# Algorithm Library*

nickluo

November 7, 2021

## Contents



*Based on the ply-template by palayutm

# 1 vimrc

# 2 Z

```
using u32 = unsigned;
using u64 = unsigned long long;
const u32 MOD = 1E9 + 7;

struct Z {
    u32 v;
    Z(u32 v = 0) : v(v) {}
    Z& operator += (const Z &z) {
        v += z.v;
        if (v >= MOD) v -= MOD;
        return *this;
    }
    Z& operator -= (const Z &z) {
        if (v < z.v) v += MOD;
        v -= z.v;
        return *this;
    }
    Z& operator *= (const Z &z) {
        v = static_cast<u64>(v) * z.v % MOD;
        return *this;
    }
};
Z operator + (const Z &x, const Z &y) {
    return Z(x.v + y.v >= MOD ? x.v + y.v - MOD :
↪  x.v + y.v);
}
Z operator - (const Z &x, const Z &y) {
    return Z(x.v < y.v ? x.v + MOD - y.v : x.v -
↪  y.v);
}
Z operator * (const Z &x, const Z &y) {
    return Z(static_cast<u64>(x.v) * y.v % MOD);
}
Z qpow(Z base, u32 e) {
    Z ret(1);
    for (; e; e >>= 1) {
        if (e & 1) ret *= base;
        base *= base;
    }
    return ret;
}
istream& operator >> (istream &in, Z &x) {
    in >> x.v;
    return in;
}
ostream& operator << (ostream &os, const Z &z) {
    return os << z.v;
}

i64 n, lim, val[N];
int id1[N], id2[N];
bool npr[N]; int pri[N], pcnt; Z pg0[N], pg1[N];
Z g0[N], g1[N];

void prep() {
    for (int i = 2; i < (int) N; ++i) {
        if (!npr[i]) {
            pri[++pcnt] = i;
            pg0[pcnt] = pg0[pcnt - 1] + i;
            pg1[pcnt] = pg1[pcnt - 1] - 1;
        }
        for (int j = 1, k; j <= pcnt && (k = i *
↪  pri[j]) < (int) N; ++j) {
            npr[k] = true;
            if (i % pri[j] == 0) break;
        }
    }
}

int get_id(i64 x) {
    return x <= lim ? id1[x] : id2[n / x];
}

Z calc_f(int p, int c) {
    return p ^ c;
}

Z S(i64 n, int x) {
    if (n <= 1 || pri[x] > n) return 0;
    Z ret = g0[get_id(n)] + g1[get_id(n)];
    if (x == 1) ret += 2; // #6035 特殊 f(2) = 2 +
↪  1 = 3 != 1
    ret -= pg0[x - 1] + pg1[x - 1];
    for (int k = x; k <= pcnt; ++k) {
        i64 p1 = pri[k], p2 = p1 * pri[k];
        if (p2 > n) break;
        for (int e = 1; p2 <= n; p2 = (p1 = p2) *
↪  pri[k], ++e) {
            ret += S(n / p1, k + 1) *
↪  calc_f(pri[k], e);
            ret += calc_f(pri[k], e + 1);
        }
    }
    return ret;
}

int main() {
    n = read();
    lim = sqrt(n + .5);
    prep();
    int cnt = 0;
    for (i64 i = 1, j; i <= n; i = j + 1) {
        i64 t;
        j = n / (t = val[++cnt] = n / i);
        (t <= lim ? id1[t] : id2[i]) = cnt;
        t %= MOD;
        g0[cnt] = (n - 1) % MOD;
        g1[cnt] = t * (t - 1) / 2 % MOD;
    }
    for (int i = 1; i <= pcnt; ++i) {
        i64 bnd = (i64) pri[i] * pri[i];
        for (int j = 1, id; val[j] >= bnd; ++j) {
            id = get_id(val[j] / pri[i]);
            g0[j] -= (g0[id] - pg0[i - 1]);
            g1[j] -= (g1[id] - pg1[i - 1]) *
↪  pri[i];
        }
```

```
    }
    cout << 1 + S(n, 1) << '\n';
    return 0;
}
```

