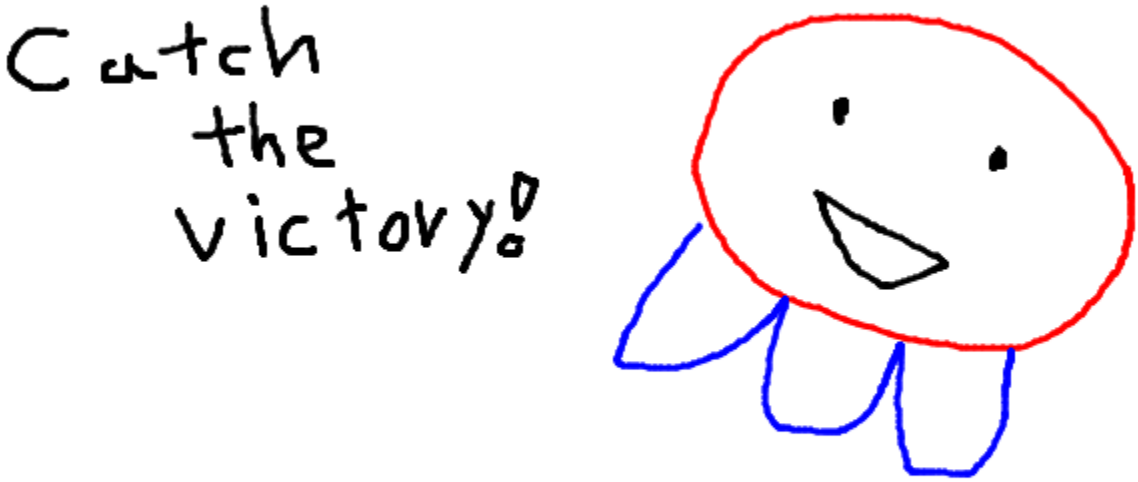
Gifted Infants(The University of Tokyo)



- CLionを起動
- .bashrcを写経
- CLionプロジェクトを作る、~/Desktop/Programs
- CMakeLists.txtに追加 : add_compiler_options(-Wall -Wextra -Wshadow -D_GLIBCXX_DEBUG -ftrapv)

```
cd cmake-build-debug
touch check.sh
chmod +x ./check.sh
mkdir tests
```

- check.shを写経

- geditの設定

```
edit -> preferences
line number
tab width: 4
auto indentation
```

- main.cppにテンプレートを写経
- A.cpp B.cpp ... を作成
- サンプルをダウンロード、testsの中に移動

.bashrc

```
setxkbmap -option ctrl:nocaps
alias c='tput reset'
xmodmap -e 'keycode 94=Shift_L'
```

check.sh

```
make $1
for f in tests/*$1*.in; do
    echo '#### Start ' $f
    ./$1 < $f
done
```

テンプレート

```
//#undef LOCAL
#include <bits/stdc++.h>

using namespace std;
using uint = unsigned int;
using ll = long long;
using ull = unsigned long long;
template<class T> using V = vector<T>;
template<class T> using VV = V<V<T>>;
constexpr ll TEN(int n) { return (n == 0) ? 1 : 10 * TEN(n-1); }
#define FOR(i, a, b) for(int i=(int)(a);i<(int)(b);i++)
#define rep(i,N) for(int i=0;i<(int)(N);i++)
#define rep1(i,N) for(int i=1;i<=(int)(N);i++)
#define fs first
#define sc second
#define eb emplace_back
#define pb eb
#define all(x) x.begin(),x.end()
template<class T, class U> void chmin(T& t, const U& u) { if (t > u) t = u; }
template<class T, class U> void chmax(T& t, const U& u) { if (t < u) t = u; }

#ifdef LOCAL
#define show(x) cerr << __LINE__ << " : " << #x << " = " << (x) << endl
#else
#define show(x) true
#endif

template <class T, class U>
ostream& operator<<(ostream& os, const pair<T, U>& p) {
    return os << "P(" << p.first << ", " << p.second << ")";
}

template <class T> ostream& operator<<(ostream& os, const V<T>& v) {
    os << "[";
    for (auto d : v) os << d << ", ";
    return os << "]";
}

// cin.tie(nullptr);
// ios::sync_with_stdio(false);
// cout << fixed << setprecision(20);
```

bit演算

```
// bit op
int popcnt(uint x) { return __builtin_popcount(x); }
int popcnt(ull x) { return __builtin_popcountll(x); }
int bsr(uint x) { return 31 - __builtin_clz(x); }
int bsr(ull x) { return 63 - __builtin_clzll(x); }
int bsf(uint x) { return __builtin_ctz(x); }
int bsf(ull x) { return __builtin_ctzll(x); }
```

ツール群

ストップウォッチ

```
struct StopWatch {
    bool f = false;
    clock_t st;
    void start() {
        f = true;
        st = clock();
    }
    int msecs() {
        assert(f);
        return (clock()-st)*1000 / CLOCKS_PER_SEC;
    }
};
```

乱数

```
ll rand_int(ll l, ll r) { //[l, r]
    static random_device rd;
    static mt19937 gen(rd());
    return uniform_int_distribution<ll>(l, r)(gen);
}
```

int128

```
istream &operator>>(istream &i, __int128 &x) {
    x = 0;
    string s;
    i >> s;
    int N = int(s.size()), it = 0;
    if (s[0] == '-') it++;
    for (; it < N; it++) x = (x * 10 + s[it] - '0');
    if (s[0] == '-') x = -x;
    return i;
}

ostream &operator<<(ostream &o, __int128 x) {
    if (x == 0) return o << 0;
    if (x < 0) o << '-', x = -x;
    deque<int> ds;
    while (x) ds.push_front(x % 10), x /= 10;
    for (int d: ds) o << d;
    return o;
}
```

stack

```
// stack sizeを拡張する 64bit用
int main() {
    static ll eord, enew;
    const int sz = 256*1024*1024;
    static unsigned char *p = (unsigned char*)malloc(sz);
    enew = ((ll)(p+sz-1) & ~0xff);
    __asm__ volatile("mov %%rsp, %0" : "=r"(eord));
    __asm__ volatile("mov %0, %%rsp" : : "r"(enew));
    main2();
    __asm__ volatile("mov %0, %%rsp" : : "r"(eord));
    return 0;
}
```

数学

数学

```
//binary gcd
ll gcd(ll _a, ll _b) {
    ull a = abs(_a), b = abs(_b);
    if (a == 0) return b;
    if (b == 0) return a;
    int shift = bsf(a|b);
    a >>= bsf(a);
    do {
        b >>= bsf(b);
        if (a > b) swap(a, b);
        b -= a;
    } while (b);
    return (a << shift);
}

/// g:gcd(a, b), ax+by=g
struct EG { ll g, x, y; };
EG ext_gcd(ll a, ll b) {
    if (b == 0) {
        if (a >= 0) return EG{a, 1, 0};
        else return EG{-a, -1, 0};
    } else {
        auto e = ext_gcd(b, a % b);
        return EG{e.g, e.y, e.x - a / b * e.y};
    }
}

ll inv_mod(ll x, ll md) {
    auto z = ext_gcd(x, md).x;
    return (z % md + md) % md;
}

template<class T, class U>
T pow_mod(T x, U n, T md) {
    T r = 1 % md;
    x %= md;
    while (n) {
        if (n & 1) r = (r * x) % md;
        x = (x * x) % md;
        n >>= 1;
    }
    return r;
}

// (rem, mod)
pair<ll, ll> crt(const V<ll>& b, const V<ll>& c) {
    int n = int(b.size());
    ll r = 0, m = 1;
    for (int i = 0; i < n; i++) {
        auto eg = ext_gcd(m, c[i]);
        ll g = eg.g, im = eg.x;
        if ((b[i] - r) % g) return {0, -1};
        ll tmp = (b[i] - r) / g * im % (c[i] / g);
        r += m * tmp;
        m *= c[i] / g;
    }
    return {(r % m + m) % m, m};
}
```

ModInt

```
template <uint MD> struct ModInt {
```

```
    using M = ModInt;
    const static M G;
    uint v;
    ModInt(ll _v = 0) { set_v(_v % MD + MD); }
    M& set_v(uint _v) {
        v = (_v < MD) ? _v : _v - MD;
        return *this;
    }
    explicit operator bool() const { return v != 0; }
    M operator-() const { return M() - *this; }
    M operator+(const M& r) const { return M().set_v(v + r.v); }
    M operator-(const M& r) const { return M().set_v(v + MD - r.v); }
    M operator*(const M& r) const { return M().set_v(ull(v) * r.v % MD); }
    M operator/(const M& r) const { return *this * r.inv(); }
    M& operator+=(const M& r) { return *this = *this + r; }
    M& operator-=(const M& r) { return *this = *this - r; }
    M& operator*=(const M& r) { return *this = *this * r; }
    M& operator/=(const M& r) { return *this = *this / r; }
    bool operator==(const M& r) const { return v == r.v; }
    M pow(ll n) const {
        M x = *this, r = 1;
        while (n) {
            if (n & 1) r *= x;
            x *= x;
            n >>= 1;
        }
        return r;
    }
    M inv() const { return pow(MD - 2); }
    friend ostream& operator<<(ostream& os, const M& r) { return os << r.v; }
};
// using Mint = ModInt<998244353>;
// template<> const Mint Mint::G = Mint(3);
```

FFT

```
using D = double;
const D PI = acos(D(-1));
using Pc = complex<D>;

void fft(bool type, V<Pc>& a) {
    int n = int(a.size()), s = 0;
    while ((1 << s) < n) s++;
    assert(1 << s == n);

    static V<Pc> ep[30];
    if (!ep[s].size()) {
        for (int i = 0; i < n; i++) {
            ep[s].push_back(polar<D>(1, i * 2 * PI / n));
        }
    }
    V<Pc> b(n);
    for (int i = 1; i <= s; i++) {
        int w = 1 << (s - i);
        for (int y = 0; y < n / 2; y += w) {
            Pc now = ep[s][y];
            if (type) now = conj(now);
            for (int x = 0; x < w; x++) {
                auto l = a[y << 1 | x];
                auto u = now, v = a[y << 1 | x | w];
                auto r = Pc(u.real() * v.real() - u.imag() * v.imag(),
                            u.real() * v.imag() + u.imag() * v.real());
                b[y | x] = l + r;
                b[y | x | n >> 1] = l - r;
            }
        }
        swap(a, b);
    }
}

V<Pc> multiply(const V<Pc>& a, const V<Pc>& b) {
    int A = int(a.size()), B = int(b.size());
    if (!A || !B) return {};
    int lg = 0;
    while ((1 << lg) < A + B - 1) lg++;
    int N = 1 << lg;
    V<Pc> ac(N), bc(N);
    for (int i = 0; i < A; i++) ac[i] = a[i];
    for (int i = 0; i < B; i++) bc[i] = b[i];
    fft(false, ac);
    fft(false, bc);
    for (int i = 0; i < N; i++) {
        ac[i] *= bc[i];
    }
    fft(true, ac);
    V<Pc> c(A + B - 1);
    for (int i = 0; i < A + B - 1; i++) {
        c[i] = ac[i] / D(N);
    }
    return c;
}

V<D> multiply(const V<D>& a, const V<D>& b) {
    int A = int(a.size()), B = int(b.size());
    if (!A || !B) return {};
    int lg = 0;
    while ((1 << lg) < A + B - 1) lg++;
    int N = 1 << lg;
    V<Pc> d(N);
    for (int i = 0; i < N; i++) d[i] = Pc(i < A ? a[i] : 0, i < B ? b[i] : 0);
    fft(false, d);
    for (int i = 0; i < N / 2 + 1; i++) {
        auto j = i ? (N - i) : 0;
        Pc x = Pc(d[i].real() + d[j].real(), d[i].imag() - d[j].imag());
        Pc y = Pc(d[i].imag() + d[j].imag(), -d[i].real() + d[j].real());
        d[i] = x * y / D(4);
        if (i != j) d[j] = conj(d[i]);
    }
    fft(true, d);
    V<D> c(A + B - 1);
    for (int i = 0; i < A + B - 1; i++) {
        c[i] = d[i].real() / N;
    }
    return c;
}

template <class Mint, int K = 3, int SHIFT = 11>
V<Mint> multiply(const V<Mint>& a, const V<Mint>& b) {
    int A = int(a.size()), B = int(b.size());
    if (!A || !B) return {};
    int lg = 0;
    while ((1 << lg) < A + B - 1) lg++;
    int N = 1 << lg;

    VV<Pc> x(K, V<Pc>(N)), y(K, V<Pc>(N));
    for (int ph = 0; ph < K; ph++) {
        V<Pc> z(N);
        for (int i = 0; i < N; i++) {
            D nx = 0, ny = 0;
            if (i < A) nx = (a[i].v >> (ph * SHIFT)) & ((1 << SHIFT) - 1);
            if (i < B) ny = (b[i].v >> (ph * SHIFT)) & ((1 << SHIFT) - 1);
            z[i] = Pc(nx, ny);
        }
        fft(false, z);
        for (int i = 0; i < N; i++) {
            z[i] *= 0.5;
        }
        for (int i = 0; i < N; i++) {
            int j = (i) ? N - i : 0;
            x[ph][i] = Pc(z[i].real() + z[j].real(), z[i].imag() - z[j].imag());
            y[ph][i] =
                Pc(z[i].imag() + z[j].imag(), -z[i].real() + z[j].real());
        }
    }
    VV<Pc> z(K, V<Pc>(N));
    for (int xp = 0; xp < K; xp++) {
        for (int yp = 0; yp < K; yp++) {
            int zp = (xp + yp) % K;
            for (int i = 0; i < N; i++) {
                if (xp + yp < K) {
                    z[zp][i] += x[xp][i] * y[yp][i];
                } else {
                    z[zp][i] += x[xp][i] * y[yp][i] * Pc(0, 1);
                }
            }
        }
    }
    for (int ph = 0; ph < K; ph++) {
        fft(true, z[ph]);
    }
    V<Mint> c(A + B - 1);
    Mint base = 1;
    for (int ph = 0; ph < 2 * K - 1; ph++) {
        for (int i = 0; i < A + B - 1; i++) {
            if (ph < K) {
                c[i] += Mint(ll(round(z[ph][i].real() / N))) * base;
            } else {
                c[i] += Mint(ll(round(z[ph - K][i].imag() / N))) * base;
            }
        }
        base *= 1 << SHIFT;
    }
    return c;
}
```

NFT

```
template <class Mint> void nft(bool type, V<Mint>& a) {
    int n = int(a.size()), s = 0;
    while ((1 << s) < n) s++;
    assert(1 << s == n);

    static V<Mint> ep, iep;
    while (int(ep.size()) <= s) {
        ep.push_back(Mint::G.pow(Mint(-1).v / (1 << ep.size())));
        iep.push_back(ep.back().inv());
    }
    V<Mint> b(n);
    for (int i = 1; i <= s; i++) {
        int w = 1 << (s - i);
        Mint base = type ? iep[i] : ep[i], now = 1;
        for (int y = 0; y < n / 2; y += w) {
            for (int x = 0; x < w; x++) {
                auto l = a[y << 1 | x];
                auto r = now * a[y << 1 | x | w];
                b[y | x] = l + r;
                b[y | x | n >> 1] = l - r;
            }
            now *= base;
        }
        swap(a, b);
    }
}

template <class Mint> V<Mint> multiply(const V<Mint>& a, const V<Mint>& b) {
    int n = int(a.size()), m = int(b.size());
    if (!n || !m) return {};
    if (min(n, m) <= 8) {
        V<Mint> ans(n + m - 1);
        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++) ans[i + j] += a[i] * b[j];
        return ans;
```

```
    }
    int lg = 0;
    while ((1 << lg) < n + m - 1) lg++;
    int z = 1 << lg;
    auto a2 = a, b2 = b;
    a2.resize(z);
    b2.resize(z);
    nft(false, a2);
    nft(false, b2);
    for (int i = 0; i < z; i++) a2[i] *= b2[i];
    nft(true, a2);
    a2.resize(n + m - 1);
    Mint iz = Mint(z).inv();
    for (int i = 0; i < n + m - 1; i++) a2[i] *= iz;
    return a2;
}
```

Frac

```
template <class I> struct Frac {
    I a, b;  // a / b
    Frac(I _a = 0) : a(_a), b(1) {}
    Frac(I _a, I _b) {
        I g = gcd(_a, _b);
        if (_b < 0) g = -g;
        a = _a / g;
        b = _b / g;
    }
    Frac operator-() const {
        Frac f;
        f.a = -a;
        f.b = b;
        return f;
    }
    Frac operator+(const Frac& r) const { return {r.b * a + b * r.a, b * r.b}; }
    Frac operator-(const Frac& r) const { return *this + (-r); }
    Frac operator*(const Frac& r) const { return {a * r.a, b * r.b}; }
    Frac operator/(const Frac& r) const { return {a * r.b, b * r.a}; }
    Frac& operator+=(const Frac& r) { return *this = *this + r; }
    Frac& operator-=(const Frac& r) { return *this = *this - r; }
    Frac& operator*=(const Frac& r) { return *this = *this * r; }
    Frac& operator/=(const Frac& r) { return *this = *this / r; }
    bool operator<(const Frac& r) const { return a * r.b < b * r.a; }
    bool operator>(const Frac& r) const { return r < *this; }
    bool operator<=(const Frac& r) const { return !(r < *this); }
    bool operator>=(const Frac& r) const { return !(*this < r); }
    bool operator==(const Frac& r) const {
        return !(*this < r) && !(r < *this);
    }
    bool operator!=(const Frac& r) const { return !(*this == r); }

    static Frac rec(Frac x, Frac y, Frac l, Frac r) {
        auto flip = [&](Frac& f) { f = Frac(1) - f; };
        auto cross = [&](const Frac& f, const Frac& g) {
            return f.a * g.b - f.b * g.a;
        };
        Frac m = {l.a + r.a, l.b + r.b};
        if (x < m && m < y) return m;
        bool s = !(x < m);
        if (s) {
            flip(l);
            flip(r);
            flip(m);
            flip(x);
            flip(y);
            swap(l, r);
            swap(x, y);
        }
        I k = cross(r, y) / cross(y, l) + 1;
        Frac p = {k * l.a + r.a, k * l.b + r.b};
        if (x < p) {
            if (s) flip(p);
            return p;
        }
        Frac q = rec(x, y, p, {(k - 1) * l.a + r.a, (k - 1) * l.b + r.b});
        if (s) flip(q);
        return q;
    }
    static Frac in_bet(Frac x, Frac y) {
        if (y < x) swap(x, y);
        Frac ret;
        I num = x.a >= 0 ? x.a / x.b : -((x.b - 1 - x.a) / x.b);
        x.a -= x.b * num;
        y.a -= y.b * num;
        if (Frac{1, 1} < y)
            ret = Frac{1, 1};
        else
            ret = rec(x, y, Frac{0, 1}, Frac{1, 1});
        ret.a += ret.b * num;
        return ret;
    }
};
```

