

Тимбук

Moscow State University: Meanwhile, Elk, MrF & YaV (Grigorii Bazilevich, Ilya Denisev, Vladimir Yakunin)

19 ноября 2022 г.

Содержание

1	Геометрия	1
1.1	Базовая геом	1
1.2	Выпуклая оболочка	4
2	Графы	4
2.1	HLD	4
2.2	Конденсация	5
2.3	Кун	5
2.4	Мосты	5
2.5	Точки сочленения	5
2.6	Центроиды	6
3	Контекст	6
3.1	CMakeLists	6
3.2	Стресс-тесты	6
3.3	Шаблон	6
4	Строки	7
4.1	Z-функция	7
4.2	Ахо-Корасик	7
4.3	Манакер	9
4.4	Префикс-функция	9
4.5	Суффиксный автомат	9
4.6	Суффиксный массив	9
5	Структуры данных	10
5.1	Sparse table	10
5.2	ДО с массивами	10

5.3	ДО снизу	11
5.4	Дерево Фенвика	11
5.5	СНМ	11
5.6	Хеш-таблица	11
6	Теория Чисел	12
6.1	КТО	12
6.2	Метод Гаусса	12
6.3	Ро-алгоритм Полларда	12

1 Геометрия

1.1 Базовая геом

```
const ld EPS = 1e-10;

int sgn(ld val) {
    return (0 < val) - (val < 0);
}

struct Point {
    ld x;
    ld y;
    ld absv() const;
    Point norm() const;
    Point rotate(ld a) const;
};

Point operator+(Point a, Point b) {
```

```
    return {a.x + b.x, a.y + b.y};
}

Point operator-(Point a, Point b) {
    return {a.x - b.x, a.y - b.y};
}

Point operator*(Point a, ld k) {
    return {a.x * k, a.y * k};
}

Point operator*(ld k, Point a) {
    return {a.x * k, a.y * k};
}

Point operator/(Point a, ld k) {
    return {a.x / k, a.y / k};
}

ld operator*(Point a, Point b) {
    return a.x * b.x + a.y * b.y;
}

ld operator%(Point a, Point b) {
    return a.x * b.y - a.y * b.x;
}

bool operator==(Point a, Point b) {
    return a.x == b.x && a.y == b.y;
}

ld angle(Point a, Point b) {
```

```

    return atan2(a % b, a * b);
}

ld Point::absv() const {
    return sqrt((*this) * (*this));
}

Point Point::norm() const {
    return (*this) / this->absv();
}

Point Point::rotate(ld a) const {
    return {
        cos(a) * x - sin(a) * y,
        sin(a) * x + cos(a) * y
    };
}

ld distance(Point p1, Point p2) {
    return (p2 - p1).absv();
}

struct Line {
    Point n{};
    Point p{};
    Line(Point a, Point b) {
        p = a;
        n = (b - a).norm();
    }
    Line(ld a, ld b, ld c) {
        n = Point{-b, a}.norm();
        if (b == 0) {
            p = {-c / a, 0};
        } else {
            p = {0, -c / b};
        }
    }
    bool contains(Point a) const;
    int halfPlane(Point p1) const;
};

Point perp(Point p, Line l) {
    Point v = l.p - p;
    return v - (l.n * v) * l.n;
}

```

```

}

bool Line::contains(Point a) const {
    return abs((p - a) % n) < EPS;
}

bool operator||(Line a, Line b) {
    return abs(a.n % b.n) < EPS;
}

bool operator==(Line a, Line b) {
    return a.contains(b.p) && (a || b);
}

pair<Point, int> operator^(Line a, Line b) {
    if (a == b) {
        return {a.p, 2};
    } else if (a || b) {
        return {a.p, 0};
    } else {
        ld k = (a.n % b.n);
        ld x = (b.n.x * (a.n % a.p) - a.n.x * (b.n
            % b.p)) / k;
        ld y = (b.n.y * (a.n % a.p) - a.n.y * (b.n
            % b.p)) / k;
        return {{x, y}, 1};
    }
}

int Line::halfPlane(Point p1) const {
    return sgn(n % (p1 - p));
}

ld distance(Point p, Line l) {
    return perp(p, l).absv();
}

struct Segment {
    Point a;
    Point b;
    Line l;
    Segment(Point p1, Point p2): a{p1}, b{p2},
        l{p1, p2} {}
    bool contains(Point p) const;
};

```

```

};

bool Segment::contains(Point p) const {
    return l.contains(p) && (a - p) * (b - p) <=
        EPS;
}

ld distance(Point p, Segment s) {
    Point pe = perp(p, s.l);
    if (s.contains(pe + p)) {
        return pe.absv();
    } else {
        return min((s.a - p).absv(), (s.b -
            p).absv());
    }
}

struct Ray {
    Point a;
    Point b;
    Line l;
    Ray(Point p1, Point p2): a{p1}, b{p2}, l{p1,
        p2} {}
    bool contains(Point p) const;
};

bool Ray::contains(Point p) const {
    return l.contains(p) && ((a - p) * (b - p) <=
        EPS || (a - b) * (p - b) <= EPS);
}

ld distance(Point p, Ray r) {
    Point pe = perp(p, r.l);
    if (r.contains(pe + p)) {
        return pe.absv();
    } else {
        return (r.a - p).absv();
    }
}

ld distance(Line l1, Line l2) {
    int res = (l1 ^ l2).second;
    if (res == 1 || res == 2) {
        return 0;
    }
}

