

Team Reference Document

The Meanwhiles (Grigorii Bazilevich, Ilia Denisev, Vladimir Yakunin)

15 ноября 2024 г.

Содержание

1 Геометрия	1	3.3 Шаблон	11	6 Структуры данных	16
1.1 Базовая геома	1	4 Матеша	11	6.1 Convex Hull Trick	16
1.2 Выпуклая оболочка	4	4.1 Бинарное возведение (без рекурсии)	11	6.2 Sparse table	17
2 Графы	4	4.2 Комбинаторика	11	6.3 ДО с массовыми	17
2.1 2-SAT	4	4.2.1 Беспорядки	11	6.4 ДО снизу	18
2.2 HLD	4	4.2.2 Биномиальные коэффы	11	6.5 Дерево Фенвика	18
2.3 LCA (бинарные подьёмы)	5	4.2.3 Числа Каталана	11	6.6 Мо	18
2.4 Коммивояжер	5	4.2.4 Числа Стирлинга 2-го вида	11	6.7 СНМ	19
2.5 Конденсация	5	4.3 Метод Гаусса	12	6.8 Хеш-таблица	19
2.6 Кратчайшие пути	6	4.4 Теория чисел	12		
2.6.1 Дейкстра ($O((V + E)\log V)$)	6	4.4.1 Диофантовы уравнения	12	1 Геометрия	
2.6.2 Дейкстра ($O(V^2 + E)$)	6	4.4.2 Дискретный логарифм	12	1.1 Базовая геома	
2.6.3 Флойд	6	4.4.3 КТО	12		
2.6.4 Форд-Беллман	6	4.4.4 Линейное решето	12		
2.7 Кун	6	4.4.5 Решето Эратосфена	13		
2.8 Мосты	7	4.4.6 Ро-алгоритм Полларда	13		
2.9 Потоки	7	4.4.7 Функция Эйлера	13		
2.9.1 Диниц	7	4.5 Числа Фибоначчи	13		
2.9.2 Минкост	8	5 Строки	13		
2.9.3 Форд-Фалкерсон	9	5.1 Z-функция	13		
2.10 Прим	9	5.2 Ахо-Корасик	13		
2.11 Точки сочленения	10	5.3 Манакер	15		
2.12 Центроиды	10	5.4 Префикс-функция	15		
3 Контест	10	5.5 Суффиксный автомат	15		
3.1 CMakeLists	10	5.6 Суффиксный массив	16		
3.2 Стресс-тесты	10	5.7 Хэши	16		

```
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;
    }
}

```

```

        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 2-SAT

Выражение $a|b$ эквивалентно $(!a \rightarrow b) \& (!b \rightarrow a)$. Построим граф импликаций: для каждой переменной в графе будет по две вершины, (обозначим их через x и $!x$), а рёбра в этом графе будут соответствовать импликациям.

Правило. Пусть $[x]$ обозначает номер компоненты сильной связности, которой принадлежит вершина x , причём номера упорядочены в порядке топологической сортировки компонент сильной связности: если есть путь из x в y , то $c[x] \leq c[y]$. Тогда, если $c[x] > c[!x]$, то выбираем значение x , иначе выбираем $!x$.

Построим граф импликаций, заменив все выражения вида $a|b$ двумя ребрами $!a \rightarrow b$ и $!b \rightarrow a$.

Найдем все компоненты сильной связности в графе

импликаций.

Проверим, что для любого значения x его отрицание лежит в другой компоненте сильной связности: $c[x] \neq c[!x]$. Если это не так, то решения не существует.

Если требуется выводить ответ, то положим условие x верным, если $c[x] < c[!x]$, и неверным в противном случае.

2.2 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]);
}
};
```