Prime

```
bool is_prime(ll n) {
    if (n <= 1) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;
    ll d = n - 1;
    while (d % 2 == 0) d /= 2;
    for (ll a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n <= a) break;
        ll t = d;
        ll y = pow_mod<__int128_t>(a, t, n);  // over
        while (t != n - 1 && y != 1 && y != n - 1) {
            y = __int128_t(y) * y % n;  // flow
            t <<= 1;
        }
        if (y != n - 1 && t % 2 == 0) {
            return false;
        }
    }
    return true;
}

ll pollard_single(ll n) {
    auto f = [&](ll x) { return (__int128_t(x) * x + 1) % n; };
    if (is_prime(n)) return n;
    if (n % 2 == 0) return 2;
    ll st = 0;
    while (true) {
        st++;
        ll x = st, y = f(x);
        while (true) {
            ll p = gcd((y - x + n), n);
            if (p == 0 || p == n) break;
            if (p != 1) return p;
            x = f(x);
            y = f(f(y));
        }
    }
}

V<ll> pollard(ll n) {
    if (n == 1) return {};
    ll x = pollard_single(n);
    if (x == n) return {x};
    V<ll> le = pollard(x);
    V<ll> ri = pollard(n / x);
    le.insert(le.end(), ri.begin(), ri.end());
    return le;
}

ll primitive_root(ll p) {
    auto v = pollard(p - 1);
    while (true) {
        ll g = rand_int(1, p - 1);  //[1, p-1]
        bool ok = true;
        for (auto d : v) {
            ll f = (p - 1) / d;
            if (pow_mod<__int128_t>(g, f, p) == 1) {
                ok = false;
                break;
            }
        }
        if (ok) return g;
    }
}
```

Poly

```
template <class D> struct Poly {
    V<D> v;
    Poly(const V<D>& _v = {}) : v(_v) { shrink(); }
    void shrink() {
        while (v.size() && !v.back()) v.pop_back();
    }

    int size() const { return int(v.size()); }
    D freq(int p) const { return (p < size()) ? v[p] : D(0); }

    Poly operator+(const Poly& r) const {
        auto n = max(size(), r.size());
        V<D> res(n);
        for (int i = 0; i < n; i++) res[i] = freq(i) + r.freq(i);
        return res;
    }
    Poly operator-(const Poly& r) const {
        int n = max(size(), r.size());
        V<D> res(n);
        for (int i = 0; i < n; i++) res[i] = freq(i) - r.freq(i);
        return res;
    }
    Poly operator*(const Poly& r) const { return {multiply(v, r.v)}; }
    Poly operator*(const D& r) const {
        int n = size();
        V<D> res(n);
        for (int i = 0; i < n; i++) res[i] = v[i] * r;
        return res;
    }
    Poly operator/(const D &r) const{
        return *this * r.inv();
    }
    Poly operator/(const Poly& r) const {
        if (size() < r.size()) return {{}};
        int n = size() - r.size() + 1;
        return (rev().pre(n) * r.rev().inv(n)).pre(n).rev();
    }
    Poly operator%(const Poly& r) const { return *this - *this / r * r; }
    Poly operator<<(int s) const {
        V<D> res(size() + s);
        for (int i = 0; i < size(); i++) res[i + s] = v[i];
        return res;
    }
    Poly operator>>(int s) const {
        if (size() <= s) return Poly();
        V<D> res(size() - s);
        for (int i = 0; i < size() - s; i++) res[i] = v[i + s];
        return res;
```

```
    }
    Poly& operator+=(const Poly& r) { return *this = *this + r; }
    Poly& operator-=(const Poly& r) { return *this = *this - r; }
    Poly& operator*=(const Poly& r) { return *this = *this * r; }
    Poly& operator*=(const D& r) { return *this = *this * r; }
    Poly& operator/=(const Poly& r) { return *this = *this / r; }
    Poly& operator/=(const D &r) {return *this = *this/r;}
    Poly& operator%=(const Poly& r) { return *this = *this % r; }
    Poly& operator<<=(const size_t& n) { return *this = *this << n; }
    Poly& operator>>=(const size_t& n) { return *this = *this >> n; }

    Poly pre(int le) const {
        return {{v.begin(), v.begin() + min(size(), le)}};
    }
    Poly rev(int n = -1) const {
        V<D> res = v;
        if (n != -1) res.resize(n);
        reverse(res.begin(), res.end());
        return res;
    }
    Poly diff() const {
        V<D> res(max(0, size() - 1));
        for (int i = 1; i < size(); i++) res[i - 1] = freq(i) * i;
        return res;
    }
    Poly inte() const {
        V<D> res(size() + 1);
        for (int i = 0; i < size(); i++) res[i + 1] = freq(i) / (i + 1);
        return res;
    }
    // f * f.inv() = 1 + g(x)x^m
    Poly inv(int m) const {
        Poly res = Poly({D(1) / freq(0)});
        for (int i = 1; i < m; i *= 2) {
            res = (res * D(2) - res * res * pre(2 * i)).pre(2 * i);
        }
        return res.pre(m);
    }
    Poly exp(int n) const {
        assert(freq(0) == 0);
        Poly f({1}), g({1});
        for (int i = 1; i < n; i *= 2) {
            g = (g * 2 - f * g * g).pre(i);
            Poly q = diff().pre(i - 1);
            Poly w = (q + g * (f.diff() - f * q)).pre(2 * i - 1);
            f = (f + f * (*this - w.inte()).pre(2 * i)).pre(2 * i);
        }
        return f.pre(n);
    }
    Poly log(int n) const {
        assert(freq(0) == 1);
        auto f = pre(n);
        return (f.diff() * f.inv(n - 1)).pre(n - 1).inte();
    }
    Poly sqrt(int n) const {
        assert(freq(0) == 1);
        Poly f = pre(n + 1);
        Poly g({1});
        for (int i = 1; i < n; i *= 2) {
            g = (g + f.pre(2 * i) * g.inv(2 * i)) / 2;
        }
        return g.pre(n + 1);
    }

    Poly pow_mod(ll n, const Poly& mod) {
        Poly x = *this, r = {{1}};
        while (n) {
            if (n & 1) r = r * x % mod;
            x = x * x % mod;
            n >>= 1;
        }
        return r;
    }

    friend ostream& operator<<(ostream& os, const Poly& p) {
        if (p.size() == 0) return os << "0";
        for (auto i = 0; i < p.size(); i++) {
            if (p.v[i]) {
                os << p.v[i] << "x^" << i;
                if (i != p.size() - 1) os << "+";
            }
        }
        return os;
    }
};

template <class Mint> struct MultiEval {
    using NP = MultiEval*;
    NP l, r;
    V<Mint> que;
    int sz;
    Poly<Mint> mul;
    MultiEval(const V<Mint>& _que, int off, int _sz) : sz(_sz) {
        if (sz <= 100) {
            que = {_que.begin() + off, _que.begin() + off + sz};
            mul = {{1}};
            for (auto x : que) mul *= {{-x, 1}};
            return;
        }
        l = new MultiEval(_que, off, sz / 2);
        r = new MultiEval(_que, off + sz / 2, sz - sz / 2);
        mul = l->mul * r->mul;
    }
    MultiEval(const V<Mint>& _que) : MultiEval(_que, 0, int(_que.size())) {}
    void query(const Poly<Mint>& _pol, V<Mint>& res) const {
        if (sz <= 100) {
            for (auto x : que) {
                Mint sm = 0, base = 1;
                for (int i = 0; i < _pol.size(); i++) {
                    sm += base * _pol.freq(i);
                    base *= x;
                }
                res.push_back(sm);
            }
            return;
        }
        auto pol = _pol % mul;
        l->query(pol, res);
        r->query(pol, res);
    }
    V<Mint> query(const Poly<Mint>& pol) const {
        V<Mint> res;
        query(pol, res);
        return res;
    }
};

template <class Mint> Poly<Mint> berlekamp_massey(const V<Mint>& s) {
    int n = int(s.size());
    V<Mint> b = {Mint(-1)}, c = {Mint(-1)};
    Mint y = Mint(1);
    for (int ed = 1; ed <= n; ed++) {
        int l = int(c.size()), m = int(b.size());
        Mint x = 0;
        for (int i = 0; i < l; i++) {
            x += c[i] * s[ed - l + i];
        }
        b.push_back(0);
        m++;
        if (!x) continue;
        Mint freq = x / y;
        if (l < m) {
            // use b
            auto tmp = c;
            c.insert(begin(c), m - l, Mint(0));
            for (int i = 0; i < m; i++) {
                c[m - 1 - i] -= freq * b[m - 1 - i];
            }
            b = tmp;
            y = x;
        } else {
            // use c
            for (int i = 0; i < m; i++) {
                c[l - 1 - i] -= freq * b[m - 1 - i];
            }
        }
    }
    return c;
}

template <class E, class Mint = decltype(E().f)>
Mint sparse_det(const VV<E>& g) {
    int n = int(g.size());
    if (n == 0) return 1;
    auto rand_v = [&]() {
        V<Mint> res(n);
        for (int i = 0; i < n; i++) {
            res[i] = Mint(rand_int(1, Mint(-1).v));
        }
        return res;
    };
    V<Mint> c = rand_v(), l = rand_v(), r = rand_v();
    // l * mat * r
    V<Mint> buf(2 * n);
    for (int fe = 0; fe < 2 * n; fe++) {
        for (int i = 0; i < n; i++) {
            buf[fe] += l[i] * r[i];
        }
        for (int i = 0; i < n; i++) {
            r[i] *= c[i];
        }
        V<Mint> tmp(n);
        for (int i = 0; i < n; i++) {
            for (auto e : g[i]) {
                tmp[i] += r[e.to] * e.f;
            }
        }
        r = tmp;
    }
    auto u = berlekamp_massey(buf);
    if (u.size() != n + 1) return sparse_det(g);
    auto acdet = u.freq(0) * Mint(-1);
    if (n % 2) acdet *= Mint(-1);
    if (!acdet) return 0;
    Mint cdet = 1;
    for (int i = 0; i < n; i++) cdet *= c[i];
    return acdet / cdet;
}
```

Matrix

```
const mint zero(0);
const mint one(1);
bool iszero(mint x){
        return x.v==0;
}
bool isone(mint x){
        return x.v==1;
}

template<class T>
struct Matrix{
        int H,W;
```

```
	VV<T> a;

	Matrix() : H(0),W(0){}
	Matrix(int H,int W) : H(H),W(W),a( VV<T>(H,V<T>(W)) ){}
	Matrix(const VV<T>& v) : H(v.size()), W(v[0].size()), a(v){}

	static Matrix E(int n){
		Matrix a(n,n);
		rep(i,n) a.set(i,i,1);
		return a;
	}

	T at(int i,int j) const {
		return a[i][j];
	}
	void set(int i,int j,T v){
		a[i][j] = v;
	}

	Matrix operator+(const Matrix& r) const {
		assert(H==r.H && W==r.W);
		VV<T> v(H,V<T>(W));
		rep(i,H) rep(j,W) v[i][j] = a[i][j] + r.a[i][j];
		return Matrix(v);
	}
	Matrix operator-(const Matrix& r) const {
		assert(H==r.H && W==r.W);
		VV<T> v(H,V<T>(W));
		rep(i,H) rep(j,W) v[i][j] = a[i][j] - r.a[i][j];
		return Matrix(v);
	}
	Matrix operator*(const Matrix& r) const {
		assert(W==r.H);
		VV<T> v(H,V<T>(r.W));
		rep(i,H) rep(k,W) rep(j,r.W) v[i][j] += a[i][k] * r.a[k][j];
		return Matrix(v);
	}
	Matrix& operator+=(const Matrix& r){return (*this)=(*this)+r;}
	Matrix& operator-=(const Matrix& r){return (*this)=(*this)-r;}
	Matrix& operator*=(const Matrix& r){return (*this)=(*this)*r;}

	/*
		副作用がある, 基本的に自分でこれを呼ぶことはない
		掃き出し法をする
		左からvar列が掃き出す対象で、それより右は同時に値を変更するだけ(e.g. 逆行列は右に単位行列おいてから掃き出す)
		行swap, 列swap は行わない

		rank を返す
	*/
	int sweep(int var){
		int rank = 0;
		vector<bool> used(H);
		rep(j,var){
			int i=0;
			while(i<H && (used[i]||iszero(a[i][j]))) i++;
			if(i==H) continue;
			used[i] = true;
			rank++;
			T t = a[i][j];
			rep(k,W) a[i][k] = a[i][k]/t;
			rep(k,H) if(k!=i){
				T t = a[k][j];
				rep(l,W) a[k][l] = a[k][l]-a[i][l]*t;
			}
		}
		return rank;
	}

	friend ostream& operator<<(ostream &o,const Matrix& A){
		rep(i,A.H){
			rep(j,A.W) o<<A.a[i][j]<<" ";
			o<<endl;
		}
		return o;
	}
};

/*
	逆行列を返す
	なければ0*0行列
*/
template<class T>
Matrix<T> inv(const Matrix<T>& A){
	assert(A.H==A.W);
	int N = A.H;
	Matrix<T> X(N,2*N);
	rep(i,N) rep(j,N) X.set(i,j,A.at(i,j));
	rep(i,N) X.set(i,i+N,one);
	int rank = X.sweep(N);
	if(rank < N) return Matrix<T>();
	Matrix<T> B(N,N);
	rep(i,N){
		rep(j,N){
			B.set(i,j,X.at(i,j+N));
		}
	}
	return B;
}

/*
	Ax = b を解く
	pair(解空間の次元, 解のうちひとつ) を返す
	解が存在しないなら(-1,{})

	解を複数得たい → ランダムな式を追加?
*/
template<class T>
pair< int, vector<T> > solveLinearEquation(const Matrix<T>& A, vector<T> b){
	assert(A.H==(int)b.size());
	int H = A.H, W = A.W;

	Matrix<T> X(H,W+1);
	rep(i,H) rep(j,W) X.set(i,j,A.at(i,j));
	rep(i,H) X.set(i,W,b[i]);
	int rank = X.sweep(W);
	rep(i,H){
		bool allzero = true;
		rep(j,W) if(!iszero(X.at(i,j))) allzero = false;
		if(allzero){
			if(!iszero(X.at(i,W))){		//0x + 0y + 0z = non0
				return pair<int,vector<T> >(-1,vector<T>());
			}
		}
	}
	vector<bool> done(H);
	vector<T> x(W);
	rep(j,W){
		int c0 = 0, c1 = 0;
		int I = -1;
		rep(i,H){
			if(iszero(X.at(i,j))) c0++;
			else if(isone(X.at(i,j))) c1++,I=i;
		}
		if(c0==H-1 && c1==1 && !done[I]){
			x[j] = X.at(I,W);
			done[I] = true;
		}
	}
	return pair<int,vector<T> >(W-rank,x);
}

/*
	determinant
*/
template<class T>
T det(Matrix<T> A){
	assert(A.H==A.W);
	int N = A.H;
	int rank = A.sweep(N);
	if(rank < N) return zero;
	T d = one;
	vector<int> to(N);
	rep(i,N){
		rep(j,N){
			if(!iszero(A.at(i,j))){
				to[i] = j;
				d *= A.at(i,j);
			}
		}
	}
	vector<bool> done(N);
	rep(i,N) if(!done[i]){
		int x = i;
		bool odd = 1;
		while(!done[x]){
			done[x] = 1;
			x = to[x];
			odd = !odd;
		}
		if(odd) d = -d;
	}
	return d;
}

/*
	rank
*/
template<class T>
int getrank(Matrix<T> A){
	return A.sweep(A.W);
}
```

Kernel

```
//ker(p)を返す
//pが空のときに備えてpの列の個数wを与える
//sigmaのmatrixで動作確認済み
VV<double> get_kernel(const VV<double>&p,int w){
	int h=p.size();
	VV<double> rev(w,V<double>(h));
	rep(i,w){
		rep(j,h)
			rev[i][j]=p[j][i];
		rep(j,w)
			rev[i].pb(i==j);
	}
	{
		auto waf=Mat(rev);
		waf.sweep(h);
		rev=waf.a;
	}
	VV<double> ker;
	rep(i,w){
		bool zero=true;
		rep(j,h)
			zero&=iszero(rev[i][j]);
```

```
                if(zero){
                        ker.eb(rev[i].begin()+h,rev[i].end());
                }
        }
        return ker;
}
```

Subset

```
// size k
int c=(1<<k)-1;
while(c<1<<n){
        // hoge
        int x=c&-c,y=c+x;
        c=(((c&~y)/x)>>1)|y;
}

//subset of b(descending)
int a=b;
do{
        //
        a=(a-1)&b;
}while(a!=b);
```

MonotoneMinima

```
using D = ll;
void minima(int lx,int rx,int ly,int ry){       //[lx,rx) について bestpos[x] を求める 調べる範囲は[ly,ry)でよい
        if(lx >= rx) return;
        int x = (lx+rx)/2;
        D best = ?;
        int besty = -1;
        for(int y = ly; y<ry; y++){
                D val = f(x,y)
//              D val = dp[k-1][y] + cost(y,m);
                if(best ? val){
                        best = val;
                        besty = y;
                }
        }

//      opt[x] = y

        minima(k,lx,m,ly,besty+1);
        minima(k,m+1,rx,besty,ry);
}
```

PeriodicMaxmin

```
/*
        a[i] = sin(2pi*i/N) みたいに、
        - 周期的であって
        - 適当な場所から見ると up down を一度する

        a という配列があって、それをクエリで取得できる時に、log回呼び出しでargmax,argminを特定する

        前提: 連続する二箇所で=になってはいけない

        例: 凸多角形の各頂点と直線との距離
            凸多角形と点が与えられて、点からどの頂点達が見えるか(偏角)
*/
D query(int i,int N){
        i%=N;
        //
}

template<class D>
int getArgmax(int N){
        if(N==1) return 0;

        D h = query(0);
        bool up = h < query(1);

        int lb = 0, ub = N;
        while(ub-lb>1){
                int m = (ub+lb)/2;
                if(query(m) < h){
                        if(up) ub = m;
                        else lb = m;
                }else{
                        D a = query(m);
                        D b = query(m+1);
                        if(a<b) lb = m;
                        else ub = m;
                }
        }
        return query(lb)<query(ub) ? ub : lb;
}

template<class D>
int getArgmin(int N){
        if(N==1) return 0;

        D h = query(0);
        bool up = h < query(1);

        int lb = 0, ub = N;
        while(ub-lb>1){
                int m = (ub+lb)/2;
                if(query(m) > h){
                        if(up) lb = m;
                        else ub = m;
                }else{
                        D a = query(m);
                        D b = query(m+1);
                        if(a>b) lb = m;
                        else ub = m;
                }
        }
        return query(lb)<query(ub) ? lb : ub;
}
```

