { NTT::idft(a.data(), a.size()); }

friend Poly conv(Poly a, Poly b, int n) {

a.resize(n), dft(a);

b.resize(n), dft(b);

return idft(a ^= b), a;

}

Poly operator\*(const Poly &a) const {

int n = deg() + a.deg() - 1;

return conv(T, a, norm(n)).pre(n);

}

// Description: f\_i = sum T\_{i + j} \* a\_j



// 卷积 相差多少就加进abs的贡献里

Poly mulT(const Poly &a) const { return T \* a.rev() >> (a.deg() - 1); }

/\*

\* Description: Calculate f\_1, ..., f\_{n - 1} with a\_i = sum\_{j > 0} f\_{i - j}

\* g\_j and f\_i = F(a\_i, i),where g and f\_0 is known, notice T\_i = g\_{i + 1}.

\* Time: O(n log^2 n)

\*/

template <typename func> Poly semiConv(Poly f, int n, func calc) {

f.resize(n);

vector<Poly> va(n); // storage dft result, faster

for (int m = 1; m < n; m++) {

int k = m & -m, l = m - k, r = min(m + k, n);

Poly &p = va[k], q(f.begin() + l, f.begin() + m);

if (p.empty())

p = pre(r - l - 1), p.resize(k \* 2), dft(p);

q.resize(k << 1), dft(q), idft(q ^= p);

// Poly q = conv(Poly(f.begin() + l, f.begin() + m), pre(r - l - 1), k <<

// 1);

for (int i = m; i < r; i++)

inc(f[i], q[i - l - 1]);

calc(f, m);

}

return f;

}

Poly inv(int n) const { // 逆

Poly x{fpow(T[0])};

for (int k = 1; k < n; k <<= 1)

x.append(-((conv(pre(k << 1), x, k << 1) >> k) \* x).pre(k));

return x.pre(n);

}

Poly inv2(int n) const { // 逆1

int i0 = fpow(T[0]);

return (T >> 1).semiConv({i0}, n,

[&](Poly &f, int m) { f[m] = mul(f[m], P - i0); });

}

Poly deriv() const { // 求导

if (empty())

return {};

Poly a(deg() - 1);

for (ll i = 1; i < deg(); ++i)

a[i - 1] = i \* T[i] % P;

return a;

}

Poly integ() const { // 积分

if (empty())

return {};

Poly a(deg() + 1);

for (int i = 1; i <= deg(); ++i)

a[i] = mul(::inv[i], T[i - 1]);

return a;

}

// 需要保证首项为 1

Poly log(int n) const { return (deriv() \* inv(n)).integ().pre(n); }

// 需要保证首项为 0

Poly exp(int n) const {

Poly x{1};

for (int k = 1; k < n; k <<= 1)

x.append((x \* ((pre(k << 1) - x.log(k << 1)) >> k)).pre(k));

return x.pre(n);

}

// 需要保证首项为 0

Poly exp2(int n) const {

return deriv().semiConv(

{1}, n, [&](Poly &f, int m) { f[m] = mul(f[m], ::inv[m]); });

}

Poly pow(int k, int n) const { return (log(n) \* k).exp(n); } // T[0] = 1

Poly pow(int k, int kp, int n) const { // k = K % P, kp = K % phi(P)

int i = 0;

while (i < deg() && !T[i])

i++;

if (1ll \* i \* k >= n)

return Poly(n);

int v = T[i], m = n - i \* k;

return ((((T >> i) \* fpow(v)).log(m) \* k).exp(m) << (i \* k)) \* fpow(v, kp);

}

// 需要保证首项为 1，开任意次方可以先 ln 再 exp 实现。

Poly sqrt(int n) const {

Poly x{1}, y{1}; // x[0] = sqrt(T[0]), default T[0] = 1

for (int k = 1; k < n; k <<= 1) {

x.append((((pre(k << 1) - x \* x) >> k) \* y).pre(k) \* ((P + 1) >> 1));

k << 1 < n ? y.append(-((conv(x.pre(k << 1), y, k << 1) >> k) \* y).pre(k))

: void();