```

```

    } else {
        return distance(l1.p, l2);
    }
}

ld distance(Ray r, Line l) {
    auto inter = r.l ^ l;
    if (inter.second == 2 ||
        (inter.second == 1 &&
         r.contains(inter.first))) {
        return 0;
    } else {
        return distance(r.a, l);
    }
}

ld distance(Segment s, Line l) {
    auto inter = s.l ^ l;
    if (inter.second == 2 ||
        (inter.second == 1 &&
         s.contains(inter.first))) {
        return 0;
    } else {
        return min(distance(s.a, l), distance(s.b,
        l));
    }
}

ld distance(Ray r1, Ray r2) {
    auto inter = r1.l ^ r2.l;
    if (inter.second == 2) {
        if (r1.contains(r2.a) ||
            r2.contains(r1.a)) {
            return 0;
        } else {
            return distance(r1.a, r2.a);
        }
    } else if (inter.second == 1 &&
        r1.contains(inter.first) &&
        r2.contains(inter.first)) {
        return 0;
    } else {
        return min(distance(r1.a, r2),
            distance(r2.a, r1));
    }
}

```

```

    }
}

ld distance(Segment s, Ray r) {
    auto inter = s.l ^ r.l;
    if (inter.second == 2) {
        if (r.contains(s.a) || r.contains(s.b)) {
            return 0;
        } else {
            return min(distance(s.a, r.a),
                distance(s.b, r.a));
        }
    } else if (inter.second == 1 &&
        s.contains(inter.first) &&
        r.contains(inter.first)) {
        return 0;
    } else {
        return min(min(distance(s.a, r),
            distance(s.b, r)), distance(r.a, s));
    }
}

ld distance(Segment s1, Segment s2) {
    auto inter = s1.l ^ s2.l;
    if (inter.second == 2) {
        if (s1.contains(s2.a) || s1.contains(s2.b)
            || s2.contains(s1.a) ||
            s2.contains(s1.b)) {
            return 0;
        } else {
            return min(distance(s1.a, s2),
                distance(s1.b, s2));
        }
    } else if (inter.second == 1 &&
        s1.contains(inter.first) &&
        s2.contains(inter.first)) {
        return 0;
    } else {
        return min(min(distance(s1.a, s2),
            distance(s1.b, s2)),
            min(distance(s2.a, s1), distance(s2.b,
            s1)));
    }
}

```

```

struct Triangle {
    Point a;
    Point b;
    Point c;
    bool contains(Point p) const;
};

bool Triangle::contains(Point p) const {
    ld p1 = (b - a) % (p - a);
    ld p2 = (c - b) % (p - b);
    ld p3 = (a - c) % (p - c);
    return p1 >= -EPS && p2 >= -EPS && p3 >= -EPS;
}

struct ConvexPolygon {
    vector<Point> vs;
    vector<ld> angles;
    ConvexPolygon(vector<Point> &av): vs{av} {
        angles.resize(vs.size());
        angles[0] = angles[1] = 0;
        Point base = vs[1] - vs[0];
        for (int i = 2; i < vs.size(); ++i) {
            angles[i] = angle(base, vs[i] - vs[0]);
        }
    }
    bool contains(Point p) const;
};

bool ConvexPolygon::contains(Point p) const {
    ld pa = angle(vs[1] - vs[0], p - vs[0]);
    auto it = upper_bound(angles.begin(),
        angles.end(), pa);
    if (it == angles.begin()) return false;
    if (it == angles.end()) {
        return Segment{vs[0],
            *(vs.end()-1)}.contains(p);
    }
    int ind = it - angles.begin();
    return Triangle{vs[0], vs[ind-1],
        vs[ind]}.contains(p);
}

struct Circle {

```

```

    Point o;
    ld r;
    bool contains(Point p) const;
};

bool Circle::contains(Point p) const {
    return distance(p, o) <= r+EPS;
}

bool operator==(Circle c1, Circle c2) {
    return c1.o == c2.o && c1.r == c2.r;
}

pair<pair<Point, Point>, int> operator^(Line l,
    Circle c) {
    ld rho = distance(c.o, l);
    if (rho > c.r+EPS) {
        return {{c.o, c.o}, 0};
    } else if (abs(rho - c.r) < EPS) {
        return {{c.o + perp(c.o, l), c.o +
            perp(c.o, l)}, 1};
    } else {
        ld d = sqrt(c.r * c.r - rho * rho);
        Point p = c.o + perp(c.o, l);
        return {{p + l.n * d, p - l.n * d}, 2};
    }
}

pair<pair<Point, Point>, int> operator^(Circle
    c1, Circle c2) {
    ld rho = distance(c1.o, c2.o);
    if (c1 == c2) {
        return {{c1.o, c1.o}, 3};
    } else if (c1.contains(c2.o)) {
        if (rho - c1.r + c2.r < EPS) {
            return {{c1.o, c1.o}, 0};
        } else if (abs(rho - c1.r + c2.r) < EPS) {
            Point p = c1.o + (c2.o - c1.o).norm()
                * c1.r;
            return {{p, p}, 1};
        } else {
            ld a = acos((c1.r * c1.r + rho * rho -
                c2.r * c2.r)/(2 * c1.r * rho));
            Point p = (c2.o - c1.o).norm() * c1.r;