2.3 LCA (бинарные подъёмы)

```
void dfs1(vvi &g, vc &used, vvi &up, vi &tin,
    vi &tout, int logn, int &t, int v) {
    for (int i = 1; i < logn; ++i)
        up[v][i] = up[up[v][i - 1]][i - 1];
    tin[v] = t++;
    used[v] = true;
    for (int to : g[v]) {
```

```

    if (!used[to]) {
        up[to][0] = v;
        dfs1(g, used, up, tin, tout, logn, t, to);
    }
}
tout[v] = t;
}

int logn = static_cast<int>(ceil(log2(n))) + 1;
vvi up(n, vi(logn, 0));
int t = 0;
dfs1(g, used, up, tin, tout, logn, t, 0);
auto anc = [&tin, &tout](int u, int v) {
    return tin[u] <= tin[v] && tin[v] < tout[u];
};

int lca(int v, int u) {
    if (anc(v, u)) return v;
    if (anc(u, v)) return u;
    for (int l = logn - 1; l >= 0; l--)
        if (!anc(up[v][l], u)) v = up[v][l];
    return up[v][0];
}

```

2.4 Коммивояжер

```

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

for (int mask = 1; mask < (1 << n); mask++) {
    for (int v = 0; v < n; v++) {
        if ((mask & (1 << v)) == 0) {
            continue;
        }

        int submask = mask ^ (1 << v);
        for (int u = 0; u < n; u++) {
            if (u == v || (submask & (1 << u))
                == 0 || g[u][v] == -1) {
                continue;
            }

```

```

        ll answer = dp[submask][u] +
            g[u][v];
        if (answer < dp[mask][v]) {
            dp[mask][v] = answer;
        }
    }
}

int mask = (1 << n) - 1;
ll answer = INF;
for (int v = 0; v < n; v++) {
    if (dp[mask][v] < answer) {
        answer = dp[mask][v];
    }
}

```

2.5 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();
        }
}

```

2.6 Кратчайшие пути

2.6.1 Дейкстра ($O((V + E)\log V)$)

```

vector<int> dijkstra(int s) {
    vector<int> d(n, inf);
    d[root] = 0;
    // объявим очередь с приоритетами для *минимума*
    // (по умолчанию ищется максимум)
    using pair<int, int> Pair;
    priority_queue<Pair, vector<Pair>,
        greater<Pair>>
        q;
    q.push({0, s});
    while (!q.empty()) {
        auto [cur_d, v] = q.top();
        q.pop();
        if (cur_d > d[v]) continue;
        for (auto [u, w] : g[v]) {
            if (d[u] > d[v] + w) {
                d[u] = d[v] + w;
                q.push({d[u], u});
            }
        }
    }
}

```

2.6.2 Дейкстра ($O(V^2 + E)$)

```
const int maxn = 1e5, inf = 1e9;
vector<pair<int, int> > g[maxn];
int n;

vector<int> dijkstra(int s) {
    vector<int> d(n, inf), a(n, 0);
    d[s] = 0;
    for (int i = 0; i < n; i++) {
        // находим вершину с минимальным d[v] из ещё
        // не помеченных
        int v = -1;
        for (int u = 0; u < n; u++)
            if (!a[u] && (v == -1 || d[u] < d[v]))
                v = u;
        // помечаем её и проводим релаксации вдоль
        // всех исходящих ребер
        a[v] = true;
        for (auto [u, w] : g[v])
            d[u] = min(d[u], d[v] + w);
    }
    return d;
}
```

2.6.3 Флойд

```
vvi dist(n, vi(n, INF));
for (int i = 0; i < n; ++i) {
    dist[i][i] = 0;
}
for (int i = 0; i < m; ++i) {
    ll a, b, w;
    cin >> a >> b >> w;
    dist[a - 1][b - 1] = min(w, dist[a - 1][b - 1]);
}

for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (dist[i][k] != INF &&
```

```
        dist[k][j] != INF) {
            dist[i][j] = min(
                dist[i][j],
                max(dist[i][k] + dist[k][j], -INF));
        }
    }
}
```

2.6.4 Форд-Беллман

```
vvi dist(k + 1, vi(n, INF));
dist[0][s] = 0;
for (int i = 1; i < k + 1; ++i) {
    for (auto&& [from, to, w] : g) {
        if (dist[i - 1][from] != INF)
            dist[i][to] =
                min(dist[i][to], dist[i - 1][from] + w);
    }
}
```

2.7 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;
        }
    }
}
```

2.8 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
                }
            }
        }
    }
}
```

2.9 Поток

2.9.1 Диниц