}

return x.pre(n);

}

vector<Poly> operator/(const Poly &a) const {

int k = deg() - a.deg() + 1;

if (k < 0)

return {{0}, T};

Poly q = (rev().pre(k) \* (a.rev().inv(k))).pre(k).rev(), r = T - a \* q;

return {q, r.pre(a.deg() - 1)};

}

/\*

\* Description: calculate [x ^ k](f / g)

\* Time: O(n log n log k)

\*/

int divAt(Poly f, Poly g, ll k) {

int n = max(f.deg(), g.deg()), m = norm(n);

for (; k; k >>= 1) {

f.resize(m \* 2), dft(f);

g.resize(m \* 2), dft(g);

for (int i = 0; i < 2 \* m; ++i)

f[i] = mul(f[i], g[i ^ 1]);

for (int i = 0; i < m; ++i)

g[i] = mul(g[2 \* i], g[2 \* i + 1]);

g.resize(m), idft(f), idft(g);

for (int i = 0, j = k & 1; i < n; i++, j += 2)

f[i] = f[j];

f.resize(n), g.resize(n);

}

return f[0];

}

/\*

\* Description: calculate T[k], where T[n] = sum c[i] \* T[n - i]

图示, 文本, 示意图

描述已自动生成

\* Time: O(n log n log k)

\*/

int recur(Poly c, ll k) {

return c[0] = P - 1, divAt((T \* c).pre(c.deg() - 1), c, k);

}

/\*

\* Description: polynomial multipoint fast evaluation

\* Time: O(n log^2 n)

\*/

Poly eval(Poly x) const {

if (empty())

return Poly(x.deg());

const int m = x.deg(), n = norm(m);

vector<Poly> q(2 \* n, {1});

Poly ans(m), temp, d(2 \* n);

for (int i = n; i < n + m; ++i)

q[i] = Poly{1, P - x[i - n]};

for (int i = n - 1; i; --i)

q[i] = q[i << 1] \* q[i << 1 | 1];

q[1] = mulT(q[1].inv(n));

for (int i = 1, l = 2, r = 3; i < n; ++i, l += 2, r += 2) {

temp = q[l], d[l] = d[r] = d[i] + 1;

q[l] = q[i].mulT(q[r]).pre(n >> d[l]);

q[r] = q[i].mulT(temp).pre(n >> d[r]);

}

for (int i = n; i < n + m; ++i)

ans[i - n] = q[i][0];

return ans;

}

#undef T

};

// useless algorithm

/\*

\* Description: G(F(x))

\* Time: O(n sqrt n log n + n ^ 2)

\*/

Poly compound(Poly f, Poly g) {

int n = f.size(), k = norm(2 \* n), L = sqrt(n) + 1;

vector<Poly> G(L + 1);

Poly H, h(k), t; // H = g ^ (iL)

auto dft = [&](Poly &a) { a.resize(k), NTT::dft(a.data(), k); };

auto idft = [&](Poly &a) { NTT::idft(a.data(), k), a.resize(n); };

G[0] = H = {1}, dft(g);

for (int i = 1; i <= L; ++i)

dft(G[i] = G[i - 1]), idft(G[i] ^= g);

dft(g = G[L]);

for (int i = 0; i < L; ++i, t = {}, idft(H)) {

for (int j = 0; j < min(L, n - i \* L); ++j)

t += G[j] \* f[i \* L + j];

dft(t), dft(H);

for (int j = 0; j < k; ++j)

h[j] = (h[j] + (ll)t[j] \* H[j]) % P, H[j] = mul(H[j], g[j]);

}

return idft(h), h;

}

/\*

\* Description: get F(x) for F(G(x)) = x

\* Time: O(n sqrt n log n + n ^ 2)

\*/

Poly compoundInv(Poly g) { //

int n = g.size(), k = norm(2 \* n), L = sqrt(n) + 1;

vector<Poly> G(L + 1);

Poly H, f(n);

auto dft = [&](Poly &a) { a.resize(k), NTT::dft(a.data(), k); };

auto idft = [&](Poly &a) { NTT::idft(a.data(), k), a.resize(n); };