# 3 随机数生成器

```
mt19937
↪ rnd(chrono::steady_clock().now().time_since_epoch().count());
```

# 4 数列

## 4.1 多项式板子

```
// SZ: size * 4
const size_t SZ = 1 << 19;
using Poly = vector<Z>;
using i64 = long long;

template <typename InputZ, typename Output>
void sp_copy(InputZ begin, InputZ end, Output
↪ output) {
    while (begin != end) *output++ = begin++->v;
}
int get_lg(int x) {
    return 32 - __builtin_clz(x) - ((x & (-x)) ==
↪ x);
}
Z inv[SZ + 5], ww[SZ];
void prep() {
    static bool has_prep = false;
    if (has_prep) return;
    inv[0] = inv[1] = 1;
    for (unsigned i = 2; i <= SZ; ++i)
        inv[i] = MOD - MOD / i * inv[MOD % i];
    ww[0] = 1;
    Z mul = qpow(3, (MOD - 1) / SZ);
    for (unsigned i = 1; i < SZ; ++i)
        ww[i] = ww[i - 1] * mul;
    has_prep = true;
}
void fft(i64 a[], int lg, bool flag) {
    prep();
    int n = 1 << lg;
    if (flag) reverse(a + 1, a + n);
    static int rev[SZ], rev_lg = -1;
    if (rev_lg != lg) {
        for (int i = 0; i < n; ++i)
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1)
↪ << lg >> 1);
        rev_lg = lg;
    }
    for (int i = 0; i < n; ++i)
        if (rev[i] > i) swap(a[i], a[rev[i]]);
    for (int m = 1, l = 2; m < n; m <<= 1, l <<=
↪ 1) {
        i64 *x = a, *y = a + m, xx, yy; int *w,
↪ mul[SZ];
        for (int i = 0, j = 0, step = SZ / l; i <
↪ m; ++i, j += step)
            mul[i] = ww[j].v;
        for (int i = 0; i < n; i += l) {
            w = mul;
            for (int j = 0; j < m; ++j, ++x, ++y,
↪ ++w) {
                xx = *x;
                yy = *y % MOD * *w;
                *x = xx + yy;
                *y = xx - yy;
            }
            x += m;
            y += m;
        }
        if (l >> 15 & 1)
            for (int i = 0; i < n; ++i)
                a[i] %= MOD;
    }
    for (int i = 0; i < n; ++i) {
        a[i] %= MOD;
        if (flag) (a[i] *= inv[n].v) %= MOD;
        if (a[i] < 0) a[i] += MOD;
    }
}
void fft(Z a[], int lg, bool flag) {
    static i64 ta[SZ];
    sp_copy(a, a + (1 << lg), ta);
    fft(ta, lg, flag);
    copy(ta, ta + (1 << lg), a);
}
Poly operator += (Poly &f, const Poly &g) {
    if (g.size() > f.size()) f.resize(g.size());
    auto it = f.begin();
    auto jt = g.begin();
    while (jt != g.end()) *it++ += *jt++;
    return f;
}
Poly operator + (const Poly &f, const Poly &g) {
    Poly ret = f; return ret += g;
}
Poly operator -= (Poly &f, const Poly &g) {
    if (g.size() > f.size()) f.resize(g.size());
    auto it = f.begin();
    auto jt = g.begin();
    while (jt != g.end()) *it++ -= *jt++;
    return f;
}
Poly operator - (const Poly &f, const Poly &g) {
    Poly ret = f; return ret -= g;
}
Poly operator * (const Poly &f, const Poly &g) {
    u32 n = f.size() + g.size() - 1;
    if ((i64) f.size() * g.size() <= 2048) {
        static u64 ans[SZ];
        memset(ans, 0, sizeof(u64) * n);
        for (u32 i = 0; i < f.size(); ++i)
            for (u32 j = 0; j < g.size(); ++j)
                if ((ans[i + j] += (u64) f[i].v *
↪ g[j].v) >> 62)
                    ans[i + j] %= MOD;
        Poly ret(n);
```

```
        for (u32 i = 0; i < n; ++i) ret[i] =
↪   ans[i] % MOD;
        return ret;
    }
    Poly ret(f.size() + g.size() - 1);
    static i64 a[SZ], b[SZ];
    int lg = get_lg(n);
    memset(a, 0, sizeof(i64) << lg);
    memset(b, 0, sizeof(i64) << lg);
    sp_copy(f.begin(), f.end(), a);
    sp_copy(g.begin(), g.end(), b);
    fft(a, lg, 0);
    fft(b, lg, 0);
    for (u32 i = 0, _ = 1 << lg; i < _; ++i)
        (a[i] *= b[i]) %= MOD;
    fft(a, lg, 1);
    copy(a, a + n, ret.begin());
    return ret;
}
Poly& operator *= (Poly &f, const Poly &g) {
    return f = f * g;
}
Poly& operator *= (Poly &f, const Z &x) {
    for (Z &c : f) c *= x;
    return f;
}
Poly operator * (const Poly &f, const Z &x) {
    Poly ret = f; return ret *= x;
}
void calc_inv(Z arr[], Z brr[], int n) {
    if (n == 1) {
        brr[0] = qpow(arr[0], MOD - 2);
        return;
    }
    calc_inv(arr, brr, n >> 1);
    int lg = get_lg(n << 1);
    static Z ta[SZ], tb[SZ];
    memset(ta, 0, sizeof(Z) << lg);
    memset(tb, 0, sizeof(Z) << lg);
    copy(arr, arr +  n      , ta);
    copy(brr, brr + (n >> 1), tb);
    fft(ta, lg, 0);
    fft(tb, lg, 0);
    for (int i = 0, _ = 1 << lg; i < _; ++i)
        ta[i] = (2 - ta[i] * tb[i]) * tb[i];
    fft(ta, lg, 1);
    copy(ta, ta + n, brr);
}
Poly calc_inv(const Poly &f) {
    static Z a[SZ], b[SZ];
    int lg = get_lg(f.size());
    memset(a, 0, sizeof(Z) << lg);
    copy(f.begin(), f.end(), a);
    calc_inv(a, b, 1 << lg);
    return Poly(b, b + f.size());
}
Poly operator / (const Poly &f, const Poly &g) {
    if (f.size() < g.size()) return Poly();
    Poly tf = f; reverse(tf.begin(), tf.end());
    Poly tg = g; reverse(tg.begin(), tg.end());
    tg.resize(f.size() - g.size() + 1);
    Poly ret = tf * calc_inv(tg);
    ret.resize(f.size() - g.size() + 1);
    reverse(ret.begin(), ret.end());
    return ret;
}
Poly& operator /= (Poly &f, const Poly &g) {
    return f = f / g;
}
Poly operator % (const Poly &f, const Poly &g) {
    Poly ret = f - (f / g) * g;
    ret.resize(g.size() - 1);
    return ret;
}
Poly& operator %= (Poly &f, const Poly &g) {
    return f = f % g;
}
Poly calc_der(const Poly &f) {
    Poly ret(f.size() - 1);
    for (u32 i = 1; i < f.size(); ++i) ret[i - 1]
↪   = f[i] * i;
    return ret;
}
Poly calc_pri(const Poly &f) {
    prep();
    Poly ret(f.size() + 1);
    for (u32 i = 1; i <= f.size(); ++i) ret[i] =
↪   f[i - 1] * inv[i];
    return ret;
}
Poly calc_ln(const Poly &f) {
    assert(f[0].v == 1);
    Poly g = calc_der(f) * calc_inv(f);
    g.resize(f.size() - 1);
    return calc_pri(g);
}
Poly calc_exp(int arr[], int n) {
    if (n == 1) {
        assert(arr[0] == 0);
        return Poly{1};
    }
    Poly f = calc_exp(arr, n >> 1);
    Poly tf = f;
    tf.resize(n);
    Poly a = Poly(arr, arr + n);
    Poly g = f * (Poly{1} - calc_ln(tf) + a);
    g.resize(n);
    return g;
}
Poly calc_exp(const Poly &f) {
    static int a[SZ];
    int lg = get_lg(f.size());
    memset(a, 0, sizeof(int) << lg);
    sp_copy(f.begin(), f.end(), a);
    Poly ret = calc_exp(a, 1 << lg);
    ret.resize(f.size());
    return ret;
}
Poly operator ^ (const Poly &f, const int &e) {
    u32 trail = 0;
    for (u32 i = 0; i < f.size(); ++i)
        if (f[i].v) break; else ++trail;
```

```
    if ((i64) trail * e >= f.size())
        return Poly(f.size(), 0);
    Z lst = f[trail], inv = qpow(lst, MOD - 2);
    Poly g;
    for (u32 i = trail; i < f.size(); ++i)
        g.emplace_back(f[i] * inv);
    Poly ret = calc_exp(calc_ln(g) * e) *
↪   qpow(lst, e);
    Poly t0 = Poly(trail * e, 0);
    ret.insert(ret.begin(), t0.begin(), t0.end());
    ret.resize(f.size());
    return ret;
}
Poly& operator ^= (Poly &f, const int &e) {
    return f = f ^ e;
}
```

