Contents

1 ax+by=c 求解

```
pair<ll,ll> exgcd(ll a, ll b, ll c)
{
    function<ll(ll, ll, ll &, ll &)> ex_gcd = [&](ll a,
        ll b, ll & x, ll & y)
    {
        if (b == 0)
        {
            x = 1;
            y = 0;
            return a;
        }
        ll x1, y1;
        ll g = ex_gcd(b, a % b, x1, y1);
        x = y1;
        y = x1 - a / b * y1;
        return g;
    };
    ll x, y;
    ll g = ex_gcd(a, b, x, y);
    if (c % g != 0)
        return {0, 0};
    ll z = abs(b / g);
    x = (__int128_t)x * (c / g) % z;
    x = (x + z) % z;
    ll w = abs(a / g);
    y = (__int128_t)y * (c / g) % w;
    y = (y + w) % w;
    ll xmin = x, ymin = y;
    ll xmax = (c - (__int128_t)b * ymin) / a;
    ll ymax = (c - (__int128_t)a * xmin) / b;
    return {x, z};
}
```

2 jly 多项式

```cpp
#include <bits/stdc++.h>

constexpr int P = 998244353;
using i64 = long long;
// assume -P <= x < 2P
int norm(int x) {
    if (x < 0) {
        x += P;
    }
    if (x >= P) {
        x -= P;
    }
    return x;
}
template<class T>
T power(T a, int b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}
struct Z {
    int x;
    Z(int x = 0) : x(norm(x)) {}
    int val() const {
        return x;
    }
    Z operator-() const {
        return Z(norm(P - x));
    }
    Z inv() const {
        assert(x != 0);
        return power(*this, P - 2);
    }
    Z &operator*=(const Z &rhs) {
        x = i64(x) * rhs.x % P;
        return *this;
    }
    Z &operator+=(const Z &rhs) {
        x = norm(x + rhs.x);
        return *this;
```

```
    }
    Z &operator-=(const Z &rhs) {
        x = norm(x - rhs.x);
        return *this;
    }
    Z &operator/=(const Z &rhs) {
        return *this *= rhs.inv();
    }
    friend Z operator*(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res *= rhs;
        return res;
    }
    friend Z operator+(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res += rhs;
        return res;
    }
    friend Z operator-(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res -= rhs;
        return res;
    }
    friend Z operator/(const Z &lhs, const Z &rhs) {
        Z res = lhs;
        res /= rhs;
        return res;
    }
    friend std::istream &operator>>(std::istream &is, Z &
        a) {
        i64 v;
        is >> v;
        a = Z(v);
        return is;
    }
    friend std::ostream &operator<<(std::ostream &os,
        const Z &a) {
        return os << a.val();
    }
};

std::vector<int> rev;
std::vector<Z> roots{0, 1};
void dft(std::vector<Z> &a) {
    int n = a.size();
```

```
    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }
    if (int(roots.size()) < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            Z e = power(Z(3), (P - 1) >> (k + 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++)
                {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = roots[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                Z u = a[i + j];
                Z v = a[i + j + k] * roots[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}
void idft(std::vector<Z> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    Z inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] *= inv;
    }
}
```

```cpp
struct Poly {
    std::vector<Z> a;
    Poly() {}
    Poly(const std::vector<Z> &a) : a(a) {}
    Poly(const std::initializer_list<Z> &a) : a(a) {}
    int size() const {
        return a.size();
    }
    void resize(int n) {
        a.resize(n);
    }
    Z operator[](int idx) const {
        if (idx < size()) {
            return a[idx];
        } else {
            return 0;
        }
    }
    Z &operator[](int idx) {
        return a[idx];
    }
    Poly mulxk(int k) const {
        auto b = a;
        b.insert(b.begin(), k, 0);
        return Poly(b);
    }
    Poly modxk(int k) const {
        k = std::min(k, size());
        return Poly(std::vector<Z>(a.begin(), a.begin() +
