

ingenious plan
if i understand
it correctly

minor fix
no need to
test on samples

i make
no mistakes

let me
just fix
and resubmit

seems
proved

*whistles a
horny molly's song*

rofl it works

it's a bitset
how can it
not pass

i kinda know
how to prove it

5-10 mins
to implement

there is
a column
of ACs so
greedy works ofc

MOSCOW PHYSICS AND TECHNOLOGY

THE
MATHEMATICIAN

THE
ADEPT

THE
PROGRAMMER

THE
COACH



Содержание

1	centroids	1
1.1	centroid_decomposition.cpp	1
2	fft	1
2.1	fft_advanced_integer.h	1
2.2	fft_double.h	3
2.3	fft_integer.h	3
2.4	fft_mod_10_9_7.h	4
3	flows	5
3.1	dinic.h	5
3.2	min_cost_bellman_queue.h	5
3.3	min_cost_dijkstra.h	5
3.4	min_cost_ford_bellman.h	7
3.5	min_cost_negative_cycles.h	7
4	geometry	8
4.1	halfplane_intersection.cpp	8
4.2	segments_and_circles.cpp	10
5	graphs	10
5.1	components.cpp	10
5.2	dominator_tree.h	11
6	maths	12
6.1	berlekamp.h	12
6.2	crt.h	12
6.3	extended_euclidean.h	13
6.4	gauss_bitset_inverse.h	13
6.5	gauss_bitset_solve_slv.h	13
6.6	gauss_double_inverse.h	13
6.7	gauss_double_solve_slv.h	14
7	misc	14
7.1	ch_trick_with_binary_summation_struct.cpp	14
7.2	tree_bidirectional_dp.h	15
8	strings	15
8.1	aho_corasick.h	15
8.2	manacher.h	16
8.3	palindromes_on_subsegment.h	16
8.4	prefix_function.h	17
8.5	suffix_array.cpp	18
8.6	suffix_automaton_kostroma.h	19
8.7	z_function.h	19
9	templates	19
9.1	main.cpp	19
9.2	sync-template.txt	20
9.3	template.h	20
10	treap	20
10.1	treap_explicit_keys.h	20
10.2	treap_implicit_keys.h	21
11	fuckups.tex	22

1 centroids

1.1 centroid_decomposition.cpp

```
vector<vector<int>> g;
vector<int> cnt, max_cnt;
vector<int> comp;

void dfs1(int v, int p) {
    cnt[v] = 1;
    max_cnt[v] = 0;
    comp.push_back(v);
    for (int to : g[v]) {
        if (to == p || used[to]) continue;
        dfs1(to, v);
        max_cnt[v] = max(max_cnt[v], cnt[to]);
        cnt[v] += cnt[to];
    }
}

void kill_center(int v, int depth) {
    if (used[v]) {
        return;
    }
    comp.clear();
    dfs1(v, v);
    int center = -1;
    for (int x : comp) {
        if (max_cnt[x] <= cnt[v] / 2 && cnt[v] - cnt[x] <= cnt[v] / 2) {
            center = x;
            break;
        }
    }
    assert(center != -1);
    v = center;
    // perform actions with center v
    used[v] = true;
    for (int to : g[v]) {
        kill_center(to, depth + 1);
    }
}
```

```
void solve(__attribute__((unused)) bool read) {
    int n;
    cin >> n;
    used.assign(n, false);
    cnt.assign(n, 0);
    max_cnt.assign(n, 0);
    kill_center(0, 0);
}

2 fft

2.1 fft_advanced_integer.h

Poly derivative(Poly a) {
    if (a.empty()) {
        return a;
    }
    for (int i = 0; i < (int)a.size(); ++i) {
        a[i] = a[i] * i % mod;
    }
    a.erase(a.begin());
    return a;
}

// returns b(x) = int_0^x{a(t)dt}
Poly primitive(Poly a) {
    if (a.empty()) {
        return a;
    }
    for (int i = 0; i < (int)a.size(); ++i) {
        a[i] = a[i] * pw(i + 1, mod - 2) % mod;
    }
    a.insert(a.begin(), 0);
    return a;
}

Poly add(Poly a, const Poly& b) {
    a.resize(max(a.size(), b.size()));
    for (int i = 0; i < (int)b.size(); ++i) {
        a[i] = (a[i] + b[i]) % mod;
    }
    return a;
}

Poly sub(Poly a, const Poly& b) {
    a.resize(max(a.size(), b.size()));
    for (int i = 0; i < (int)b.size(); ++i) {
        a[i] = (a[i] + mod - b[i]) % mod;
    }
    return a;
}

Poly normalize(Poly a) {
    while (!a.empty() && a.back() == 0) {
        a.pop_back();
    }
    return a;
}

// get such b that a * b = 1 mod x^{prec}
```

```

Poly getInversed(Poly a, int prec) {
    assert(a[0]);

    Poly res = {pw(a[0], mod - 2)};
    int k = 1;
    while (k < prec) {
        k *= 2;
        Poly tmp = multiply(res, Poly({a.begin(),
    ↪ a.begin() + min(k, (int)a.size())}));
        for (auto& x : tmp) {
            x = x ? mod - x : 0;
        }
        tmp[0] = (tmp[0] + 2) % mod;

        res = multiply(tmp, res);
        res.resize(k);
    }
    res.resize(prec);
    return res;
}

// get such q and r that a = b * q + r, deg(r) < deg(b)
pair<Poly, Poly> divMod(Poly a, Poly b) {
    int n = a.size();
    int m = b.size();
    if (n < m) {
        return {{0}, a};
    }
    reverse(all(a));
    reverse(all(b));
    auto quotient = multiply(a, getInversed(b, n -
    ↪ m + 1));
    quotient.resize(n - m + 1);
    reverse(all(a));
    reverse(all(b));
    reverse(all(quotient));
    auto remainder = sub(a, multiply(b, quotient));
    while (!remainder.empty() && remainder.back()
    ↪ == 0) {
        remainder.pop_back();
    }
    return {quotient, remainder};
}

// this is for multipoint and interpolate functions
vector<Poly> getSegmentProducts(const vector<long
    ↪ long>& pts) {
    vector<Poly> segment_polys;
    function<int(int, int)> fill_polys = [&](int
    ↪ l, int r) {
        if (l + 1 == r) {
            segment_polys.push_back({(mod -
    ↪ pts[l]) % mod, 1});
            return (int)segment_polys.size() - 1;
        }
        int m = (l + r) / 2;
        int i = fill_polys(l, m);
        int j = fill_polys(m, r);
        auto new_poly = multiply(segment_polys[i],
    ↪ segment_polys[j]);
        segment_polys.push_back(new_poly);
        return (int)segment_polys.size() - 1;
    };
    fill_polys(0, pts.size());

    return segment_polys;
}

// get p and {x1, x2, ..., xn}, return {p(x1),
    ↪ p(x2), ..., p(xn)}
vector<long long> multipoint(const Poly& poly,
    ↪ const vector<long long>& pts) {
    if (pts.empty()) {
        return {};
    }

    vector<Poly> segment_polys =
    ↪ getSegmentProducts(pts);
    vector<long long> ans;
    function<void(const Poly&)> fill_ans =
    ↪ [&](const Poly& p) {
        if ((int)segment_polys.back().size() <= 2) {
            ans.push_back(p.empty() ? 0 : p[0]);
        }
        segment_polys.pop_back();
        return;
    }
    segment_polys.pop_back();
    fill_ans(divMod(p,
    ↪ segment_polys.back()).second);
    fill_ans(divMod(p,
    ↪ segment_polys.back()).second);
    };
    fill_ans(poly);
    reverse(all(ans));

    return ans;
}

// get {x1, ..., xn} and {y1, ..., yn}, return
    ↪ such p that p(xi) = yi
Poly interpolate(const vector<long long>& xs,
    ↪ const vector<long long>& ys) {
    assert(xs.size() == ys.size());
    if (xs.empty()) {
        return {0};
    }

    vector<Poly> segment_polys =
    ↪ getSegmentProducts(xs);
    auto der = derivative(segment_polys.back());
    auto coeffs = multipoint(der, xs);
    for (auto& c : coeffs) {
        c = pw(c, mod - 2);
    }
    for (int i = 0; i < (int)ys.size(); ++i) {
        coeffs[i] = coeffs[i] * ys[i] % mod;
    }

    function<Poly()> get_ans = [&]() {
        Poly res;
        if (segment_polys.back().size() <= 2) {
            segment_polys.pop_back();
            res = {coeffs.back()};
            coeffs.pop_back();
        } else {
            segment_polys.pop_back();

            auto p1 = segment_polys.back();
            auto q1 = get_ans();

            auto p2 = segment_polys.back();
            auto q2 = get_ans();

            res = add(multiply(p1, q2),
    ↪ multiply(p2, q1));
            return res;
        };
        return normalize(get_ans());
    }

    // takes 1 + b, returns b - b^2/2 + b^3/3 - ...
    ↪ mod x^{prec}
    // ofc b must be divisible by x
Poly logarithm(Poly a, int prec) {
    assert(a[0] == 1);
    auto res = primitive(multiply(derivative(a),
    ↪ getInversed(a, prec)));
    res.resize(prec);
    return res;
}

// returns 1 + a + a^2/2 + a^3/6 + ... mod x^{prec}
    ↪ // ofc a must be divisible by x
Poly exponent(Poly a, int prec) {
    assert(a[0] == 0);

    Poly res = {0};
    int k = 1;
    while (k < prec) {
        k *= 2;
        Poly tmp = {a.begin(), a.begin() + min(k,
    ↪ (int)a.size())};
        tmp[0] += 1;
        tmp = sub(tmp, logarithm(res, k));
    }

```

```

        res = multiply(tmp, res);
        res.resize(k);
    }
    res.resize(prec);
    return res;
}

```

2.2 fft_double.h

```

const int L = 22;
const int N = 1 << L;
bool fft_initialized = false;

using ld = long double;
using base = complex<ld>;
using Poly = vector<ld>;

const ld pi = acos(-1);
base angles[N + 1];
int bitrev[N];

// don't know why such eps, may be changed
const ld eps = 1e-7;

inline bool eq(ld x, ld y) {
    return abs(x - y) < eps;
}

void fft_init() {
    for (int i = 0; i <= N; ++i) {
        angles[i] = {cos(2 * pi * i / N), sin(2
        * pi * i / N)};
    }

    for (int i = 0; i < N; ++i) {
        int x = i;
        for (int j = 0; j < L; ++j) {
            bitrev[i] = (bitrev[i] << 1) | (x & 1);
            x >>= 1;
        }
    }

    fft_initialized = true;
}

inline int revBit(int x, int len) {
    return bitrev[x] >> (L - len);
}

void fft(vector<base>& a, bool inverse = false) {
    assert(fft_initialized &&
    "you fucking cunt just write fft_init()");
    int n = a.size();
    assert(!(n & (n - 1))); // work only with
    powers of two
    int l = __builtin_ctz(n);

    for (int i = 0; i < n; ++i) {
        int j = revBit(i, l);
        if (i < j) {
            swap(a[i], a[j]);
        }
    }

    for (int len = 1; len < n; len *= 2) {
        for (int start = 0; start < n; start += 2
        * len) {
            for (int i = 0; i < len; ++i) {
                base x = a[start + i], y = a[start
                + len + i];
                int idx = N / 2 / len * i;
                base w = y * angles[inverse ? N -
                idx : idx];
                a[start + i] = x + w;
                a[start + len + i] = x - w;
            }
        }
    }

    if (inverse) {
        for (auto& x : a) {
            x /= n;
        }
    }
}