G[0] = H = {1}, H.resize(n), dft(g = (g >> 1).inv(n));

for (int i = 1; i <= L; ++i)

dft(G[i] = G[i - 1]), idft(G[i] ^= g);

dft(g = G[L]);

for (int i = 0, t = 1; i < L; ++i) {

for (int j = 1; j <= L && t < n; ++j, ++t)

for (int r = 0; r < t; ++r)

f[t] = (f[t] + (ll)G[j][r] \* H[t - 1 - r]) % P;

dft(H), idft(H ^= g);

}

return f ^ Poly(inv, inv + N);

}

// P + Q: O(n);

// P - Q: O(n);

// P \* Q: O(n^2) or O(n log n);

// P / Q, polynomial div: O(n log n);

// P % Q, polynomial mod: O(n log n);

// P.divmod(), polynomial divmod: O(n log n);

// P.inv(k) == 1 / P mod x^k: O(n log n);

// P.deriv(): O(n);

// P.integr(): O(n);

// P.exp(k) == exp(P) mod x^k: O(n log n);

// P.log(k) == log(P) mod x^k: O(n log n);

// P.pow(y, k) == P^y mod x^k: O(n log n);

FFT

using ll = long long ;

using db = double;

/\*---------------------------------------------------------------------------\*/

struct cp {

db x, y;

cp(db real = 0, db imag = 0) : x(real), y(imag){};

cp operator+(cp b) const { return {x + b.x, y + b.y}; }

cp operator-(cp b) const { return {x - b.x, y - b.y}; }

cp operator\*(cp b) const { return {x \* b.x - y \* b.y, x \* b.y + y \* b.x}; }

};

using vcp = vector<cp>;

using Poly = vector<ll>;

namespace FFT {

const db pi = acos(-1);

vcp Omega(int L) {

vcp w(L); w[1] = 1;

for (int i = 2; i < L; i <<= 1) {

auto w0 = w.begin() + i / 2, w1 = w.begin() + i;

cp wn(cos(pi / i), sin(pi / i));

for (int j = 0; j < i; j += 2)

w1[j] = w0[j >> 1], w1[j + 1] = w1[j] \* wn;

}

return w;

}

auto W = Omega(1 << 21); // NOLINT

void DIF(cp \*a, int n) {

cp x, y;

for (int k = n >> 1; k; k >>= 1)

for (int i = 0; i < n; i += k << 1)

for (int j = 0; j < k; ++j)

x = a[i + j], y = a[i + j + k],

a[i + j + k] = (a[i + j] - y) \* W[k + j], a[i + j] = x + y;

}

void IDIT(cp \*a, int n) {

cp x, y;

for (int k = 1; k < n; k <<= 1)

for (int i = 0; i < n; i += k << 1)

for (int j = 0; j < k; ++j)

x = a[i + j], y = a[i + j + k] \* W[k + j],

a[i + j + k] = x - y, a[i + j] = x + y;

const db Inv = 1. / n;

for(int i = 0; i <= n - 1; ++i) {

a[i].x \*= Inv, a[i].y \*= Inv;

}

reverse(a + 1, a + n);

}

}

namespace Polynomial {

// basic operator

void DFT(vcp &a) { FFT::DIF(a.data(), a.size()); }

void IDFT(vcp &a) { FFT::IDIT(a.data(), a.size()); }

int norm(int n) { return 1 << ((int)(log(n) / log(2))+ 1); }

// Poly mul

vcp &dot(vcp &a, vcp &b) {

for(int i = 0; i < a.size(); ++i) {

a[i] = a[i] \* b[i];

}

return a;

}

Poly operator+(Poly a, Poly b) {

int maxlen = max(a.size(), b.size());

Poly ans(maxlen + 1);

a.resize(maxlen + 1), b.resize(maxlen + 1);

for (int i = 0; i < maxlen;i++)

ans[i] = a[i] + b[i];

return ans;

}

Poly operator\*(ll k, Poly a) {

Poly ans;

for(auto i:a)

ans.push\_back(k \* i);

return ans;

