ICPC Template Library

postpone | `Shu-i64`

November 14, 2025

v1.0 ●●●●●

ref: jiangly, skip2004, zeemanz, capps
Attention: 0-indexed, $[l, r)$.

Contents

蒋凌宇

1 Data Structure

1.1 Disjoint Set Union

```cpp
struct DSU {
    std::vector<int> f, siz;
    DSU() {}
    DSU(int n) {
        init(n);
    }
    void init(int n) {
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }
    int find(int x) {
        while (x != f[x]) {
            x = f[x] = f[f[x]];
        }
        return x;
    }
    bool same(int x, int y) {
        return find(x) == find(y);
    }
    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }
    int size(int x) {
        return siz[find(x)];
    }
};

// DSU with Rollback
struct DSU {
    std::vector<std::pair<int &, int>> his;
    std::vector<int> f, siz;
    DSU () {}
    DSU(int n) {
        init(n);
    }
    void init(int n) {
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        siz.assign(n, 1);
    }
    void set(int &a, int b) {
        his.emplace_back(a, a);
        a = b;
    }
    int find(int x) {
        while (x != f[x]) {
            x = f[x];
        }
        return x;
    }
    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        if (siz[x] < siz[y]) {
            std::swap(x, y);
        }
        set(siz[x], siz[x] + siz[y]);
        set(f[y], x);
        return true;
    }
    bool same(int x, int y) {
        return find(x) == find(y);
    }
    int cur() {
        return his.size();
    }
    void rollback(int t) {
        while (his.size() > t) {
            auto [x, y] = his.back();
            x = y;
            his.pop_back();
        }
    }
};

// Maintain whether each connected component is bipartite
struct DSU {
    std::vector<std::pair<int &, int>> his;
    int n;
    std::vector<int> f, g, bip;
    DSU(int n_) : n(n_), f(n, -1), g(n), bip(n, 1) {}
    std::pair<int, int> find(int x) {
        if (f[x] < 0) {
            return {x, 0};
        }
        auto [u, v] = find(f[x]);
        return {u, v ^ g[x]};
    }
    void set(int &a, int b) {
        his.emplace_back(a, a);
        a = b;
    }
    void merge(int a, int b, int &ans) {
        auto [u, xa] = find(a);
        auto [v, xb] = find(b);
        int w = xa ^ xb ^ 1;
        if (u == v) {
            if (bip[u] && w) {
                set(bip[u], 0);
                ans--;
            }
            return;
        }
        if (f[u] > f[v]) {
            std::swap(u, v);
        }
        ans -= bip[u];
        ans -= bip[v];
        set(bip[u], bip[u] && bip[v]);
        set(f[u], f[u] + f[v]);
        set(f[v], u);
        set(g[v], w);
        ans += bip[u];
    }
    int cur() {
        return his.size();
    }
    void rollback(int t) {
        while (his.size() > t) {
            auto [x, y] = his.back();
            x = y;
            his.pop_back();
        }
    }
};
```

1.2 Fenwick Tree

```cpp
template <typename T>
struct Fenwick {
    int n;
    std::vector<T> a;
    Fenwick(int n_ = 0) {
        init(n_);
    }
    void init(int n_) {
        n = n_;
        a.assign(n, T{});
    }
    void add(int x, const T &v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] = a[i - 1] + v;
        }
    }
    T sum(int x) {
        T ans{};
        for (int i = x; i > 0; i -= i & -i) {
            ans = ans + a[i - 1];
        }
        return ans;
    }
    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }
    int select(const T &k) {
        int x = 0;
        T cur{};
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && cur + a[x + i - 1] <= k) {
                x += i;
                cur = cur + a[x - 1];
            }
        }
        return x;
    }
};
```

1.3 Lazy Segment Tree

```cpp
template <class Info, class Tag>
struct LazySegmentTree {
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;

    LazySegmentTree() = delete;
    LazySegmentTree(int n_, const Info &v_ = {}) { init(std::vector<Info>(n_, v_)); }
    template <class T>
    LazySegmentTree(const std::vector<T> &data) { init(data); }

    template <class T>
    void init(const std::vector<T> &data) {
        n = data.size();
        info.assign(4 << std::__lg(n), {});
        tag.assign(4 << std::__lg(n), {});

        auto build = [&](auto self, int p, int l, int r) -> void {
            if (r - l == 1) {
                info[p] = data[l];
                return;
            }
            int m = (l + r) / 2;
            self(self, 2 * p, l, m);
            self(self, 2 * p + 1, m, r);
            pull(p);
        };
        build(build, 1, 0, n);
    }

    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void apply(int p, const Tag &v) {
        info[p].apply(v);
        tag[p].apply(v);
    }
    void push(int p) {
        apply(2 * p, tag[p]);
        apply(2 * p + 1, tag[p]);
        tag[p] = {};
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p);
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return {};
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p);
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    void rangeApply(int p, int l, int r, int x, int y, const Tag &v) {
        if (l >= y || r <= x) {
            return;
        }
        if (l >= x && r <= y) {
            apply(p, v);
            return;
        }
        int m = (l + r) / 2;
        push(p);
        rangeApply(2 * p, l, m, x, y, v);
        rangeApply(2 * p + 1, m, r, x, y, v);
        pull(p);
    }
    void rangeApply(int l, int r, const Tag &v) {
        return rangeApply(1, 0, n, l, r, v);
    }
    template <class F>
    int findFirst(int p, int l, int r, int x, int y, const F &pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        push(p);
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template <class F>
    int findFirst(int l, int r, const F &pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template <class F>
    int findLast(int p, int l, int r, int x, int y, const F &pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        push(p);
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template <class F>
    int findLast(int l, int r, const F &pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};

struct Tag {
    void apply(const Tag &t) {
    }
};

struct Info {
    void apply(const Tag &t) {
    }
};

Info operator+(const Info &a, const Info &b) {
}
```

1.4 Sparse Table

```cpp
// O(n log n) - O(1)
// template <class T, T e, T (*F)(T, T)>, if cpp < 20
template <class T, T e, auto F>
struct SparseTable {
    int n;
    std::vector<std::vector<T>> a;
    SparseTable(const std::vector<T> &v = {}) {
        init(v);
    }
    void init(const std::vector<T> &v) {
        n = v.size();
        if (n == 0) {
            return;
        }
        const int m = std::__lg(n);
        a.assign(m + 1, std::vector<T>(n));
        a[0] = v;
        for (int j = 0; j < m; j++) {
            for (int i = 0; i + (2 << j) <= n; i++) {
                a[j + 1][i] = F(a[j][i], a[j][i + (1 << j)]);
            }
        }
    }
    T operator()(int l, int r) const {
        if (l >= r) {
            return e;
        } else {
            const int k = std::__lg(r - l);
            return F(a[k][l], a[k][r - (1 << k)]);
        }
    }
};

// O(n) - O(1)
// only for minmax
template<class T, T e, class Cmp = std::less<>>
struct SparseTable {
    const Cmp cmp = Cmp();
    static constexpr unsigned B = 64;
    int n;
    std::vector<std::vector<T>> a;
    std::vector<T> pre, suf, ini;
    std::vector<u64> stk;
    SparseTable(const std::vector<T> &v = {}) {
        init(v);
    }
    void init(const std::vector<T> &v) {
        n = v.size();
        if (n == 0) {
            return;
        }
        pre = suf = ini = v;
        stk.resize(n);
        const int M = (n - 1) / B + 1;
        const int lg = std::__lg(M);
        a.assign(lg + 1, std::vector<T>(M));
        for (int i = 0; i < M; i++) {
            a[0][i] = v[i * B];
            for (int j = 1; j < B && i * B + j < n; j++) {
                a[0][i] = std::min(a[0][i], v[i * B + j], cmp);
            }
        }
        for (int i = 1; i < n; i++) {
            if (i % B) {
                pre[i] = std::min(pre[i], pre[i - 1], cmp);
            }
        }
        for (int i = n - 2; i >= 0; i--) {
            if (i % B != B - 1) {
                suf[i] = std::min(suf[i], suf[i + 1], cmp);
            }
        }
        for (int j = 0; j < lg; j++) {
            for (int i = 0; i + (2 << j) <= M; i++) {
                a[j + 1][i] = std::min(a[j][i], a[j][i + (1 << j)], cmp);
            }
        }
        for (int i = 0; i < M; i++) {
            const int l = i * B;
            const int r = std::min(1U * n, l + B);
            u64 s = 0;
            for (int j = l; j < r; j++) {
                while (s && cmp(v[j], v[std::__lg(s) + l])) {
                    s ^= 1ULL << std::__lg(s);
                }
                s |= 1ULL << (j - l);
                stk[j] = s;
            }
        }
    }
    T operator()(int l, int r) {
        if (l >= r) {
            return e;
        }
        if (l / B != (r - 1) / B) {
            T ans = std::min(suf[l], pre[r - 1], cmp);
            l = l / B + 1;
            r = r / B;
            if (l < r) {
                int k = std::__lg(r - l);
                ans = std::min({ans, a[k][l], a[k][r - (1 << k)]}, cmp);
            }
            return ans;
        } else {
            int x = B * (l / B);
            return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + l];
        }
    }
};
```