SumFloor

```
//x_i=floor((a*i+b)/c), i=0,1,..n-1
//a,c>0, b>=0
//verified: CF530E
ll gauss_sum(ll n, ll a, ll b, ll c){
        if(n==0)return 0;
        ll res=0;
        {
                ll p=a/c;
                res+=n*(n-1)/2*p;
                a%=c;
        }
        {
                ll p=b/c;
                res+=n*p;
                b%=c;
        }
        if(a==0)return res;
        ll top=(a*(n-1)+b)/c;
        res+=top*n;
        ll h=(b+1+c-1)/c;
        if(h<=top)
                res-=gauss_sum(top-h+1,c,c*h-(b+1),a)+top-h+1;
        return res;
}
```

Simplex

```
/*
        LP
        max cx
        s.t. Ax <= b, x >= 0

        time complexity: exponential. fast $O(HW^2)$ in experiment. dependent on the modeling.
*/
template<class T>
struct LPsolver{
        const T eps = 1e-7;

        int type;
        T cx;
        V<T> sol;   //W

        // 0: found solution, 1: infeasible, 2: unbounded
        LPsolver(VV<T> A, V<T> b, V<T> c){
                int H = A.size(), W = A[0].size();
                assert((int)b.size() == H);
                assert((int)c.size() == W);

                Left.resize(H);
                Down.resize(W);
                sol.resize(W);
                cx = 0;

                auto pivot = [&](int x, int y) {
                        swap(Left[x], Down[y]);
                        T k = A[x][y];
                        A[x][y] = 1;
                        vector<int> nz;
                        rep(i,W){
                                A[x][i] /= k;
                                if (!eq(A[x][i], 0)) nz.push_back(i);
                        }
                        b[x] /= k;

                        rep(i,H){
                                if (i == x || eq(A[i][y], 0)) continue;
                                k = A[i][y];
                                A[i][y] = 0;
                                b[i] -= k * b[x];
                                for (int j : nz) A[i][j] -= k * A[x][j];
                        }
                        if (eq(c[y], 0)) return;
                        k = c[y];
                        c[y] = 0;
                        cx += k * b[x];
                        for (int i : nz) c[i] -= k * A[x][i];
                };

                rep(i,W) Down[i] = i;
                rep(i,H) Left[i] = W + i;
                while (1) { // Eliminating negative b[i]
                        int x = -1, y = -1;
                        rep(i,H) if(ls(b[i], 0) && (x == -1 || b[i] < b[x])) x = i;
                        if (x == -1) break;
```

```
                        rep(i,W) if(ls(A[x][i], 0) && (y == -1 || A[x][i] <
A[x][y])) y = i;
                        if (y == -1){
                                type = 1;   //infeasible
                                cx = -1e100;
                                return;
                        }
                        pivot(x, y);
                }
                while (1) {
                        int x = -1, y = -1;
                        rep(i,W) if(ls(0, c[i]) && (y == -1 || c[i] > c[y])) y =
i;
                        if (y == -1) break;
                        rep(i,H) if(ls(0, A[i][y]) && (x == -1 || b[i] / A[i][y]
< b[x] / A[x][y])) x = i;
                        if (x == -1){
                                type = 2;   //unbounded
                                cx = 1e100;
                                return;
                        }
                        pivot(x, y);
                }
                rep(i,H) if (Left[i] < W) sol[Left[i]] = b[i];
                type = 0;
        }

        V<int> Left,Down;   //H,W
        bool eq(T a, T b) { return fabs(a - b) < eps; }
        bool ls(T a, T b) { return a < b && !eq(a, b); }
};
```

Graph

SCC

```
struct SCC {
    V<int> id;
    VV<int> groups;
};

template <class E> struct SCCExec : SCC {
    int n;
    const VV<E>& g;
    int tm = 0;
    V<bool> flag;
    V<int> low, ord, st;
    void dfs(int v) {
        low[v] = ord[v] = tm++;
        st.push_back(v);
        flag[v] = true;
        for (auto e : g[v]) {
            if (ord[e.to] == -1) {
                dfs(e.to);
                low[v] = min(low[v], low[e.to]);
            } else if (flag[e.to]) {
                low[v] = min(low[v], ord[e.to]);
            }
        }
        if (low[v] == ord[v]) {
            V<int> gr;
            while (true) {
                int u = st.back();
                st.pop_back();
                gr.push_back(u);
                if (u == v) break;
            }
            for (int x : gr) flag[x] = false;
            groups.push_back(gr);
        }
    }
    SCCExec(const VV<E>& _g)
        : n(int(_g.size())), g(_g), flag(n), low(n), ord(n, -1) {
        id = V<int>(n);
        for (int i = 0; i < n; i++) {
            if (ord[i] == -1) dfs(i);
        }
        reverse(groups.begin(), groups.end());
        for (int i = 0; i < int(groups.size()); i++) {
            for (int x : groups[i]) {
                id[x] = i;
            }
        }
    }
};

template <class E> SCC get_scc(const VV<E>& g) { return SCCExec<E>(g); }

template <size_t N> struct BitsetSCCExec : SCC {
    using B = bitset<N>;
    int n;
    const V<B>& g;
    const V<B>& rg;
    V<int> vs;
    B unvis;
    void dfs(int v) {
        unvis.reset(v);
        while (true) {
            int d = (unvis & g[v])._Find_first();
            if (d >= n) break;
            dfs(d);
        }
        vs.push_back(v);
    }
    void rdfs(int v, int k) {
        unvis.reset(v);
        id[v] = k;
        groups[k].push_back(v);
        while (true) {
            int d = (unvis & rg[v])._Find_first();
            if (d >= n) break;
            rdfs(d, k);
        }
    }

    BitsetSCCExec(const V<B>& _g, const V<B>& _rg)
        : n(int(_g.size())), g(_g), rg(_rg) {
        unvis.set();
        for (int i = 0; i < n; i++) {
            if (unvis[i]) dfs(i);
        }
        reverse(vs.begin(), vs.end());
        unvis.set();
        id = V<int>(n);
        int k = 0;
        for (int i : vs) {
            if (unvis[i]) {
                groups.push_back({});
                rdfs(i, k++);
            }
        }
    }
};

template <size_t N>
SCC get_bitset_scc(const V<bitset<N>>& g, const V<bitset<N>>& rg) {
    return BitsetSCCExec<N>(g, rg);
}
```

TwoSat

```
struct TwoSat {
    V<bool> res;

    struct Edge { int to; };
    VV<Edge> g;

    //(a == a_exp) || (b == b_exp)
    void add_cond(int a, bool a_exp, int b, bool b_exp) {
        g[2 * a + (a_exp ? 0 : 1)].push_back(Edge{2 * b + (b_exp ? 1 : 0)});
        g[2 * b + (b_exp ? 0 : 1)].push_back(Edge{2 * a + (a_exp ? 1 : 0)});
    }
    bool exec() {
        int n = int(res.size());
        auto s = get_scc(g);
        for (int i = 0; i < n; i++) {
            if (s.id[2 * i] == s.id[2 * i + 1]) return false;
            res[i] = s.id[2 * i] < s.id[2 * i + 1];
        }
        return true;
    }
    TwoSat() {}
    TwoSat(int n) {
        g = VV<Edge>(2 * n);
        res = V<bool>(n);
    }
};
```

関節点

```
struct Ariticulation {
    VV<int> tr;
};

template <class E> struct AriticulationExec : Ariticulation {
    const VV<E>& g;
    int n;
    int ordc = 0;
    V<int> low, ord;
    V<int> used;
    AriticulationExec(const VV<E>& _g)
            : g(_g), n(int(g.size())), low(n), ord(n), used(n) {
        tr = VV<int>(n);
        for (int i = 0; i < n; i++) {
            if (used[i]) continue;
            dfs1(i, -1);
            dfs2(i, -1);
        }
    }
    void dfs1(int p, int b) {
        used[p] = 1;
        low[p] = ord[p] = ordc++;
        bool rt = true;
        for (auto e : g[p]) {
            int d = e.to;
            if (rt && d == b) {
                rt = false;
                continue;
            }
            if (!used[d]) {
                dfs1(d, p);
                low[p] = min(low[p], low[d]);
            } else {
                low[p] = min(low[p], ord[d]);
            }
        }
    }
    void dfs2(int p, int bid = -1) {
```

```
        used[p] = 2;
        if (bid != -1) {
            tr[p].push_back(bid);
            tr[bid].push_back(p);
        }
        for (auto e: g[p]) {
            int d = e.to;
            if (used[d] == 2) continue;
            if (low[d] < ord[p]) {
                dfs2(d, bid);
                continue;
            }
            int nid = int(tr.size());
            tr.push_back({});
            tr[p].push_back(nid);
            tr[nid].push_back(p);
            dfs2(d, nid);
        }
    }
};

template <class E> Ariticulation get_ariticulation(const VV<E>& g) {
    return AriticulationExec<E>(g);
}
```

橋

```
struct Bridge {
    V<int> id;
    VV<int> groups;
    VV<int> tr;
};

template <class E> struct BridgeExec : Bridge {
    const VV<E>& g;
    int n;
    int ordc = 0;
    V<int> low, ord, vlist;
    V<int> used;
    BridgeExec(const VV<E>& _g)
        : g(_g), n(int(g.size())), low(n), ord(n), used(n) {
        id = V<int>(n);
        for (int i = 0; i < n; i++) {
            if (used[i]) continue;
            dfs1(i, -1);
            dfs2(i, -1);
        }
    }
    void dfs1(int p, int b) {
        used[p] = 1;
        low[p] = ord[p] = ordc++;
        vlist.push_back(p);
        bool rt = true;
        for (auto e : g[p]) {
            int d = e.to;
            if (rt && d == b) {
                rt = false;
                continue;
            }
            if (!used[d]) {
                dfs1(d, p);
                low[p] = min(low[p], low[d]);
            } else {
                low[p] = min(low[p], ord[d]);
            }
        }
    }
    void dfs2(int p, int b) {
        used[p] = 2;
        bool is_root = low[p] == ord[p];
        if (is_root) {
            int idc = int(groups.size());
            id[p] = idc;
            groups.push_back({p});
            tr.push_back({});
            if (b != -1) {
                tr[idc].push_back(id[b]);
                tr[id[b]].push_back(idc);
            }
        } else {
            id[p] = id[b];
            groups[id[p]].push_back(p);
        }
        for (auto e : g[p]) {
            int d = e.to;
            if (d == b || used[d] == 2) continue;
            dfs2(d, p);
        }
    }
};

template <class E> Bridge get_bridge(const VV<E>& g) {
    return BridgeExec<E>(g);
}
```

MaxMatching

```
// Gabow Edmond's blossom algorithm
// Reference: https://qiita.com/Kutimoti_T/items/5b579773e0a24d650bdf
template <class E> struct MaxMatching {
    int n;
    const VV<E>& g;
    V<int> mt;
    using P = pair<int, int>;
    V<int> is_ev, gr_buf;
    V<P> nx;
    int st;
    int group(int x) {
        if (gr_buf[x] == -1 || is_ev[gr_buf[x]] != st) return gr_buf[x];
        return gr_buf[x] = group(gr_buf[x]);
    }

    void match(int p, int b) {
        int d = mt[p];
        mt[p] = b;
        if (d == -1 || mt[d] != p) return;
        if (nx[p].second == -1) {
            mt[d] = nx[p].first;
            match(nx[p].first, d);
        } else {
            match(nx[p].first, nx[p].second);
            match(nx[p].second, nx[p].first);
        }
    }

    bool arg() {
        is_ev[st] = st;
        gr_buf[st] = -1;
        nx[st] = P(-1, -1);
        queue<int> q;
        q.push(st);
        while (q.size()) {
            int a = q.front();
            q.pop();
            for (auto e : g[a]) {
                int b = e.to;
                if (b == st) continue;
                if (mt[b] == -1) {
                    mt[b] = a;
                    match(a, b);
                    return true;
                }
                if (is_ev[b] == st) {
                    int x = group(a), y = group(b);
                    if (x == y) continue;
                    int z = -1;
                    while (x != -1 || y != -1) {
                        if (y != -1) swap(x, y);
                        if (nx[x] == P(a, b)) {
                            z = x;
                            break;
                        }
                        nx[x] = P(a, b);
                        x = group(nx[mt[x]].first);
                    }
                    for (int v : {group(a), group(b)}) {
                        while (v != z) {
                            q.push(v);
                            is_ev[v] = st;
                            gr_buf[v] = z;
                            v = group(nx[mt[v]].first);
                        }
                    }
                } else if (is_ev[mt[b]] != st) {
                    is_ev[mt[b]] = st;
                    nx[b] = P(-1, -1);
                    nx[mt[b]] = P(a, -1);
                    gr_buf[mt[b]] = b;
                    q.push(mt[b]);
                }
            }
        }
        return false;
    }
    MaxMatching(const VV<E>& _g)
        : n(int(_g.size())), g(_g), mt(n, -1), is_ev(n, -1), gr_buf(n), nx(n) {
        for (st = 0; st < n; st++)
            if (mt[st] == -1) arg();
    }
};
```

最小有向全域木

```
//Union-Find without Rank
struct UnionFind{
        vi par;
        UnionFind(int n){
                par.resize(n,-1);
        }
        int Find(int a){
                return par[a]==-1?a:(par[a]=Find(par[a]));
        }
        void Unite(int a,int b){
                assert(par[a]==-1);
                assert(par[b]==-1);
                par[b]=a;
        }
};

const int Nmax=100010;
struct Edge{
        int to,cost;
};
vector<Edge> g[Nmax];

struct EQ{
        using pqpi=priority_queue<pi,vector<pi>,greater<pi>>;
        pqpi* q;
```

```
	int off;
	void Init(const vector<Edge>&es){
		q=new pqpi();
		for(auto e:es)
			q->push(pi(e.cost,e.to));
		off=0;
	}
	void Merge(EQ& x){
		if(q->size()<x.q->size())
			swap(*this,x);
		while(!x.q->empty()){
			pi e=x.q->top();
			x.q->pop();
			q->push(pi(e.first+x.off-off,e.second));
		}
	}
	Edge Pop(){
		pi e=q->top();
		q->pop();
		return Edge{e.second,e.first+off};
	}
	void Del(){
		delete q;
	}
};
//leaf->root
int directed_mst(int n,int root){
	vector<EQ> eqs(n);
	REP(i,n)
		eqs[i].Init(g[i]);
	vi state(n,0);
	state[root]=2;
	vi curCost(n);

	UnionFind uf(n);
	int ans=0;
	REP(i,n)if(!state[i]){
		vi vs{i};
		state[i]=1;
		while(1){
			Edge e=eqs[vs.back()].Pop();
			curCost[vs.back()]=e.cost;
			ans+=e.cost;
			int to=uf.Find(e.to);
			if(state[to]==0){
				vs.PB(to);
				state[to]=1;
			}else if(state[to]==1){
				int r=vs.size(),l=r-1;
				while(vs[l]!=to)
					l--;
				FOR(j,l,r){
					eqs[vs[j]].off-=curCost[vs[j]];
					if(l<j){
						eqs[vs[l]].Merge(eqs[vs[j]]);
						uf.Unite(vs[l],vs[j]);
					}
				}
				vs.resize(l+1);
			}else{
				break;
			}
		}
		for(auto v:vs)
			state[v]=2;
	}

	REP(i,n)
		eqs[i].Del();

	return ans;
}
```

最小直径全域木

```
template<class Num>
Num MDST(const vector<vector<Num>>&g){
	int n=g.size();
	if(n==1)return 0;
	vector<vector<Num>> dist=g;
	REP(k,n)REP(i,n)REP(j,n)
		chmin(dist[i][j],dist[i][k]+dist[k][j]);
	Num ans=inf;
	REP(i,n){
		vi ord(n);
		REP(j,n)ord[j]=j;
		sort(ALL(ord),[&](int a,int b){
			return dist[i][a]<dist[i][b];
		});
		FOR(j,i+1,n)if(g[i][j]<inf){
			vi idx;
			for(auto k:ord){
				while(!idx.empty()&&dist[j][idx.back()]
<=dist[j][k])
					idx.pop_back();
				idx.PB(k);
			}
			chmin(ans,dist[i][idx.back()]*2);
			chmin(ans,dist[j][idx.front()]*2);
			REP(w,int(idx.size())-1){
				chmin(ans,dist[i][idx[w]]+dist[j][idx[w+1]]+g[i]
[j]);
			}
		}
	}
	return ans;
}
```

Dominator Tree

```
struct DominatorTree{
	int n;
	vector<vi> g,rG,bct;
	vi idom,semi,us,id,rId,par,mn,anc;

	DominatorTree(int nn):n(nn){
		g.resize(n);
		rG.resize(n);
		bct.resize(n);
		idom.resize(n,-1);
		semi.resize(n);
		us.resize(n);
		id.resize(n,-1);
		rId.resize(n);
		par.resize(n,-1);
		mn.resize(n);
		anc.resize(n,-1);
		REP(i,n){
			semi[i]=i;
			mn[i]=i;
		}
	}

	void AddEdge(int a,int b){
		g[a].PB(b);
		rG[b].PB(a);
	}

	int Find(int v){
		if(anc[v]==-1)
			return v;
		int a=Find(anc[v]);
		if(id[semi[mn[anc[v]]]]<id[semi[mn[v]]])
			mn[v]=mn[anc[v]];
		return anc[v]=a;
	}

	void Link(int c,int p){
		anc[c]=p;
	}

	void dfs(int v,int p,int& i){
		if(id[v]!=-1)
			return;
		id[v]=i;
		rId[i++]=v;
		par[v]=p;
		for(int c:g[v])
			dfs(c,v,i);
	}

	vi Calc(int root){
		int sz=0;
		dfs(root,-1,sz);
		for(int i=sz-1;i>0;i--){
			int w=rId[i];
			for(int v:rG[w])if(id[v]!=-1){
				Find(v);
				if(id[semi[mn[v]]]<id[semi[w]])
					semi[w]=semi[mn[v]];
			}
			bct[semi[w]].PB(w);
			for(int v:bct[par[w]]){
				Find(v);
				us[v]=mn[v];
			}
			bct[par[w]].clear();
			Link(w,par[w]);
		}
		FOR(i,1,sz){
			int w=rId[i];
			if(semi[w]==semi[us[w]])
				idom[w]=semi[w];
			else
				idom[w]=idom[us[w]];
		}
		return idom;
	}
};
```