## 4.2 MTT

```
// N: size * 4
// MOD
const size_t N = 1 << 18;
const int MOD = 1E9 + 7;
struct Complex {
    double a, b;
    Complex() {}
    Complex(double a, double b) : a(a), b(b) {}
    Complex operator + (const Complex &c) const {
        return Complex(a + c.a, b + c.b);
    }
    Complex operator - (const Complex &c) const {
        return Complex(a - c.a, b - c.b);
    }
    Complex operator * (const Complex &c) const {
        return Complex(a * c.a - b * c.b, a * c.b
↪   + b * c.a);
    }
    Complex conj() const {
        return Complex(a, -b);
    }
} w[N];
void prep() {
    const double PI = acos(-1);
    for (int i = 0; i <= N >> 1; ++i) {
        double ang = 2 * i * PI / N;
        w[i] = Complex(cos(ang), sin(ang));
    }
}
struct _ {
    _() { prep(); }
} __;
void fft(Complex a[], int lg) {
    int n = 1 << lg;
    static int rev[N], rev_lg = -1;
    if (rev_lg != lg) {
        for (int i = 0; i < n; ++i)
            rev[i] = rev[i >> 1] >> 1 | ((i & 1)
↪   << lg >> 1);
        rev_lg = lg;
    }
    for (int i = 0; i < n; ++i)
        if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int m = 1, l = 2; m < n; m <<= 1, l <<=
↪   1) {
        static Complex ww[N];
        for (int i = 0, j = 0, step = N / l; i <
↪   m; ++i, j += step)
            ww[i] = w[j];
        Complex *xx = a, *yy = a + m, x, y;
        for (int i = 0, j; i < n; i += l) {
            for (j = 0; j < m; ++j, ++xx, ++yy) {
                x = *xx; y = *yy * ww[j];
                *xx = x + y;
                *yy = x - y;
            }
            xx += m;
            yy += m;
        }
    }
}
void mul(int a[], int b[], int c[], int n, int m)
↪   {
    static Complex d[N], e[N], f[N], g[N];
    int lg = 0;
    while ((1 << lg) < n + m) ++lg;
    int tot = 1 << lg;
    for (int i = 0; i < n; ++i)
        d[i] = Complex(a[i] & 32767, a[i] >> 15);
    for (int i = 0; i < m; ++i)
        e[i] = Complex(b[i] & 32767, b[i] >> 15);
    fft(d, lg); fft(e, lg);
    for (int i = 0; i < tot; ++i) {
        int j = i ? tot - i : 0;
        Complex da = (d[i] + d[j].conj()) *
↪   Complex(.5,  0);
        Complex db = (d[i] - d[j].conj()) *
↪   Complex(0, -.5);
        Complex dc = (e[i] + e[j].conj()) *
↪   Complex(.5,  0);
        Complex dd = (e[i] - e[j].conj()) *
↪   Complex(0, -.5);
        f[j] = da * dc + da * dd * Complex(0, 1);
        g[j] = db * dc + db * dd * Complex(0, 1);
    }
    fft(f, lg); fft(g, lg);
    for (int i = 0; i < n + m - 1; ++i) {
        i64 da = round(f[i].a / tot); da %= MOD;
        i64 db = round(f[i].b / tot); db %= MOD;
        i64 dc = round(g[i].a / tot); dc %= MOD;
        i64 dd = round(g[i].b / tot); dd %= MOD;
        c[i] = (da + ((db + dc) << 15) + (dd <<
↪   30)) % MOD;
    }
}
```

## 4.3 FWT

```
// N: size * 2
const size_t N = 1 << 17;
void div2(Z &x) {
    if (x.v & 1) x.v += MOD;
    x.v >>= 1;
```

```
}
void fwt_and(Z a[], int n, bool rev) {
    for (int m = 1, l = 2; m < n; m <<= 1, l <<=
↪ 1)
        for (int i = 0; i < n; i += l)
            for (int j = 0; j < m; ++j)
                if (rev) a[i + j] -= a[i + j + m];
                else a[i + j] += a[i + j + m];
}
void fwt_or(Z a[], int n, bool rev) {
    for (int m = 1, l = 2; m < n; m <<= 1, l <<=
↪ 1)
        for (int i = 0; i < n; i += l)
            for (int j = 0; j < m; ++j)
                if (rev) a[i + j + m] -= a[i + j];
                else a[i + j + m] += a[i + j];
}
void fwt_xor(Z a[], int n, bool rev) {
    for (int m = 1, l = 2; m < n; m <<= 1, l <<=
↪ 1)
        for (int i = 0; i < n; i += l)
            for (int j = 0; j < m; ++j) {
                Z xx = a[i + j], yy = a[i + j +
↪ m];
                a[i + j] = xx + yy;
                a[i + j + m] = xx - yy;
                if (rev) {
                    div2(a[i + j]);
                    div2(a[i + j + m]);
                }
            }
}
```

## 4.4 BM

```
// N: size * 2
const size_t N = 1E4 + 5;
using Poly = vector<Z>;
namespace Rec {
u64 tmp[N];
void mul(Z a[], Z b[], Z c[], int n, int m) {
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            if ((tmp[i + j] += (u64) a[i].v *
↪ b[j].v) >> 62)
                tmp[i + j] %= MOD;
    for (int i = 0; i < n + m - 1; ++i) {
        c[i] = tmp[i] % MOD; tmp[i] = 0;
    }
}
void get_mod(Z a[], Z b[], Z c[], int n, int m) {
    static Z tc[N];
    copy(a, a + n, tc);
    Z iv = qpow(b[m - 1], MOD - 2);
    for (int i = n; i-- >= m; ) {
        Z mul = tc[i] * iv;
        for (int j = m, k = i; j--; --k)
            tc[k] -= mul * b[j];
    }
    copy(tc, tc + m - 1, c);
}
void _solve(Z a[], Z b[], i64 n, int m) {
    if (n < m - 1) {
        b[n] = 1; return;
    }
    static Z ta[N], tb[N];
    if (n & 1) {
        _solve(a, b, n - 1, m);
        ta[1] = 1;
        mul(b, ta, tb, m, 2);
        get_mod(tb, a, b, m + 1, m);
    } else {
        _solve(a, b, n >> 1, m);
        mul(b, b, tb, m, m);
        get_mod(tb, a, b, (m << 1) - 1, m);
    }
}
Z solve(const Poly &init, const Poly &a, i64 n) {
    int m = a.size();
    static Z ta[N], b[N];
    for (int i = 0; i < m; ++i)
        ta[i] = 0 - a[m - 1 - i];
    ta[m] = 1;
    _solve(ta, b, n, m + 1);
    Z ans = 0;
    for (int i = 0; i < m; ++i)
        ans += init[i] * b[i];
    return ans;
}
}

namespace BM {
Poly& operator += (Poly &p, const Poly &q) {
    if (q.size() > p.size()) p.resize(q.size());
    for (size_t i = 0; i < q.size(); ++i)
        p[i] += q[i];
    return p;
}
Poly operator * (const Poly &p, Z x) {
    Poly ret(p.size());
    for (size_t i = 0; i < p.size(); ++i)
        ret[i] = p[i] * x;
    return ret;
}
Poly solve(const Poly &a) {
    Poly P, R; int cnt = 1;
    for (size_t i = 0; i < a.size(); ++i) {
        Poly tmp = P; tmp.insert(begin(tmp), MOD -
↪ 1);
        Z delta = 0;
        for (size_t j = 0; j < tmp.size(); ++j)
            delta += tmp[j] * a[i - j];
        if (delta.v) {
            vector<Z> t(cnt);
            R.insert(begin(R), begin(t), end(t));
            P += R * (MOD - delta);
            R = tmp * qpow(delta, MOD - 2);
            cnt = 0;
        } else {
            ++cnt;
        }
    }
```

```
    for (size_t i = P.size(); i < a.size(); ++i) {
        Z cur = 0;
        for (size_t j = 0; j < P.size(); ++j)
            cur += a[i - 1 - j] * P[j];
        assert(cur.v == a[i].v);
    }
    return P;
}
}
int main() {
    vector<Z> p(read());
    i64 m = read();
    generate(begin(p), end(p), read);
    Poly P = BM::solve(p);
    for (Z x : P) cout << x << ' ';
    cout << '\n';
    cout << Rec::solve(p, P, m) << '\n';
    return 0;
}
```