1.5 Mo

```cpp
{
    std::vector<std::array<int, 3>> ask(q);
    // ...
    std::ranges::sort(ask, [&](auto i, auto j) {
        if (i[0] / B != j[0] / B) {
            return i[0] < j[0];
        }
        return (i[0] / B) % 2 ? i[1] > j[1] : i[1] < j[1];
    });
    // ...
    // [l, r)
    int L = 0, R = 0;
    for (auto [l, r, i] : ask) {
        while (L > l) {
            add(--L);
        }
        while (R < r) {
            add(R++);
        }
        while (L < l) {
            del(L++);
        }
        while (R > r) {
            del(--R);
        }
        // ans[i] = ?
    }
}

// Mo With Modify
// B = n ^ (2 / 3)
{
    // ...
    std::vector<std::array<int, 4>> ask;
    std::vector<std::pair<int, int>> mod;
    for (int i = 0; i < m; i++) {
        char o;
        int x, y;
        std::cin >> o >> x >> y;
        x--;
        if (modify) {
            mod.emplace_back(x, y);
        } else {
            ask.push_back({x, y, (int)mod.size(), (int)ask.size()});
        }
    }
    std::ranges::sort(ask, [&](auto i, auto j) {
        if (i[0] / B != j[0] / B)
            return i[0] < j[0];
        if (i[1] / B != j[1] / B)
            return i[1] < j[1];
        return (i[1] / B) & 1 ? i[2] < j[2] : i[2] > j[2];
    });
    auto add = [&](int c) {
        // ...
    };
    auto del = [&](int c) {
        // ...
    };
    auto modify = [&](int p, int l, int r) {
        // if (l <= x and x < r) then del(ori) and add(cur)

        // first modify : x -> y
        // second modify : y -> x
        // using swap
    };
    // [l, r)
    int L = 0, R = 0, T = 0;
    for (auto [l, r, t, i] : que) {
        while (l < L) {
            add(a[--L]);
        }
        while (R < r) {
            add(a[R++]);
        }
        while (L < l) {
            del(a[L++]);
        }
        while (R > r) {
            del(a[--R]);
        }
        while (T < t) {
            modify(T++, l, r);
        }
        while (T > t) {
            modify(--T, l, r);
        }
        // ans[i] = ?
    }
}
```

1.6 Cartesian Tree

```
// root = rangeMin
vector<int> lc(n, -1), rc(n, -1);
vector<int> stk;
for (int i = 0; i < n; i++) {
    while (not stk.empty() and p[i] < p[stk.back()]) {
        int x = stk.back();
        stk.pop_back();

        rc[x] = lc[i];
        lc[i] = x;
    }
    stk.push_back(i);
}
while (stk.size() > 1) {
    int x = stk.back();
    stk.pop_back();
    rc[stk.back()] = x;
}
```

1.7 Lichao Segment Tree

```
template <class T, T inf, class C = std::less<>>
struct LiChaoSegmentTree {
    static constexpr C cmp = {};
    struct Line {
        int i;
        T k, b;
        constexpr Line(int i = std::min(-1, 1, cmp), T k = 0, T b = std::max
          (-inf, inf, cmp)) : i{i}, k{k}, b{b} {}
        constexpr std::pair<T, int> operator()(int x) const {
            return {k * x + b, i};
        }
    };
    struct Node {
        Node *l, *r;
        Line f;
        Node() : l{}, r{}, f{} {}
    };
    int n;
    Node *t;
    LiChaoSegmentTree(int n = 0) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        t = nullptr;
    }
    void insert(Node *&p, int l, int r, int x, int y, Line f) {
        if (l >= y || r <= x) {
            return;
        }
        if (p == nullptr) {
            p = new Node();
        }
        int m = (l + r) / 2;
        if (l >= x && r <= y) {
            if (cmp(f(m), p->f(m))) {
                std::swap(f, p->f);
            }
            if (r - l == 1) {
                return;
            }
```

```cpp
            if (cmp(f(l), p->f(l))) {
                insert(p->l, l, m, x, y, f);
            } else {
                insert(p->r, m, r, x, y, f);
            }
        } else {
            insert(p->l, l, m, x, y, f);
            insert(p->r, m, r, x, y, f);
        }
    }
    void insert(int l, int r, Line f) {
        insert(t, 0, n, l, r, f);
    }
    void insert(Line f) {
        insert(t, 0, n, 0, n, f);
    }
    // {val, id}
    std::pair<T, int> query(Node *p, int l, int r, int x) {
        if (p == nullptr) {
            return Line()(x);
        }
        if (r - l == 1) {
            return p->f(x);
        }
        int m = (l + r) / 2;
        if (x < m) {
            return std::min(p->f(x), query(p->l, l, m, x), cmp);
        } else {
            return std::min(p->f(x), query(p->r, m, r, x), cmp);
        }
    }
    std::pair<T, int> query(int x) {
        return query(t, 0, n, x);
    }
};
```

2 Math

2.1 Formula

- Ex Euler's Theorem:

$$a^b \equiv \begin{cases} a^{b \bmod \varphi(m)}, & \gcd(a,b) = 1 \\ a^{(b \bmod \varphi(m)) + \varphi(m)}, & \gcd(a,b) \neq 1, b \geq \varphi(m) \end{cases} \pmod{m}$$

- Euclidean algorithm: $\gcd(a,b) = \gcd(b, a \bmod b)$
- Binomial Inversion

  $f(n) = |\bigcap_{1 \leq i \leq n} A_i|, g(n) = |\bigcap_{1 \leq i \leq n} A_i^C|$

$$f(n) = \sum_{i=0}^{n} (-1)^i \binom{n}{i} g(i)$$

$$g(n) = \sum_{i=0}^{n} (-1)^i \binom{n}{i} f(i)$$

$$f(n) = \sum_{i=0}^{n} \binom{n}{i} h(i)$$

$$h(n) = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} f(i)$$

$f(n)$: select at least n. $\leftrightarrow$ $g(n)$: select exactly n.

$$f(n) = \sum_{i=n}^{m} \binom{i}{n} g(i)$$

$$g(n) = \sum_{i=n}^{m} (-1)^{i-n} \binom{i}{n} f(i)$$

2.2 Exgcd

```cpp
// ax + by = gcd(a, b)
// if a != 0, b != 0 then -b <= x <= b, -a <= y <= a
int exgcd(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int g = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return g;
}

// {x, mod} -> {_, inv(x)}
constexpr std::pair<i64, i64> invGcd(i64 a, i64 b) {
    a %= b;
    if (a < 0) {
        a += b;
    }
    if (a == 0) {
        return {b, 0};
    }
    i64 s = b, t = a;
    i64 m0 = 0, m1 = 1;
    while (t) {
        i64 u = s / t;
        s -= t * u;
        m0 -= m1 * u;

        std::swap(s, t);
        std::swap(m0, m1);
    }
    if (m0 < 0) {
        m0 += b / s;
    }
    return {s, m0};
}
```

2.3 Phi

```cpp
int phi(int n) {
    int res = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0) {
                n /= i;
            }
            res = res / i * (i - 1);
        }
    }
    if (n > 1) {
        res = res / n * (n - 1);
    }
    return res;
}
```

2.4 Sieve

```cpp
vector<int> minp, primes;
vector<int> phi, mu;

void sieve(int n) {
    minp.assign(n + 1, 0);
    phi.assign(n + 1, 0);
    mu.assign(n + 1, 0);
    primes.clear();

    phi[1] = 1;
    mu[1] = 1;

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            primes.push_back(i);
            phi[i] = i - 1;
            mu[i] = -1;
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                phi[i * p] = phi[i] * p;
                mu[i * p] = 0;
                break;
            } else {
                phi[i * p] = phi[i] * (p - 1);
                mu[i * p] = -mu[i];
            }
        }
    }
}

bool isprime(int n) {
    return minp[n] == n;
}
```

2.5 Pollard's Rho

```cpp
std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().count());

i64 factor(i64 n) {
    if (n % 2 == 0) {
        return 2;
    }
    if (isPrime(n)) {
        return n;
    }
    i64 m = 2;
    while (true) {
        i64 c = (rng() % (n - 1)) + 1;
        auto f = [&](i64 x) { return (mul(x, x, n) + c) % n; };
        i64 d = 1, x = m, y = m, p = 1, q = 0, v = 1;
        while (d == 1) {
            y = f(y);
            q++;
            v = mul(v, std::abs(x - y), n);
            if (q % 127 == 0) {
                d = std::gcd(v, n);
                v = 1;
            }
            if (p == q) {
                x = y;
                p *= 2;
                q = 0;
                d = std::gcd(v, n);
                v = 1;
            }
        }
        if (d != n) {
            return d;
        }
        m++;
    }
}

std::vector<i64> factorize(i64 n) {
    std::vector<i64> p;
    auto dfs = [&](auto &&self, i64 n) -> void {
        if (isPrime(n)) {
            p.push_back(n);
            return;
        }
        i64 d = factor(n);
        self(self, d);
        self(self, n / d);
    };
    dfs(dfs, n);
    std::sort(p.begin(), p.end());
    return p;
}
```