Max Clique

```
template <int N, class E> struct MaxClique {
    using B = bitset<N>;
    int n;
    V<B> g, col_buf;
    V<int> clique, now;
    struct P {
        int id, col, deg;
    };
    VV<P> rems;
    void dfs(int dps = 0) {
        if (clique.size() < now.size()) clique = now;
        auto& rem = rems[dps];
        sort(rem.begin(), rem.end(), [&](P a, P b) { return a.deg > b.deg; });
        int max_c = 1;
        for (auto& p : rem) {
            p.col = 0;
            while ((g[p.id] & col_buf[p.col]).any()) p.col++;
```

```
            max_c = max(max_c, p.id + 1);
            col_buf[p.col].set(p.id);
        }
        for (int i = 0; i < max_c; i++) col_buf[i].reset();
        sort(rem.begin(), rem.end(), [&](P a, P b) { return a.col < b.col; });

        while (!rem.empty()) {
            auto p = rem.back();
            if (now.size() + p.col + 1 <= clique.size()) break;

            auto& nrem = rems[dps + 1];
            nrem.clear();
            B bs = B();
            for (auto q : rem) {
                if (g[p.id][q.id]) {
                    nrem.push_back({q.id, -1, 0});
                    bs.set(q.id);
                }
            }
            for (auto& q : nrem) {
                q.deg = (bs & g[q.id]).count();
            }
            now.push_back(p.id);
            dfs(dps + 1);
            now.pop_back();

            rem.pop_back();
        }
    }

    MaxClique(VV<E> _g) : n(int(_g.size())), g(n), col_buf(n), rems(n + 1) {
        for (int i = 0; i < n; i++) {
            rems[0].push_back({i, -1, int(_g[i].size())});
            for (auto e : _g[i]) g[i][e.to] = 1;
        }
        dfs();
    }
};
```

Flow

MaxFlow

```
/*
struct E {
    int to, rev, cap;
};
VV<E> g;
auto add_edge = [&](int from, int to, int cap) {
    g[from].push_back(E{to, int(g[to].size()), cap});
    g[to].push_back(E{from, int(g[from].size())-1, 0});
};
*/

template<class C>
struct MaxFlow {
    C flow;
    V<char> dual;  // false: S-side true: T-side
};

template<class C, class E>
struct MFExec {
    static constexpr C INF = numeric_limits<C>::max();
    C eps;
    VV<E>& g;
    int s, t;
    V<int> level, iter;

    C dfs(int v, C f) {
        if (v == t) return f;
        C res = 0;
        for (int& i = iter[v]; i < int(g[v].size()); i++) {
            E& e = g[v][i];
            if (e.cap <= eps || level[v] >= level[e.to]) continue;
            C d = dfs(e.to, min(f, e.cap));
            e.cap -= d;
            g[e.to][e.rev].cap += d;
            res += d;
            f -= d;
            if (f == 0) break;
        }
        return res;
    }

    MaxFlow<C> info;
    MFExec(VV<E>& _g, int _s, int _t, C _eps)
        : eps(_eps), g(_g), s(_s), t(_t) {
        int N = int(g.size());

        C& flow = (info.flow = 0);
        while (true) {
            queue<int> que;
            level = V<int>(N, -1);
            level[s] = 0;
            que.push(s);
            while (!que.empty()) {
                int v = que.front(); que.pop();
                for (E e: g[v]) {
                    if (e.cap <= eps || level[e.to] >= 0) continue;
                    level[e.to] = level[v] + 1;
                    que.push(e.to);
                }
            }
            if (level[t] == -1) break;
            iter = V<int>(N, 0);
            while (true) {
                C f = dfs(s, INF);
                if (!f) break;
                flow += f;
            }
        }
        for (int i = 0; i < N; i++) info.dual.push_back(level[i] == -1);
    }
};

template<class C, class E>
MaxFlow<C> get_mf(VV<E>& g, int s, int t, C eps) {
    return MFExec<C, E>(g, s, t, eps).info;
}
```

Hungarian

```
/**
割当問題を解き，以下の条件を満たすle, ri, permを得る
- le[i] <= 0, ri[j] >= 0
- cost[i][j] + le[i] + ri[j] >= 0
- cost[i][perm[i]] + le[i] + ri[perm[i]] = 0
*/
template<class D>
struct Hungarian {
    V<D> le, ri;
    V<int> perm;

    Hungarian(const VV<D>& c) {
        int n = int(c.size()), m = int(c[0].size());
        assert(n <= m);
        le = V<D>(n, D(0)); ri = V<D>(m, D(0));
        perm = V<int>(n);
        V<int> to_r(n, -1), to_l(m, -1);

        for (int s = 0; s < n; s++) {
            V<char> l_u(n), r_u(m);
            l_u[s] = true;
            V<int> tr(m, -1), min_l(m, s);
            V<D> min_cost(m);
            for (int j = 0; j < m; j++) min_cost[j] = c[s][j] + le[s] + ri[j];
            while (true) {
                int r = -1;
                D d = numeric_limits<D>::max();
                for (int j = 0; j < m; j++) {
                    if (!r_u[j] && min_cost[j] < d) {
                        r = j;
                        d = min_cost[j];
                    }
                }
                for (int i = 0; i < n; i++) if (l_u[i]) le[i] -= d;
                for (int j = 0; j < m; j++) {
                    if (r_u[j]) ri[j] += d;
                    else min_cost[j] -= d;
                }
                tr[r] = min_l[r];
                int l = to_l[r];
                if (l == -1) {
                    while (r != -1) {
                        int nl = tr[r], nr = to_r[nl];
                        to_l[r] = nl; to_r[nl] = r;
                        r = nr;
                    }
                    break;
                }
                l_u[l] = r_u[r] = true;
                for (int j = 0; j < m; j++) {
                    D cost = c[l][j] + le[l] + ri[j];
                    if (cost < min_cost[j]) {
                        min_l[j] = l;
                        min_cost[j] = cost;
                    }
                }
            }
        }
        perm = to_r;
    }
};
```

MinCostFlow

```
/*
struct E {
    int to, rev, cap, dist;
};
VV<E> g;
auto add_edge = [&](int from, int to, int cap, int dist) {
    g[from].push_back(E{to, int(g[to].size()), cap, dist});
    g[to].push_back(E{from, int(g[from].size())-1, 0, -dist});
};

auto res = min_cost_flow<int, int>(g, s, t, false);
res.max_flow(TEN(9));

// cap_flow : 最大流量
// flow : 最小費用
*/

template<class C, class D, class E>
struct MinCostFlow {
    static constexpr D INF = numeric_limits<D>::max();
    int n;
    VV<E> g;
```

```
    int s, t;
    C nc, cap_flow = 0;
    D nd, flow = 0;

    V<D> dual;
    V<int> pv, pe;

    MinCostFlow(VV<E> _g, int _s, int _t, bool neg)
        : n(int(_g.size())), g(_g), s(_s), t(_t) {
        assert(s != t);
        dual = V<D>(n);
        pv = pe = V<int>(n);
        if (neg) {
            V<D> dist(g.size(), D(INF));
            dist[s] = 0;
            for (int ph = 0; ph < n; ph++) {
                for (int i = 0; i < n; i++) {
                    for (auto e: g[i]) {
                        if (!e.cap || dist[i] == INF) continue;
                        dist[e.to] = min(dist[e.to], dist[i] + e.dist);
                    }
                }
            }
            for (int v = 0; v < int(g.size()); v++) {
                dual[v] += dist[v];
            }
        }
        dual_ref();
    }

    C single_flow(C c) {
        if (nd == INF) return nc;
        c = min(c, nc);
        for (int v = t; v != s; v = pv[v]) {
            E& e = g[pv[v]][pe[v]];
            e.cap -= c;
            g[v][e.rev].cap += c;
        }
        cap_flow += c;
        flow += nd * c;
        nc -= c;
        if (!nc) dual_ref();
        return c;
    }

    void max_flow(C c) {
        while (c) {
            C f = single_flow(c);
            if (!f) break;
            c -= f;
        }
    }

    void dual_ref() {
        V<D> dist(g.size(), D(INF));
        pv = pe = V<int>(n, -1);
        struct Q {
            D key;
            int to;
            bool operator<(Q r) const { return key > r.key; }
        };
        priority_queue<Q> que;
        dist[s] = 0;
        que.push(Q{D(0), s});
        V<char> vis(n);
        while (!que.empty()) {
            int v = que.top().to; que.pop();
            if (v == t) break;
            if (vis[v]) continue;
            vis[v] = true;
            for (int i = 0; i < int(g[v].size()); i++) {
                E e = g[v][i];
                if (vis[e.to] || !e.cap) continue;
                D cost = dist[v] + e.dist + dual[v] - dual[e.to];
                if (dist[e.to] > cost) {
                    dist[e.to] = cost;
                    pv[e.to] = v; pe[e.to] = i;
                    que.push(Q{dist[e.to], e.to});
                }
            }
        }
        if (dist[t] == INF) {
            nd = INF; nc = 0;
            return;
        }
        for (int v = 0; v < int(g.size()); v++) {
            if (!vis[v]) continue;
            dual[v] += dist[v] - dist[t];
        }
        nd = dual[t] - dual[s];
        assert(0 <= nd);
        nc = numeric_limits<C>::max();
        for (int v = t; v != s; v = pv[v]) {
            nc = min(nc, g[pv[v]][pe[v]].cap);
        }
    }
};

template<class C, class D, class E>
MinCostFlow<C, D, E> get_mcf(const VV<E>& g, int s, int t, bool neg = false) {
    return MinCostFlow<C, D, E>(g, s, t, neg);
}
```

Tree

LCA

```
struct LCA {
    int lg;
    VV<int> anc;
    V<int> dps;
    /// lとrの頂点のLCAを求める
    int query(int l, int r) {
        if (dps[l] < dps[r]) swap(l, r);
        int dd = dps[l] - dps[r];
        for (int i = lg - 1; i >= 0; i--) {
            if (dd < (1 << i)) continue;
            dd -= 1 << i;
            l = anc[i][l];
        }
        if (l == r) return l;
        for (int i = lg - 1; i >= 0; i--) {
            if (anc[i][l] == anc[i][r]) continue;
            tie(l, r) = tie(anc[i][l], anc[i][r]);
        }
        return anc[0][l];
    }
};

template <class E> struct LCAExec : LCA {
    const VV<E>& g;

    /// 事前処理を行う rはroot頂点のid
    LCAExec(const VV<E>& _g, int r) : g(_g) {
        int N = int(g.size());
        lg = 1;
        while ((1 << lg) < N) lg++;
        anc = VV<int>(lg, V<int>(N, -1));
        dps = V<int>(N);
        dfs(r, -1, 0);
        for (int i = 1; i < lg; i++) {
            for (int j = 0; j < N; j++) {
                anc[i][j] =
                    (anc[i - 1][j] == -1) ? -1 : anc[i - 1][anc[i - 1][j]];
            }
        }
    }

    void dfs(int p, int b, int now) {
        anc[0][p] = b;
        dps[p] = now;
        for (E e : g[p]) {
            if (e.to == b) continue;
            dfs(e.to, p, now + 1);
        }
    }
};

template <class E> LCA get_lca(const VV<E>& g, int r) {
    return LCAExec<E>(g, r);
}
```

全方位木DP

```
template <class N, class E> struct AllTree {
    int n;
    const VV<E>& g;
    V<N> sm;
    VV<N> dp;  // tree
    void dfs1(int p, int b) {
        sm[p] = N();
        for (auto e : g[p]) {
            int d = e.to;
            if (d == b) continue;
            dfs1(d, p);
            sm[p] = sm[p] + sm[d].to_subs(p, e);
        }
        sm[p] = sm[p].join(p);
    }
    void dfs2(int p, int b, N top) {
        int deg = int(g[p].size());
        dp[p] = V<N>(deg + 1);
        dp[p][0] = N();
        for (int i = 0; i < deg; i++) {
            int d = g[p][i].to;
            dp[p][i + 1] =
                dp[p][i] + (d == b ? top : sm[d]).to_subs(p, g[p][i]);
        }
        N rnode = N();
        dp[p].back() = dp[p].back().join(p);
        for (int i = deg - 1; i >= 0; i--) {
            dp[p][i] = (dp[p][i] + rnode).join(p);
            int d = g[p][i].to;
            if (d != b) dfs2(d, p, dp[p][i]);
            rnode = rnode + (d == b ? top : sm[d]).to_subs(p, g[p][i]);
        }
    }
    AllTree(const VV<E>& _g) : n(int(_g.size())), g(_g), sm(n), dp(n) {
        dfs1(0, -1);
        dfs2(0, -1, N());
    }
};

template <class N, class E> VV<N> get_all_tree(const VV<E>& g) {
    return AllTree<N, E>(g).dp;
}

struct Node {
    // Educational DP Contest V - Subtree
    Mint sm = Mint(1);
    template <class E> Node to_subs(int, const E&) const {
```

```
        // tree -> subtrees
        return {sm + 1};
    }
    Node operator+(const Node& r) const {
        // subtrees + subtrees
        return {sm * r.sm};
    }
    Node join(int) const {
        // subtrees -> tree
        return *this;
    }
};

struct Node {
    // Diameter of Tree
    int rad = 0, dia = 0;         // radius(tree), diameter
    array<int, 2> rd = {{0, 0}};  // radiuses(subtrees)
    template <class E> Node to_subs(int, const E& e) const {
        // tree -> subtrees
        return {-1, dia, {rad + e.dist, 0}};
    }
    Node operator+(const Node& r) const {
        // subtrees + subtrees
        array<int, 4> v = {rd[0], rd[1], r.rd[0], r.rd[1]};
        sort(v.begin(), v.end(), greater<>());
        return {-1, max(dia, r.dia), {v[0], v[1]}};
    }
    Node join(int) const {
        // subtrees -> tree
        return {rd[0], max(dia, rd[0] + rd[1]), {}};
    }
};
```

HL-Decomp

```
const int Nmax=100010;
vi tr[Nmax];
int
par[Nmax],dep[Nmax],sub[Nmax],grp[Nmax],pos[Nmax],gsz[Nmax],gpar[Nmax],gdep[Nmax
],gord;
int dfs1(int v,int p,int d){
        par[v]=p;
        sub[v]=1;
        dep[v]=d;
        for(auto to:tr[v])if(to!=p)
                sub[v]+=dfs1(to,v,d+1);
        return sub[v];
}
void dfs2(int v,int p,int g,int d){
        grp[v]=g;
        if(gsz[g]==0){
                gpar[g]=p;
                gdep[g]=d;
        }
        pos[v]=gsz[g]++;
        pi si(0,-1);
        for(auto to:tr[v])if(to!=p)
                chmax(si,pi(sub[to],to));
        for(auto to:tr[v])if(to!=p){
                if(to==si.second)
                        dfs2(to,v,g,d);
                else
                        dfs2(to,v,gord++,d+1);
        }
}
int LCA(int a,int b){
        if(gdep[grp[a]]>gdep[grp[b]])
                swap(a,b);
        while(gdep[grp[a]]<gdep[grp[b]])
                b=gpar[grp[b]];
        while(grp[a]!=grp[b]){
                a=gpar[grp[a]];
                b=gpar[grp[b]];
        }
        if(dep[a]>dep[b])swap(a,b);
        return a;
}

dfs1(0,-1,0);
dfs2(0,-1,gord++,0);
```

重心分解

```
struct Edge{
        int to,dist,idx;
};
vector<Edge> tree[Nmax];
bool vRem[Nmax];

int TreeSize(int v,int p){
        int ret=1;
        for(auto e:tree[v])if(e.to!=p&&!vRem[e.to])
                ret+=TreeSize(e.to,v);
        return ret;
}

int FindCentroid(int v,int p,int s){
        int ret=1,mx=0;
        for(auto e:tree[v])if(e.to!=p&&!vRem[e.to]){
                int f=FindCentroid(e.to,v,s);
                if(f<=0)
                        return f;
                else{
                        ret+=f;
                        mx=max(mx,f);
                }
        }
        mx=max(mx,s-ret);
        if(mx*2<=s)
                return -v;
        else
                return ret;
}

void Solve(int root){
        int ts=TreeSize(root,-1);
        if(ts==1)
                return;
        root=-FindCentroid(root,-1,ts);
}
```

木圧縮

```
/*
        usage:
                CompressedSubtree CS(tree)
                のあと好きなだけ CS.ComputeSubtree(vs) を呼べばいい
                返り値は縮約後の木の辺の集合 vector<pair<int,int>> !!もとの頂点番号!!
                もしreindexしたいならindex[v] を見ればいい
*/
int bsr(int x){ //4~7 -> 2
        if(x==0) return -1;
        return 31 ^ __builtin_clz(x);
}

template<class E>
struct CompressedSubtree{
        int N,n;
        V<int> depth;
        VV<int> par;
        V<int> in;
        int I;

        V<int> index;
        V<int> vs;

        CompressedSubtree(const VV<E>& G, int r =
0):N((int)G.size()),n(bsr(N)),depth(N),par(N,V<int>(n+1)),in(N),I(0),index(N){
                dfs(r,-1,G);
                rep1(i,n){
                        rep(v,N){
                                if(par[v][i-1] == -1) par[v][i] = -1;
                                else par[v][i] = par[par[v][i-1]][i-1];
                        }
                }
        }

        V<pair<int,int>> ComputeTree(const V<int>& _vs){
                auto comp = [&](int x,int y){
                        return in[x] < in[y];
                };
                vs = _vs;
                sort(all(vs),comp);
                vs.erase(unique(vs.begin(),vs.end()),vs.end());

                int K = vs.size();
                rep(i,K-1){
                        vs.pb(lca(vs[i],vs[i+1]));
                }
                sort(all(vs),comp);
                vs.erase(unique(vs.begin(),vs.end()),vs.end());
                K = vs.size();
                rep(i,K) index[vs[i]] = i;
                V<pair<int,int>> es;
                rep1(i,K-1){
                        int p = lca(vs[i-1],vs[i]);
                        es.pb(pair<int,int>(vs[i],p));
                }
                return es;
        }

        void dfs(int v,int p,const VV<E>& G){
                in[v] = I++;
                par[v][0] = p;
                for(auto& e : G[v]){
                        int u = e.to;
                        if(u == p) continue;
                        depth[u] = depth[v] + 1;
                        dfs(u,v,G);
                }
        }

        int lca(int u,int v){
                if(depth[u]<depth[v]) swap(u,v);
                int d = depth[u]-depth[v];
                rep(i,n+1){
                        if((d>>i)&1) u=par[u][i];
                }
                if(u==v) return u;
                for(int i=n;i>=0;i--){
                        if(par[u][i]!=par[v][i]){
                                u=par[u][i];
                                v=par[v][i];
                        }
                }
                return par[v][0];
        }
        int distance(int u,int v){
```

```
            return depth[u]+depth[v]-2*depth[lca(u,v)];
        }
};
struct edge{int to;};
```