```

```

        return {{c1.o + p.rotate(a), c1.o +
            p.rotate(-a)}, 2};
        }
    } else if (c2.contains(c1.o)) {
        if (rho - c2.r + c1.r < EPS) {
            return {{c2.o, c2.o}, 0};
        } else if (abs(rho - c2.r + c1.r) < EPS) {
            Point p = c2.o + (c1.o - c2.o).norm()
                * c2.r;
            return {{p, p}, 1};
        } else {
            ld a = acos((c2.r * c2.r + rho * rho -
                c1.r * c1.r)/(2 * c2.r * rho));
            Point p = (c1.o - c2.o).norm() * c2.r;
            return {{c2.o + p.rotate(a), c2.o +
                p.rotate(-a)}, 2};
        }
    } else {
        if (c1.r + c2.r - rho < EPS) {
            return {{c1.o, c1.o}, 0};
        } else if (abs(c1.r + c2.r - rho) < EPS) {
            Point p = c1.o + (c2.o - c1.o).norm()
                * c1.r;
            return {{p, p}, 1};
        } else {
            ld a = acos((c1.r * c1.r + rho * rho -
                c2.r * c2.r)/(2 * c1.r * rho));
            Point p = (c2.o - c1.o).norm() * c1.r;
            return {{c1.o + p.rotate(a), c1.o +
                p.rotate(-a)}, 2};
        }
    }
}

pair<pair<Point, Point>, int> tangents(Point p,
    Circle c) {
    ld rho = distance(p, c.o);
    if (c.contains(p)) {
        if (abs(rho - c.r) < EPS) {
            return {{p, p}, 1};
        } else {
            return {{p, p}, 0};
        }
    } else {

```

```

        ld d = sqrt(rho * rho - c.r * c.r);
        return Circle {p, d} ^ c;
    }
}

```

1.2 Выпуклая оболочка

```

vector<V> graham(vector<V> points) {
    V p0 = points[0];
    for (V p : points)
        if (p.x < p0.x || (p.x == p0.x && p.y < p0.y))
            p0 = p;
    sort_by_angle(points, p0);
    vector<V> hull;
    for (V p : points) {
        while (hull.size() >= 2 &&
            ((p - hull.back()) ^ (hull[hull.size() -
                2] - hull.back())) <= 0)
            hull.pop_back();
        hull.push_back(p);
    }
    return hull;
}

```

2 Графы

2.1 HLD

```

//
// Heavy-Light Decomposition
// Позволяет выполнять запросы на путях в дереве.
// Источник:
// https://codeforces.com/blog/entry/53170
//

struct HLD {
    int n, t = 0;
    vector<vector<int>> g;
    vector<int> tin, siz, par, h, up;

```

```

SegmentTree st;
HLD(int n, const vector<vector<int>> &g)
    : n(n), g(g), tin(n), siz(n, 1), par(n),
      h(n), up(n), st(n) {
    par[0] = -1, h[0] = 0, up[0] = 0;
    dfs_sz(0);
    dfs_up(0);
}

void dfs_sz(int v) {
    if (par[v] != -1) g[v].erase(find(all(g[v]),
        par[v]));
    for (int &u : g[v]) {
        par[u] = v, h[u] = h[v] + 1;
        dfs_sz(u);
        siz[v] += siz[u];
        if (siz[u] > siz[g[v][0]]) swap(u, g[v][0]);
    }
}

void dfs_up(int v) {
    tin[v] = t++;
    for (int u : g[v]) {
        if (u == g[v][0])
            up[u] = up[v];
        else
            up[u] = u;
        dfs_up(u);
    }
}

int get_path(int v, int u) { // запрос на пути
    // от v до u
    int res = 0;
    for (; up[v] != up[u]; v = par[up[v]]) {
        if (h[up[v]] < h[up[u]]) swap(v, u);
        res += st.get(tin[up[v]], tin[v] + 1);
        // если значения на рёбрах, прибавить к левой
        // границе +1
    }
    if (h[v] > h[u]) swap(v, u);
    res += st.get(tin[v], tin[u] + 1);
    return res;
}

int get_subtree(int v) { return st.get(tin[v],
    tin[v] + siz[v]); }
};

```

2.2 Конденсация

```

vector<vector<int>> adj, adj_rev;
vector<bool> used;
vector<int> order, component;
void dfs1(int v) {
    used[v] = true;
    for (auto u : adj[v])
        if (!used[u])
            dfs1(u);
    order.push_back(v);
}

void dfs2(int v) {
    used[v] = true;
    component.push_back(v);
    for (auto u : adj_rev[v])
        if (!used[u])
            dfs2(u);
}

int main() {
    int n;
    // ... read n ...
    for (;;) {
        int a, b;
        // ... read next directed edge (a,b) ...
        adj[a].push_back(b);
        adj_rev[b].push_back(a);
    }
    used.assign(n, false);
    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs1(i);
    used.assign(n, false);
    reverse(order.begin(), order.end());
    for (auto v : order)
        if (!used[v]) {
            dfs2(v);
            // ... processing next component ...
            component.clear();
        }
}