            k));
    }
    Poly divxk(int k) const {
        if (size() <= k) {
            return Poly();
        }
        return Poly(std::vector<Z>(a.begin() + k, a.end()
           ));
    }
    friend Poly operator+(const Poly &a, const Poly &b) {
        std::vector<Z> res(std::max(a.size(), b.size()));
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = a[i] + b[i];
        }
        return Poly(res);
    }
    friend Poly operator-(const Poly &a, const Poly &b) {
```

```
        std::vector<Z> res(std::max(a.size(), b.size()));
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = a[i] - b[i];
        }
        return Poly(res);
    }
    friend Poly operator*(Poly a, Poly b) {
        if (a.size() == 0 || b.size() == 0) {
            return Poly();
        }
        int sz = 1, tot = a.size() + b.size() - 1;
        while (sz < tot) {
            sz *= 2;
        }
        a.a.resize(sz);
        b.a.resize(sz);
        dft(a.a);
        dft(b.a);
        for (int i = 0; i < sz; ++i) {
            a.a[i] = a[i] * b[i];
        }
        idft(a.a);
        a.resize(tot);
        return a;
    }
    friend Poly operator*(Z a, Poly b) {
        for (int i = 0; i < int(b.size()); i++) {
            b[i] *= a;
        }
        return b;
    }
    friend Poly operator*(Poly a, Z b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] *= b;
        }
        return a;
    }
    Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
```

```
Poly deriv() const {
    if (a.empty()) {
        return Poly();
    }
    std::vector<Z> res(size() - 1);
    for (int i = 0; i < size() - 1; ++i) {
        res[i] = (i + 1) * a[i + 1];
    }
    return Poly(res);
}
Poly integr() const {
    std::vector<Z> res(size() + 1);
    for (int i = 0; i < size(); ++i) {
        res[i + 1] = a[i] / (i + 1);
    }
    return Poly(res);
}
Poly inv(int m) const {
    Poly x{a[0].inv()};
    int k = 1;
    while (k < m) {
        k *= 2;
        x = (x * (Poly{2} - modxk(k) * x)).modxk(k);
    }
    return x.modxk(m);
}
Poly log(int m) const {
    return (deriv() * inv(m)).integr().modxk(m);
}
Poly exp(int m) const {
    Poly x{1};
    int k = 1;
    while (k < m) {
        k *= 2;
        x = (x * (Poly{1} - x.log(k) + modxk(k))).
            modxk(k);
    }
    return x.modxk(m);
}
Poly pow(int k, int m) const {
    int i = 0;
    while (i < size() && a[i].val() == 0) {
        i++;
    }
    if (i == size() || 1LL * i * k >= m) {
        return Poly(std::vector<Z>(m));
```

```
    }
    Z v = a[i];
    auto f = divxk(i) * v.inv();
    return (f.log(m - i * k) * k).exp(m - i * k).
        mulxk(i * k) * power(v, k);
}
Poly sqrt(int m) const {
    Poly x{1};
    int k = 1;
    while (k < m) {
        k *= 2;
        x = (x + (modxk(k) * x.inv(k)).modxk(k)) * ((
            P + 1) / 2);
    }
    return x.modxk(m);
}
Poly mulT(Poly b) const {
    if (b.size() == 0) {
        return Poly();
    }
    int n = b.size();
    std::reverse(b.a.begin(), b.a.end());
    return ((*this) * b).divxk(n - 1);
}
std::vector<Z> eval(std::vector<Z> x) const {
    if (size() == 0) {
        return std::vector<Z>(x.size(), 0);
    }
    const int n = std::max(int(x.size()), size());
    std::vector<Poly> q(4 * n);
    std::vector<Z> ans(x.size());
    x.resize(n);
    std::function<void(int, int, int)> build = [&](
        int p, int l, int r) {
        if (r - l == 1) {
            q[p] = Poly{1, -x[l]};
        } else {
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            q[p] = q[2 * p] * q[2 * p + 1];
        }
    };
    build(1, 0, n);
    std::function<void(int, int, int, const Poly &)>
        work = [&](int p, int l, int r, const Poly &
```

```
        num) {
            if (r - l == 1) {
                if (l < int(ans.size())) {
                    ans[l] = num[0];
                }
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).