Data Structure

UnionFind

```
struct UnionFind {
    V<int> p, r;
    int gn;
    UnionFind(int n = 0) : p(n, -1), r(n, 1), gn(n) {}
    void merge(int a, int b) {
        int x = group(a), y = group(b);
        if (x == y) return;  // same
        gn--;
        if (r[x] < r[y]) {
            p[x] = y;
        } else {
            p[y] = x;
            if (r[x] == r[y]) r[x]++;
        }
    }
    int group(int a) {
        if (p[a] == -1) return a;
        return p[a] = group(p[a]);
    }
    bool same(int a, int b) { return group(a) == group(b); }
};
```

QuickFind

```
struct QuickFind {
    V<int> id;
    VV<int> groups;
    int gc;  // group count
    QuickFind(int n = 0) {
        id = V<int>(n);
        groups = VV<int>(n);
        for (int i = 0; i < n; i++) {
            id[i] = i;
            groups[i] = {i};
        }
        gc = n;
    }
    void merge(int a, int b) {
        if (same(a, b)) return;
        gc--;
        int x = id[a], y = id[b];
        if (groups[x].size() < groups[y].size()) swap(x, y);
        for (int j : groups[y]) {
            id[j] = x;
            groups[x].push_back(j);
        }
        groups[y].clear();
    }
    bool same(int a, int b) { return id[a] == id[b]; }
};
```

Fenwick

```
template <class T> struct Fenwick {
    int n;
    V<T> seg;
    Fenwick(int _n = 0) : n(_n), seg(n + 1) {}
    /// i番目の要素にxを追加する
    void add(int i, T x) {
        i++;
        while (i <= n) {
            seg[i] += x;
            i += i & -i;
        }
    }
    /// [0, i)のsum
    T sum(int i) {
        T s = 0;
        while (i > 0) {
            s += seg[i];
            i -= i & -i;
        }
        return s;
    }
    /// [a, b)のsum
    T sum(int a, int b) { return sum(b) - sum(a); }
    /// sum[0, idx) >= xなる最小のidx(sum[0, n) < x なら n+1)
    int sum_lower_bound(T x) {
        if (x <= 0) return 0;
        int res = 0, len = 1;
        while (2 * len <= n) len *= 2;
        for (; len >= 1; len /= 2) {
            if (res + len <= n && seg[res + len] < x) {
                res += len;
                x -= seg[res];
            }
        }
        return res + 1;
    }
};
```

Fenwick2D

```
template <class D, class I> struct Fenwick2D {
    using P = pair<I, I>;
    V<P> points;
    VV<I> ys;
    V<Fenwick<D>> fws;
    int lg, sz;
    Fenwick2D(V<P> _points) : points(_points) {
        sort(points.begin(), points.end());
        points.erase(unique(points.begin(), points.end()), points.end());
        int n = int(points.size());
        lg = 1;
        while ((1 << lg) < n) lg++;
        sz = 1 << lg;
        ys = VV<I>(2 * sz);
        for (int i = 0; i < n; i++) ys[sz + i].push_back(points[i].second);
        for (int i = sz - 1; i >= 1; i--) {
            ys[i] = V<I>(ys[2 * i].size() + ys[2 * i + 1].size());
            merge(ys[2 * i].begin(), ys[2 * i].end(), ys[2 * i + 1].begin(),
                  ys[2 * i + 1].end(), ys[i].begin());
        }
        fws = V<Fenwick<D>>(2 * sz);
        for (int i = 1; i < 2 * sz; i++) {
            fws[i] = Fenwick<D>(int(ys[i].size()));
        }
    }

    void add(P p, D x) {
        int k =
            int(lower_bound(points.begin(), points.end(), p) - points.begin());
        k += sz;
        while (k) {
            int yid = lower_bound(ys[k].begin(), ys[k].end(), p.second) -
                      ys[k].begin();
            fws[k].add(yid, x);
            k >>= 1;
        }
    }

    D sum(int a, int b, I lw, I up, int l, int r, int k) {
        if (b <= l || r <= a) return D(0);
        if (a <= l && r <= b) {
            int lid =
                lower_bound(ys[k].begin(), ys[k].end(), lw) - ys[k].begin();
            int uid =
                lower_bound(ys[k].begin(), ys[k].end(), up) - ys[k].begin();
            return fws[k].sum(lid, uid);
        }
        int mid = (l + r) / 2;
        return sum(a, b, lw, up, l, mid, 2 * k) +
               sum(a, b, lw, up, mid, r, 2 * k + 1);
    }

    D sum(P lw, P up) {
        int a = lower_bound(points.begin(), points.end(), lw.first,
                            [&](P p, I x) { return p.first < x; }) -
                points.begin();
        int b = lower_bound(points.begin(), points.end(), up.first,
                            [&](P p, I x) { return p.first < x; }) -
                points.begin();
        return sum(a, b, lw.second, up.second, 0, sz, 1);
    }
};
```

SparseTable

```
template <class D, class OP> struct SparseTable {
    D e;
    OP op;
    VV<D> data;
    SparseTable(V<D> v = V<D>(), D _e = D(), OP _op = OP()) : e(_e), op(_op) {
        int n = int(v.size());
        if (n == 0) return;
        int lg = bsr(uint(n)) + 1;
        data = VV<D>(lg);
        data[0] = v;
        int l = 1;
        for (int s = 1; s < lg; s++) {
            data[s] = V<D>(n);
            for (int i = 0; i < n - l; i++) {
                data[s][i] = op(data[s - 1][i], data[s - 1][i + l]);
            }
            l <<= 1;
        }
    }
    D query(int l, int r) const {
        assert(l <= r);
        if (l == r) return e;
        int u = bsr(uint(r - l));
        return op(data[u][l], data[u][r - (1 << u)]);
    }
};
template <class D, class OP>
SparseTable<D, OP> get_sparse_table(V<D> v, D e, OP op) {
    return SparseTable<D, OP>(v, e, op);
}

template <class D, class OP> struct LowMemorySparseTable {
    static constexpr int B = 16;
    V<D> data;
    D e;
    OP op;
    SparseTable<D, OP> st;
    V<D> comp_arr(V<D> v) {
        int n = int(v.size());
        V<D> comp(n / B);
```

```
        for (int i = 0; i < n / B; i++) {
            D res = data[i * B];
            for (int j = 1; j < B; j++) {
                res = op(res, data[i * B + j]);
            }
            comp[i] = res;
        }
        return comp;
    }
    LowMemorySparseTable(V<D> v = V<D>(), D _e = D(), OP _op = OP())
        : data(v), e(_e), op(_op), st(comp_arr(v), _e, _op) {}
    D query(int l, int r) const {
        assert(l <= r);
        if (l == r) return e;
        int lb = (l + B - 1) / B, rb = r / B;
        D res = e;
        if (lb >= rb) {
            for (int i = l; i < r; i++) {
                res = op(res, data[i]);
            }
            return res;
        }

        while (l % B) {
            res = op(res, data[l]);
            l++;
        }
        while (r % B) {
            r--;
            res = op(res, data[r]);
        }
        res = op(res, st.query(lb, rb));
        return res;
    }
};
template <class D, class OP>
LowMemorySparseTable<D, OP> get_low_memory_sparse_table(V<D> v, D e, OP op) {
    return LowMemorySparseTable<D, OP>(v, e, op);
}
```

DisjointTable

```
template <class D, class OP> struct DisjointTable {
    D e;
    OP op;
    VV<D> data;
    DisjointTable(V<D> v = V<D>(), D _e = D(), OP _op = OP()) : e(_e), op(_op) {
        int lg = 1;
        while ((1 << lg) < int(v.size())) lg++;
        int n = 1 << lg;
        v.resize(n, e);
        data = VV<D>(lg, V<D>(n));
        data[0] = v;
        for (int h = 1; h < lg; h++) {
            int u = (1 << h);
            for (int i = 0; i < n / (2 * u); i++) {
                int base = i * (2 * u) + u;
                D res;
                res = e;
                for (int j = base - 1; j >= base - u; j--) {
                    res = op(v[j], res);
                    data[h][j] = res;
                }
                res = e;
                for (int j = base; j < base + u; j++) {
                    res = op(res, v[j]);
                    data[h][j] = res;
                }
            }
        }
    }
    D query(int l, int r) {
        r--;
        if (l > r) return e;
        if (l == r) return data[0][l];
        int u = bsr(uint(l ^ r));
        return op(data[u][l], data[u][r]);
    }
};

template <class D, class OP>
DisjointTable<D, OP> get_disjoint_table(V<D> v, D e, OP op) {
    return DisjointTable<D, OP>(v, e, op);
}
```

fastset

```
struct FastSet {
    static constexpr uint B = 64;
    int n, lg;
    VV<ull> seg;
    FastSet(int _n) : n(_n) {
        do {
            seg.push_back(V<ull>((_n + B - 1) / B));
            _n = (_n + B - 1) / B;
        } while (_n > 1);
        lg = seg.size();
    }
    bool operator[](int i) const {
        return (seg[0][i / B] >> (i % B) & 1) != 0;
    }
    void set(int i) {
        for (int h = 0; h < lg; h++) {
            seg[h][i / B] |= 1ULL << (i % B);
            i /= B;
        }
    }
    void reset(int i) {
        for (int h = 0; h < lg; h++) {
            seg[h][i / B] &= ~(1ULL << (i % B));
            if (seg[h][i / B]) break;
            i /= B;
        }
    }
    // x以上最小の要素
    int next(int i) {
        for (int h = 0; h < lg; h++) {
            if (i / B == seg[h].size()) break;
            ull d = seg[h][i / B] >> (i % B);
            if (!d) {
                i = i / B + 1;
                continue;
            }
            // find
            i += bsf(d);
            for (int g = h - 1; g >= 0; g--) {
                i *= B;
                i += bsf(seg[g][i / B]);
            }
            return i;
        }
        return n;
    }
    // x未満最大の要素
    int prev(int i) {
        i--;
        for (int h = 0; h < lg; h++) {
            if (i == -1) break;
            ull d = seg[h][i / B] << (63 - i % 64);
            if (!d) {
                i = i / B - 1;
                continue;
            }
            // find
            i += bsr(d) - (B - 1);
            for (int g = h - 1; g >= 0; g--) {
                i *= B;
                i += bsr(seg[g][i / B]);
            }
            return i;
        }
        return -1;
    }
};
```

ConvexHull

```
template<class T>
struct ConvexHull {
    using L = array<T, 2>;

    bool que_incr;
    ConvexHull(bool _que_incr) : que_incr(_que_incr) {}

    deque<L> lines;
    // can remove mid?
    static bool is_need(L mid, L left, L right) {
        assert(left[0] <= mid[0] && mid[0] <= right[0]);
        return (right[0]-mid[0])*(left[1]-mid[1]) < (mid[0]-left[0])*(mid[1]-
right[1]);
    }
    //work with 2^(60 + 64)
    /*static bool is_need(L mid, L left, L right) {
        assert(left[0] <= mid[0] && mid[0] <= right[0]);
        ll a = (right[0]-mid[0]), b = (left[1]-mid[1]), c = (mid[0]-left[0]), d
= (mid[1]-right[1]);
        long double x = (long double)(a) * b - (long double)(c) * d;
        if (abs(x) > (1LL << 60)) return x < 0;
        int fl = b < 0, fr = d < 0;
        if (fl != fr) return fl == 1;
        ull z = ull(a) * ull(abs(b)) - ull(c) * ull(abs(d));
        if (fl == 0) return (1ULL << 63) < z;
        return z < (1ULL << 63);
    }*/

    void insert_front(L l) {
        if (lines.empty()) {
            lines.push_front(l);
            return;
        }
        assert(l[0] <= lines[0][0]);
        if (l[0] == lines[0][0]) {
            if (l[1] <= lines[0][1]) return;
            lines.pop_front();
        }
        while (lines.size() >= 2 && !is_need(lines.front(), l, lines[1])) {
            lines.pop_front();
        }
        lines.push_front(l);
    }
    void insert_back(L l) {
        if (lines.empty()) {
            lines.push_back(l);
            return;
        }
        assert(lines.back()[0] <= l[0]);
        if (lines.back()[0] == l[0]) {
            if (l[1] <= lines.back()[1]) return;
            lines.pop_back();
```

```
        }
        while (lines.size() >= 2 && !is_need(lines.back(),
lines[lines.size()-2], l)) {
            lines.pop_back();
        }
        lines.push_back(l);
    }
    /**
    Insert line
    line's degree must be minimum or maximum
     */
    void insert_line(L line) {
        if (lines.empty()) {
            lines.push_back(line);
            return;
        }
        if (line[0] <= lines[0][0]) insert_front(line);
        else if (lines.back()[0] <= line[0]) insert_back(line);
        else assert(false); //line's degree must be minimum or maximum
    }
    /// get maximum y
    T b_x;
    T first = true;
    T max_y(T x) {
        assert(lines.size());
        auto value = [&](L l) { return l[0] * x + l[1]; };
        if (que_incr) {
            assert(first || b_x <= x);
            first = false; b_x = x;
            while (lines.size() >= 2 &&
                   value(lines[0]) <= value(lines[1])) {
                lines.pop_front();
            }
            return value(lines.front());
        } else {
            assert(first || x <= b_x);
            first = false; b_x = x;
            while (lines.size() >= 2 &&
                   value(lines[lines.size()-2]) >= value(lines.back())) {
                lines.pop_back();
            }
            return value(lines.back());
        }
    }
};
```

DynamicConvexHull

```
//Author: Simon Lindholm
//https://github.com/kth-competitive-programming/kactl/blob/master/content/data-
structures/LineContainer.h
struct Line {
        mutable ll k, m, p;
        bool operator<(const Line& o) const { return k < o.k; }
        bool operator<(ll x) const { return p < x; }
};

struct LineContainer : multiset<Line, less<>> {
        // (for doubles, use inf = 1/.0, div(a,b) = a/b)
        const ll inf = LLONG_MAX;
        ll div(ll a, ll b) { // floored division
                return a / b - ((a ^ b) < 0 && a % b); }
        bool isect(iterator x, iterator y) {
                if (y == end()) { x->p = inf; return false; }
                if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
                else x->p = div(y->m - x->m, x->k - y->k);
                return x->p >= y->p;
        }
        void add(ll k, ll m) {
                auto z = insert({k, m, 0}), y = z++, x = y;
                while (isect(y, z)) z = erase(z);
                if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
                while ((y = x) != begin() && (--x)->p >= y->p)
                        isect(x, erase(y));
        }
        ll query(ll x) {//gives max value
                assert(!empty());
                auto l = *lower_bound(x);
                return l.k * x + l.m;
        }
};
```

RBST

```
inline int xorshift(){
        static int w=1234567890;
        w=w^(w<<17);
        w=w^(w>>13);
        w=w^(w<<5);
        return w;
}
struct Node{
        Node *l,*r;
        int v,s;
} buf[200010];
//Not Verified
int bufUsed;
Node* NewNode(int v){
        Node* ptr=buf+(bufUsed++);
        ptr->l=NULL;
        ptr->r=NULL;
        ptr->s=1;
        ptr->v=v;
        return ptr;
}
Node* Merge(Node*a,Node*b){
        if(!a)return b;
        if(!b)return a;
        int s=a->s+b->s,x=xorshift()%s;
        if(x<a->s){
                a->r=Merge(a->r,b);
                a->s=s;
                return a;
        }else{
                b->l=Merge(a,b->l);
                b->s=s;
                return b;
        }
}
using pn=pair<Node*,Node*>;
pn Split(Node*x,int t){
        if(!x)return pn(0,0);
        if(t<=x->v){
                pn c=Split(x->l,t);
                if(c.first)x->s-=c.first->s;
                x->l=c.second;
                return pn(c.first,x);
        }else{
                pn c=Split(x->r,t);
                if(c.second)x->s-=c.second->s;
                x->r=c.first;
                return pn(x,c.second);
        }
}
int MaxValue(Node* x){
        while(x->r)x=x->r;
        return x->v;
}
```

Link Cut

```
struct Node{
        typedef Node* NP;
        NP l,r,p;
        bool rev;
        int v,mx,lz;
        Node():l(NULL),r(NULL),p(NULL),rev(false),v(-inf),mx(-inf),lz(-inf){}
        void Propagate(){
                if(rev){
                        swap(l,r);
                        if(l) l->rev^=true;
                        if(r) r->rev^=true;
                        rev=false;
                }
                if(l)chmax(l->lz,lz);
                if(r)chmax(r->lz,lz);
                chmax(v,lz);
                chmax(mx,lz);
                lz=-inf;
        }
        int GetMax(){
                return max(mx,lz);
        }
        int GetVert(){
                return max(v,lz);
        }
        void Update(){
                assert(lz==-inf);
                mx=v;
                if(l){
                        chmax(mx,l->GetMax());
                }
                if(r){
                        chmax(mx,r->GetMax());
                }
        }
        int Pos(){
                if(p&&p->l==this) return -1;
                if(p&&p->r==this) return 1;
                return 0;
        }
        void Prepare(){
                if(Pos())
                        p->Prepare();
                Propagate();
        }
        void Rotate(){
                NP q=p,c;
                if(Pos()==1){
                        c=l;
                        l=p;
                        p->r=c;
                }else{
                        c=r;
                        r=p;
                        p->l=c;
                }
                if(c) c->p=p;
                p=p->p;
                q->p=this;
                if(p&&p->l==q) p->l=this;
                if(p&&p->r==q) p->r=this;
                q->Update();
        }
        void Splay(){
                Prepare();
                while(Pos()){
                        int a=Pos(),b=p->Pos();
                        if(b&&a==b) p->Rotate();
                        if(b&&a!=b) Rotate();
```

```
                        Rotate();
                }
                Update();
        }
        void Expose(){
                for(NP x=this;x;x=x->p) x->Splay();
                for(NP x=this;x->p;x=x->p){
                        x->p->r=x;
                        x->p->Update();
                }
                Splay();
        }
        void Evert(){
                Expose();
                if(l){
                        l->rev^=true;
                        l=NULL;
                        Update();
                }
        }
        void Link(NP x){
                Evert();
                p=x;
        }
        void Set(int q){
                Expose();
                r=NULL;
                chmax(lz,q);
        }
        void Cut() {
                Expose();
                assert(l);
                l->p = NULL;
                l = NULL;
                Update();
        }
        int Get(){
                Expose();
                r=NULL;
                Update();
                return GetMax();
        }
};

Node* LCA(Node* a, Node* b) {
        a->Expose();
        b->Expose();
        if (!a->p) {
                return NULL;
        }

        Node* d = a;
        while (a->p != b) {
                if (a->Pos()==0) {
                        d = a->p;
                }
                a = a->p;
        }
        if (a == b->l) {
                return d;
        }
        else {
                return b;
        }
}
```