```

```

    }
}

Poly multiply(Poly a, Poly b) {
    int n = 1;
    while (n < (int)a.size() || n < (int)b.size()) {
        n *= 2;
    }
    clock_t start = clock();
    vector<base> ar(n + n), br(n + n);
    for (int i = 0; i < (int)a.size(); ++i) {
        ar[i] = a[i];
    }
    for (int i = 0; i < (int)b.size(); ++i) {
        br[i] = b[i];
    }
    fft(ar);
    fft(br);
    for (int i = 0; i < n + n; ++i) {
        ar[i] = ar[i] * br[i];
    }
    fft(ar, true);
    while (!ar.empty() && eq(norm(ar.back()), 0)) {
        ar.pop_back();
    }
    a.resize(ar.size());
    for (int i = 0; i < (int)a.size(); ++i) {
        a[i] = real(ar[i]);
    }
    return a;
}

```

2.3 fft_integer.h

```

const int mod = 998244353;
const int L = 22; // can be 23 for 998244353
const int N = 1 << L;
bool fft_initialized = false;

using Poly = vector<long long>;

long long pw(long long a, long long b) {
    long long res = 1;
    while (b) {
        if (b & 1) {
            res = res * a % mod;
        }
        b >>= 1;
        a = a * a % mod;
    }
    return res;
}

int getRoot() {
    int root = 1;
    while (pw(root, 1 << L) != 1 || pw(root, 1 <<
    (L - 1)) == 1) {
        ++root;
    }
    return root;
}

const int root = getRoot();

long long angles[N + 1];
int bitrev[N];

void fft_init() {
    angles[0] = 1;
    for (int i = 1; i <= N; ++i) {
        angles[i] = angles[i - 1] * root % mod;
    }

    for (int i = 0; i < N; ++i) {
        int x = i;
        for (int j = 0; j < L; ++j) {
            bitrev[i] = (bitrev[i] << 1) | (x & 1);
            x >>= 1;
        }
    }

    fft_initialized = true;
}

```

```

inline int revBit(int x, int len) {
    return bitrev[x] >> (L - len);
}

void fft(vector<long long>& a, bool inverse = false) {
    assert(fft_initialized &&
    'you fucking cunt just write fft_init()');
    int n = a.size();
    assert(!(n & (n - 1))); // work only with
    powers of two
    int l = __builtin_ctz(n);

    for (int i = 0; i < n; ++i) {
        int j = revBit(i, l);
        if (i < j) {
            swap(a[i], a[j]);
        }

        for (int len = 1; len < n; len *= 2) {
            for (int start = 0; start < n; start += 2
            * len) {
                for (int i = 0; i < len; ++i) {
                    long long x = a[start + i], y =
                    a[start + len + i];
                    int idx = N / 2 / len * i;
                    long long w = angles[inverse ? N -
                    idx : idx];
                    w = w * y % mod;
                    a[start + i] = x + w;
                    if (a[start + i] >= mod) {
                        a[start + i] -= mod;
                    }
                    a[start + len + i] = x - w;
                    if (a[start + len + i] < 0) {
                        a[start + len + i] += mod;
                    }
                }
            }

            if (inverse) {
                int rev_deg = 1;
                for (int i = 0; i < l; ++i) {
                    rev_deg = (rev_deg % 2) ? ((rev_deg +
                    mod) / 2) : (rev_deg / 2);
                }
                for (auto& x : a) {
                    x = x * rev_deg % mod;
                }
            }
        }
    }

    Poly multiply(Poly a, Poly b) {
        int n = 1;
        while (n < (int)a.size() || n < (int)b.size()) {
            n *= 2;
        }
        a.resize(n + n);
        b.resize(n + n);
        fft(a);
        fft(b);
        for (int i = 0; i < n + n; ++i) {
            a[i] = a[i] * b[i] % mod;
        }
        fft(a, true);
        while (!a.empty() && a.back() == 0) {
            a.pop_back();
        }
        return a;
    }
}

```

2.4 fft_mod_10_9_7.h

```

const int MAX_FFT_N = 1 << 19;

ld PI = acos((ld)-1.0);

ld co[MAX_FFT_N], si[MAX_FFT_N];

void precalc() {
    for (int i = 0; i < MAX_FFT_N; ++i) {

```

```

        co[i] = cos(2 * PI * i / MAX_FFT_N);
        si[i] = sin(2 * PI * i / MAX_FFT_N);
    }
}

#define double ld

using base = complex<double>;
using Poly = vector<long long>;

void fft(vector<base> & a, bool invert) {
    int n = (int)a.size();

    for (int i = 1, j = 0; i < n; ++i) {
        int bit = n >> 1;
        for (; j >= bit; bit >>= 1)
            j -= bit;
        j += bit;
        if (i < j)
            swap(a[i], a[j]);
    }

    for (int len = 2; len <= n; len <= 1) {
        int dom = (invert ? -1 : 1);
        int DOM = MAX_FFT_N / len;
        for (int i = 0; i < n; i += len) {
            base w(1);
            for (int j = 0; j < len / 2; ++j) {
                w = base(co[DOM * j], si[DOM * j]
                * dom);
                auto u = a[i + j], v = a[i + j +
                len / 2] * w;
                a[i + j] = u + v;
                a[i + j + len / 2] = u - v;
            }
        }
        if (invert) {
            for (int i = 0; i < n; ++i) {
                a[i] /= (double)n;
            }
        }
    }

    Poly multiply(const Poly& a, const Poly& b) {
        int n = 1;
        while (n <= a.size() || n <= b.size())
            n <= 1;
        n <= 1;
        vector<base> input[2];
        for (int w = 0; w < 2; ++w)
            input[w].assign(n, base(0, 0));
        for (int i = 0; i < a.size(); ++i)
            input[0][i] = a[i];
        for (int i = 0; i < b.size(); ++i)
            input[1][i] = b[i];
        for (auto& vec : input) {
            fft(vec, false);
        }
        vector<base> res(n);
        for (int i = 0; i < n; ++i)
            res[i] = input[0][i] * input[1][i];
        fft(res, true);
        Poly answer(n);
        for (int i = 0; i < n; ++i) {
            answer[i] = (ld)(res[i].real() + 0.5);
            answer[i] %= mod;
        }
        return answer;
    }

    const int shift = 15;

    const int first_mod = 1 << shift;

    Poly large_part(const Poly& a) {
        Poly res(a.size());
        for (int i = 0; i < a.size(); ++i) {
            res[i] = a[i] >> shift;
        }
        return res;
    }
}

```

```

Poly small_part(const Poly& a) {
    Poly res(a.size());
    for (int i = 0; i < a.size(); ++i) {
        res[i] = a[i] & (first_mod - 1);
    }
    return res;
}

Poly add(const Poly& q, const Poly& w) {
    auto res = q;
    res.resize(max(q.size(), w.size()));
    for (int i = 0; i < w.size(); ++i) {
        res[i] += w[i];
    }
    return res;
}

Poly multiply_large(const Poly& a, const Poly& b,
    ↪ int k) {
    Poly largeA = large_part(a), largeB =
    ↪ large_part(b);
    Poly smallA = small_part(a), smallB =
    ↪ small_part(b);
    Poly large_mult = multiply(largeA, largeB);
    Poly small_mult = multiply(smallA, smallB);
    Poly middle_mult = multiply(add(smallA,
    ↪ largeA), add(smallB, largeB));

    Poly result(large_mult.size());
    for (int i = 0; i < result.size(); ++i) {
        result[i] = ((large_mult[i] * first_mod) %
    ↪ mod * first_mod + small_mult[i] +
        first_mod * (middle_mult[i] -
    ↪ large_mult[i] - small_mult[i]) % mod) % mod;
    }
    if (result.size() > k + 1) {
        result.resize(k + 1);
    }
    return result;
}

```

3 flows

3.1 dinic.h

```

struct Edge {
    int from, to, cap, flow;
};

const int INF = (int)2e9;

struct Dinic {
    int n;
    vector<Edge> edges;
    vector<vector<int>>> g;

    Dinic(int n) : n(n) {
        g.resize(n);
    }

    void add_edge(int from, int to, int cap) {
        Edge e = {from, to, cap, 0};
        g[from].push_back(edges.size());
        edges.push_back(e);
        e = {to, from, 0, 0};
        g[to].push_back(edges.size());
        edges.push_back(e);
    }

    vector<int> d;

    bool bfs(int s, int t) {
        d.assign(n, INF);
        d[s] = 0;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (auto id : g[v]) {
                auto e = edges[id];

```

```

                if (e.cap > e.flow && d[e.to] == INF) {
                    d[e.to] = d[v] + 1;
                    q.push(e.to);
                }
            }
        }
        return d[t] != INF;
    }

    vector<int> pointer;

    int dfs(int v, int t, int flow_add) {
        if (!flow_add) {
            return 0;
        }
        if (v == t) {
            return flow_add;
        }
        int added_flow = 0;
        for (int& i = pointer[v]; i < g[v].size();
    ↪ ++i) {
            int id = g[v][i];
            int to = edges[id].to;
            if (d[to] != d[v] + 1) {
                continue;
            }
            int pushed = dfs(to, t, min(flow_add,
    ↪ edges[id].cap - edges[id].flow));
            if (pushed) {
                edges[id].flow += pushed;
                edges[id ^ 1].flow -= pushed;
                return pushed;
            }
        }
        return 0;
    }

    int max_flow(int s, int t) {
        int flow = 0;
        while (bfs(s, t)) {
            pointer.assign(n, 0);
            while (int pushed = dfs(s, t, INF)) {
                flow += pushed;
            }
        }
        return flow;
    }
};

```

3.2 min_cost_bellman_queue.h

```

using cost_type = li;
const cost_type COST_INF = (int)1e18;
const int FLOW_INF = (int)1e9;

struct MinCost {
    explicit MinCost(int n) {
        g.resize(n);
    }

    struct edge {
        int from, to;
        int cap;
        cost_type cost;
        int flow;
    };

    vector<edge> edges;
    vector<vector<int>>> g;

    void add_edge(int from, int to, cost_type
    ↪ cost, int cap) {
        edge e = {from, to, cap, cost, 0};
        g[from].push_back(edges.size());
        edges.push_back(e);
        edge e2 = {to, from, 0, -cost, 0};
        g[to].push_back(edges.size());
        edges.push_back(e2);
    }

    pair<int, cost_type> min_cost(int n, int s,
    ↪ int t, bool need_max_flow, int max_flow_value
    ↪ = FLOW_INF) {