```

2.3 Кун

```

vi lr(k, -1), rl(n, -1);
vi used(k, 0);
int curr = 1;
auto dfs = [&g, &used, &lr, &rl, &curr]\
(const auto &rec, int v) -> bool {
    if (used[v] == curr)
        return false;
    used[v] = curr;
    for (int to : g[v]) {
        if (rl[to] == -1 || rec(rec, rl[to])) {
            rl[to] = v;
            lr[v] = to;
            return true;
        }
    }
    return false;
};

ll ans = 0;
for (bool run = true; run;) {
    run = false;
    ++curr;
    for (int i = 0; i < k; ++i) {
        if (lr[i] == -1 && dfs(dfs, i)) {
            run = true;
            ++ans;
        }
    }
}

```

2.4 Мосты

```

void dfs(int v, int p = -1) {
    used[v] = true;
    d[v] = h[v] = (p == -1 ? 0 : h[p] + 1);
    for (int u : g[v]) {
        if (u != p) {
            if (used[u]) // если ребро обратное
                d[v] = min(d[v], h[u]);
            else { // если ребро прямое

```

```

    dfs(u, v);
    d[v] = min(d[v], d[u]);
    if (h[v] < d[u]) {
        // если нельзя другим путем
        // добраться в v или выше,
        // то ребро (v, u) - мост
    }
}
}
}
}
}

```

2.5 Точки сочленения

```
void dfs(int v, int p = -1) {
    used[v] = 1;
    d[v] = h[v] = (p == -1 ? 0 : h[p] + 1);
    int children = 0;
    for (int u : g[v]) {
        if (u != p) {
            if (used[u])
                d[v] = min(d[v], h[u]);
            else {
                dfs(u, v);
                d[v] = min(d[v], d[u]);
                if (h[v] <= d[u] && p != -1) {
                    // v - точка сочленения
                    // (это условие может выполняться много
                     раз для разных
                     // детей)
                }
                children++;
            }
        }
    }
    if (p == -1 && children > 1) {
        // v - корень и точка сочленения
    }
}
```

2.6 Центроиды

```
// Центроиды с алгоритммики

int l = 179;
int ans = 0;
bool used[maxn];
int s[maxn];

void sizes(int v, int p) {
    s[v] = 1;
    for (int u : g[v])
        if (u != p && !used[u])
            sizes(u, v), s[v] += s[u];
}

int centroid(int v, int p, int n) {
    for (int u : g[v])
        if (u != p && !used[u] && s[u] > n/2)
            return centroid(u, v, n);
    return v;
}

void dfs(int v, int p, int d, vector<int> &t) {
    t.push_back(d);
    for (int u : g[v])
        if (u != p && !used[u])
            dfs(u, v, d + 1, t);
}

void solve(int v) {
    sizes(v);
    vector<int> d(s[v], 0);
    d[0] = 1;
    for (int u : g[v]) {
        if (!used[u]) {
            vector<int> t;
            dfs(u, v, 1, t);
            for (int x : t)
                if (x <= 1)
                    ans += d[1-x];
            for (int x : t)
                d[x]++;
        }
    }
}
```

```

    }
}

used[v] = 1;
for (int u : g[v])
    if (!used[u])
        solve(centroid(u, v, s[u]));
}

```

3 Контекст

3.1 CMakeLists

```
cmake_minimum_required(VERSION 3.14)
project(mrc)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_FLAGS "-Wall -Wextra -pedantic
-Wfloat-equal -Wconversion -Wlogical-op
-Wshift-overflow=2
-fsanitize=address,undefined,\
signed-integer-overflow,pointer-compare,\
pointer-subtract,shadow-call-stack,\
leak,bounds,pointer-overflow
-fno-sanitize-recover -D_GLIBCXX_DEBUG
-D_GLIBCXX_DEBUG_PEDANTIC -DONPC")

add_executable(A A.cpp)
```

3.2 Стресс-тесты

```
import subprocess, sys

_, f1, f2, gen, iters = sys.argv

for i in range(int(iters)):
    print('Test', i+1)
    test = subprocess.run(["python", gen],
        encoding="utf-8",
```

```

        capture_output=True).stdout
    #print(test)
    v1 = subprocess.run(["./%s" % f1],
        input=test, encoding="utf-8",
        capture_output=True).stdout
    v2 = subprocess.run(["./%s" % f2],
        input=test, encoding="utf-8",
        capture_output=True).stdout
    if v1 != v2:
        print("FAIL!\nInput:")
        print(test)
        print("Correct output:")
        print(v1)
        print("Wrong output:")
        print(v2)
        sys.exit()
    print("No output differences found.")

```

3.3 Шаблон

```

#ifdef ONPC
#pragma GCC optimize("O3")
#pragma GCC target("avx2,avx,sse,sse2,\
sse3,ssse3,sse4,sse4.1,sse4.2,\
lzcnt,popcnt,abm,bmi,bmi2")
#pragma GCC optimize("unroll-loops")
#endif

```

```

#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

```

```

using namespace std;
using ll = int64_t;
using ii = pair<int, int>;
using vi = vector<int>;
using vll = vector<ll>;
using vii = vector<ii>;
using vvi = vector<vi>;
using vc = vector<char>;
using vvc = vector<vc>;