                    modxk(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).
                    modxk(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};
```

3 mint

```
template<class T>
constexpr T power(T a, ll b) {
T res = 1;
for (; b; b /= 2, a *= a) {
if (b % 2) {
res *= a;
}
}
return res;
}

template<int P>
struct MInt {
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(ll x) : x{norm(x % P)} {}

    constexpr int norm(int x) const {
        if (x < 0) {
            x += P;
        }
        if (x >= P) {
            x -= P;
        }
        return x;
    }
    constexpr MInt operator-() const {
        MInt res;
        res.x = norm(P - x);
        return res;
    }
    constexpr MInt inv() const {
        assert(x != 0);
        return power(*this, P - 2);
    }
    constexpr MInt &operator*=(MInt rhs) {
        x = 1LL * x * rhs.x % P;
        return *this;
    }
    constexpr MInt &operator+=(MInt rhs) {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MInt &operator-=(MInt rhs) {
```

```
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MInt &operator/=(MInt rhs) {
        return *this *= rhs.inv();
    }
    friend constexpr MInt operator*(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MInt operator+(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MInt operator-(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MInt operator/(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::
       istream &is, MInt &a) {
        ll v;
        is >> v;
        a = MInt(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::
       ostream &os, const MInt &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MInt lhs, MInt rhs)
       {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MInt lhs, MInt rhs)
       {
        return lhs.val() != rhs.val();
    }
};
```

```cpp
template<int V, int P>
constexpr MInt<P> CInv = MInt<P>(V).inv();

constexpr int P = 998244353;
// constexpr int P = 1e9+7;
using Z = MInt<P>;

struct Comb {
    int n;
    std::vector<Z> _fac;
    std::vector<Z> _invfac;
    std::vector<Z> _inv;

    Comb() : n{0}, _fac{1}, _invfac{1}, _inv{0} {}
    Comb(int n) : Comb() {
        init(n);
    }

    void init(int m) {
        if (m <= n) return;
        _fac.resize(m + 1);
        _invfac.resize(m + 1);
        _inv.resize(m + 1);

        for (int i = n + 1; i <= m; i++) {
            _fac[i] = _fac[i - 1] * i;
        }
        _invfac[m] = _fac[m].inv();
        for (int i = m; i > n; i--) {
            _invfac[i - 1] = _invfac[i] * i;
            _inv[i] = _invfac[i] * _fac[i - 1];
        }
        n = m;
    }

    Z fac(int m) {
        if (m > n) init(2 * m);
        return _fac[m];
    }
    Z invfac(int m) {
        if (m > n) init(2 * m);
        return _invfac[m];
    }
    Z inv(int m) {
        if (m > n) init(2 * m);
```

```
        return _inv[m];
    }
    Z binom(int n, int m) {
        if (n < m || m < 0) return 0;
        return fac(n) * invfac(m) * invfac(n - m);
    }
} comb;
struct Inversion {
    static constexpr int B = (1 << 10), T = (1 << 20);
    std::array < int, T + 1 > f, p;
    std::array < int, T * 3 + 3 > buf;
    int *I = buf.begin() + T;
    Inversion() {
        for (int i = 1; i <= B; i++) {
            int s = 0, d = (i << 10);
            for (int j = 1; j <= T; j++) {
                if ((s += d) >= P) s -= P;