```



```

cost_type cost = 0;
int flow = 0;
while (flow < max_flow_value) {
    queue<int> q;
    q.push(s);
    vector<int> in_q(n, 0);
    in_q[s] = 1;
    vector<int> p(n, -1);
    vector<cost_type> d(n);
    d[s] = 0;
    p[s] = s;
    while (!q.empty()) {
        int v = q.front();
        q.pop();
        in_q[v] = false;
        for (size_t i: g[v]) {
            edge& e = edges[i];
            if (e.cap == e.flow ||
→ p[e.from] == -1)
                continue;
            if (p[e.to] == -1 || d[e.to] > ←
→ d[e.from] + e.cost) {
                d[e.to] = d[e.from] + e.cost;
                p[e.to] = i;
                if (!in_q[e.to]) {
                    in_q[e.to] = 1;
                    q.push(e.to);
                }
            }
        }
        if (p[t] == -1)
            break;

        if (d[t] >= 0 && !need_max_flow) {
            break;
        }

        int cur = t;
        int maxAdd = max_flow_value - flow;
        while (cur != s) {
            edge& e = edges[p[cur]];
            cur = e.from;
            maxAdd = min(maxAdd, e.cap - e.flow);
        }

        flow += maxAdd;
        cost += d[t] * maxAdd;
        cur = t;
        while (cur != s) {
            int id = p[cur];
            edges[id].flow += maxAdd;
            edges[id ^ 1].flow -= maxAdd;
            cur = edges[id].from;
        }

        return make_pair(flow, cost);
    }
};

```

3.3 min_cost_dijkstra.h

```

#define int li

using cost_type = li;
const cost_type COST_INF = (int)1e18;
const int FLOW_INF = (int)1e9;

struct MinCost {
    explicit MinCost(int n) {
        g.resize(n);
    }

    struct edge {
        int from, to;
        int cap;
        cost_type cost;
        int flow;
    };

    vector<edge> edges;
    vector<vector<int>> g;

```

```

    void add_edge(int from, int to, cost_type ←
→ cost, int cap) {
        edge e = {from, to, cap, cost, 0};
        g[from].push_back(edges.size());
        edges.push_back(e);
        edge e2 = {to, from, 0, -cost, 0};
        g[to].push_back(edges.size());
        edges.push_back(e2);
    }

    pair<int, cost_type> min_cost(int n, int s, ←
→ int t, bool need_max_flow, int max_flow_value ←
→ = FLOW_INF) {
        cost_type cost = 0;
        int flow = 0;
        vector<cost_type> potential;
        {
            vector<int> p(n, -1);
            vector<cost_type> d(n);
            d[s] = 0;
            p[s] = s;
            bool changed = true;
            while (changed) {
                changed = false;
                for (size_t i = 0; i < ←
→ edges.size(); ++i) {
                    edge &e = edges[i];
                    if (e.cap == e.flow || ←
→ p[e.from] == -1)
                        continue;
                    if (p[e.to] == -1 || d[e.to] > ←
→ d[e.from] + e.cost) {
                        d[e.to] = d[e.from] + e.cost;
                        p[e.to] = i;
                        changed = true;
                    }
                }
                potential = std::move(d);
            }
            while (flow < max_flow_value) {
                vector<cost_type> d(n);
                vector<int> p(n, -1);

                using queue_type = pair<cost_type, int>;
                priority_queue<queue_type, ←
→ vector<queue_type>, greater<queue_type>> q;

                q.push({0, s});

                while (!q.empty()) {
                    int v = q.top().second;
                    cost_type oldD = q.top().first;
                    q.pop();
                    if (oldD != d[v])
                        continue;
                    for (int id: g[v]) {
                        edge &e = edges[id];
                        if (e.to == s)
                            continue;
                        if (e.cap > e.flow) {
                            cost_type newd = d[v] + ←
→ e.cost + potential[e.from] - potential[e.to];
                            if (p[e.to] == -1 || ←
→ d[e.to] > newd) {
                                d[e.to] = newd;
                                p[e.to] = id;
                                q.push({d[e.to], e.to});
                            }
                        }
                    }
                }

                if (p[t] == -1) {
                    break;
                }

                if (d[t] + potential[t] >= 0 && ←
→ !need_max_flow) {
                    break;
                }
            }

```

```

int cur = t;
int maxAdd = max_flow_value - flow;
while (cur != s) {
    edge &e = edges[p[cur]];
    cur = e.from;
    maxAdd = min(maxAdd, e.cap - e.flow);
}

flow += maxAdd;
cost += (potential[t] + d[t]) * maxAdd;
cur = t;
while (cur != s) {
    int id = p[cur];
    edges[id].flow += maxAdd;
    edges[id ^ 1].flow -= maxAdd;
    cur = edges[id].from;
}

for (int i = 0; i < n; ++i) {
    if (i != s && p[i] == -1) {
        potential[i] = COST_INF;
    } else
        potential[i] =
    ↪ min(potential[i] + d[i], COST_INF);
}

return make_pair(flow, cost);
}
};

```

3.4 min_cost_ford_bellman.h

```

using cost_type = li;
const cost_type COST_INF = (int)1e18;
const int FLOW_INF = (int)1e9;

struct MinCost {
    explicit MinCost(int n) {
        g.resize(n);
    }

    struct edge {
        int from, to;
        int cap;
        cost_type cost;
        int flow;
    };

    vector<edge> edges;
    vector<vector<int>>> g;

    void add_edge(int from, int to, cost_type
    ↪ cost, int cap) {
        edge e = {from, to, cap, cost, 0};
        g[from].push_back(edges.size());
        edges.push_back(e);
        edge e2 = {to, from, 0, -cost, 0};
        g[to].push_back(edges.size());
        edges.push_back(e2);
    }

    pair<int, cost_type> min_cost(int n, int s,
    ↪ int t, bool need_max_flow, int max_flow_value
    ↪ = FLOW_INF) {
        cost_type cost = 0;
        int flow = 0;
        while (flow < max_flow_value) {
            vector<int> p(n, -1);
            vector<cost_type> d(n);
            d[s] = 0;
            p[s] = s;
            bool changed = true;
            while (changed) {
                changed = false;
                for (size_t i = 0; i <
                ↪ edges.size(); ++i) {
                    edge& e = edges[i];
                    if (e.cap == e.flow ||
                    ↪ p[e.from] == -1)
                        continue;
                    if (p[e.to] == -1 || d[e.to] >
                    ↪ d[e.from] + e.cost) {

```

```

d[e.to] = d[e.from] + e.cost;
p[e.to] = i;
changed = true;
        }
    }
    if (p[t] == -1)
        break;

    if (d[t] >= 0 && !need_max_flow) {
        break;
    }

    int cur = t;
    int maxAdd = max_flow_value - flow;
    while (cur != s) {
        edge& e = edges[p[cur]];
        cur = e.from;
        maxAdd = min(maxAdd, e.cap - e.flow);
    }

    flow += maxAdd;
    cost += d[t] * maxAdd;
    cur = t;
    while (cur != s) {
        int id = p[cur];
        edges[id].flow += maxAdd;
        edges[id ^ 1].flow -= maxAdd;
        cur = edges[id].from;
    }

    return make_pair(flow, cost);
}
};

```

3.5 min_cost_negative_cycles.h

```

using cost_type = int;
const cost_type COST_INF = (cost_type)1e9;
const int FLOW_INF = (int)1e9;

struct MinCost {
    explicit MinCost(int n) {
        g.resize(n);
    }

    struct edge {
        int from, to;
        int cap;
        cost_type cost;
        int flow;
    };

    vector<edge> edges;
    vector<vector<int>>> g;

    void add_edge(int from, int to, cost_type
    ↪ cur_cost, int cap) {
        edge e = {from, to, cap, cur_cost, 0};
        g[from].push_back(edges.size());
        edges.push_back(e);
        edge e2 = {to, from, 0, -cur_cost, 0};
        g[to].push_back(edges.size());
        edges.push_back(e2);
    }

    pair<int, cost_type> min_cost(int n, int s,
    ↪ int t, int max_flow_value = FLOW_INF) {
        cost_type cost = 0;
        int flow = 0;

        vector<int> p(n);
        vector<cost_type> d(n, 0);
        vector<int> to_add;
        while (flow < max_flow_value) {
            p.assign(n, -1);
            d.assign(n, COST_INF);
            d[s] = 0;
            set<pair<cost_type, int>>> q;
            q.insert({0, s});
            vector<char> used(n, false);
            while (!q.empty()) {
                int v = q.begin()->second;

```



```

        q.erase(q.begin());
        used[v] = true;
        for (int i : g[v]) {
            auto& e = edges[i];
            if (e.cap == e.flow ||
→ used[e.to]) {
                continue;
            }
            cost_type new_d = d[v] + e.cost;
            if (d[e.to] > new_d) {
                q.erase({d[e.to], e.to});
                d[e.to] = new_d;
                q.insert({d[e.to], e.to});
                p[e.to] = i;
            }
        }
        if (p[t] == -1) {
            return {-1, 0};
        }
        int add_flow = max_flow_value - flow;
        int cur = t;
        to_add.clear();
        int add_cost = 0;
        while (cur != s) {
            auto& e = edges[p[cur]];
            add_flow = min(add_flow, e.cap -
→ e.flow);
            to_add.push_back(p[cur]);
            cur = e.from;
            add_cost += e.cost;
        }
        assert(add_flow > 0);
        flow += add_flow;
        cost += add_flow * add_cost;
        for (int x : to_add) {
            edges[x].flow += add_flow;
            edges[x ^ 1].flow -= add_flow;
        }
    }

    int TIMER = 0;
    vector<int> used_timer(n, 0);
    vector<char> used(n, false);
    vector<int> cur_edges;
    vector<int> edges_to_add;
    while (true) {
        p.assign(n, -1);
        d.assign(n, COST_INF);
        bool found = false;
        int iter = 0;
        for (int st = 0; st < s; ++st) {
            if (d[st] != COST_INF) {
                continue;
            }
            ++iter;
            d[st] = 0;
            vector<int> q, new_q;
            q.push_back(st);
            for (int it = 0; it < n; ++it) {
                ++TIMER;
                int changed = -1;
                for (int v : q) {
                    for (int i : g[v]) {
                        edge &e = edges[i];
                        if (e.cap == e.flow)
                            continue;
                        cost_type new_d = d[v]
→ + e.cost;
                        if (d[e.to] > new_d) {
                            d[e.to] = new_d;
                            p[e.to] = i;
                            changed = e.to;
                            if
→ (used_timer[e.to] != TIMER) {
                                used_timer[e.to] = TIMER;
                                new_q.push_back(e.to);
                            }
                        }
                    }
                }
            }
            if (changed == -1) {
                break;
            }
            sort(all(new_q));
            q.swap(new_q);
            new_q.clear();
            if (d[st] < 0) {
                changed = st;
                it = n - 1;
            }
            if (it == n - 1) {
                found = true;
                int bad_end = changed;
                used.assign(n, false);
                int cur = bad_end;
                cur_edges.clear();
                while (!used[cur]) {
                    used[cur] = true;
                    cur_edges.push_back(p[cur]);
                    cur = edges[p[cur]].from;
                }
                edges_to_add.clear();
                while
→ (edges[cur_edges.back()].to != cur) {
                    edges_to_add.push_back(cur_edges.back());
                    cur_edges.pop_back();
                }
                edges_to_add.push_back(cur_edges.back());
                int add_cost = 0, add_flow
→ = FLOW_INF;
                for (auto e_id :
→ edges_to_add) {
                    add_flow =
→ min(add_flow, edges[e_id].cap -
→ edges[e_id].flow);
                    add_cost +=
→ edges[e_id].cost;
                }
                cost += add_cost * add_flow;
                assert(add_flow > 0);
                assert(add_cost < 0);
                for (auto e_id :
→ edges_to_add) {
                    edges[e_id].flow +=
→ add_flow;
                    edges[e_id ^ 1].flow
→ -= add_flow;
                }
            }
            if (!found) {
                break;
            }
            return make_pair(flow, cost);
        }
    }
};