```
using ll = long long;

struct Edge {
    int u;
```

```

    ll f, c;
    Edge* rev;

    Edge(int u_, ll f_, ll c_) {
        u = u_;
        f = f_;
        c = c_;

        rev = nullptr;
    }
};

const int MAXN = 1000;
const ll INF = 1e18;

int n, m, s, f;
vector<Edge*> g[MAXN];
vector<ll> d(MAXN), p(MAXN);

bool bfs() {
    d.assign(MAXN, INF);
    d[s] = 0;

    queue<int> q;
    q.push(s);

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

        for (auto e : g[v]) {
            if (e->f < e->c && d[e->u] == INF) {
                d[e->u] = d[v] + 1;
                q.push(e->u);
            }
        }
    }

    return d[f] != INF;
}

int dfs(int v, ll min_flow = INF) {
    if (v == f) {
        return min_flow;
    }

```

```

    }

    for (; p[v] < g[v].size(); p[v]++) {
        auto e = g[v][p[v]];
        if (e->f < e->c && d[e->u] == d[v] + 1) {
            int flow =
                dfs(e->u, min(min_flow, e->c - e->f));
            if (flow > 0) {
                e->f += flow;
                e->rev->f -= flow;
                return flow;
            }
        }
    }

    return 0;
}

vector<Edge*> edges;
void addEdge(int v, int u, ll w = 0) {
    Edge* normal = new Edge(u, 0, w);
    Edge* rev = new Edge(v, 0, 0);

    normal->rev = rev;
    rev->rev = normal;

    g[v].push_back(normal);
    g[u].push_back(rev);
}

int32_t main() {
    cin.tie(0);
    cout.tie(0);
    ios::sync_with_stdio(0);

    cin >> n >> m;

    s = 0;
    f = n - 1;

    for (int i = 0; i < m; i++) {
        int v, u;
        ll w;
        cin >> v >> u >> w;
    }

```

```

        addEdge(v - 1, u - 1, w);
    }

    ll flow = 0;
    while (bfs()) {
        p.assign(MAXN, 0);
        while (true) {
            ll new_flow = dfs(s);
            if (new_flow == 0) {
                break;
            }

            flow += new_flow;
        }
    }

    cout << flow;
    return 0;
}

```

2.9.2 Минкост

```

using ll = long long;

struct Edge {
    int v, u;
    ll f, c;
    ll w;
    Edge* rev;

    Edge(int v_, int u_, ll f_, ll c_, ll w_) {
        v = v_;
        u = u_;
        f = f_;
        c = c_;
        w = w_;

        rev = nullptr;
    }
};

```



```

const int MAXN = 150;
const ll INF = 1e18;

ll ans = 0;
int n, m, s, f;
vector<Edge*> edges, p(MAXN);
vector<ll> d(MAXN);

void addEdge(int v, int u, ll c = 0, ll w = 0) {
    Edge* normal = new Edge(v, u, 0, c, w);
    Edge* rev = new Edge(u, v, 0, 0, -w);

    normal->rev = rev;
    rev->rev = normal;

    edges.push_back(normal);
    edges.push_back(rev);
}

ll ford_bellman() {
    d.assign(MAXN, INF);
    p.assign(MAXN, nullptr);
    d[s] = 0;

    ll flow = INF;
    for (int i = 1; i < n; i++) {
        for (auto e : edges) {
            if (e->c > e->f &&
                d[e->u] > d[e->v] + e->w) {
                d[e->u] = d[e->v] + e->w;
                p[e->u] = e;
                flow = min(flow, e->c - e->f);
            }
        }
    }

    if (d[f] == INF) {
        return 0;
    }

    Edge* e = p[f];
    while (e != nullptr) {
        e->f += flow;
        e->rev->f -= flow;
    }
}

```

```

        e = p[e->v];
    }

    ans += flow * d[f];
    return flow;
}

int32_t main() {
    cin.tie(0);
    cout.tie(0);
    ios::sync_with_stdio(0);

    cin >> n >> m;

    s = 0;
    f = n - 1;

    for (int i = 0; i < m; i++) {
        int v, u;
        ll c, w;
        cin >> v >> u >> c >> w;

        addEdge(v - 1, u - 1, c, w);
    }

    ll flow = 0;
    while (true) {
        ll new_flow = ford_bellman();
        if (new_flow == 0) {
            break;
        }

        flow += new_flow;
    }

    cerr << flow << "\n";
    cout << ans;
    return 0;
}

```

2.9.3 Форд-Фалкерсон