# 5 数论

## 5.1 判素数（miller-rabin）

```
i64 Rand() {
    return (i64) rand() * rand() + rand();
};

i64 mul_mod(i64 a, i64 b, i64 mod) {
    i64 tmp = (long double) a * b / mod;
    i64 ret = a * b - tmp * mod;
    while (ret >= mod) ret -= mod;
    while (ret < 0) ret += mod;
    return ret;
};

i64 pow_mod(i64 base, i64 e, i64 mod) {
    i64 ret = 1;
    for (; e; e >>= 1) {
        if (e & 1) ret = mul_mod(ret, base, mod);
        base = mul_mod(base, base, mod);
    }
    return ret;
};

const int pri[] {
    2, 3, 5, 7, 11, 13, 17, 19, 23, 29
};

bool isp(i64 num) {
    for (int x : pri) if (num == x) return true;
    i64 a = num - 1;
    int b = 0;
    while (!(a & 1)) {
        a >>= 1; ++b;
    }
    for (int p : pri) {
        i64 x = pow_mod(p, a, num), y = x;
        for (int i = 0; i < b; ++i) {
            y = mul_mod(x, x, num);
            if (y == 1 && x != 1 && x != num - 1)
                return false;
            x = y;
        }
        if (y != 1) return false;
    }
    return true;
}

vector<i64> fac;

i64 gcd(i64 a, i64 b) {
    return b ? gcd(b, a % b) : a;
}

void rho(i64 n) {
    if (isp(n)) {
        fac.emplace_back(n);
        return;
    }
    while (true) {
        i64 x0 = Rand() % n, x1 = x0, d = 1, c =
↪  Rand() % n, cnt = 0;
        while (d == 1) {
            x0 = (mul_mod(x0, x0, n) + c) % n;
            d = gcd(abs(x1 - x0), n);
            ++cnt;
            if (!(cnt & (cnt - 1))) x1 = x0; //
↪  Floyd 倍增判环
        }
        if (d < n) {
            rho(d); rho(n / d); return;
        }
    }
}
```

## 5.2 二次剩余（Cipolla）

欧拉判定:

$$x^{\frac{p-1}{2}} \equiv \left(\frac{x}{p}\right) \pmod{p}$$

```
// input mod
// method: cipolla(int n)
int mod;
namespace Cipolla {
int omega;
int sqr(int x) {
    return (i64) x * x % mod;
}
struct Number {
    int x, y;
    Number() {}
    Number(int x, int y = 0) : x(x), y(y) {}
    Number operator * (const Number &n) const {
        Number ret;
        ret.x = ((i64) x * n.x + (i64) y * n.y %
↪  mod * omega) % mod;
        ret.y = ((i64) x * n.y + (i64) y * n.x) %
↪  mod;
        return ret;
```

```
    }
    Number& operator *= (const Number &n) {
        return *this = *this * n;
    }
};
Number npow(Number base, int e) {
    Number ret(1);
    for (; e; e >>= 1) {
        if (e & 1) ret *= base;
        base *= base;
    }
    return ret;
}
int get_num(int n) {
    while (true) {
        int x = rand();
        int tmp = (sqr(x) - n) % mod;
        if (tmp < 0) tmp += mod;
        if (qpow(tmp, (mod - 1) / 2) == mod - 1) {
            omega = tmp;
            return x;
        }
    }
}
int cipolla(int n) {
    if (!n) return 0;
    if (qpow(n, (mod - 1) / 2) != 1) {
        return -1;
    }
    int a = get_num(n);
    Number res = npow(Number(a, 1), (mod + 1) /
↪   2);
    assert(!res.y);
    return res.x;
}
}
```

## 5.3 杜教筛

```
// prep_calc[N]: pre-calculated
map<i64, i64> mp;
i64 calc(i64 n) {
    if (n < N) return pre_calc[n];
    if (mp.count(n)) return mp[n];
    i64 ret = 1LL * n * (n + 1) / 2; // 这里改成
↪   (f * g) 的前缀和
    for (i64 l = 2, r; l <= n; l = r) {
        r = n / (n / l) + 1;
        ret -= (r - l) * calc(n / l); // 这里 r -
↪   l 改成 g 在 [l, r] 的和
    }
    return mp[n] = ret;
}
```

## 5.4 min_25

```
const size_t N = 2E5 + 5; // 2 * sqrt(N)

i64 n, lim, val[N];
int id1[N], id2[N];
bool npr[N]; int pri[N], pcnt; Z pg0[N], pg1[N];
Z g0[N], g1[N];

void prep() {
    for (int i = 2; i < (int) N; ++i) {
        if (!npr[i]) {
            pri[++pcnt] = i;
            pg0[pcnt] = pg0[pcnt - 1] - 1;
            pg1[pcnt] = pg1[pcnt - 1] + i;
        }
        for (int j = 1, k; j <= pcnt && (k = i *
↪   pri[j]) < (int) N; ++j) {
            npr[k] = true;
            if (i % pri[j] == 0) break;
        }
    }
}

int get_id(i64 x) {
    return x <= lim ? id1[x] : id2[n / x];
}

Z calc_f(int p, int c) {
    return p ^ c;
}

Z S(i64 n, int x) {
    // 求 \sum f(1 ~ n 中最小质因子 >= pri[x])
    if (n <= 1 || pri[x] > n) return 0;
    Z ret = g0[get_id(n)] + g1[get_id(n)];
    if (x == 1) ret += 2; // #6035 特殊 f(2) = 2 +
↪   1 = 3 != 1
    ret -= pg0[x - 1] + pg1[x - 1];
    // 当前 ret 为 \sum f(1 ~ n 中 >= pri[x] 的质
↪   数)
    for (int k = x; k <= pcnt; ++k) {
        i64 p1 = pri[k], p2 = p1 * pri[k];
        if (p2 > n) break;
        for (int e = 1; p2 <= n; p2 = (p1 = p2) *
↪   pri[k], ++e) {
            ret += S(n / p1, k + 1) *
↪   calc_f(pri[k], e);
            ret += calc_f(pri[k], e + 1);
        }
    }
    return ret;
}

int main() {
    n = read();
    lim = sqrt(n + .5);
    prep();
    int cnt = 0;
    for (i64 i = 1, j; i <= n; i = j + 1) {
        i64 t;
        j = n / (t = val[++cnt] = n / i);
        (t <= lim ? id1[t] : id2[i]) = cnt;
        t %= MOD;
        g0[cnt] = 1 - Z(t);
        g1[cnt] = (t - 1) * (t + 2) / 2 % MOD;
    }
```

```
    for (int i = 1; i <= pcnt; ++i) {
        // 筛掉最小质因子为 pri[i] 的数
        i64 bnd = (i64) pri[i] * pri[i];
        if (bnd > n) break;
        for (int j = 1, id; val[j] >= bnd; ++j) {
            id = get_id(val[j] / pri[i]);
            g0[j] -= (g0[id] - pg0[i - 1]);
            g1[j] -= (g1[id] - pg1[i - 1]) *
↪   pri[i];
        }
    }
    // g[i] = \sum 1~val[i] 中质数
    cout << 1 + S(n, 1) << '\n';
    return 0;
}
```

# 6 线性代数

## 6.1 线性基

```
// N: size
const size_t N = 50;
u64 base[N];
void add(u64 val) {
    for (int i = 49; ~i; --i) if (val >> i & 1)
        if (!base[i]) {
            for (int j = 0; j < i; ++j) if (val >>
↪   j & 1) val ^= base[j];
            base[i] = val;
            for (int j = i + 1; j < 50; ++j) if
↪   (base[j] >> i & 1) base[j] ^= val;
            break;
        } else {
            val ^= base[i];
        }
}
```