Persistent AVL

```
//Range Add/Sum
const int bufSize=18000000;
struct Node{
        Node const *l,*r;
        ll lz,sm;
        int sz,dep;
} buf[bufSize];

using Np=Node const*;
using LR=Node const;

int bufUsed;
Np newNode(Np l,Np r,ll lz,ll sm,int sz,int dep){
        buf[bufUsed]=Node{l,r,lz,sm,sz,dep};
        return buf+(bufUsed++);
}

ll GetSum(Np x){
        return x->sm+x->lz*x->sz;
}

Np newPar(Np l,Np r){
        assert(l&&r);
        return newNode(
                l,r,
                0,
                GetSum(l)+GetSum(r),
                l->sz+r->sz,
                max(l->dep,r->dep)+1
        );
}

Np addLz(Np x,ll alz){
        assert(x);
        return alz?newNode(x->l,x->r,x->lz+alz,x->sm,x->sz,x->dep):x;
}

LR Merge1(Np a,Np b,Np c){
        assert(a&&b&&c);
        if(abs(max(a->dep,b->dep)+1-c->dep)<=1)
                return LR{newPar(a,b),c};
        if(abs(max(b->dep,c->dep)+1-a->dep)<=1)
                return LR{a,newPar(b,c)};
        assert(b->l&&b->r);
        return LR{newPar(a,addLz(b->l,b->lz)),newPar(addLz(b->r,b->lz),c)};
}

LR Merge2(Np a,Np b,ll alz,ll blz){
        assert(a&&b);
        if(a->dep<b->dep){
                blz+=b->lz;
                assert(b->l&&b->r);
                LR x=Merge2(a,b->l,alz,blz);
                return Merge1(x.l,x.r,addLz(b->r,blz));
        }else if(a->dep>b->dep){
                alz+=a->lz;
                assert(a->l&&a->r);
                LR x=Merge2(a->r,b,alz,blz);
                return Merge1(addLz(a->l,alz),x.l,x.r);
        }else
                return LR{addLz(a,alz),addLz(b,blz)};
}

Np Merge(Np a,Np b){
        if(!a)return b;
        if(!b)return a;
        LR x=Merge2(a,b,0,0);
        return newPar(x.l,x.r);
}

LR Split(Np x,int s,ll lz){
        assert(x);
        if(x->sz==s)
                return LR{addLz(x,lz),NULL};
        if(s==0)
                return LR{NULL,addLz(x,lz)};
        lz+=x->lz;
        if(s<=x->l->sz){
                LR g=Split(x->l,s,lz);
                return LR{g.l,Merge(g.r,addLz(x->r,lz))};
        }else{
                LR g=Split(x->r,s-x->l->sz,lz);
                return LR{Merge(addLz(x->l,lz),g.l),g.r};
        }
}

Np Build(vi arr){
        vector<Np> tmp;
        for(auto v:arr)
                tmp.PB(newNode(NULL,NULL,v,0,1,0));
        int n=tmp.size();
        for(int s=1;s<n;s*=2)
                for(int i=0;i<n;i+=2*s)
                        if(i+s<n)
                                tmp[i]=Merge(tmp[i],tmp[i+s]);
        return tmp[0];
}
void dfs(Np x,ll lz,vi& dst){
        lz+=x->lz;
        if(!x->l)
                dst.PB(lz);
        else{
                dfs(x->l,lz,dst);
                dfs(x->r,lz,dst);
        }
}
Np Rebuild(Np root){
        vi tmp;
        dfs(root,0,tmp);
        bufUsed=0;
        return Build(tmp);
}
```

Leftist Heap

```
template<class T>
struct LeftistHeap{
        T t;
        LeftistHeap<T> *l,*r;
        int s;
        LeftistHeap():l(NULL),r(NULL),s(1){}
};

template<class T>
LeftistHeap<T>* NewNode(T t,LeftistHeap<T>*buf){
        static int bufUsed=0;
        auto res=buf+bufUsed++;
        res->t=t;
        return res;
}

template<class T>
int Depth(LeftistHeap<T>*a){
        if(!a)return 0;
        return a->s;
}

template<class T>
LeftistHeap<T>* Merge(LeftistHeap<T>*a,LeftistHeap<T>*b){
        if(!a)return b;
        if(!b)return a;
        if(a->t<b->t)swap(a,b);
        a->r=Merge(a->r,b);
        if(Depth(a->l)<Depth(a->r))
```

```
                swap(a->l,a->r);
        a->s=Depth(a->r)+1;
        return a;
}

template<class T>
LeftistHeap<T>* Pop(LeftistHeap<T>*a){
        return Merge(a->l,a->r);
}
```

Top Tree

```
struct TTNode {
    using NP = TTNode*;

    bool rev = false;
    array<array<NP, 2>, 2> ch = {};  // tree, light-tree
    NP p = nullptr, lt = nullptr;

    struct D {
        ll cnt = 0, pd = 0, upd = 0, dwd = 0;
        // l is parent of r
        static D merge_h(const D& l, const D& r) {
            return {l.cnt + r.cnt, l.pd + r.pd, l.upd + r.upd + l.pd * r.cnt,
                    l.dwd + r.dwd + r.pd * l.cnt};
        }
        // l and r is parallel
        static D merge_w(const D& l, const D& r) {
            assert(!l.pd && !r.pd && !l.dwd && !r.dwd);
            return {l.cnt + r.cnt, 0, l.upd + r.upd, 0};
        }
        // add parent for r(subtrees)
        static D join(const D& l, const D& r) {
            assert(l.upd == l.dwd);
            return {l.cnt + r.cnt, l.pd, l.upd + r.upd + l.pd * r.cnt,
                    l.dwd + r.upd + l.pd * r.cnt};
        }
        D rev() { return D{cnt, pd, dwd, upd}; }
        D to_subs() { return D{cnt, 0, upd, 0}; }
    };
    D single = D(), sub = D(), subs = D();

    NP search(ll nw, ll f) {
        // search heavy
        assert(sub.cnt >= nw);
        push();
        if (ch[0][1] && ch[0][1]->sub.cnt >= nw) return ch[0][1]->search(nw, f);
        if (ch[0][1]) nw -= ch[0][1]->sub.cnt;
        if (single.cnt + (lt ? lt->subs.cnt : 0) >= nw) {
            if (!lt || lt->subs.cnt < f) return this;
            auto q = lt->search_light(f);
            if (!q) return this;
            return q;
        }
        nw -= single.cnt;
        if (lt) nw -= lt->subs.cnt;
        assert(ch[0][0]);
        return ch[0][0]->search(nw, f);
    }
    NP search_light(ll f) {
        assert(subs.cnt >= f);
        if (sub.cnt >= f) {
            return search(f, f);
        }
        push();
        if (ch[1][0] && ch[1][0]->subs.cnt >= f)
            return ch[1][0]->search_light(f);
        if (ch[1][1] && ch[1][1]->subs.cnt >= f)
            return ch[1][1]->search_light(f);
        expose();
        return nullptr;
    }
    void init_node(ll cnt, ll d) {
        single.cnt = cnt;
        single.pd = d;
        single.upd = single.dwd = cnt * d;
        update();
    }
    void update_subs() {
        subs = sub.to_subs();
        if (ch[1][0]) subs = D::merge_w(ch[1][0]->subs, subs);
        if (ch[1][1]) subs = D::merge_w(subs, ch[1][1]->subs);
    }
    void update() {
        assert(!rev);
        sub = single;
        if (lt) sub = D::join(single, lt->subs);
        if (ch[0][0]) sub = D::merge_h(ch[0][0]->sub, sub);
        if (ch[0][1]) sub = D::merge_h(sub, ch[0][1]->sub);
        update_subs();
    }
    void revdata() {
        rev ^= true;
        swap(ch[0][0], ch[0][1]);  // Important
        sub = sub.rev();
        update_subs();
    }
    void push() {
        if (rev) {
            if (ch[0][0]) ch[0][0]->revdata();
            if (ch[0][1]) ch[0][1]->revdata();
            rev = false;
        }
    }
    // optimize? : template<int ty>
    int pos(int ty) {
        if (p) {
            if (p->ch[ty][0] == this) return 0;
            if (p->ch[ty][1] == this) return 1;
        }
        return -1;
    }
    static void con(NP p, NP& cp, NP ch) {
        cp = ch;
        if (ch) ch->p = p;
    }
    void rot(int ty) {
        int ps = pos(ty);
        NP _p = p, q = p->p;
        if (ty == 0) {
            ch[1] = _p->ch[1];
            _p->ch[1].fill(nullptr);
            for (auto& x : ch[1])
                if (x) x->p = this;
        }
        con(_p, _p->ch[ty][ps], ch[ty][1 - ps]);
        con(this, ch[ty][1 - ps], _p);
        _p->update();
        update();
        p = q;
        if (!q) return;
        if (q->lt == _p) q->lt = this;
        for (auto& v : q->ch)
            for (auto& x : v)
                if (x == _p) x = this;
        q->update();
    }
    void splay(int ty) {
        int ps;
        while ((ps = pos(ty)) != -1) {
            int pps = p->pos(ty);
            if (pps == -1) {
                rot(ty);
            } else if (ps == pps) {
                p->rot(ty);
                rot(ty);
            } else {
                rot(ty);
                rot(ty);
            }
        }
    }
    void expose() {
        supush();
        splay(0);
        splay(1);
        if (NP z = ch[0][1]) {
            z->push();
            con(z, z->ch[1][1], lt);
            lt = z;
            ch[0][1] = nullptr;
            z->update();
            update();
        }
        NP u = p;
        while (u) {
            u->splay(0);
            u->splay(1);
            NP ur = u->lt;
            if (auto r = u->ch[0][1]) {
                // swap ur, r
                r->push();
                r->ch[1] = ur->ch[1];
                ur->ch[1].fill(nullptr);
                for (auto& x : r->ch[1])
                    if (x) x->p = r;
                r->update();
                u->lt = r;
            } else if (!ur->ch[1][0]) {
                // use ur->ch[1][1]
                con(u, u->lt, ur->ch[1][1]);
            } else {
                // use prev(ur) in light-tree
                NP q = ur->ch[1][0];
                con(u, u->lt, q);
                q->push();
                while (q->ch[1][1]) {
                    q = q->ch[1][1];
                    q->push();
                }
                q->splay(1);
                con(q, q->ch[1][1], ur->ch[1][1]);
                q->update();
            }
            ur->ch[1].fill(nullptr);
            ur->update();
            u->ch[0][1] = ur;
            u->update();
            u = u->p;
        }
        splay(0);
    }
    void supush() {
        if (p) p->supush();
        push();
    }
    void link(NP r) {
        evert();
        r->expose();
        assert(!r->ch[0][1]);
        con(r, r->ch[0][1], this);
        r->update();
    }
```

```
    void cut() {
        expose();
        assert(ch[0][0]);
        ch[0][0]->p = nullptr;
        ch[0][0] = nullptr;
        update();
    }
    void evert() {
        expose();
        revdata();
        expose();
    }
};
```

pb_ds

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

//not a multiset
//find_by_order(k) -> itr of k-th(0-based) element
//order_of_key(k) -> index of lower_bound(k)
using ordered_set=tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>;
```

文字列

AhoCorasick

```
struct ACTrie {
    using NP = ACTrie*;
    V<int> acc;
    map<int, NP> next;
    NP fail = nullptr, dnx = nullptr;

  private:
    void add(const string& s, int id, int p = 0) {
        if (p == int(s.size())) {
            acc.push_back(id);
            return;
        }
        if (next[s[p]] == nullptr) {
            next[s[p]] = new ACTrie();
        }
        next[s[p]]->add(s, id, p + 1);
    }
    template <class OP> NP count(OP op, int p) {
        if (fail == nullptr) return this;
        for (int id : acc) {
            op(id, p);
        }
        if (dnx) {
            dnx->count(op, p);
        } else {
            dnx = fail->count(op, p);
        }
        return acc.size() ? this : dnx;
    }

  public:
    // パターンにマッチするたびにop(string ID, 発見位置の終端)を呼び出す
    // 終端が同じで複数マッチする文字列が存在する場合，長い順に呼び出される
    // s = "abaaba", pattern = {"ab", "ba"} なら
    // op(0, 2), op(1, 3), op(0, 5), op(1, 6)
    template <class OP> void match(const string& s, OP op, int p = 0) {
        if (p == int(s.size())) return;
        if (next[s[p]]) {
            next[s[p]]->count(op, p + 1);
            next[s[p]]->match(s, op, p + 1);
        } else {
            if (!fail)
                match(s, op, p + 1);  // root
            else
                fail->match(s, op, p);  // other
        }
    }
    static NP make(V<string> v) {
        NP tr = new ACTrie();
        for (int i = 0; i < int(v.size()); i++) {
            tr->add(v[i], i);
        }
        queue<NP> q;
        q.push(tr);
        tr->fail = nullptr;
        while (!q.empty()) {
            NP ntr = q.front();
            q.pop();
            for (auto p : ntr->next) {
                int i = p.first;
                NP fail = ntr->fail;
                while (fail && !fail->next.count(i)) {
                    fail = fail->fail;
                }
                ntr->next[i]->fail = (fail == nullptr) ? tr : fail->next[i];
                q.push(ntr->next[i]);
            }
        }
        return tr;
    }
};
```

Rolling Hash

```
using Mint0 = ModInt<TEN(9) + 7>;
using Mint1 = ModInt<TEN(9) + 9>;

V<Mint0> powB0{1}, powiB0{1};
V<Mint1> powB1{1}, powiB1{1};
Mint0 B0 = rand_int(1, TEN(9)), iB0 = B0.inv();
Mint1 B1 = rand_int(1, TEN(9)), iB1 = B1.inv();
void first() {
    for (int i = 0; i < TEN(6); i++) {
        powB0.push_back(powB0.back() * B0);
        powiB0.push_back(powiB0.back() * iB0);
        powB1.push_back(powB1.back() * B1);
        powiB1.push_back(powiB1.back() * iB1);
    }
}

struct H {
    int le = 0;
    Mint0 h0;
    Mint1 h1;
    H() : le(0), h0(0), h1(0) {}
    H(int _le, Mint0 _h0, Mint1 _h1) : le(_le), h0(_h0), h1(_h1) {}
    H(int c) : le(1), h0(c), h1(c) {}
    // H(l) + H(r) = H(lr)
    H operator+(const H& r) const {
        return H{le + r.le, h0 + r.h0 * powB0[le], h1 + r.h1 * powB1[le]};
    }
    H& operator+=(const H& r) { return *this = *this + r; }

    bool operator==(const H& r) const {
        return le == r.le && h0 == r.h0 && h1 == r.h1;
    }
    bool operator!=(const H& r) const {
        return !(*this == r);
    }
    // H(lr).strip_left(H(l)) = H(r)
    H strip_left(const H& l) const {
        return H{le - l.le, (h0 - l.h0) * powiB0[l.le], (h1 - l.h1) *
powiB1[l.le]};
    }
    // H(lr).strip_right(H(r)) = H(l)
    H strip_right(const H& r) const {
        return H{le - r.le, h0 - r.h0 * powB0[le - r.le], h1 - r.h1 * powB1[le -
r.le]};
    }
};
```

Suffix Array

```
template <class Str> struct SA {
    Str s;
    V<int> sa, rsa, lcp;
    SA() {}
    SA(Str _s, V<int> _sa) : s(_s), sa(_sa) {
        int n = int(s.size());
        // make rsa
        rsa = V<int>(n + 1);
        for (int i = 0; i <= n; i++) {
            rsa[sa[i]] = i;
        }
        // make lcp
        lcp = V<int>(n);
        int h = 0;
        for (int i = 0; i < n; i++) {
            int j = sa[rsa[i] - 1];
            if (h > 0) h--;
            for (; j + h < n && i + h < n; h++) {
                if (s[j + h] != s[i + h]) break;
            }
            lcp[rsa[i] - 1] = h;
        }
    }
};

template <class Str> V<int> sa_is(Str s, int B = 200) {
    int n = int(s.size());
    V<int> sa(n + 1);
    if (n == 0) return sa;

    for (int i = 0; i < n; i++) s[i]++;
    s.push_back(0);
    B++;

    V<bool> ls(n + 1);
    ls[n] = true;
    for (int i = n - 1; i >= 0; i--) {
        ls[i] = (s[i] == s[i + 1]) ? ls[i + 1] : (s[i] < s[i + 1]);
    }
    V<int> sum_l(B + 1), sum_s(B + 1);
    for (int i = 0; i <= n; i++) {
        if (!ls[i])
            sum_s[s[i]]++;
        else
            sum_l[s[i] + 1]++;
    }
    for (int i = 0; i < B; i++) {
        sum_l[i + 1] += sum_s[i];
        sum_s[i + 1] += sum_l[i + 1];
```

```
    }

    auto induce = [&](const V<int>& lms) {
        fill(begin(sa), end(sa), -1);
        auto buf0 = sum_s;
        for (auto d : lms) {
            sa[buf0[s[d]]++] = d;
        }
        auto buf1 = sum_l;
        for (int i = 0; i <= n; i++) {
            int v = sa[i];
            if (v >= 1 && !ls[v - 1]) {
                sa[buf1[s[v - 1]]++] = v - 1;
            }
        }
        auto buf2 = sum_l;
        for (int i = n; i >= 0; i--) {
            int v = sa[i];
            if (v >= 1 && ls[v - 1]) {
                sa[--buf2[s[v - 1] + 1]] = v - 1;
            }
        }
    };

    V<int> lms, lms_map(n + 1, -1);
    for (int i = 1; i <= n; i++) {
        if (!ls[i - 1] && ls[i]) {
            lms_map[i] = int(lms.size());
            lms.push_back(i);
        }
    }

    induce(lms);

    if (lms.size() >= 2) {
        int m = int(lms.size()) - 1;
        V<int> lms2;
        for (int v : sa) {
            if (lms_map[v] != -1) lms2.push_back(v);
        }
        int rec_n = 1;
        V<int> rec_s(m);
        rec_s[lms_map[lms2[1]]] = 1;
        for (int i = 2; i <= m; i++) {
            int l = lms2[i - 1], r = lms2[i];
            int nl = lms[lms_map[l] + 1], nr = lms[lms_map[r] + 1];
            if (nl - l != nr - r)
                rec_n++;
            else {
                while (l <= nl) {
                    if (s[l] != s[r]) {
                        rec_n++;
                        break;
                    }
                    l++;
                    r++;
                }
            }
            rec_s[lms_map[lms2[i]]] = rec_n;
        }