```

```

using ui = uint32_t;
using ull = uint64_t;
using ld = long double;

using namespace __gnu_pbds;
template<typename T>
using OrderedSet<T> tree<T, null_type, less<>,
    rb_tree_tag,
    tree_order_statistics_node_update>;

#define all(x) (x).begin(), (x).end()
#define nl '\n'

template<typename T>
istream& operator>>(istream& s, vector<T>& v) {
    for (auto &&el : v)
        s >> el;
    return s;
}

```

```

template<typename T>
ostream& operator<<(ostream& s, const vector<T>&
    v) {
    for (auto &&el : v)
        s << el << ' ';
    s << '\n';
    return s;
}

signed main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
}

```

4 Строки

4.1 Z-функция

```

void z_func(int n) {
    z[0] = n;
}

```

```

int l = 0, r = 0;
for (int i = 1; i < n; ++i) {
    z[i] = MIN(r - i, z[i - 1]);
    if (z[i] < 0)
        z[i] = 0;
    while (i + z[i] < n && s[z[i]] == s[i +
        z[i]])
        ++z[i];
    if (i + z[i] > r) {
        l = i;
        r = i + z[i];
    }
}
}

```

4.2 Ахо-Корасик

// Здесь код какой-то задачи на Ахо-Корасик
 // Вряд ли понадобится, и я уже не помню, что
 здесь происходит

```

struct query {
    int l, r;
};

bool operator<(const query &a, const query &b) {
    if (a.l == b.l)
        return a.r < b.r;
    return a.l < b.l;
}

```

```

const int ALPHA = 26;
const char OFF = 'a';

```

```

struct Node {
    array<int, ALPHA> next;
    int p, link, zip;
    bool term;
    int pchar;
    int len;
};

```

```

struct Trie {
    vector<Node> t;
    int n = 0;

    Trie(const vector<string> &vec) {
        create(-1, -1);
        for (auto &s : vec)
            insert(s);
        build();
    }

    int create(int p, int pchar) {
        t.push_back({});
        fill(all(t[n].next), -1);
        t[n].p = p;
        t[n].pchar = pchar;
        t[n].link = t[n].zip = -1;
        t[n].term = false;
        t[n].len = -1;
        ++n;
        return n - 1;
    }

    void insert(const string &s, int pos = 0, int
        v = 0) {
        if (pos == static_cast<int>(s.size())) {
            t[v].term = true;
            t[v].len = pos;
            return;
        }
        int c = s[pos] - OFF;
        if (t[v].next[c] == -1)
            t[v].next[c] = create(v, c);
        insert(s, pos + 1, t[v].next[c]);
    }

    void build() {
        deque<int> q;
        for (int i = 0; i < ALPHA; ++i) {
            if (t[0].next[i] == -1)
                t[0].next[i] = 0;
            else
                q.push_back(t[0].next[i]);
        }
    }
}

```

```

t[0].link = 0;

while (!q.empty()) {
    int v = q.front();
    q.pop_front();
    if (t[v].p == 0)
        t[v].link = 0;
    else
        t[v].link =
            t[t[v].p].link].next[t[v].pchar];
    if (t[t[v].link].term)
        t[v].zip = t[v].link;
    else if (t[v].link == 0)
        t[v].zip = -1;
    else
        t[v].zip = t[t[v].link].zip;
    for (int i = 0; i < ALPHA; ++i) {
        if (t[v].next[i] == -1)
            t[v].next[i] =
                t[t[v].link].next[i];
        else
            q.push_back(t[v].next[i]);
    }
}

};

struct vertex {
    vi e;
    int d;
};

void dfs(vector<vertex> &g, vi &tin, vi &tout,
    int &t, int v, int p, int d) {
    tin[v] = t++;
    g[v].d = d;
    for (int to : g[v].e) {
        if (to == p)
            continue;
        dfs(g, tin, tout, t, to, v, d + 1);
    }
    tout[v] = t;
}

```

```

int32_t main() {
    fast_io

    int n;
    cin >> n;
    vector<string> vec(n);
    for (auto &s : vec) {
        cin >> s;
        reverse(all(s));
    }
    Trie trie(vec);

    string s;
    cin >> s;
    int len = static_cast<int>(s.size());
    vi v(len);
    int curr = 0;
    for (int i = len - 1; i >= 0; --i) {
        curr = trie.t[curr].next[s[i] - OFF];
        v[i] = curr;
    }
    vector<vertex> g(len + 1);
    for (int i = len - 1; i >= 0; --i) {
        int ln = -1;
        int currv = v[i];
        if (trie.t[currv].term)
            ln = trie.t[currv].len;
        if (ln == -1 && trie.t[currv].zip != -1) {
            currv = trie.t[currv].zip;
            ln = trie.t[currv].len;
        }
        if (ln != -1) {
            g[i + ln].e.push_back(i);
        }
    }
    vi tin(len + 1, -1), tout(len + 1, -1);
    int t = 0;
    for (int root = len; root >= 0; --root) {
        if (tin[root] == -1)
            dfs(g, tin, tout, t, root, -1, 0);
    }

    auto anc = [&tin, &tout](int u, int v) ->
        bool {