```

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

struct Edge {
    int u, f, c;
    Edge* rev;

    Edge(int u_, int f_, int c_) {
        u = u_;
        f = f_;
        c = c_;

        rev = nullptr;
    }
};

const int MAXN = 1e5 + 10;

int n, m, s, f;
vector<Edge*> g[MAXN];
vector<bool> used(MAXN);
vector<int> path;

int dfs(int v, int min_flow = 1e9) {
    if (v == f) {
        return min_flow;
    }

    used[v] = true;
    for (auto e : g[v]) {
        if (e->f < e->c && !used[e->u]) {
            int flow =
                dfs(e->u, min(min_flow, e->c - e->f));
            if (flow > 0) {
                e->f += flow;
                e->rev->f -= flow;
                return flow;
            }
        }
    }
}

```

```

    return 0;
}

int32_t main() {
    cin.tie(0);
    cout.tie(0);
    ios::sync_with_stdio(0);

    cin >> n >> m;

    s = 0;
    f = n - 1;
    vector<Edge*> edges;
    for (int i = 0; i < m; i++) {
        int v, u, w;
        cin >> v >> u >> w;

        v--;
        u--;

        if (u == v) {
            continue;
        }

        Edge* normal = new Edge(u, 0, w);
        Edge* rev = new Edge(v, 0, w);

        normal->rev = rev;
        rev->rev = normal;

        g[v].push_back(normal);
        g[u].push_back(rev);
        edges.push_back(normal);
    }

    int flow = 0;
    while (true) {
        used.assign(n, 0);
        int new_flow = dfs(s);
        if (new_flow == 0) {
            break;
        }

        flow += new_flow;
    }
}

```

```

    }

    cout << flow << "\n";
    for (auto e : edges) {
        cout << e->f << "\n";
    }

    return 0;
}

2.10 Прим

const int maxn = 1e5, inf = 1e9;
bool used[maxn];
vector<pair<int, int> > g[maxn];
int min_edge[maxn] = {inf}, best_edge[maxn];
min_edge[0] = 0;

// ...

for (int i = 0; i < n; i++) {
    int v = -1;
    for (int u = 0; u < n; u++)
        if (!used[u] &&
            (v == -1 || min_edge[u] < min_edge[v]))
            v = u;

    used[v] = 1;
    if (v != 0)
        cout << v << " " << best_edge[v] << endl;

    for (auto e : g[v]) {
        int u = e.first, w = e.second;
        if (w < min_edge[u]) {
            min_edge[u] = w;
            best_edge[u] = v;
        }
    }
}

```

2.11 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
    }
}

```

2.12 Центроиды

// Центроиды с алгоритмики

```

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
import 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
    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

```

#ifndef 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 Матеша

4.1 Бинарное возведение (без рекурсии)

```
int binpow(int a, int n) {
    int res = 1;
    while (n) {
        if (n & 1) res *= a;
        a *= a;
        n >>= 1;
    }
    return res;
}
```

4.2 Комбинаторика

4.2.1 Беспорядки

$n!$ — кол-во беспорядков, т.е. перестановок, где каждый элемент не на своём изначальном месте.

$$!0 = 1, \quad !1 = 0$$

$$!n = (n-1) \cdot (n-1)! = n! \sum_{i=0}^n \frac{(-1)^i}{i!} = \left\lfloor \frac{n!}{e} \right\rfloor$$