        V<int> nx_lms;
        auto ch_sa = sa_is(rec_s, rec_n);
        for (int d : ch_sa) {
            nx_lms.push_back(lms[d]);
        }
        induce(nx_lms);
    }

    return sa;
}

template <class Str> V<int> doublingSA(Str s) {
    int n = (int)s.size();
    V<int> sa(n + 1);
    V<int> rsa(n + 1);
    iota(sa.begin(), sa.end(), 0);
    for (int i = 0; i <= n; i++) {
        rsa[i] = i < n ? s[i] : -1;
    }
    vector<int> tmp(n + 1);
    for (int k = 1; k <= n; k *= 2) {
        auto cmp = [&](int x, int y) {
            if (rsa[x] != rsa[y]) return rsa[x] < rsa[y];
            int rx = x + k <= n ? rsa[x + k] : -1;
            int ry = y + k <= n ? rsa[y + k] : -1;
            return rx < ry;
        };
        sort(sa.begin(), sa.end(), cmp);
        tmp[sa[0]] = 0;
        for (int i = 1; i <= n; i++) {
            tmp[sa[i]] = tmp[sa[i - 1]] + (cmp(sa[i - 1], sa[i]) ? 1 : 0);
        }
        copy(tmp.begin(), tmp.end(), begin(rsa));
    }
    return sa;
}

// tを完全に含む範囲を出力,O(|t|logn)
template <class Str> array<int, 2> find(const SA<Str>& sa, const string& t) {
    int n = (int)sa.s.size(), m = (int)t.size();
    array<int, 2> ans;
    int l, r;

    l = 0, r = n + 1;
    while (r - l > 1) {
        int md = (l + r) / 2;
        if (sa.s.compare(sa.sa[md], m, t) < 0) {
            l = md;
        } else {
            r = md;
        }
    }
    ans[0] = r;

    l = 0, r = n + 1;
    while (r - l > 1) {
        int md = (l + r) / 2;
        if (sa.s.compare(sa.sa[md], m, t) <= 0) {
            l = md;
        } else {
            r = md;
        }
    }
    ans[1] = r;
    return ans;
}
```

MP

```
template <class S> V<int> mp(const S& s) {
    int n = int(s.size());
    V<int> r(n + 1);
    r[0] = -1;
    for (int i = 0, j = -1; i < n; i++) {
        while (j >= 0 && s[i] != s[j]) j = r[j];
        j++;
        r[i + 1] = j;
    }
    return r;
}
```

Manacher

```
template <class S> V<int> manacher(const S& s) {
    int n = int(s.size());
    V<int> r(n);
    if (n == 0) return r;
    r[0] = 1;
    for (int i = 1, j = 0; i < n; i++) {
        int& k = r[i];
        k = (j + r[j] <= i) ? 0 : min(j + r[j] - i, r[2 * j - i]);
        while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) k++;
        if (j + r[j] < i + r[i]) j = i;
    }
    return r;
}


template <class S> V<int> manacher_even(const S& s) {
    int n = int(s.size());
    V<int> r(n + 1);
    for (int i = 1, j = 0; i < n; i++) {
        int& k = r[i];
        k = (j + r[j] <= i) ? 0 : min(j + r[j] - i, r[2 * j - i]);
        while (0 <= i - 1 - k && i + k < n && s[i - 1 - k] == s[i + k]) k++;
        if (j + r[j] < i + r[i]) j = i;
    }
    return r;
}
```

Z-Algorithm

```
// s[0..r[i]] == s[i..i+r[i]]
template <class S> V<int> z_algo(const S& s) {
    int n = int(s.size());
    V<int> r(n + 1);
    r[0] = 0;
    for (int i = 1, j = 0; i <= n; i++) {
        int& k = r[i];
        k = (j + r[j] <= i) ? 0 : min(j + r[j] - i, r[i - j]);
        while (i + k < n && s[k] == s[i + k]) k++;
        if (j + r[j] < i + r[i]) j = i;
    }
    r[0] = n;
    return r;
}
```

Run

```
struct RunExec {
    VV<pair<int, int>> runs;

    int n;
    V<int> a;

    V<int> rev(V<int> l) {
        reverse(l.begin(), l.end());
        return l;
    }

    V<int> sub_a(int l, int r) {
        return {a.begin() + l, a.begin() + r};
    }
    V<int> concat(V<int> l, const V<int>& r) {
        l.insert(l.end(), r.begin(), r.end());
        return l;
    }
```

```
    void run(int l, int r, int f) {
        if (l + 1 == r) return;
        int md = (l + r + f) / 2;
        run(l, md, f);
        run(md, r, f);
        auto z1 = z_algo(rev(sub_a(l, md)));
        auto z2 = z_algo(concat(sub_a(md, r), sub_a(l, r)));
        for (int i = md - 1; i >= l; i--) {
            int l1 = min(i - l, z1[md - i]);
            int l2 = min(r - md, z2[(r - l) - (md - i)]);
            int le = i - l1, ri = md + l2, peri = md - i;
            if (ri - le >= 2 * peri) runs[md - i].push_back({i - l1, md + l2});
        }
    }

    template <class Str>
    RunExec(Str _a) : a(_a) {
        n = int(a.size());
        runs.resize(n / 2 + 1);
        reverse(a.begin(), a.end());
        run(0, n, 0);
        for (auto& run: runs) {
            for (auto& p: run) {
                tie(p.first, p.second) =
                        make_pair(n - p.second, n - p.first);
            }
        }
        reverse(a.begin(), a.end());
        run(0, n, 1);

        for (auto& run: runs) {
            sort(run.begin(), run.end());
            V<pair<int, int>> res;
            for (auto p: run) {
                if (!res.empty() && p.second <= res.back().second) continue;
                res.push_back(p);
            }
            run = res;
        }
    }
};
```

SuffixAutomaton

```
struct SuffixAutomaton{
        SuffixAutomaton(int N){
                init(N);
        }
        SuffixAutomaton(string s){
                init(s.size());
                NP last = nodes[0];
                for(char c: s){
                        last = AddChar(last, c);
                }
                TopologicalSort();
        }
        void init(int N){
                assert(N>0);
                sz = 0;
                nodes = V<Node*>(2*N,nullptr);
                nodes[0] = MakeNode(0,false);
        }

        /*
                Node information

                構築時に作る情報を増やしたいなら、
                        - Node の定義
                        - MakeNode() での初期化
                        - clone のときの deep copy
                を変更する

                例えば Node v にマッチするsubstr を S(v) = {abb,aabb,caabb} とする
                このように一文字ずつ前に追加した形になる

                len :
                        マッチするsubstrのうちlongest (4)
                link:
                        shortest から更に削ったとき(bb) どこにマッチするか
                        reverse(s) の suffix tree になる (0がroot)

                なので shortest は v -> link -> len + 1
        */
        struct Node{
                using NP = Node*;

                int id;
                int len;
                NP link;
                bool isCloned;
                map<char,NP> next;

                Node(){}

                int getLongest(){
                        return len;
                }
                int getShortest(){
                        return link == nullptr ? 0 : link->len+1;
                }

                void putNext(char c, NP to){
                        next[c] = to;
                }
                bool hasNext(char c){
                        return next.count(c);
                }
                NP getNext(char c){
                        return hasNext(c) ? next[c] : nullptr;
                }
                V<pair<char,NP>> getAllTransitions(){
                        V<pair<char,NP>> res;
                        for(auto it: next) res.pb(it);
                        return res;
                }
        };
        using NP = Node*;

        int sz;                 // the number of nodes
        V<NP> nodes;

        NP MakeNode(int len, bool isCloned){
                nodes[sz] = new Node();
                nodes[sz] -> id = sz;
                nodes[sz] -> len = len;
                nodes[sz] -> link = nullptr;
                nodes[sz] -> isCloned = isCloned;
                return nodes[sz++];
        }
        /*
                Add c to nodes[last]
                return the new node id
        */
        NP AddChar(NP last, char c){
                NP cur = MakeNode(last->len+1, false);
                NP p;
                for(p = last; p && !p->hasNext(c); p = p->link){
                        p -> putNext(c,cur);
                }
                if(p == nullptr){
                        cur -> link = nodes[0];
                }else{
                        NP q = p -> getNext(c);
                        if(p->len+1 == q->len){
                                cur -> link = q;
                        }else{  //clone!
                                NP clone = MakeNode(p->len+1, true);
                                /*
                                        deep copy !
                                */
                                clone -> next = q -> next;
                                clone -> link = q -> link;
                                for(;p!=nullptr && p->getNext(c) == q; p =
p->link){
                                        p -> putNext(c,clone);
                                }
                                q -> link = cur -> link = clone;
                        }
                }
                return cur;
        }

        int where(string t){
                NP now = nodes[0];
                for(char c: t){
                        now = now -> getNext(c);
                        if(now == nullptr) return -1;
                }
                return now -> id;
        }

        void debug(){
                cerr << "======= DEBUG !! ========" << endl;
                rep(v,sz){
                        NP n = nodes[v];
                        cerr << "---- Node " << v << " ----" << endl;
                        for(auto p: n->next){
                                char c = p.fs;
                                int to = p.sc->id;
                                cerr << "  --(" << c << ")--> " << to << endl;
                        }
                        cerr << "  suf link : " << (n->link == nullptr ? -1 :
n->link->id) << endl;
                        cerr << "  shortest : " << n -> getShortest() << endl;
                        cerr << "  longest  : " << n -> getLongest() << endl;
                        cerr << endl;
                }
                cerr << "======= DEBUG END ========" << endl;
        }

        void TopologicalSort(){
                V<int> indeg(sz);
                V<NP> sorted(sz);
                rep(v,sz){
                        auto trans = nodes[v] -> getAllTransitions();
                        for(auto it: trans){
                                indeg[it.sc->id]++;
                        }
                }
                sorted[0] = nodes[0];
                int idx = 1;
                rep(i,sz){
                        NP n = sorted[i];
                        auto trans = nodes[n->id] -> getAllTransitions();
                        for(auto it: trans){
                                if(--indeg[it.sc->id] == 0){
                                        sorted[idx++] = it.sc;
                                }
```

```
                }
            }
            nodes = sorted;
            rep(i,sz) nodes[i]->id = i;
        }


        /*
            https://www.spoj.com/problems/SUBLEX/
            辞書順K番目 のsubstring
        */
        string GetKthSubstr(ll K){
            V<ll> dp(sz);
            for(int v=sz-1;v>=0;v--){
                dp[v] = 1;
                auto trans = nodes[v] -> getAllTransitions();
                for(auto it: trans){
                    int u = it.sc->id;
                    dp[v] += dp[u];
                }
            }
//          debug();
//          show(dp);

            K++;            //eps
            if(dp[0]<K) return "Eel";
            NP now = nodes[0];
            string res;
            while(K>1){
                K--;
                auto trans = now -> getAllTransitions();
                for(auto it: trans){
                    int u = it.sc->id;
                    if(K<=dp[u]){
                        res += it.fs;
                        now = it.sc;
                        break;
                    }else{
                        K -= dp[u];
                    }
                }
            }
            return res;
        }
};

void test(string s){
        SuffixAutomaton SA(s);

        V<string> substrs;
        int N = s.size();
        rep(i,N) for(int j=i;j<=N;j++){
            substrs.pb(s.substr(i,j-i));
        }
        sort(all(substrs));
        substrs.erase(unique(all(substrs)),substrs.end());
        int sz = SA.sz;
        VV<string> node2strs(sz);
        show(sz);
        SA.TopologicalSort();
        SA.debug();

        for(string t: substrs){
            int v = SA.where(t);
            assert(v != -1);
            node2strs[v].pb(t);
        }
        rep(v,sz){
            cerr << "-- v = " << v << " ----" << endl;
            for(string t: node2strs[v]) cerr << t << endl;
        }

        show(timer.ms());
}
```

幾何

Primitive

```
using D = double;
const D PI = acos(D(-1)), EPS = 1e-10;

int sgn(D a) { return (a < -EPS) ? -1 : (a > EPS); }
int sgn(D a, D b) { return sgn(a - b); }
struct P {
    D x, y;
    P(D _x = D(), D _y = D()) : x(_x), y(_y) {}
    P operator+(const P& r) const { return {x + r.x, y + r.y}; }
    P operator-(const P& r) const { return {x - r.x, y - r.y}; }
    P operator*(const P& r) const {
        return {x * r.x - y * r.y, x * r.y + y * r.x};
    }

    P operator*(const D& r) const { return {x * r, y * r}; }
    P operator/(const D& r) const { return {x / r, y / r}; }

    P& operator+=(const P& r) { return *this = *this + r; }
    P& operator-=(const P& r) { return *this = *this - r; }
    P& operator*=(const P& r) { return *this = *this * r; }
    P& operator*=(const D& r) { return *this = *this * r; }
    P& operator/=(const D& r) { return *this = *this / r; }
    P operator-() const { return {-x, -y}; }

    bool operator<(const P& r) const {
        return 2 * sgn(x, r.x) + sgn(y, r.y) < 0;
    }
    bool operator==(const P& r) const { return sgn((*this - r).rabs()) == 0; }
    bool operator!=(const P& r) const { return !(*this == r); }

    D norm() const { return x * x + y * y; }
    D abs() const { return sqrt(norm()); }
    D rabs() const { return max(std::abs(x), std::abs(y)); }  // robust abs
    D arg() const { return atan2(y, x); }

    pair<D, D> to_pair() const { return {x, y}; }
    static P polar(D le, D th) { return {le * cos(th), le * sin(th)}; }
};
ostream& operator<<(ostream& os, const P& p) {
    return os << "P(" << p.x << ", " << p.y << ")";
}

struct L {
    P s, t;
    L(P _s = P(), P _t = P()) : s(_s), t(_t) {}
    P vec() const { return t - s; }
    D abs() const { return vec().abs(); }
    D arg() const { return vec().arg(); }
};
ostream& operator<<(ostream& os, const L& l) {
    return os << "L(" << l.s << ", " << l.t << ")";
}

D crs(P a, P b) { return a.x * b.y - a.y * b.x; }
D dot(P a, P b) { return a.x * b.x + a.y * b.y; }
// cross(a, b) is too small?
int sgncrs(P a, P b) {
    D cr = crs(a, b);
    if (abs(cr) <= (a.rabs() + b.rabs()) * EPS) return 0;
    return (cr < 0) ? -1 : 1;
}

// -2, -1, 0, 1, 2 : front, clock, on, cclock, back
int ccw(P b, P c) {
    int s = sgncrs(b, c);
    if (s) return s;
    if (!sgn(c.rabs()) || !sgn((c - b).rabs())) return 0;
    if (dot(b, c) < 0) return 2;
    if (dot(-b, c - b) < 0) return -2;
    return 0;
}
int ccw(P a, P b, P c) { return ccw(b - a, c - a); }
int ccw(L l, P p) { return ccw(l.s, l.t, p); }
```

Intersect

```
/**
 * 幾何(衝突判定)
 */

P project(const L& l, const P& p) {
    P v = l.vec();
    return l.s + v * dot(v, p - l.s) / v.norm();
}

bool insSL(const L& s, const L& l) {
    int a = ccw(l, s.s), b = ccw(l, s.t);
    return (a % 2 == 0 || b % 2 == 0 || a != b);
}

bool insSS(const L& s, const L& t) {
    int a = ccw(s, t.s), b = ccw(s, t.t);
    int c = ccw(t, s.s), d = ccw(t, s.t);
    if (a * b <= 0 && c * d <= 0) return true;
    return false;
}

D distLP(const L& l, const P& p) {
    return abs(crs(l.vec(), p - l.s)) / l.abs();
}

D distSP(const L& s, const P& p) {
    P q = project(s, p);
    if (ccw(s, q) == 0)
        return (p - q).abs();
    else
        return min((s.s - p).abs(), (s.t - p).abs());
}

D distSS(const L& s, const L& t) {
    if (insSS(s, t)) return 0;
    return min(
        {distSP(s, t.s), distSP(s, t.t), distSP(t, s.s), distSP(t, s.t)});
}

int crossLL(const L& l, const L& m, P& r) {
    D cr1 = crs(l.vec(), m.vec()), cr2 = crs(l.vec(), l.t - m.s);
    if (sgncrs(l.vec(), m.vec()) == 0) {
        r = l.s;
        if (sgncrs(l.vec(), l.t - m.s)) return 0;
        return -1;
    }
    r = m.s + m.vec() * cr2 / cr1;
    return 1;
}

int crossSS(L l, L m, P& r) {
```

```
    int u = crossLL(l, m, r);
    if (u == 0) return 0;
    if (u == -1) {
        r = max(min(l.s, l.t), min(m.s, m.t));
        P q = min(max(l.s, l.t), max(m.s, m.t));
        return (q < r) ? 0 : (q == r ? 1 : -1);
    }
    if (ccw(l, r) == 0 && ccw(m, r) == 0) return 1;
    return 0;
}
```

Polygon

```
using Pol = V<P>;

D area2(const Pol& pol) {
    D u = 0;
    if (!pol.size()) return u;
    P a = pol.back();
    for (auto b : pol) u += crs(a, b), a = b;
    return u;
}

// (1:left) | (2: right) is inside between v[i] ~ v[i + 1]
V<pair<P, int>> insPolL(Pol pol, L l) {
    using Pi = pair<P, int>;
    V<Pi> v;
    P a, b = pol.back();
    for (auto now: pol) {
        a = b; b = now;
        P p;
        if (crossLL({a, b}, l, p) != 1) continue;
        int sa = ccw(l, a) % 2, sb = ccw(l, b) % 2;
        if (sa > sb) swap(sa, sb);
        if (sa != 1 && sb == 1) v.push_back({p, 1});
        if (sa == -1 && sb != -1) v.push_back({p, 2});
    }
    sort(v.begin(), v.end(), [&](Pi x, Pi y){
        return dot(l.vec(), x.first - l.s) < dot(l.vec(), y.first - l.s);
    });
    int m = int(v.size());
    V<Pi> res;
    for (int i = 0; i < m; i++) {
        if (i) v[i].second ^= v[i - 1].second;
        if (!res.empty() && res.back().first == v[i].first) res.pop_back();
        res.push_back(v[i]);
    }
    return res;
}

// 0: outside, 1: on line, 2: inside
int contains(const Pol& pol, P p) {
    if (!pol.size()) return 0;
    int in = -1;
    P _a, _b = pol.back();
    for (auto now : pol) {
        _a = _b;
        _b = now;
        P a = _a, b = _b;
        if (ccw(a, b, p) == 0) return 1;
        if (a.y > b.y) swap(a, b);
        if (!(a.y <= p.y && p.y < b.y)) continue;
        if (sgn(a.y, p.y) ? (crs(a - p, b - p) > 0) : (a.x > p.x)) in *= -1;
    }
    return in + 1;
}

// p must be sorted and uniqued!
Pol convex_down(const V<P>& ps) {
    assert(ps.size() >= 2);
    Pol dw;
    for (P d : ps) {
        size_t n;
        while ((n = dw.size()) > 1) {
            // if (ccw(dw[n - 2], dw[n - 1], d) != -1) break; // line上も取る
            if (ccw(dw[n - 2], dw[n - 1], d) == 1) break;
            dw.pop_back();
        }
        dw.push_back(d);
    }
    return dw;
}