2.6 Combination

```cpp
{
    vector<int> fac(n + 1), invfac(n + 1);
    fac[0] = 1;
    for (int i = 1; i <= n; i++) {
        fac[i] = mul(fac[i - 1], i);
    }
    invfac[n] = power(fac[n], P - 2);
    for (int i = n; i >= 1; i--) {
        invfac[i - 1] = mul(invfac[i], i);
    }
    auto binom = [&](int n, int m) -> int {
        if (n < m or m < 0) {
            return 0;
        }
        return i64(fac[n]) * invfac[m] % P * invfac[n - m] % P;
    };
}
```

2.7 Linear Basis

```cpp
// a : base
// k : dimension
{
    auto insert = [&](int x) {
        for (int d = k - 1; d >= 0 and x != 0; d--) {
            if (x >> d & 1) {
                if (a[d] == 0) {
                    a[d] = x;
                }
                x ^= a[d];
            }
        }
    };
    // ...
    int ans = 0;
    for (int d = k - 1; d >= 0; d--) {
        ans = max(ans, ans ^ a[d]);
    }
}
// t : time stamp
{
    auto insert = [&](int x, int i) {
        for (int d = k - 1; d >= 0; d--) {
            if (x >> d & 1) {
                if (i > t[d]) {
                    swap(i, t[d]);
                    swap(x, a[d]);
                }
                x ^= a[d];
            }
        }
    };
    auto query = [&](int i) {
        int res = 0;
        for (int d = k - 1; d >= 0; d--) {
            if (t[d] >= i) {
                res = max(res, res ^ a[d]);
            }
        }
        return res;
    };
}
```

2.8 Gaussian Elimination

```cpp
std::vector<int> operator*(const std::vector<int> &lhs, const std::vector<int
  > &rhs) {
    std::vector<int> res(lhs.size() + rhs.size() - 1);
    for (int i = 0; i < int(lhs.size()); ++i)
        for (int j = 0; j < int(rhs.size()); ++j)
            res[i + j] = (res[i + j] + 1ll * lhs[i] * rhs[j]) % P;
    return res;
}
std::vector<int> operator%(const std::vector<int> &lhs, const std::vector<int> &rhs) {
    auto res = lhs;
    int m = rhs.size() - 1;
    int inv = power(rhs.back(), P - 2);
    for (int i = res.size() - 1; i >= m; --i) {
        int x = 1ll * inv * res[i] % P;
        for (int j = 0; j < m; ++j)
            res[i - m + j] = (res[i - m + j] + 1ll * (P - x) * rhs[j]) % P;
    }
    if (int(res.size()) > m)
        res.resize(m);
    return res;
}
std::vector<int> gauss(std::vector<std::vector<int>> a, std::vector<int> b) {
    int n = a.size();
    for (int i = 0; i < n; ++i) {
        int r = i;
        while (a[r][i] == 0)
            ++r;
        std::swap(a[i], a[r]);
        std::swap(b[i], b[r]);
        int inv = power(a[i][i], P - 2);
        for (int j = i; j < n; ++j)
            a[i][j] = 1ll * a[i][j] * inv % P;
        b[i] = 1ll * b[i] * inv % P;
        for (int j = 0; j < n; ++j) {
            if (i == j)
                continue;
            int x = a[j][i];
            for (int k = i; k < n; ++k)
                a[j][k] = (a[j][k] + 1ll * (P - x) * a[i][k]) % P;
            b[j] = (b[j] + 1ll * (P - x) * b[i]) % P;
        }
    }
    return b;
}


std::vector<double> gauss(std::vector<std::vector<double>> a, std::vector<double> b) {
    int n = a.size();
    for (int i = 0; i < n; ++i) {
        double x = a[i][i];
        for (int j = i; j < n; ++j) a[i][j] /= x;
        b[i] /= x;
        for (int j = 0; j < n; ++j) {
            if (i == j) continue;
            x = a[j][i];
            for (int k = i; k < n; ++k) a[j][k] -= a[i][k] * x;
            b[j] -= b[i] * x;
        }
    }
    return b;
}
```

2.9 Polynomial

2.9.1 Lagrange Interpolation

```
// n points -> n - 1 polynomial     O(n ^ 2)
for (int i = 0; i < n; i++) {
    Z num = y[i];
    Z den = 1;
    for (int j = 0; j < n; j++) {
        if (i == j) {
            continue;
        }
        num *= (k - x[j]); // f(k)
        den *= (x[i] - x[j]);
    }
    ans += num / den;
}

// Continuous x     O(n)
vector<Z> fac(n + 1);
fac[0] = 1;
for (int i = 1; i <= n; i++) {
    fac[i] = fac[i - 1] * i;
}

vector<Z> pre(n + 1);
pre[0] = 1;
for (int i = 0; i < n; i++) {
    pre[i + 1] = pre[i] * (k - i);
}

vector<Z> suf(n + 1);
suf[n] = 1;
for (int i = n - 1; i >= 0; i--) {
    suf[i] = suf[i + 1] * (k - i);
}

Z ans = 0;
for (int i = 0; i < n; i++) {
    Z res = y[i];
```

```cpp
    res *= pre[i] * suf[i + 1];
    res /= ((n - 1 - i) % 2 ? -1 : 1) * fac[i] * fac[n - 1 - i];

    ans += res;
}
```