4.2.2 Биномиальные коэффициенты

$$C_n^k = \binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} = \frac{n}{k} \binom{n-1}{k-1} = \frac{n}{n-k} \binom{n-1}{k} = \frac{n-k+1}{k} \binom{n}{k-1}$$

$$\binom{n}{h} \binom{n-h}{k} = \binom{n}{k} \binom{n-k}{h} = \binom{n}{h+k} \binom{h+k}{h}$$

$$\sum_{k=0}^n \binom{n}{k} = 2^n$$

$$\sum_{j=0}^k \binom{m}{j} \binom{n-m}{k-j} = \binom{n}{k}$$

$$\sum_{m=0}^n \binom{m}{j} \binom{n-m}{k-j} = \binom{n+1}{k+1}$$

4.2.3 Числа Каталана

Кол-во ПСП с n парами скобок:

$$C_n = \binom{2n}{n} - \binom{2n}{n+1} = \frac{1}{n+1} \binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^n \frac{n+k}{k}$$

4.2.4 Числа Стирлинга 2-го вида

Кол-во способов разбить n объектов на k непустых множеств:

$$\left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n = \sum_{i=0}^k \frac{(-1)^{k-i} i^n}{(k-i)! i!}$$

4.3 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;
    }
}
```

4.4 Теория чисел

4.4.1 Диофантовы уравнения

$$\begin{aligned}
 ax + by &= c \\
 ax' + by' &= d \\
 \begin{cases} x_0 = x' \cdot \frac{c}{d} \\ y_0 = y' \cdot \frac{c}{d} \end{cases} \\
 \begin{cases} x = x_0 + k \cdot \frac{b}{d} \\ y = y_0 - k \cdot \frac{a}{d} \end{cases}
 \end{aligned}$$

4.4.2 Дискретный логарифм

```

const ll MOD = 998'244'353;
const ll BLOCK =
    500; // типа sqrt(MOD), но подбирать ручками

ll dlog(ll a, ll b, unordered_map<ll, ll> &fs) {
    ll an = binpow(a, BLOCK, MOD);
    unordered_map<ll, ll> fs;
    ll curr = an;
    for (int i = 1; i <= MOD / BLOCK + 1; ++i) {
        if (fs.find(curr) == fs.end()) fs[curr] = i;
        curr = (curr * an) % MOD;
    }
    ll curr = b;
    for (int i = 0; i <= BLOCK; ++i) {
        if (fs.find(curr) != fs.end()) {
            ll ans = fs[curr] * BLOCK - i;
            if (ans < MOD) return ans;
        }
        curr = (curr * a) % MOD;
    }
    return -1;
}

```

4.4.3 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;
}

```

4.4.4 Линейное решето

```

const int N = 10000000;
int lp[N + 1]; // минимальный простой делитель
vector<int> pr; // простые числа

for (int i = 2; i <= N; ++i) {
    if (lp[i] == 0) {
        lp[i] = i;
        pr.push_back(i);
    }
    for (int j = 0;
        j < (int)pr.size() && pr[j] <= lp[i] &&
        i * pr[j] <= N; ++j)
        lp[i * pr[j]] = pr[j];
}

```

4.4.5 Решето Эратосфена

```

int n;
vector<char> prime(n + 1, true);
prime[0] = prime[1] = false;
for (int i = 2; i * i <= n; ++i)
    if (prime[i])
        if (i * 1ll * i <= n)
            for (int j = i * i; j <= n; j += i)
                prime[j] = false;

```

4.4.6 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;
}

```

4.4.7 Функция Эйлера

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

4.5 Числа Фибоначчи

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

$$F(n+1) = \sum_{k=0}^{\lfloor n/2 \rfloor} \binom{n-k}{k}$$

5 Строки

5.1 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 - l]);
        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];
        }
    }
}
```

5.2 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';
}
}

```

5.3 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;
}

```

5.4 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;
    }
}

```

5.5 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] == -1;
                 p = nodes[p].suf)
                nodes[p].nx[x] = u;
        }
    }
    return last = nodes[nodes[cur].par].nx[x];
}
};

```

5.6 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;
}

```


5.7 Хэши

```
ll m = 1e9 + 7;
ll k = 53;
vll h1(n);
vll kp(n + 1);
kp[0] = 1;
h1[0] = mod((s1[0] - 'A' + 1), m);
for (int i = 1; i < n; ++i) {
    kp[i] = mod((kp[i - 1] * k), m);
    ll c1 = s1[i] - 'A' + 1;
    h1[i] = mod((h1[i - 1] * k + c1), m);
}
kp[n] = mod((kp[n - 1] * k), m);
auto ss1 = [&h1, &kp, m](int l, int r) -> ll {
    if (l == 0) return h1[r];
    return mod((h1[r] - h1[l - 1] * kp[r - l + 1]),
               m);
};
```

6 Структуры данных

6.1 Convex Hull Trick