}

Poly operator\*(Poly a, Poly b) {

int n = a.size() + b.size() - 1;

vcp c(norm(n));

int sz = a.size();

for(int i = 0; i <= sz - 1; ++i) {

c[i].x = a[i];

}

sz = b.size();

for(int i = 0; i <= sz - 1; ++i) {

c[i].y = b[i];

}

DFT(c), dot(c, c), IDFT(c), a.resize(n);

for(int i = 0; i <= n - 1; ++i) {

a[i] = int(c[i].y \* .5 + .5);

}

return a;

}

}

/\*---------------------------------------------------------------------------\*/

using namespace Polynomial;

扩展Lucas

using ll = long long;

namespace ExtendGcd {

ll extend\_gcd(ll a, ll b, ll& x, ll& y) {

ll d = a;

if (b != 0) {

d = extend\_gcd(b, a % b, y, x);

y -= (a / b) \* x;

}

else x = 1, y = 0;

return d;

}

}

namespace CRT {

ll quick\_mul(ll a, ll b, ll mod) {

ll res = 0;

while (a) {

if (a & 1)res = (res + b) % mod;

b = 2 \* b % mod;

a >>= 1;

}

return res;

}

ll inv(ll t, ll p) {//如果不存在，返回-1

ll x, y, d;

d = ExtendGcd::extend\_gcd(t, p, x, y);

return d == 1 ? (x % p + p) % p : -1;

}

ll gcd(ll a, ll b) {

return b ? gcd(b, a % b) : a;

}

ll lcm(ll a, ll b) {

return b / gcd(a, b) \* a;

}

//x = a(mod m)

ll solve(const vector<ll>& a, const vector<ll> m) {

ll M = 1, res = 0;

for (int i = 0; i < m.size(); ++i)M = lcm(M, m[i]);

for (int i = 0; i < a.size(); ++i) {

ll Mi = M / m[i];

ll ti = inv(Mi, m[i]);

res = (res + quick\_mul(quick\_mul(a[i], ti, M), Mi, M)) % M;

}

return res;

}

}

namespace ExLucas {

const int MAXN = 1e6;

vector<ll> A, M;

ll fac[MAXN + 5];

ll quick\_pow(ll base, ll index, ll p) {

ll res = 1;

while (index) {

if (index & 1)res = res \* base % p;

base = base \* base % p;

index >>= 1;

}

return res;

}

ll solve(ll n, ll p, ll pk) {

if (n == 0)return 1;

ll ans = quick\_pow(fac[pk - 1], n / pk, pk) \* fac[n % pk] % pk;

return ans \* solve(n / p, p, pk) % pk;

}

ll C(ll n, ll m, ll p, ll pk) {

if (n < m)return 0;

fac[0] = 1;

for (int i = 1; i < pk; ++i) {

fac[i] = fac[i - 1];

if (i % p != 0)fac[i] = fac[i] \* i % pk;

}

ll f1 = solve(n, p, pk), f2 = solve(m, p, pk), f3 = solve(n - m, p, pk), 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;

return f1 \* CRT::inv(f2, pk) % pk \* CRT::inv(f3, pk) % pk \* quick\_pow(p, cnt, pk) % pk;

}

ll exlucas(ll n, ll m, ll p) {

A.clear(), M.clear();

ll block = sqrt(p);

for (int i = 2; i <= block && p > 1; ++i) {

ll u = 1;

while (p % i == 0) {

p /= i;

u \*= i;

}

if (u > 1) {

A.push\_back(C(n, m, i, u));

M.push\_back(u);

}

}

if (p > 1) {

A.push\_back(C(n, m, p, p));

M.push\_back(p);

}

return CRT::solve(A, M);

}

}

线性基

using ll = long long;

struct L\_B {

long long d[61], p[61];

int cnt;

L\_B() {

memset(d, 0, sizeof(d));

memset(p, 0, sizeof(p));

cnt = 0;

}

bool insert(ll val) {//插入

for (int i = 60; i >= 0; i--)

if (val & (1LL << i)) {

if (!d[i]) {

d[i] = val;

break;

}

val ^= d[i];