2.9.2 Number Theory Transform

```cpp
constexpr int P = 998244353;

int power(int a, int b) {
    int res = 1;
    for (; b; b /= 2, a = 1LL * a * a % P) {
        if (b % 2) {
            res = 1LL * res * a % P;
        }
    }
    return res;
}

std::vector<int> rev, roots {0, 1};

void dft(std::vector<int> &a) {
    int n = a.size();
    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }
    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }
    if (roots.size() < n) {
        int k = __builtin_ctz(roots.size());
        roots.resize(n);
        while ((1 << k) < n) {
            int e = power(31, 1 << (__builtin_ctz(P - 1) - k - 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots[2 * i] = roots[i];
                roots[2 * i + 1] = 1LL * roots[i] * e % P;
            }
            k++;
        }
    }

    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                int u = a[i + j];
                int v = 1LL * a[i + j + k] * roots[k + j] % P;
                a[i + j] = (u + v) % P;
                a[i + j + k] = (u - v) % P;
            }
        }
    }
}

void idft(std::vector<int> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    int inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] = 1LL * a[i] * inv % P;
    }
}

std::vector<int> mul(std::vector<int> a, std::vector<int> b) {
    int n = 1, tot = a.size() + b.size() - 1;
    while (n < tot) {
        n *= 2;
    }
    a.resize(n);
    b.resize(n);
    dft(a);
    dft(b);
    for (int i = 0; i < n; i++) {
        a[i] = 1LL * a[i] * b[i] % P;
    }
    idft(a);
    a.resize(tot);
    return a;
}
```

```cpp
// with ModIntBase
std::vector<int> rev;
template <u32 P>
std::vector<ModInt<P>> roots{0, 1};

template <u32 P>
constexpr ModInt<P> findPrimitiveRoot() {
    ModInt<P> i = 2;
    int k = __builtin_ctz((int)P - 1);
    while (true) {
        if (power(i, (P - 1) / 2) != 1) {
            break;
        }
        i += 1;
    }
    return power(i, (P - 1) >> k);
}

template <u32 P>
constexpr ModInt<P> primitiveRoot = findPrimitiveRoot<P>();

template <>
constexpr ModInt<998244353> primitiveRoot<998244353>{31};

template <u32 P>
constexpr void dft(std::vector<ModInt<P>> &a) {
    int n = a.size();

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

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }

    if (roots<P>.size() < n) {
        int k = __builtin_ctz(roots<P>.size());
        roots<P>.resize(n);
        while ((1 << k) < n) {
            auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots<P>[2 * i] = roots<P>[i];
                roots<P>[2 * i + 1] = roots<P>[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                ModInt<P> u = a[i + j];
                ModInt<P> v = a[i + j + k] * roots<P>[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}

template <u32 P>
constexpr void idft(std::vector<ModInt<P>> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    ModInt<P> inv = (1 - (int)P) / n;
    for (int i = 0; i < n; i++) {
        a[i] *= inv;
    }
}

template <int P = 998244353>
struct Poly : public std::vector<ModInt<P>> {
    using V = ModInt<P>;

    Poly() : std::vector<V>() {}
    explicit constexpr Poly(int n) : std::vector<V>(n) {}

    explicit constexpr Poly(const std::vector<V> &a) : std::vector<V>(a) {}
    constexpr Poly(const std::initializer_list<V> &a) : std::vector<V>(a) {}

    template <class InputIt, class = std::_RequireInputIter<InputIt>>
    explicit constexpr Poly(InputIt first, InputIt last) : std::vector<V>(first, last) {}

    template <class F>
    explicit constexpr Poly(int n, F f) : std::vector<V>(n) {
        for (int i = 0; i < n; i++) {
            (*this)[i] = f(i);
        }
    }

    constexpr Poly shift(int k) const {
        if (k >= 0) {
            auto b = *this;
            b.insert(b.begin(), k, 0);
            return b;
        } else if (this->size() <= -k) {
            return Poly();
        } else {
            return Poly(this->begin() + (-k), this->end());
        }
    }
    constexpr Poly trunc(int k) const {
        Poly f = *this;
        f.resize(k);
        return f;
    }
    constexpr friend Poly operator+(const Poly &a, const Poly &b) {
        Poly res(std::max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i++) {
            res[i] += a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            res[i] += b[i];
        }
        return res;
    }
    constexpr friend Poly operator-(const Poly &a, const Poly &b) {
        Poly res(std::max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i++) {
            res[i] += a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            res[i] -= b[i];
        }
        return res;
    }
    constexpr friend Poly operator-(const Poly &a) {
        std::vector<V> res(a.size());
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = -a[i];
        }
        return Poly(res);
    }
    constexpr friend Poly operator*(Poly a, Poly b) {
        if (a.size() == 0 || b.size() == 0) {
            return Poly();
        }
        if (a.size() < b.size()) {
            std::swap(a, b);
        }
        int n = 1, tot = a.size() + b.size() - 1;
        while (n < tot) {
            n *= 2;
        }
        if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
            Poly c(a.size() + b.size() - 1);
            for (int i = 0; i < a.size(); i++) {
                for (int j = 0; j < b.size(); j++) {
                    c[i + j] += a[i] * b[j];
                }
            }
            return c;
        }
        a.resize(n);
        b.resize(n);
        dft<P>(a);
        dft<P>(b);
        for (int i = 0; i < n; ++i) {
            a[i] *= b[i];
        }
        idft<P>(a);
        a.resize(tot);
        return a;
    }
    constexpr friend Poly operator*(V a, Poly b) {
        for (int i = 0; i < int(b.size()); i++) {
            b[i] *= a;
        }
        return b;
    }
    constexpr friend Poly operator*(Poly a, V b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] *= b;
        }
        return a;
    }
    constexpr friend Poly operator/(Poly a, V b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] /= b;
        }
        return a;
    }
    constexpr Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    constexpr Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    constexpr Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    constexpr Poly &operator*=(V b) {
        return (*this) = (*this) * b;
    }
    constexpr Poly &operator/=(V b) {
        return (*this) = (*this) / b;
    }
    constexpr Poly deriv() const {
        if (this->empty()) {
            return Poly();
        }
        Poly res((int)this->size() - 1);
        for (int i = 0; i < this->size() - 1; ++i) {
            res[i] = (i + 1) * (*this)[i + 1];
        }
        return res;
    }
    constexpr Poly integr() const {
        Poly res(this->size() + 1);
        for (int i = 0; i < this->size(); ++i) {
            res[i + 1] = (*this)[i] / (i + 1);
        }
        return res;
    }
    constexpr Poly inv(int m) const {
        Poly x{(*this)[0].inv()};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
        }
        return x.trunc(m);
    }
    constexpr Poly log(int m) const {
        return (deriv() * inv(m)).integr().trunc(m);
    }
    constexpr Poly exp(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
        }
        return x.trunc(m);
    }
    constexpr Poly pow(int k, int m) const {
        int i = 0;
        while (i < this->size() && (*this)[i] == 0) {
            i++;
        }
        if (i == this->size() || 1LL * i * k >= m) {
            return Poly(m);
        }
        V v = (*this)[i];
        auto f = shift(-i) * v.inv();
        return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v,
          k);
    }
    constexpr Poly sqrt(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
        }
        return x.trunc(m);
    }
    constexpr Poly mulT(Poly b) const {
        if (b.size() == 0) {
            return Poly();
        }
        int n = b.size();
        std::reverse(b.begin(), b.end());
        return ((*this) * b).shift(-(n - 1));
    }
    constexpr std::vector<V> eval(std::vector<V> x) const {
        if (this->size() == 0) {
            return std::vector<V>(x.size(), 0);
        }
        const int n = std::max(x.size(), this->size());
        std::vector<Poly> q(4 * n);
        std::vector<V> ans(x.size());
        x.resize(n);
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = Poly{1, -x[l]};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
        build(1, 0, n);
        std::function<void(int, int, int, const Poly &)> work = [&](int p,
          int l, int r, const Poly &num) {
            if (r - l == 1) {
                if (l < int(ans.size())) {
                    ans[l] = num[0];
                }
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).trunc(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).trunc(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};

template <int P = 998244353>
Poly<P> berlekampMassey(const Poly<P> &s) {
    Poly<P> c;
    Poly<P> oldC;
    int f = -1;
    for (int i = 0; i < s.size(); i++) {
        auto delta = s[i];
        for (int j = 1; j <= c.size(); j++) {
            delta -= c[j - 1] * s[i - j];
        }
        if (delta == 0) {
            continue;
        }
        if (f == -1) {
            c.resize(i + 1);
            f = i;
        } else {
            auto d = oldC;
            d *= -1;
            d.insert(d.begin(), 1);
            ModInt<P> df1 = 0;
            for (int j = 1; j <= d.size(); j++) {
                df1 += d[j - 1] * s[f + 1 - j];
            }
            assert(df1 != 0);
            auto coef = delta / df1;
            d *= coef;
            Poly<P> zeros(i - f - 1);
            zeros.insert(zeros.end(), d.begin(), d.end());
            d = zeros;
            auto temp = c;
            c += d;
            if (i - temp.size() > f - oldC.size()) {
                oldC = temp;
                f = i;
            }
        }
    }
    c *= -1;
    c.insert(c.begin(), 1);
    return c;
}

template <int P = 998244353>
ModInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
    int m = q.size() - 1;
    while (n > 0) {
        auto newq = q;
        for (int i = 1; i <= m; i += 2) {
            newq[i] *= -1;
        }
        auto newp = p * newq;
        newq = q * newq;
        for (int i = 0; i < m; i++) {
            p[i] = newp[i * 2 + n % 2];
        }
        for (int i = 0; i <= m; i++) {
            q[i] = newq[i * 2];
        }
        n /= 2;
    }
    return p[0] / q[0];
}

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

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

    void init(int m) {
        m = std::min(m, (int)Z::mod() - 1);
        if (m <= n)
            return;
        _fac.resize(m + 1);
        _invfac.resize(m + 1);
        _inv.resize(m + 1);

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

    Z fac(int m) {
        if (m > n)
            init(2 * m);
        return _fac[m];
    }
    Z invfac(int m) {
        if (m > n)
            init(2 * m);
        return _invfac[m];
    }
    Z inv(int m) {
        if (m > n)
            init(2 * m);
        return _inv[m];
    }
    Z binom(int n, int m) {
        if (n < m || m < 0)
```

```
            return 0;
        return fac(n) * invfac(m) * invfac(n - m);
    }
} comb;

template <int P = 998244353>
Poly<P> get(int n, int m) {
    if (m == 0) {
        return Poly(n + 1);
    }
    if (m % 2 == 1) {
        auto f = get(n, m - 1);
        Z p = 1;
        for (int i = 0; i <= n; i++) {
            f[n - i] += comb.binom(n, i) * p;
            p *= m;
        }
        return f;
    }
    auto f = get(n, m / 2);
    auto fm = f;
    for (int i = 0; i <= n; i++) {
        fm[i] *= comb.fac(i);
    }
    Poly pw(n + 1);
    pw[0] = 1;
    for (int i = 1; i <= n; i++) {
        pw[i] = pw[i - 1] * (m / 2);
    }
    for (int i = 0; i <= n; i++) {
        pw[i] *= comb.invfac(i);
    }
    fm = fm.mulT(pw);
    for (int i = 0; i <= n; i++) {
        fm[i] *= comb.invfac(i);
    }
    return f + fm;
}
```

2.9.3 Fast Walsh Transform

```
void andFwt(auto &a, int t) {
    int n = a.size();
    for (int i = 1; i < n; i *= 2) {
        for (int s = 0; s < n; s++) {
            if (~s & i) {
                a[s] += t * a[s | i];
            }
        }
    }
}
void orFwt(auto &a, int t) {
    int n = a.size();
    for (int i = 1; i < n; i *= 2) {
        for (int s = 0; s < n; s++) {
            if (~s & i) {
                a[s | i] += t * a[s];
            }
        }
    }
}
void xorFwt(auto &a) {
    int n = a.size();
    for (int i = 1; i < n; i *= 2) {
        for (int j = 0; j < n; j += 2 * i) {
            for (int k = 0; k < i; k++) {
                auto u = a[j + k], v = a[i + j + k];
                a[j + k] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}
auto xorConv(auto a, auto b) {
    int n = a.size();
    xorFwt(a);
    xorFwt(b);

    for (int i = 0; i < n; i++) {
        a[i] *= b[i];
    }
    xorFwt(a);
    auto invn = Z(n).inv();
    for (int i = 0; i < n; i++) {
        a[i] *= invn;
    }
    // if not module:
    // for (int i = 0; i < n; i++) {
    //     a[i] >>= __lg(n);
    // }
    return move(a);
}
```