```



```

    if (tin[u] == -1 || tin[v] == -1)
        return false;
    return tin[u] <= tin[v] && tin[v] <
        tout[u];
};

int m;
cin >> m;
while (m--) {
    int l, r;
    cin >> l >> r;
    --l;
    if (!anc(l, r) && !anc(r, l))
        cout << "-1\n";
    else
        cout << abs(g[l].d - g[r].d) << '\n';
}

int k;
cin >> k;
while (k--) {
    ll sum = 0;
    int T, a, b, c, d, e, li, ri;
    cin >> T >> a >> b >> c >> d >> e >> li >>
        ri;
    int l, r;
    for (int i = 0; i < T; ++i) {
        l = min(li % len, ri % len);
        r = max(li % len + 1, ri % len + 1);
        int ans = 0;
        if (!anc(l, r) && !anc(r, l)) {
            ans = -1;
        } else {
            ans = abs(g[l].d - g[r].d);
            sum += ans;
            sum %= e;
        }
        li = (a * li + b) % e;
        ri = (c * ri + d + ans) % e;
    }
    cout << sum << '\n';
}

```

4.3 Манакер

```

vector<int> manacher_odd(string s) {
    int n = (int)s.size();
    vector<int> d(n, 1);
    int l = 0, r = 0;
    for (int i = 1; i < n; i++) {
        if (i < r)
            d[i] = min(r - i + 1, d[l + r - i]);
        while (i - d[i] >= 0 && i + d[i] < n &&
            s[i - d[i]] == s[i + d[i]])
            d[i]++;
        if (i + d[i] - 1 > r)
            l = i - d[i] + 1, r = i + d[i] - 1;
    }
    return d;
}

vector<int> manacher_even(string s) {
    int n = (int)s.size();
    vector<int> d(n, 0);
    int l = -1, r = -1;
    for (int i = 0; i < n - 1; i++) {
        if (i < r)
            d[i] = min(r - i, d[l + r - i - 1]);
        while (i - d[i] >= 0 && i + d[i] + 1 < n &&
            s[i - d[i]] == s[i + d[i] + 1])
            d[i]++;
        if (i + d[i] > r)
            l = i - d[i] + 1, r = i + d[i];
    }
    return d;
}

```

4.4 Префикс-функция

```

void pref_func(size_t n, const char str[n], int
    p[n]) {
    p[0] = 0;
    for (int i = 1; i < n; ++i) {
        int k = p[i - 1];
        while (k > 0 && str[i] != str[k])

```

```

        k = p[k - 1];
        if (str[i] == str[k])
            ++k;
        p[i] = k;
    }
}

```

4.5 Суффиксный автомат

```

//
// Суффиксный автомат
// Автомат, который принимает все подстроки строки
// s.
// Каждая вершина отвечает за несколько
// последовательных суффиксов какого-то
// префикса. Строится побуквенно, поддерживает
// построение от нескольких строк.
//

struct SuffixAutomaton {
    struct Node {
        int suf = -1, par = -1, nx[26];
        Node(int suf, int par) : suf(suf), par(par) {
            fill(nx, nx + 26, -1);
        }
    };
    vector<Node> nodes;
    int root = 0, last = 0;
    SuffixAutomaton() { nodes.emplace_back(-1, -1);
    }

    int addChar(int x) {
        int cur = sz(nodes);
        nodes.emplace_back(0, last);
        int p = last;
        for (; p != -1 && nodes[p].nx[x] == -1; p =
            nodes[p].suf)
            nodes[p].nx[x] = cur;
        if (p != -1) {
            int q = nodes[p].nx[x];
            if (nodes[q].par == p) {
                nodes[cur].suf = q;
            } else {
                int u = sz(nodes);
                nodes.push_back(nodes[q]);

```

```

    nodes[u].par = p;
    nodes[q].suf = nodes[cur].suf = u;
    for (; p != -1 && nodes[p].nx[x] == q; p =
        nodes[p].suf)
        nodes[p].nx[x] = u;
    }
}
return last = nodes[nodes[cur].par].nx[x];
};

```

4.6 Суффиксный массив

```

//
// Суффиксный массив
// Возвращает список индексов в порядке
// возрастания соответствующих суффиксов.
// sa[0] = n, потому что пустой суффикс считается
// минимальным.

```

```

vector<int> suffix_array(string s) {
    int n = sz(s) + 1, a, b, lim = 256;
    vector<int> x(all(s) + 1), y(n), ws(max(n,
        lim)), rank(n), sa(n);
    iota(all(sa), 0);
    for (int j = 0, p = 0; p < n; j = max(1, j *
        2), lim = p) {
        p = j, iota(all(y), n - j);
        for (int i = 0; i < n; i++)
            if (sa[i] >= j) y[p++] = sa[i] - j;
        fill(all(ws), 0);
        for (int i = 0; i < n; i++) ws[x[i]]++;
        for (int i = 1; i < lim; i++) ws[i] += ws[i -
            1];
        for (int i = n; i--;) sa[--ws[x[y[i]]]] =
            y[i];
        swap(x, y), p = 1, x[sa[0]] = 0;
        for (int i = 1; i < n; i++)
            a = sa[i - 1], b = sa[i],
            x[b] = (y[a] == y[b] && y[a + j] == y[b +
                j]) ? p - 1 : p++;
    }
}

```

```

    return sa;
}
vector<int> largest_common_prefix(string s, const
    vector<int> &sa) {
    int n = sz(sa);
    vector<int> rank(n), lcp(n);
    for (int i = 1; i < n; i++) rank[sa[i]] = i;
    for (int i = 0, k = 0; i < n - 1; i++) {
        if (k) k--;
        int j = sa[rank[i] - 1];
        while (s[i + k] == s[j + k]) k++;
        lcp[rank[i]] = k;
    }
    return lcp;
}