## 6.2 矩阵求逆

```
struct Matrix {
    size_t n, m;
    vector<vector<Z>> a;
    Matrix() {}
    Matrix(size_t n, size_t m) : n(n), m(m) {
        a = vector<vector<Z>>(n, vector<Z>(m));
    }
    void do_diag(Z x) {
        for (size_t i = 0; i < n && i < m; ++i)
↪   a[i][i] = x;
    }
    Matrix& operator += (const Matrix &mat) {
        assert(n == mat.n && m == mat.m);
        for (size_t i = 0; i < n; ++i) for (size_t
↪   j = 0; j < m; ++j) a[i][j] += mat.a[i][j];
        return *this;
    }
    Matrix operator + (const Matrix &mat) const {
        Matrix ret = *this; return ret += mat;
    }
    Matrix operator * (const Matrix &mat) const {
        assert(m == mat.n);
        Matrix ret(n, mat.m);
        for (size_t i = 0; i < n; ++i)
            for (size_t j = 0; j < mat.m; ++j)
                for (size_t k = 0; k < m; ++k)
                    ret.a[i][j] += a[i][k] *
↪   mat.a[k][j];
        return ret;
    }
    Matrix& operator *= (const Z &x) {
        for (size_t i = 0; i < n; ++i) for (size_t
↪   j = 0; j < m; ++j) a[i][j] *= x;
        return *this;
    }
    Matrix operator * (const Z &x) const {
        Matrix ret = *this; return ret *= x;
    }
    Matrix& operator *= (const Matrix &mat) {
↪   return *this = *this * mat; }
    Matrix get_inv() const {
        assert(n == m);
        Matrix m = *this, r(n, n); r.do_diag(1);
        for (size_t i = 0; i < n; ++i) {
            int pivot = -1;
            for (size_t j = i; j < n; ++j) if
↪   (m.a[j][i].v && !~pivot) pivot = j;
            assert(~pivot);
            for (size_t j = i; j < n; ++j) {
                swap(m.a[i][j], m.a[pivot][j]);
                swap(r.a[i][j], r.a[pivot][j]);
            }
            Z mul = qpow(m.a[i][i], MOD - 2);
            for (size_t j = 0; j < n; ++j) { // 矩
↪   阵求逆时切勿从 i 开始枚举
                m.a[i][j] *= mul; r.a[i][j] *=
↪   mul;
            }
            for (size_t j = 0; j < n; ++j) {
                if (j == i) continue;
                Z mul = m.a[j][i]; if (!mul.v)
↪   continue;
                for (size_t k = 0; k < n; ++k) {
                    m.a[j][k] -= mul * m.a[i][k];
                    r.a[j][k] -= mul * r.a[i][k];
                }
                assert(!m.a[j][i].v);
            }
        }
        for (size_t i = 0; i < n; ++i) {
            for (size_t j = 0; j < n; ++j)
↪   assert(m.a[i][j].v == (i == j));
        }
        return r;
    }
};

Matrix qpow(Matrix base, int e) {
    Matrix ret(2, 2); ret.do_diag(1);
    for (; e; e >>= 1) {
        if (e & 1) ret *= base;
```

```
        base *= base;
    }
    return ret;
}

ostream& operator << (ostream &os, const Matrix
↪  &mat) {
    for (size_t i = 0; i < mat.n; ++i) {
        for (size_t j = 0; j < mat.m; ++j) os <<
↪  mat.a[i][j] << ' ';
        os << '\n';
    }
    return os;
}
```

# 7 数据结构

## 7.1 左偏树

```
// N
struct Node {
    int lc, rc, val, dis;
    Node() {}
} t[N];
int arr[N], rt[N];
bool del[N];
int merge(int x, int y) {
    if (!x || !y) return x | y;
    if (arr[y] < arr[x]) swap(x, y);
    t[x].rc = merge(t[x].rc, y);
    if (t[t[x].lc].dis < t[t[x].rc].dis)
        swap(t[x].lc, t[x].rc);
    t[x].dis = t[t[x].rc].dis + 1;
    return x;
}
```

## 7.2 LCT

```
// N
const size_t N = 1E5 + 5;
int pa[N], ch[N][2], siz[N], val[N];
bool tag[N];
void update(int x) {
    swap(ch[x][0], ch[x][1]);
    tag[x] ^= 1;
}
void pushdown(int x) {
    if (tag[x]) {
        if (ch[x][0]) update(ch[x][0]);
        if (ch[x][1]) update(ch[x][1]);
        tag[x] = 0;
    }
}
void pushup(int x) {
    siz[x] = siz[ch[x][0]] + val[x] +
↪  siz[ch[x][1]];
}
int getd(int x) {
    return ch[pa[x]][0] == x ? 0 : ch[pa[x]][1] ==
↪  x ? 1 : -1;
}
void rotate(int x) {
    int y = pa[x], z = pa[y], k = getd(x);
    if (~getd(y)) ch[z][getd(y)] = x;
    pa[x] = z; pa[y] = x;
    ch[y][k] = ch[x][k ^ 1];
    ch[x][k ^ 1] = y;
    if (ch[y][k]) pa[ch[y][k]] = y;
    pushup(y);
}
void splay(int x) {
    static int stk[N];
    int y = x, tp = 0;
    stk[++tp] = y;
    while (~getd(y)) stk[++tp] = y = pa[y];
    while (tp) pushdown(stk[tp--]);
    while (~getd(x)) {
        y = pa[x];
        if (~getd(y))
            rotate(getd(x) ^ getd(y) ? x : y);
        rotate(x);
    }
    pushup(x);
}
void access(int x) {
    for (int y = 0; x; x = pa[y = x]) {
        splay(x);
        val[x] += siz[ch[x][1]];
        ch[x][1] = y;
        val[x] -= siz[ch[x][1]];
        pushup(x);
    }
}
void makeroot(int x) {
    access(x);
    splay(x);
    update(x);
}
void link(int x, int y) {
    makeroot(x);
    access(y); splay(y);
    pa[x] = y;
    val[y] += siz[x];
    pushup(y);
}
i64 split(int x, int y) {
    makeroot(y);
    access(x); splay(x);
    // x -> y is now a link from the root
    return (i64) (siz[x] - siz[y]) * siz[y];
}
```