```

4 geometry

4.1 halfplane_intersection.cpp

```

#include <bits/stdc++.h>

#define all(x) (x).begin(), (x).end()

using namespace std;

using ld = double;

const ld eps = 1e-9;

struct point {
    ld x, y;

    point(ld x = 0, ld y = 0): x(x), y(y) {}

```

```

    point operator+(const point& p) const { return
↪ point(x + p.x, y + p.y); }
    point operator-(const point& p) const { return
↪ point(x - p.x, y - p.y); }

    point operator*(ld t) const { return point(x *
↪ t, y * t); }
    point operator/(ld t) const { return point(x /
↪ t, y / t); }

    point rot() const { return point(-y, x); }

    ld vprod(const point& p) const { return x *
↪ p.y - y * p.x; }
    ld sprod(const point& p) const { return x *
↪ p.x + y * p.y; }

    int half() const {
        if (y)
            return y < -eps;
        else
            return x < -eps;
    }

    ld sql() const { return x * x + y * y; }
    ld len() const { return sqrt(sql()); }

    bool operator<(const point& p) const { return
↪ make_pair(x, y) < make_pair(p.x, p.y); }
};

int sign(ld x) {
    return abs(x) > eps ? (x > 0 ? 1 : -1) : 0;
}

int vecLess(const point& a, const point& b) {
    if (a.half() != b.half())
        return a.half() < b.half() ? 1 : -1;
    else {
        return sign(a.vprod(b));
    }
}

struct halfplane {
    // ax + by + c >= 0
    ld a, b, c;
    int type;

    tuple<ld, ld, ld> get() const { return
↪ make_tuple(a, b, c); }
    bool operator<(const halfplane& rhs) const {
↪ return get() < rhs.get(); }

    point norm() const { return point(a, b); }

    point intersect(const halfplane& h) const {
        ld x = -c * h.b + b * h.c;
        ld y = a * -h.c + c * h.a;
        ld denum = a * h.b - b * h.a;
        return point(x / denum, y / denum);
    }
};

// does intersection of a and c belong to b?
// assumes that a.vprod(c) > 0!
bool interAccepted(const halfplane& a, const
↪ halfplane& b, const halfplane& c) {
    // Determinant of 3x3 matrix formed by a, b, c
    return a.a * (b.b * c.c - b.c * c.b) - a.b *
↪ (b.a * c.c - b.c * c.a) + a.c * (b.a * c.b -
↪ b.b * c.a) < 0;
}

void sanitizeHalfplanes(vector<halfplane>& planes,
↪ bool doAdd, bool doSort) {
    // Add bounding box
    const ld INF = 1e9;
    if (doAdd) {
        planes.push_back(halfplane { 1, 0, INF });
        planes.push_back(halfplane { -1, 0, INF });
        planes.push_back(halfplane { 0, 1, INF });
        planes.push_back(halfplane { 0, -1, INF });
    }

    // Normalize halfplanes. This is used when
    // selecting strictest of parallel halfplanes
    // NOT NEEDED if there are no collinear (and
    // not antiparallel) normals, but may improve
    // precision
    for (halfplane& h: planes) {
        ld len = h.norm().len();
        h.a /= len;
        h.b /= len;
        h.c /= len;
    }

    if (doSort)
        sort(all(planes), [&](halfplane& a,
↪ halfplane& b) { return vecLess(a.norm(),
↪ b.norm()) > 0; });
}

class polygon {
public:
    vector<point> pts;

    polygon(const vector<point>& pts =
↪ vector<point>()): pts(pts) {}

    ld getDoubleSquare() const {
        ld result = 0;
        int n = pts.size();
        for (int i = 1; i < n - 1; ++i) {
            result += (pts[i] -
↪ pts[0]).vprod(pts[i + 1] - pts[0]);
        }
        return abs(result);
    }

    // Returns halfplane through points a and b,
    // inner part is counter-clockwise from a->b segment
    halfplane byPoints(point a, point b) {
        // rot counter clockwise, n points to area
        // inside halfplane intersection
        point n = (b - a).rot();
        return halfplane { n.x, n.y, -n.sprod(a) };
    }

    // empty return polygon/vector denotes empty
    // intersection
    // degenerate intersections are reported as empty

    // CALL sanitizeHalfplanes WITH SORT AND/OR ADD
    // BOUNDING BOX BEFORE USING!
    polygon getPolygon(const vector<halfplane>& planes) {
        int l = 0, r = 0;
        static vector<halfplane> ans;
        ans.clear();
        ans.reserve(planes.size());

        for (int L = 0; L < planes.size(); ) {
            int R = L + 1;
            while (R < planes.size() &&
↪ abs(planes[L].norm().vprod(planes[R].norm()))
↪ < eps) ++R;

            // choose most powerful inequality among
            // those with equal normals
            // assumes that normals are identity!
            const halfplane& h =
↪ *min_element(planes.begin() + L,
↪ planes.begin() + R, [&](const halfplane& a,
↪ const halfplane& b) { return a.c < b.c; });
            L = R;

            while (r - l > 1 && !interAccepted(ans[r -
↪ 2], h, ans[r - 1])) {
                ans.pop_back();
                --r;
            }

            while (r - l > 1 && !interAccepted(ans[l],
↪ h, ans[l + 1])) {
                ++l;
            }
        }
    }
}

```

```

// WATCH OUT: you may need to tweak eps
↪ here for severe problems
    if (r - 1 > 0 && ans[r -
↪ 1].norm().vprod(h.norm()) <= -1e-7) {
        return polygon();
    }

    if (r - 1 < 2 || interAccepted(ans[r - 1],
↪ ans[l], h)) {
        ans.push_back(h);
        r++;
    }

    assert(r == ans.size());

// IF YOU NEED HALFPLANES:
// return vector<halfplane>(ans.begin() + 1,
↪ ans.end());

int n = r - 1;

polygon poly;
poly.pts.reserve(n);
for (int i = 0; i < n; ++i) {
    poly.pts.push_back(ans[l +
↪ i].intersect(ans[l + (i + 1) % n]));
}

return poly;
}

```

4.2 segments_and_circles.cpp

```

struct point {
    ld x, y;

    point(ld x = 0, ld y = 0): x(x), y(y) {}

    point operator+(const point& p) const { return
↪ point(x + p.x, y + p.y); }
    point operator-(const point& p) const { return
↪ point(x - p.x, y - p.y); }

    point operator*(ld t) const { return point(x *
↪ t, y * t); }
    point operator/(ld t) const { return point(x /
↪ t, y / t); }

    ld vprod(const point& p) const { return x *
↪ p.y - y * p.x; }
    ld sprod(const point& p) const { return x *
↪ p.x + y * p.y; }

    point rot() const { return point(-y, x); }

    point norm() const { return *this / len(); }
    bool valid() const { return isfinite(x); }

    ld len() const { return hypot(x, y); }
    ld sql() const { return x * x + y * y; }

    int half() const {
        if (abs(y) > eps)
            return y < 0;
        else
            return x < -eps;
    }
};

point invalid(INFINITY, INFINITY);

point segmentIntersect(point a, point b, point c,
↪ point d) {
    b = b - a;
    d = d - c;

    if (abs(b.vprod(d)) < eps) return invalid;

    // a + bu = c + dv
    ld u = (c - a).vprod(d) / b.vprod(d);
    ld v = (a - c).vprod(b) / d.vprod(b);

```

```

    if (u >= -eps && v >= -eps && u <= 1 + eps &&
↪ v <= 1 + eps)
        return a + b * u;

    return invalid;
}

vector<point> lineCircleIntersect(point a, point
↪ b, point c, ld r) {
    point n = (b - a).norm().rot();
    ld d = n.sprod(a - c);
    if (abs(d) > r + eps) return {};

    if (abs(abs(d) - r) < eps)
        return { c + n * d };

    ld x = sqrt(max<ld>(0, r * r - d * d));
    return { c + n * d + n.rot() * x, c + n * d -
↪ n.rot() * x };
}

vector<point> segmentCircleIntersect(point a,
↪ point b, point c, ld r) {
    auto pts = lineCircleIntersect(a, b, c, r);

    vector<point> ans;
    for (point& p: pts) {
        assert(abs((p - c).len() - r) < eps);
        assert(abs((p - a).vprod(b - a)) < eps);

        if ((p - a).sprod(p - b) <= eps)
            ans.push_back(p);
    }

    return ans;
}

vector<point> circleCircleIntersect(point c1, ld
↪ r1, point c2, ld r2) {
    // r1^2 - h^2 = x^2
    // r2^2 - h^2 = (d - x)^2 = x^2 - 2dx + d^2
    // d^2 - 2dx = r2^2 - r1^2

    ld d = (c2 - c1).len();

    if (d > r1 + r2 + eps || d < abs(r2 - r1) -
↪ eps || abs(d) < eps) return {};

    ld x = (d * d - r2 * r2 + r1 * r1) / (2 * d);
    point dir = (c2 - c1).norm();

    ld h = sqrt(max<ld>(r1 * r1 - x * x, 0));

    if (h < eps)
        return { c1 + dir * x };
    else
        return { c1 + dir * x + dir.rot() * h, c1
↪ + dir * x - dir.rot() * h };
}

```

5 graphs

5.1 components.cpp

```

struct Graph {
    void read() {
        int m;
        cin >> n >> m;

        e.resize(n);

        for (int i = 0; i < m; ++i) {
            int u, v;
            cin >> u >> v;
            --u; --v;
            e[u].push_back(v);
            e[v].push_back(u);
        }
    }

    /* COMMON PART */

```

```

int n;
vector<vector<int>> e;

int counter = 1;
vector<int> inTime, minInTime;

void dfs(int v, int p = -1) {
    minInTime[v] = inTime[v] = counter++;

    for (int u: e[v]) {
        if (u == p) continue;

        if (!inTime[u]) {
            dfs(u, v);
            minInTime[v] = min(minInTime[v],
↪ minInTime[u]);
        }
        else {
            minInTime[v] = min(minInTime[v],
↪ inTime[u]);
        }
    }

    vector<char> used;

    /* COMPONENTS SEPARATED BY BRIDGES (COLORING) */

    int nColors;
    vector<int> color;

    void colorDfs(int v, int curColor) {
        color[v] = curColor;

        for (int u: e[v]) {
            if (color[u] != -1) continue;

            colorDfs(u, minInTime[u] > inTime[v] ?
↪ nColors++ : curColor);
        }
    }

    void findVertexComponents() {
        inTime.assign(n, 0);
        minInTime.assign(n, 0);
        counter = 1;

        for (int i = 0; i < n; ++i)
            if (!inTime[i])
                dfs(i);

        nColors = 0;
        color.assign(n, -1);
        for (int i = 0; i < n; ++i)
            if (color[i] == -1) {
                colorDfs(i, nColors++);
            }
    }

    /* COMPONENTS SEPARATED BY JOINTS (EDGE
↪ COMPONENTS) */

    struct Edge {
        int u, v;
    };

    // Cactus loops can be parsed as .u of every edge
    vector<vector<Edge>> edgeComps;

    vector<int> colorStack;

    void edgeCompDfs(int v, int p = -1) {
        used[v] = true;

        for (int u: e[v]) {
            if (used[u]) {
                if (inTime[u] < inTime[v] && u != p) {
                    // NOTE: && u != p makes
↪ one-edge components contain exactly one edge;
                    // if you need them as
↪ two-edge loops, remove this part of if
↪ condition
                }
                edgeComps[colorStack.back()].push_back({v,
↪ u});
            }
            else {
                bool newComp = minInTime[u] >= inTime[v];

                if (newComp) {
                    colorStack.push_back(edgeComps.size());
                    edgeComps.emplace_back();
                }

                edgeCompDfs(u, v);

                if (newComp) {
                    colorStack.pop_back();
                }
            }
        }
    }

    void findEdgeComponents() {
        inTime.assign(n, 0);
        minInTime.assign(n, 0);
        counter = 1;

        for (int i = 0; i < n; ++i)
            if (!inTime[i])
                dfs(i);

        used.assign(n, false);
        colorStack.clear();
        edgeComps.clear();
        for (int i = 0; i < n; ++i)
            if (!used[i]) {
                assert(colorStack.empty());
                edgeCompDfs(i);
            }
    }
};

5.2 dominator_tree.h

struct DominatorTree {
    int n;
    int root;
    vector<int> tin, revin;
    vector<int> sdom, idom;
    vector<vector<int>> g, revg;
    vector<int> parent;

    vector<int> dsu;
    vector<int> min_v;
    int cnt = 0;

    int get(int v) {
        ++cnt;
        if (dsu[v] == v) {
            return v;
        }
        int next_v = get(dsu[v]);
        if (sdom[min_v[dsu[v]]] < sdom[min_v[v]]) {
            min_v[v] = min_v[dsu[v]];
        }
        dsu[v] = next_v;
        return next_v;
    }

    void merge(int from, int to) {
        dsu[from] = to;
    }

    DominatorTree(int n, int root): n(n),
↪ root(root), dsu(n) {
        tin.resize(n, -1);
        revin.resize(n, -1);
        sdom.resize(n);
        idom.resize(n);
    }
};