                if (s <= T) {
                    if (!f[j]) f[j] = i, p[j] = s;
                }
                else if (s >= P - T) {
                    if (!f[j]) f[j] = i, p[j] = s - P;
                }
                else {
                    int t = (P - T - s - 1) / d;
                    s += t * d, j += t;
                }
            }
        }
        I[1] = f[0] = 1;
        for (int i = 2; i <= (T << 1); i++)
            I[i] = 1ll * (P - P / i) * I[P % i] % P;

        for (int i = -1; i >= -T; i--)
            I[i] = P - I[-i];

    }
    Z inv(int x) {
        return Z(1)* I[p[x >> 10] + (x & 1023) * f[x >>
            10]] * f[x >> 10];
    }
};
```

4 Mlong

```
template<class T>
constexpr T power(T a, ll b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}

constexpr ll mul(ll a, ll b, ll p) {
    ll res = a * b - ll(1.L * a * b / p) * p;
    res %= p;
    if (res < 0) {
        res += p;
    }
    return res;
}
template<ll P>
struct MLong {
    ll x;
    constexpr MLong() : x{} {}
    constexpr MLong(ll x) : x{norm(x % getMod())} {}

    static ll Mod;
    constexpr static ll getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(ll Mod_) {
        Mod = Mod_;
    }
    constexpr ll norm(ll x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
```

```cpp
}
constexpr ll val() const {
    return x;
}
explicit constexpr operator ll() const {
    return x;
}
constexpr MLong operator-() const {
    MLong res;
    res.x = norm(getMod() - x);
    return res;
}
constexpr MLong inv() const {
    assert(x != 0);
    return power(*this, getMod() - 2);
}
constexpr MLong &operator*=(MLong rhs) & {
    x = mul(x, rhs.x, getMod());
    return *this;
}
constexpr MLong &operator+=(MLong rhs) & {
    x = norm(x + rhs.x);
    return *this;
}
constexpr MLong &operator-=(MLong rhs) & {
    x = norm(x - rhs.x);
    return *this;
}
constexpr MLong &operator/=(MLong rhs) & {
    return *this *= rhs.inv();
}
friend constexpr MLong operator*(MLong lhs, MLong rhs
    ) {
    MLong res = lhs;
    res *= rhs;
    return res;
}
friend constexpr MLong operator+(MLong lhs, MLong rhs
    ) {
    MLong res = lhs;
    res += rhs;
    return res;
}
friend constexpr MLong operator-(MLong lhs, MLong rhs
    ) {
    MLong res = lhs;
```

```
        res -= rhs;
        return res;
    }
    friend constexpr MLong operator/(MLong lhs, MLong rhs
        ) {
        MLong res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::
        istream &is, MLong &a) {
        ll v;
        is >> v;
        a = MLong(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::
        ostream &os, const MLong &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MLong lhs, MLong rhs
        ) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MLong lhs, MLong rhs
        ) {
        return lhs.val() != rhs.val();
    }
};

template<>
ll MLong<0LL>::Mod = ll(1E18) + 9;

template<int V, int P>
constexpr MLong<P> CInv = MLong<P>(V).inv();

constexpr ll P = ll(1E18) + 9;
using Z = MLong<P>;
```

5 Pollard-pho

```
ll mul(ll a, ll b, ll m) {
    return static_cast<__int128>(a) * b % m;
}
ll power(ll a, ll b, ll m) {
    ll res = 1 % m;
    for (; b; b >>= 1, a = mul(a, a, m))
        if (b & 1)
            res = mul(res, a, m);
    return res;
}
bool isprime(ll n) {
    if (n < 2)
        return false;