## 7.3 KD-Tree

```
// N
using P = pair<int, int>;
#define fi first
#define se second
const size_t N = 2E5 + 5;
struct Node {
    int xl, yl, xm, ym, xr, yr;
    int lc, rc, pa;
```

```
    i64 sum, val, tag;
    int cnt; bool exist;
    Node() {}
} t[N];
int tot;
P point[N];
map<P, int> mp;
int build(int l, int r, bool d = 0, int pa = 0) {
    if (l > r) return 0;
    int x = ++tot;
    t[x].pa = pa;
    int mid = (l + r) >> 1;
    nth_element(point + l, point + mid, point + r
↪ + 1,
            [&](const P &p, const P &q) {
        P a = p, b = q;
        if (d) swap(a.fi, a.se), swap(b.fi, b.se);
        return a < b;
    });
    mp[point[mid]] = x;
    t[x].xl = t[x].xm = t[x].xr = point[mid].fi;
    t[x].yl = t[x].ym = t[x].yr = point[mid].se;
    if ((t[x].lc = build(l, mid - 1, d ^ 1, x))) {
        int y = t[x].lc;
        chkmin(t[x].xl, t[y].xl); chkmax(t[x].xr,
↪ t[y].xr);
        chkmin(t[x].yl, t[y].yl); chkmax(t[x].yr,
↪ t[y].yr);
    }
    if ((t[x].rc = build(mid + 1, r, d ^ 1, x))) {
        int y = t[x].rc;
        chkmin(t[x].xl, t[y].xl); chkmax(t[x].xr,
↪ t[y].xr);
        chkmin(t[x].yl, t[y].yl); chkmax(t[x].yr,
↪ t[y].yr);
    }
    return x;
}
void pushup(int x) {
    t[x].sum = t[t[x].lc].sum + t[t[x].rc].sum;
    if (t[x].exist) t[x].sum += t[x].val;
}
void update(int x, i64 v) {
    t[x].sum += v * t[x].cnt;
    t[x].val += v;
    t[x].tag += v;
}
void pushdown(int x) {
    if (t[x].tag) {
        if (t[x].lc) update(t[x].lc, t[x].tag);
        if (t[x].rc) update(t[x].rc, t[x].tag);
        t[x].tag = 0;
    }
}
void link_pd(int x) {
    static int stk[N];
    int tp = 0;
    for (; x; x = t[x].pa) stk[++tp] = x;
    while (tp) pushdown(stk[tp--]);
}
void modify(int x, int a, int b, int val) {
    if (!x || t[x].xr < a || t[x].yr < b) return;
    if (t[x].xl >= a && t[x].yl >= b) return
↪ update(x, val);
    pushdown(x);
    if (t[x].xm >= a && t[x].ym >= b) t[x].val +=
↪ val;
    modify(t[x].lc, a, b, val);
    modify(t[x].rc, a, b, val);
    pushup(x);
}
void doit(int x, int y, int d) {
    int u = mp[{x, y}];
    link_pd(u);
    i64 e = t[u].val * d;
    t[u].exist ^= 1;
    for (; u; u = t[u].pa) {
        t[u].cnt += d;
        t[u].sum += e;
    }
    modify(1, x + 1, y + 1, d);
}
void query(int x, int a, int b, i64 &sum, int
↪ &cnt) {
    if (!x || t[x].xl > a || t[x].yl > b) return;
    if (t[x].xr <= a && t[x].yr <= b) {
        sum += t[x].sum;
        cnt += t[x].cnt;
        return;
    }
    pushdown(x);
    if (t[x].xm <= a && t[x].ym <= b &&
↪ t[x].exist) {
        sum += t[x].val;
        cnt += 1;
    }
    query(t[x].lc, a, b, sum, cnt);
    query(t[x].rc, a, b, sum, cnt);
}
```

# 8 图论

## 8.1 点双

```
void dfs1(int u, int p = 0) {
    static int tme = 0, stk[N], tp;
    dfn[u] = low[u] = ++tme;
    stk[++tp] = u;
    int child = 0;
    for (int v: g[u]) {
        if (!dfn[v]) {
            dfs1(v, u); ++child;
            low[u] = min(low[u], low[v]);
            if (low[v] >= dfn[u]) {
                cut[u] = true;
                ++cc;
                do bcc[cc].emplace_back(stk[tp]);
                while (stk[tp--] != v);
                bcc[cc].emplace_back(u);
            }
        } else
```

```
            low[u] = min(low[u], dfn[v]);
        }
        if (!child) {
            cut[u] = true;
            bcc[++cc].emplace_back(u);
        }
    }
```

## 8.2 全局平衡二叉树

```
vector<int> g[];
int siz[], son[], lsiz[];
int pa[], ch[][2];
T val[], sum[];
void dfs1(int u, int p = 0) {
    siz[u] = 1;
    for (int v : g[u]) {
        if (v == p) continue;
        dfs1(v, u);
        siz[u] += siz[v];
        if (siz[v] > siz[son[u]]) son[u] = v;
    }
}
void dfs2(int u, int p = 0) {
    for (int v : g[u]) {
        if (v == p) continue;
        dfs2(v, u);
        if (v == son[u]) continue;
        lsiz[u] += siz[v];
        // val[v] -> val[u]
    }
    sum[u] = val[u];
}
int build(vector<int> &vc, int l, int r) {
    if (l > r) return 0;
    int tot = 0;
    for (int i = l; i <= r; ++i) tot +=
↪   lsiz[vc[i]];
    for (int i = l, sum = 0; i <= r; ++i)
        if ((sum += lsiz[vc[i]]) * 2 >= tot) {
            int x = vc[i];
            if ((ch[x][0] = build(vc, l, i - 1)))
↪   pa[ch[x][0]] = x;
            if ((ch[x][1] = build(vc, i + 1, r)))
↪   pa[ch[x][1]] = x;
            return x;
        }
}

int build(int u) {
    static bool vis[N];
    vector<int> stk;
    for (int v = u; v; v = son[v]) {
        vis[v] = true;
        stk.emplace_back(v);
    }
    int x = build(stk, 0, (int) stk.size() - 1);
    for (int v = u; v; v = son[v])
        for (int w : g[v])
            if (!vis[w]) pa[build(w)] = v;
    return x;
}
int rt;
int build() { rt = build(1); }
void pushup(x) {
    sum[x] = val[x];
    if (ch[x][0]) sum[x] = sum[ch[x][0]] + sum[x];
    if (ch[x][1]) sum[x] = sum[x] + sum[ch[x][1]];
}
void modify(int x) {
    int y;
    while ((x = pa[y = x])) {
        if (ch[x][0] != y && ch[x][1] != y)
            // del sum[y] -> val[x]
        pushup(y);
        if (ch[x][0] != y && ch[x][1] != y)
            // add sum[y] -> val[x]
    }
    pushup(y);
}
```

## 8.3 MCS 求 PEO

```
// 一个图是弦图当且仅当它有 PEO
// input: N
// n: number of vertices
// g: edges

const size_t N = 1E4 + 5;

int n; vector<int> g[N];

int label[N], pos[N], peo[N];
vector<int> que[N];

int main() {
    for (int i = 1; i <= n; ++i) {
        que[0].emplace_back(i);
    }
    int j = 0;
    for (int i = n; i >= 1; --i) {
        int u;
        while (j >= 0) {
            while (!que[j].empty()) {
                u = que[j].back();
                if (pos[u]) {
                    que[j].pop_back();
                } else {
                    break;
                }
            }
            if (!que[j].empty()) break;
            --j;
        }
        assert(j >= 0);
        pos[u] = i; peo[i] = u;
        for (int v : g[u]) {
            if (!pos[v]) {
                ++label[v];
                que[label[v]].emplace_back(v);
                if (label[v] > j) j = label[v];
            }
```

```
        }
    }
}
```

### 8.4 求欧拉回路

```
// input: N, k, graph
// output: print_ans (an euler tour whose length
↪  is \geq k)

int k;
bool vis[N];
vector<int> g[N];
vector<int> ans1, ans2;
void print_ans(const vector<int> &vc) {
    for (int x : vc) cout << x << ' ';
    exit(0);
}
void dfs(int u) {
    vis[u] = true;
    if (ans1.size() >= k) print_ans(ans1);
    for (int v : g[u]) {
        if (vis[v]) continue;
        ans1.emplace_back(u);
        dfs(v);
        ans1.pop_back(); ans2.emplace_back(u);
        if (ans2.size() >= k) {
            reverse(begin(ans2), end(ans2));
            print_ans(ans2);
        }
    }
}
```