```

```

    g.resize(n);
    revg.resize(n);
    dsu.resize(n);
    parent.assign(n, -1);
    min_v.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        dsu[i] = i;
        min_v[i] = i;
        sdom[i] = i;
        idom[i] = i;
    }
}

void dfs(int v, vector<vector<int>>& cur_g,
    ↪ int& timer) {
    tin[v] = timer++;
    for (int to : cur_g[v]) {
        if (tin[to] == -1) {
            dfs(to, cur_g, timer);
            parent[tin[to]] = tin[v];
        }
        revg[tin[to]].push_back(tin[v]);
    }
}

vector<int> get_tree(vector<vector<int>> cur_g) {
    vector<char> used(n, false);
    int timer = 0;
    dfs(root, cur_g, timer);
    for (int i = 0; i < n; ++i) {
        if (tin[i] == -1) {
            continue;
        }
        revin[tin[i]] = i;
        for (int to : cur_g[i]) {
            g[tin[i]].push_back(tin[to]);
        }
    }

    vector<vector<int>> buckets(n);
    for (int i = n - 1; i >= 0; --i) {
        for (int to : revg[i]) {
            get(to);
            sdom[i] = min(sdom[i],
    ↪ sdom[min_v[to]]);
        }
        if (revin[i] == -1) {
            continue;
        }
        if (i) {
            buckets[sdom[i]].push_back(i);
        }
        for (int w : buckets[i]) {
            get(w);
            int v = min_v[w];
            if (sdom[v] == sdom[w]) {
                idom[w] = sdom[w];
            } else {
                idom[w] = v;
            }
        }
        for (int to : g[i]) {
            if (parent[to] == i) {
                merge(to, i);
            }
        }
    }
    for (int i = 0; i < n; ++i) {
        if (revin[i] == -1) {
            continue;
        }
        if (idom[i] == sdom[i]) {
            continue;
        } else {
            idom[i] = idom[idom[i]];
        }
    }

    vector<int> res(n, -1);
    for (int i = 0; i < n; ++i) {
        if (revin[i] == -1) {
            continue;
        }
    }
}

```

```

        res[revin[i]] = revin[idom[i]];
    }
    return res;
}
};

```

6 maths

6.1 berlekamp.h

```

vector<int> massey(vector<int> dp) {
    //dp.erase(dp.begin(), dp.begin() + 1);
    vector<int> C(1, 1);
    int L = 0;
    vector<int> B(1, 1);
    int b = 1;
    for (int n = 0; n < dp.size(); ++n) {
        int d = 0;
        for (int i = 0; i <= L; ++i) {
            d += C[i] * dp[n - i];
            d %= mod;
            if (d < 0) {
                d += mod;
            }
        }
        B.insert(B.begin(), 0);
        if (d == 0) {
            continue;
        }
        auto prevC = C;
        if (C.size() < B.size()) {
            C.resize(B.size(), 0);
        }
        int cur_mult = d * binpow(b, mod - 2) % mod;
        for (int i = 0; i < B.size(); ++i) {
            C[i] -= B[i] * cur_mult;
            C[i] %= mod;
            if (C[i] < 0) {
                C[i] += mod;
            }
        }
        if (2 * L <= n) {
            b = d;
            L = n - L + 1;
            B = prevC;
        }
    }
    return C;
}

```

6.2 crt.h

```

inline int inv(int a, int b) {
    return a == 1 ? 1 : b - 111 * inv(b % a, a) *
    ↪ b / a % b;
}

pair<int, int> euc(int a, int b) {
    // returns {x, y} s.t. ax + by = g
    int g = __gcd(a, b);
    a /= g, b /= g;
    int x = inv(a, b);
    int y = (1 - 111 * a * x) / b;

    return {x, y};
}

// be careful if the whole base is long long
pair<int, int> crt(const vector<int>& mods,
    ↪ vector<int>& rems) {
    int rem = 0, mod = 1;
    for (int i = 0; i < (int)mods.size(); ++i) {
        long long g = __gcd(mods[i], mod);
        if (rem % g != rems[i] % g) {
            return {-1, -1};
        }

        int k = euc(mod, mods[i]).first * 111 *
    ↪ (rems[i] - rem + mods[i]) % mods[i];
        rem += mod / g * k;
        mod = mod / g * mods[i];
    }
}

```

```

    return {rem, mod};
}

```

6.3 extended_euclidean.h

```

inline int inv(int a, int b) {
    return a == 1 ? 1 : b - 111 * inv(b % a, a) *
    ↪ b / a % b;
}

pair<int, int> euc(int a, int b) {
    // returns {x, y} s.t. ax + by = g
    int g = __gcd(a, b);
    a /= g, b /= g;
    int x = inv(a, b);
    int y = (1 - 111 * a * x) / b;

    return {x, y};
}

```

6.4 gauss_bitset_inverse.h

```

const int N = 100;
using Bs = bitset<N>;
using Matrix = vector<Bs>;

Matrix getInverse(Matrix a) {
    assert(!a.empty());
    int n = a.size();

    Matrix b(n);
    for (int i = 0; i < n; ++i) {
        b[i][i] = 1;
    }

    int row = 0;
    for (int col = 0; col < n; ++col) {
        if (!a[row][col]) {
            int i = row + 1;
            while (i < n && !a[i][col]) {
                ++i;
            }
            if (i == n) {
                return {}; // assert(false);
                ↪ throw PoshelNahuiException(); etc
            }
            swap(a[i], a[row]);
            swap(b[i], b[row]);
        }

        for (int i = row + 1; i < n; ++i) {
            if (a[i][col]) {
                a[i] ^= a[row];
                b[i] ^= b[row];
            }
        }

        ++row;
    }

    for (int i = n - 1; i >= 0; --i) {
        for (int j = 0; j < i; ++j) {
            if (a[j][i]) {
                a[j] ^= a[i];
                b[j] ^= b[i];
            }
        }
    }

    return b;
}

```

6.5 gauss_bitset_solve_sluh.h

```

const int N = 100;
using Bs = bitset<N>;
using Matrix = vector<Bs>;

Bs solveLinearSystem(Matrix a, Bs b) {
    // solves Av = b
    assert(!a.empty());
    int n = a.size();

```

```

    int row = 0;
    vector<int> cols(n);
    for (int col = 0; col < N; ++col) {
        if (row == n) {
            break;
        }
        if (!a[row][col]) {
            int i = row + 1;
            while (i < n && !a[i][col]) {
                ++i;
            }
            if (i == n) {
                continue;
            }
            swap(a[i], a[row]);
            b[i] = b[i] ^ b[row];
            b[row] = b[row] ^ b[i];
            b[i] = b[i] ^ b[row];
        }

        for (int i = row + 1; i < n; ++i) {
            if (a[i][col]) {
                a[i] ^= a[row];
                b[i] = b[i] ^ b[row];
            }
        }

        cols[row] = col;
        ++row;
    }

    for (int i = row; i < n; ++i) {
        if (b[i]) {
            return {}; // assert(false); throw
            ↪ PoshelNahuiException(); etc
        }
    }

    Bs result = {};
    while (row) {
        --row;
        for (int i = cols[row] + 1; i < N; ++i) {
            b[row] = b[row] ^ (a[row][i] * result[i]);
        }
        result[cols[row]] = b[row];
    }

    return result;
}

```

6.6 gauss_double_inverse.h

```

using Matrix = vector<vector<ld>>>;

const ld eps = 1e-6;

Matrix getInverse(Matrix a) {
    assert(!a.empty());
    int n = a.size();
    assert(n == (int)a[0].size());

    Matrix b(n, vector<ld>(n, 0));
    for (int i = 0; i < n; ++i) {
        b[i][i] = 1;
    }

    int row = 0;
    for (int col = 0; col < n; ++col) {
        if (abs(a[row][col]) < eps) {
            int i = row + 1;
            while (i < n && abs(a[i][col]) < eps) {
                ++i;
            }
            if (i == n) {
                return {}; // assert(false);
                ↪ throw PoshelNahuiException(); etc
            }
            a[i].swap(a[row]);
            b[i].swap(b[row]);
        }

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

```



```

        ld k = a[i][col] / a[row][col];
        for (int j = col; j < n; ++j) {
            a[i][j] -= k * a[row][j];
        }
        for (int j = 0; j < n; ++j) {
            b[i][j] -= k * b[row][j];
        }
    }

    ++row;
}

for (int i = n - 1; i >= 0; --i) {
    for (int j = 0; j < i; ++j) {
        ld k = a[j][i] / a[i][i];
        for (int l = 0; l < n; ++l) {
            a[j][l] -= a[i][l] * k;
            b[j][l] -= b[i][l] * k;
        }
    }
    ld k = a[i][i];
    for (int l = 0; l < n; ++l) {
        b[i][l] /= k;
    }
    a[i][i] /= k;
}

return b;
}

```

6.7 gauss_double_solve_slv.h

```

using Matrix = vector<vector<ld>>;

const ld eps = 1e-6;

vector<ld> solveLinearSystem(Matrix a, vector<ld> b) {
    // solves Av = b
    assert(!a.empty());
    int n = a.size(), m = a[0].size();
    assert(n == (int)b.size());

    int row = 0;
    vector<int> cols(n);
    for (int col = 0; col < m; ++col) {
        if (row == n) {
            break;
        }
        if (abs(a[row][col]) < eps) {
            int i = row + 1;
            while (i < n && abs(a[i][col]) < eps) {
                ++i;
            }
            if (i == n) {
                continue;
            }
            a[i].swap(a[row]);
            swap(b[i], b[row]);
        }

        for (int i = row + 1; i < n; ++i) {
            ld k = a[i][col] / a[row][col];
            for (int j = col; j < m; ++j) {
                a[i][j] -= k * a[row][j];
            }
            b[i] -= b[row] * k;
        }

        cols[row] = col;
        ++row;
    }

    for (int i = row; i < n; ++i) {
        if (abs(b[i]) < eps) {
            return {}; // assert(false); throw ←
        }
    }
    ← PoshelNahuiException(); etc

    vector<ld> result(m);
    while (row) {
        --row;
        for (int i = cols[row] + 1; i < m; ++i) {