2.9.4 Sum of Subset

```
{
    std::vector<int> f(1 << n);
    // ...

    // prefix
```

```cpp
    for (int i = 0; i < n; i++) {
        for (int s = 0; s < 1 << n; s++) {
            if (s >> i & 1) {
                f[s] += f[s ^ (1 << i)];
            }
        }
    }
    // suffix
    for (int i = 0; i < n; i++) {
        for (int s = 0; s < 1 << n; s++) {
            if (~s >> i & 1) {
                f[s] += f[s ^ (1 << i)];
            }
        }
    }
}
```

3 Graph

3.1 Tree

3.1.1 Lowest Common Ancestor

```cpp
// Binary Lifting O(n log n) - O(log n)
{
    const int logn = std::__lg(n);

    std::vector<int> dep(n);
    std::vector p(logn + 1, std::vector<int>(n));
    auto dfs = [&](auto &&self, int x) -> void {
        for (auto y : adj[x]) {
            if (y == p[0][x]) {
                continue;
            }
            p[0][y] = x;
            dep[y] = dep[x] + 1;
            self(self, y);
        }
    };

    p[0][s] = s;
    dfs(dfs, s);

    for (int j = 0; j < logn; j++) {
        for (int i = 0; i < n; i++) {
            p[j + 1][i] = p[j][p[j][i]];
        }
    }

    auto lca = [&](int x, int y) {
        if (dep[x] < dep[y]) {
            std::swap(x, y);
        }
        while (dep[x] > dep[y]) {
            x = p[std::__lg(dep[x] - dep[y])][x];
        }
        if (x == y) {
            return x;
        }
        for (int i = std::__lg(dep[x]); i >= 0; i--) {
            if (p[i][x] != p[i][y]) {
                x = p[i][x];
                y = p[i][y];
            }
        }
        return p[0][x];
    };
}
```

3.1.2 Heavy-Light Decomposition

```cpp
struct HLD {
    int n;
    std::vector<int> siz, top, dep, parent, in, out, seq;
    std::vector<std::vector<int>> adj;
    int cur;

    HLD() {}
    HLD(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        siz.resize(n);
        top.resize(n);
        dep.resize(n);
        parent.resize(n);
        in.resize(n);
        out.resize(n);
        seq.resize(n);
        cur = 0;
        adj.assign(n, {});
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void work(int root = 0) {
        top[root] = root;
        dep[root] = 0;
        parent[root] = -1;
        dfs1(root);
        dfs2(root);
    }
    void dfs1(int u) {
        if (parent[u] != -1) {
            adj[u].erase(find(adj[u].begin(), adj[u].end(), parent[u]));
        }

        siz[u] = 1;
        for (auto &v : adj[u]) {
            parent[v] = u;
            dep[v] = dep[u] + 1;
            dfs1(v);
            siz[u] += siz[v];
            if (siz[v] > siz[adj[u][0]]) {
                std::swap(v, adj[u][0]);
            }
        }
    }
    void dfs2(int u) {
        in[u] = cur++;
        seq[in[u]] = u;
        for (auto v : adj[u]) {
            top[v] = v == adj[u][0] ? top[u] : v;
            dfs2(v);
        }
        out[u] = cur;
    }
    int lca(int u, int v) {
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                u = parent[top[u]];
            } else {
                v = parent[top[v]];
            }
        }
        return dep[u] < dep[v] ? u : v;
    }
    int dist(int u, int v) {
        return dep[u] + dep[v] - 2 * dep[lca(u, v)];
    }
    bool isAncester(int fa, int son) {
        return in[fa] <= in[son] && in[son] < out[fa];
    }
    // [u, v]
    auto getPath(int u, int v) {
        std::vector<std::pair<int, int>> ret;
        while (top[u] != top[v]) {
            if (dep[top[u]] > dep[top[v]]) {
                ret.push_back({in[top[u]], in[u]});
                u = parent[top[u]];
            } else {
                ret.push_back({in[top[v]], in[v]});
                v = parent[top[v]];
            }
        }
        if (dep[u] > dep[v]) {
            ret.push_back({in[v], in[u]});
        } else {
            ret.push_back({in[u], in[v]});
        }
        return std::move(ret);
    }
    // [u, v)
    std::pair<int, int> getTree(int u) {
        return pair(in[u], out[u]);
    }
    int jump(int u, int k) {
        if (dep[u] < k) {
            return -1;
        }
        int d = dep[u] - k;
        while (dep[top[u]] > d) {
            u = parent[top[u]];
        }
        return seq[in[u] - dep[u] + d];
    }
    int rootedParent(int u, int v) {
        std::swap(u, v);
        if (u == v) {
            return u;
        }
        if (!isAncester(u, v)) {
            return parent[u];
        }
        auto it = std::upper_bound(adj[u].begin(), adj[u].end(), v, [&](int x, int y) {
            return in[x] < in[y];
        }) - 1;
        return *it;
    }
    int rootedSize(int u, int v) {
        if (u == v) {
            return n;
        }
        if (!isAncester(v, u)) {
            return siz[v];
        }
        return n - siz[rootedParent(u, v)];
    }
    int rootedLca(int a, int b, int c) {
        return lca(a, b) ^ lca(b, c) ^ lca(c, a);
    }
};
```

3.1.3 DSU on Tree

```
{
    int n;
    std::vector<std::vector<int>> adj(n);
    // ...
    std::vector<int> siz(n);
    [&](this auto &&self, int x, int p) -> void {
        if (p != -1) {
            adj[x].erase(find(adj[x].begin(), adj[x].end(), p));
        }
        siz[x] = 1;
        // &y
        for (auto &y : adj[x]) {
            self(y, x);
            siz[x] += siz[y];
            if (siz[y] > siz[adj[x][0]]) {
                swap(adj[x][0], y);
            }
        }
    } (0, -1);

    auto addv = [&](int color, int t) {
        // freq[color] += t
    };
    auto add = [&](auto &&self, int x, int t) -> void {
        // addv(color[x], t);
        for (auto y : adj[x]) {
            self(self, y, t);
        }
    };
    auto dfs = [&](auto &&self, int x) -> void {
        for (auto y : adj[x]) {
            if (y != adj[x][0]) {
                self(self, y);
                add(add, y, -1);
            }
        }
        if (not adj[x].empty()) {
            self(self, adj[x][0]);
            for (auto y : adj[x]) {
                if (y != adj[x][0]) {
                    add(add, y, 1);
                }
            }
        }
        addv(color[x], 1);
        // ans[x] = ?
    };
    // dfs(0);
}
```

3.2 Connectivity

3.2.1 Strongly Connected Component

```
struct SCC {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> stk;
    std::vector<int> dfn, low, bel;
    int cur, cnt;

    SCC() {}
    SCC(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        bel.assign(n, -1);
        stk.clear();
        cur = cnt = 0;
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
    }
    void dfs(int x) {
        dfn[x] = low[x] = cur++;
        stk.push_back(x);
        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = std::min(low[x], low[y]);
            } else if (bel[y] == -1) {
                low[x] = std::min(low[x], dfn[y]);
            }
        }
        if (dfn[x] == low[x]) {
            int y;
            do {
                y = stk.back();
                bel[y] = cnt;
                stk.pop_back();
```

```cpp
            } while (y != x);
            cnt++;
        }
    }
    std::vector<int> work() {
        for (int i = 0; i < n; i++) {
            if (dfn[i] == -1) {
                dfs(i);
            }
        }
        return bel;
    }
};
```

3.2.2 Block Cut Tree

```cpp
struct BlockCutTree {
    int n;
    std::vector<std::vector<int>> adj;
    std::vector<int> dfn, low, stk;
    int cnt, cur;
    std::vector<std::pair<int, int>> edges;
    BlockCutTree() {}
    BlockCutTree(int n) {
        init(n);
    }
    void init(int n) {
        this->n = n;
        adj.assign(n, {});
        dfn.assign(n, -1);
        low.resize(n);
        stk.clear();
        cnt = cur = 0;
        edges.clear();
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    void dfs(int x) {
        stk.push_back(x);
        dfn[x] = low[x] = cur++;
        for (auto y : adj[x]) {
            if (dfn[y] == -1) {
                dfs(y);
                low[x] = std::min(low[x], low[y]);
                if (low[y] == dfn[x]) {
                    int v;
                    do {
                        v = stk.back();
                        stk.pop_back();
                        edges.emplace_back(n + cnt, v);
                    } while (v != y);
                    edges.emplace_back(x, n + cnt);
                    cnt++;
                }
            } else {
                low[x] = std::min(low[x], dfn[y]);
            }
        }
    }
    std::pair<int, std::vector<std::pair<int, int>>> work() {
        for (int i = 0; i < n; i++) {
            if (dfn[i] == -1) {
                stk.clear();
                dfs(i);
            }
        }
        return {cnt, edges};
    }
};
```