}

return val > 0ll;

}

ll query\_max() {//寻找最大值

ll ret = 0;

for (int i = 60; i >= 0; i--)

if ((ret ^ d[i]) > ret)

ret ^= d[i];

return ret;

}

ll query\_min() {//寻找最小值

for (int i = 0; i <= 60; i++)

if (d[i])

return d[i];

return 0;

}

void rebuild() {//将线性基里的每个元素转化为两两互不影响的二进制表示(也就是前面异或后面为0的情况(大概))

for (int i = 60; i >= 0; i--)

for (int j = i - 1; j >= 0; j--)

if (d[i] & (1LL << j))

d[i] ^= d[j];

for (int i = 0;i <= 60; i++)

if (d[i])

p[cnt++] = d[i];

}

ll kthquery(ll k) {//第k小的异或

int ret = 0;

if (k >= (1LL << cnt))

return -1;

for (int i = 60;i >= 0; i--)

if (k & (1LL << i))

ret ^= p[i];

return ret;

}

L\_B merge(const L\_B &n1, const L\_B &n2) {

L\_B ret = n1;

for (int i = 60; i >= 0; i--)

if (n2.d[i])

ret.insert(n1.d[i]);

return ret;

}

};

BSGS

#include<bits/stdc++.h>//a^x=b(mod p)//找到最小的x使等式成立

#define Mod1(rt,mod) ((rt>=mod)&&(rt-=mod))

using namespace std;

long long a, b, p, x;

map<long long, long long> mp;

inline long long gcd(long long x, long long y)

{

return y ? gcd(y, x % y) : x;

}

inline long long ksm(long long x, long long y, long long mod)

{

long long res = 1;

while (y)

{

if (y & 1)

res = (res \* x) % mod;

x = (1ll\*x \* x)%mod;

y >>= 1;

}

return res;

}

void exgcd(long long a, long long b, long long& x, long long& y)

{

if (!b)

{

x = 1;

y = 0;

}

else

{

exgcd(b, a % b, y, x);

y -= a / b \* x;

}

}

inline void exBSGS(long long a, long long b, long long p)

{

if (b == 1)

{

printf("0\n");

return;

}

long long d = gcd(a, p), t = 1, k = 0;

while (d != 1)

{

if (b % d)

{

printf("No Solution\n");

return;

}

++k;

b /= d;

p /= d;

t = (t \* (a / d)) % p;//[1,k]的处理

d = gcd(a, p);

if (b == t)

{

printf("%lld\n", k);

return;

}

}

mp.clear();

long long m = ceil(sqrt(p)), ans, x, y;

exgcd(t, p, x, y);

x = (x % p + p) % p;

if (!x)

x += p;

b = b \* x;

for (int j = 0; j <= m; ++j)

{

if (j == 0)

{

ans = b % p;

mp[ans] = j;

continue;

}

ans = (ans \* a) % p;

mp[ans] = j;

}

long long pd = 0;

ans = 1;

x = ksm(a, m, p);

for (int i = 1; i <= m; ++i)

{

ans = (ans \* x) % p;

if (mp.find(ans)!=mp.end())//懂了，长记性了，下辈子都不可能直接找了，就用find了

{

x = i \* m - mp[ans];

printf("%lld\n", x + k);

pd = 1;

break;

}

}

if (!pd)

printf("No Solution\n");

return;

}

int main()

{

while (~scanf("%lld%lld%lld", &a, &p, &b))

{

if (a == 0 && b == 0 && p == 0)

break;

a = a % p;

b = b % p;

exBSGS(a, b, p);

}

return 0;

}

二次剩余

x的平方=n(mod p),求满足x的解，可能无解，1个解，2个解

ll w;

struct num {

ll x, y;

};

num mul(num a, num b, ll p)

{

num ans = { 0,0 };

ans.x = ((a.x \* b.x % p + a.y \* b.y % p \* w % p) % p + p) % p;

ans.y = ((a.x \* b.y % p + a.y \* b.x % p) % p + p) % p;

return ans;