Pol convex(V<P> ps) {
    sort(ps.begin(), ps.end());
    ps.erase(unique(ps.begin(), ps.end()), ps.end());
    if (ps.size() <= 1) return ps;
    Pol dw = convex_down(ps);
    reverse(ps.begin(), ps.end());
    Pol up = convex_down(ps);
    dw.insert(dw.begin(), up.begin() + 1, up.end() - 1);
    return dw;
}

Pol convex_cut(const Pol& po, const L& l) {
    if (!po.size()) return Pol{};
    Pol q;
    P a, b = po.back();
    for (auto now : po) {
        a = b;
        b = now;
        if ((ccw(l, a) % 2) * (ccw(l, b) % 2) < 0) {
            P buf;
            crossLL(l, L(a, b), buf);
            q.push_back(buf);
        }
        if (ccw(l, b) != -1) q.push_back(b);
    }
    return q;
}

// pol must be convex
D diameter(const Pol& p) {
    int n = int(p.size());
    if (n == 2) return (p[1] - p[0]).abs();
    int x = 0, y = 0;
    for (int i = 1; i < n; i++) {
        if (p[i] < p[x]) x = i;
        if (p[y] < p[i]) y = i;
    }
    D ans = 0;
    int sx = x, sy = y;
    /*
    do {
        ...
    } while (sx != x || sy != y);
    で1周する
    */
    while (sx != y || sy != x) {
        ans = max(ans, (p[x] - p[y]).abs());
        int nx = (x + 1 < n) ? x + 1 : 0, ny = (y + 1 < n) ? y + 1 : 0;
        if (crs(p[nx] - p[x], p[ny] - p[y]) < 0)
            x = nx;
        else
            y = ny;
    }
    return ans;
}
```

Circle

```
struct C {
    P p;
    D r;
    C(P _p = P(), D _r = D()) : p(_p), r(_r) {}
};

// need Intersect/distLP, r.sはよりl.sに近い
int crossCL(const C& c, const L& l, L& r) {
    D u = distLP(l, c.p);
    int si = sgn(u, c.r);
    if (si == 1) return 0;
    P v = project(l, c.p);
    P di = (si == 0) ? P(0, 0) : l.vec() * (sqrt(c.r * c.r - u * u) / l.abs());
    r = L(v - di, v + di);
    if (si == 0) return 1;
    return 2;
}

// need Intersect/distLP, r.sはよりl.sに近い
int crossCS(const C& c, const L& s, L& l) {
    if (!crossCL(c, s, l)) return 0;
    bool f1 = !ccw(s, l.s), f2 = !ccw(s, l.t);
    if (f1 && f2) return 2;
    if (!f1 && !f2) return 0;
    if (f1)
        l.t = l.s;
    else
        l.s = l.t;
    return 1;
}

// return number of cross point
// l, rはcから見た交点の角度、[l, r]がdに覆われている
int crossCC(const C& c, const C& d, D& l, D& r) {
    if (c.p == d.p) {
        assert(sgn(c.r - d.r));  // prohibit same circles
        return 0;
    }
    D di = (c.p - d.p).abs(), bth = (d.p - c.p).arg();
    D costh = (c.r * c.r + di * di - d.r * d.r) / (2 * c.r * di);
    int ty = min(sgn(c.r + d.r, di), sgn(di, abs(c.r - d.r)));
    if (ty == -1) return 0;
    if (ty == 0) costh = sgn(costh);
    D th = acos(costh);
    l = bth - th;
    r = bth + th;
    return ty + 1;
}

// pからcに接線を引く，交点はp1, p2
int tangent(const C& c, const P& p, P& p1, P& p2) {
    D di = (c.p - p).abs();
    int si = sgn(c.r, di);
    if (si == 1) return 0;
    D th = acos(min(D(1), c.r / di));
    D ar = (p - c.p).arg();
    p1 = c.p + P::polar(c.r, ar - th);
    p2 = c.p + P::polar(c.r, ar + th);
    if (si == 0) return 1;
    return 2;
}

//共通接線
int tangent(const C& c, const C& d, L& l, L& r, bool inter) {
    D di = (d.p - c.p).abs(), ar = (d.p - c.p).arg();
    if (sgn(di) == 0) {
        assert(sgn(c.r - d.r));  // prohibit same circles
        return 0;
```

```
    }
    D costh = c.r + (inter ? d.r : -d.r);
    int si = sgn(abs(costh), di);
    costh /= di;
    if (si == 1)
        return 0;
    else if (si == 0)
        costh = sgn(costh);
    D th = acos(costh);
    P base;
    base = P::polar(1, ar - th);
    l = L(c.p + base * c.r, d.p + base * d.r * (inter ? -1 : 1));
    base = P::polar(1, ar + th);
    r = L(c.p + base * c.r, d.p + base * d.r * (inter ? -1 : 1));
    if (si == 0) return 1;
    return 2;
}

C circum_circle(P a, P b, P c) {
    b -= a;
    c -= a;
    D s = 2 * crs(b, c);
    D x = (b - c).norm(), y = c.norm(), z = b.norm();
    D S = x + y + z;
    P r = (b * (S - 2 * y) * y + c * z * (S - 2 * z)) / (s * s);
    return C(r + a, r.abs());
}

C smallest_circle(const Pol& p, int i, array<P, 3> q, int j) {
    if (i == int(p.size())) {
        switch (j) {
            case 0:
                return C(P(0, 0), -1);
            case 1:
                return C(q[0], 0);
            case 2:
                return C((q[0] + q[1]) / D(2.0), (q[0] - q[1]).abs() / D(2.0));
            case 3:
                return circum_circle(q[0], q[1], q[2]);
        }
        assert(false);
    }
    C c = smallest_circle(p, i + 1, q, j);
    if (sgn((p[i] - c.p).abs(), c.r) == 1) {
        q[j] = p[i];
        return smallest_circle(p, i + 1, q, j + 1);
    }
    return c;
}
C smallest_circle(Pol p) {
    random_shuffle(begin(p), end(p));
    return smallest_circle(p, 0, array<P, 3>(), 0);
}

// C(P(0, 0), r)とTri((0, 0), a, b)の共有面積
D area2CT(const C& c, const P& _a, const P& _b) {
    P a = _a - c.p, b = _b - c.p;
    D r = c.r;
    if (a == b) return 0;
    auto single = [&](P x, P y, bool tri) {
        if (tri)
            return crs(x, y);
        else
            return r * r * ((y * P(x.x, -x.y)).arg());
    };
    bool ia = sgn(a.abs(), r) != 1, ib = sgn(b.abs(), r) != 1;
    if (ia && ib) return single(a, b, true);
    L l;
    if (!crossCS(C(P(0, 0), r), L(a, b), l)) return single(a, b, false);
    return single(a, l.s, ia) + single(l.s, l.t, true) + single(l.t, b, ib);
}

// p, cの共有面積
D area2CPol(const C& c, const Pol& po) {
    D sm = 0;
    P a, b = po.back();
    for (auto p : po) {
        a = b;
        b = p;
        sm += area2CT(c, a, b);
    }
    return sm;
}
```

複雑系

```
/*
return arg(l) < arg(r)
arg(-1, 0) = PI, arg(0, 0) = arg(1, 0) = 0
*/
int argcmp(P l, P r) {
    auto psgn = [&](P p) {
        if (int u = sgn(p.y)) return u;
        if (sgn(p.x) == -1) return 2;
        return 0;
    };
    int lsgn = psgn(l), rsgn = psgn(r);
    if (lsgn < rsgn) return -1;
    if (lsgn > rsgn) return 1;
    return sgncrs(r, l);
}

V<L> halfplane_intersects(V<L> lines) {
    sort(lines.begin(), lines.end(), [&](const L& a, const L& b) {
        if (int u = argcmp(a.vec(), b.vec())) return u == -1;
        return sgncrs(a.vec(), b.s - a.s) < 0;
    });
    lines.erase(unique(lines.begin(), lines.end(),
                       [&](const L& a, const L& b) {
                           return argcmp(a.vec(), b.vec()) == 0;
                       }),
                lines.end());

    deque<L> st;
    for (auto l : lines) {
        bool err = false;
        auto is_need = [&](L a, L b, L c) {
            D ab_dw = crs(a.vec(), b.vec()), ab_up = crs(a.vec(), a.t - b.s);
            D bc_dw = crs(b.vec(), c.vec()), bc_up = crs(c.t - b.s, c.vec());
            if (ab_dw <= 0 || bc_dw <= 0) return true;
            bool f = bc_up * ab_dw > bc_dw * ab_up;
            if (!f && crs(a.vec(), c.vec()) < 0) err = true;
            return f;
        };
        while (st.size() >= 2 && !is_need(l, st[0], st[1])) st.pop_front();
        while (st.size() >= 2 &&
               !is_need(st[st.size() - 2], st[st.size() - 1], l))
            st.pop_back();
        if (st.size() < 2 || is_need(st.back(), l, st.front())) st.push_back(l);
        if (err) return {};
    }
    if (st.size() == 2 && !sgncrs(st[0].vec(), st[1].vec()) &&
        sgncrs(st[0].vec(), st[1].s - st[0].s) <= 0)
        return {};

    return V<L>(st.begin(), st.end());
}

struct Arrange {
    V<P> ps;
    VV<int> g;
    Arrange(const V<L>& l) {
        int n = int(l.size());
        for (int i = 0; i < n; i++) {
            ps.push_back(l[i].s);
            ps.push_back(l[i].t);
            for (int j = i + 1; j < n; j++) {
                P p;
                if (crossSS(l[i], l[j], p) != 1) continue;
                ps.push_back(p);
            }
        }
        sort(ps.begin(), ps.end());
        ps.erase(unique(ps.begin(), ps.end()), ps.end());
        int m = int(ps.size());
        g = VV<int>(m);
        for (int i = 0; i < n; i++) {
            V<int> v;
            for (int j = 0; j < m; j++) {
                if (!ccw(l[i].s, l[i].t, ps[j])) v.push_back(j);
            }
            sort(v.begin(), v.end(), [&](int x, int y) {
                return (ps[x] - l[i].s).rabs() < (ps[y] - l[i].s).rabs();
            });
            for (int j = 0; j < int(v.size()) - 1; j++) {
                g[v[j]].push_back(v[j + 1]);
                g[v[j + 1]].push_back(v[j]);
            }
        }

        for (int i = 0; i < m; i++) {
            sort(g[i].begin(), g[i].end());
            g[i].erase(unique(g[i].begin(), g[i].end()), g[i].end());
        }
    }
};

struct DualGraph {
    V<Pol> pols;
    VV<int> g;
    V<int> tr;

    DualGraph(V<P> ps, VV<int> pg) {
        // prohibit self-loop, multi edge
        int n = int(ps.size());
        using Pi = pair<int, int>;
        map<Pi, int> mp;
        for (int i = 0; i < n; i++) {
            if (pg[i].empty()) continue;
            sort(pg[i].begin(), pg[i].end(), [&](int l, int r) {
                return (ps[l] - ps[i]).arg() < (ps[r] - ps[i]).arg();
            });
            int a, b = pg[i].back();
            for (int now : pg[i]) {
                a = b;
                b = now;
                mp[{b, i}] = a;
            }
        }

        map<Pi, int> vis;
        int m = 0;
        for (int i = 0; i < n; i++) {
            for (int j : pg[i]) {
                if (vis.count({i, j})) continue;
                int id = m++;
                pols.push_back({});
                Pi pi = {i, j};
                while (!vis.count(pi)) {
                    vis[pi] = id;
                    pols.back().push_back(ps[pi.first]);
                    pi = {pi.second, mp[pi]};
```

```
                }
            }
        }

        g = VV<int>(m);
        for (int i = 0; i < n; i++) {
            for (int j : pg[i]) {
                g[vis[{i, j}]].push_back(vis[{j, i}]);
            }
        }
    }
    V<int> scan(V<P> que, D eps) {
        int n = int(pols.size()), m = int(que.size());
        tr = V<int>(n);
        iota(tr.begin(), tr.end(), 0);
        struct S {
            P s, t;
            int id;
            D get_y(D x) const {
                if (!sgn(s.x, t.x)) return s.y;
                return (s.y * (t.x - x) + t.y * (x - s.x)) / (t.x - s.x);
            }
            bool operator<(const S& r) const {
                D x = (max(s.x, r.s.x) + min(t.x, r.t.x)) / 2;
                return get_y(x) < r.get_y(x);
            }
        };
        struct Q {
            D x;
            int ty;
            S l;
        };
        V<Q> ev;
        for (int i = 0; i < int(que.size()); i++) {
            auto p = que[i];
            ev.push_back({p.x, 0, {p, p, n + i}});
        }
        for (int ph = 0; ph < n; ph++) {
            auto v = pols[ph];
            P a, b = v.back();
            for (auto now : v) {
                a = b;
                b = now;
                if (sgn(b.x, a.x) == -1) {
                    ev.push_back({b.x, 2, {b, a, ph}});
                    ev.push_back({a.x, 1, {b, a, ph}});
                }
            }
            if (area2(v) <= eps) {
                P mi = *min_element(v.begin(), v.end());
                tr[ph] = -1;
                ev.push_back({mi.x, 0, {mi, mi, ph}});
            }
        }
        sort(ev.begin(), ev.end(), [&](Q a, Q b) {
            if (sgn(a.x, b.x)) return sgn(a.x, b.x) == -1;
            return a.ty < b.ty;
        });
        V<int> res(m);
        set<S> st;
        for (auto e : ev) {
            if (e.ty == 0) {
                // get
                auto it = st.lower_bound(e.l);
                int u = (it == st.end() ? -1 : tr[it->id]);
                if (e.l.id < n)
                    tr[e.l.id] = u;
                else
                    res[e.l.id - n] = u;
            } else {
                if (e.ty == 1)
                    st.erase(e.l);
                else
                    st.insert(e.l);
            }
        }
        return res;
    }
};
```

3D

```
struct Pt3 {
    D x, y, z;
    Pt3() {}
    Pt3(D _x, D _y, D _z) : x(_x), y(_y), z(_z) {}
    Pt3 operator+(const Pt3& r) const { return Pt3(x+r.x, y+r.y, z+r.z); }
    Pt3 operator-(const Pt3& r) const { return Pt3(x-r.x, y-r.y, z-r.z); }
    Pt3& operator-=(const Pt3& r) { return *this = *this - r; }
    Pt3 operator-() const { return Pt3(-x, -y, -z); }

    D abs() const { return sqrt(x * x + y * y + z * z); }
    D rabs() const { return max({std::abs(x), std::abs(y), std::abs(z)}); }
};

struct L3 {
    Pt3 s, t;
    L3() {}
    L3(Pt3 s, Pt3 t) : s(s), t(t) {}
    Pt3 vec() const { return t - s; }
    D abs() const { return vec().abs(); }
};

/*
cng viewpoint
l.sを(0, 0, 0), l.tを(0, 0, l.abs()), pを(0, y>0, z)へ移したときのqの座標を返す
*/
Pt3 cng_vp(L3 l, Pt3 p, Pt3 q) {
    l.t -= l.s; p -= l.s; q -= l.s;

    Pt2 base;
    base = Pt2::polar(1, PI / 2 - Pt2(l.t.x, l.t.y).arg());
    tie(l.t.x, l.t.y) = (Pt2(l.t.x, l.t.y) * base).to_pair();
    tie(p.x, p.y) = (Pt2(p.x, p.y) * base).to_pair();
    tie(q.x, q.y) = (Pt2(q.x, q.y) * base).to_pair();

    base = Pt2::polar(1, PI / 2 - Pt2(l.t.y, l.t.z).arg());
    tie(l.t.y, l.t.z) = (Pt2(l.t.y, l.t.z) * base).to_pair();
    tie(p.y, p.z) = (Pt2(p.y, p.z) * base).to_pair();
    tie(q.y, q.z) = (Pt2(q.y, q.z) * base).to_pair();

    base = Pt2::polar(1, PI / 2 - Pt2(p.x, p.y).arg());
    tie(p.x, p.y) = (Pt2(p.x, p.y) * base).to_pair();
    tie(q.x, q.y) = (Pt2(q.x, q.y) * base).to_pair();

    return q;
}
```

Visualizer

```
struct Vis {
    struct Col {
        int r, g, b;
    };
    FILE* fp;
    D off, f;
    Vis(string s, D d, D u) : off(d), f(1000 / (u - d)) {
        fp = fopen(s.c_str(), "w");
        fprintf(
                fp,
                "<html><head><script>"
                "onload = function draw() {"
                "var cv = document.getElementById('c');"
                "var ct = cv.getContext('2d');");
    }
    D norm(D x) { return (x - off) * f; }
    void set_col(Col c) {
        fprintf(fp, "ct.fillStyle = \"rgb(%d, %d, %d)\";", c.r, c.g, c.b);
        fprintf(fp, "ct.strokeStyle = \"rgb(%d, %d, %d)\";", c.r, c.g, c.b);
    }
    void line(L l, Col col = {0, 0, 0}) {
        set_col(col);
        fprintf(fp,
                "ct.beginPath();"
                "ct.moveTo(%Lf, %Lf);"
                "ct.lineTo(%Lf, %Lf);"
                "ct.closePath();"
                "ct.stroke();",
                norm(l.s.x), norm(l.s.y), norm(l.t.x), norm(l.t.y));
    }
    void point(P p, Col col = {0, 0, 0}) {
        set_col(col);
        fprintf(fp,
                "ct.beginPath();"
                "ct.arc(%Lf, %Lf, 2, 0, Math.PI * 2, true);"
                "ct.fill();",
                norm(p.x), norm(p.y));
    }
    ~Vis() {
        fprintf(fp,
                "}</script></head><body>"
                "<canvas id=\"c\" width=\"1000\" height=\"1000\">"
                "</canvas></body></html>");
        fclose(fp);
    }
};
```

メモ

$$\int_a^b f(x)dx = \frac{b-a}{6}(f(a) + 4f(\frac{a+b}{2}) + f(b))$$

$$\int_a^b f(x)dx = \frac{b-a}{90}(7f(a) + 32f(\frac{3a+b}{4}) + 12f(\frac{2a+2b}{4}) + 32f(\frac{a+3b}{4}) + 7f(b))$$

$$\int_a^b f(x)dx = \frac{b-a}{3}(2f(\frac{3a+b}{4}) - f(\frac{2a+2b}{4}) + 2f(\frac{a+3b}{4}))$$

$$\lfloor \frac{a}{b} \rfloor = a\lceil \frac{2^{64}}{b} \rceil \gg 64 (b \leq 2^{31})$$