```

```

            b[row] -= a[row][i] * result[i];
        }
        result[cols[row]] = b[row] / a[row][cols[row]];
    }

    return result;
}

```

7 misc

7.1 ch_trick_with_binary_summation_struct.cpp

```

const int INF = (int)1e6;

struct Line {
    int k;
    li b;
    bool operator < (const Line& ot) const {
        if (k != ot.k) {
            return k > ot.k;
        }
        return b < ot.b;
    }
    li eval(li x) {
        return k * 1LL * x + b;
    }
};

double get_intersect(Line& q, Line& w) {
    return (q.b - w.b) / 1.0 / (w.k - q.k);
}

struct Hull {
    vector<Line> lines;
    vector<double> borders;
    int Size = 0;
    void append(Line cur) {
        lines.push_back(cur);
    }
    void set_size(int val) {
        Size = val;
    }
    void build() {
        sort(all(lines));
        borders.clear();
        vector<Line> new_lines;
        for (auto& line : lines) {
            if (!new_lines.empty() && new_lines.back().k ←
→ == line.k) {
                continue;
            }
            while (new_lines.size() > 1 && ←
→ get_intersect(new_lines[new_lines.size() - 2], ←
→ new_lines.back()) > ←
→ get_intersect(new_lines.back(), line)) {
                new_lines.pop_back();
                borders.pop_back();
            }
            if (new_lines.empty()) {
                borders.push_back(-INF);
            } else {
                borders.push_back(get_intersect(new_lines.back(), ←
→ line));
            }
            new_lines.push_back(line);
        }
        new_lines.swap(lines);
    }
    int size() {
        return Size;
    }
    li get_min(li x) {
        int id = (int)(lower_bound(all(borders), ←
→ (double)x) - borders.begin());
        li res = (li)1e18;
        for (int i = max(id - 1, 0); i < min(id + 2, ←
→ (int)lines.size()); ++i) {
            res = min(res, lines[i].eval(x));
        }
        return res;
    }
}

```

```

};

struct Lupa {
    vector<Hull> hulls;
    int Size = 0;
    void append_line(Line cur) {
        hulls.push_back(Hull());
        hulls.back().append(cur);
        hulls.back().set_size(1);
        while (hulls.size() >= 2 &&
        hulls.back().size() == hulls[hulls.size() -
        2].size()) {
            for (auto& item : hulls.back().lines) {
                hulls[hulls.size() - 2].append(item);
            }
            hulls.pop_back();
            hulls.back().set_size(hulls.back().size() * 2);
        }
        hulls.back().build();
        ++Size;
    }
    li get_min(li x) {
        li res = (li)1e18;
        for (auto& vec : hulls) {
            res = min(res, vec.get_min(x));
        }
        return res;
    }
    int size() {
        return Size;
    }
    void merge_with(Lupa& ot) {
        for (auto& vec : ot.hulls) {
            for (auto& item : vec.lines) {
                append_line(item);
            }
            vec.lines.clear();
        }
    }
    void make_swap(Lupa& ot) {
        swap(ot.Size, Size);
        ot.hulls.swap(hulls);
    }
};

```

7.2 tree_bidirectional_dp.h

```

/* For any commutative function f({x, y, ..., z})
↳ = f(x, f(y, f(..., z)))
* like sum, min, max, or, xor, and, etc
* calculates in dp[i][j] f(subtree),
* where subtree is a connectivity component of G
↳ \ (i, a[i][j]) with vertex a[i][j]
*/

const int N = 222222;
vector<int> a[N];
vector<int> dp[N];
int par[N];

#define data asdf
int data[N];

inline int f(int x, int y) {
    return x | y;
}

int dfsDown(int v) {
    int res = data[v];
    for (int i = 0; i < (int)a[v].size(); ++i) {
        int to = a[v][i];
        if (to == par[v]) {
            continue;
        }
        par[to] = v;
        res = f(res, dp[v][i] = dfsDown(to));
    }
    return res;
}

void dfsUp(int v, int to_parent = 0) {
    vector<int> pref, suf;
    pref.reserve(a[v].size());

```

```

    suf.reserve(a[v].size());
    int j = 0;
    for (int i = 0; i < (int)a[v].size(); ++i) {
        int to = a[v][i];
        if (to == par[v]) {
            dp[v][i] = to_parent;
            continue;
        }
        pref.push_back(j ? f(pref[j - 1],
        dp[v][i]) : dp[v][i]);
        ++j;
    }
    j = 0;
    for (int i = (int)a[v].size() - 1; i >= 0; --i) {
        int to = a[v][i];
        if (to == par[v]) {
            continue;
        }
        suf.push_back(j ? f(dp[v][i], suf[j - 1])
        : dp[v][i]);
        ++j;
    }
    reverse(all(suf));

    j = 0;
    to_parent = f(to_parent, data[v]);
    for (int i = 0; i < (int)a[v].size(); ++i) {
        int to = a[v][i];
        if (to == par[v]) {
            continue;
        }
        int new_to_parent = to_parent;
        if (j > 0) {
            new_to_parent = f(pref[j - 1],
            new_to_parent);
        }
        if (j < (int)suf.size() - 1) {
            new_to_parent = f(new_to_parent, suf[j
            + 1]);
        }
        dfsUp(to, new_to_parent);
        ++j;
    }
}

```

8 strings

8.1 aho_corasick.h

```

const int ALPHABET = 26;

struct state {
    array<int, ALPHABET> transition = {};
    int link = 0;

    bool isTerminal = false;
};

struct automaton {
    vector<state> states = { state() };
    int numStates = 1;

    void addString(const string& s) {
        int cur = 0;
        for (char c: s) {
            c -= 'a';
            int& to = states[cur].transition[c];
            if (to) {
                cur = to;
            }
            else {
                cur = to = states.size();
                states.push_back(state());
            }
        }
        states[cur].isTerminal = true;
    }

    void build() {
        deque<int> q;
        q.push_back(0);
    }

```

```

    while (!q.empty()) {
        int v = q.front();
        q.pop_front();
        states[v].isTerminal =
    → states[v].isTerminal ||
    → states[states[v].link].isTerminal;

        for (int c = 0; c < ALPHABET; ++c) {
            if (int u = states[v].transition[c]) {
    → states[states[v].link].transition[c] : 0;
                q.push_back(u);
            }
            else {
                states[v].transition[c] =
    → states[states[v].link].transition[c];
            }
        }
    }
};

```

8.2 manacher.h

```

array<vector<int>, 2> manacher(const string& s) {
    int n = s.length();
    array<vector<int>, 2> res;
    for (auto& v : res) {
        v.assign(n, 0);
    }
    for (int z = 0, l = 0, r = 0; z < 2; ++z, l =
    → 0, r = 0) {
        for (int i = 0; i < n; ++i) {
            if (i < r) {
                res[z][i] = min(r - i + !z,
    → res[z][l + r - i + !z]);
            }
            int L = i - res[z][i], R = i +
    → res[z][i] - !z;
            while (L - 1 >= 0 && R + 1 < n && s[L
    → - 1] == s[R + 1]) {
                ++res[z][i];
                --L;
                ++R;
            }
            if (R > r) {
                l = L;
                r = R;
            }
        }
    }
    return res;
}

```

8.3 palindromes_on_subsegment.h

```

struct Node {
    int len;
    int link;
    vector<int> trans;
    bool all_equal;
    Node() {
        len = 0;
        link = 0;
        trans.assign(26, -1);
        all_equal = true;
    }
};

struct Eertree {
    vector<Node> nodes;
    vector<int> one_len;
    Eertree() {
        nodes.push_back(Node());
        one_len.assign(26, -1);
    }
    vector<int> feed_string(const string& s) {
        int v = 0;
        int n = s.length();
        vector<int> state(n);
        for (int i = 0; i < s.length(); ++i) {
            int c = s[i] - 'a';

```

```

        if (nodes[cur_v].trans[c] ←
→ != -1) {
            nodes[v].trans[c] = ←
→ nodes[cur_v].trans[c];
            break;
        }
        if (cur_v == 0) {
            nodes[v].trans[c] = 0;
            break;
        }
        cur_v = nodes[cur_v].link;
    }
}
};

struct Query {
    int l, r;
    int id;
    bool operator < (const Query& ot) const {
        if (r != ot.r) {
            return r < ot.r;
        }
        return l < ot.l;
    }
};

void solve(bool read) {
    string s;
    cin >> s;
    Eertree tree;
    tree.feed_string(s);
    tree.enclose();
    int Q;
    cin >> Q;
    int n = s.length();
    int block_size = max((int)(sqrt(n) * 1.5), 1);
    int blocks = (n - 1) / block_size + 1;
    for (int i = 0; i < Q; ++i) {
        Query cur;
        cin >> cur.l >> cur.r;
        --cur.l;
        cur.id = i;
        q[cur.l / block_size].push_back(cur);
    }
    vector<int> ans(Q);
    vector<int> used(tree.nodes.size(), 0);
    vector<int> left_used(tree.nodes.size(), 0);
    int TIMER = 0;
    int LEFT_TIMER = 0;
    for (int block = 0; block < blocks; ++block) {
        sort(all(q[block]));
        int right_border = min((block + 1) * ←
→ block_size, n);
        int uk = 0;
        while (uk < q[block].size() && ←
→ q[block][uk].r < right_border) {
            ++TIMER;
            int res = 0;
            int v = 0;
            for (int pos = q[block][uk].l; pos < ←
→ q[block][uk].r; ++pos) {
                v = tree.nodes[v].trans[s[pos] - 'a'];
                if (s[pos] != s[pos - ←
→ tree.nodes[v].len + 1]) {
                    v = tree.nodes[v].link;
                }
                if (tree.nodes[v].len > pos + 1 - ←
→ q[block][uk].l) {
                    v = tree.nodes[v].link;
                }
                if (used[v] != TIMER) {
                    ++res;
                    used[v] = TIMER;
                }
            }
            ans[q[block][uk].id] = res;
            ++uk;
        }
    }
}

int cur_r = right_border;
int overall_pals = 0;
int right_state = 0;
int left_state = 0;
++TIMER;
while (uk < q[block].size()) {
    while (cur_r < q[block][uk].r) {
        right_state = ←
→ tree.nodes[right_state].trans[s[cur_r] - 'a'];
        if (s[cur_r] != s[cur_r - ←
→ tree.nodes[right_state].len + 1]) {
            right_state = ←
→ tree.nodes[right_state].link;
        }
        if (tree.nodes[right_state].len > ←
→ cur_r + 1 - right_border) {
            right_state = ←
→ tree.nodes[right_state].link;
        }
        if (used[right_state] != TIMER) {
            ++overall_pals;
            used[right_state] = TIMER;
        }
        if (tree.nodes[right_state].len == ←
→ cur_r + 1 - right_border) {
            left_state = right_state;
        }
        ++cur_r;
    }
    ++LEFT_TIMER;
    int cur_l = right_border;
    int cur_left_state = left_state;
    int cur_res = overall_pals;
    while (cur_l > q[block][uk].l) {
        --cur_l;
        cur_left_state = ←
→ tree.nodes[cur_left_state].trans[s[cur_l] - ←
→ 'a'];
        if (s[cur_l] != s[cur_l + ←
→ tree.nodes[cur_left_state].len - 1]) {
            cur_left_state = ←
→ tree.nodes[cur_left_state].link;
        }
        if (tree.nodes[cur_left_state].len ←
→ > cur_r - cur_l) {
            cur_left_state = ←
→ tree.nodes[cur_left_state].link;
        }
        if (used[cur_left_state] != TIMER ←
→ && left_used[cur_left_state] != LEFT_TIMER) {
            ++cur_res;
            left_used[cur_left_state] = ←
→ LEFT_TIMER;
        }
    }
    ans[q[block][uk].id] = cur_res;
    ++uk;
}
}

for (int i = 0; i < Q; ++i) {
    cout << ans[i] << "\n";
}
}

8.4 prefix_function.h

void prefixFunction(const string& s, vector<int>& p) {
    if (s.length() == 0)
        return;
    p[0] = 0;
    for (size_t i = 1; i < s.length(); ++i) {
        int j = p[i - 1];
        while (j > 0 && s[i] != s[j])
            j = p[j - 1];
        if (s[i] == s[j])
            ++j;
        p[i] = j;
    }
}

const char first = 'a';
const int alphabet = 26;