}

ll powwR(ll a, ll b, ll p) {

ll ans = 1;

while (b) {

if (b & 1)ans = 1ll \* ans % p \* a % p;

a = a % p \* a % p;

b >>= 1;

}

return ans % p;

}

ll powwi(num a, ll b, ll p) {

num ans = { 1,0 };

while (b) {

if (b & 1)ans = mul(ans, a, p);

a = mul(a, a, p);

b >>= 1;

}

return ans.x % p;

}

ll solve(ll n, ll p)

{

n %= p;

if (p == 2)return n;

if (powwR(n, (p - 1) / 2, p) == p - 1)return -1;//不存在

ll a;

while (1)

{

a = rand() % p;

w = ((a \* a % p - n) % p + p) % p;

if (powwR(w, (p - 1) / 2, p) == p - 1)break;

}

num x = { a,1 };

return powwi(x, (p + 1) / 2, p);

}

Ans2 = p – ans1;

高斯消元

int n, line, id[MAXN]; double a[MAXN][MAXN];

int main() {

cin>>n;

for(int i = 1; i <= n; ++i)

for(int j = 1; j <= n+1; ++j)

cin>>a[i][j];

line = 1; //分开记录当前方程和当前主元以便调换消元顺序

for(int i = 1; i <= n; ++i) {

int \_max = line;

for(int j = line+1; j <= n; ++j)

if(fabs(a[j][i]) > fabs(a[\_max][i])) \_max = j;

if(!a[\_max][i]) continue;

for(int j = 1; j <= n+1; ++j) swap(a[line][j], a[\_max][j]);

for(int j = 1; j <= n; ++j) {

if(j == line) continue;

double t = 1.0 \* a[j][i] / a[line][i];

for(int k = i+1; k <= n+1; ++k)

a[j][k] -= a[line][k] \* t;

} id[i] = line; ++line;

}

vector <int> tt;

for(int i = 1; i <= n; ++i) if(!id[i]) tt.push\_back(i);

for(int j = 0; line <= n; ++line, ++j) id[tt[j]] = line;

//给剩下主元随便分配一个剩下的方程

for(int i = 1; i <= n; ++i) if(!a[id[i]][i] && a[id[i]][n+1]) {

puts("-1"); return 0;

//无解

}

for(int i = 1; i <= n; ++i) if(!a[id[i]][i] && !a[id[i]][n+1]) {

puts("0"); return 0;

//无穷多实数解

}

for(int i = 1; i <= n; ++i) printf("x%d=%.2lf\n", i, fabs(a[id[i]][n+1] / a[id[i]][i]) < eps ? 0.0 : a[id[i]][n+1] / a[id[i]][i]);

}

Miller-rabin

using u128 = long long;

namespace Factorizer {

constexpr bool miller\_rabin(long long n) {

if (n <= 1 || (n != 2 && n % 2 == 0)) return false;

for (auto a : {3, 5, 7, 11, 13, 17, 19, 23, 29}) {

if (n % a == 0) return n == a;

}

if (n < 31 \* 31) return true;

long long d = n - 1;

while (d % 2 == 0) d /= 2;

constexpr long long bases[] = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};

for (long long a : bases) {

if (n == a) return true;

long long t = d;

long long y = 1 % n;

for (long long \_t = t; \_t != 0; \_t >>= 1) {

if (\_t & 1) y = (u128) y \* a % n;

a = (u128) a \* a % n;

}

while (t != n - 1 && y != 1 && y != n - 1) {

y = (u128) y \* y % n;

t <<= 1;

}

if (y != n - 1 && t % 2 == 0) return false;

}

return true;

}

vector<int> primes;

vector<int> least;

void euler\_sieve(int n) {

primes = vector<int>();

least.assign(n + 1, 0);

for (int i = 2; i <= n; ++i) {

if (least[i] == 0) {

least[i] = i;

primes.push\_back(i);

}

int now = 0;

while (now < (int) primes.size()

&& primes[now] <= least[i] && i \* primes[now] <= n) {

least[i \* primes[now]] = primes[now];

now += 1;