    static constexpr int A[] = {2, 3, 5, 7, 11, 13, 17,
       19, 23};
    int s = __builtin_ctzll(n - 1);
    ll d = (n - 1) >> s;
    for (auto a : A) {
        if (a == n)
            return true;
        ll x = power(a, d, n);
        if (x == 1 || x == n - 1)
            continue;
        bool ok = false;
        for (int i = 0; i < s - 1; ++i) {
            x = mul(x, x, n);
            if (x == n - 1) {
                ok = true;
                break;
            }
        }
        if (!ok)
            return false;
    }
    return true;
}
std::vector<ll> factorize(ll n) {
    std::vector<ll> p;
    std::function<void(ll)> f = [&](ll n) {
        if (n <= 10000) {
            for (int i = 2; i * i <= n; ++i)
                for (; n % i == 0; n /= i)
                    p.push_back(i);
            if (n > 1)
```

```
            p.push_back(n);
        return;
    }
    if (isprime(n)) {
        p.push_back(n);
        return;
    }
    auto g = [&](ll x) {
        return (mul(x, x, n) + 1) % n;
    };
    ll x0 = 2;
    while (true) {
        ll x = x0;
        ll y = x0;
        ll d = 1;
        ll power = 1, lam = 0;
        ll v = 1;
        while (d == 1) {
            y = g(y);
            ++lam;
            v = mul(v, std::abs(x - y), n);
            if (lam % 127 == 0) {
                d = std::gcd(v, n);
                v = 1;
            }
            if (power == lam) {
                x = y;
                power *= 2;
                lam = 0;
                d = std::gcd(v, n);
                v = 1;
            }
        }
        if (d != n) {
            f(d);
            f(n / d);
            return;
        }
        ++x0;
    }
};
f(n);
std::sort(p.begin(), p.end());
vector<ll>pp=p;
vector<ll>ppp;
pp.erase(unique(pp.begin(),pp.end()),pp.end());
```

```
    for(auto q:pp){
        ll w=count(p.begin(),p.end(),q);
        ll res=1;
        while(w){
            res*=q;
            w--;
        }
        ppp.push_back(res);
    }
    return p;
}
```

6 skip,Pollard-pho

```
namespace factor {
    using f64 = long double;
    ll p;
    f64 invp;
    void setmod(ll x) {
        p = x, invp = (f64) 1 / x;
    }
    ll mul(ll a, ll b) {
        ll z = a * invp * b + 0.5;
        ll res = a * b - z * p;
        return res + (res >> 63 & p);
    }
    ll power(ll a, ll x, ll res = 1) {
        for(;x;x >>= 1, a = mul(a, a))
            if(x & 1) res = mul(res, a);
        return res;
    }
    inline ll rho(ll n) {
        if(!(n & 1)) return 2;
        static std::mt19937_64 gen((size_t)"hehezhou");
        ll x = 0, y = 0, prod = 1;
        auto f = [&](ll o) { return mul(o, o) + 1; };
        setmod(n);
        for(int t = 30, z = 0;t % 64 || std::gcd(prod, n)
            == 1;++t) {
            if (x == y) x = ++ z, y = f(x);
            if(ll q = mul(prod, x + n - y)) prod = q;
            x = f(x), y = f(f(y));
        }
        return std::gcd(prod, n);
    }
    bool checkprime(ll p) {
        if(p == 1) return 0;
        setmod(p);
        ll d = __builtin_ctzll(p - 1), s = (p - 1) >> d;
        for(ll a : {2, 3, 5, 7, 11, 13, 82, 373}) {
            if(a % p == 0)
                continue;
            ll x = power(a, s), y;
            for(int i = 0;i < d;++i, x = y) {
                y = mul(x, x);
                if(y == 1 && x != 1 && x != p - 1)
                    return 0;
            }
```