3.3 Two Sat

```cpp
struct TwoSat {
    int n;
    std::vector<std::vector<int>> e;
    std::vector<bool> ans;
    TwoSat(int n) : n(n), e(2 * n), ans(n) {}
    void addClause(int u, bool f, int v, bool g) {
        e[2 * u + !f].push_back(2 * v + g);
        e[2 * v + !g].push_back(2 * u + f);
    }
    bool satisfiable() {
        std::vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
        std::vector<int> stk;
        int now = 0, cnt = 0;
        std::function<void(int)> tarjan = [&](int u) {
            stk.push_back(u);
            dfn[u] = low[u] = now++;
            for (auto v : e[u]) {
                if (dfn[v] == -1) {
                    tarjan(v);
                    low[u] = std::min(low[u], low[v]);
                } else if (id[v] == -1) {
                    low[u] = std::min(low[u], dfn[v]);
                }
            }
            if (dfn[u] == low[u]) {
                int v;
                do {
                    v = stk.back();
                    stk.pop_back();
                    id[v] = cnt;
                } while (v != u);
                ++cnt;
            }
        };
        for (int i = 0; i < 2 * n; ++i) if (dfn[i] == -1) tarjan(i);
        for (int i = 0; i < n; ++i) {
            if (id[2 * i] == id[2 * i + 1]) return false;
            ans[i] = id[2 * i] > id[2 * i + 1];
        }
        return true;
    }
    std::vector<bool> answer() { return ans; }
};
```

4 String

4.1 String Hash

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

using i64 = long long;
using Hash = array<int, 2>;

constexpr int P = 998244353;
constexpr Hash base = {567, 1234}; // any

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    string s;
    cin >> s;

    const int n = s.size();

    vector<Hash> h(n + 1), p(n + 1);
    p[0] = {1, 1};
    for (int j = 0; j < 2; j++) {
        for (int i = 0; i < n; i++) {
            h[i + 1][j] = (i64(base[j]) * h[i][j] + s[i]) % P;
            p[i + 1][j] = (i64(base[j]) * p[i][j]) % P;
        }
    }

    auto get = [&](int l, int r) {
        Hash res{};
        for (int i = 0; i < 2; i++) {
            res[i] = (h[r][i] + 1ll * (P - h[l][i]) * p[r - l][i]) % P;
        }
        return res;
    };

    return 0;
}
```

4.2 KMP

```
std::vector<int> kmp(std::string s) {
    int n = s.size();
    std::vector<int> f(n + 1);
    for (int i = 1, j = 0; i < n; i++) {
        while (j && s[i] != s[j]) {
            j = f[j];
        }
        j += (s[i] == s[j]);
        f[i + 1] = j;
    }
    return f;
}
```

4.3 Z-function

```
std::vector<int> Z(std::string s) {
    int n = s.size();
    std::vector<int> z(n + 1);
    z[0] = n;
    for (int i = 1, j = 1; i < n; i++) {
        z[i] = std::max(0, std::min(j + z[j] - i, z[i - j]));
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if (i + z[i] > j + z[j]) {
            j = i;
        }
    }
    return z;
}
```

4.4 AhoCorasick

```
struct AhoCorasick {
    static constexpr int ALPHABET = 26;
    struct Node {
        int len;
        int link;
        std::array<int, ALPHABET> next;
        Node() : len{0}, link{0}, next{} {}
    };

    std::vector<Node> t;

    AhoCorasick() {
        init();
    }

    void init() {
        t.assign(2, Node());
        t[0].next.fill(1);
        t[0].len = -1;
    }

    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }

    int add(const std::string &a) {
        int p = 1;
        for (auto c : a) {
            int x = c - 'a';
            if (t[p].next[x] == 0) {
                t[p].next[x] = newNode();
                t[t[p].next[x]].len = t[p].len + 1;
            }
            p = t[p].next[x];
        }
        return p;
    }

    void work() {
        std::queue<int> q;
        q.push(1);

        while (!q.empty()) {
            int x = q.front();
            q.pop();

            for (int i = 0; i < ALPHABET; i++) {
                if (t[x].next[i] == 0) {
                    t[x].next[i] = t[t[x].link].next[i];
                } else {
                    t[t[x].next[i]].link = t[t[x].link].next[i];
                    q.push(t[x].next[i]);
                }
            }
        }
    }

    int next(int p, int x) {
        return t[p].next[x];
    }

    int link(int p) {
        return t[p].link;
    }

    int len(int p) {
        return t[p].len;
    }

    int size() {
        return t.size();
    }
};
```

4.5 Suffix Array

```
struct SuffixArray {
    int n;
    std::vector<int> sa, rk, lc;
    SuffixArray(const std::string &s) {
        n = s.length();
        sa.resize(n);
        lc.resize(n - 1);
        rk.resize(n);
        std::iota(sa.begin(), sa.end(), 0);
        std::sort(sa.begin(), sa.end(),
            [&](int a, int b) {
                return s[a] < s[b];
            });
        rk[sa[0]] = 0;
        for (int i = 1; i < n; i++) {
            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
        }
        int k = 1;
        std::vector<int> tmp, cnt(n);
        tmp.reserve(n);
        while (rk[sa[n - 1]] < n - 1) {
            tmp.clear();
            for (int i = 0; i < k; i++) {
                tmp.push_back(n - k + i);
            }
```

```cpp
            for (auto i : sa) {
                if (i >= k) {
                    tmp.push_back(i - k);
                }
            }
            std::fill(cnt.begin(), cnt.end(), 0);
            for (int i = 0; i < n; i++) {
                cnt[rk[i]]++;
            }
            for (int i = 1; i < n; i++) {
                cnt[i] += cnt[i - 1];
            }
            for (int i = n - 1; i >= 0; i--) {
                sa[--cnt[rk[tmp[i]]]] = tmp[i];
            }
            std::swap(rk, tmp);
            rk[sa[0]] = 0;
            for (int i = 1; i < n; i++) {
                rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] || sa[i - 1] + k == n || tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
            }
            k *= 2;
        }
        for (int i = 0, j = 0; i < n; i++) {
            if (rk[i] == 0) {
                j = 0;
            } else {
                for (j -= (j > 0); i + j < n && sa[rk[i] - 1] + j < n && s[i + j] == s[sa[rk[i] - 1] + j]; j++)
                    ;
                lc[rk[i] - 1] = j;
            }
        }
    }
};
```

4.6 Suffix Automaton

```cpp
struct SAM {
    static constexpr int ALPHABET_SIZE = 26;
    struct Node {
        int len;
        int link;
        std::array<int, ALPHABET_SIZE> next;
        Node() : len{}, link{}, next{} {}
    };
    std::vector<Node> t;
    SAM() {
        init();
    }
    void init() {
        t.assign(2, Node());
        t[0].next.fill(1);
        t[0].len = -1;
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
    int extend(int p, int c) {
        if (t[p].next[c]) {
            int q = t[p].next[c];
            if (t[q].len == t[p].len + 1) {
                return q;
            }
            int r = newNode();
            t[r].len = t[p].len + 1;
            t[r].link = t[q].link;
            t[r].next = t[q].next;
            t[q].link = r;
            while (t[p].next[c] == q) {
                t[p].next[c] = r;
                p = t[p].link;
            }
            return r;
        }
        int cur = newNode();
        t[cur].len = t[p].len + 1;
        while (!t[p].next[c]) {
            t[p].next[c] = cur;
            p = t[p].link;
        }
        t[cur].link = extend(p, c);
        return cur;
    }
    int extend(int p, char c, char offset = 'a') {
        return extend(p, c - offset);
    }

    int next(int p, int x) {
        return t[p].next[x];
    }

    int next(int p, char c, char offset = 'a') {
        return next(p, c - 'a');
    }

    int link(int p) {
        return t[p].link;
    }

    int len(int p) {
        return t[p].len;
    }

    int size() {
        return t.size();
    }
};
```

4.7 Palindromic Tree

```cpp
struct PAM {
    static constexpr int ALPHABET_SIZE = 26;
    struct Node {
        int len;
        int link;
        int cnt;
        std::array<int, ALPHABET_SIZE> next;
        Node() : len{}, link{}, cnt{}, next{} {}
    };
    std::vector<Node> t;
    int suff;
    std::string s;
    PAM() {
        init();
    }
    void init() {
        t.assign(2, Node());
        t[0].len = -1;
        suff = 1;
        s.clear();
    }
    int newNode() {
        t.emplace_back();
        return t.size() - 1;
    }
    bool add(char c) {
        int pos = s.size();
        s += c;
        int let = c - 'a';
        int cur = suff, curlen = 0;
        while (true) {
            curlen = t[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
                break;
            }
            cur = t[cur].link;
        }
        if (t[cur].next[let]) {
            suff = t[cur].next[let];
            return false;
        }
        int num = newNode();
        suff = num;
        t[num].len = t[cur].len + 2;
        t[cur].next[let] = num;
        if (t[num].len == 1) {
            t[num].link = 1;
            t[num].cnt = 1;
            return true;
        }
        while (true) {
            cur = t[cur].link;
            curlen = t[cur].len;
            if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
                t[num].link = t[cur].next[let];
                break;
            }
        }
        t[num].cnt = 1 + t[t[num].link].cnt;
        return true;
    }
    int next(int p, int x) {
        return t[p].next[x];
    }
    int link(int p) {
        return t[p].link;
    }
    int len(int p) {
        return t[p].len;
    }
    int size() {
        return t.size();
    }
};
```

5 Geometry