}

}

}

long long pollard\_rho(long long n) {

if (miller\_rabin(n)) return n;

long long now = 0;

do {

long long t = gcd(++now, n), r = t;

if (t != 1 && t != n) return t;

long long g = 1;

do {

t = ((u128) t \* t % n + now) % n;

r = ((u128) r \* r % n + now) % n;

r = ((u128) r \* r % n + now) % n;

} while ((g = gcd(abs(t - r), n)) == 1);

if (g != n) return g;

} while (now < n / now);

return 0;

}

template <typename T>

vector<T> factor(T n) {

vector<T> g, d;

d.push\_back(n);

while (!d.empty()) {

auto v = d.back();

d.pop\_back();

auto rho = pollard\_rho(v);

if (rho == v) {

g.push\_back(rho);

} else {

d.push\_back(rho);

d.push\_back(v / rho);

}

}

sort(g.begin(), g.end());

return g;

}

}

using namespace Factorizer;

莫比乌斯反演：

日历

描述已自动生成

文本, 白板

描述已自动生成

图示, 示意图

描述已自动生成

分块：

int ans = 0;

for(int l = 1, r = 0; l <= n; l = r + 1) {

r = n / (n / l);

ans += (r - l + 1) \* (n / l);

}

扩展孙子定理

template<typename T>

T exgcd(T a, T b, T &x, T &y) {

if (a == 0) {

x = 0, y = 1;

return b;

}

T p = b / a;

T g = exgcd(b - p \* a, a, y, x);

x -= 1ll \* p \* y;

return g;

}

template<typename T>

T gcd(T x, T y) {

return y == 0 ? x : gcd(y, x % y);

}

template<typename T>

T lcm(T a, T b) {

return a / gcd(a, b) \* b;

}

/\*

run = a1(mod b1)

run = a2(mod b2)

...

\*/

template<typename T>

T excrt(T k, T \*a, T \*b) {

T M = b[0], ans = a[0];

for (int i = 1; i <= k - 1; i++) {

T x0, y0;

T c = a[i] - ans;

T g = exgcd(M, b[i], x0, y0);

if (c % g != 0) return -1;

x0 = 1ll \* x0 \* (c / g) % (b[i] / g);

ans = 1ll \* x0 \* M + ans;

M = lcm(M, b[i]);

ans = (ans % M + M) % M;

}

return ans;

}

第二类斯特林数：

图片包含 示意图

描述已自动生成

Ntt加速，待补：

生成函数：

图片包含 游戏机, 物体, 钟表

描述已自动生成

文本

中度可信度描述已自动生成

文本, 信件

描述已自动生成

示意图

描述已自动生成

牛顿迭代法：

double x1,x2,x3,a,b,c,d;

double f(double x){return a\*x\*x\*x+b\*x\*x+c\*x+d;}

double df(double x){return 3\*a\*x\*x+2\*b\*x+c;}

double slove(double l,double r)

{

double x,x0=(l+r)/2;

while(abs(x0-x)>eps)

x=x0-f(x0)/df(x0),swap(x0,x);

return x;

}

错排公式：

D(n) = n \* D(n - 1) + (-1) ^ n

D(n) = (n - 1) \* (D(n - 1) + D(n - 2))

D(n) = n! / e + 0.5

e = 2.7.......

莫比乌斯反演

文本, 信件

描述已自动生成

文本

描述已自动生成

杜教筛  
图形用户界面

描述已自动生成

求积性函数前缀和

  rep(i, 1, maxn) {  
    mu[i] += mu[i - 1];  
    phi[i] += phi[i - 1];  
  }  
  return;  
}  
ll getid(ll x) { // 卷积函数I(x)=x  
  return x;  
}  
ll getphi(ll x) {  
  if (x <= maxn)  
    return phi[x];  
  if (sum\_phi[x])  
    return sum\_phi[x];  
  ll ans = (x + 1) \* 1ll \* x / 2;  
  for (ll l = 2, r; l <= x; l = r + 1) {  
    r = x / (x / l);  
    ans -= 1ll \* (getid(r) - getid(l - 1)) \* getphi(x / l);  
  }  
  return sum\_phi[x] = ans / getid(1);  
}