```
            if(x != 1) return 0;
        }
        return 1;
    }
    std::vector<ll> get_factor(ll x) {
        std::queue<ll> q; q.push(x);
        std::vector<ll> res;
        for(;q.size();) {
            ll x = q.front(); q.pop();
            if(x == 1) continue;
            if(checkprime(x)) {
                res.push_back(x);
                continue;
            }
            ll y = rho(x);
            q.push(y), q.push(x / y);
        }
        sort(res.begin(), res.end());
        return res;
    }
}
```

7 Umint

```
using u32 = unsigned;
using ull = unsigned long long;
template<typename T>
constexpr T power(T a, ull b) {
    T res {1};
    for (; b != 0; b /= 2, a *= a) {
        if (b % 2 == 1) {
            res *= a;
        }
    }
    return res;
}

template<u32 P>
constexpr u32 mulMod(u32 a, u32 b) {
    return 1ULL * a * b % P;
}

template<ull P>
constexpr ull mulMod(ull a, ull b) {
    ull res = a * b - ull(1.L * a * b / P - 0.5L) * P;
    res %= P;
    return res;
}

template<typename U, U P>
struct ModIntBase {
public:
    constexpr ModIntBase() : x {0} {}

    template<typename T, typename = std::enable_if_t<std
        ::is_integral<T>::value>>
    constexpr ModIntBase(T x_) : x {norm(x_ %P)} {}

    constexpr static U norm(U x) {
        if ((x >> (8 * sizeof(U) - 1) & 1) == 1) {
            x += P;
        }
        if (x >= P) {
            x -= P;
        }
        return x;
    }
```

```
constexpr U val() const {
    return x;
}

constexpr ModIntBase operator-() const {
    ModIntBase res;
    res.x = norm(P - x);
    return res;
}

constexpr ModIntBase inv() const {
    return power(*this, P - 2);
}

constexpr ModIntBase &operator*=(const ModIntBase &
   rhs) & {
    x = mulMod<P>(x, rhs.val());
    return *this;
}

constexpr ModIntBase &operator+=(const ModIntBase &
   rhs) & {
    x = norm(x + rhs.x);
    return *this;
}

constexpr ModIntBase &operator-=(const ModIntBase &
   rhs) & {
    x = norm(x - rhs.x);
    return *this;
}

constexpr ModIntBase &operator/=(const ModIntBase &
   rhs) & {
    return *this *= rhs.inv();
}

friend constexpr ModIntBase operator*(ModIntBase lhs,
     const ModIntBase &rhs) {
    lhs *= rhs;
    return lhs;
}

friend constexpr ModIntBase operator+(ModIntBase lhs,
     const ModIntBase &rhs) {
    lhs += rhs;
```

```
        return lhs;
    }

    friend constexpr ModIntBase operator-(ModIntBase lhs,
         const ModIntBase &rhs) {
        lhs -= rhs;
        return lhs;
    }

    friend constexpr ModIntBase operator/(ModIntBase lhs,
         const ModIntBase &rhs) {
        lhs /= rhs;
        return lhs;
    }

    friend constexpr std::ostream &operator<<(std::
        ostream &os, const ModIntBase &a) {
        return os << a.val();
    }

    friend constexpr bool operator==(ModIntBase lhs,
        ModIntBase rhs) {
        return lhs.val() == rhs.val();
    }

    friend constexpr bool operator!=(ModIntBase lhs,
        ModIntBase rhs) {
        return lhs.val() != rhs.val();
    }

    friend constexpr bool operator<(ModIntBase lhs,
        ModIntBase rhs) {
        return lhs.val() < rhs.val();
    }

private:
    U x;
};

template<u32 P>
using ModInt = ModIntBase<u32, P>;

template<ull P>
using ModInt64 = ModIntBase<ull, P>;
constexpr ull P = ull(1E18) + 9;
using Z = ModInt64<P>;
```

8 分数类

```
//
// Created by 墨华 on 2024/6/2.