### 8.5 SPFA

```
// input: N, n - number of vertices
// output: dis - distance, return - no negative
↪  loops
int dis[N], cnt[N];
bool inque[N];
bool spfa(int n) {
    memset(dis, 0x3f, sizeof dis);
    queue<int> que;
    que.emplace(0);
    dis[0] = 0; inque[0] = true; cnt[0] = 1;
    while (!que.empty()) {
        int u = que.front(); que.pop();
        inque[u] = false;
        for (auto [v, w] : g[u]) {
            if (chkmin(dis[v], dis[u] + w) &&
↪  !inque[v]) {
                que.emplace(v);
                inque[v] = true;
                if (++cnt[v] > n) return false;
            }
        }
    }
    return true;
}
```

## 9 字符串

### 9.1 后缀树组

```
// input: n, s
// output: sa, rnk, hei
// method: init(const string&); calc_sa();
↪  calc_hei();
struct GetSa {
    int n;
    string s;
    vector<int> sa, rnk, hei;
    GetSa() {}
    void init(const string &_s) {
        s = _s; n = _s.size();
    }
    void calc_sa() {
        sa.resize(n);
        rnk.resize(n);
        vector<int> x(n), y(n);
        for (int i = 0; i < n; ++i) x[i] = s[i];
        int tot = *max_element(ALL(x)) + 1;
        vector<int> cnt(tot);
        for (int i = 0; i < n; ++i) ++cnt[x[i]];
        partial_sum(ALL(cnt), begin(cnt));
        for (int i = 0; i < n; ++i)
            sa[--cnt[x[i]]] = i;
        for (int l = 1; ; l <<= 1) {
            vector<int> cnt(tot);
            int p = n;
            for (int i = n - l; i < n; ++i) y[--p]
↪  = i;
            for (int i = 0; i < n; ++i)
                if (sa[i] >= l) y[--p] = sa[i] -
↪  l;
            for (int i = 0; i < n; ++i)
↪  ++cnt[x[y[i]]];
            partial_sum(ALL(cnt), begin(cnt));
            for (int i = 0; i < n; ++i)
                sa[--cnt[x[y[i]]]] = y[i];
            y[sa[0]] = 0;
            for (int i = 1; i < n; ++i)
                y[sa[i]] = y[sa[i-1]] +
                    (x[sa[i-1]] < x[sa[i]] ||
↪  (sa[i]+l < n && (sa[i-1]+l >= n ||
↪  x[sa[i-1]+l] < x[sa[i]+l])));
            tot = y[sa.back()] + 1;
            x.swap(y);
            if (tot == n) break;
        }
        copy(ALL(x), begin(rnk));
    }
    void calc_hei() {
        hei.resize(n);
        for (int i = 0, j = 0; i < n; ++i) {
            if (!rnk[i]) continue;
            int ii = sa[rnk[i]-1];
            if (j) --j;
            while (ii+j < n && i+j < n && s[ii+j]
↪  == s[i+j]) ++j;
            hei[rnk[i]] = j;
```

```
        }
    }
};
```

## 9.2 后缀自动机

```
// N: length of string
// AL: alphabet size
// method: add(), build()
namespace Sam {
const size_t V = N << 1;
const size_t AL = 26;
int ch[V][AL], par[V], len[V], pos[V], tot = 1,
↪ lst = 1, s[N];
bool ed[V];
void add(int po, int c) {
    int p = lst, np = ++tot;
    s[po] = c;
    len[np] = len[lst] + 1;
    pos[np] = po;
    ed[np] = true;
    for (; p && !ch[p][c]; p = par[p])
        ch[p][c] = np;
    if (p) {
        int q = ch[p][c];
        if (len[p] + 1 == len[q]) {
            par[np] = q;
        } else {
            int nq = ++tot;
            len[nq] = len[p] + 1;
            par[nq] = par[q];
            pos[nq] = pos[q];
            memcpy(ch[nq], ch[q], sizeof ch[q]);
            for (; p && ch[p][c] == q; p = par[p])
                ch[p][c] = nq;
            par[q] = par[np] = nq;
        }
    } else {
        par[np] = 1;
    }
    lst = np;
}
int fch[V][AL], cnt;
void dfs(int u = 1) {
    if (!u) return;
    if (ed[u]) {
        ++cnt;
        sa[cnt] = pos[u];
        rnk[pos[u]] = cnt;
    }
    for (int v : fch[u]) dfs(v);
}
void build() {
    for (int i = 2; i <= tot; ++i)
        fch[par[i]][s[pos[i] + len[par[i]]]] = i;
    dfs();
}
```

## 9.3 Manacher

```
void manacher(int n, char s[], int f[]) {
    int id = 0, r = 0;
    for (int i = 1; i < n; ++i) {
        f[i] = r > i ? min(f[2 * id - i], r - i) :
↪ 1;
        while (f[i] <= i && i + f[i] < n && s[i +
↪ f[i]] == s[i - f[i]])
            ++f[i];
        if (i + f[i] > r) { id = i; r = i + f[i];
↪ }
    }
}
```

## 9.4 回文自动机

```
// N: length of string
// method: prep, add
namespace PAM {
const size_t AL = 26;
int n, s[N];
int tot, lst, ch[N][AL], par[N], len[N], dep[N];
void prep() {
    par[0] = par[1] = 1;
    s[0] = len[1] = -1;
    lst = tot = 1;
}
int get_link(int x) {
    for (; s[n] != s[n - len[x] - 1]; x = par[x])
↪ {}
    return x;
}
int add(int c) {
    s[++n] = c;
    int p = get_link(lst);
    if (!ch[p][c]) {
        int np = ++tot;
        len[np] = len[p] + 2;
        par[np] = ch[get_link(par[p])][c];
        dep[np] = dep[par[np]] + 1;
        ch[p][c] = np;
    }
    return dep[lst = ch[p][c]];
}
}
```

## 9.5 Lyndon 分解

```
// input: n, s[]
void lyndon() {
    for (int i = 0; i < n; ) {
        int j = i, k = i + 1;
        for (; k < n && s[j] <= s[k]; ++k)
            j = s[j] < s[k] ? i : j + 1;
        while (i <= j) i += k - j; // right pos
    }
    return 0;
}
```

### 9.6 Z Function

```
void z_func(string s, int f[]) {
    int l = 0, r = 0;
    for (int i = 1; i < (int) s.size(); ++i) {
        f[i] = i < r ? min(r - i, f[i - l]) : 0;
        while (i + f[i] < (int) s.size() &&
            s[f[i]] == s[i + f[i]]) ++f[i];
        if (i + f[i] > r) r = (l = i) + f[i];
    }
}
```

## 10 其他

### 10.1 网络流（ISAP）

```
// N: vertices, M: edges
// method: add_edge(int u, int v, int cap),
↪  maxflow(int s, int t)
struct Maxflow {
    struct Edge {
        int to, cap, nxt;
        Edge(int to = 0, int cap = 0, int nxt =
↪  0):
            to(to), cap(cap), nxt(nxt) {}
    } e[M];
    int head[N], cur[N], d[N], f[N], tot = 1;
    int n, s, t;
    void add_edge(int u, int v, int cap) {
        e[++tot] = Edge(v, cap, head[u]); head[u]
↪  = tot;
        e[++tot] = Edge(u,  0,  head[v]); head[v]
↪  = tot;
    }
    int dfs(int v, int fl = INF) {
        if (v == t) return fl;
        int ret = 0;
        for (int &i = cur[v]; i; i = e[i].nxt) {
            if (e[i].cap && d[e[i].to] + 1 ==
↪  d[v]) {
                int tmp = dfs(e[i].to, min(fl,
↪  e[i].cap));
                ret += tmp; fl -= tmp;
                e[i].cap -= tmp;
                e[i ^ 1].cap += tmp;
                if (!fl) return ret;
            }
        }
        cur[v] = head[v];
        if (!(--f[d[v]])) d[s] = n;
        ++f[++d[v]];
        return ret;
    }
    int maxflow(int _s, int _t) {
        n = _n; s = _s; t = _t;
        memset(cur, 0, sizeof cur);
        memset(d, 0, sizeof d);
        memset(f, 0, sizeof f);
        f[0] = n;
        int ret = 0;
        while (d[s] < n) ret += dfs(s);
        return ret;
    }
} flow;
```