```cpp
template<class T>
struct Point {
    T x;
    T y;
    Point(const T &x_ = 0, const T &y_ = 0) : x(x_), y(y_) {}

    template<class U>
    operator Point<U>() {
        return Point<U>(U(x), U(y));
    }
    Point &operator+=(const Point &p) & {
        x += p.x;
        y += p.y;
        return *this;
    }
    Point &operator-=(const Point &p) & {
        x -= p.x;
        y -= p.y;
        return *this;
    }
    Point &operator*=(const T &v) & {
        x *= v;
        y *= v;
        return *this;
    }
    Point &operator/=(const T &v) & {
        x /= v;
        y /= v;
        return *this;
    }
    Point operator-() const {
        return Point(-x, -y);
    }
    friend Point operator+(Point a, const Point &b) {
        return a += b;
    }
    friend Point operator-(Point a, const Point &b) {
        return a -= b;
    }
    friend Point operator*(Point a, const T &b) {
        return a *= b;
    }
    friend Point operator/(Point a, const T &b) {
        return a /= b;
    }
    friend Point operator*(const T &a, Point b) {
        return b *= a;
    }
    friend bool operator==(const Point &a, const Point &b) {
        return a.x == b.x && a.y == b.y;
    }
    friend std::istream &operator>>(std::istream &is, Point &p) {
        return is >> p.x >> p.y;
    }
    friend std::ostream &operator<<(std::ostream &os, const Point &p) {
        return os << "(" << p.x << ", " << p.y << ")";
    }
};

template<class T>
struct Line {
    Point<T> a;
    Point<T> b;
    Line(const Point<T> &a_ = Point<T>(), const Point<T> &b_ = Point<T>()) :
      a(a_), b(b_) {}
};

template<class T>
T dot(const Point<T> &a, const Point<T> &b) {
    return a.x * b.x + a.y * b.y;
}

template<class T>
T cross(const Point<T> &a, const Point<T> &b) {
    return a.x * b.y - a.y * b.x;
}

template<class T>
T square(const Point<T> &p) {
    return dot(p, p);
}

template<class T>
double length(const Point<T> &p) {
    return std::sqrt(square(p));
}

template<class T>
double length(const Line<T> &l) {
    return length(l.a - l.b);
}

template<class T>
Point<T> normalize(const Point<T> &p) {
    return p / length(p);
}

template<class T>
bool parallel(const Line<T> &l1, const Line<T> &l2) {
    return cross(l1.b - l1.a, l2.b - l2.a) == 0;
}

template<class T>
double distance(const Point<T> &a, const Point<T> &b) {
    return length(a - b);
}

template<class T>
double distancePL(const Point<T> &p, const Line<T> &l) {
    return std::abs(cross(l.a - l.b, l.a - p)) / length(l);
}

template<class T>
```

```cpp
double distancePS(const Point<T> &p, const Line<T> &l) {
    if (dot(p - l.a, l.b - l.a) < 0) {
        return distance(p, l.a);
    }
    if (dot(p - l.b, l.a - l.b) < 0) {
        return distance(p, l.b);
    }
    return distancePL(p, l);
}

template<class T>
Point<T> rotate(const Point<T> &a) {
    return Point(-a.y, a.x);
}

template<class T>
int sgn(const Point<T> &a) {
    return a.y > 0 || (a.y == 0 && a.x > 0) ? 1 : -1;
}

template<class T>
bool pointOnLineLeft(const Point<T> &p, const Line<T> &l) {
    return cross(l.b - l.a, p - l.a) > 0;
}

template<class T>
Point<T> lineIntersection(const Line<T> &l1, const Line<T> &l2) {
    return l1.a + (l1.b - l1.a) * (cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1.a - l1.b));
}

template<class T>
bool pointOnSegment(const Point<T> &p, const Line<T> &l) {
    return cross(p - l.a, l.b - l.a) == 0 && std::min(l.a.x, l.b.x) <= p.x && p.x <= std::max(l.a.x, l.b.x)
        && std::min(l.a.y, l.b.y) <= p.y && p.y <= std::max(l.a.y, l.b.y);
}

template<class T>
bool pointInPolygon(const Point<T> &a, const std::vector<Point<T>> &p) {
    int n = p.size();
    for (int i = 0; i < n; i++) {
        if (pointOnSegment(a, Line(p[i], p[(i + 1) % n]))) {
            return true;
        }
    }

    int t = 0;
    for (int i = 0; i < n; i++) {
        auto u = p[i];
        auto v = p[(i + 1) % n];
        if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line(v, u))) {
            t ^= 1;
        }
        if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line(u, v))) {
            t ^= 1;
        }
    }

    return t == 1;
}

// 0 : not intersect
// 1 : strictly intersect
// 2 : overlap
// 3 : intersect at endpoint
template<class T>
std::tuple<int, Point<T>, Point<T>> segmentIntersection(const Line<T> &l1, const Line<T> &l2) {
    if (std::max(l1.a.x, l1.b.x) < std::min(l2.a.x, l2.b.x)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (std::min(l1.a.x, l1.b.x) > std::max(l2.a.x, l2.b.x)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (std::max(l1.a.y, l1.b.y) < std::min(l2.a.y, l2.b.y)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (std::min(l1.a.y, l1.b.y) > std::max(l2.a.y, l2.b.y)) {
        return {0, Point<T>(), Point<T>()};
    }
    if (cross(l1.b - l1.a, l2.b - l2.a) == 0) {
        if (cross(l1.b - l1.a, l2.a - l1.a) != 0) {
            return {0, Point<T>(), Point<T>()};
        } else {
            auto maxx1 = std::max(l1.a.x, l1.b.x);
            auto minx1 = std::min(l1.a.x, l1.b.x);
            auto maxy1 = std::max(l1.a.y, l1.b.y);
            auto miny1 = std::min(l1.a.y, l1.b.y);
            auto maxx2 = std::max(l2.a.x, l2.b.x);
            auto minx2 = std::min(l2.a.x, l2.b.x);
            auto maxy2 = std::max(l2.a.y, l2.b.y);
            auto miny2 = std::min(l2.a.y, l2.b.y);
            Point<T> p1(std::max(minx1, minx2), std::max(miny1, miny2));
            Point<T> p2(std::min(maxx1, maxx2), std::min(maxy1, maxy2));
            if (!pointOnSegment(p1, l1)) {
                std::swap(p1.y, p2.y);
            }
            if (p1 == p2) {
                return {3, p1, p2};
            } else {
                return {2, p1, p2};
            }
        }
    }
    auto cp1 = cross(l2.a - l1.a, l2.b - l1.a);
    auto cp2 = cross(l2.a - l1.b, l2.b - l1.b);
    auto cp3 = cross(l1.a - l2.a, l1.b - l2.a);
    auto cp4 = cross(l1.a - l2.b, l1.b - l2.b);

    if ((cp1 > 0 && cp2 > 0) || (cp1 < 0 && cp2 < 0) || (cp3 > 0 && cp4 > 0)
      || (cp3 < 0 && cp4 < 0)) {
        return {0, Point<T>(), Point<T>()};
    }