//
template<class T>
struct Frac {
    T num;
    T den;
    Frac(T num_, T den_) : num(num_), den(den_) {
        if (den < 0) {
            den = -den;
            num = -num;
        }
    }
    Frac() : Frac(0, 1) {}
    Frac(T num_) : Frac(num_, 1) {}
    explicit operator double() const {
        return 1. * num / den;
    }
    Frac &operator+=(const Frac &rhs) {
        num = num * rhs.den + rhs.num * den;
        den *= rhs.den;
        T g = std::gcd(num, den);
        num/=g;
        den/=g;
        return *this;
    }
    Frac &operator-=(const Frac &rhs) {
        num = num * rhs.den - rhs.num * den;
        den *= rhs.den;
        T g = std::gcd(num, den);
        num/=g;
        den/=g;
        return *this;
    }
    Frac &operator*=(const Frac &rhs) {
        num *= rhs.num;
        den *= rhs.den;
        T g = std::gcd(num, den);
        num/=g;
        den/=g;
        return *this;
    }
    Frac &operator/=(const Frac &rhs) {
        num *= rhs.den;
```

```
        den *= rhs.num;
        if (den < 0) {
            num = -num;
            den = -den;
        }
        T g = std::gcd(num, den);
        num/=g;
        den/=g;
        return *this;
    }
    friend Frac operator+(Frac lhs, const Frac &rhs) {
        return lhs += rhs;
    }
    friend Frac operator-(Frac lhs, const Frac &rhs) {
        return lhs -= rhs;
    }
    friend Frac operator*(Frac lhs, const Frac &rhs) {
        return lhs *= rhs;
    }
    friend Frac operator/(Frac lhs, const Frac &rhs) {
        return lhs /= rhs;
    }
    friend Frac operator-(const Frac &a) {
        return Frac(-a.num, a.den);
    }
    friend bool operator==(const Frac &lhs, const Frac &
        rhs) {
        return lhs.num * rhs.den == rhs.num * lhs.den;
    }
    friend bool operator!=(const Frac &lhs, const Frac &
        rhs) {
        return lhs.num * rhs.den != rhs.num * lhs.den;
    }
    friend bool operator<(const Frac &lhs, const Frac &
        rhs) {
        return lhs.num * rhs.den < rhs.num * lhs.den;
    }
    friend bool operator>(const Frac &lhs, const Frac &
        rhs) {
        return lhs.num * rhs.den > rhs.num * lhs.den;
    }
    friend bool operator<=(const Frac &lhs, const Frac &
        rhs) {
        return lhs.num * rhs.den <= rhs.num * lhs.den;
    }
```

```
    friend bool operator>=(const Frac &lhs, const Frac &
        rhs) {
        return lhs.num * rhs.den >= rhs.num * lhs.den;
    }
    friend std::ostream &operator<<(std::ostream &os,
        Frac x) {
        T g = std::gcd(x.num, x.den);
        if (x.den == g) {
            return os << x.num / g;
        } else {
            return os << x.num / g << "/" << x.den / g;
        }
    }
};

using F = Frac<ll>;
```

9 可变模数 mint

```
template<class T>
constexpr T power(T a, ll b) {
    T res {1};
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}

constexpr ll mul(ll a, ll b, ll p) {
    ll res = a * b - ll(1.L * a * b / p) * p;
    res %= p;
    if (res < 0) {
        res += p;
    }
    return res;
}

template<ll P>
struct MInt {
    ll x;
    constexpr MInt() : x {0} {}
    constexpr MInt(ll x) : x {norm(x % getMod())} {}

    static ll Mod;
    constexpr static ll getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(ll Mod_) {
        Mod = Mod_;
    }
    constexpr ll norm(ll x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
```

```
        return x;
    }
    constexpr ll val() const {
        return x;
    }
    constexpr MInt operator-() const {
        MInt res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MInt inv() const {
        return power(*this, getMod() - 2);
    }
    constexpr MInt &operator*=(MInt rhs) & {
        if (getMod() < (1ULL << 31)) {
            x = x * rhs.x % int(getMod());
        } else {
            x = mul(x, rhs.x, getMod());
        }
        return *this;
    }
    constexpr MInt &operator+=(MInt rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MInt &operator-=(MInt rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MInt &operator/=(MInt rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MInt operator*(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MInt operator+(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MInt operator-(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res -= rhs;
        return res;
```

```
    }
    friend constexpr MInt operator/(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::
        istream &is, MInt &a) {
        ll v;
        is >> v;
        a = MInt(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::
        ostream &os, const MInt &a) {
        return os << a.val();
    }
    friend constexpr bool operator==(MInt lhs, MInt rhs)
        {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MInt lhs, MInt rhs)
        {
        return lhs.val() != rhs.val();
    }
    friend constexpr bool operator<(MInt lhs, MInt rhs) {
        return lhs.val() < rhs.val();
    }
};
template<>
ll MInt<0>::Mod = 998244353;
constexpr int P = 998244353;
using Z = MInt<0>;
```

10 拓展 exgcd 求逆元

```
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    ll d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
ll inv(ll n, ll M) {
    ll x, y;
    exgcd(n, M, x, y);
    x = (x % M + M) % M;
    return x;
}
```

11 矩阵乘法

```
#include<bits/stdc++.h>
using namespace std;
using ll=long long;
#define INF 0x3f3f3f3f
#define IOS ios::sync_with_stdio(false);cin.tie(0);cout.