```

```
// выходит из массива, после того, как совпадет
// все. можно добавить aut[n] = aut[pi[n - 1]]
// это эмулирует переход по суф ссылке
vector<vi> pfautomaton(const string& s) {
    vi p(s.length());
    prefixFunction(s, p);
    vector<vi> aut(s.length(), vi(alphabet));
    for (size_t i = 0; i < s.length(); ++i) {
        for (char c = 0; c < alphabet; ++c) {
            if (i > 0 && c != s[i] - first) {
                aut[i][c] = aut[p[i - 1]][c];
            }
            else {
                aut[i][c] = i + (c == s[i] - first);
            }
        }
    }
    return aut;
}
```

8.5 suffix_array.cpp

```
void Build(const string& init, vector<int>& lcp) {
    string s = init;
    s.push_back(char(0));
    vector<int> head;
    vector<int> color;
    vector<int> colorSub;
    vector<int> suffArraySub;
    head.assign(max((int)s.size(), 256), 0);
    suffArray.resize(s.size());
    color.resize(s.size());
    colorSub.resize(s.size());
    suffArraySub.resize(s.size());
    lcp.resize(s.size());

    for (int i = 0; i < s.size(); ++i) {
        ++head[s[i]];
    }
    for (int i = 1; i < 256; ++i) {
        head[i] += head[i - 1];
    }
    for (int i = 255; i > 0; --i) {
        head[i] = head[i - 1];
    }
    head[0] = 0;
    for (int i = 0; i < s.size(); ++i) {
        suffArray[head[s[i]]] = i;
        ++head[s[i]];
    }
    int numberOfClasses = 1;
    head[0] = 0;
    for (int i = 1; i < s.size(); ++i) {
        if (s[suffArray[i - 1]] != s[suffArray[i]]) {
            ++numberOfClasses;
            head[numberOfClasses - 1] = i;
        }
        color[suffArray[i]] = numberOfClasses - 1;
    }
    for (int k = 1; k < s.size(); k *= 2) {
        for (int i = 0; i < s.size(); ++i) {
            int firstPartBeginning = suffArray[i] - k;
            if (firstPartBeginning < 0) {
                firstPartBeginning += s.size();
            }
        }
        suffArraySub[head[color[firstPartBeginning]]]
        = firstPartBeginning;
        ++head[color[firstPartBeginning]];
    }
    suffArray = suffArraySub;

    int secondPartBeginning;
    pair<int, int> prevSuffClasses,
    curSuffClasses;
    curSuffClasses = make_pair(-1, 0);
    numberOfClasses = 0;

    for (int i = 0; i < s.size(); ++i) {
        prevSuffClasses = curSuffClasses;

        secondPartBeginning = suffArray[i] + k;

```

```
        if (secondPartBeginning >= s.size()) {
            secondPartBeginning -= s.size();
        }
        curSuffClasses =
        make_pair(color[suffArray[i]],
        color[secondPartBeginning]);

        if (curSuffClasses != prevSuffClasses) {
            ++numberOfClasses;
            head[numberOfClasses - 1] = i;
        }
        colorSub[suffArray[i]] =
        numberOfClasses - 1;

        color = colorSub;

        if (numberOfClasses == s.size())
            break;
    }
    vector<int> pos;
    int curLcp = 0;
    pos.resize(s.size());
    for (int i = 0; i < s.size(); ++i) {
        pos[suffArray[i]] = i;
    }
    lcp.resize(s.size());
    for (int i = 0; i < s.size(); ++i) {
        if (pos[i] == s.size() - 1) {
            lcp[pos[i]] = 0;
            curLcp = 0;
            continue;
        }
        while (s[(i + curLcp) % s.size()] ==
        s[(suffArray[pos[i] + 1] + curLcp) %
        s.size()]) {
            ++curLcp;
        }
        lcp[pos[i]] = curLcp;

        --curLcp;
        if (curLcp < 0)
            curLcp = 0;
    }

    void BuildSparseTable(const vector<int>& a,
    vector<vector<int>>& sparseTable) {
        int logSize = 0;
        while ((1 << logSize) < a.size()) {
            ++logSize;
        }
        logSize = 19;
        sparseTable.assign(a.size(), vector<int>
        (logSize + 1));

        for (int i = 0; i < a.size(); ++i) {
            sparseTable[i][0] = a[i];
        }

        for (int k = 1; k <= logSize; ++k) {
            for (int i = 0; i + (1 << k) <= a.size();
            ++i) {
                sparseTable[i][k] =
                min(sparseTable[i][k - 1], sparseTable[i + (1
                << (k - 1))][k - 1]);
            }
        }

    int GetMin(int l, int r, const vector<vector<int>>& sparseTable) {
        assert(l < r);
        int sz = 31 - __builtin_clz(r - l);
        return min(sparseTable[l][sz], sparseTable[r -
        (1 << sz)][sz]);
    }

    void solve(__attribute__((unused)) bool read) {
        string s;
        cin >> s;
        int n = s.length();

```

```

vector<int> suff_array, lcp;
Build(s, suff_array, lcp);
suff_array.erase(suff_array.begin());
lcp.erase(lcp.begin());
vector<int> pos_in_array(n);
for (int i = 0; i < suff_array.size(); ++i) {
    pos_in_array[suff_array[i]] = i;
}
vector<vector<int>> sparse;
BuildSparseTable(lcp, sparse);
}

```

8.6 suffix_automaton_kostroma.h

```

const int UNDEFINED_VALUE = -1;

class SuffixAutomaton {
public:
    struct State {
        map<char, int> transitions;
        int link;
        int maxlen;
        int firstPos, lastPos;
        int cnt;
        State():link(UNDEFINED_VALUE),
        firstPos(UNDEFINED_VALUE),
        lastPos(UNDEFINED_VALUE), maxlen(0), cnt(0) {}
    };
    vector<State> states;
    int lastState;
    SuffixAutomaton(const string& s) {
        states.push_back(State());
        lastState = 0;
        for (int i = 0; i < s.length(); ++i)
            append(s[i]);
        vector<pair<int, int>> p(states.size());
        for (int i = 0; i < p.size(); ++i) {
            p[i].second = i;
            p[i].first = states[i].maxLen;
        }
        sort(all(p));
        reverse(all(p));
        for (int i = 0; i < p.size(); ++i) {
            int curState = p[i].second;
            if (states[curState].lastPos ==
                UNDEFINED_VALUE)
                states[curState].lastPos =
                states[curState].firstPos;
            if (states[curState].link !=
                UNDEFINED_VALUE) {
                states[states[curState].link].lastPos =
                max(states[states[curState].link].lastPos,
                states[curState].lastPos);
                states[states[curState].link].cnt
                += states[curState].cnt;
            }
        }
    }

private:
    void append(char c) {
        int curState = states.size();
        states.push_back(State());
        states[curState].maxLen =
        states[lastState].maxLen + 1;
        states[curState].firstPos =
        states[lastState].maxLen;
        states[curState].cnt = 1;
        int prevState = lastState;
        for (; prevState != UNDEFINED_VALUE;
        prevState = states[prevState].link) {
            if (states[prevState].transitions.count(c))
                break;
            states[prevState].transitions[c] =
            curState;
        }

        if (prevState == UNDEFINED_VALUE) {
            states[curState].link = 0;
        }
        else {

```

```

            int nextState =
            states[prevState].transitions[c];
            if (states[nextState].maxLen ==
                states[prevState].maxLen + 1) {
                states[curState].link = nextState;
            }
            else {
                int cloneState = states.size();
                states.push_back(State());
                states[cloneState].maxLen =
                states[prevState].maxLen + 1;
                states[cloneState].link =
                states[nextState].link;
                states[cloneState].firstPos =
                states[nextState].firstPos;
                states[curState].link =
                states[nextState].link = cloneState;

                states[cloneState].transitions =
                states[nextState].transitions;
                for (; prevState !=
                UNDEFINED_VALUE &&
                states[prevState].transitions[c] == nextState;
                prevState = states[prevState].link)
                    states[prevState].transitions[c]
                    = cloneState;
            }
            lastState = curState;
        }
    };
};

```

8.7 z_function.h

```

vector<int> zFunction(const string& s) {
    int n = s.length();

    vector<int> z(n);
    int l = 0, r = 0;
    for (int i = 1; i < n; ++i) {
        z[i] = max(min(z[i - l], r - i), 0);

        while (i + z[i] < n && s[i + z[i]] == s[z[i]])
            ++z[i];

        if (i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }

    if (n)
        z[0] = n;

    return z;
}

```

9 templates

9.1 main.cpp

```

#undef NDEBUG
#define AIM
void print_stats(long long delta_time);
extern "C" int usleep(unsigned int usec);
#include "E.h"

#include <malloc.h>

void print_stats(long long delta_time) {
    cerr << "\n time: " << delta_time / 1.0 /
    CLOCKS_PER_SEC;

    extern char *__progname;
    system((string("size ") + __progname +
    "> /tmp/size.txt").c_str());
    ifstream in("/tmp/size.txt");
    string s;

    getline(in, s);
    int sz = 0;

```



```

    if (in >> sz >> sz >> sz >> sz) {
        cerr << ", static memory: " << sz /
        ↪ (double)(1024 * 1024) << " MB";
    }

    cerr << endl << endl;
}

// -D_GLIBCXX_DEBUG

```

9.2 sync-template.txt

```

// Executable: sed
// Arguments: -s 's/#include "'.*"/#include
↪ "$FileName$"/' main.cpp
// Working directory: $ProjectFileDir$
// ! Synchronize files after execution
// ! Open console for tool output