    Point p = lineIntersection(l1, l2);
    if (cp1 != 0 && cp2 != 0 && cp3 != 0 && cp4 != 0) {
        return {1, p, p};
    } else {
        return {3, p, p};
    }
}

template<class T>
double distanceSS(const Line<T> &l1, const Line<T> &l2) {
    if (std::get<0>(segmentIntersection(l1, l2)) != 0) {
        return 0.0;
    }
    return std::min({distancePS(l1.a, l2), distancePS(l1.b, l2), distancePS(
      l2.a, l1), distancePS(l2.b, l1)});
}

template<class T>
bool segmentInPolygon(const Line<T> &l, const std::vector<Point<T>> &p) {
    int n = p.size();
    if (!pointInPolygon(l.a, p)) {
        return false;
    }
    if (!pointInPolygon(l.b, p)) {
        return false;
    }
    for (int i = 0; i < n; i++) {
        auto u = p[i];
        auto v = p[(i + 1) % n];
        auto w = p[(i + 2) % n];
        auto [t, p1, p2] = segmentIntersection(l, Line(u, v));

        if (t == 1) {
            return false;
        }
        if (t == 0) {
            continue;
        }
        if (t == 2) {
            if (pointOnSegment(v, l) && v != l.a && v != l.b) {
                if (cross(v - u, w - v) > 0) {
                    return false;
                }
            }
        } else {
            if (p1 != u && p1 != v) {
                if (pointOnLineLeft(l.a, Line(v, u))
                    || pointOnLineLeft(l.b, Line(v, u))) {
                    return false;
                }
            } else if (p1 == v) {
                if (l.a == v) {
                    if (pointOnLineLeft(u, l)) {
                        if (pointOnLineLeft(w, l)
                            && pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, l)
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    }
                } else if (l.b == v) {
                    if (pointOnLineLeft(u, Line(l.b, l.a))) {
                        if (pointOnLineLeft(w, Line(l.b, l.a))
                            && pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, Line(l.b, l.a))
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    }
                } else {
                    if (pointOnLineLeft(u, l)) {
                        if (pointOnLineLeft(w, Line(l.b, l.a))
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    } else {
                        if (pointOnLineLeft(w, l)
                            || pointOnLineLeft(w, Line(u, v))) {
                            return false;
                        }
                    }
                }
            }
        }
    }
    return true;
}

template<class T>
std::vector<Point<T>> hp(std::vector<Line<T>> lines) {
    std::sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
        auto d1 = l1.b - l1.a;
        auto d2 = l2.b - l2.a;

        if (sgn(d1) != sgn(d2)) {
            return sgn(d1) == 1;
        }

        return cross(d1, d2) > 0;
    });

    std::deque<Line<T>> ls;
    std::deque<Point<T>> ps;
    for (auto l : lines) {
        if (ls.empty()) {
            ls.push_back(l);
            continue;
        }

        while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
            ps.pop_back();
            ls.pop_back();
        }

        while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
            ps.pop_front();
            ls.pop_front();
        }

        if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
            if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {

                if (!pointOnLineLeft(ls.back().a, l)) {
                    assert(ls.size() == 1);
                    ls[0] = l;
                }
                continue;
            }
            return {};
        }

        ps.push_back(lineIntersection(ls.back(), l));
        ls.push_back(l);
    }

    while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
        ps.pop_back();
        ls.pop_back();
    }
    if (ls.size() <= 2) {
        return {};
    }
    ps.push_back(lineIntersection(ls[0], ls.back()));

    return std::vector(ps.begin(), ps.end());
}

using P = Point<double>;
```

6 Util

6.1 Mod Integer

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

```cpp
template <class T>
constexpr T power(T a, u64 b, T res = 1) {
    for (; b != 0; b /= 2, a *= a) {
        if (b & 1) {
            res *= a;
        }
    }
    return res;
}
template <u32 P>
constexpr u32 mulMod(u32 a, u32 b) {
    return u64(a) * b % P;
}
template <u64 P>
constexpr u64 mulMod(u64 a, u64 b) {
    u64 res = a * b - u64(1.L * a * b / P - 0.5L) * P;
    res %= P;
    return res;
}
template <std::unsigned_integral U, U P>
struct ModIntBase {
public:
    constexpr ModIntBase() : x(0) {}
    template <std::unsigned_integral T>
    constexpr ModIntBase(T x_) : x(x_ % mod()) {}
    template <std::signed_integral T>
    constexpr ModIntBase(T x_) {
        using S = std::make_signed_t<U>;
        S v = x_ % S(mod());
        if (v < 0) {
            v += mod();
        }
        x = v;
    }
    constexpr static U mod() {
        return P;
    }
    constexpr U val() const {
        return x;
    }
    constexpr ModIntBase operator-() const {
        ModIntBase res;
        res.x = (x == 0 ? 0 : mod() - x);
        return res;
    }
    constexpr ModIntBase inv() const {
        return power(*this, mod() - 2);
    }
    constexpr ModIntBase pow(u64 b) const {
        return power(*this, b);
    }
    constexpr ModIntBase &operator*=(const ModIntBase &rhs) & {
        x = mulMod<mod()>(x, rhs.val());
        return *this;
    }
    constexpr ModIntBase &operator+=(const ModIntBase &rhs) & {
        x += rhs.val();
        if (x >= mod()) {
            x -= mod();
        }
        return *this;
    }
    constexpr ModIntBase &operator-=(const ModIntBase &rhs) & {
        x -= rhs.val();
        if (x >= mod()) {
            x += mod();
        }
        return *this;
    }
    constexpr ModIntBase &operator/=(const ModIntBase &rhs) & {
        return *this *= rhs.inv();
    }

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

    friend constexpr std::istream &operator>>(std::istream &is, ModIntBase &a) {
        i64 i;
        is >> i;
        a = i;
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const ModIntBase &a) {
        return os << a.val();
    }

    friend constexpr bool operator==(const ModIntBase &lhs, const ModIntBase &rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr std::strong_ordering operator<=>(const ModIntBase &lhs, const ModIntBase &rhs) {
        return lhs.val() <=> rhs.val();
    }

private:
    U x;
};

template <u32 P>
using ModInt = ModIntBase<u32, P>;
template <u64 P>
using ModInt64 = ModIntBase<u64, P>;

template <int V, u32 P>
constexpr ModInt<P> CInv = ModInt<P>(V).inv();

using Z = ModInt<998244353>;
```

6.2 Fraction

```cpp
struct Frac {
    i64 num, den;
    constexpr Frac(i64 x = 0) : Frac(x, 1) {}
    constexpr Frac(i64 num, i64 den) : num{num}, den{den} {
        norm();
    }
    constexpr void norm() {
        if (den < 0) {
            num = -num;
            den = -den;
        }
    }
    constexpr Frac operator-() const {
        return Frac(-num, den);
    }
    constexpr Frac &operator+=(const Frac &rhs) {
        num = num * rhs.den + rhs.num * den;
        den *= rhs.den;
        norm();
        return *this;
    }
    constexpr Frac &operator-=(const Frac &rhs) {
        return *this += -rhs;
    }
    constexpr Frac &operator*=(const Frac &rhs) {
        num *= rhs.num;
        den *= rhs.den;
        norm();
        return *this;
    }
    constexpr Frac &operator/=(const Frac &rhs) {
        num *= rhs.den;
        den *= rhs.num;
        norm();
        return *this;
    }
    friend constexpr Frac operator+(const Frac &lhs, const Frac &rhs) {
        Frac res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr Frac operator-(const Frac &lhs, const Frac &rhs) {
        Frac res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr Frac operator*(const Frac &lhs, const Frac &rhs) {
        Frac res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr Frac operator/(const Frac &lhs, const Frac &rhs) {
        Frac res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr bool operator<(const Frac &lhs, const Frac &rhs) {
        return static_cast<__int128>(lhs.num) * rhs.den < static_cast<__int128>(rhs.num) * lhs.den;
    }
    friend constexpr bool operator>(const Frac &lhs, const Frac &rhs) {
        return static_cast<__int128>(lhs.num) * rhs.den > static_cast<__int128>(rhs.num) * lhs.den;
    }
    friend std::ostream &operator<<(std::ostream &os, const Frac &rhs) {
        return os << rhs.num << "/" << rhs.den;
    }
};
```

7 Tables

7.1 Constant

n	$\log_{10} n$	$n!$	$C(n, n/2)$	$\mathrm{LCM}(1 \dots n)$	P_n
2	0.30102999	2	2	2	2
3	0.47712125	6	3	6	3
4	0.60205999	24	6	12	5
5	0.69897000	120	10	60	7
6	0.77815125	720	20	60	11
7	0.84509804	5040	35	420	15
8	0.90308998	40320	70	840	22
9	0.95424251	362880	126	2520	30
10	1	3628800	252	2520	42
11	1.04139269	39916800	462	27720	56
12	1.07918125	479001600	924	27720	77
15	1.17609126	1.31×10^{12}	6435	360360	176
20	1.30103000	2.43×10^{18}	184756	232792560	627
25	1.39794001	1.55×10^{25}	5200300	26771144400	1958
30	1.47712125	2.65×10^{32}	155117520	1.444×10^{14}	5604

$n \le$	10	100	10^3	10^4	10^5	10^6
$\max \omega(n)$	2	3	4	5	6	7
$\max d(n)$	4	12	32	64	128	240
$\pi(n)$	4	25	168	1229	9592	78498
$n \le$	10^7	10^8	10^9	10^{10}	10^{11}	10^{12}
$\max \omega(n)$	8	8	9	10	10	11
$\max d(n)$	448	768	1344	2304	4032	6720
$\pi(n)$	664579	5761455	5.08×10^7	4.55×10^8	4.12×10^9	3.7×10^{10}
$n \le$	10^{13}	10^{14}	10^{15}	10^{16}	10^{17}	10^{18}
$\max \omega(n)$	12	12	13	13	14	15
$\max d(n)$	10752	17280	26880	41472	64512	103680

Prime number theorem: $\pi(x) \sim x/\log(x)$