    tie(0);
#define pll pair<int,int>
#define fi first
#define se second
const int N=200,P=100000007;
ll a[N+1][N+1],fa[N+1];
void aa(){
    ll w=[N+1][N+1];
    memset(w,0,sizeof(w));
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++){
            for(int k=1;k<=n;k++)
                w[i][j]+=a[i][k]*a[k][j],w[i][j]%=P;

        }
    }
    memcpy(a,w,sizeof(a));
}
void fa(){
    ll w[N+1];
    memset(w,0,sizeof(w));
    for(int i=1;i<=n;i++){
        for(int j=1;j<=n;j++)w[i]+=f[j]*a[j][i],w[i]%=P;
    }
    memcpy(f,w,sizeof(f));

}
void matrixpow(int k){
    for(;k;k/2){
        if(k&1)fa();
        aa();
    }
}
int main(){

}
```

12 线性基

```
#include<bits/extc++.h>
using namespace __gnu_cxx;
using namespace __gnu_pbds;
using namespace std;
using ll=long long;
#define LNF 0x3f3f3f3f3f3f3f3f
#define INF 0x3f3f3f3f
#define IOS ios::sync_with_stdio(false);cin.tie(0);cout.
    tie(0);
#define pll pair<int,int>
#define fi first
#define se second
const int N=210;
const int B=60;
struct linear_basis{
     ll num[B+1];
     void init(){
          for(int i=B-1;i>=0;i--)num[i]=0;
     }
     bool insert(ll x){
          for(int i=B-1;i>=0;i--){
               if(x&(1ll<<i)){
                    if(num[i]==0){num[i]=x;
                         return true;
                    };
                    x^=num[i];
               }
          }
          return false;
     }
     ll querymin(ll x){
          for(int i=B-1;i>=0;i--){
               x=min(x,x^num[i]);
          }
          return x;
     }
     ll querymax(ll x){
          for(int i=B-1;i>=0;i--){
               x=max(x,x^num[i]);
          }
          return x;
     }
};
void solve(void){
```

```
}
int main() {

    IOS;
    int t=1;
    // cin>>t;
    while(t--)
    solve();
    return 0;
}
```

13 莫比乌斯反演

```
const int maxn = 1e5 + 7;
int p[maxn];
int pr[maxn];
int tot = 0;
int phi[maxn];
int mu[maxn];
void get(ll n) {
    p[1] = phi[1] = mu[1] = 1;
    for (int i = 2; i < n; ++i) {
        if (!p[i]) mu[i] = -1, pr[++tot] = i, phi[i] = i
            - 1;
        for (int j = 1; j <= tot && i * pr[j] < n; ++j) {
            p[i * pr[j]] = pr[j];
            if (i % pr[j]) {
                mu[i * pr[j]] = -mu[i];
                phi[i * pr[j]] = phi[i] * phi[pr[j]];
            } else {
                phi[pr[j]*i] = phi[i] * pr[j];
                break;
            }
        }
    }
}
```