```
struct line {
    int k, b;
    line() {}
    line(int a, int _b) { k = a, b = _b; }
    int get(int x) { return k * x + b; }
};

vector<line>
    lines; // храним прямые нижней огибающей
vector<int> dots; // храним x-координаты точек
                // нижней огибающей
// первое правило вещественных чисел
// считаем, что в dots лежит округленная вниз
// x-координата

int cross(line a,
```

```
    line b) { // считаем точку пересечения
               // считаем a.k > b.k
    int x = (b.b - a.b) / (a.k - b.k);
    if (b.b < a.b)
        x--; // боремся с округлением у отрицательных
             // чисел
    return x;
}

void add(line cur) {
    while (lines.size() &&
           lines.back().get(dots.back()) >
           cur.get(dots.back())) {
        lines.pop_back();
        dots.pop_back();
    }
    if (lines.empty())
        dots.push_back(-inf);
    else
        dots.push_back(cross(lines.back(), cur));
    lines.push_back(cur);
}

int get(int x) {
    int pos =
        lower_bound(dots.begin(), dots.end(), x) -
        dots.begin() - 1;
    return lines[pos].get(x);
}
```

6.2 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-->) {
        int l, r;
        cin >> l >> r;
        int len = logs[r - l];
        cout << min(sp[l][len],
                    sp[r - (1 << len)][len])
              << '\n';
    }
}
```

6.3 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 = (1 + 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;
}

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

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

```

```

    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 = (1 + 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;
}
};

```

6.4 ДО снизу

// ДО снизу от peltorator

```

vector<long long> tree;
int n;

void build(const vector<int>& arr) {
    n = arr.size();
    tree.assign(2 * n, 0);
    for (int i = 0; i < n; i++) {
        tree[n + i] = arr[i];
    }
    for (int i = n - 1; i > 0; i--) {
        tree[i] = tree[i << 1] + tree[(i << 1) | 1];
    }
}

```

```

}

void update_point(
    int pos, int newval) { // arr[pos] := newval
    pos += n;
    tree[pos] = newval;
    pos >= 1;
    while (pos > 0) {
        tree[pos] =
            tree[pos << 1] + tree[(pos << 1) | 1];
        pos >= 1;
    }
}

long long find_sum(int l, int r) { // [l, r)
    l += n;
    r += n;
    long long ans = 0;
    while (l < r) {
        if (l & 1) {
            ans += tree[l++];
        }
        if (r & 1) {
            ans += tree[--r];
        }
        l >>= 1;
        r >>= 1;
    }
    return ans;
}

```

6.5 Дерево Фенвика

// 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;
}
};

```

6.6 Мо

```

// границы и номер запроса, на который нужно
// ответить
struct query {
    int l, r, idx;
};

int a[maxn],
    ans[maxq]; // исходный массив и массив
               // ответов на запросы
vector<query> b[c];

// где-то в main:

for (query q : queries) b[q.l / c].push_back(q);

for (int i = 0; i < c; i++) {
    sort(
        b[i].begin(), b[i].end(),

```

```

        [](query a, query b) { return a.r < b.r; });
}

int cnt[maxn];
int res;

void add(int k) {
    if (cnt[a[k]]++ == 0) res++;
}

void del(int k) {
    if (--cnt[a[k]] == 0) res--;
}

for (int i = 0; i < c; i++) {
    // обнуляем переменные
    int l = i * c, r = i * c - 1;
    memset(cnt, 0, sizeof cnt);
    res = 0;
    for (query q : b[i]) {
        // пока правая граница не дошла до границы
        // запроса
        while (r < q.r) add(++r);
        // дальше делаем так, чтобы левая граница
        // совпала
        while (l < q.l) del(l++);
        while (l > q.l) add(--l);
        ans[q.idx] = res;
    }
}

```

6.7 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];
    }
};

```

6.8 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];
    }
};

```