```

5 Структуры данных

5.1 Sparse table

```

vi logs(n + 1, 0);
int last_pow = 1;
for (int i = 2; i <= n; ++i) {
    logs[i] = logs[i - 1];
    if (i >= last_pow * 2) {
        last_pow *= 2;
        ++logs[i];
    }
}

vvi sp(n, vi(logs[n] + 1, INF));
for (int i = 0; i < n; ++i)
    sp[i][0] = a[i];
for (int l = 1; l < logs[n] + 1; ++l) {
    for (int i = 0; i < n; ++i) {
        if (i + (1 << (l - 1)) >= n)
            break;
        sp[i][l] = min(sp[i][l - 1], sp[i + (1 <<
            (l - 1))][l - 1]);
    }
}

```

```

while (q-- > 0) {
    int l, r;
    cin >> l >> r;
    int len = logs[r - l + 1];
    cout << min(sp[l][len], sp[r - (1 <<
        len)][len]) << '\n';
}

```

5.2 ДО с массивами

```

// ДО с массивными операциями для операция вида
// ax+b и суммы на отрезке по модулю

```

```

struct affine {
    ll k, b;
};

```

```

struct segtree {
    int n;
    vll t;
    vector<affine> aff;

    segtree(const vi &a) {
        n = a.size();
        t.resize(n * 4, 0);
        aff.resize(n * 4, {1, 0});
        build(a, 1, 0, n);
    }

    ll build(const vi &a, int v, int l, int r) {
        if (l + 1 == r) {
            return t[v] = a[l];
        }
        int m = (l + r) / 2;
        return t[v] = (build(a, 2 * v, l, m) +
            build(a, 2 * v + 1, m, r)) % M;
    }

    void push(int v, int l, int r) {
        t[v] *= aff[v].k;
        t[v] += aff[v].b * (r - l);
    }
}

```

```

t[v] %= M;

aff[2 * v].k *= aff[v].k;
aff[2 * v].b *= aff[v].k;
aff[2 * v].b += aff[v].b;
aff[2 * v].k %= M;
aff[2 * v].b %= M;

aff[2 * v + 1].k *= aff[v].k;
aff[2 * v + 1].b *= aff[v].k;
aff[2 * v + 1].b += aff[v].b;
aff[2 * v + 1].k %= M;
aff[2 * v + 1].b %= M;

aff[v].k = 1;
aff[v].b = 0;
}

11 fresh(int v, int l, int r) const {
    return (t[v] * aff[v].k + aff[v].b * (r -
        l)) % M;
}

11 get(int ql, int qr, int v = 1, int l = 0,
    int r = -1) {
    if (r == -1)
        r = n;
    if (qr <= l || r <= ql)
        return 0;
    if (ql <= l && r <= qr)
        return fresh(v, l, r);

    push(v, l, r);
    int m = (l + r) / 2;
    return (get(ql, qr, 2 * v, l, m) + get(ql,
        qr, 2 * v + 1, m, r)) % M;
}

void upd(int ql, int qr, int k, int b, int v
    = 1, int l = 0, int r = -1) {
    if (r == -1)
        r = n;
    if (qr <= l || r <= ql)
        return;

```

```

    if (ql <= l && r <= qr) {
        aff[v].k *= k;
        aff[v].b *= k;
        aff[v].b += b;
        aff[v].k %= M;
        aff[v].b %= M;
        return;
    }

    push(v, l, r);
    int m = (l + r) / 2;
    upd(ql, qr, k, b, 2 * v, l, m);
    upd(ql, qr, k, b, 2 * v + 1, m, r);
    t[v] = (fresh(2 * v, l, m) + fresh(2 * v +
        1, m, r)) % M;
}
};

```

5.3 ДО снизу

```

//
// Дерево отрезков снизу
// Поддерживает любые запросы, не требующие
// отложенных операций.
// Источник:
// https://codeforces.com/blog/entry/18051
//

struct SegmentTree {
    int n;
    vector<int> t;
    SegmentTree(int n) : n(n), t(2 * n, 0) {}
    void update(int i, int d) {
        for (i += n; i > 0; i >>= 1) t[i] += d;
    }
    void get(int l, int r) { // сумма на интервале
        [l; r)
        int left_res = 0, right_res = 0;
        for (l += n, r += n; l < r; l >>= 1, r >>= 1)
            {
                if (l & 1) left_res += t[l++];
                if (r & 1) right_res += t[--r];
            }
    }
};

```

```

    }
    return left_res + right_res;
}
};

```

5.4 Дерево Фенвика

```

//
// Дерево Фенвика
// Позволяет выполнять запросы на префиксе
// массива.
// В дереве используется 1-индексация.
//

struct FenwickTree {
    static int F(int x) { return x & -x; }
    int n;
    vector<int> t;
    FenwickTree(int n) : n(n), t(n + 1, 0) {}
    void update(int i, int d) { // a[i] += d, i in
        [1; n]
        for (; i <= n; i += F(i)) t[i] += d;
    }
    int get(int r) { // сумма на отрезке [1; r]
        int res = 0;
        for (; r > 0; r -= F(r)) res += t[r];
        return res;
    }
    int lower_bound(int sum) { // вернёт первое r
        такое, что get(r) >= sum
        // или n + 1, если такого r нет
        const int K = 20;
        int i = 0;
        for (int k = 1 << K; k > 0; k >>= 1) {
            if (i + k <= n && t[i + k] < sum) {
                sum -= t[i + k];
            }
        }
        return i + 1;
    }
};
};