### 10.2 网络流（HLPP）

```
// N: vertices, M: edges
// method: add_edge(int u, int v, i64 cap),
↪  maxflow(int s, int t)
struct Maxflow {
    int n;
    struct Edge {
        int to; i64 cap; int nxt;
        Edge() {}
        Edge(int to, i64 cap, int nxt) : to(to),
↪  cap(cap), nxt(nxt) {}
    } e[M << 1];
    int tot_e, head[N], cur[N], deg[N];
    Maxflow() {
        memset(this, 0, sizeof *this);
        tot_e = 1;
    }
    void add_edge(int u, int v, i64 cap) {
        e[++tot_e] = {v, cap, head[u]}; head[u] =
↪  tot_e;
        e[++tot_e] = {u,  0 , head[v]}; head[v] =
↪  tot_e;
        ++deg[u]; ++deg[v];
    }
    int cnt_upd_h, max_h, h[N], cnt[N]; i64
↪  rest[N];
    vector<int> vc1[N], vc2[N];
    void update_h(int v, int nh) {
        ++cnt_upd_h;
        if (h[v] < INF) --cnt[h[v]];
        h[v] = nh;
        if (h[v] == INF) return;
        ++cnt[h[v]];
        max_h = h[v];
        vc1[h[v]].emplace_back(v);
        if (rest[v]) vc2[h[v]].emplace_back(v);
    }
    void relabel(int t) {
        cnt_upd_h = max_h = 0;
        fill(h, h + n + 1, INF);
        fill(cnt, cnt + n + 1, 0);
        for (int i = 0; i <= max_h; ++i) {
            vc1[i].clear();
            vc2[i].clear();
        }
        queue<int> que;
        que.emplace(t);
        update_h(t, 0);
        while (!que.empty()) {
            int u = que.front(); que.pop();
            for (int i = head[u]; i; i = e[i].nxt)
↪  {
                int v = e[i].to;
                if (h[u] + 1 < h[v] && e[i ^
↪  1].cap) {
                    update_h(v, h[u] + 1);
```

```
                    que.emplace(v);
                }
            }
        }
    }
    void push(int i) {
        int u = e[i ^ 1].to, v = e[i].to;
        i64 w = min((i64) rest[u], e[i].cap);
        if (!w) return;
        if (!rest[v]) vc2[h[v]].emplace_back(v);
        e[i].cap -= w; e[i ^ 1].cap += w;
        rest[u] -= w; rest[v] += w;
    }
    void push_flow(int u) {
        int nh = INF;
        for (int &i = cur[u], j = 0; j < deg[u]; i
↪  = e[i].nxt, ++j) {
            if (!i) i = head[u];
            int v = e[i].to;
            if (e[i].cap) {
                if (h[u] == h[v] + 1) {
                    push(i);
                    if (!rest[u]) return;
                } else if (nh > h[v] + 1) {
                    nh = h[v] + 1;
                }
            }
        }
        if (cnt[h[u]] > 1) {
            update_h(u, nh);
        } else {
            for (int i = h[u]; i <= max_h; ++i) {
                for (int v : vc1[i]) update_h(v,
↪  INF);
                vc1[i].clear();
            }
        }
    }
    int maxflow(int s, int t, int lim = 10000) {
        rest[s] = 1E18;
        relabel(t);
        for (int i = head[s]; i; i = e[i].nxt)
↪  push(i);
        for (int &i = max_h; ~i; --i) {
            while (!vc2[i].empty()) {
                int u = vc2[i].back();
                vc2[i].pop_back();
                if (h[u] != i) continue;
                push_flow(u);
                if (cnt_upd_h > lim) relabel(t);
            }
        }
        return rest[t];
    }
} flow;
```

## 10.3 二分图最大权匹配（KM）

```
int n;
// n, N 两侧点数
// 需定义 INF
namespace KM {
i64 arr[N][N];
bool visl[N], visr[N];
int matchl[N], matchr[N], matcht[N];
i64 slack[N], expl[N], expr[N];
void change_match(int v) {
    for (; v; swap(v, matchl[matcht[v]])) {
        matchr[v] = matcht[v];
    }
}
void find_path(int s) {
    queue<int> que;
    que.emplace(s);
    visl[s] = true;
    while (true) {
        while (!que.empty()) {
            int l = que.front();
            que.pop();
            for (int r = 1; r <= n; ++r) {
                if (visr[r]) continue;
                i64 gap = expl[l] + expr[r] -
↪  arr[l][r];
                if (gap > slack[r]) continue;
                matcht[r] = l;
                if (gap == 0) {
                    if (!matchr[r]) return
↪  change_match(r);
                    que.emplace(matchr[r]);
                    visl[matchr[r]] = visr[r] =
↪  true;
                } else {
                    slack[r] = gap;
                }
            }
        }
        int v = -1;
        for (int r = 1; r <= n; ++r) {
            if (!visr[r] && (!~v || slack[r] <
↪  slack[v])) {
                v = r;
            }
        }
        assert(~v);
        i64 delta = slack[v];
        for (int i = 1; i <= n; ++i) {
            if (visl[i]) expl[i] -= delta;
            if (visr[i]) expr[i] += delta; else
↪  slack[i] -= delta;
        }
        if (!matchr[v]) return change_match(v);
        que.emplace(matchr[v]);
        visl[matchr[v]] = visr[v] = true;
    }
}
i64 km() {
    for (int l = 1; l <= n; ++l) {
        for (int r = 1; r <= n; ++r) {
            expl[l] = max(expl[l], arr[l][r]);
        }
    }
    for (int l = 1; l <= n; ++l) {
```

```
        fill(slack + 1, slack + n + 1, INF);
        memset(visl, 0, sizeof(bool) * (n + 1));
        memset(visr, 0, sizeof(bool) * (n + 1));
        memset(matcht, 0, sizeof(int) * (n + 1));
        find_path(l);
    }
    i64 ans = 0;
    for (int i = 1; i <= n; ++i) ans +=
↪   arr[i][matchl[i]];
    return ans;
}
}
```

## 10.4 模拟退火

```
void simulateAnneal() {
    const double INIT_TEMP = 2e5;
    const double DELTA = 0.997;
    const double EPS = 1e-14;
    double curx = ansx, cury = ansy;
    for (double temp = INIT_TEMP; temp > EPS; temp
↪   *= DELTA) {
        double xx = curx + ((rand() << 1) -
↪   RAND_MAX) * temp;
        double yy = cury + ((rand() << 1) -
↪   RAND_MAX) * temp;
        double cure = calcEnergy(xx, yy);
        double diff = cure - anse;
        if (diff < 0) {
            ansx = curx = xx;
            ansy = cury = yy;
            anse = cure;
        } else if (exp(-diff / temp) * RAND_MAX >
↪   rand()) {
            curx = xx;
            cury = yy;
        }
    }
}
```