```

9.3 template.h

```

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

#define all(a) a.begin(), a.end()
typedef long long li;
typedef long double ld;
void solve(__attribute__((unused)) bool);
void precalc();
clock_t start;

int main() {
#ifdef CRYPTO
    freopen("/PATH/input.txt", "r", stdin);
#endif
    start = clock();
    int t = 1;
#ifdef CRYPTO
    cout.sync_with_stdio(0);
    cin.tie(0);
#endif
    precalc();
    cout.precision(20);
    cout << fixed;
    // cin >> t;
    int testNum = 1;
    while (t--) {
        //cout << "Case #" << testNum++ << ": ";
        solve(true);
    }
    cout.flush();
#ifdef CRYPTO1
    while (true) {
        solve(false);
    }
#endif
#ifdef CRYPTO
    cout.flush();
    auto end = clock();

    usleep(10000);
    print_stats(end - start);
    usleep(10000);
#endif

    return 0;
}

void precalc() {
}

template<typename T>
inline T nxt() {
    T cur;
    cin >> cur;
    return cur;
}

// #define int li
// const int mod = 1000000007;

```

```

void solve(__attribute__((unused)) bool read) {
}

```

10 treap

10.1 treap_explicit_keys.h

```

class Treap {
public:
    typedef struct _node {
        int key;
        int cnt;
        int prior;
        int val;
        _node* l;
        _node* r;
        _node(int key, int val) :key(key),
        ↪ val(val), l(nullptr), r(nullptr), cnt(1) {
        ↪ prior = rand(); }

        void push() {
        }

        void recalc() {
            cnt = 1 + Cnt(l) + Cnt(r);
        }

        static int Cnt(_node* v) {
            if (!v)
                return 0;
            return v->cnt;
        }
    }*node;

    static int Cnt(node v) {
        if (!v)
            return 0;
        return v->cnt;
    }

    node root;

    size_t Size;

    node merge(node l, node r) {
        if (!l)
            return r;
        if (!r)
            return l;
        if (l->prior < r->prior) {
            l->push();
            l->r = merge(l->r, r);
            l->recalc();
            return l;
        }
        else {
            r->push();
            r->l = merge(l, r->l);
            r->recalc();
            return r;
        }
    }

    void split(node v, int key, node& l, node& r) {
        l = r = nullptr;
        if (!v)
            return;
        v->push();
        if (v->key < key) {
            l = v;
            split(l->r, key, l->r, r);
            l->recalc();
        }
        else {
            r = v;
            split(r->l, key, l, r->l);
            r->recalc();
        }
    }
}

```

```

public:
    Treap() {
        root = nullptr;
        Size = 0;
    }

    size_t size() const {
        return Size;
    }

    node get_min() const {
        node v = root;
        if (!v) {
            throw runtime_error("Treap is empty");
        }
        while (v->l) {
            v = v->l;
        }
        return v;
    }

    node get_max() const {
        node v = root;
        if (!v) {
            throw runtime_error("Treap is empty");
        }
        while (v->r) {
            v = v->r;
        }
        return v;
    }

    void insert(int key, int val) {
        node l = nullptr, r = nullptr;
        split(root, key, l, r);
        node cur_node = new _node(key, val);
        root = merge(merge(l, cur_node), r);
        ++Size;
    }

    node operator [] (int key) {
        node l = nullptr, m = nullptr, r = nullptr;
        split(root, key, l, r);
        split(r, key + 1, m, r);
        if (m == nullptr) {
            throw
↪ runtime_error("IndexTreapOutOfBounds");
        }
        root = merge(merge(l, m), r);
        return m;
    }
};

typedef Treap::node Node;

```

10.2 treap_implicit_keys.h

```

class Treap {
public:
    typedef struct _node {
        int cnt;
        int prior;
        int val;
        _node* l;
        _node* r;
        _node* p;
        _node(int val) : val(val), l(nullptr),
↪ r(nullptr), cnt(1), p(nullptr) { prior =
↪ rand(); }

        void push() {

        }

        void recalc() {
            cnt = 1 + Cnt(l) + Cnt(r);
            if (l) {
                l->p = this;
            }
            if (r) {
                r->p = this;
            }
            p = nullptr;
        }
    };

```

```

    }

    static int Cnt(_node* v) {
        if (!v)
            return 0;
        return v->cnt;
    }
}*node;

    static int Cnt(node v) {
        if (!v)
            return 0;
        return v->cnt;
    }

    node root;

    size_t Size;

    node merge(node l, node r) {
        if (!l)
            return r;
        if (!r)
            return l;
        if (l->prior < r->prior) {
            l->push();
            l->r = merge(l->r, r);
            l->recalc();
            return l;
        }
        else {
            r->push();
            r->l = merge(l, r->l);
            r->recalc();
            return r;
        }
    }

    void split(node v, int idx, node& l, node& r) {
        l = r = nullptr;
        if (!v)
            return;
        v->push();
        if (Cnt(v->l) < idx) {
            l = v;
            split(l->r, idx - Cnt(v->l) - 1, l->r, r);
            l->recalc();
        }
        else {
            r = v;
            split(r->l, idx, l, r->l);
            r->recalc();
        }
    }
}

```

```

public:
    Treap() {
        root = nullptr;
        Size = 0;
    }

    size_t size() const {
        return Size;
    }

    void insert(int idx, int val) {
        node l = nullptr, r = nullptr;
        split(root, idx, l, r);
        node cur_node = new _node(val);
        root = merge(merge(l, cur_node), r);
        ++Size;
    }

    void erase(int idx) {
        node l = nullptr, m = nullptr, r = nullptr;
        split(root, idx, l, r);
        split(r, 1, m, r);
        root = merge(l, r);
        --Size;
    }

    int get_index(node v) {
        if (!v) {

```

```

        throw
    runtime_error("No such node in the treap");
    }
    int res = Cnt(v->l);
    while (v->p) {
        if (v->p->r == v) {
            res += Cnt(v->p->l) + 1;
        }
        v = v->p;
    }
    return res;
}

void push_back(int val) {
    return insert(Size, val);
}
void push_front(int val) {
    return insert(0, val);
}

node operator [] (int idx) {
    node l = nullptr, m = nullptr, r = nullptr;
    split(root, idx, l, r);
    split(r, 1, m, r);
    if (m == nullptr) {
        throw
    runtime_error("IndexTreapOutOfBound");
    }
    root = merge(merge(l, m), r);
    return m;
}
};

typedef Treap::node Node;

```

11 fuckups.tex

- Всегда выводим ответ на запрос!

Неправильно:

```

while (q--) {
    int u, v;
    cin >> u >> v;
    --u, --v;
    if (!dsu.merge(u, v)) {
        // ниче ж не поменялось)))))) можно
        // сделать continue))))))
        continue;
    }
    make_some_logic(u, v);
    cout << get_cur_ans() << "\n";
}

```

Правильно:

```

while (q--) {
    int u, v;
    cin >> u >> v;
    --u, --v;
    if (dsu.merge(u, v)) {
        make_some_logic(u, v);
    }
    cout << get_cur_ans() << "\n";
}

```

- m рёбер, а не n.

Неправильно:

```

int n, m;
cin >> n >> m;
vector<vector<int>> a(n);
for (int i = 0; i < n; ++i) {
    int u, v;
    cin >> u >> v;
    --u, --v;
    a[u].push_back(v);
    a[v].push_back(u);
}

```

Правильно:

```

int n, m;
cin >> n >> m;
vector<vector<int>> a(n);
for (int i = 0; i < m; ++i) {
    int u, v;
    cin >> u >> v;
    --u, --v;
    a[u].push_back(v);
    a[v].push_back(u);
}

```

- Не забываем построить дерево отрезков после инициализации листьев.

Неправильно:

```

for (int i = 0; i < n; ++i) {
    tree.set(i, a[i]);
}
for (int i = 0; i < Q; ++i) {
    int pos, val;
    cin >> pos >> val;
    tree.update(pos, val);
}

```

Правильно:

```

for (int i = 0; i < n; ++i) {
    tree.set(i, a[i]);
}
tree.build();
for (int i = 0; i < Q; ++i) {
    int pos, val;
    cin >> pos >> val;
    tree.update(pos, val);
}

```

- Лучше struct с понятными названиями полей, а не std::pair.

Неправильно:

```

set<pair<int, int>> a;
for (int i = 0; i < n; ++i) {
    int pos, val;
    cin >> pos >> val;
    a.insert({pos, val});
}
sort(all(a));

```

```

int q;
cin >> q;
while (q--) {
    int pos, val;
    cin >> pos >> val;
    auto it = a.lower_bound({pos, 0});
    if (it != a.end() && it->first > val) { //
        // эээ ну в сете же по first сортируем в 1ю
        // очередь
        cout << "YES\n";
    } else {
        cout << "NO\n";
    }
}

```

Правильно:

```

struct Shit {
    int pos;
    int val;

    bool operator <(const Shit& ot) const {
        return make_pair(pos, val) <
        make_pair(ot.pos, ot.val);
    }
}

set<Shit> a;
for (int i = 0; i < n; ++i) {
    int pos, val;
    cin >> pos >> val;
    a.insert({pos, val});
}
sort(all(a));

int q;

```

```

cin >> q;
while (q-->0) {
    int pos, val;
    cin >> pos >> val;
    auto it = a.lower_bound({pos, 0});
    if (it != a.end() && it->val > val) { // ←
        хуй проебёшься
        cout << "YES\n";
    } else {
        cout << "NO\n";
    }
}

```

- Перенумерация в эйлеровом обходе.

Неправильно:

```

for (int i = 0; i < n; ++i) {
    tree.update(i, 1);
}
for (int i = 0; i < n; ++i) {
    cout << tree.get_val(i) << endl;
}

```

Правильно:

```

for (int i = 0; i < n; ++i) {
    tree.update(tin[i], 1);
}
for (int i = 0; i < n; ++i) {
    cout << tree.get_val(tin[i]) << endl;
}

```

- `vector<char>` хранит числа до 255.

Неправильно:

```

vector<char> used(n), num_comp(n);
int cur = 0;
for (int i = 0; i < n; ++i) {
    if (!used[i]) {
        dfs(i, cur++);
    }
}

```

Правильно:

```

vector<char> used(n);
vector<int> num_comp(n);
int cur = 0;
for (int i = 0; i < n; ++i) {
    if (!used[i]) {
        dfs(i, cur++);
    }
}

```

- `bool f()` возвращает `bool`.

Неправильно:

```

bool occurs(const string& s, const string& t) {
    for (int i = 0; i + (int)s.length() <= (int)t.length(); ++i) { ←
        // падажи ебана
        // если содержится, то нужен индекс
        if (t.substr(i, s.length()) == s) {
            return i;
        }
    }
    // иначе пускай будет -1
    return -1;
}

```

Правильно:

```

int occurs(const string& s, const string& t) {
    ...
}

```

- Индексы в `dsu` до `n`, а не до `num_comps`.
- В `merge` для вершин дерева отрезков `push_val = UNDEFINED`.

Неправильно:

```

Node merge(const Node& q, const Node& w) {
    Node res; // или res = q
    res.min = min(q.min, w.min); // или if ←
    (w.min < res.min) res = w
    return res;
}

```

Правильно:

```

Node merge(const Node& q, const Node& w) {
    Node res;
    res.push_add = 0; // или в объявлении res ←
    = {}, если в конструкторе по умолчанию ←
    прописано заполнение
    res.min = min(q.min, w.min);
    return res;
}

```

- Считываем размеры в нужном порядке

Неправильно:

```

int n, m;
cin >> n >> m; // w, h
vector<vector<int>>> a(n, vector<int>(m, 0));
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) {
        cin >> a[i][j];
    }
}

```

Правильно:

```

int n, m;
cin >> m >> n; // w, h
vector<vector<int>>> a(n, vector<int>(m, 0));
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) {
        cin >> a[i][j];
    }
}

```