```

5.5 CHM

```

struct dsu {
    vi p;
    vi rank;

    dsu(int n) {
        p.resize(n);
        rank.resize(n, 1);
        iota(all(p), 0);
    }

    int root(int x) {
        if (p[x] == x)
            return x;
        return p[x] = root(p[x]);
    }

    void unite(int a, int b) {
        a = root(a);
        b = root(b);
        if (a == b)
            return;
        if (rank[a] < rank[b])
            swap(a, b);
        p[b] = a;
        rank[a] += rank[b];
    }
};

```

5.6 Хеш-таблица

```

struct HashMap {
    const uint64_t C = ll(4e18 * acos(0)) | 71;
    uint64_t hash(uint64_t x) const { return
        __builtin_bswap64(x * C); }
    int n;
    vector<uint64_t> keys;
    vector<int> values;
    HashMap(int n) : n(n), keys(n, -1), values(n) {}
    int position(uint64_t key) const {

```

```

        uint64_t h = hash(key);
        int i = h % n;
        while (keys[i] != -1 && keys[i] != key)
            if (++i == n) i = 0;
        return i;
    }
    int& operator[](uint64_t key) {
        int i = position(key);
        if (keys[i] == -1) keys[i] = key, values[i] =
            -1;
        return values[i];
    }
};

```

6 Теория Чисел

6.1 КТО

```

ll euclid(ll a, ll b, ll &x, ll &y) {
    if (!b) return x = 1, y = 0, a;
    ll d = euclid(b, a % b, y, x);
    return y -= a / b * x, d;
}

ll crt(ll a, ll m, ll b, ll n) {
    if (n > m) swap(a, b), swap(m, n);
    ll x, y, g = euclid(m, n, x, y);
    if (a - b) % g == 0) return -1;
    x = (b - a) % n * x % n / g * m + a;
    return x < 0 ? x + m * n / g : x;
}

```

6.2 Метод Гаусса

```

int gauss(vector<vector<double>> a,
          vector<double>& ans) {
    int n = (int)a.size();
    int m = (int)a[0].size() - 1;
    vector<int> where(m, -1);

```

```

    for (int col = 0, row = 0; col < m && row < n;
        ++col) {
        int sel = row;
        for (int i = row; i < n; ++i)
            if (abs(a[i][col]) > abs(a[sel][col])) sel =
                i;
        if (abs(a[sel][col]) < EPS) continue;
        for (int i = col; i <= m; ++i)
            swap(a[sel][i], a[row][i]);
        where[col] = row;
        for (int i = 0; i < n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j = col; j <= m; ++j) a[i][j] -=
                    a[row][j] * c;
            }
        ++row;
    }
    ans.assign(m, 0);
    for (int i = 0; i < m; ++i)
        if (where[i] != -1) ans[i] = a[where[i]][m] /
            a[where[i]][i];
    for (int i = 0; i < n; ++i) {
        double sum = 0;
        for (int j = 0; j < m; ++j) sum += ans[j] *
            a[i][j];
        if (abs(sum - a[i][m]) > EPS) return 0;
    }
    for (int i = 0; i < m; ++i)
        if (where[i] == -1) return INF;
    return 1;
}

// бинарный
int gauss(vector<bitset<N>> a, int n, int m,
          bitset<N>& ans) {
    vector<int> where(m, -1);
    for (int col = 0, row = 0; col < m && row < n;
        ++col) {
        for (int i = row; i < n; ++i)
            if (a[i][col]) {
                swap(a[i], a[row]);
                break;
            }

```

```

    if (!a[row][col]) continue;
    where[col] = row;
    for (int i = 0; i < n; ++i)
        if (i != row && a[i][col]) a[i] ^= a[row];
        ++row;
    }
}

```

6.3 Ро-алгоритм Полларда

```

const int maxc = 500010;
ll n, x[maxc];
ll mul(ll a, ll b, ll m) { // m <= 8e18
    ll k = ((ld)a * b) / m;
    ll r = a * b - k * m;
}

```

```

while (r < 0) r += m;
while (r >= m) r -= m;
return r;
}
void slow(int n) {
    for (int i = 2; i * i <= n; i++)
        if (n % i == 0) {
            cout << i << " " << n / i << endl;
            exit(0);
        }
    cout << "IMPOSSIBLE" << endl;
    exit(0);
}
int main() {
    cin >> n;
    if (n <= (int)1e6) Slow(n);
    ll C = 2 * pow(n, 1.0 / 4);
}

```

```

for (int cnt = 0; cnt < 4; cnt++) {
    x[0] = abs((int)rnd()) % (n - 1) + 1;
    for (i = 0; i < C; i++) x[i + 1] = (mul(x[i],
        x[i], n) + 3) % n;
    for (int i = 0; i < C; i++) {
        ll g = gcd(abs(x[i] - x[C]), n);
        if (g != 1 && g != n) {
            cout << g << " " << n / g << endl;
            return 0;
        }
    }
}
cout << "IMPOSSIBLE" << endl;
return 0;
}

```
